

**Richard M. Reese, Jennifer L. Reese,
Boštjan Kaluža, Dr. Uday Kamath,
Krishna Choppella**

Machine Learning: End-to-End guide for Java developers

Learning Path

Data Analysis, Machine Learning and Neural Networks
simplified



Packt

Machine Learning: End-to-End guide for Java developers

Table of Contents

[Machine Learning: End-to-End guide for Java developers](#)

[Credits](#)

[Preface](#)

[What this learning path covers](#)

[What you need for this learning path](#)

[Who this learning path is for](#)

[Reader feedback](#)

[Customer support](#)

[Downloading the example code](#)

[Errata](#)

[Piracy](#)

[Questions](#)

[1. Module 1](#)

[1. Getting Started with Data Science](#)

[Problems solved using data science](#)

[Understanding the data science problem - solving approach](#)

[Using Java to support data science](#)

[Acquiring data for an application](#)

[The importance and process of cleaning data](#)

[Visualizing data to enhance understanding](#)

[The use of statistical methods in data science](#)

[Machine learning applied to data science](#)

[Using neural networks in data science](#)

[Deep learning approaches](#)

[Performing text analysis](#)

[Visual and audio analysis](#)

[Improving application performance using parallel techniques](#)

[Assembling the pieces](#)

[Summary](#)

[2. Data Acquisition](#)

[Understanding the data formats used in data science applications](#)

[Overview of CSV data](#)

[Overview of spreadsheets](#)

[Overview of databases](#)

[Overview of PDF files](#)

[Overview of JSON](#)

[Overview of XML](#)

[Overview of streaming data](#)

[Overview of audio/video/images in Java](#)

[Data acquisition techniques](#)

[Using the HttpURLConnection class](#)

[Web crawlers in Java](#)

[Creating your own web crawler](#)

[Using the crawler4j web crawler](#)

[Web scraping in Java](#)

[Using API calls to access common social media sites](#)

[Using OAuth to authenticate users](#)

[Handling Twitter](#)

[Handling Wikipedia](#)

[Handling Flickr](#)

[Handling YouTube](#)

[Searching by keyword](#)

[Summary](#)

[3. Data Cleaning](#)

[Handling data formats](#)

[Handling CSV data](#)

[Handling spreadsheets](#)

[Handling Excel spreadsheets](#)

[Handling PDF files](#)

[Handling JSON](#)

[Using JSON streaming API](#)

[Using the JSON tree API](#)

[The nitty gritty of cleaning text](#)

[Using Java tokenizers to extract words](#)

[Java core tokenizers](#)

[Third-party tokenizers and libraries](#)

[Transforming data into a usable form](#)

[Simple text cleaning](#)

[Removing stop words](#)

[Finding words in text](#)

[Finding and replacing text](#)

[Data imputation](#)

[Subsetting data](#)

[Sorting text](#)

[Data validation](#)

[Validating data types](#)

[Validating dates](#)

[Validating e-mail addresses](#)

[Validating ZIP codes](#)

[Validating names](#)

[Cleaning images](#)

[Changing the contrast of an image](#)

[Smoothing an image](#)

[Brightening an image](#)

[Resizing an image](#)

[Converting images to different formats](#)

[Summary](#)

[4. Data Visualization](#)

[Understanding plots and graphs](#)

[Visual analysis goals](#)

[Creating index charts](#)

[Creating bar charts](#)

[Using country as the category](#)

[Using decade as the category](#)

[Creating stacked graphs](#)

[Creating pie charts](#)

[Creating scatter charts](#)

[Creating histograms](#)

[Creating donut charts](#)

[Creating bubble charts](#)

[Summary](#)

[5. Statistical Data Analysis Techniques](#)

[Working with mean, mode, and median](#)

[Calculating the mean](#)

[Using simple Java techniques to find mean](#)

[Using Java 8 techniques to find mean](#)

[Using Google Guava to find mean](#)

[Using Apache Commons to find mean](#)

[Calculating the median](#)

[Using simple Java techniques to find median](#)

- [Using Apache Commons to find the median](#)
 - [Calculating the mode](#)
 - [Using ArrayLists to find multiple modes](#)
 - [Using a HashMap to find multiple modes](#)
 - [Using a Apache Commons to find multiple modes](#)
 - [Standard deviation](#)
 - [Sample size determination](#)
 - [Hypothesis testing](#)
 - [Regression analysis](#)
 - [Using simple linear regression](#)
 - [Using multiple regression](#)
 - [Summary](#)
- ## [6. Machine Learning](#)
- [Supervised learning techniques](#)
 - [Decision trees](#)
 - [Decision tree types](#)
 - [Decision tree libraries](#)
 - [Using a decision tree with a book dataset](#)
 - [Testing the book decision tree](#)
 - [Support vector machines](#)
 - [Using an SVM for camping data](#)
 - [Testing individual instances](#)
 - [Bayesian networks](#)
 - [Using a Bayesian network](#)
 - [Unsupervised machine learning](#)
 - [Association rule learning](#)
 - [Using association rule learning to find buying relationships](#)
 - [Reinforcement learning](#)
 - [Summary](#)
- ## [7. Neural Networks](#)
- [Training a neural network](#)
 - [Getting started with neural network architectures](#)
 - [Understanding static neural networks](#)
 - [A basic Java example](#)
 - [Understanding dynamic neural networks](#)
 - [Multilayer perceptron networks](#)
 - [Building the model](#)
 - [Evaluating the model](#)

[Predicting other values](#)

[Saving and retrieving the model](#)

[Learning vector quantization](#)

[Self-Organizing Maps](#)

[Using a SOM](#)

[Displaying the SOM results](#)

[Additional network architectures and algorithms](#)

[The k-Nearest Neighbors algorithm](#)

[Instantaneously trained networks](#)

[Spiking neural networks](#)

[Cascading neural networks](#)

[Holographic associative memory](#)

[Backpropagation and neural networks](#)

[Summary](#)

[8. Deep Learning](#)

[Deeplearning4j architecture](#)

[Acquiring and manipulating data](#)

[Reading in a CSV file](#)

[Configuring and building a model](#)

[Using hyperparameters in ND4J](#)

[Instantiating the network model](#)

[Training a model](#)

[Testing a model](#)

[Deep learning and regression analysis](#)

[Preparing the data](#)

[Setting up the class](#)

[Reading and preparing the data](#)

[Building the model](#)

[Evaluating the model](#)

[Restricted Boltzmann Machines](#)

[Reconstruction in an RBM](#)

[Configuring an RBM](#)

[Deep autoencoders](#)

[Building an autoencoder in DL4J](#)

[Configuring the network](#)

[Building and training the network](#)

[Saving and retrieving a network](#)

[Specialized autoencoders](#)

[Convolutional networks](#)

[Building the model](#)

[Evaluating the model](#)

[Recurrent Neural Networks](#)

[Summary](#)

[9. Text Analysis](#)

[Implementing named entity recognition](#)

[Using OpenNLP to perform NER](#)

[Identifying location entities](#)

[Classifying text](#)

[Word2Vec and Doc2Vec](#)

[Classifying text by labels](#)

[Classifying text by similarity](#)

[Understanding tagging and POS](#)

[Using OpenNLP to identify POS](#)

[Understanding POS tags](#)

[Extracting relationships from sentences](#)

[Using OpenNLP to extract relationships](#)

[Sentiment analysis](#)

[Downloading and extracting the Word2Vec model](#)

[Building our model and classifying text](#)

[Summary](#)

[10. Visual and Audio Analysis](#)

[Text-to-speech](#)

[Using FreeTTS](#)

[Getting information about voices](#)

[Gathering voice information](#)

[Understanding speech recognition](#)

[Using CMUPhinx to convert speech to text](#)

[Obtaining more detail about the words](#)

[Extracting text from an image](#)

[Using Tess4j to extract text](#)

[Identifying faces](#)

[Using OpenCV to detect faces](#)

[Classifying visual data](#)

[Creating a Neuroph Studio project for classifying visual images](#)

[Training the model](#)

[Summary](#)

11. Mathematical and Parallel Techniques for Data Analysis

Implementing basic matrix operations

Using GPUs with DeepLearning4j

Using map-reduce

Using Apache's Hadoop to perform map-reduce

Writing the map method

Writing the reduce method

Creating and executing a new Hadoop job

Various mathematical libraries

Using the jblas API

Using the Apache Commons math API

Using the ND4J API

Using OpenCL

Using Aparapi

Creating an Aparapi application

Using Aparapi for matrix multiplication

Using Java 8 streams

Understanding Java 8 lambda expressions and streams

Using Java 8 to perform matrix multiplication

Using Java 8 to perform map-reduce

Summary

12. Bringing It All Together

Defining the purpose and scope of our application

Understanding the application's architecture

Data acquisition using Twitter

Understanding the TweetHandler class

Extracting data for a sentiment analysis model

Building the sentiment model

Processing the JSON input

Cleaning data to improve our results

Removing stop words

Performing sentiment analysis

Analysing the results

Other optional enhancements

Summary

2. Module 2

1. Applied Machine Learning Quick Start

Machine learning and data science

[What kind of problems can machine learning solve?](#)

[Applied machine learning workflow](#)

[Data and problem definition](#)

[Measurement scales](#)

[Data collection](#)

[Find or observe data](#)

[Generate data](#)

[Sampling traps](#)

[Data pre-processing](#)

[Data cleaning](#)

[Fill missing values](#)

[Remove outliers](#)

[Data transformation](#)

[Data reduction](#)

[Unsupervised learning](#)

[Find similar items](#)

[Euclidean distances](#)

[Non-Euclidean distances](#)

[The curse of dimensionality](#)

[Clustering](#)

[Supervised learning](#)

[Classification](#)

[Decision tree learning](#)

[Probabilistic classifiers](#)

[Kernel methods](#)

[Artificial neural networks](#)

[Ensemble learning](#)

[Evaluating classification](#)

[Precision and recall](#)

[Roc curves](#)

[Regression](#)

[Linear regression](#)

[Evaluating regression](#)

[Mean squared error](#)

[Mean absolute error](#)

[Correlation coefficient](#)

[Generalization and evaluation](#)

[Underfitting and overfitting](#)

[Train and test sets](#)
[Cross-validation](#)
[Leave-one-out validation](#)
[Stratification](#)

[Summary](#)

[2. Java Libraries and Platforms for Machine Learning](#)

[The need for Java](#)

[Machine learning libraries](#)

[Weka](#)

[Java machine learning](#)

[Apache Mahout](#)

[Apache Spark](#)

[Deeplearning4j](#)

[MALLET](#)

[Comparing libraries](#)

[Building a machine learning application](#)

[Traditional machine learning architecture](#)

[Dealing with big data](#)

[Big data application architecture](#)

[Summary](#)

[3. Basic Algorithms – Classification, Regression, and Clustering](#)

[Before you start](#)

[Classification](#)

[Data](#)

[Loading data](#)

[Feature selection](#)

[Learning algorithms](#)

[Classify new data](#)

[Evaluation and prediction error metrics](#)

[Confusion matrix](#)

[Choosing a classification algorithm](#)

[Regression](#)

[Loading the data](#)

[Analyzing attributes](#)

[Building and evaluating regression model](#)

[Linear regression](#)

[Regression trees](#)

[Tips to avoid common regression problems](#)

Clustering

Clustering algorithms

Evaluation

Summary

4. Customer Relationship Prediction with Ensembles

Customer relationship database

Challenge

Dataset

Evaluation

Basic naive Bayes classifier baseline

Getting the data

Loading the data

Basic modeling

Evaluating models

Implementing naive Bayes baseline

Advanced modeling with ensembles

Before we start

Data pre-processing

Attribute selection

Model selection

Performance evaluation

Summary

5. Affinity Analysis

Market basket analysis

Affinity analysis

Association rule learning

Basic concepts

Database of transactions

Itemset and rule

Support

Confidence

Apriori algorithm

FP-growth algorithm

The supermarket dataset

Discover patterns

Apriori

FP-growth

Other applications in various areas

[Medical diagnosis](#)
[Protein sequences](#)
[Census data](#)
[Customer relationship management](#)
[IT Operations Analytics](#)

[Summary](#)

[6. Recommendation Engine with Apache Mahout](#)

[Basic concepts](#)

[Key concepts](#)
[User-based and item-based analysis](#)
[Approaches to calculate similarity](#)
[Collaborative filtering](#)
[Content-based filtering](#)
[Hybrid approach](#)

[Exploitation versus exploration](#)

[Getting Apache Mahout](#)

[Configuring Mahout in Eclipse with the Maven plugin](#)

[Building a recommendation engine](#)

[Book ratings dataset](#)

[Loading the data](#)

[Loading data from file](#)
[Loading data from database](#)
[In-memory database](#)

[Collaborative filtering](#)

[User-based filtering](#)
[Item-based filtering](#)
[Adding custom rules to recommendations](#)
[Evaluation](#)
[Online learning engine](#)

[Content-based filtering](#)

[Summary](#)

[7. Fraud and Anomaly Detection](#)

[Suspicious and anomalous behavior detection](#)

[Unknown-unknowns](#)

[Suspicious pattern detection](#)

[Anomalous pattern detection](#)

[Analysis types](#)

[Pattern analysis](#)

[Transaction analysis](#)

[Plan recognition](#)

[Fraud detection of insurance claims](#)

[Dataset](#)

[Modeling suspicious patterns](#)

[Vanilla approach](#)

[Dataset rebalancing](#)

[Anomaly detection in website traffic](#)

[Dataset](#)

[Anomaly detection in time series data](#)

[Histogram-based anomaly detection](#)

[Loading the data](#)

[Creating histograms](#)

[Density based k-nearest neighbors](#)

[Summary](#)

[8. Image Recognition with Deeplearning4j](#)

[Introducing image recognition](#)

[Neural networks](#)

[Perceptron](#)

[Feedforward neural networks](#)

[Autoencoder](#)

[Restricted Boltzmann machine](#)

[Deep convolutional networks](#)

[Image classification](#)

[Deeplearning4j](#)

[Getting DL4J](#)

[MNIST dataset](#)

[Loading the data](#)

[Building models](#)

[Building a single-layer regression model](#)

[Building a deep belief network](#)

[Build a Multilayer Convolutional Network](#)

[Summary](#)

[9. Activity Recognition with Mobile Phone Sensors](#)

[Introducing activity recognition](#)

[Mobile phone sensors](#)

[Activity recognition pipeline](#)

[The plan](#)

Collecting data from a mobile phone

Installing Android Studio

Loading the data collector

Feature extraction

Collecting training data

Building a classifier

Reducing spurious transitions

Plugging the classifier into a mobile app

Summary

10. Text Mining with Mallet – Topic Modeling and Spam Detection

Introducing text mining

Topic modeling

Text classification

Installing Mallet

Working with text data

Importing data

Importing from directory

Importing from file

Pre-processing text data

Topic modeling for BBC news

BBC dataset

Modeling

Evaluating a model

Reusing a model

Saving a model

Restoring a model

E-mail spam detection

E-mail spam dataset

Feature generation

Training and testing

Model performance

Summary

11. What is Next?

Machine learning in real life

Noisy data

Class unbalance

Feature selection is hard

Model chaining

Importance of evaluation
Getting models into production
Model maintenance
Standards and markup languages
CRISP-DM
SEMMA methodology
Predictive Model Markup Language
Machine learning in the cloud
 Machine learning as a service
Web resources and competitions
 Datasets
 Online courses
 Competitions
 Websites and blogs
 Venues and conferences
Summary

A. References

3. Module 3

1. Machine Learning Review

Machine learning – history and definition
What is not machine learning?
Machine learning – concepts and terminology
Machine learning – types and subtypes
Datasets used in machine learning
Machine learning applications
Practical issues in machine learning
Machine learning – roles and process

Roles

Process

Machine learning – tools and datasets

Datasets

Summary

2. Practical Approach to Real-World Supervised Learning

Formal description and notation

Data quality analysis

Descriptive data analysis

Basic label analysis

Basic feature analysis

Visualization analysis

Univariate feature analysis

Categorical features

Continuous features

Multivariate feature analysis

Data transformation and preprocessing

Feature construction

Handling missing values

Outliers

Discretization

Data sampling

Is sampling needed?

Undersampling and oversampling

Stratified sampling

Training, validation, and test set

Feature relevance analysis and dimensionality reduction

Feature search techniques

Feature evaluation techniques

Filter approach

Univariate feature selection

Information theoretic approach

Statistical approach

Multivariate feature selection

Minimal redundancy maximal relevance (mRMR)

Correlation-based feature selection (CFS)

Wrapper approach

Embedded approach

Model building

Linear models

Linear Regression

Algorithm input and output

How does it work?

Advantages and limitations

Naïve Bayes

Algorithm input and output

How does it work?

Advantages and limitations

Logistic Regression

[Algorithm input and output](#)

[How does it work?](#)

[Advantages and limitations](#)

[Non-linear models](#)

[Decision Trees](#)

[Algorithm inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[K-Nearest Neighbors \(KNN\)](#)

[Algorithm inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Support vector machines \(SVM\)](#)

[Algorithm inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Ensemble learning and meta learners](#)

[Bootstrap aggregating or bagging](#)

[Algorithm inputs and outputs](#)

[How does it work?](#)

[Random Forest](#)

[Advantages and limitations](#)

[Boosting](#)

[Algorithm inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Model assessment, evaluation, and comparisons](#)

[Model assessment](#)

[Model evaluation metrics](#)

[Confusion matrix and related metrics](#)

[ROC and PRC curves](#)

[Gain charts and lift curves](#)

[Model comparisons](#)

[Comparing two algorithms](#)

[McNemar's Test](#)

[Paired-t test](#)

[Wilcoxon signed-rank test](#)

[Comparing multiple algorithms](#)

ANOVA test

Friedman's test

Case Study – Horse Colic Classification

Business problem

Machine learning mapping

Data analysis

Label analysis

Features analysis

Supervised learning experiments

Weka experiments

Sample end-to-end process in Java

Weka experimenter and model selection

RapidMiner experiments

Visualization analysis

Feature selection

Model process flow

Model evaluation metrics

Evaluation on Confusion Metrics

ROC Curves, Lift Curves, and Gain Charts

Results, observations, and analysis

Summary

References

3. Unsupervised Machine Learning Techniques

Issues in common with supervised learning

Issues specific to unsupervised learning

Feature analysis and dimensionality reduction

Notation

Linear methods

Principal component analysis (PCA)

Inputs and outputs

How does it work?

Advantages and limitations

Random projections (RP)

Inputs and outputs

How does it work?

Advantages and limitations

Multidimensional Scaling (MDS)

Inputs and outputs

[How does it work?](#)

[Advantages and limitations](#)

[Nonlinear methods](#)

[Kernel Principal Component Analysis \(KPCA\)](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Manifold learning](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Clustering](#)

[Clustering algorithms](#)

[k-Means](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[DBSCAN](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Mean shift](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Expectation maximization \(EM\) or Gaussian mixture modeling \(GMM\)](#)

[Input and output](#)

[How does it work?](#)

[Advantages and limitations](#)

[Hierarchical clustering](#)

[Input and output](#)

[How does it work?](#)

[Advantages and limitations](#)

[Self-organizing maps \(SOM\)](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Spectral clustering](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Affinity propagation](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Clustering validation and evaluation](#)

[Internal evaluation measures](#)

[Notation](#)

[R-Squared](#)

[Dunn's Indices](#)

[Davies-Bouldin index](#)

[Silhouette's index](#)

[External evaluation measures](#)

[Rand index](#)

[F-Measure](#)

[Normalized mutual information index](#)

[Outlier or anomaly detection](#)

[Outlier algorithms](#)

[Statistical-based](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Distance-based methods](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Density-based methods](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Clustering-based methods](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[High-dimensional-based methods](#)

[Inputs and outputs](#)

How does it work?
Advantages and limitations
One-class SVM
 Inputs and outputs
 How does it work?
 Advantages and limitations
Outlier evaluation techniques
 Supervised evaluation
 Unsupervised evaluation
Real-world case study
 Tools and software
 Business problem
 Machine learning mapping
 Data collection
 Data quality analysis
 Data sampling and transformation
 Feature analysis and dimensionality reduction
 PCA
 Random projections
 ISOMAP
 Observations on feature analysis and dimensionality reduction
 Clustering models, results, and evaluation
 Observations and clustering analysis
 Outlier models, results, and evaluation
 Observations and analysis
Summary
References

4. Semi-Supervised and Active Learning

Semi-supervised learning
 Representation, notation, and assumptions
 Semi-supervised learning techniques
 Self-training SSL
 Inputs and outputs
 How does it work?
 Advantages and limitations
 Co-training SSL or multi-view SSL
 Inputs and outputs
 How does it work?

[Advantages and limitations](#)

[Cluster and label SSL](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Transductive graph label propagation](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Transductive SVM \(TSVM\)](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Case study in semi-supervised learning](#)

[Tools and software](#)

[Business problem](#)

[Machine learning mapping](#)

[Data collection](#)

[Data quality analysis](#)

[Data sampling and transformation](#)

[Datasets and analysis](#)

[Feature analysis results](#)

[Experiments and results](#)

[Analysis of semi-supervised learning](#)

[Active learning](#)

[Representation and notation](#)

[Active learning scenarios](#)

[Active learning approaches](#)

[Uncertainty sampling](#)

[How does it work?](#)

[Least confident sampling](#)

[Smallest margin sampling](#)

[Label entropy sampling](#)

[Advantages and limitations](#)

[Version space sampling](#)

[Query by disagreement \(QBD\)](#)

[How does it work?](#)

[Query by Committee \(QBC\)](#)

How does it work?
Advantages and limitations
Data distribution sampling
 How does it work?
 Expected model change
 Expected error reduction
 Variance reduction
 Density weighted methods
 Advantages and limitations
Case study in active learning
 Tools and software
 Business problem
 Machine learning mapping
 Data Collection
 Data sampling and transformation
 Feature analysis and dimensionality reduction
 Models, results, and evaluation
 Pool-based scenarios
 Stream-based scenarios
 Analysis of active learning results
Summary
References

5. Real-Time Stream Machine Learning

Assumptions and mathematical notations
Basic stream processing and computational techniques
 Stream computations
 Sliding windows
 Sampling
Concept drift and drift detection
 Data management
 Partial memory
 Full memory
 Detection methods
 Monitoring model evolution
 Widmer and Kubat
 Drift Detection Method or DDM
 Early Drift Detection Method or EDDM
 Monitoring distribution changes

Welch's t test
Kolmogorov-Smirnov's test
CUSUM and Page-Hinckley test

Adaptation methods
Explicit adaptation
Implicit adaptation

Incremental supervised learning

Modeling techniques
Linear algorithms

Online linear models with loss functions
Inputs and outputs
How does it work?
Advantages and limitations

Online Naïve Bayes
Inputs and outputs
How does it work?
Advantages and limitations

Non-linear algorithms

Hoeffding trees or very fast decision trees (VFDT)
Inputs and outputs
How does it work?
Advantages and limitations

Ensemble algorithms

Weighted majority algorithm
Inputs and outputs
How does it work?
Advantages and limitations

Online Bagging algorithm
Inputs and outputs
How does it work?
Advantages and limitations

Online Boosting algorithm
Inputs and outputs
How does it work?
Advantages and limitations

Validation, evaluation, and comparisons in online setting

Model validation techniques
Prequential evaluation

[Holdout evaluation](#)

[Controlled permutations](#)

[Evaluation criteria](#)

[Comparing algorithms and metrics](#)

[Incremental unsupervised learning using clustering](#)

[Modeling techniques](#)

[Partition based](#)

[Online k-Means](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Hierarchical based and micro clustering](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Density based](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Grid based](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Validation and evaluation techniques](#)

[Key issues in stream cluster evaluation](#)

[Evaluation measures](#)

[Cluster Mapping Measures \(CMM\)](#)

[V-Measure](#)

[Other external measures](#)

[Unsupervised learning using outlier detection](#)

[Partition-based clustering for outlier detection](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Distance-based clustering for outlier detection](#)

Inputs and outputs

How does it work?

Exact Storm

Abstract-C

Direct Update of Events (DUE)

Micro Clustering based Algorithm (MCOD)

Approx Storm

Advantages and limitations

Validation and evaluation techniques

Case study in stream learning

Tools and software

Business problem

Machine learning mapping

Data collection

Data sampling and transformation

Feature analysis and dimensionality reduction

Models, results, and evaluation

Supervised learning experiments

Concept drift experiments

Clustering experiments

Outlier detection experiments

Analysis of stream learning results

Summary

References

6. Probabilistic Graph Modeling

Probability revisited

Concepts in probability

Conditional probability

Chain rule and Bayes' theorem

Random variables, joint, and marginal distributions

Marginal independence and conditional independence

Factors

Factor types

Distribution queries

Probabilistic queries

MAP queries and marginal MAP queries

Graph concepts

Graph structure and properties

Subgraphs and cliques

Path, trail, and cycles

Bayesian networks

Representation

Definition

Reasoning patterns

Causal or predictive reasoning

Evidential or diagnostic reasoning

Intercausal reasoning

Combined reasoning

Independencies, flow of influence, D-Separation, I-Map

Flow of influence

D-Separation

I-Map

Inference

Elimination-based inference

Variable elimination algorithm

Input and output

How does it work?

Advantages and limitations

Clique tree or junction tree algorithm

Input and output

How does it work?

Advantages and limitations

Propagation-based techniques

Belief propagation

Factor graph

Messaging in factor graph

Input and output

How does it work?

Advantages and limitations

Sampling-based techniques

Forward sampling with rejection

Input and output

How does it work?

Advantages and limitations

Learning

Learning parameters

[Maximum likelihood estimation for Bayesian networks](#)

[Bayesian parameter estimation for Bayesian network](#)

[Prior and posterior using the Dirichlet distribution](#)

[Learning structures](#)

[Measures to evaluate structures](#)

[Methods for learning structures](#)

[Constraint-based techniques](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Search and score-based techniques](#)

[Inputs and outputs](#)

[How does it work?](#)

[Advantages and limitations](#)

[Markov networks and conditional random fields](#)

[Representation](#)

[Parameterization](#)

[Gibbs parameterization](#)

[Factor graphs](#)

[Log-linear models](#)

[Independencies](#)

[Global](#)

[Pairwise Markov](#)

[Markov blanket](#)

[Inference](#)

[Learning](#)

[Conditional random fields](#)

[Specialized networks](#)

[Tree augmented network](#)

[Input and output](#)

[How does it work?](#)

[Advantages and limitations](#)

[Markov chains](#)

[Hidden Markov models](#)

[Most probable path in HMM](#)

[Posterior decoding in HMM](#)

[Tools and usage](#)

[OpenMarkov](#)

Weka Bayesian Network GUI

Case study

Business problem

Machine learning mapping

Data sampling and transformation

Feature analysis

Models, results, and evaluation

Analysis of results

Summary

References

7. Deep Learning

Multi-layer feed-forward neural network

Inputs, neurons, activation function, and mathematical notation

Multi-layered neural network

Structure and mathematical notations

Activation functions in NN

Sigmoid function

Hyperbolic tangent ("tanh") function

Training neural network

Empirical risk minimization

Parameter initialization

Loss function

Gradients

Gradient at the output layer

Gradient at the Hidden Layer

Parameter gradient

Feed forward and backpropagation

How does it work?

Regularization

L2 regularization

L1 regularization

Limitations of neural networks

Vanishing gradients, local optimum, and slow training

Deep learning

Building blocks for deep learning

Rectified linear activation function

Restricted Boltzmann Machines

Definition and mathematical notation

[Conditional distribution](#)

[Free energy in RBM](#)

[Training the RBM](#)

[Sampling in RBM](#)

[Contrastive divergence](#)

[Inputs and outputs](#)

[How does it work?](#)

[Persistent contrastive divergence](#)

[Autoencoders](#)

[Definition and mathematical notations](#)

[Loss function](#)

[Limitations of Autoencoders](#)

[Denoising Autoencoder](#)

[Unsupervised pre-training and supervised fine-tuning](#)

[Deep feed-forward NN](#)

[Input and outputs](#)

[How does it work?](#)

[Deep Autoencoders](#)

[Deep Belief Networks](#)

[Inputs and outputs](#)

[How does it work?](#)

[Deep learning with dropouts](#)

[Definition and mathematical notation](#)

[Inputs and outputs](#)

[How does it work?](#)

[Learning Training and testing with dropouts](#)

[Sparse coding](#)

[Convolutional Neural Network](#)

[Local connectivity](#)

[Parameter sharing](#)

[Discrete convolution](#)

[Pooling or subsampling](#)

[Normalization using ReLU](#)

[CNN Layers](#)

[Recurrent Neural Networks](#)

[Structure of Recurrent Neural Networks](#)

[Learning and associated problems in RNNs](#)

[Long Short Term Memory](#)

Gated Recurrent Units

Case study

Tools and software

Business problem

Machine learning mapping

Data sampling and transfor

Feature analysis

Models, results, and evaluation

Basic data handling

Multi-layer perceptron

Parameters used for MLP

Code for MLP

Convolutional Network

Parameters used for ConvNet

Code for CNN

Variational Autoencoder

Parameters used for the Variational Autoencoder

Code for Variational Autoencoder

DBN

Parameter search using Arbiter

Results and analysis

Summary

References

8. Text Mining and Natural Language Processing

NLP, subfields, and tasks

Text categorization

Part-of-speech tagging (POS tagging)

Text clustering

Information extraction and named entity recognition

Sentiment analysis and opinion mining

Coreference resolution

Word sense disambiguation

Machine translation

Semantic reasoning and inferencing

Text summarization

Automating question and answers

Issues with mining unstructured data

Text processing components and transformations

Document collection and standardization

Inputs and outputs

How does it work?

Tokenization

Inputs and outputs

How does it work?

Stop words removal

Inputs and outputs

How does it work?

Stemming or lemmatization

Inputs and outputs

How does it work?

Local/global dictionary or vocabulary?

Feature extraction/generation

Lexical features

Character-based features

Word-based features

Part-of-speech tagging features

Taxonomy features

Syntactic features

Semantic features

Feature representation and similarity

Vector space model

Binary

Term frequency (TF)

Inverse document frequency (IDF)

Term frequency-inverse document frequency (TF-IDF)

Similarity measures

Euclidean distance

Cosine distance

Pairwise-adaptive similarity

Extended Jaccard coefficient

Dice coefficient

Feature selection and dimensionality reduction

Feature selection

Information theoretic techniques

Statistical-based techniques

Frequency-based techniques

Dimensionality reduction

Topics in text mining

Text categorization/classification

Topic modeling

Probabilistic latent semantic analysis (PLSA)

Input and output

How does it work?

Advantages and limitations

Text clustering

Feature transformation, selection, and reduction

Clustering techniques

Generative probabilistic models

Input and output

How does it work?

Advantages and limitations

Distance-based text clustering

Non-negative matrix factorization (NMF)

Input and output

How does it work?

Advantages and limitations

Evaluation of text clustering

Named entity recognition

Hidden Markov models for NER

Input and output

How does it work?

Advantages and limitations

Maximum entropy Markov models for NER

Input and output

How does it work?

Advantages and limitations

Deep learning and NLP

Tools and usage

Mallet

KNIME

Topic modeling with mallet

Business problem

Machine Learning mapping

Data collection

[Data sampling and transformation](#)

[Feature analysis and dimensionality reduction](#)

[Models, results, and evaluation](#)

[Analysis of text processing results](#)

[Summary](#)

[References](#)

[9. Big Data Machine Learning – The Final Frontier](#)

[What are the characteristics of Big Data?](#)

[Big Data Machine Learning](#)

[General Big Data framework](#)

[Big Data cluster deployment frameworks](#)

[Hortonworks Data Platform](#)

[Cloudera CDH](#)

[Amazon Elastic MapReduce](#)

[Microsoft Azure HDInsight](#)

[Data acquisition](#)

[Publish-subscribe frameworks](#)

[Source-sink frameworks](#)

[SQL frameworks](#)

[Message queueing frameworks](#)

[Custom frameworks](#)

[Data storage](#)

[HDFS](#)

[NoSQL](#)

[Key-value databases](#)

[Document databases](#)

[Columnar databases](#)

[Graph databases](#)

[Data processing and preparation](#)

[Hive and HQL](#)

[Spark SQL](#)

[Amazon Redshift](#)

[Real-time stream processing](#)

[Machine Learning](#)

[Visualization and analysis](#)

[Batch Big Data Machine Learning](#)

[H2O as Big Data Machine Learning platform](#)

[H2O architecture](#)

[Machine learning in H2O](#)

[Tools and usage](#)

[Case study](#)

[Business problem](#)

[Machine Learning mapping](#)

[Data collection](#)

[Data sampling and transformation](#)

[Experiments, results, and analysis](#)

[Feature relevance and analysis](#)

[Evaluation on test data](#)

[Analysis of results](#)

[Spark MLlib as Big Data Machine Learning platform](#)

[Spark architecture](#)

[Machine Learning in MLlib](#)

[Tools and usage](#)

[Experiments, results, and analysis](#)

[k-Means](#)

[k-Means with PCA](#)

[Bisecting k-Means \(with PCA\)](#)

[Gaussian Mixture Model](#)

[Random Forest](#)

[Analysis of results](#)

[Real-time Big Data Machine Learning](#)

[SAMOA as a real-time Big Data Machine Learning framework](#)

[SAMOA architecture](#)

[Machine Learning algorithms](#)

[Tools and usage](#)

[Experiments, results, and analysis](#)

[Analysis of results](#)

[The future of Machine Learning](#)

[Summary](#)

[References](#)

[A. Linear Algebra](#)

[Vector](#)

[Scalar product of vectors](#)

[Matrix](#)

[Transpose of a matrix](#)

[Matrix addition](#)

[Scalar multiplication](#)

[Matrix multiplication](#)

[Properties of matrix product](#)

[Linear transformation](#)

[Matrix inverse](#)

[Eigendecomposition](#)

[Positive definite matrix](#)

[Singular value decomposition \(SVD\)](#)

[B. Probability](#)

[Axioms of probability](#)

[Bayes' theorem](#)

[Density estimation](#)

[Mean](#)

[Variance](#)

[Standard deviation](#)

[Gaussian standard deviation](#)

[Covariance](#)

[Correlation coefficient](#)

[Binomial distribution](#)

[Poisson distribution](#)

[Gaussian distribution](#)

[Central limit theorem](#)

[Error propagation](#)

[D. Bibliography](#)

[Index](#)

Machine Learning: End-to-End guide for Java developers

Machine Learning: End-to-End guide for Java developers

Copyright © 2017 Packt Publishing

All rights reserved. No part of this course may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this course to ensure the accuracy of the information presented. However, the information contained in this course is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this course.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this course by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Published on: September 2017

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78862-221-9

www.packtpub.com

Credits

Authors

Richard M. Reese

Jennifer L. Reese

Boštjan Kaluža

Dr. Uday Kamath

Krishna Choppella

Reviewers

Walter Molina

Shilpi Saxena

Abhik Banerjee

Wei Di

Manjunath Narayana

Ravi Sharma

Samir Sahli

Prashant Verma

Content Development Editor

Aishwarya Pandere

Production Coordinator

Arvindkumar Gupta

Preface

Machine learning is a subfield of artificial intelligence. It helps computers to learn and act like human beings with the help of algorithms and data. With a given set of data, an ML algorithm learns different properties of the data and infers the properties of the data that it may encounter in future.

What this learning path covers

Module 1, Java for Data Science, investigates the support provided for low-level math operations and how they can be supported in a multiple processor environment. Data analysis, at its heart, necessitates the ability to manipulate and analyze large quantities of numeric data.

Module 2, Machine Learning in Java, reviews the various Java libraries and platforms dedicated to machine learning, what each library brings to the table, and what kind of problems it is able to solve. The review includes Weka, Java-ML, Apache Mahout, Apache Spark, deeplearning4j, and Mallet.

Module 3, Mastering Java Machine Learning, presents many advanced methods in clustering and outlier techniques, with applications. Topics covered are feature selection and reduction in unsupervised data, clustering algorithms, evaluation methods in clustering, and anomaly detection using statistical, distance, and distribution techniques. At the end of the chapter, we perform a case study for both clustering and outlier detection using a real-world image dataset, MNIST. We use the Smile API to do feature reduction and ELKI for learning.

What you need for this learning path

Module 1:

Many of the examples in this module use Java 8 features. There are a number of Java APIs demonstrated, each of which is introduced before it is applied. An IDE is not required but is desirable.

Module 2:

To follow the examples throughout the module, you'll need a personal computer with the JDK installed. All the examples and source code that you can download assume Eclipse IDE with support for Maven, a dependency management and build automation tool; and Git, a version control system. Examples in the chapters rely on various libraries, including Weka, deeplearning4j, Mallet, and Apache Mahout. Instructions on how to get and install the libraries are provided in the chapter where the library will be first used.

The module has a dedicated web site, <http://machine-learning-in-java.com>, where you can find all the example code, errata, and additional materials that will help you to get started.

Module 3:

This book assumes you have some experience of programming in Java and a basic understanding of machine learning concepts. If that doesn't apply to you, but you are curious nonetheless and self-motivated, fret not, and read on! For those who do have some background, it means that you are familiar with simple statistical analysis of data and concepts involved in supervised and unsupervised learning. Those who may not have the requisite math or must poke the far reaches of their memory to shake loose the odd formula or funny symbol, do not be disheartened. If you are the sort that loves a challenge, the short primer in the appendices may be all you need to kick-start your engines—a bit of tenacity will see you through the rest! For those who have never been introduced to machine learning, the first chapter was equally written for you as for those needing a refresher—it is your starter-kit to jump in feet first and find out what it's all about. You can augment your basics with any number of online resources. Finally, for those innocent of Java, here's a secret: many of the tools featured in the book have powerful GUIs. Some include wizard-like interfaces,

making them quite easy to use, and do not require any knowledge of Java. So if you are new to Java, just skip the examples that need coding and learn to use the GUI-based tools instead!

Who this learning path is for

This course is the right resource for anyone with some knowledge of Java programming who wants to get started with Data Science and Machine learning as quickly as possible. If you want to gain meaningful insights from big data and develop intelligent applications using Java, this course is also a must-have.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail <feedback@packtpub.com>, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at <http://www.packtpub.com>. If you purchased this course elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Ziipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at <https://github.com/PacktPublishing/repository-name>. We also have other code bundles from our rich catalog of books, videos, and courses available at <https://github.com/PacktPublishing/>. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at [<questions@packtpub.com>](mailto:questions@packtpub.com), and we will do our best to address the problem.

Part 1. Module 1

Java for Data Science

Examine the techniques and Java tools supporting the growing field of data science

Chapter 1. Getting Started with Data Science

Data science is not a single science as much as it is a collection of various scientific disciplines integrated for the purpose of analyzing data. These disciplines include various statistical and mathematical techniques, including:

- Computer science
- Data engineering
- Visualization
- Domain-specific knowledge and approaches

With the advent of cheaper storage technology, more and more data has been collected and stored permitting previously unfeasible processing and analysis of data. With this analysis came the need for various techniques to make sense of the data. These large sets of data, when used to analyze data and identify trends and patterns, become known as **big data**.

This in turn gave rise to cloud computing and concurrent techniques such as **map-reduce**, which distributed the analysis process across a large number of processors, taking advantage of the power of parallel processing.

The process of analyzing big data is not simple and evolves to the specialization of developers who were known as **data scientists**. Drawing upon a myriad of technologies and expertise, they are able to analyze data to solve problems that previously were either not envisioned or were too difficult to solve.

Early big data applications were typified by the emergence of search engines capable of more powerful and accurate searches than their predecessors. For example, **AltaVista** was an early popular search engine that was eventually superseded by Google. While big data applications were not limited to these search engine functionalities, these applications laid the groundwork for future work in big data.

The term, data science, has been used since 1974 and evolved over time to include statistical analysis of data. The concepts of data mining and data analytics have been associated with data science. Around 2008, the term data scientist appeared and was used to describe a person who performs data analysis. A more in-depth discussion of the history of data science can be found at

<http://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/#3d9ea08369fd>.

This book aims to take a broad look at data science using Java and will briefly touch on many topics. It is likely that the reader may find topics of interest and pursue these at greater depth independently. The purpose of this book, however, is simply to introduce the reader to the significant data science topics and to illustrate how they can be addressed using Java.

There are many algorithms used in data science. In this book, we do not attempt to explain how they work except at an introductory level. Rather, we are more interested in explaining how they can be used to solve problems. Specifically, we are interested in knowing how they can be used with Java.

Problems solved using data science

The various data science techniques that we will illustrate have been used to solve a variety of problems. Many of these techniques are motivated to achieve some economic gain, but they have also been used to solve many pressing social and environmental problems. Problem domains where these techniques have been used include finance, optimizing business processes, understanding customer needs, performing DNA analysis, foiling terrorist plots, and finding relationships between transactions to detect fraud, among many other data-intensive problems.

Data mining is a popular application area for data science. In this activity, large quantities of data are processed and analyzed to glean information about the dataset, to provide meaningful insights, and to develop meaningful conclusions and predictions. It has been used to analyze customer behavior, detecting relationships between what may appear to be unrelated events, and to make predictions about future behavior.

Machine learning is an important aspect of data science. This technique allows the computer to solve various problems without needing to be explicitly programmed. It has been used in self-driving cars, speech recognition, and in web searches. In data mining, the data is extracted and processed. With machine learning, computers use the data to take some sort of action.

Understanding the data science problem - solving approach

Data science is concerned with the processing and analysis of large quantities of data to create models that can be used to make predictions or otherwise support a specific goal. This process often involves the building and training of models. The specific approach to solve a problem is dependent on the nature of the problem. However, in general, the following are the high-level tasks that are used in the analysis process:

- **Acquiring the data:** Before we can process the data, it must be acquired. The data is frequently stored in a variety of formats and will come from a wide range of data sources.
- **Cleaning the data:** Once the data has been acquired, it often needs to be converted to a different format before it can be used. In addition, the data needs to be processed, or cleaned, so as to remove errors, resolve inconsistencies, and otherwise put it in a form ready for analysis.
- **Analyzing the data:** This can be performed using a number of techniques including:
 - **Statistical analysis:** This uses a multitude of statistical approaches to provide insight into data. It includes simple techniques and more advanced techniques such as regression analysis.
 - **AI analysis:** These can be grouped as machine learning, neural networks, and deep learning techniques:
 - Machine learning approaches are characterized by programs that can learn without being specifically programmed to complete a specific task
 - Neural networks are built around models patterned after the neural connection of the brain
 - Deep learning attempts to identify higher levels of abstraction within a set of data
 - **Text analysis:** This is a common form of analysis, which works with natural languages to identify features such as the names of people and places, the relationship between parts of text, and the implied meaning of text.
 - **Data visualization:** This is an important analysis tool. By displaying the data in a visual form, a hard-to-understand set of numbers can be more

readily understood.

- **Video, image, and audio processing and analysis:** This is a more specialized form of analysis, which is becoming more common as better analysis techniques are discovered and faster processors become available. This is in contrast to the more common text processing and analysis tasks.

Complementing this set of tasks is the need to develop applications that are efficient. The introduction of machines with multiple processors and GPUs contributes significantly to the end result.

While the exact steps used will vary by application, understanding these basic steps provides the basis for constructing solutions to many data science problems.

Using Java to support data science

Java and its associated third-party libraries provide a range of support for the development of data science applications. There are numerous core Java capabilities that can be used, such as the basic string processing methods. The introduction of lambda expressions in Java 8 helps enable more powerful and expressive means of building applications. In many of the examples that follow in subsequent chapters, we will show alternative techniques using lambda expressions.

There is ample support provided for the basic data science tasks. These include multiple ways of acquiring data, libraries for cleaning data, and a wide variety of analysis approaches for tasks such as natural language processing and statistical analysis. There are also myriad of libraries supporting neural network types of analysis.

Java can be a very good choice for data science problems. The language provides both object-oriented and functional support for solving problems. There is a large developer community to draw upon and there exist multiple APIs that support data science tasks. These are but a few reasons as to why Java should be used.

The remainder of this chapter will provide an overview of the data science tasks and Java support demonstrated in the book. Each section is only able to present a brief introduction to the topics and the available support. The subsequent chapter will go into considerably more depth regarding these topics.

Acquiring data for an application

Data acquisition is an important step in the data analysis process. When data is acquired, it is often in a specialized form and its contents may be inconsistent or different from an application's need. There are many sources of data, which are found on the Internet. Several examples will be demonstrated in [Chapter 2, Data Acquisition](#).

Data may be stored in a variety of formats. Popular formats for text data include **HTML**, **Comma Separated Values (CSV)**, **JavaScript Object Notation (JSON)**, and **XML**. Image and audio data are stored in a number of formats. However, it is frequently necessary to convert one data format into another format, typically plain text.

For example, JSON (<http://www.JSON.org/>) is stored using blocks of curly braces containing key-value pairs. In the following example, parts of a YouTube result is shown:

```
{  
    "kind": "youtube#searchResult",  
    "etag": etag,  
    "id": {  
        "kind": string,  
        "videoId": string,  
        "channelId": string,  
        "playlistId": string  
    },  
    ...  
}
```

Data is acquired using techniques such as **processing live streams**, **downloading compressed files**, and through **screen scraping**, where the information on a web page is extracted. **Web crawling** is a technique where a program examines a series of web pages, moving from one page to another, acquiring the data that it needs.

With many popular media sites, it is necessary to acquire a user ID and password to access data. A commonly used technique is **OAuth**, which is an open standard used to authenticate users to many different websites. The technique delegates access to a server resource and works over HTTPS. Several companies use OAuth 2.0, including PayPal, Facebook, Twitter, and Yelp.

The importance and process of cleaning data

Once the data has been acquired, it will need to be cleaned. Frequently, the data will contain errors, duplicate entries, or be inconsistent. It often needs to be converted to a simpler data type such as text. **Data cleaning** is often referred to as **data wrangling, reshaping, or munging**. They are effectively synonyms.

When data is cleaned, there are several tasks that often need to be performed, including checking its validity, accuracy, completeness, consistency, and uniformity. For example, when the data is incomplete, it may be necessary to provide substitute values.

Consider CSV data. It can be handled in one of several ways. We can use simple Java techniques such as the `String` class' `split` method. In the following sequence, a string array, `csvArray`, is assumed to hold comma-delimited data. The `split` method populates a second array, `tokenArray`.

```
for(int i=0; i<csvArray.length; i++) {  
    tokenArray[i] = csvArray[i].split(",");  
}
```

More complex data types require APIs to retrieve the data. For example, in [Chapter 3, Data Cleaning](#), we will use the Jackson Project (<https://github.com/FasterXML/jackson>) to retrieve fields from a JSON file. The example uses a file containing a JSON-formatted presentation of a person, as shown next:

```
{  
    "firstname": "Smith",  
    "lastname": "Peter",  
    "phone": 8475552222,  
    "address": ["100 Main Street", "Corpus", "Oklahoma"]  
}
```

The code sequence that follows shows how to extract the values for fields of a person. A parser is created, which uses `getCurrentName` to retrieve a field name. If the name is `firstname`, then the `getText` method returns the value for that field. The other fields are handled in a similar manner.

```

try {
    JsonFactory jsonfactory = new JsonFactory();
    JsonParser parser = jsonfactory.createParser(
        new File("Person.json"));
    while (parser.nextToken() != JsonToken.END_OBJECT) {
        String token = parser.getCurrentName();
        if ("firstname".equals(token)) {
            parser.nextToken();
            String fname = parser.getText();
            out.println("firstname : " + fname);
        }
        ...
    }
    parser.close();
} catch (IOException ex) {
    // Handle exceptions
}

```

The output of this example is as follows:

```
firstname : Smith
```

Simple data cleaning may involve converting the text to lowercase, replacing certain text with blanks, and removing multiple whitespace characters with a single blank. One way of doing this is shown next, where a combination of the `String` class' `toLowerCase`, `replaceAll`, and `trim` methods are used. Here, a string containing dirty text is processed:

```

dirtyText = dirtyText
    .toLowerCase()
    .replaceAll("[\\d[^\\w\\s]]+", " "
    .trim();
while(dirtyText.contains(" ")){
    dirtyText = dirtyText.replaceAll("  ", " ");
}

```

Stop words are words such as *the*, *and*, or *but* that do not always contribute to the analysis of text. Removing these stop words can often improve the results and speed up the processing.

The **LingPipe API** can be used to remove stop words. In the next code sequence, a `TokenizerFactory` class instance is used to tokenize text. Tokenization is the process of returning individual words. The `EnglishStopTokenizerFactory` class is a special tokenizer that removes common English stop words.

```
text = text.toLowerCase().trim();
TokenizerFactory fact = IndoEuropeanTokenizerFactory.INSTANCE;
fact = new EnglishStopTokenizerFactory(fact);
Tokenizer tok = fact.tokenizer(
    text.toCharArray(), 0, text.length());
for(String word : tok){
    out.print(word + " ");
}
```

Consider the following text, which was pulled from the book, Moby Dick:

Call me Ishmael. Some years ago- never mind how long precisely - having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world.

The output will be as follows:

```
call me ishmael . years ago - never mind how long precisely - having
little money my purse , nothing particular interest me shore , i
thought i sail little see watery part world .
```

These are just a couple of the data cleaning tasks discussed in [Chapter 3, Data Cleaning](#).

Visualizing data to enhance understanding

The analysis of data often results in a series of numbers representing the results of the analysis. However, for most people, this way of expressing results is not always intuitive. A better way to understand the results is to create graphs and charts to depict the results and the relationship between the elements of the result.

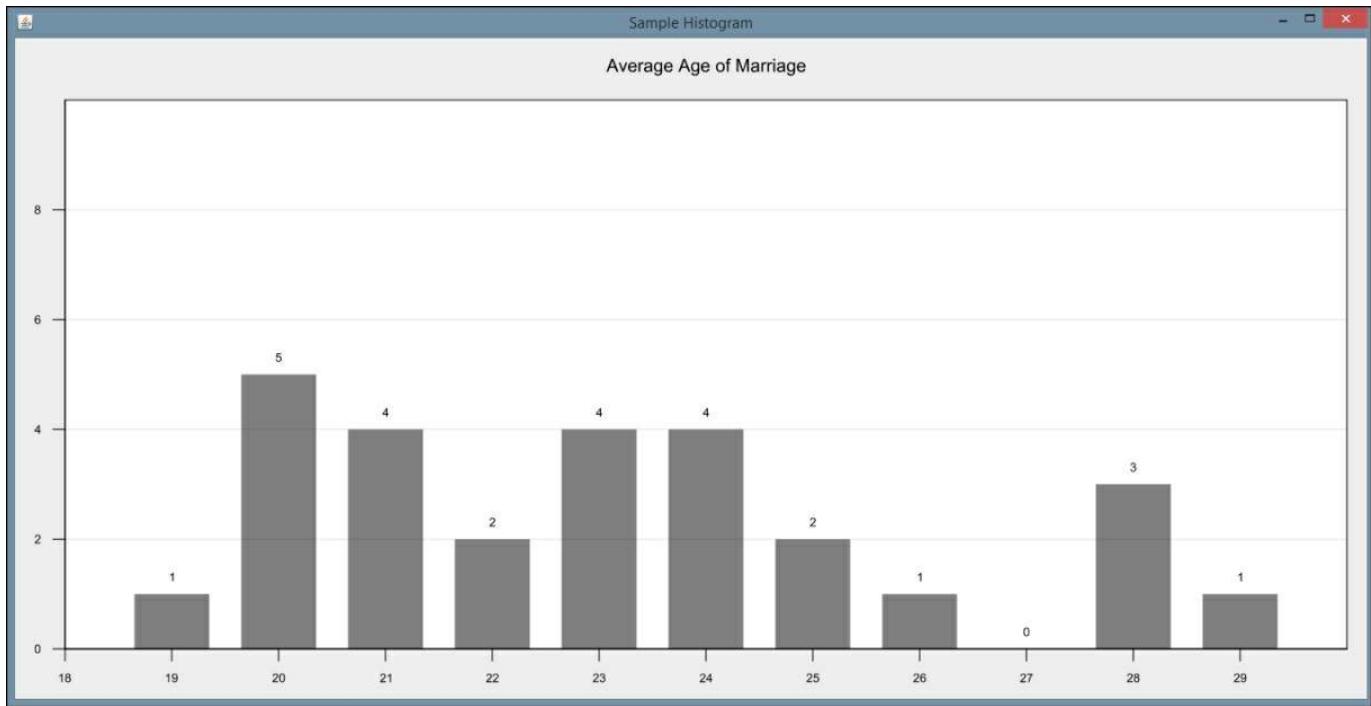
The human mind is often good at seeing patterns, trends, and outliers in visual representation. The large amount of data present in many data science problems can be analyzed using visualization techniques. Visualization is appropriate for a wide range of audiences ranging from analysts to upper-level management to clientele. In this chapter, we present various visualization techniques and demonstrate how they are supported in Java.

In [Chapter 4, Data Visualization](#), we illustrate how to create different types of graphs, plots, and charts. These examples use JavaFX using a free library called **GRAL**(<http://trac.erichseifert.de/gral/>).

Visualization allows users to examine large datasets in ways that provide insights that are not present in the mass of the data. Visualization tools helps us identify potential problems or unexpected data results and develop meaningful interpretations of the data.

For example, outliers, which are values that lie outside of the normal range of values, can be hard to spot from a sea of numbers. Creating a graph based on the data allows users to quickly see outliers. It can also help spot errors quickly and more easily classify data.

For example, the following chart might suggest that the upper two values should be outliers that need to be dealt with:



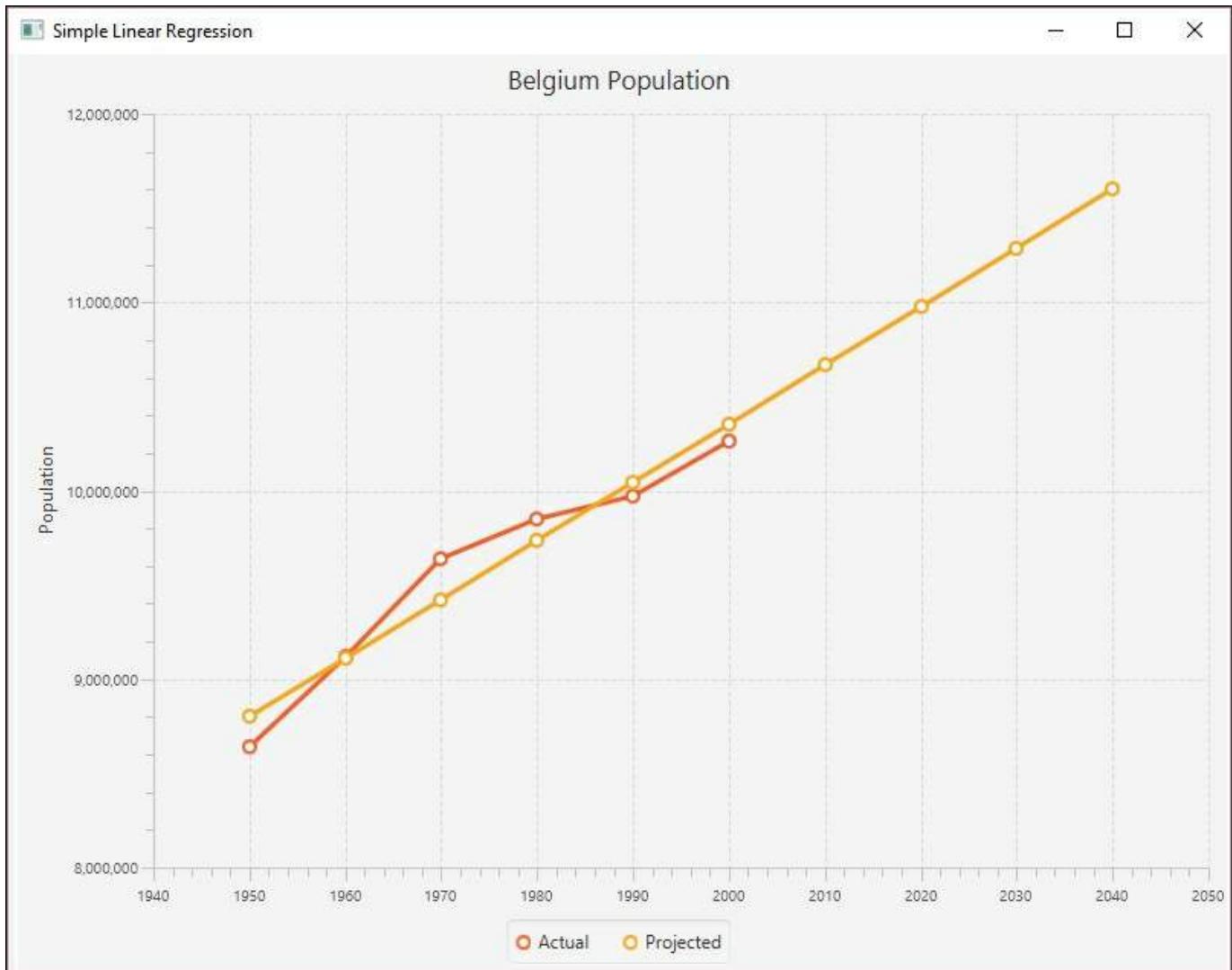
The use of statistical methods in data science

Statistical analysis is the key to many data science tasks. It is used for many types of analysis ranging from the computation of simple mean and medium to complex multiple regression analysis. [Chapter 5, Statistical Data Analysis Techniques](#), introduces this type of analysis and the Java support available.

Statistical analysis is not always an easy task. In addition, advanced statistical techniques often require a particular mindset to fully comprehend, which can be difficult to learn. Fortunately, many techniques are not that difficult to use and various libraries mitigate some of these techniques' inherent complexity.

Regression analysis, in particular, is an important technique for analyzing data. The technique attempts to draw a line that matches a set of data. An equation representing the line is calculated and can be used to predict future behavior. There are several types of regression analysis, including simple and multiple regression. They vary by the number of variables being considered.

The following graph shows the straight line that closely matches a set of data points representing the population of Belgium over several decades:



Simple statistical techniques, such as mean and standard deviation, can be computed using basic Java. They can also be handled by libraries such as Apache Commons. For example, to calculate the median, we can use the Apache Commons DescriptiveStatistics class. This is illustrated next where the median of an array of doubles is calculated. The numbers are added to an instance of this class, as shown here:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2,
    12.5, 17.8, 16.5, 12.5};
DescriptiveStatistics statTest =
    new SynchronizedDescriptiveStatistics();
for(double num : testData){
    statTest.addValue(num);
}
```

The `getPercentile` method returns the value stored at the percentile specified in its argument. To find the median, we use the value of 50.

```
out.println("The median is " + statTest.getPercentile(50));
```

Our output is as follows:

```
The median is 16.2
```

In [Chapter 5, Statistical Data Analysis Techniques](#), we will demonstrate how to perform regression analysis using the Apache Commons `SimpleRegression` class.

Machine learning applied to data science

Machine learning has become increasingly important for data science analysis as it has been for a multitude of other fields. A defining characteristic of machine learning is the ability of a model to be trained on a set of representative data and then later used to solve similar problems. There is no need to explicitly program an application to solve the problem. A model is a representation of the real-world object.

For example, customer purchases can be used to train a model. Subsequently, predictions can be made about the types of purchases a customer might subsequently make. This allows an organization to tailor ads and coupons for a customer and potentially providing a better customer experience.

Training can be performed in one of several different approaches:

- **Supervised learning:** The model is trained with annotated, labeled, data showing corresponding correct results
- **Unsupervised learning:** The data does not contain results, but the model is expected to find relationships on its own
- **Semi-supervised:** A small amount of labeled data is combined with a larger amount of unlabeled data
- **Reinforcement learning:** This is similar to supervised learning, but a reward is provided for good results

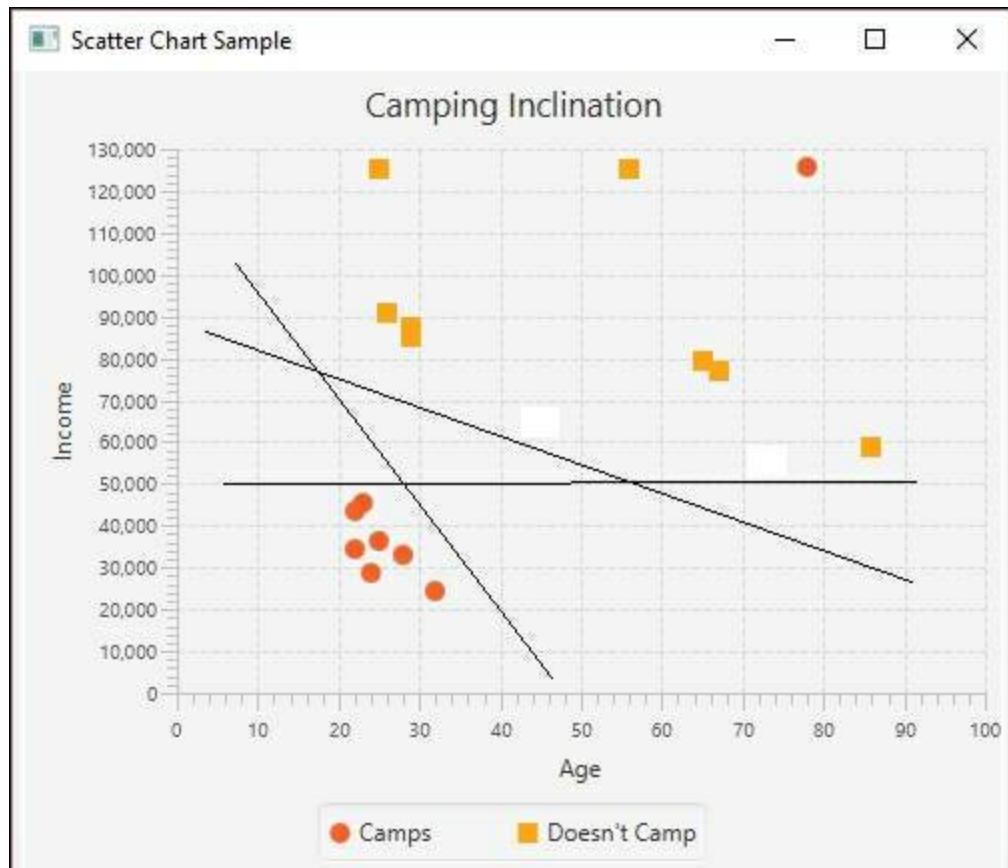
There are several approaches that support machine learning. In [Chapter 6, Machine Learning](#), we will illustrate three techniques:

- **Decision trees:** A tree is constructed using features of the problem as internal nodes and the results as leaves
- **Support vector machines:** This is used for classification by creating a hyperplane that partitions the dataset and then makes predictions
- **Bayesian networks:** This is used to depict probabilistic relationships between events

A **Support Vector Machine (SVM)** is used primarily for classification type problems. The approach creates a hyperplane to categorize data, which can be envisioned as a **geometric plane** that separates two regions. In a two-dimensional

space, it will be a line. In a three-dimensional space, it will be a two-dimensional plane. In [Chapter 6, Machine Learning](#), we will demonstrate how to use the approach using a set of data relating to the propensity of individuals to camp. We will use the Weka class, `SMO`, to demonstrate this type of analysis.

The following figure depicts a hyperplane using a distribution of two types of data points. The lines represent possible hyperplanes that separate these points. The lines clearly separate the data points except for one outlier.



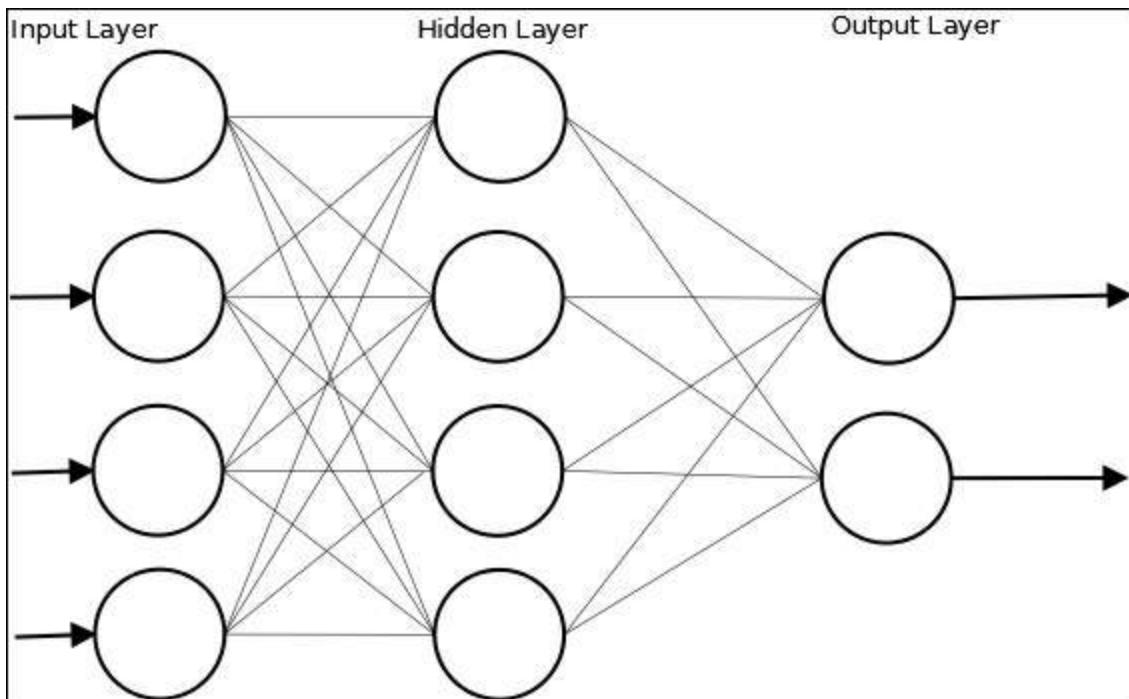
Once the model has been trained, the possible hyperplanes are considered and predictions can then be made using similar data.

Using neural networks in data science

An **Artificial Neural Network (ANN)**, which we will call a **neural network**, is based on the neuron found in the brain. A **neuron** is a cell that has **dendrites** connecting it to input sources and other neurons. Depending on the input source, a weight allocated to a source, the neuron is activated, and then **fires** a signal down a dendrite to another neuron. A collection of neurons can be trained to respond to a set of input signals.

An artificial neuron is a node that has one or more inputs and a single output. Each input has a **weight** assigned to it that can change over time. A neural network can learn by feeding an input into a network, invoking an **activation function**, and comparing the results. This function combines the inputs and creates an output. If outputs of multiple neurons match the expected result, then the network has been trained correctly. If they don't match, then the network is modified.

A neural network can be visualized as shown in the following figure, where **Hidden Layer** is used to augment the process:



In [Chapter 7, Neural Networks](#), we will use the Weka class,

MultilayerPerceptron, to illustrate the creation and use of a **Multi Layer Perceptron (MLP)** network. As we will explain, this type of network is a feedforward neural network with multiple layers. The network uses supervised learning with backpropagation. The example uses a dataset called `dermatology.arff` that contains 366 instances that are used to diagnose erythematous-squamous diseases. It uses 34 attributes to classify the disease into one of the five different categories.

The dataset is split into a training set and a testing set. Once the data has been read, the MLP instance is created and initialized using the method to configure the attributes of the model, including how quickly the model is to learn and the amount of time spent training the model.

```
String trainingFileName = "dermatologyTrainingSet.arff";
String testingFileName = "dermatologyTestingSet.arff";

try (FileReader trainingReader = new FileReader(trainingFileName);
     FileReader testingReader =
         new FileReader(testingFileName)) {
    Instances trainingInstances = new Instances(trainingReader);
    trainingInstances.setClassIndex(
        trainingInstances.numAttributes() - 1);
    Instances testingInstances = new Instances(testingReader);
    testingInstances.setClassIndex(
        testingInstances.numAttributes() - 1);

    MultilayerPerceptron mlp = new MultilayerPerceptron();
    mlp.setLearningRate(0.1);
    mlp.setMomentum(0.2);
    mlp.setTrainingTime(2000);
    mlp.setHiddenLayers("3");
    mlp.buildClassifier(trainingInstances);
    ...
} catch (Exception ex) {
    // Handle exceptions
}
```

The model is then evaluated using the testing data:

```
Evaluation evaluation = new Evaluation(trainingInstances);
evaluation.evaluateModel(mlp, testingInstances);
```

The results can then be displayed:

```
System.out.println(evaluation.toSummaryString());
```

The truncated output of this example is shown here where the number of correctly and incorrectly identified diseases are listed:

```
Correctly Classified Instances 73 98.6486 %
```

```
Incorrectly Classified Instances 1 1.3514 %
```

The various attributes of the model can be tweaked to improve the model. In [Chapter 7, Neural Networks](#), we will discuss this and other techniques in more depth.

Deep learning approaches

Deep learning networks are often described as neural networks that use multiple intermediate layers. Each layer will train on the outputs of a previous layer potentially identifying features and subfeatures of a dataset. The features refer to those aspects of the data that may be of interest. In [Chapter 8, Deep Learning](#), we will examine these types of networks and how they can support several different data science tasks.

These networks often work with unstructured and unlabeled datasets, which is the vast majority of the data available today. A typical approach is to take the data, identify features, and then use these features and their corresponding layers to reconstruct the original dataset, thus validating the network. The **Restricted Boltzmann Machines (RBM)** is a good example of the application of this approach.

The deep learning network needs to ensure that the results are accurate and minimizes any error that can creep into the process. This is accomplished by adjusting the internal weights assigned to neurons based on what is known as **gradient descent**. This represents the slope of the weight changes. The approach modifies the weight so as to minimize the error and also speeds up the learning process.

There are several types of networks that have been classified as a deep learning network. One of these is an **autoencoder** network. In this network, the layers are symmetrical where the number of input values is the same as the number of output values and the intermediate layers effectively compress the data to a single smaller internal layer. Each layer of the autoencoder is a RBM.

This structure is reflected in the following example, which will extract the numbers found in a set of images containing hand-written numbers. The details of the complete example are not shown here, but notice that 1,000 input and output values are used along with internal layers consisting of RBMs. The size of the layers are specified in the `nOut` and `nIn` methods.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(numberOfIterations)
    .optimizationAlgo(
        OptimizationAlgorithm.LINE_GRADIENT_DESCENT)
```

```

.list()
.layer(0, new RBM.Builder()
    .nIn(numberOfRows * numberOfRows).nOut(1000)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(1, new RBM.Builder().nIn(1000).nOut(500)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(2, new RBM.Builder().nIn(500).nOut(250)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(3, new RBM.Builder().nIn(250).nOut(100)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(4, new RBM.Builder().nIn(100).nOut(30)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build()) //encoding stops
.layer(5, new RBM.Builder().nIn(30).nOut(100)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build()) //decoding starts
.layer(6, new RBM.Builder().nIn(100).nOut(250)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(7, new RBM.Builder().nIn(250).nOut(500)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(8, new RBM.Builder().nIn(500).nOut(1000)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
.layer(9, new OutputLayer.Builder(
    LossFunctions.LossFunction.RMSE_XENT).nIn(1000)
    .nOut(numberOfRows * numberOfRows).build())
.pretrain(true).backprop(true)
.build();

```

Once the model has been trained, it can be used for predictive and searching tasks. With a search, the compressed middle layer can be used to match other compressed images that need to be classified.

Performing text analysis

The field of **Natural Language Processing (NLP)** is used for many different tasks including text searching, language translation, sentiment analysis, speech recognition, and classification to mention a few. Processing text is difficult due to a number of reasons, including the inherent ambiguity of natural languages.

There are several different types of processing that can be performed such as:

- **Identifying Stop words:** These are words that are common and may not be necessary for processing
- **Name Entity Recognition (NER):** This is the process of identifying elements of text such as people's names, location, or things
- **Parts of Speech (POS):** This identifies the grammatical parts of a sentence such as noun, verb, adjective, and so on
- **Relationships:** Here we are concerned with identifying how parts of text are related to each other, such as the subject and object of a sentence

As with most data science problems, it is important to preprocess and clean text. In [Chapter 9, Text Analysis](#), we examine the support Java provides for this area of data science.

For example, we will use Apache's OpenNLP (<https://opennlp.apache.org/>) library to find the parts of speech. This is just one of the several NLP APIs that we could have used including LingPipe (<http://alias-i.com/lingpipe/>), Apache UIMA (<https://uima.apache.org/>), and Standford NLP (<http://nlp.stanford.edu/>). We chose OpenNLP because it is easy to use for this example.

In the following example, a model used to identify POS elements is found in the `en-pos-maxent.bin` file. An array of words is initialized and the POS model is created:

```
try (InputStream input = new FileInputStream(
        new File("en-pos-maxent.bin"))); {
    String sentence = "Let's parse this sentence.";
    ...
    String[] words;
    ...
    list.toArray(words);
    POSModel posModel = new POSModel(input);
    ...
} catch (IOException ex) {
```

```
// Handle exceptions  
}
```

The `tag` method is passed an array of `words` and returns an array of tags. The words and tags are then displayed.

```
String[] postTags = posTagger.tag(words);  
for(int i=0; i<postTags.length; i++) {  
    out.println(words[i] + " - " + postTags[i]);  
}
```

The output for this example is as follows:

Let's - NNP

parse - NN

this - DT

sentence. - NN

The abbreviations `NNP` and `DT` stand for a singular proper noun and determiner respectively. We examine several other NLP techniques in [Chapter 9, Text Analysis](#).

Visual and audio analysis

In [Chapter 10](#), *Visual and Audio Analysis*, we demonstrate several Java techniques for processing sounds and images. We begin by demonstrating techniques for sound processing, including speech recognition and text-to-speech APIs. Specifically, we will use the FreeTTS (<http://freetts.sourceforge.net/docs/index.php>) API to convert text to speech. We also include a demonstration of the **CMU Sphinx** toolkit for speech recognition.

The **Java Speech API (JSAPI)** (<http://www.oracle.com/technetwork/java/index-140170.html>) supports speech technology. This API, created by third-party vendors, supports speech recognition and speech synthesizers. FreeTTS and Festival (<http://www.cstr.ed.ac.uk/projects/festival/>) are examples of vendors supporting JSAPI.

In the second part of the chapter, we examine image processing techniques such as facial recognition. This demonstration involves identifying faces within an image and is easy to accomplish using OpenCV (<http://opencv.org/>).

Also, in [Chapter 10](#), *Visual and Audio Analysis*, we demonstrate how to extract text from images, a process known as **OCR**. A common data science problem involves extracting and analyzing text embedded in an image. For example, the information contained in license plate, road signs, and directions can be significant.

In the following example, explained in more detail in [Chapter 11](#), *Mathematical and Parallel Techniques for Data Analysis* accomplishes OCR using Tess4j (<http://tess4j.sourceforge.net/>) a Java JNA wrapper for Tesseract OCR API. We perform OCR on an image captured from the Wikipedia article on OCR (https://en.wikipedia.org/wiki/Optical_character_recognition#Applications), shown here:

OCR engines have been developed into many kinds of object-oriented OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- Automatic insurance documents key information extraction
- Extracting business card information into a contact list^[9]
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg
- Make electronic images of printed documents searchable, e.g. Google Books
- Converting handwriting in real time to control a computer (pen computing)
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR^{[10][11][12]}
- Assistive technology for blind and visually impaired users

The `ITesseract` interface provides numerous OCR methods. The `doOCR` method takes a file and returns a string containing the words found in the file as shown here:

```
ITesseract instance = new Tesseract();
try {
    String result = instance.doOCR(new File("OCRExample.png"));
    System.out.println(result);
} catch (TesseractException e) {
    System.err.println(e.getMessage());
}
```

A part of the output is shown next:

OCR engines nave been developed into many lunds oiobiectorlented OCR
applicatlons, sucn as reoeipt OCR, involoe OCR, check OCR, legal
billing document OCR

They can be used ior

- Data entry ior business documents, e g check, passport, involoe, bank statement and receipt
- Automatic number plate recognnnlon

As you can see, there are numerous errors in this example that need to be addressed. We build upon this example in [Chapter 11, Mathematical and Parallel Techniques for Data Analysis](#), with a discussion of enhancements and considerations to ensure the OCR process is as effective as possible.

We will conclude the chapter with a discussion of NeurophStudio, a neural network Java-based editor, to classify images and perform image recognition. We train a neural network to recognize and classify faces in this section.

Improving application performance using parallel techniques

In [Chapter 11, Mathematical and Parallel Techniques for Data Analysis](#), we consider some of the parallel techniques available for data science applications. Concurrent execution of a program can significantly improve performance. In relation to data science, these techniques range from low-level mathematical calculations to higher-level API-specific options.

This chapter includes a discussion of basic performance enhancement considerations. Algorithms and application architecture matter as much as enhanced code, and this should be considered when attempting to integrate parallel techniques. If an application does not behave in the expected or desired manner, any gains from parallel optimizing are irrelevant.

Matrix operations are essential to many data applications and supporting APIs. We will include a discussion in this chapter about matrix multiplication and how it is handled using a variety of approaches. Even though these operations are often hidden within the API, it can be useful to understand how they are supported.

One approach we demonstrate utilizes the Apache Commons Math API (<http://commons.apache.org/proper/commons-math/>). This API supports a large number of mathematical and statistical operations, including matrix multiplication. The following example illustrates how to perform matrix multiplication.

We first declare and initialize matrices A and B:

```
double[][] A = {  
    {0.1950, 0.0311},  
    {0.3588, 0.2203},  
    {0.1716, 0.5931},  
    {0.2105, 0.3242}};  
  
double[][] B = {  
    {0.0502, 0.9823, 0.9472},  
    {0.5732, 0.2694, 0.916}};
```

Apache Commons uses the `RealMatrix` class to store a matrix. Next, we use the `Array2DRowRealMatrix` constructor to create the corresponding matrices for A and B:

```
RealMatrix aRealMatrix = new Array2DRowRealMatrix(A);
RealMatrix bRealMatrix = new Array2DRowRealMatrix(B);
```

We perform multiplication simply using the `multiply` method:

```
RealMatrix cRealMatrix = aRealMatrix.multiply(bRealMatrix);
```

Finally, we use a `for` loop to display the results:

```
for (int i = 0; i < cRealMatrix.getRowDimension(); i++) {
    System.out.println(cRealMatrix.getRowVector(i));
}
```

The output is as follows:

```
{0.02761552; 0.19992684; 0.2131916}
{0.14428772; 0.41179806; 0.54165016}
{0.34857924; 0.32834382; 0.70581912}
{0.19639854; 0.29411363; 0.4963528}
```

Another approach to concurrent processing involves the use of Java threads. Threads are used by APIs such as Aparapi when multiple CPUs or GPUs are not available.

Data science applications often take advantage of the map-reduce algorithm. We will demonstrate parallel processing by using Apache's Hadoop to perform map-reduce. Designed specifically for large datasets, Hadoop reduces processing time for large scale data science projects. We demonstrate a technique for calculating the average value of a large dataset.

We also include examples of APIs that support multiple processors, including CUDA and OpenCL. CUDA is supported using Java bindings for CUDA (JCuda) (<http://jcuda.org/>). We also discuss OpenCL and its Java support. The Aparapi API provides high-level support for using multiple CPUs or GPUs and we include a demonstration of Aparapi in support of matrix multiplication.

Assembling the pieces

In the final chapter of this book, we will tie together many of the techniques explored in the previous chapters. We will create a simple console-based application for acquiring data from Twitter and performing various types of data manipulation and analysis. Our goal in this chapter is to demonstrate a simple project exploring a variety of data science concepts and provide insights and considerations for future projects.

Specifically, the application developed in the final chapter performs several high-level tasks, including data acquisition, data cleaning, sentiment analysis, and basic statistical collection. We demonstrate these techniques using Java 8 Streams and focus on Java 8 approaches whenever possible.

Summary

Data science is a broad, diverse field of study and it would be impossible to explore exhaustively within this book. We hope to provide a solid understanding of important data science concepts and equip the reader for further study. In particular, this book will provide concrete examples of different techniques for all stages of data science related inquiries. This ranges from data acquisition and cleaning to detailed statistical analysis.

So let's start with a discussion of data acquisition and how Java supports it as illustrated in the next chapter.

Chapter 2. Data Acquisition

It is never much fun to work with code that is not formatted properly or uses variable names that do not convey their intended purpose. The same can be said of data, except that bad data can result in inaccurate results. Thus, data acquisition is an important step in the analysis of data. Data is available from a number of sources but must be retrieved and ultimately processed before it can be useful. It is available from a variety of sources. We can find it in numerous public data sources as simple files, or it may be found in more complex forms across the Internet. In this chapter, we will demonstrate how to acquire data from several of these, including various Internet sites and several social media sites.

We can access data from the Internet by downloading specific files or through a process known as **web scraping**, which involves extracting the contents of a web page. We also explore a related topic known as **web crawling**, which involves applications that examine a web site to determine whether it is of interest and then follows embedded links to identify other potentially relevant pages.

We can also extract data from social media sites. These types of sites often hold a treasure trove of data that is readily available if we know how to access it. In this chapter, we will demonstrate how to extract data from several sites, including:

- Twitter
- Wikipedia
- Flickr
- YouTube

When extracting data from a site, many different data formats may be encountered. We will examine three basic types: text, audio, and video. However, even within text, audio, and video data, many formats exist. For audio data alone, there are 45 audio coding formats compared at

https://en.wikipedia.org/wiki/Comparison_of_audio_coding_formats. For textual data, there are almost 300 formats listed at <http://fileinfo.com/filetypes/text>. In this chapter, we will focus on how to download and extract these types of text as plain text for eventual processing.

We will briefly examine different data formats, followed by an examination of possible data sources. We need this knowledge to demonstrate how to obtain data

using different data acquisition techniques.

Understanding the data formats used in data science applications

When we discuss data formats, we are referring to content format, as opposed to the underlying file format, which may not even be visible to most developers. We cannot examine all available formats due to the vast number of formats available. Instead, we will tackle several of the more common formats, providing adequate examples to address the most common data retrieval needs. Specifically, we will demonstrate how to retrieve data stored in the following formats:

- HTML
- PDF
- CSV/TSV
- Spreadsheets
- Databases
- JSON
- XML

Some of these formats are well supported and documented elsewhere. For example, XML has been in use for years and there are several well-established techniques for accessing XML data in Java. For these types of data, we will outline the major techniques available and show a few examples to illustrate how they work. This will provide those readers who are not familiar with the technology some insight into their nature.

The most common data format is binary files. For example, Word, Excel, and PDF documents are all stored in binary. These require special software to extract information from them. Text data is also very common.

Overview of CSV data

Comma Separated Values (CSV) files, contain tabular data organized in a row-column format. The data, stored as plaintext, is stored in rows, also called **records**. Each record contains fields separated by commas. These files are also closely related to other delimited files, most notably **Tab-Separated Values (TSV)** files. The following is a part of a simple CSV file, and these numbers are not intended to represent any specific type of data:

```
JURISDICTION NAME,COUNT PARTICIPANTS,COUNT FEMALE,PERCENT FEMALE  
10001,44,22,0.5  
10002,35,19,0.54  
10003,1,1,1
```

Notice that the first row contains header data to describe the subsequent records. Each value is separated by a comma and corresponds to the header in the same position. In [Chapter 3, Data Cleaning](#), we will discuss CSV files in more depth and examine the support available for different types of delimiters.

Overview of spreadsheets

Spreadsheets are a form of tabular data where information is stored in rows and columns, much like a two-dimensional array. They typically contain numeric and textual information and use formulas to summarize and analyze their contents. Most people are familiar with Excel spreadsheets, but they are also found as part of other product suites, such as OpenOffice.

Spreadsheets are an important data source because they have been used for the past several decades to store information in many industries and applications. Their tabular nature makes them easy to process and analyze. It is important to know how to extract data from this ubiquitous data source so that we can take advantage of the wealth of information that is stored in them.

For some of our examples, we will use a simple Excel spreadsheet that consists of a series of rows containing an ID, along with minimum, maximum, and average values. These numbers are not intended to represent any specific type of data. The spreadsheet looks like this:

ID	Minimum	Maximum	Average
12345	45	89	65.55
23456	78	96	86.75
34567	56	89	67.44
45678	86	99	95.67

In [Chapter 3, Data Cleaning](#), we will learn how to extract data from spreadsheets.

Overview of databases

Data can be found in **Database Management Systems (DBMS)**, which, like spreadsheets, are ubiquitous. Java provides a rich set of options for accessing and processing data in a DBMS. The intent of this section is to provide a basic introduction to database access using Java.

We will demonstrate the essence of connecting to a database, storing information, and retrieving information using JDBC. For this example, we used the MySQL DBMS. However, it will work for other DBMSes as well with a change in the database driver. We created a database called `example` and a table called `URLTABLE` using the following command within the **MySQL Workbench**. There are other tools that can achieve the same results:

```
CREATE TABLE IF NOT EXISTS `URLTABLE` (
  `RecordID` INT(11) NOT NULL AUTO_INCREMENT,
  `URL` text NOT NULL,
  PRIMARY KEY (`RecordID`)
);
```

We start with a `try` block to handle exceptions. A driver is needed to connect to the DBMS. In this example, we used `com.mysql.jdbc.Driver`. To connect to the database, the `getConnection` method is used, where the database server location, user ID, and password are passed. These values depend on the DBMS used:

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    String url = "jdbc:mysql://localhost:3306/example";
    connection = DriverManager.getConnection(url, "user ID",
        "password");
    ...
} catch (SQLException | ClassNotFoundException ex) {
    // Handle exceptions
}
```

Next, we will illustrate how to add information to the database and then how to read it. The SQL `INSERT` command is constructed in a string. The name of the MySQL database is `example`. This command will insert values into the `URLTABLE` table in the database where the question mark is a placeholder for the value to be inserted:

```
String insertSQL = "INSERT INTO `example`.`URLTABLE` "
    + "(`url`) VALUES " + "(?);";
```

The `PreparedStatement` class represents an SQL statement to execute. The `prepareStatement` method creates an instance of the class using the `INSERT` SQL statement:

```
PreparedStatement stmt = connection.prepareStatement(insertSQL);
```

We then add URLs to the table using the `setString` method and the `execute` method. The `setString` method possesses two arguments. The first specifies the column index to insert the data and the second is the value to be inserted. The `execute` method does the actual insertion. We add two URLs in the next sequence:

```
stmt.setString(1, "https://en.wikipedia.org/wiki/Data_science");
stmt.execute();
stmt.setString(1,
    "https://en.wikipedia.org/wiki/Bishop_Rock,_Isles_of_Scilly");
stmt.execute();
```

To read the data, we use a SQL `SELECT` statement as declared in the `selectSQL` string. This will return all the rows and columns from the `URLTABLE` table. The `createStatement` method creates an instance of a `Statement` class, which is used for `INSERT` type statements. The `executeQuery` method executes the query and returns a `ResultSet` instance that holds the contents of the table:

```
String selectSQL = "select * from URLTABLE";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(selectSQL);
```

The following sequence iterates through the table, displaying one row at a time. The argument of the `getString` method specifies that we want to use the second column of the result set, which corresponds to the URL field:

```
out.println("List of URLs");
while (resultSet.next()) {
    out.println(resultSet.getString(2));
}
```

The output of this example, when executed, is as follows:

```
List of URLs
https://en.wikipedia.org/wiki/Data_science
https://en.wikipedia.org/wiki/Bishop_Rock,_Isles_of_Scilly
```

If you need to empty the contents of the table, use the following sequence:

```
Statement statement = connection.createStatement();
statement.execute("TRUNCATE URLTABLE;");
```

This was a brief introduction to database access using Java. There are many resources available that will provide more in-depth coverage of this topic. For example, Oracle provides a more in-depth introduction to this topic at <https://docs.oracle.com/javase/tutorial/jdbc/>.

Overview of PDF files

The **Portable Document Format (PDF)** is a format not tied to a specific platform or software application. A PDF document can hold formatted text and images. PDF is an open standard, making it useful in a variety of places.

There are a large number of documents stored as PDF, making it a valuable source of data. There are several Java APIs that allow access to PDF documents, including **Apache POI** and **PDFBox**. Techniques for extracting information from a PDF document are illustrated in [Chapter 3, Data Cleaning](#).

Overview of JSON

JavaScript Object Notation (JSON) (<http://www.JSON.org/>) is a data format used to interchange data. It is easy for humans or machines to read and write. JSON is supported by many languages, including Java, which has several JSON libraries listed at <http://www.JSON.org/>.

A JSON entity is composed of a set of name-value pairs enclosed in curly braces. We will use this format in several of our examples. In handling YouTube, we will use a JSON object, some of which is shown next, representing the results of a request from a YouTube video:

```
{  
  "kind": "youtube#searchResult",  
  "etag": "etag",  
  "id": {  
    "kind": string,  
    "videoId": string,  
    "channelId": string,  
    "playlistId": string  
  },  
  ...  
}
```

Accessing the fields and values of such a document is not hard and is illustrated in [Chapter 3, Data Cleaning](#).

Overview of XML

Extensible Markup Language (XML) is a markup language that specifies a standard document format. Widely used to communicate between applications and across the Internet, XML is popular due to its relative simplicity and flexibility. Documents encoded in XML are character-based and easily read by machines and humans.

XML documents contain markup and content characters. These characters allow parsers to classify the information contained within the document. The document consists of tags, and elements are stored within the tags. Elements may also contain other markup tags and form child elements. Additionally, elements may contain attributes or specific characteristics stored as a name-and-value pair.

An XML document must be well-formed. This means it must follow certain rules such as always using closing tags and only a single root tag. Other rules are discussed at https://en.wikipedia.org/wiki/XML#Well-formedness_and_error-handling.

The **Java API for XML Processing (JAXP)** consists of three interfaces for parsing XML data. The **Document Object Model (DOM)** interface parses an XML document and returns a tree structure delineating the structure of the document. The DOM interface parses an entire document as a whole. Alternatively, the **Simple API for XML (SAX)** parses a document one element at a time. SAX is preferable when memory usage is a concern as DOM requires more resources to construct the tree. DOM, however, offers flexibility over SAX in that any element can be accessed at any time and in any order.

The third Java API is known as **Streaming API for XML (StAX)**. This streaming model was designed to accommodate the best parts of DOM and SAX models by granting flexibility without sacrificing resources. StAX exhibits higher performance, with the trade-off being that access is only available to one location in a document at a time. StAX is the preferred technique if you already know how you want to process the document, but it is also popular for applications with limited available memory.

The following is a simple XML file. Each `<text>` represents a tag, labelling the element contained within the tags. In this case, the largest node in our file is `<music>` and contained within it are sets of song data. Each tag within a `<song>` tag describes

another element corresponding to that song. Every tag will eventually have a closing tag, such as </song>. Notice that the first tag contains information about which XML version should be used to parse the file:

```
<?xml version="1.0"?>
<music>
    <song id="1234">
        <artist>Patton, Courtney</artist>
        <name>So This Is Life</name>
        <genre>Country</genre>
        <price>2.99</price>
    </song>
    <song id="5678">
        <artist>Eady, Jason</artist>
        <name>AM Country Heaven</name>
        <genre>Country</genre>
        <price>2.99</price>
    </song>
</music>
```

There are numerous other XML-related technologies. For example, we can validate a specific XML document using either a DTD document or XML schema writing specifically for that XML document. XML documents can be transformed into a different format using XSLT.

Overview of streaming data

Streaming data refers to data generated in a continuous stream and accessed in a sequential, piece-by-piece manner. Much of the data the average Internet user accesses is streamed, including video and audio channels, or text and image data on social media sites. Streaming data is the preferred method when the data is new and changing quickly, or when large data collections are sought.

Streamed data is often ideal for data science research because it generally exists in large quantities and raw format. Much public streaming data is available for free and supported by Java APIs. In this chapter, we are going to examine how to acquire data from streaming sources, including Twitter, Flickr, and YouTube. Despite the use of different techniques and APIs, you will notice similarities between the techniques used to pull data from these sites.

Overview of audio/video/images in Java

There are a large number of formats used to represent images, videos, and audio. This type of data is typically stored in binary format. Analog audio streams are sampled and digitized. Images are often simply collections of bits representing the color of a pixel. The following are links that provide a more in-depth discussion of some of these formats:

- Audio: https://en.wikipedia.org/wiki/Audio_file_format
- Images:https://en.wikipedia.org/wiki/Image_file_formats
- Video: https://en.wikipedia.org/wiki/Video_file_format

Frequently, this type of data can be quite large and must be compressed. When data is compressed two approaches are used. The first is a lossless compression, where less space is used and there is no loss of information. The second is lossy, where information is lost. Losing information is not always a bad thing as sometimes the loss is not noticeable to humans.

As we will demonstrate in [Chapter 3, Data Cleaning](#), this type of data often is compromised in an inconvenient fashion and may need to be cleaned. For example, there may be background noise in an audio recording or an image may need to be smoothed before it can be processed. Image smoothing is demonstrated in [Chapter 3, Data Cleaning](#), using the OpenCV library.

Data acquisition techniques

In this section, we will illustrate how to acquire data from web pages. Web pages contain a potential bounty of useful information. We will demonstrate how to access web pages using several technologies, starting with a low-level approach supported by the `HttpURLConnection` class. To find pages, a web crawler application is often used. Once a useful page has been identified, then information needs to be extracted from the page. This is often performed using an HTML parser. Extracting this information is important because it is often buried amid a clutter of HTML tags and JavaScript code.

Using the HttpURLConnection class

The contents of a web page can be accessed using the `HttpURLConnection` class. This is a low-level approach that requires the developer to do a lot of footwork to extract relevant content. However, he or she is able to exercise greater control over how the content is handled. In some situations, this approach may be preferable to using other API libraries.

We will demonstrate how to download the content of Wikipedia's data science page using this class. We start with a `try/catch` block to handle exceptions. A URL object is created using the data science URL string. The `openConnection` method will create a connection to the Wikipedia server as shown here:

```
try {
    URL url = new URL(
        "https://en.wikipedia.org/wiki/Data_science");
    HttpURLConnection connection = (HttpURLConnection)
        url.openConnection();
    ...
} catch (MalformedURLException ex) {
    // Handle exceptions
} catch (IOException ex) {
    // Handle exceptions
}
```

The `connection` object is initialized with an HTTP GET command. The `connect` method is then executed to connect to the server:

```
connection.setRequestMethod("GET");
connection.connect();
```

Assuming no errors were encountered, we can determine whether the response was successful using the `getResponseCode` method. A normal return value is 200. The content of a web page can vary. For example, the `getContentType` method returns a string describing the page's content. The `getContentLength` method returns its length:

```
out.println("Response Code: " + connection.getResponseCode());
out.println("Content Type: " + connection.getContentType());
out.println("Content Length: " + connection.getContentLength());
```

Assuming that we get an HTML formatted page, the next sequence illustrates how to

get this content. A `BufferedReader` instance is created where one line at a time is read in from the web site and appended to a `BufferedReader` instance. The buffer is then displayed:

```
InputStreamReader isr = new InputStreamReader((InputStream)
    connection.getContent());
BufferedReader br = new BufferedReader(isr);
StringBuilder buffer = new StringBuilder();
String line;
do {
    line = br.readLine();
    buffer.append(line + "\n");
} while (line != null);
out.println(buffer.toString());
```

The abbreviated output is shown here:

```
Response Code: 200
Content Type: text/html; charset=UTF-8
Content Length: -1
<!DOCTYPE html>
<html lang="en" dir="ltr" class="client-nojs">
<head>
<meta charset="UTF-8"/>
<title>Data science - Wikipedia, the free encyclopedia</title>
<script>document.documentElement.className =
...
"wgHostname": "mw1251"})});</script>
</body>
</html>
```

While this is feasible, there are easier methods for getting the contents of a web page. One of these techniques is discussed in the next section.

Web crawlers in Java

Web crawling is the process of traversing a series of interconnected web pages and extracting relevant information from those pages. It does this by isolating and then following links on a page. While there are many precompiled datasets readily available, it may still be necessary to collect data directly off the Internet. Some sources such as news sites are continually being updated and need to be revisited from time to time.

A web crawler is an application that visits various sites and collects information. The web crawling process consists of a series of steps:

1. Select a URL to visit
2. Fetch the page
3. Parse the page
4. Extract relevant content
5. Extract relevant URLs to visit

This process is repeated for each URL visited.

There are several issues that need to be considered when fetching and parsing a page such as:

- **Page importance:** We do not want to process irrelevant pages.
- **Exclusively HTML:** We will not normally follow links to images, for example.
- **Spider traps:** We want to bypass sites that may result in an infinite number of requests. This can occur with dynamically generated pages where one request leads to another.
- **Repetition:** It is important to avoid crawling the same page more than once.
- **Politeness:** Do not make an excessive number of requests to a website. Observe the `robot.txt` files; they specify which parts of a site should not be crawled.

The process of creating a web crawler can be daunting. For all but the simplest needs, it is recommended that one of several open source web crawlers be used. A partial list follows:

- **Nutch:** <http://nutch.apache.org>
- **crawler4j:** <https://github.com/yasserg/crawler4j>
- **JSpider:** <http://j-spider.sourceforge.net/>

- **WebSPHINX**: <http://www.cs.cmu.edu/~rcm/websphinx/>
- **Heritrix**: <https://webarchive.jira.com/wiki/display/Heritrix>

We can either create our own web crawler or use an existing crawler and in this chapter we will examine both approaches. For specialized processing, it can be desirable to use a custom crawler. We will demonstrate how to create a simple web crawler in Java to provide more insight into how web crawlers work. This will be followed by a brief discussion of other web crawlers.

Creating your own web crawler

Now that we have a basic understanding of web crawlers, we are ready to create our own. In this simple web crawler, we will keep track of the pages visited using `ArrayList` instances. In addition, jsoup will be used to parse a web page and we will limit the number of pages we visit. Jsoup (<https://jsoup.org/>) is an open source HTML parser. This example demonstrates the basic structure of a web crawler and also highlights some of the issues involved in creating a web crawler.

We will use the `SimpleWebCrawler` class, as declared here:

```
public class SimpleWebCrawler {

    private String topic;
    private String startingURL;
    private String urlLimiter;
    private final int pageLimit = 20;
    private ArrayList<String> visitedList = new ArrayList<>();
    private ArrayList<String> pageList = new ArrayList<>();
    ...
    public static void main(String[] args) {
        new SimpleWebCrawler();
    }
}
```

The instance variables are detailed here:

Variable	Use
<code>topic</code>	The keyword that needs to be in a page for the page to be accepted
<code>startingURL</code>	The URL of the first page

<code>urlLimiter</code>	A string that must be contained in a link before it will be followed
<code>pageLimit</code>	The maximum number of pages to retrieve
<code>visitedList</code>	The <code>ArrayList</code> containing pages that have already been visited
<code>pageList</code>	An <code>ArrayList</code> containing the URLs of the pages of interest

In the `SimpleWebCrawler` constructor, we initialize the instance variables to begin the search from the Wikipedia page for Bishop Rock, an island off the coast of Italy. This was chosen to minimize the number of pages that might be retrieved. As we will see, there are many more Wikipedia pages dealing with Bishop Rock than one might think.

The `urlLimiter` variable is set to `Bishop_Rock`, which will restrict the embedded links to follow to just those containing that string. Each page of interest must contain the value stored in the `topic` variable. The `visitPage` method performs the actual crawl:

```
public SimpleWebCrawler() {
    startingURL = https://en.wikipedia.org/wiki/Bishop_Rock, "
                  + "Isles_of_Scilly";
    urlLimiter = "Bishop_Rock";
    topic = "shipping route";
    visitPage(startingURL);
}
```

In the `visitPage` method, the `pageList` `ArrayList` is checked to see whether the maximum number of accepted pages has been exceeded. If the limit has been exceeded, then the search terminates:

```
public void visitPage(String url) {
    if (pageList.size() >= pageLimit) {
        return;
    }
    ...
}
```

If the page has already been visited, then we ignore it. Otherwise, it is added to the visited list:

```
if (visitedList.contains(url)) {
    // URL already visited
```

```

    } else {
        visitedList.add(url);
        ...
    }
}

```

`Jsoup` is used to parse the page and return a `Document` object. There are many different exceptions and problems that can occur such as a malformed URL, retrieval timeouts, or simply bad links. The `catch` block needs to handle these types of problems. We will provide a more in-depth explanation of `jsoup` in web scraping in Java:

```

try {
    Document doc = Jsoup.connect(url).get();
    ...
}
} catch (Exception ex) {
    // Handle exceptions
}

```

If the document contains the topic text, then the link is displayed and added to the `pageList` `ArrayList`. Each embedded link is obtained, and if the link contains the limiting text, then the `visitPage` method is called recursively:

```

if (doc.text().contains(topic)) {
    out.println((pageList.size() + 1) + ": [" + url + "]");
    pageList.add(url);

    // Process page links
    Elements questions = doc.select("a[href]");
    for (Element link : questions) {
        if (link.attr("href").contains(urlLimiter)) {
            visitPage(link.attr("abs:href"));
        }
    }
}

```

This approach only examines links in those pages that contain the topic text. Moving the `for` loop outside of the `if` statement will test the links for all pages.

The output follows:

```

1: [https://en.wikipedia.org/wiki/Bishop_Rock,_Isles_of_Scilly]
2: [https://en.wikipedia.org/wiki/Bishop_Rock_Lighthouse]
3: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&oldid=717634231#Lighthouse]

```

```

4: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&diff=prev&oldid=717634231]
5: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&oldid=716622943]
6: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&diff=prev&oldid=716622943]
7: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&oldid=716608512]
8: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&diff=prev&oldid=716608512]
...
20: [https://en.wikipedia.org/w/index.php?
title=Bishop_Rock,_Isles_of_Scilly&diff=prev&oldid=716603919]

```

In this example, we did not save the results of the crawl in an external source. Normally this is necessary and can be stored in a file or database.

Using the crawler4j web crawler

Here we will illustrate the use of the crawler4j (<https://github.com/yasserg/crawler4j>) web crawler. We will use an adapted version of the basic crawler found at <https://github.com/yasserg/crawler4j/tree/master/src/test/java/edu/uci/ics/crawler4j/ex>. We will create two classes: `CrawlerController` and `SampleCrawler`. The former class sets up the crawler while the latter contains the logic that controls what pages will be processed.

As with our previous crawler, we will crawl the Wikipedia article dealing with Bishop Rock. The results using this crawler will be smaller as many extraneous pages are ignored.

Let's look at the `CrawlerController` class first. There are several parameters that are used with the crawler as detailed here:

- **Crawl storage folder:** The location where crawl data is stored
- **Number of crawlers:** This controls the number of threads used for the crawl
- **Politeness delay:** How many seconds to pause between requests
- **Crawl depth:** How deep the crawl will go
- **Maximum number of pages to fetch:** How many pages to fetch
- **Binary data:** Whether to crawl binary data such as PDF files

The basic class is shown here:

```
public class CrawlerController {
```

```

public static void main(String[] args) throws Exception {
    int numberOfCrawlers = 2;
    CrawlConfig config = new CrawlConfig();
    String crawlStorageFolder = "data";

    config.setCrawlStorageFolder(crawlStorageFolder);
    config.setPolitenessDelay(500);
    config.setMaxDepthOfCrawling(2);
    config.setMaxPagesToFetch(20);
    config.setIncludeBinaryContentInCrawling(false);
    ...
}
}

```

Next, the `CrawlController` class is created and configured. Notice the `RobotstxtConfig` and `RobotstxtServer` classes used to handle `robot.txt` files. These files contain instructions that are intended to be read by a web crawler. They provide direction to help a crawler to do a better job such as specifying which parts of a site should not be crawled. This is useful for auto generated pages:

```

PageFetcher pageFetcher = new PageFetcher(config);
RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
RobotstxtServer robotstxtServer =
    new RobotstxtServer(robotstxtConfig, pageFetcher);
CrawlController controller =
    new CrawlController(config, pageFetcher, robotstxtServer);

```

The crawler needs to start at one or more pages. The `addSeed` method adds the starting pages. While we used the method only once here, it can be used as many times as needed:

```

controller.addSeed(
    "https://en.wikipedia.org/wiki/Bishop_Rock,_Isles_of_Scilly");

```

The `start` method will begin the crawling process:

```

controller.start(SampleCrawler.class, numberOfCrawlers);

```

The `SampleCrawler` class contains two methods of interest. The first is the `shouldVisit` method that determines whether a page will be visited and the `visit` method that actually handles the page. We start with the class declaration and the declaration of a Java regular expression class `Pattern` object. It will be one way of determining whether a page will be visited. In this declaration, standard images are

specified and will be ignored:

```
public class SampleCrawler extends WebCrawler {  
    private static final Pattern IMAGE_EXTENSIONS =  
        Pattern.compile(".*\\.(bmp|gif|jpg|png)$");  
  
    ...  
}
```

The `shouldVisit` method is passed a reference to the page where this URL was found along with the URL. If any of the images match, the method returns `false` and the page is ignored. In addition, the URL must start with <https://en.wikipedia.org/wiki/>. We added this to restrict our searches to the Wikipedia website:

```
public boolean shouldVisit(Page referringPage, WebURL url) {  
    String href = url.getURL().toLowerCase();  
    if (IMAGE_EXTENSIONS.matcher(href).matches()) {  
        return false;  
    }  
    return href.startsWith("https://en.wikipedia.org/wiki/");  
}
```

The `visit` method is passed a `Page` object representing the page being visited. In this implementation, only those pages containing the string `shipping route` will be processed. This further restricts the pages visited. When we find such a page, its `URL`, `Text`, and `Text.length` are displayed:

```
public void visit(Page page) {  
    String url = page.getWebURL().getURL();  
  
    if (page.getParseData() instanceof HtmlParseData) {  
        HtmlParseData htmlParseData =  
            (HtmlParseData) page.getParseData();  
        String text = htmlParseData.getText();  
        if (text.contains("shipping route")) {  
            out.println("\nURL: " + url);  
            out.println("Text: " + text);  
            out.println("Text length: " + text.length());  
        }  
    }  
}
```

The following is the truncated output of the program when executed:

```
URL: https://en.wikipedia.org/wiki/Bishop_Rock,_Isles_of_Scilly
Text: Bishop Rock, Isles of Scilly...From Wikipedia, the free
encyclopedia ... Jump to: ... navigation, search For the Bishop Rock in
the Pacific Ocean, see Cortes Bank. Bishop Rock Bishop Rock Lighthouse
(2005)
...
Text length: 14677
```

Notice that only one page was returned. This web crawler was able to identify and ignore previous versions of the main web page.

We could perform further processing, but this example provides some insight into how the API works. Significant amounts of information can be obtained when visiting a page. In the example, we only used the URL and the length of the text. The following is a sample of other data that you may be interested in obtaining:

- URL path
- Parent URL
- Anchor
- HTML text
- Outgoing links
- Document ID

Web scraping in Java

Web scraping is the process of extracting information from a web page. The page is typically formatted using a series of HTML tags. An HTML parser is used to navigate through a page or series of pages and to access the page's data or metadata.

Jsoup (<https://jsoup.org/>) is an open source Java library that facilitates extracting and manipulating HTML documents using an HTML parser. It is used for a number of purposes, including web scraping, extracting specific elements from an HTML page, and cleaning up HTML documents.

There are several ways of obtaining an HTML document that may be useful. The HTML document can be extracted from a:

- URL
- String
- File

The first approach is illustrated next where the Wikipedia page for data science is loaded into a `Document` object. This `Jsoup` object represents the HTML document. The `connect` method connects to the site and the `get` method retrieves the `document`:

```
try {
    Document document = Jsoup.connect(
        "https://en.wikipedia.org/wiki/Data_science").get();
    ...
} catch (IOException ex) {
    // Handle exception
}
```

Loading from a file uses the `File` class as shown next. The overloaded `parse` method uses the file to create the `document` object:

```
try {
    File file = new File("Example.html");
    Document document = Jsoup.parse(file, "UTF-8", "");
    ...
} catch (IOException ex) {
    // Handle exception
}
```

The `Example.html` file follows:

```

<html>
<head><title>Example Document</title></head>
<body>
<p>The body of the document</p>
Interesting Links:
<br>
<a href="https://en.wikipedia.org/wiki/Data_science">Data Science</a>
<br>
<a href="https://en.wikipedia.org/wiki/Jsoup">Jsoup</a>
<br>
Images:
<br>

</body>
</html>

```

To create a `Document` object from a string, we will use the following sequence where the `parse` method processes the string that duplicates the previous HTML file:

```

String html = "<html>\n"
    + "<head><title>Example Document</title></head>\n"
    + "<body>\n"
    + "<p>The body of the document</p>\n"
    + "Interesting Links:\n"
    + "<br>\n"
    + "<a href='https://en.wikipedia.org/wiki/Data_science'>" +
        "DataScience</a>\n"
    + "<br>\n"
    + "<a href='https://en.wikipedia.org/wiki/Jsoup'>" +
        "Jsoup</a>\n"
    + "<br>\n"
    + "Images:\n"
    + "<br>\n"
    + "  <img src='eyechart.jpg' alt='Eye Chart'> \n"
    + "</body>\n"
    + "</html>";
Document document = Jsoup.parse(html);

```

The `Document` class possesses a number of useful methods. The `title` method returns the title. To get the text contents of the document, the `select` method is used. This method uses a string specifying the element of a document to retrieve:

```

String title = document.title();
out.println("Title: " + title);
Elements element = document.select("body");
out.println("  Text: " + element.text());

```

The output for the Wikipedia data science page is shown here. It has been shortened to conserve space:

```
Title: Data science - Wikipedia, the free encyclopedia
Text: Data science From Wikipedia, the free encyclopedia Jump to:
navigation, search Not to be confused with information science. Part of
a
...
policy About Wikipedia Disclaimers Contact Wikipedia Developers Cookie
statement Mobile view
```

The parameter type of the `select` method is a string. By using a string, the type of information selected is easily changed. Details on how to formulate this string are found at the jsoup Javadocs for the `Selector` class at <https://jsoup.org/apidocs/>:

We can use the `select` method to retrieve the images in a document, as shown here:

```
Elements images = document.select("img[src$=.png]");
for (Element image : images) {
    out.println("\nImage: " + image);
}
```

The output for the Wikipedia data science page is shown here. It has been shortened to conserve space:

```
Image: 
```

Links can be easily retrieved as shown next:

```
Elements links = document.select("a[href]");
for (Element link : links) {
    out.println("Link: " + link.attr("href")
        + " Text: " + link.text());
}
```

The output for the `Example.html` page is shown here:

```
Link: https://en.wikipedia.org/wiki/Data_science Text: Data Science
Link: https://en.wikipedia.org/wiki/Jsoup Text: Jsoup
```

jsoup possesses many additional capabilities. However, this example demonstrates the web scraping process. There are also other Java HTML parsers available. A comparison of Java HTML parser, among others, can be found at

[https://en.wikipedia.org/wiki/Comparison_of_HTML_parsers.](https://en.wikipedia.org/wiki/Comparison_of_HTML_parsers)

Using API calls to access common social media sites

Social media contain a wealth of information that can be processed and is used by many data analysis applications. In this section, we will illustrate how to access a few of these sources using their Java APIs. Most of them require some sort of access key, which is normally easy to obtain. We start with a discussion on the `OAuth` class, which provides one approach to authenticating access to a data source.

When working with the type of data source, it is important to keep in mind that the data is not always public. While it may be accessible, the owner of the data may be an individual who does not necessarily want the information shared. Most APIs provide a means to determine how the data can be distributed, and these requests should be honored. When private information is used, permission from the author must be obtained.

In addition, these sites have limits on the number of requests that can be made. Keep this in mind when pulling data from a site. If these limits need to be exceeded, then most sites provide a way of doing this.

Using OAuth to authenticate users

OAuth is an open standard used to authenticate users to many different websites. A resource owner effectively delegates access to a server resource without having to share their credentials. It works over HTTPS. OAuth 2.0 succeeded OAuth and is not backwards compatible. It provides client developers a simple way of providing authentication. Several companies use OAuth 2.0 including PayPal, Comcast, and Blizzard Entertainment.

A list of OAuth 2.0 providers is found at https://en.wikipedia.org/wiki/List_of_OAuth_providers. We will use several of these in our discussions.

Handing Twitter

The sheer volume of data and the popularity of the site, among celebrities and the general public alike, make Twitter a valuable resource for mining social media data. Twitter is a popular social media platform allowing users to read and post short

messages called **tweets**. Twitter provides API support for posting and pulling tweets, including streaming data from all public users. While there are services available for pulling the entire set of public tweet data, we are going to examine other options that, while limiting in the amount of data retrieved at one time, are available at no cost.

We are going to focus on the Twitter API for retrieving streaming data. There are other options for retrieving tweets from a specific user as well as posting data to a specific account but we will not be addressing those in this chapter. The public stream API, at the default access level, allows the user to pull a sample of public tweets currently streaming on Twitter. It is possible to refine the data by specifying parameters to track keywords, specific users, and location.

We are going to use HBC, a Java HTTP client, for this example. You can download a sample HBC application at <https://github.com/twitter/hbc>. If you prefer to use a different HTTP client, ensure it will return incremental response data. The Apache HTTP client is one option. Before you can create the HTTP connection, you must first create a Twitter account and an application within that account. To get started with the app, visit apps.twitter.com. Once your app is created, you will be assigned a consumer key, consumer secret, access token, and access secret token. We will also use OAuth, as discussed previously in this chapter.

First, we will write a method to perform the authentication and request data from Twitter. The parameters for our method are the authentication information given to us by Twitter when we created our app. We will create a `BlockingQueue` object to hold our streaming data. For this example, we will set a default capacity of 10,000. We will also specify our endpoint and turn off stall warnings:

```
public static void streamTwitter(
    String consumerKey, String consumerSecret,
    String accessToken, String accessSecret)
    throws InterruptedException {

    BlockingQueue<String> statusQueue =
        new LinkedBlockingQueue<String>(10000);
    StatusesSampleEndpoint ending =
        new StatusesSampleEndpoint();
    ending.stallWarnings(false);
    ...
}
```

Next, we create an `Authentication` object using `OAuth1`, a variation of the `OAuth` class. We can then build our connection client and complete the HTTP connection:

```
Authentication twitterAuth = new OAuth1(consumerKey,
    consumerSecret, accessToken, accessSecret);
BasicClient twitterClient = new ClientBuilder()
    .name("Twitter client")
    .hosts(Constants.STREAM_HOST)
    .endpoint(ending)
    .authentication(twitterAuth)
    .processor(new StringDelimitedProcessor(statusQueue) )
    .build();
twitterClient.connect();
```

For the purposes of this example, we will simply read the messages received from the stream and print them to the screen. The messages are returned in JSON format and the decision of how to process them in a real application will depend upon the purpose and limitations of that application:

```
for (int msgRead = 0; msgRead < 1000; msgRead++) {
    if (twitterClient.isDone()) {
        out.println(twitterClient.getExitEvent().getMessage());
        break;
    }

    String msg = statusQueue.poll(10, TimeUnit.SECONDS);
    if (msg == null) {
        out.println("Waited 10 seconds - no message received");
    } else {
        out.println(msg);
    }
}
twitterClient.stop();
```

To execute our method, we simply pass our authentication information to the `streamTwitter` method. For security purposes, we have replaced our personal keys here. Authentication information should always be protected:

```
public static void main(String[] args) {

    try {
        SampleStreamExample.streamTwitter(
            myKey, mySecret, myToken, myAccess);
    } catch (InterruptedException e) {
        out.println(e);
    }
}
```

```
}
```

Here is truncated sample data retrieved using the methods listed above. Your data will vary based upon Twitter's live stream, but it should resemble this example:

```
{"created_at": "Fri May 20 15:47:21 +0000  
2016", "id": 733685552789098496, "id_str": "733685552789098496", "text": "bwi  
sit si em bahala sya", "source": "\u003ca href=\"http://twitter.com\"\nrel=\"nofollow\"\u003eTwitter Web  
...  
ntions": [], "symbols":  
[]}, "favorited": false, "retweeted": false, "filter_level": "low", "lang": "tl  
", "timestamp_ms": "1463759241660"}
```

Twitter also provides support for pulling all data for one specific user account, as well as posting data directly to an account. A REST API is also available and provides support for specific queries via the search API. These also use the OAuth standard and return data in JSON files.

Handling Wikipedia

Wikipedia (<https://www.wikipedia.org/>) is a useful source of text and image type information. It is an Internet encyclopedia that hosts 38 million articles written in over 250 languages (<https://en.wikipedia.org/wiki/Wikipedia>). As such, it is useful to know how to programmatically access its contents.

MediaWiki is an open source wiki application that supports wiki type sites. It is used to support Wikipedia and many other sites. The MediaWiki API (<http://www.mediawiki.org/wiki/API>) provides access to a wiki's data and metadata over HTTP. An application, using this API, can log in, read data, and post changes to a site.

There are several Java APIs that support programmatic access to a wiki site as listed at https://www.mediawiki.org/wiki/API:Client_code#Java. To demonstrate Java access to a wiki we will use Bliki found at <https://bitbucket.org/axelclk/info.bliki.wiki/wiki/Home>. It provides good access and is easy to use for most basic operations.

The MediaWiki API is complex and has many features. The intent of this section is to illustrate the basic process of obtaining text from a Wikipedia article using this API. It is not possible to cover the API completely here.

We will use the following classes from the `info.bliki.api` and `info.bliki.wiki.model` packages:

- `Page`: Represents a retrieved page
- `User`: Represents a user
- `WikiModel`: Represents the wiki

Javadocs for Bliki are found at <http://www.javadoc.io/doc/info.bliki.wiki/bliki-core/3.1.0>.

The following example has been adapted from <http://www.integratingstuff.com/2012/04/06/hook-into-wikipedia-using-java-and-the-mediawiki-api/>. This example will access the English Wikipedia page for the subject, data science. We start by creating an instance of the `User` class. The first two arguments of the three-argument constructor are the `user ID` and `password`, respectively. In this case, they are empty strings. This combination allows us to read a page without having to set up an account. The third argument is the URL for the MediaWiki API page:

```
User user = new User("", "",  
    "http://en.wikipedia.org/w/api.php");  
user.login();
```

An account will enable us to modify the document. The `queryContent` method returns a list of `Page` objects for the subjects found in a string array. Each string should be the title of a page. In this example, we access a single page:

```
String[] titles = {"Data science"};  
List<Page> pageList = user.queryContent(titles);
```

Each `Page` object contains the content of a page. There are several methods that will return the contents of the page. For each page, a `WikiModel` instance is created using the two-argument constructor. The first argument is the image base URL and the second argument is the link base URL. These URLs use Wiki variables called `image` and `title`, which will be replaced when creating links:

```
for (Page page : pageList) {  
    WikiModel wikiModel = new WikiModel("${image}",  
        "${title}");  
    ...  
}
```

The `render` method will take the wiki page and render it to HTML. There is also a method to render the page to a PDF document:

```
String htmlText = wikiModel.render(page.toString());
```

The HTML text is then displayed:

```
out.println(htmlText);
```

A partial listing of the output follows:

```
<p>PageID: 35458904; NS: 0; Title: Data science;  
Image url:  
Content:  
{{distinguish}}  
{{Use dmy dates}}  
{{Data Visualization}}</p>  
<p><b>Data science</b> is an interdisciplinary field about processes  
and systems to extract <a href="Knowledge" >knowledge</a>  
...
```

We can also obtain basic information about the article using one of several methods as shown here:

```
out.println("Title: " + page.getTitle() + "\n" +  
"Page ID: " + page.getPageid() + "\n" +  
"Timestamp: " + page.getCurrentRevision().getTimestamp());
```

It is also possible to obtain a list of references in the article and a list of the headers. Here, a list of the references is displayed:

```
List <Reference> referenceList = wikiModel.getReferences();  
out.println(referenceList.size());  
for(Reference reference : referenceList) {  
    out.println(reference.getRefString());  
}
```

The following illustrates the process of getting the section headers:

```
ITableOfContent toc = wikiModel.getTableOfContent();  
List<SectionHeader> sections = toc.getSectionHeaders();  
for(SectionHeader sh : sections) {  
    out.println(sh.getFirst());  
}
```

The entire content of Wikipedia can be downloaded. This process is discussed at

https://en.wikipedia.org/wiki/Wikipedia:Database_download.

It may be desirable to set up your own Wikipedia server to handle your request.

Handling Flickr

Flickr (<https://www.flickr.com/>) is an online photo management and sharing application. It is a possible source for images and videos. The Flickr Developer Guide (<https://www.flickr.com/services/developer/>) is a good starting point to learn more about Flickr's API.

One of the first steps to using the Flickr API is to request an API key. This key is used to sign your API requests. The process to obtain a key starts at <https://www.flickr.com/services/apps/create/>. Both commercial and noncommercial keys are available. When you obtain a key you will also get a "secret." Both of these are required to use the API.

We will illustrate the process of locating and downloading images from Flickr. The process involves:

- Creating a Flickr class instance
- Specifying the search parameters for a query
- Performing the search
- Downloading the image

A `FlickrException` or `IOException` may be thrown during this process. There are several APIs that support Flickr access. We will be using Flickr4Java, found at <https://github.com/callmeal/Flickr4Java>. The Flickr4Java Javadocs is found at <http://flickrj.sourceforge.net/api/>. We will start with a `try` block and the `apikey` and `secret` declarations:

```
try {
    String apikey = "Your API key";
    String secret = "Your secret";

} catch (FlickrException | IOException ex) {
    // Handle exceptions
}
```

The `flickr` instance is created next, where the `apikey` and `secret` are supplied as the first two parameters. The last parameter specifies the transfer technique used to

access Flickr servers. Currently, the REST transport is supported using the `REST` class:

```
Flickr flickr = new Flickr(apikey, secret, new REST());
```

To search for images, we will use the `SearchParameters` class. This class supports a number of criteria that will narrow down the number of images returned from a query and includes such criteria as latitude, longitude, media type, and user ID. In the following sequence, the `setBBox` method specifies the longitude and latitude for the search. The parameters are (in order): minimum longitude, minimum latitude, maximum longitude, and maximum latitude. The `setMedia` method specifies the type of media. There are three possible arguments — "all", "photos", and "videos":

```
SearchParameters searchParameters = new SearchParameters();
searchParameters.setBBox("-180", "-90", "180", "90");
searchParameters.setMedia("photos");
```

The `PhotosInterface` class possesses a `search` method that uses the `SearchParameters` instance to retrieve a list of photos. The `getPhotosInterface` method returns an instance of the `PhotosInterface` class, as shown next. The `SearchParameters` instance is the first parameter. The second parameter determines how many photos are retrieved per page and the third parameter is the offset. A `PhotoList` class instance is returned:

```
PhotosInterface pi = new PhotosInterface(apikey, secret,
                                         new REST());
PhotoList<Photo> list = pi.search(searchParameters, 10, 0);
```

The next sequence illustrates the use of several methods to get information about the images retrieved. Each `Photo` instance is accessed using the `get` method. The title, image format, public flag, and photo URL are displayed:

```
out.println("Image List");
for (int i = 0; i < list.size(); i++) {
    Photo photo = list.get(i);
    out.println("Image: " + i +
               `"\nTitle: " + photo.getTitle() +
               "\nMedia: " + photo.getOriginalFormat() +
               "\nPublic: " + photo.isPublicFlag() +
               "\nUrl: " + photo.getUrl() +
               "\n");
```

```
out.println();
```

A partial listing is shown here where many of the specific values have been modified to protect the original data:

```
Image List
Image: 0
Title: XYZ Image
Media: jpg
Public: true
Url: https://flickr.com/photos/7723...@N02/269...
Image: 1
Title: IMG_5555.jpg
Media: jpg
Public: true
Url: https://flickr.com/photos/2665...@N07/264...
Image: 2
Title: DSC05555
Media: jpg
Public: true
Url: https://flickr.com/photos/1179...@N04/264...
```

The list of images returned by this example will vary since we used a fairly wide search range and images are being added all of the time.

There are two approaches that we can use to download an image. The first uses the image's URL and the second uses a `Photo` object. The image's URL can be obtained from a number of sources. We use the `Photo` class `getUrl` method for this example.

In the following sequence, we obtain an instance of `PhotosInterface` using its constructor to illustrate an alternate approach:

```
PhotosInterface pi = new PhotosInterface(apikey, secret,
    new REST());
```

We get the first `Photo` instance from the previous list and then its `getUrl` to get the image's URL. The `PhotosInterface` class's `getImage` method returns a `BufferedImage` object representing the image as shown here:

```
Photo currentPhoto = list.get(0);
BufferedImage bufferedImage =
    pi.getImage(currentPhoto.getUrl());
```

The image is then saved to a file using the `ImageIO` class:

```
File outputfile = new File("image.jpg");
ImageIO.write(bufferedImage, "jpg", outputfile);
```

The `getImage` method is overloaded. Here, the `Photo` instance and the size of the image desired are used as arguments to get the `BufferedImage` instance:

```
bufferedImage = pi.getImage(currentPhoto, Size.SMALL);
```

The image can be saved to a file using the previous technique.

The Flickr4Java API supports a number of other techniques for working with Flickr images.

Handling YouTube

YouTube is a popular video site where users can upload and share videos (<https://www.youtube.com/>). It has been used to share humorous videos, provide instructions on how to do any number of things, and share information among its viewers. It is a useful source of information as it captures the thoughts and ideas of a diverse group of people. This provides an interesting opportunity to analysis and gain insight into human behavior.

YouTube can serve as a useful source of videos and video metadata. A Java API is available to access its contents (<https://developers.google.com/youtube/v3/>). Detailed documentation of the API is found at <https://developers.google.com/youtube/v3/docs/>.

In this section, we will demonstrate how to search for videos by keyword and retrieve information of interest. We will also show how to download a video. To use the YouTube API, you will need a Google account, which can be obtained at <https://www.google.com/accounts/NewAccount>. Next, create an account in the Google Developer's Console (<https://console.developers.google.com/>). API access is supported using either API keys or OAuth 2.0 credentials. The project creation process and keys are discussed at https://developers.google.com/youtube/registering_an_application#create_project.

Searching by keyword

The process of searching for videos by keyword is adapted from https://developers.google.com/youtube/v3/code_samples/java#search_by_keyword. Other potentially useful code examples can be found at

https://developers.google.com/youtube/v3/code_samples/java. The process has been simplified so that we can focus on the search process. We start with a try block and the creation of a YouTube instance. This class provides the basic access to the API. Javadocs for this API is found at <https://developers.google.com/resources/api-libraries/documentation/youtube/v3/java/latest/>.

The YouTube.Builder class is used to construct a YouTube instance. Its constructor takes three arguments:

- Transport: Object used for HTTP
- JSONFactory: Used to process JSON objects
- HttpRequestInitializer: None is needed for this example

Many of the APIs responses will be in the form of JSON objects. The YouTube class' setApplicationName method gives it a name and the build method creates a new YouTube instance:

```
try {
    YouTube youtube = new YouTube.Builder(
        Auth.HTTP_TRANSPORT,
        Auth.JSON_FACTORY,
        new HttpRequestInitializer() {
            public void initialize(HttpRequest request)
                throws IOException {
            }
        })
        .setApplicationName("application_name")
    ...
} catch (GoogleJSONException ex) {
    // Handle exceptions
} catch (IOException ex) {
    // Handle exceptions
}
```

Next, we initialize a string to hold the search term of interest. In this case, we will look for videos containing the word cats:

```
String queryTerm = "cats";
```

The class, YouTube.Search.List, maintains a collection of search results. The YouTube class's search method specifies the type of resource to be returned. In this case, the string specifies the id and snippet portions of the search result to be returned:

```
YouTube.Search.List search = youtube
    .search()
    .list("id,snippet");
```

The search result is a JSON object that has the following structure. It is described in more detail at

<https://developers.google.com/youtube/v3/docs/playlistItems#methods>. In the previous sequence, only the `id` and `snippet` parts of a search will be returned, resulting in a more efficient operation:

```
{
  "kind": "youtube#searchResult",
  "etag": etag,
  "id": {
    "kind": string,
    "videoId": string,
    "channelId": string,
    "playlistId": string
  },
  "snippet": {
    "publishedAt": datetime,
    "channelId": string,
    "title": string,
    "description": string,
    "thumbnails": {
      (key): {
        "url": string,
        "width": unsigned integer,
        "height": unsigned integer
      }
    },
    "channelTitle": string,
    "liveBroadcastContent": string
  }
}
```

Next, we need to specify the API key and various search parameters. The query term is specified, as well as the type of media to be returned. In this case, only videos will be returned. The other two options include `channel` and `playlist`:

```
String apiKey = "Your API key";
search.setKey(apiKey);
search.setQ(queryTerm);
search.setType("video");
```

In addition, we further specify the fields to be returned as shown here. These

correspond to fields of the JSON object:

```
search.setFields("items(id/kind,id/videoId,snippet/title," +
    "snippet/description,snippet/thumbnails/default/url)");
```

We also specify the maximum number of results to retrieve using the `setMaxResults` method:

```
search.setMaxResults(10L);
```

The `execute` method will perform the actual query, returning a `SearchListResponse` object. Its `getItems` method returns a list of `SearchResult` objects, one for each video retrieved:

```
SearchListResponse searchResponse = search.execute();
List<SearchResult> searchResultList =
    searchResponse.getItems();
```

In this example, we do not iterate through each video returned. Instead, we retrieve the first video and display information about the video. The `SearchResult` `video` variable allows us to access different parts of the JSON object, as shown here:

```
SearchResult video = searchResultList.iterator().next();
Thumbnail thumbnail = video
    .getSnippet().getThumbnails().getDefault();

out.println("Kind: " + video.getKind());
out.println("Video Id: " + video.getId().getVideoId());
out.println("Title: " + video.getSnippet().getTitle());
out.println("Description: " +
    video.getSnippet().getDescription());
out.println("Thumbnail: " + thumbnail.getUrl());
```

One possible output follows where parts of the output have been modified:

```
Kind: null
Video Id: tnt0...
Title: Funny Cats ...
Description: Check out the ...
Thumbnail: https://i.ytimg.com/vi/tnt0.../default.jpg
```

We have skipped many error checking steps to simplify the example, but these should be considered when implementing this in a business application.

If we need to download the video, one of the simplest ways is to use axet/wget found

at <https://github.com/axet/wget>. It provides an easy-to-use technique to download the video using its video ID.

In the following example, a URL is created using the video ID. You will need to provide a video ID for this to work properly. The file is saved to the current directory with the video's title as the filename:

```
String url = "http://www.youtube.com/watch?v=videoID";
String path = ".";
VGet vget = new VGet(new URL(url), new File(path));
vget.download();
```

There are other more sophisticated download techniques found at the GitHub site.

Summary

In this chapter, we discussed types of data that are useful for data science and readily accessible on the Internet. This discussion included details about file specifications and formats for the most common types of data sources.

We also examined Java APIs and other techniques for retrieving data, and illustrated this process with multiple sources. In particular, we focused on types of text-based document formats and multimedia files. We used web crawlers to access websites and then performed web scraping to retrieve data from the sites we encountered.

Finally, we extracted data from social media sites and examined the available Java support. We retrieved data from Twitter, Wikipedia, Flickr, and YouTube and examined the available API support.

Chapter 3. Data Cleaning

Real-world data is frequently dirty and unstructured, and must be reworked before it is usable. Data may contain errors, have duplicate entries, exist in the wrong format, or be inconsistent. The process of addressing these types of issues is called **data cleaning**. Data cleaning is also referred to as **data wrangling, massaging, reshaping , or munging**. Data merging, where data from multiple sources is combined, is often considered to be a data cleaning activity.

We need to clean data because any analysis based on inaccurate data can produce misleading results. We want to ensure that the data we work with is quality data. Data quality involves:

- **Validity:** Ensuring that the data possesses the correct form or structure
- **Accuracy:** The values within the data are truly representative of the dataset
- **Completeness:** There are no missing elements
- **Consistency:** Changes to data are in sync
- **Uniformity:** The same units of measurement are used

There are several techniques and tools used to clean data. We will examine the following approaches:

- Handling different types of data
- Cleaning and manipulating text data
- Filling in missing data
- Validating data

In addition, we will briefly examine several image enhancement techniques.

There are often many ways to accomplish the same cleaning task. For example, there are a number of GUI tools that support data cleaning, such as OpenRefine (<http://openrefine.org/>). This tool allows a user to read in a dataset and clean it using a variety of techniques. However, it requires a user to interact with the application for each dataset that needs to be cleaned. It is not conducive to automation.

We will focus on how to clean data using Java code. Even then, there may be different techniques to clean the data. We will show multiple approaches to provide the reader with insights on how it can be done. Sometimes, this will use core Java string classes, and at other time, it may use specialized libraries.

These libraries often are more expressive and efficient. However, there are times when using a simple string function is more than adequate to address the problem. Showing complimentary techniques will improve the reader's skill set.

The basic text based tasks include:

- Data transformation
- Data imputation (handling missing data)
- Subsetting data
- Sorting data
- Validating data

In this chapter, we are interested in cleaning data. However, part of this process is extracting information from various data sources. The data may be stored in plaintext or in binary form. We need to understand the various formats used to store data before we can begin the cleaning process. Many of these formats were introduced in [Chapter 2, Data Acquisition](#), but we will go into greater detail in the following sections.

Handling data formats

Data comes in all types of forms. We will examine the more commonly used formats and show how they can be extracted from various data sources. Before we can clean data it needs to be extracted from a data source such as a file. In this section, we will build upon the introduction to data formats found in [Chapter 2, Data Acquisition](#), and show how to extract all or part of a dataset. For example, from an HTML page we may want to extract only the text without markup. Or perhaps we are only interested in its figures.

These data formats can be quite complex. The intent of this section is to illustrate the basic techniques commonly used with that data format. Full treatment of a specific data format is beyond the scope of this book. Specifically, we will introduce how the following data formats can be processed from Java:

- CSV data
- Spreadsheets
- Portable Document Format, or PDF files
- Javascript Object Notation, or JSON files

There are many other file types not addressed here. For example, jsoup is useful for parsing HTML documents. Since we introduced how this is done in the Web scraping in Java section of [Chapter 2, Data Acquisition](#), we will not duplicate the effort here.

Handling CSV data

A common technique for separating information is to use commas or similar separators. Knowing how to work with CSV data allows us to utilize this type of data in our analysis efforts. When we deal with CSV data there are several issues including escaped data and embedded commas.

We will examine a few basic techniques for processing comma-separated data. Due to the row-column structure of CSV data, these techniques will read data from a file and place the data in a two-dimensional array. First, we will use a combination of the `Scanner` class to read in tokens and the `String` class `split` method to separate the data and store it in the array. Next, we will explore using the third-party library, OpenCSV, which offers a more efficient technique.

However, the first approach may only be appropriate for quick and dirty processing of data. We will discuss each of these techniques since they are useful in different situations.

We will use a dataset downloaded from <https://www.data.gov/> containing U.S. demographic statistics sorted by ZIP code. This dataset can be downloaded at <https://catalog.data.gov/dataset/demographic-statistics-by-zip-code-acfc9>. For our purposes, this dataset has been stored in the file `Demographics.csv`. In this particular file, every row contains the same number of columns. However, not all data will be this clean and the solutions shown next take into account the possibility for jagged arrays.

Note

A jagged array is an array where the number of columns may be different for different rows. For example, row 2 may have 5 elements while row 3 may have 6 elements. When using jagged arrays you have to be careful with your column indexes.

First, we use the `Scanner` class to read in data from our data file. We will temporarily store the data in an `ArrayList` since we will not always know how many rows our data contains.

```
try (Scanner csvData = new Scanner(new File("Demographics.csv"))) {  
    ArrayList<String> list = new ArrayList<String>();
```

```
        while (csvData.hasNext()) {
            list.add(csvData.nextLine());
        } catch (FileNotFoundException ex) {
            // Handle exceptions
        }
    }
```

The list is converted to an array using the `toArray` method. This version of the method uses a `String` array as an argument so that the method will know what type of array to create. A two-dimension array is then created to hold the CSV data.

```
String[] tempArray = list.toArray(new String[1]);
String[][] csvArray = new String[tempArray.length][];
```

The `split` method is used to create an array of `Strings` for each row. This array is assigned to a row of the `csvArray`.

```
for(int i=0; i<tempArray.length; i++) {
    csvArray[i] = tempArray[i].split(",");
}
```

Our next technique will use a third-party library to read in and process CSV data. There are multiple options available, but we will focus on the popular OpenCSV (<http://opencsv.sourceforge.net>). This library offers several advantages over our previous technique. We can have an arbitrary number of items on each row without worrying about handling exceptions. We also do not need to worry about embedded commas or embedded carriage returns within the data tokens. The library also allows us to choose between reading the entire file at once or using an iterator to process data line-by-line.

First, we need to create an instance of the `CSVReader` class. Notice the second parameter allows us to specify the delimiter, a useful feature if we have similar file format delimited by tabs or dashes, for example. If we want to read the entire file at one time, we use the `readAll` method.

```
CSVReader dataReader = new CSVReader(new
FileReader("Demographics.csv"), ',', ',');
ArrayList<String> holdData = (ArrayList) dataReader.readAll();
```

We can then process the data as we did above, by splitting the data into a two-dimension array using `String` class methods. Alternatively, we can process the data one line at a time. In the example that follows, each token is printed out individually but the tokens can also be stored in a two-dimension array or other data structure as

appropriate.

```
CSVReader dataReader = new CSVReader(new  
FileReader("Demographics.csv"), ',', ',');  
String[] nextLine;  
while ((nextLine = dataReader.readNext()) != null) {  
for(String token : nextLine){  
    out.println(token);  
}  
}  
dataReader.close();
```

We can now clean or otherwise process the array.

Handling spreadsheets

Spreadsheets have proven to be a very popular tool for processing numeric and textual data. Due to the wealth of information that has been stored in spreadsheets over the past decades, knowing how to extract information from spreadsheets enables us to take advantage of this widely available data source. In this section, we will demonstrate how this is accomplished using the Apache POI API.

Open Office also supports a spreadsheet application. Open Office documents are stored in XML format which makes it readily accessible using XML parsing technologies. However, the Apache ODF Toolkit

(<http://incubator.apache.org/odftoolkit/>) provides a means of accessing data within a document without knowing the format of the OpenOffice document. This is currently an incubator project and is not fully mature. There are a number of other APIs that can assist in processing OpenOffice documents as detailed on the **Open Document Format (ODF)** for developers (<http://www.opendocumentformat.org/developers/>) page.

Handling Excel spreadsheets

Apache POI (<http://poi.apache.org/index.html>) is a set of APIs providing access to many Microsoft products including Excel and Word. It consists of a series of components designed to access a specific Microsoft product. An overview of these components is found at <http://poi.apache.org/overview.html>.

In this section we will demonstrate how to read a simple Excel spreadsheet using the XSSF component to access Excel 2007+ spreadsheets. The Javadocs for the Apache POI API is found at <https://poi.apache.org/apidocs/index.html>.

We will use a simple Excel spreadsheet consisting of a series of rows containing an ID along with minimum, maximum, and average values. These numbers are not intended to represent any specific type of data. The spreadsheet follows:

ID	Minimum	Maximum	Average
12345	45	89	65.55
23456	78	96	86.75
34567	56	89	67.44

45678	86	99	95.67
-------	----	----	-------

We start with a try-with-resources block to handle any `IOExceptions` that may occur:

```
try (FileInputStream file = new FileInputStream(
    new File("Sample.xlsx"))) {
    ...
}
} catch (IOException e) {
    // Handle exceptions
}
```

An instance of a `XSSFWorkbook` class is created using the spreadsheet. Since a workbook may consists of multiple spreadsheets, we select the first one using the `getSheetAt` method.

```
XSSFWorkbook workbook = new XSSFWorkbook(file);
XSSFSheet sheet = workbook.getSheetAt(0);
```

The next step is to iterate through the rows, and then each column, of the spreadsheet:

```
for(Row row : sheet) {
    for (Cell cell : row) {
        ...
    }
out.println();
```

Each cell of the spreadsheet may use a different format. We use the `getCellType` method to determine its type and then use the appropriate method to extract the data in the cell. In this example we are only working with numeric and text data.

```
switch (cell.getCellType()) {
    case Cell.CELL_TYPE_NUMERIC:
        out.print(cell.getNumericCellValue() + "\t");
        break;
    case Cell.CELL_TYPE_STRING:
        out.print(cell.getStringCellValue() + "\t");
        break;
}
```

When executed we get the following output:

ID Minimum Maximum Average

12345.0 45.0 89.0 65.55

23456.0 78.0 96.0 86.75

34567.0 56.0 89.0 67.44

45678.0 86.0 99.0 95.67

POI supports other more sophisticated classes and methods to extract data.

Handling PDF files

There are several APIs supporting the extraction of text from a PDF file. Here we will use PDFBox. The Apache PDFBox (<https://pdfbox.apache.org/>) is an open source API that allows Java programmers to work with PDF documents. In this section we will illustrate how to extract simple text from a PDF document. Javadocs for the PDFBox API is found at <https://pdfbox.apache.org/docs/2.0.1/javadocs/>.

This is a simple PDF file. It consists of several bullets:

- Line 1
- Line 2
- Line 3

This is the end of the document.

A `try` block is used to catch `IOExceptions`. The `PDDocument` class will represent the PDF document being processed. Its `load` method will load in the PDF file specified by the `File` object:

```
try {  
    PDDocument document = PDDocument.load(new File("PDF File.pdf"));  
    ...  
} catch (Exception e) {  
    // Handle exceptions  
}
```

Once loaded, the `PDFTextStripper` class `getText` method will extract the text from the file. The text is then displayed as shown here:

```
PDFTextStripper Tstripper = new PDFTextStripper();  
String documentText = Tstripper.getText(document);  
System.out.println(documentText);
```

The output of this example follows. Notice that the bullets are returned as question marks.

```
This is a simple PDF file. It consists of several bullets:  
? Line 1  
? Line 2  
? Line 3  
This is the end of the document.
```

This is a brief introduction to the use of PDFBox. It is a very powerful tool when we need to extract and otherwise manipulate PDF documents.

Handling JSON

In [Chapter 2, Data Acquisition](#) we learned that certain YouTube searches return JSON formatted results. Specifically, the `SearchResult` class holds information relating to a specific search. In that section we illustrate how to use YouTube specific techniques to extract information. In this section we will illustrate how to extract JSON information using the Jackson JSON implementation.

JSON supports three models for processing data:

- **Streaming API** - JSON data is processed token by token
- **Tree model** - The JSON data is held entirely in memory and then processed
- **Data binding** - The JSON data is transformed to a Java object

Using JSON streaming API

We will illustrate the first two approaches. The first approach is more efficient and is used when a large amount of data is processed. The second technique is convenient but the data must not be too large. The third technique is useful when it is more convenient to use specific Java classes to process data. For example, if the JSON data represent an address then a specific Java address class can be defined to hold and process the data.

There are several Java libraries that support JSON processing including:

- Flexjson (<http://flexjson.sourceforge.net/>)
- Genson (<http://owlike.github.io/genson/>)
- Google-Gson (<https://github.com/google/gson>)
- Jackson library (<https://github.com/FasterXML/jackson>)
- JSON-io (<https://github.com/jdereg/json-io>)
- JSON-lib (<http://json-lib.sourceforge.net/>)

We will use the Jackson Project (<https://github.com/FasterXML/jackson>). Documentation is found at <https://github.com/FasterXML/jackson-docs>. We will use two JSON files to demonstrate how it can be used. The first file, `Person.json`, is shown next where a single person data is stored. It consists of four fields where the last field is an array of location information.

```
{  
  "firstname": "Smith",  
  "lastname": "Peter",  
  "location": [  
    {"city": "London", "country": "UK"},  
    {"city": "Paris", "country": "France"}]  
}
```

```

    "phone":8475552222,
    "address":["100 Main Street","Corpus","Oklahoma"]
}

```

The code sequence that follows shows how to extract the values for each of the fields. Within the try-catch block a `JsonFactory` instance is created which then creates a `JsonParser` instance based on the `Person.json` file.

```

try {
    JsonFactory jsonfactory = new JsonFactory();
    JsonParser parser = jsonfactory.createParser(new
File("Person.json"));
    ...
    parser.close();
} catch (IOException ex) {
    // Handle exceptions
}

```

The `nextToken` method returns a token. However, the `JsonParser` object keeps track of the current token. In the `while` loop the `nextToken` method returns and advances the parser to the next token. The `getCurrentName` method returns the field name for the token. The `while` loop terminates when the last token is reached.

```

while (parser.nextToken() != JsonToken.END_OBJECT) {
    String token = parser.getCurrentName();
    ...
}

```

The body of the loop consists of a series of `if` statements that processes the field by its name. Since the `address` field is an array, another loop will extract each of its elements until the ending array token is reached.

```

if ("firstname".equals(token)) {
    parser.nextToken();
    String fname = parser.getText();
    out.println("firstname : " + fname);
}
if ("lastname".equals(token)) {
    parser.nextToken();
    String lname = parser.getText();
    out.println("lastname : " + lname);
}
if ("phone".equals(token)) {
    parser.nextToken();
    long phone = parser.getLongValue();
}

```

```

        out.println("phone : " + phone);
    }
    if ("address".equals(token)) {
        out.println("address :");
        parser.nextToken();
        while (parser.nextToken() != JsonToken.END_ARRAY) {
            out.println(parser.getText());
        }
    }
}

```

The output of this example follows:

```

firstname : Smith
lastname : Peter
phone : 8475552222
address :
100 Main Street
Corpus
Oklahoma

```

However, JSON objects are frequently more complex than the previous example. Here a `Persons.json` file consists of an array of three persons:

```

{
    "persons": {
        "groupname": "school",
        "person": [
            {
                "firstname": "Smith",
                "lastname": "Peter",
                "phone": 8475552222,
                "address": ["100 Main Street", "Corpus", "Oklahoma"] },
            {"firstname": "King",
                "lastname": "Sarah",
                "phone": 8475551111,
                "address": ["200 Main Street", "Corpus", "Oklahoma"] },
            {"firstname": "Frost",
                "lastname": "Nathan",
                "phone": 8475553333,
                "address": ["300 Main Street", "Corpus", "Oklahoma"] }
        ]
    }
}

```

To process this file, we use a similar set of code as shown previously. We create the parser and then enter a loop as before:

```

try {
    JsonFactory jsonfactory = new JsonFactory();
    JsonParser parser = jsonfactory.createParser(new
File("Person.json"));
    while (parser.nextToken() != JsonToken.END_OBJECT) {
        String token = parser.getCurrentName();
        ...
    }
    parser.close();
} catch (IOException ex) {
    // Handle exceptions
}

```

However, we need to find the `persons` field and then extract each of its elements. The `groupname` field is extracted and displayed as shown here:

```

if ("persons".equals(token)) {
    JsonToken jsonToken = parser.nextToken();
    jsonToken = parser.nextToken();
    token = parser.getCurrentName();
    if ("groupname".equals(token)) {
        parser.nextToken();
        String groupname = parser.getText();
        out.println("Group : " + groupname);
        ...
    }
}

```

Next, we find the `person` field and call a `parsePerson` method to better organize the code:

```

parser.nextToken();
token = parser.getCurrentName();
if ("person".equals(token)) {
    out.println("Found person");
    parsePerson(parser);
}

```

The `parsePerson` method follows which is very similar to the process used in the first example.

```

public void parsePerson(JsonParser parser) throws IOException {
    while (parser.nextToken() != JsonToken.END_ARRAY) {
        String token = parser.getCurrentName();
        if ("firstname".equals(token)) {
            parser.nextToken();
            String fname = parser.getText();

```

```

        out.println("firstname : " + fname);
    }
    if ("lastname".equals(token)) {
        parser.nextToken();
        String lname = parser.getText();
        out.println("lastname : " + lname);
    }
    if ("phone".equals(token)) {
        parser.nextToken();
        long phone = parser.getLongValue();
        out.println("phone : " + phone);
    }
    if ("address".equals(token)) {
        out.println("address :");
        parser.nextToken();
        while (parser.nextToken() != JsonToken.END_ARRAY) {
            out.println(parser.getText());
        }
    }
}
}
}

```

The output follows:

```

Group : school
Found person
firstname : Smith
lastname : Peter
phone : 8475552222
address :
100 Main Street
Corpus
Oklahoma
firstname : King
lastname : Sarah
phone : 8475551111
address :
200 Main Street
Corpus
Oklahoma
firstname : Frost
lastname : Nathan
phone : 8475553333
address :
300 Main Street
Corpus
Oklahoma

```

Using the JSON tree API

The second approach is to use the tree model. An `ObjectMapper` instance is used to create a `JsonNode` instance using the `Persons.json` file. The `fieldNames` method returns `Iterator` allowing us to process each element of the file.

```
try {
    ObjectMapper mapper = new ObjectMapper();
    JsonNode node = mapper.readTree(new File("Persons.json"));
    Iterator<String> fieldNames = node.fieldNames();
    while (fieldNames.hasNext()) {
        ...
        fieldNames.next();
    }
} catch (IOException ex) {
    // Handle exceptions
}
```

Since the JSON file contains a `persons` field, we will obtain a `JsonNode` instance representing the field and then iterate over each of its elements.

```
JsonNode personsNode = node.get("persons");
Iterator<JsonNode> elements = personsNode.iterator();
while (elements.hasNext()) {
    ...
}
```

Each element is processed one at a time. If the element type is a string, we assume that this is the `groupname` field.

```
JsonNode element = elements.next();
JsonNodeType nodeType = element.getNodeType();

if (nodeType == JsonNodeType.STRING) {
    out.println("Group: " + element.textValue());
}
```

If the element is an array, we assume it contains a series of persons where each person is processed by the `parsePerson` method:

```
if (nodeType == JsonNodeType.ARRAY) {
    Iterator<JsonNode> fields = element.iterator();
    while (fields.hasNext()) {
        parsePerson(fields.next());
    }
}
```

The `parsePerson` method is shown next:

```
public void parsePerson(JsonNode node) {  
    Iterator<JsonNode> fields = node.iterator();  
    while(fields.hasNext()) {  
        JsonNode subNode = fields.next();  
        out.println(subNode.asText());  
    }  
}
```

The output follows:

```
Group: school  
Smith  
Peter  
8475552222  
King  
Sarah  
8475551111  
Frost  
Nathan  
8475553333
```

There is much more to JSON than we are able to illustrate here. However, this should give you an idea of how this type of data can be handled.

The nitty gritty of cleaning text

Strings are used to support text processing so using a good string library is important. Unfortunately, the `java.lang.String` class has some limitations. To address these limitations, you can either implement your own special string functions as needed or you can use a third-party library.

Creating your own library can be useful, but you will basically be reinventing the wheel. It may be faster to write a simple code sequence to implement some functionality, but to do things right, you will need to test them. Third-party libraries have already been tested and have been used on hundreds of projects. They provide a more efficient way of processing text.

There are several text processing APIs in addition to those found in Java. We will demonstrate two of these:

- **Apache Commons:** <https://commons.apache.org/>
- **Guava:** <https://github.com/google/guava>

Java provides many supports for cleaning text data, including methods in the `String` class. These methods are ideal for simple text cleaning and small amounts of data but can also be efficient with larger, complex datasets. We will demonstrate several `String` class methods in a moment. Some of the most helpful `String` class methods are summarized in the following table:

Method Name	Return Type	Description
trim	String	Removes leading and trailing blank spaces
toUpperCase/toLowerCase	String	Changes the casing of the entire string
replaceAll	String	Replaces all occurrences of a character sequence within the string
contains	boolean	Determines whether a given character sequence exists within the string
compareTo compareToIgnoreCase	int	Compares two strings lexicographically and returns an integer representing their relationship
matches	boolean	Determines whether the string matches a given regular expression

join	String	Combines two or more strings with a specified delimiter
split	String[]	Separates elements of a given string using a specified delimiter

Many text operations are simplified by the use of regular expressions. Regular expressions use standardized syntax to represent patterns in text, which can be used to locate and manipulate text matching the pattern.

A regular expression is simply a string itself. For example, the string `Hello, my name is Sally` can be used as a regular expression to find those exact words within a given text. This is very specific and not broadly applicable, but we can use a different regular expression to make our code more effective. `Hello, my name is \w` will match any text that starts with `Hello, my name is` and ends with a word character.

We will use several examples of more complex regular expressions, and some of the more useful syntax options are summarized in the following table. Note each must be double-escaped when used in a Java application.

Option	Description
\d	Any digit: <i>0-9</i>
\D	Any non-digit
\s	Any whitespace character
\S	Any non-whitespace character
\w	Any word character (including digits): <i>A-Z, a-z, and 0-9</i>
\W	Any non-word character

The size and source of text data varies wildly from application to application but the methods used to transform the data remain the same. You may actually need to read data from a file, but for simplicity's sake, we will be using a string containing the beginning sentences of Herman Melville's Moby Dick for several examples within this chapter. Unless otherwise specified, the text will assumed to be as shown next:

```
String dirtyText = "Call me Ishmael. Some years ago- never mind how";  
dirtyText += " long precisely - having little or no money in my  
purse,";  
dirtyText += " and nothing particular to interest me on shore, I  
thought";  
dirtyText += " I would sail about a little and see the watery part of  
the world.";
```

Using Java tokenizers to extract words

Often it is most efficient to analyze text data as tokens. There are multiple tokenizers available in the core Java libraries as well as third-party tokenizers. We will demonstrate various tokenizers throughout this chapter. The ideal tokenizer will depend upon the limitations and requirements of an individual application.

Java core tokenizers

`StringTokenizer` was the first and most basic tokenizer and has been available since Java 1. It is not recommended for use in new development as the `String` class's `split` method is considered more efficient. While it does provide a speed advantage for files with narrowly defined and set delimiters, it is less flexible than other tokenizer options. The following is a simple implementation of the `StringTokenizer` class that splits a string on spaces:

```
StringTokenizer tokenizer = new StringTokenizer(dirtyText, " ");
while(tokenizer.hasMoreTokens()) {
    out.print(tokenizer.nextToken() + " ");
}
```

When we set the `dirtyText` variable to hold our text from Moby Dick, shown previously, we get the following truncated output:

```
Call me Ishmael. Some years ago- never mind how long precisely...
```

`StreamTokenizer` is another core Java tokenizer. `StreamTokenizer` grants more information about the tokens retrieved, and allows the user to specify data types to parse, but is considered more difficult to use than `StringTokenizer` or the `split` method. The `String` class `split` method is the simplest way to split strings up based on a delimiter, but it does not provide a way to parse the split strings and you can only specify one delimiter for the entire string. For these reasons, it is not a true tokenizer, but it can be useful for data cleaning.

The `Scanner` class is designed to allow you to parse strings into different data types. We used it previously in the *Handling CSV data* section and we will address it again in the *Removing stop words* section.

Third-party tokenizers and libraries

Apache Commons consists of sets of open source Java classes and methods. These provide reusable code that complements the standard Java APIs. One popular class included in the Commons is `StrTokenizer`. This class provides more advanced support than the standard `StringTokenizer` class, specifically more control and flexibility. The following is a simple implementation of the `StrTokenizer`:

```
StrTokenizer tokenizer = new StrTokenizer(text);
while (tokenizer.hasNext()) {
    out.print(tokenizer.nextToken() + " ");
}
```

This operates in a similar fashion to `StringTokenizer` and by default parses tokens on spaces. The constructor can specify the delimiter as well as how to handle double quotes contained in data.

When we use the string from Moby Dick, shown previously, the first tokenizer implementation produces the following truncated output:

```
Call me Ishmael. Some years ago- never mind how long precisely - having
little or no money in my purse...
```

We can modify our constructor as follows:

```
StrTokenizer tokenizer = new StrTokenizer(text, ",");
```

The output for this implementation is:

```
Call me Ishmael. Some years ago- never mind how long precisely - having
little or no money in my purse
and nothing particular to interest me on shore
I thought I would sail about a little and see the watery part of the
world.
```

Notice how each line is split where commas existed in the original text. This delimiter can be a simple char, as we have shown, or a more complex `StrMatcher` object.

Google Guava is an open source set of utility Java classes and methods. The primary goal of Guava, as with many APIs, is to relieve the burden of writing basic Java utilities so developers can focus on business processes. We are going to talk about two main tools in Guava in this chapter: the `Joiner` class and the `Splitter` class. Tokenization is accomplished in Guava using its `Splitter` class's `split` method. The following is a simple example:

```
Splitter simpleSplit =
Splitter.on(',').omitEmptyStrings().trimResults();
Iterable<String> words = simpleSplit.split(dirtyText);
for(String token: words) {
    out.print(token);
}
```

This splits each token on commas and produces output like our last example. We can modify the parameter of the `on` method to split on the character of our choosing. Notice the method chaining which allows us to omit empty strings and trim leading and trailing spaces. For these reasons, and other advanced capabilities, Google Guava is considered by some to be the best tokenizer available for Java.

LingPipe is a linguistic toolkit available for language processing in Java. It provides more specialized support for text splitting with its `TokenizerFactory` interface. We implement a LingPipe `IndoEuropeanTokenizerFactory` tokenizer in the *Simple text cleaning* section.

Transforming data into a usable form

Data often needs to be cleaned once it has been acquired. Datasets are often inconsistent, are missing in information, and contain extraneous information. In this section, we will examine some simple ways to transform text data to make it more useful and easier to analyse.

Simple text cleaning

We will use the string shown before from Moby Dick to demonstrate some of the basic `String` class methods. Notice the use of the `toLowerCase` and `trim` methods. Datasets often have non-standard casing and extra leading or trailing spaces. These methods ensure uniformity of our dataset. We also use the `replaceAll` method twice. In the first instance, we use a regular expression to replace all numbers and anything that is not a word or whitespace character with a single space. The second instance replaces all back-to-back whitespace characters with a single space:

```
out.println(dirtyText);
dirtyText = dirtyText.toLowerCase().replaceAll("[\\d[^\\w\\s]]+", " ");
dirtyText = dirtyText.trim();
while(dirtyText.contains(" ")){
    dirtyText = dirtyText.replaceAll("  ", " ");
}
out.println(dirtyText);
```

When executed, the code produces the following output, truncated:

```
Call me Ishmael. Some years ago- never mind how long precisely -
call me ishmael some years ago never mind how long precisely
```

Our next example produces the same result but approaches the problem with regular expressions. In this case, we replace all of the numbers and other special characters first. Then we use method chaining to standardize our casing, remove leading and trailing spaces, and split our words into a `String` array. The `split` method allows you to break apart text on a given delimiter. In this case, we chose to use the regular expression `\\W`, which represents anything that is not a word character:

```
out.println(dirtyText);
dirtyText = dirtyText.replaceAll("[\\d[^\\w\\s]]+", " ");
String[] cleanText = dirtyText.toLowerCase().trim().split("[\\W]+");
for(String clean : cleanText) {
```

```
    out.print(clean + " ");
}
```

This code produces the same output as shown previously.

Although arrays are useful for many applications, it is often important to recombine text after cleaning. In the next example, we employ the `join` method to combine our words once we have cleaned them. We use the same chained methods as shown previously to clean and split our text. The `join` method joins every word in the array `words` and inserts a space between each word:

```
out.println(dirtyText);
String[] words = dirtyText.toLowerCase().trim().split("\\w+");
String cleanText = String.join(" ", words);
out.println(cleanText);
```

Again, this code produces the same output as shown previously. An alternate version of the `join` method is available using Google Guava. Here is a simple implementation of the same process we used before, but using the Guava `Joiner` class:

```
out.println(dirtyText);
String[] words = dirtyText.toLowerCase().trim().split("\\w+");
String cleanText = Joiner.on(" ").skipNulls().join(words);
out.println(cleanText);
```

This version provides additional options, including skipping nulls, as shown before. The output remains the same.

Removing stop words

Text analysis sometimes requires the omission of common, non-specific words such as *the*, *and*, or *but*. These words are known as stop words and there are several tools available for removing them from text. There are various ways to store a list of stop words, but for the following examples, we will assume they are contained in a file. To begin, we create a new `Scanner` object to read in our stop words. Then we take the text we wish to transform and store it in an `ArrayList` using the `Arrays` class's `asList` method. We will assume here the text has already been cleaned and normalized. It is essential to consider casing when using `String` class methods—*and* is not the same as *AND* or *And*, although all three may be stop words you wish to eliminate:

```

Scanner readStop = new Scanner(new File("C://stopwords.txt"));
ArrayList<String> words = new ArrayList<String>
(Arrays.asList((dirtyText));
out.println("Original clean text: " + words.toString());

```

We also create a new `ArrayList` to hold a list of stop words actually found in our text. This will allow us to use the `ArrayList` class `removeAll` method shortly. Next, we use our `Scanner` to read through our file of stop words. Notice how we also call the `toLowerCase` and `trim` methods against each stop word. This is to ensure that our stop words match the formatting in our text. In this example, we employ the `contains` method to determine whether our text contains the given stop word. If so, we add it to our `foundWords` `ArrayList`. Once we have processed all the stop words, we call `removeAll` to remove them from our text:

```

ArrayList<String> foundWords = new ArrayList();
while(readStop.hasNextLine()){
    String stopWord = readStop.nextLine().toLowerCase();
    if(words.contains(stopWord)) {
        foundWords.add(stopWord);
    }
}
words.removeAll(foundWords);
out.println("Text without stop words: " + words.toString());

```

The output will depend upon the words designated as stop words. If your stop words file contains different words than used in this example, your output will differ slightly. Our output follows:

```

Original clean text: [call, me, ishmael, some, years, ago, never, mind,
how, long, precisely, having, little, or, no, money, in, my, purse,
and, nothing, particular, to, interest, me, on, shore, i, thought, i,
would, sail, about, a, little, and, see, the, watery, part, of, the,
world]
Text without stop words: [call, ishmael, years, ago, never, mind, how,
long, precisely]

```

There is also support outside of the standard Java libraries for removing stop words. We are going to look at one example, using LingPipe. In this example, we start by ensuring that our text is normalized in lowercase and trimmed. Then we create a new instance of the `TokenizerFactory` class. We set our factory to use default English stop words and then tokenize the text. Notice that the `tokenizer` method uses a `char` array, so we call `toCharArray` against our text. The second parameter specifies where to begin searching within the text, and the last parameter specifies where to

end:

```
text = text.toLowerCase().trim();
TokenizerFactory fact = IndoEuropeanTokenizerFactory.INSTANCE;
fact = new EnglishStopTokenizerFactory(fact);
Tokenizer tok = fact.tokenizer(text.toCharArray(), 0, text.length());
for(String word : tok){
    out.print(word + " ");
}
```

The output follows:

Call me Ishmael. Some years ago- never mind how long precisely - having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world.

call me ishmael . years ago - never mind how long precisely - having little money my purse , nothing particular interest me shore , i thought i sail little see watery part world .

Notice the differences between our previous examples. First of all, we did not clean the text as thoroughly and allowed special characters, such as the hyphen, to remain in the text. Secondly, the LingPipe list of stop words differs from the file we used in the previous example. Some words are removed, but LingPipe was less restrictive and allowed more words to remain in the text. The type and number of stop words you use will depend upon your particular application.

Finding words in text

The standard Java libraries offer support for searching through text for specific tokens. In previous examples, we have demonstrated the `matches` method and regular expressions, which can be useful when searching text. In this example, however, we will demonstrate a simple technique using the `contains` method and the `equals` method to locate a particular string. First, we normalize our text and the word we are searching for to ensure we can find a match. We also create an integer variable to hold the number of times the word is found:

```
dirtyText = dirtyText.toLowerCase().trim();
toFind = toFind.toLowerCase().trim();
int count = 0;
```

Next, we call the `contains` method to determine whether the word exists in our text. If it does, we split the text into an array and then loop through, using the `equals` method to compare each word. If we encounter the word, we increment our counter by one. Finally, we display the output to show how many times our word was encountered:

```
if(dirtyText.contains(toFind)) {
    String[] words = dirtyText.split(" ");
    for(String word : words) {
        if(word.equals(toFind)) {
            count++;
        }
    }
}
out.println("Found " + toFind + " " + count + " times in the text.");
}
```

In this example, we set `toFind` to the letter `i`. This produced the following output:

```
Found i 2 times in the text.
```

We also have the option to use the `Scanner` class to search through an entire file. One helpful method is the `findWithinHorizon` method. This uses a `Scanner` to parse the text up to a given horizon specification. If zero is used for the second parameter, as shown next, the entire `Scanner` will be searched by default:

```
dirtyText = dirtyText.toLowerCase().trim();
toFind = toFind.toLowerCase().trim();
Scanner textLine = new Scanner(dirtyText);
```

```
out.println("Found " + textLine.findWithinHorizon(toFind, 10));
```

This technique can be more efficient for locating a particular string, but it does make it more difficult to determine where, and how many times, the string was found.

It can also be more efficient to search an entire file using a `BufferedReader`. We specify the file to search and use a try-catch block to catch any IO exceptions. We create a new `BufferedReader` object from our path and process our file as long as the next line is not empty:

```
String path = "C:// MobyDick.txt";
try {
    String textLine = "";
    toFind = toFind.toLowerCase().trim();
    BufferedReader textToClean = new BufferedReader(
        new FileReader(path));
    while((textLine = textToClean.readLine()) != null) {
        line++;
        if(textLine.toLowerCase().trim().contains(toFind)) {
            out.println("Found " + toFind + " in " + textLine);
        }
    }
    textToClean.close();
} catch (IOException ex) {
    // Handle exceptions
}
```

We again test our data by searching for the word `i` in the first sentences of Moby Dick. The truncated output follows:

```
Found i in Call me Ishmael...
```

Finding and replacing text

We often not only want to find text but also replace it with something else. We begin our next example much like we did the previous examples, by specifying our text, our text to locate, and invoking the `contains` method. If we find the text, we call the `replaceAll` method to modify our string:

```
text = text.toLowerCase().trim();
toFind = toFind.toLowerCase().trim();
out.println(text);

if(text.contains(toFind)) {
    text = text.replaceAll(toFind, replaceWith);
```

```
    out.println(text);
}
```

To test this code, we set `toFind` to the word `I` and `replaceWith` to `Ishmael`. Our output follows:

```
call me ishmael. some years ago- never mind how long precisely - having
little or no money in my purse, and nothing particular to interest me
on shore, i thought i would sail about a little and see the watery part
of the world.
```

```
call me ishmael. some years ago- never mind how long precisely - having
little or no money in my purse, and nothing particular to interest me
on shore, Ishmael thought Ishmael would sail about a little and see the
watery part of the world.
```

Apache Commons also provides a `replace` method with several variations in the `StringUtils` class. This class provides much of the same functionality as the `String` class, but with more flexibility and options. In the following example, we use our string from Moby Dick and replace all instances of the word `me` with `x` to demonstrate the `replace` method:

```
out.println(text);
out.println(StringUtils.replace(text, "me", "X"));
```

The truncated output follows:

```
Call me Ishmael. Some years ago- never mind how long precisely -
Call X Ishmael. SoX years ago- never mind how long precisely -
```

Notice how every instance of `me` has been replaced, even those instances contained within other words, such as `some`. This can be avoided by adding spaces around `me`, although this will ignore any instances where `me` is at the end of the sentence, like `me`. We will examine a better alternative using Google Guava in a moment.

The `StringUtils` class also provides a `replacePattern` method that allows you to search for and replace text based upon a regular expression. In the following example, we replace all non-word characters, such as hyphens and commas, with a single space:

```
out.println(text);
text = StringUtils.replacePattern(text, "\\W\\s", " ");
out.println(text);
```

This will produce the following truncated output:

```
Call me Ishmael. Some years ago- never mind how long precisely -  
Call me Ishmael Some years ago never mind how long precisely
```

Google Guava provides additional support for matching and modify text data using the `CharMatcher` class. `CharMatcher` not only allows you to find data matching a particular char pattern, but also provides options as to how to handle the data. This includes allowing you to retain the data, replace the data, and trim whitespaces from within a particular string.

In this example, we are going to use the `replace` method to simply replace all instances of the word `me` with a single space. This will produce series of empty spaces within our text. We will then collapse the extra whitespace using the `trimAndCollapseFrom` method and print our string again:

```
text = text.replace("me", " ");  
out.println("With double spaces: " + text);  
String spaced = CharMatcher.WHITESPACE.trimAndCollapseFrom(text, ' ');  
out.println("With double spaces removed: " + spaced);
```

Our output is truncated as follows:

```
With double spaces: Call Ishmael. So years ago- ...  
With double spaces removed: Call Ishmael. So years ago- ...
```

Data imputation

Data imputation refers to the process of identifying and replacing missing data in a given dataset. In almost any substantial case of data analysis, missing data will be an issue, and it needs to be addressed before data can be properly analysed. Trying to process data that is missing information is a lot like trying to understand a conversation where every once in while a word is dropped. Sometimes we can understand what is intended. In other situations, we may be completely lost as to what is trying to be conveyed.

Among statistical analysts, there exist differences of opinion as to how missing data should be handled but the most common approaches involve replacing missing data with a reasonable estimate or with an empty or null value.

To prevent skewing and misalignment of data, many statisticians advocate for replacing missing data with values representative of the average or expected value for that dataset. The methodology for determining a representative value and assigning it to a location within the data will vary depending upon the data and we cannot illustrate every example in this chapter. However, for example, if a dataset contained a list of temperatures across a range of dates, and one date was missing a temperature, that date can be assigned a temperature that was the average of the temperatures within the dataset.

We will examine a rather trivial example to demonstrate the issues surrounding data imputation. Let's assume the variable `tempList` contains average temperature data for each month of one year. Then we perform a simple calculation of the average and print out our results:

```
double[] tempList = {50, 56, 65, 70, 74, 80, 82, 90, 83, 78, 64, 52};  
double sum = 0;  
for(double d : tempList){  
    sum += d;  
}  
out.printf("The average temperature is %1$.2f", sum/12);
```

Notice that for the numbers used in this execution, the output is as follows:

The average temperature is 70.33

Next we will mimic missing data by changing the first element of our array to zero

before we calculate our `sum`:

```
double sum = 0;
tempList[0] = 0;
for(double d : tempList) {
    sum += d;
}
out.printf("The average temperature is %1$.2f", sum/12);
```

This will change the average temperature displayed in our output:

The average temperature is 66.17

Notice that while this change may seem rather minor, it is statistically significant. Depending upon the variation within a given dataset and how far the average is from zero or some other substituted value, the results of a statistical analysis may be significantly skewed. This does not mean zero should never be used as a substitute for null or otherwise invalid values, but other alternatives should be considered.

One alternative approach can be to calculate the average of the values in the array, excluding zeros or nulls, and then substitute the average in each position with missing data. It is important to consider the type of data and purpose of data analysis when making these decisions. For example, in the preceding example, will zero always be an invalid average temperature? Perhaps not if the temperatures were averages for Antarctica.

When it is essential to handle null data, Java's `Optional` class provides helpful solutions. Consider the following example, where we have a list of names stored as an array. We have set one value to `null` for the purposes of demonstrating these methods:

```
String useName = "";
String[] nameList =
    {"Amy", "Bob", "Sally", "Sue", "Don", "Rick", null, "Betsy"};
Optional<String> tempName;
for(String name : nameList) {
    tempName = Optional.ofNullable(name);
    useName = tempName.orElse("DEFAULT");
    out.println("Name to use = " + useName);
}
```

We first created a variable called `useName` to hold the name we will actually print

out. We also created an instance of the `Optional` class called `tempName`. We will use this to test whether a value in the array is null or not. We then loop through our array and create and call the `Optional` class `ofNullable` method. This method tests whether a particular value is null or not. On the next line, we call the `orElse` method to either assign a value from the array to `useName` or, if the element is null, assign `DEFAULT`. Our output follows:

```
Name to use = Amy
Name to use = Bob
Name to use = Sally
Name to use = Sue
Name to use = Don
Name to use = Rick
Name to use = DEFAULT
Name to use = Betsy
```

The `Optional` class contains several other methods useful for handling potential null data. Although there are other ways to handle such instances, this Java 8 addition provides simpler and more elegant solutions to a common data analysis problem.

Subsetting data

It is not always practical or desirable to work with an entire set of data. In these cases, we may want to retrieve a subset of data to either work with or remove entirely from the dataset. There are a few ways of doing this supported by the standard Java libraries. First, we will use the `subSet` method of the `SortedSet` interface. We will begin by storing a list of numbers in a `TreeSet`. We then create a new `TreeSet` object to hold the subset retrieved from the list. Next, we print out our original list:

```
Integer[] nums = {12, 46, 52, 34, 87, 123, 14, 44};  
TreeSet<Integer> fullNumsList = new TreeSet<Integer>(new  
ArrayList<>(Arrays.asList(nums)));  
SortedSet<Integer> partNumsList;  
out.println("Original List: " + fullNumsList.toString()  
+ " " + fullNumsList.last());
```

The `subSet` method takes two parameters, which specify the range of integers within the data we want to retrieve. The first parameter is included in the results while the second is exclusive. In our example that follows, we want to retrieve a subset of all numbers between the first number in our array 12 and 46:

```
partNumsList = fullNumsList.subSet(fullNumsList.first(), 46);  
out.println("SubSet of List: " + partNumsList.toString()  
+ " " + partNumsList.size());
```

Our output follows:

```
Original List: [12, 14, 34, 44, 46, 52, 87, 123]  
SubSet of List: [12, 14, 34, 44]
```

Another option is to use the `stream` method in conjunction with the `skip` method. The `stream` method returns a Java 8 Stream instance which iterates over the set. We will use the same `numsList` as in the previous example, but this time we will specify how many elements to skip with the `skip` method. We will also use the `collect` method to create a new `Set` to hold the new elements:

```
out.println("Original List: " + numsList.toString());  
Set<Integer> fullNumsList = new TreeSet<Integer>(numsList);  
Set<Integer> partNumsList = fullNumsList  
    .stream()  
    .skip(5)
```

```
.collect(toCollection(TreeSet::new));
out.println("SubSet of List: " + partNumsList.toString());
```

When we print out the new subset, we get the following output where the first five elements of the sorted set are skipped. Because it is a `SortedSet`, we will actually be omitting the five lowest numbers:

```
Original List: [12, 46, 52, 34, 87, 123, 14, 44]
SubSet of List: [52, 87, 123]
```

At times, data will begin with blank lines or header lines that we wish to remove from our dataset to be analysed. In our final example, we will read data from a file and remove all blank lines. We use a `BufferedReader` to read our data and employ a lambda expression to test for a blank line. If the line is not blank, we print the line to the screen:

```
try (BufferedReader br = new BufferedReader(new
FileReader("C:\\text.txt"))) {
    br
        .lines()
        .filter(s -> !s.equals(""))
        .forEach(s -> out.println(s));
} catch (IOException ex) {
    // Handle exceptions
}
```

Sorting text

Sometimes it is necessary to sort data during the cleaning process. The standard Java library provides several resources for accomplishing different types of sorts, with improvements added with the release of Java 8. In our first example, we will use the `Comparator` interface in conjunction with a lambda expression.

We start by declaring our `Comparator` variable `compareInts`. The first set of parenthesis after the equals sign contains the parameters to be passed to our method. Within the lambda expression, we call the `compare` method, which determines which integer is larger:

```
Comparator<Integer> compareInts = (Integer first, Integer second) ->  
    Integer.compare(first, second);
```

We can now call the `sort` method as we did previously:

```
Collections.sort(numsList, compareInts);  
out.println("Sorted integers using Lambda: " + numsList.toString());
```

Our output follows:

```
Sorted integers using Lambda: [12, 14, 34, 44, 46, 52, 87, 123]
```

We then mimic the process with our `wordsList`. Notice the use of the `compareTo` method rather than `compare`:

```
Comparator<String> compareWords = (String first, String second) ->  
    first.compareTo(second);  
Collections.sort(wordsList, compareWords);  
out.println("Sorted words using Lambda: " + wordsList.toString());
```

When this code is executed, we should see the following output:

```
Sorted words using Lambda: [boat, cat, dog, house, road, zoo]
```

In our next example, we are going to use the `Collections` class to perform basic sorting on `String` and `integer` data. For this example, `wordList` and `numsList` are both `ArrayList` and are initialized as follows:

```
List<String> wordsList
    = Stream.of("cat", "dog", "house", "boat", "road", "zoo")
        .collect(Collectors.toList());
List<Integer> numsList = Stream.of(12, 46, 52, 34, 87, 123, 14, 44)
    .collect(Collectors.toList());
```

First, we will print our original version of each list followed by a call to the `sort` method. We then display our data, sorted in ascending fashion:

```
out.println("Original Word List: " + wordsList.toString());
Collections.sort(wordsList);
out.println("Ascending Word List: " + wordsList.toString());
out.println("Original Integer List: " + numsList.toString());
Collections.sort(numsList);
out.println("Ascending Integer List: " + numsList.toString());
```

The output follows:

```
Original Word List: [cat, dog, house, boat, road, zoo]
Ascending Word List: [boat, cat, dog, house, road, zoo]
Original Integer List: [12, 46, 52, 34, 87, 123, 14, 44]
Ascending Integer List: [12, 14, 34, 44, 46, 52, 87, 123]
```

Next, we will replace the `sort` method with the `reverse` method of the `Collections` class in our integer data example. This method simply takes the elements and stores them in reverse order:

```
out.println("Original Integer List: " + numsList.toString());
Collections.reverse(numsList);
out.println("Reversed Integer List: " + numsList.toString());
```

The output displays our new `numsList`:

```
Original Integer List: [12, 46, 52, 34, 87, 123, 14, 44]
Reversed Integer List: [44, 14, 123, 87, 34, 52, 46, 12]
```

In our next example, we handle the sort using the `Comparator` interface. We will continue to use our `numsList` and assume that no sorting has occurred yet. First we create two objects that implement the `Comparator` interface. The `sort` method will use these objects to determine the desired order when comparing two elements. The expression `Integer::compare` is a Java 8 method reference. This is can be used where a lambda expression is used:

```
out.println("Original Integer List: " + numsList.toString());
Comparator<Integer> basicOrder = Integer::compare;
Comparator<Integer> descendOrder = basicOrder.reversed();
Collections.sort(numsList, descendOrder);
out.println("Descending Integer List: " + numsList.toString());
```

After we execute this code, we will see the following output:

```
Original Integer List: [12, 46, 52, 34, 87, 123, 14, 44]
Descending Integer List: [123, 87, 52, 46, 44, 34, 14, 12]
```

In our last example, we will attempt a more complex sort involving two comparisons. Let's assume there is a `Dog` class that contains two properties, `name` and `age`, along with the necessary accessor methods. We will begin by adding elements to a new `ArrayList` and then printing the names and ages of each `Dog`:

```
ArrayList<Dogs> dogs = new ArrayList<Dogs>();
dogs.add(new Dogs("Zoey", 8));
dogs.add(new Dogs("Roxie", 10));
dogs.add(new Dogs("Kylie", 7));
dogs.add(new Dogs("Shorty", 14));
dogs.add(new Dogs("Ginger", 7));
dogs.add(new Dogs("Penny", 7));
out.println("Name " + " Age");
for(Dogs d : dogs) {
    out.println(d.getName() + " " + d.getAge());
}
```

Our output should resemble:

Name	Age
Zoey	8
Roxie	10
Kylie	7
Shorty	14
Ginger	7
Penny	7

Next, we are going to use method chaining and the double colon operator to reference methods from the `Dog` class. We first call `comparing` followed by `thenComparing` to specify the order in which comparisons should occur. When we execute the code, we expect to see the `Dog` objects sorted first by `Name` and then by `Age`:

```
dogs.sort(Comparator.comparing(Dogs::getName).thenComparing(Dogs::getAge));
out.println("Name " + " Age");
for(Dogs d : dogs) {
    out.println(d.getName() + " " + d.getAge());
}
```

Our output follows:

Name	Age
Ginger	7
Kylie	7
Penny	7
Roxie	10
Shorty	14
Zoey	8

Now we will switch the order of comparison. Notice how the age of the dog takes priority over the name in this version:

```
dogs.sort(Comparator.comparing(Dogs::getAge).thenComparing(Dogs::getName));
out.println("Name " + " Age");
for(Dogs d : dogs) {
    out.println(d.getName() + " " + d.getAge());
}
```

And our output is:

Name	Age
Ginger	7
Kylie	7
Penny	7
Zoey	8
Roxie	10
Shorty	14

Data validation

Data validation is an important part of data science. Before we can analyze and manipulate data, we need to verify that the data is of the type expected. We have organized our code into simple methods designed to accomplish very basic validation tasks. The code within these methods can be adapted into existing applications.

Validating data types

Sometimes we simply need to validate whether a piece of data is of a specific type, such as integer or floating point data. We will demonstrate in the next example how to validate integer data using the `validateInt` method. This technique is easily modified for the other major data types supported in the standard Java library, including `Float` and `Double`.

We need to use a try-catch block here to catch a `NumberFormatException`. If an exception is thrown, we know our data is not a valid integer. We first pass our text to be tested to the `parseInt` method of the `Integer` class. If the text can be parsed as an integer, we simply print out the integer. If an exception is thrown, we display information to that effect:

```
public static void validateInt(String toValidate) {  
    try{  
        int validInt = Integer.parseInt(toValidate);  
        out.println(validInt + " is a valid integer");  
    }catch(NumberFormatException e){  
        out.println(toValidate + " is not a valid integer");  
    }  
}
```

We will use the following method calls to test our method:

```
validateInt("1234");  
validateInt("Ishmael");
```

The output follows:

```
1234 is a valid integer  
Ishmael is not a valid integer
```

The Apache Commons contain an `IntegerValidator` class with additional useful functionalities. In this first example, we simply duplicate the process from before, but use `IntegerValidator` methods to accomplish our goal:

```
public static String validateInt(String text) {
    IntegerValidator intValidator = IntegerValidator.getInstance();
    if(intValidator.isValid(text)) {
        return text + " is a valid integer";
    }else{
        return text + " is not a valid integer";
    }
}
```

We again use the following method calls to test our method:

```
validateInt("1234");
validateInt("Ishmael");
```

The output follows:

```
1234 is a valid integer
Ishmael is not a valid integer
```

The `IntegerValidator` class also provides methods to determine whether an integer is greater than or less than a specific value, compare the number to a range of numbers, and convert `Number` objects to `Integer` objects. Apache Commons has a number of other validator classes. We will examine a few more in the rest of this section.

Validating dates

Many times our data validation is more complex than simply determining whether a piece of data is the correct type. When we want to verify that the data is a date for example, it is insufficient to simply verify that it is made up of integers. We may need to include hyphens and slashes, or ensure that the year is in two-digit or four-digit format.

To do this, we have created another simple method called `validateDate`. The method takes two `String` parameters, one to hold the date to validate and the other to hold the acceptable date format. We create an instance of the `SimpleDateFormat` class using the format specified in the parameter. Then we call the `parse` method to convert our `String` date to a `Date` object. Just as in our previous integer example, if

the data cannot be parsed as a date, an exception is thrown and the method returns. If, however, the `String` can be parsed to a date, we simply compare the format of the test date with our acceptable format to determine whether the date is valid:

```
public static String validateDate(String theDate, String dateFormat) {  
    try {  
        SimpleDateFormat format = new SimpleDateFormat(dateFormat);  
        Date test = format.parse(theDate);  
        if(format.format(test).equals(theDate)) {  
            return theDate.toString() + " is a valid date";  
        } else {  
            return theDate.toString() + " is not a valid date";  
        }  
    } catch (ParseException e) {  
        return theDate.toString() + " is not a valid date";  
    }  
}
```

We make the following method calls to test our method:

```
String dateFormat = "MM/dd/yyyy";  
out.println(validateDate("12/12/1982",dateFormat));  
out.println(validateDate("12/12/82",dateFormat));  
out.println(validateDate("Ishmael",dateFormat));
```

The output follows:

```
12/12/1982 is a valid date  
12/12/82 is not a valid date  
Ishmael is not a valid date
```

This example highlights why it is important to consider the restrictions you place on data. Our second method call did contain a legitimate date, but it was not in the format we specified. This is good if we are looking for very specifically formatted data. But we also run the risk of leaving out useful data if we are too restrictive in our validation.

Validating e-mail addresses

It is also common to need to validate e-mail addresses. While most e-mail addresses have the @ symbol and require at least one period after the symbol, there are many variations. Consider that each of the following examples can be valid e-mail addresses:

- myemail@mail.com
- MyEmail@some.mail.com
- My.Email.123!@mail.net

One option is to use regular expressions to attempt to capture all allowable e-mail addresses. Notice that the regular expression used in the method that follows is very long and complex. This can make it easy to make mistakes, miss valid e-mail addresses, or accept invalid addresses as valid. But a carefully crafted regular expression can be a very powerful tool.

We use the `Pattern` and `Matcher` classes to compile and execute our regular expression. If the pattern of the e-mail we pass in matches the regular expression we defined, we will consider that text to be a valid e-mail address:

```
public static String validateEmail(String email) {
    String emailRegex = "^[a-zA-Z0-9.!$'*+/=?^`{|}~-" +
        "]@[\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\." +
        "[0-9]{1,3}\\]]|(([a-zA-Z\\-0-9]+\\.)+[a-zA-Z]{2,}))$";
    Pattern.compile(emailRegex);
    Matcher matcher = pattern.matcher(email);
    if(matcher.matches()){
        return email + " is a valid email address";
    }else{
        return email + " is not a valid email address";
    }
}
```

We make the following method calls to test our data:

```
out.println(validateEmail("myemail@mail.com"));
out.println(validateEmail("My.Email.123!@mail.net"));
out.println(validateEmail("myEmail"));
```

The output follows:

```
myemail@mail.com is a valid email address
My.Email.123!@mail.net is a valid email address
myEmail is not a valid email address
```

There is a standard Java library for validating e-mail addresses as well. In this example, we use the `InternetAddress` class to validate whether a given string is a valid e-mail address or not:

```
public static String validateEmailStandard(String email) {
```

```

        try{
            InternetAddress testEmail = new InternetAddress(email);
            testEmail.validate();
            return email + " is a valid email address";
        }catch(AddressException e){
            return email + " is not a valid email address";
        }
    }
}

```

When tested against the same data as in the previous example, our output is identical. However, consider the following method call:

```
out.println(validateEmailStandard("myEmail@mail"));
```

Despite not being in standard e-mail format, the output is as follows:

```
myEmail@mail is a valid email address
```

Additionally, the `validate` method by default accepts local e-mail addresses as valid. This is not always desirable, depending upon the purpose of the data.

One last option we will look at is the Apache Commons `EmailValidator` class. This class's `isValid` method examines an e-mail address and determines whether it is valid or not. Our `validateEmail` method shown previously is modified as follows to use `EmailValidator`:

```

public static String validateEmailApache(String email){
    email = email.trim();
    EmailValidator eValidator = EmailValidator.getInstance();
    if(eValidator.isValid(email)){
        return email + " is a valid email address.";
    }else{
        return email + " is not a valid email address.";
    }
}

```

Validating ZIP codes

Postal codes are generally formatted specific to their country or local requirements. For this reason, regular expressions are useful because they can accommodate any postal code required. The example that follows demonstrates how to validate a standard United States postal code, with or without the hyphen and last four digits:

```

public static void validateZip(String zip) {
    String zipRegex = "^[0-9]{5}(-[0-9]{4})?\\$";
    Pattern pattern = Pattern.compile(zipRegex);
    Matcher matcher = pattern.matcher(zip);
    if(matcher.matches()){
        out.println(zip + " is a valid zip code");
    }else{
        out.println(zip + " is not a valid zip code");
    }
}

```

We make the following method calls to test our data:

```

out.println(validateZip("12345"));
out.println(validateZip("12345-6789"));
out.println(validateZip("123"));

```

The output follows:

```

12345 is a valid zip code
12345-6789 is a valid zip code
123 is not a valid zip code

```

Validating names

Names can be especially tricky to validate because there are so many variations. There are no industry standards or technical limitations, other than what characters are available on the keyboard. For this example, we have chosen to use Unicode in our regular expression because it allows us to match any character from any language. The Unicode property `\p{L}` provides this flexibility. We also use `\s-`, to allow spaces, apostrophes, commas, and hyphens in our name fields. It is possible to perform string cleaning, as discussed earlier in this chapter, before attempting to match names. This will simplify the regular expression required:

```

public static void validateName(String name) {
    String nameRegex = "^[\\p{L}\\s-]+\\$";
    Pattern pattern = Pattern.compile(nameRegex);
    Matcher matcher = pattern.matcher(name);
    if(matcher.matches()){
        out.println(name + " is a valid name");
    }else{
        out.println(name + " is not a valid name");
    }
}

```

We make the following method calls to test our data:

```
validateName("Bobby Smith, Jr.");
validateName("Bobby Smith the 4th");
validateName("Albrecht Müller");
validateName("François Moreau");
```

The output follows:

```
Bobby Smith, Jr. is a valid name
Bobby Smith the 4th is not a valid name
Albrecht Müller is a valid name
François Moreau is a valid name
```

Notice that the comma and period in `Bobby Smith, Jr.` are acceptable, but the 4 in `4th` is not. Additionally, the special characters in `François` and `Müller` are considered valid.

Cleaning images

While image processing is a complex task, we will introduce a few techniques to clean and extract information from an image. This will provide the reader with some insight into image processing. We will also demonstrate how to extract text data from an image using **Optical Character Recognition (OCR)**.

There are several techniques used to improve the quality of an image. Many of these require tweaking of parameters to get the improvement desired. We will demonstrate how to:

- Enhance an image's contrast
- Smooth an image
- Brighten an image
- Resize an image
- Convert images to different formats

We will use OpenCV (<http://opencv.org/>), an open source project for image processing. There are several classes that we will use:

- `Mat`: This represents an n-dimensional array holding image data such as channel, grayscale, or color values
- `Imgproc`: Possesses many methods that process an image
- `Imgcodecs`: Possesses methods to read and write image files

The OpenCV Javadocs is found at <http://docs.opencv.org/java/2.4.9/>. In the examples that follow, we will use Wikipedia images since they can be freely downloaded. Specifically we will use the following images:

- **Parrot**
`image`: https://en.wikipedia.org/wiki/Grayscale#/media/File:Grayscale_8bits_p
- **Cat image**: https://en.wikipedia.org/wiki/Cat#/media/File:Kittyply_edit1.jpg

Changing the contrast of an image

Here we will demonstrate how to enhance a black-and-white image of a parrot. The `Imgcodecs` class's `imread` method reads in the image. Its second parameter specifies the type of color used by the image, which is grayscale in this case. A new `Mat` object is created for the enhanced image using the same size and color type as the original.

The actual work is performed by the `equalizeHist` method. This equalizes the histogram of the image which has the effect of normalizing the brightness and increases the contrast of the image. An image histogram is a histogram representing the tonal distribution of an image. **Tonal** is also referred to as lightness. It represents the variation in the brightness found in an image.

The last step is to write out the image.

```
Mat source = Imgcodecs.imread("GrayScaleParrot.png",
    Imgcodecs.CV_LOAD_IMAGE_GRAYSCALE);
Mat destination = new Mat(source.rows(), source.cols(), source.type());
Imgproc.equalizeHist(source, destination);
Imgcodecs.imwrite("enhancedParrot.jpg", destination);
```

The following is the original image:



The enhanced image follows:



Smoothing an image

Smoothing an image, also called **blurring**, will make the edges of an image smoother. Blurring is the process of making an image less distinct. We recognize blurred objects when we take a picture with the camera out of focus. Blurring can be used for special effects. Here, we will use it to create an image that we will then sharpen.

The following example loads an image of a cat and repeatedly applies the `blur` method to the image. In this example, the process is repeated 25 times. Increasing the number of iterations will result in more blur or smoothing.

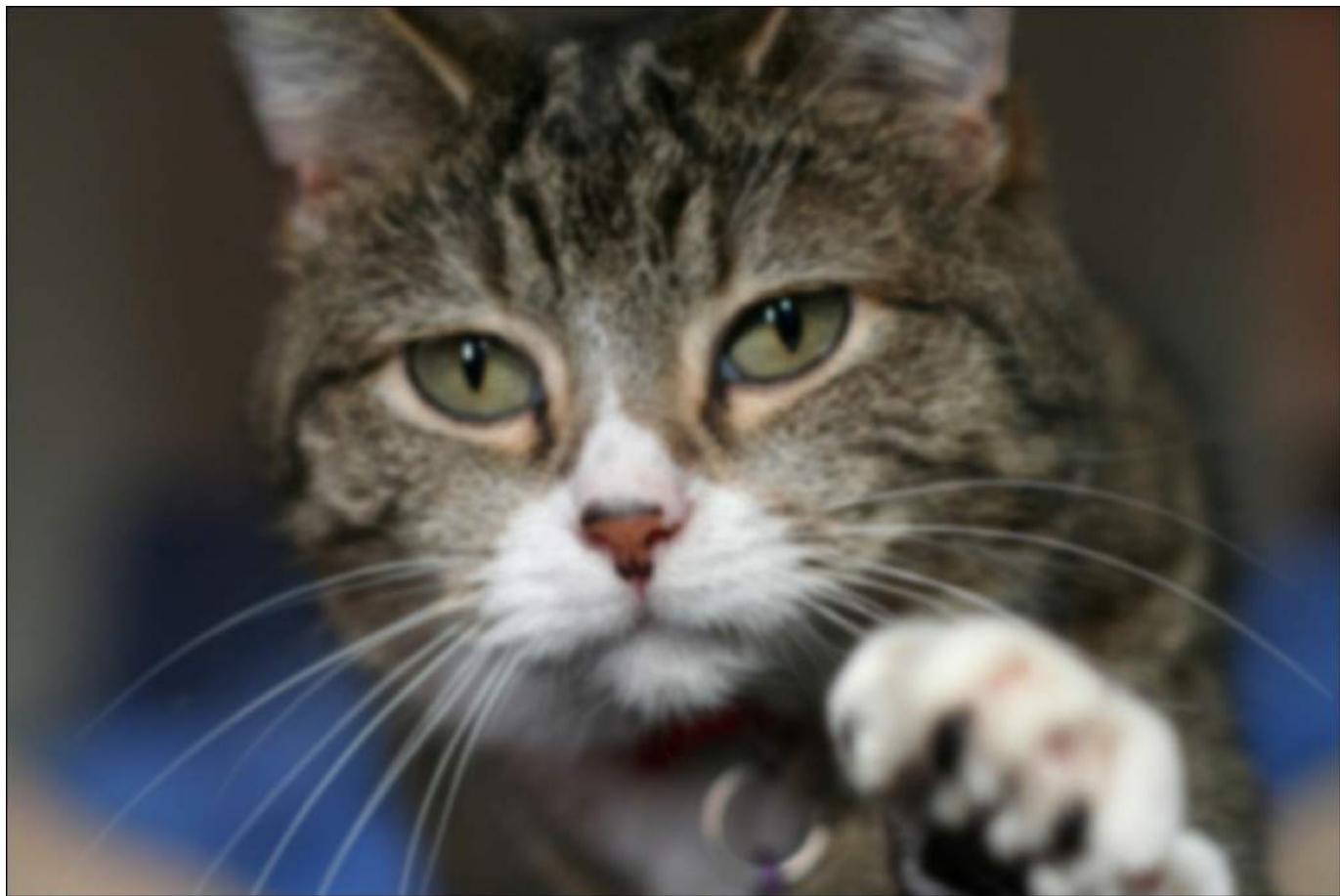
The third argument of the `blur` method is the blurring kernel size. The kernel is a matrix of pixels, 3 by 3 in this example, that is used for convolution. This is the process of multiplying each element of an image by weighted values of its neighbors. This allows neighboring values to effect an element's value:

```
Mat source = Imgcodecs.imread("cat.jpg");
Mat destination = source.clone();
for (int i = 0; i < 25; i++) {
    Mat sourceImage = destination.clone();
    Imgproc.blur(sourceImage, destination, new Size(3.0, 3.0));
}
Imgcodecs.imwrite("smoothCat.jpg", destination);
```

The following is the original image:



The enhanced image follows:



Brightening an image

The `convertTo` method provides a means of brightening an image. The original image is copied to a new image where the contrast and brightness is adjusted. The first parameter is the destination image. The second specifies that the type of image should not be changed. The third and fourth parameters control the contrast and brightness respectively. The first value is multiplied by this value while the second is added to the multiplied value:

```
Mat source = Imgcodecs.imread("cat.jpg");
Mat destination = new Mat(source.rows(), source.cols(),
    source.type());
source.convertTo(destination, -1, 1, 50);
Imgcodecs.imwrite("brighterCat.jpg", destination);
```

The enhanced image follows:



Resizing an image

Sometimes it is desirable to resize an image. The `resize` method shown next illustrates how this is done. The image is read in and a new `Mat` object is created. The `resize` method is then applied where the width and height are specified in the `Size` object parameter. The resized image is then saved:

```
Mat source = Imgcodecs.imread("cat.jpg");
Mat resizeimage = new Mat();
Imgproc.resize(source, resizeimage, new Size(250, 250));
Imgcodecs.imwrite("resizedCat.jpg", resizeimage);
```

The enhanced image follows:



Converting images to different formats

Another common operation is to convert an image that uses one format into an image that uses a different format. In OpenCV, this is easy to accomplish as shown next. The image is read in and then immediately written out. The extension of the file is used by the `imwrite` method to convert the image to the new format:

```
Mat source = Imgcodecs.imread("cat.jpg");
Imgcodecs.imwrite("convertedCat.jpg", source);
Imgcodecs.imwrite("convertedCat.jpeg", source);
Imgcodecs.imwrite("convertedCat.webp", source);
Imgcodecs.imwrite("convertedCat.png", source);
Imgcodecs.imwrite("convertedCat.tiff", source);
```

The images can now be used for specialized processing if necessary.

Summary

Many times, half the battle in data science is manipulating data so that it is clean enough to work with. In this chapter, we examined many techniques for taking real-world, messy data and transforming it into workable datasets. This process is generally known as data cleaning, wrangling, reshaping, or munging. Our focus was on core Java techniques, but we also examined third-party libraries.

Before we can clean data, we need to have a solid understanding of the format of our data. We discussed CSV data, spreadsheets, PDF, and JSON file types, as well as provided several examples of manipulating text file data. As we examined text data, we looked at multiple approaches for processing the data, including tokenizers, Scanners, and `BufferedReader`s. We showed ways to perform simple cleaning operations, remove stop words, and perform find and replace functions.

This chapter also included a discussion on data imputation and the importance of identifying and rectifying missing data situations. Missing data can cause problems during data analysis and we proposed different methods for dealing with this problem. We demonstrated how to retrieve subsets of data and sort data as well.

Finally, we discussed image cleaning and demonstrated several methods of modifying image data. This included changing contrast, smoothing, brightening, and resizing information. We concluded with a discussion on extracting text imposed on an image.

With this background, we will introduce basic statistical methods and their Java support in the next chapter.

Chapter 4. Data Visualization

The human mind is often good at seeing patterns, trends, and outliers in visual representations. The large amount of data present in many data science problems can be analyzed using visualization techniques. Visualization is appropriate for a wide range of audiences, ranging from analysts, to upper-level management, to clientele. In this chapter, we present various visualization techniques and demonstrate how they are supported in Java.

In this chapter, we will illustrate how to create different types of graph, plot, and chart. The majority of the examples use JavaFX, with a few using a free library called **GRAping Library (GRAL)**. There are several open source Java plotting libraries available. A brief comparison of several of these libraries can be found at <https://github.com/eseifert/gral/wiki/comparison>. We chose JavaFX because it is packaged as part of Java SE.

GRAL is used to illustrate plots that are not as easily created using JavaFX. GRAL is a free Java library useful for creating a variety of charts and graphs. This graphing library provides flexibility in types of plots, axis formatting, and export options. GRAL resources (<http://trac.erichseifert.de/gral/>) include example code and helpful how to sections.

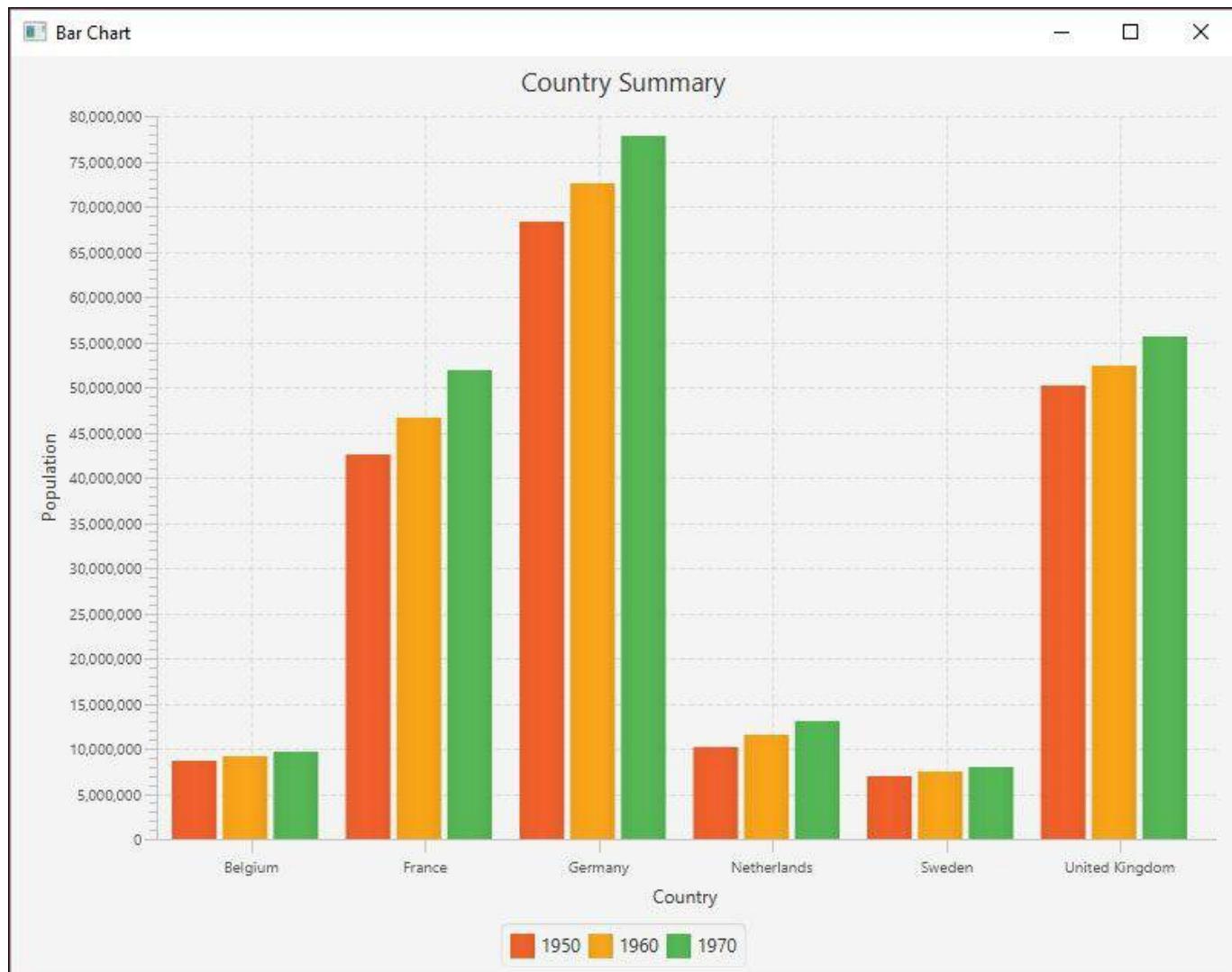
Visualization is an important step in data analysis because it allows us to conceive of large datasets in practical and meaningful ways. We can look at small datasets of values and perhaps draw conclusions from the patterns we see, but this is an overwhelming and unreliable process. Using visualization tools helps us identify potential problems or unexpected data results, as well as construct meaningful interpretations of good data.

One example of the usefulness of data visualization comes with the presence of **outliers**. Visualizing data allows us to quickly see data results significantly outside of our expectations, and we can choose how to modify the data to build a clean and usable dataset. This process allows us to see errors quickly and deal with them before they become a problem later on. Additionally, visualization allows us to easily classify information and help analysts organize their inquiries in a manner best suited to their particular dataset.

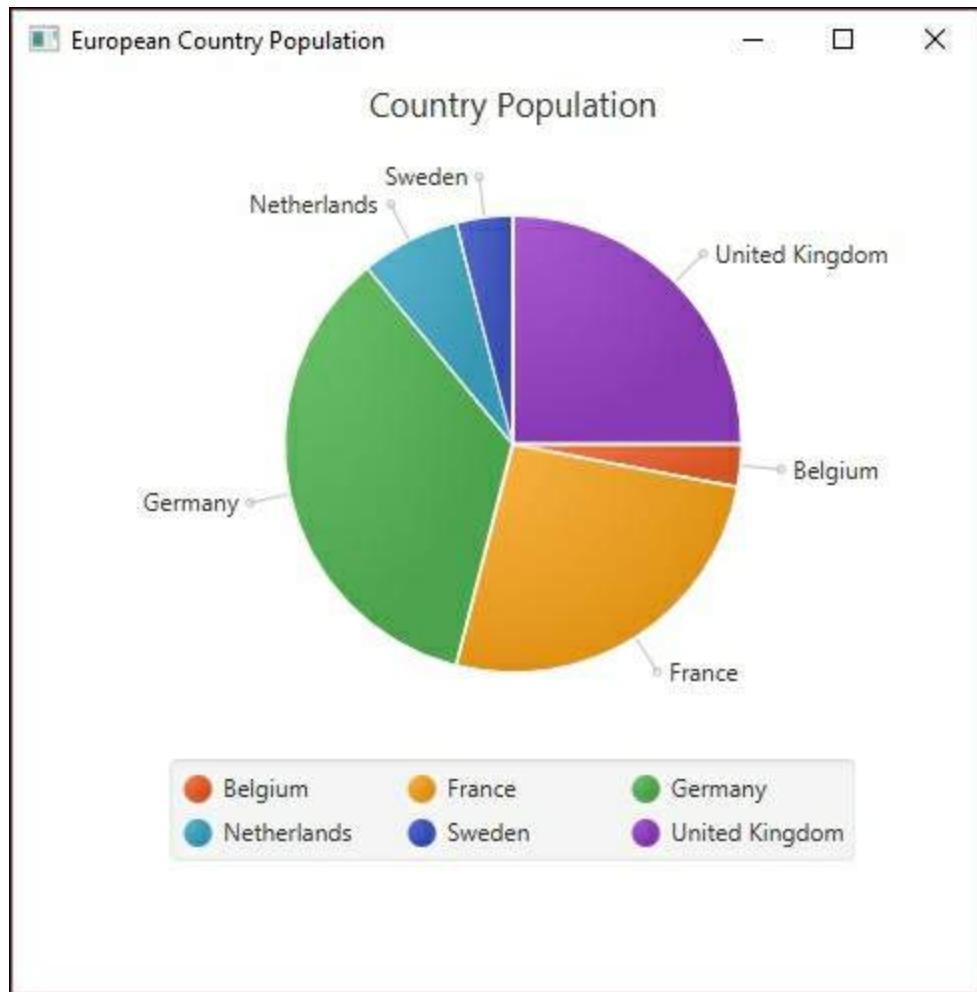
Understanding plots and graphs

There are many types of visual expression available to aid in visualization. We are going to briefly discuss the most common and useful ones, and then demonstrate several Java techniques for achieving these types of expression. The choice of graph, or other visualization tool will depend upon the dataset and application needs and constraints.

A **bar chart** is a very common technique for displaying relationships in data. In this type of graph, data is represented in either vertical or horizontal bars placed along an *X* and *Y* axis. The data is scaled so the values represented by each bar can be compared to one another. The following is a simple example of a bar chart we will create in the *Using country as the category* section:



A **pie chart** is most useful when you want to demonstrate a value in relation to a larger set. Think of this as a way to visualize how large the piece of pie is in relation to the entire pie. The following is a simple example of a pie chart showing the distribution of population for selected European countries:

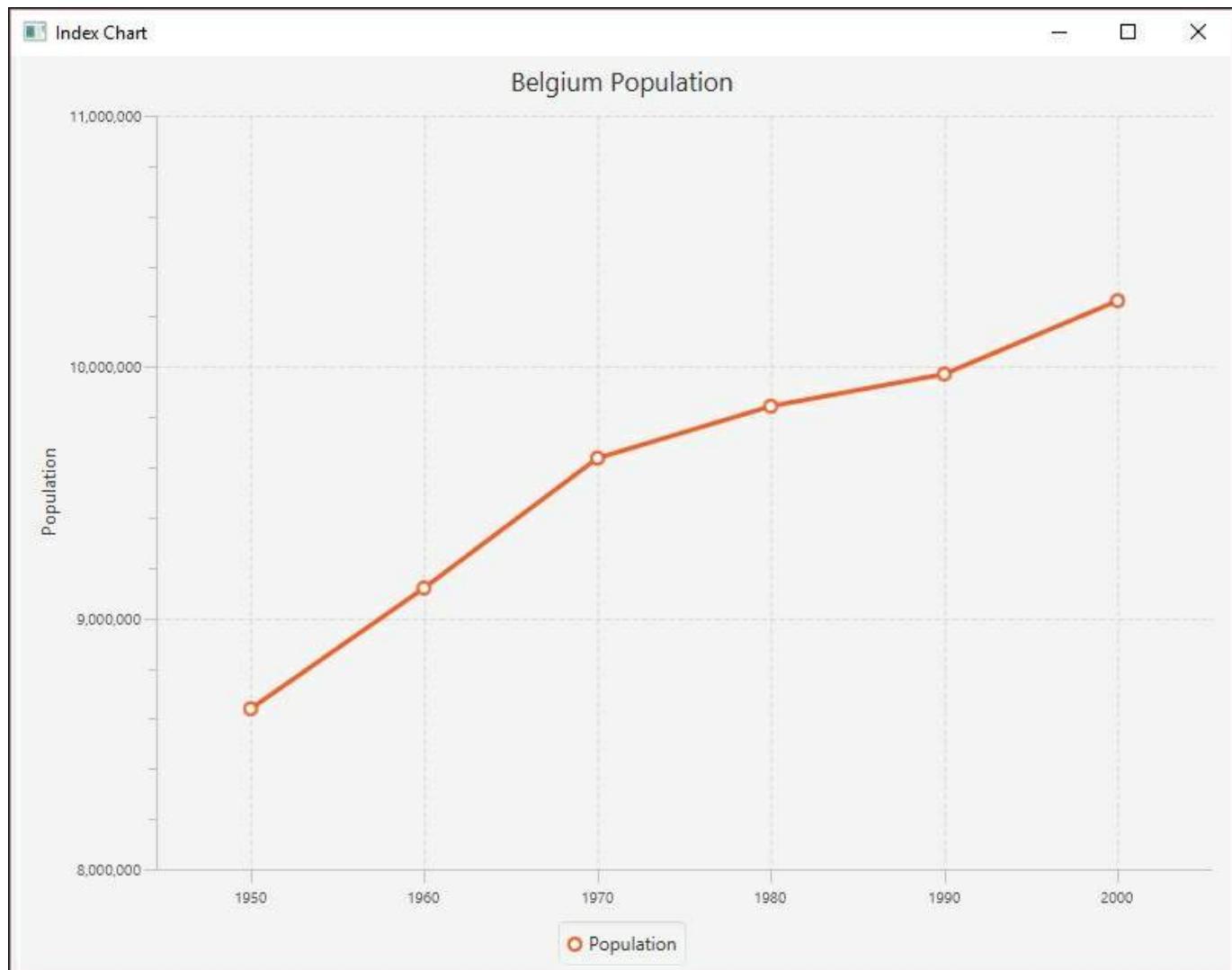


Time series graphs are a special type of graph used for displaying time-related values. These are most appropriate when the data analysis requires an understanding of how data changes over a period of time. In these graphs, the vertical axis corresponds to the values and the horizontal axis corresponds to particular points in time. In particular, this type of graph can be useful for identifying trends across time, or suggesting correlations between data values and particular events within a given time period.

For example, stock prices and home prices will change, but their rate of change varies. Pollution levels and crime rates also change over time. There are several techniques that visualize this type of data. Often, specific values are not as important as their trend over time.

An **index chart** is also called a line chart. **Line charts** use the *X* and *Y* axis to plot points on a grid. They can be used to represent time series data. These points are connected by lines, and these lines are used to compare values of multiple data at one time. This comparison is usually achieved by plotting independent variables, such as time, along the *X* axis, and independent variables, such as frequency or percentages, along the *Y* axis.

The following is a simple example of an index chart showing the distribution of population for selected European countries:



When we wish to arrange larger amounts of data in a compact and useful manner, we may opt for a stem and leaf plot. This type of visual expression allows you to demonstrate the correlation of one value to many values in a readable manner. The stem refers to a data value, and the leaves are the corresponding data points. One common example of this is a train timetable. In the following table, the departure times for a train are listed:

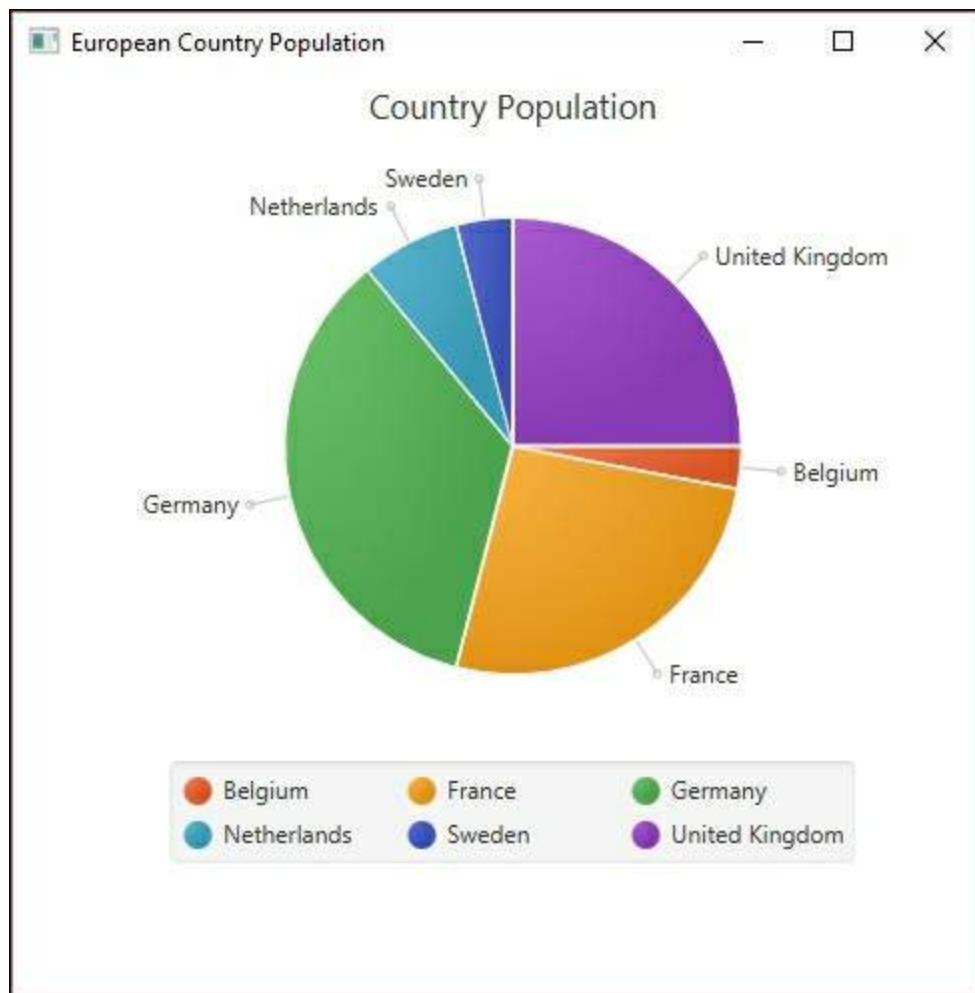
06:15	06 :20	06:25	06:30
06:40	06:45	06:55	07:15
07:20	07:25	07:30	07:40
07:45	07:55	08:00	08:12
08:24	08:36	08:48	09:00
09:12	09:24	09:36	09:48
10:00	10:12	10:24	10:36
10:48			

However, this table can be hard to read. Instead, in the following partial stem and leaf plot, the stem represents the hours at which a train may depart, while the leaves represent the minutes within each hour:

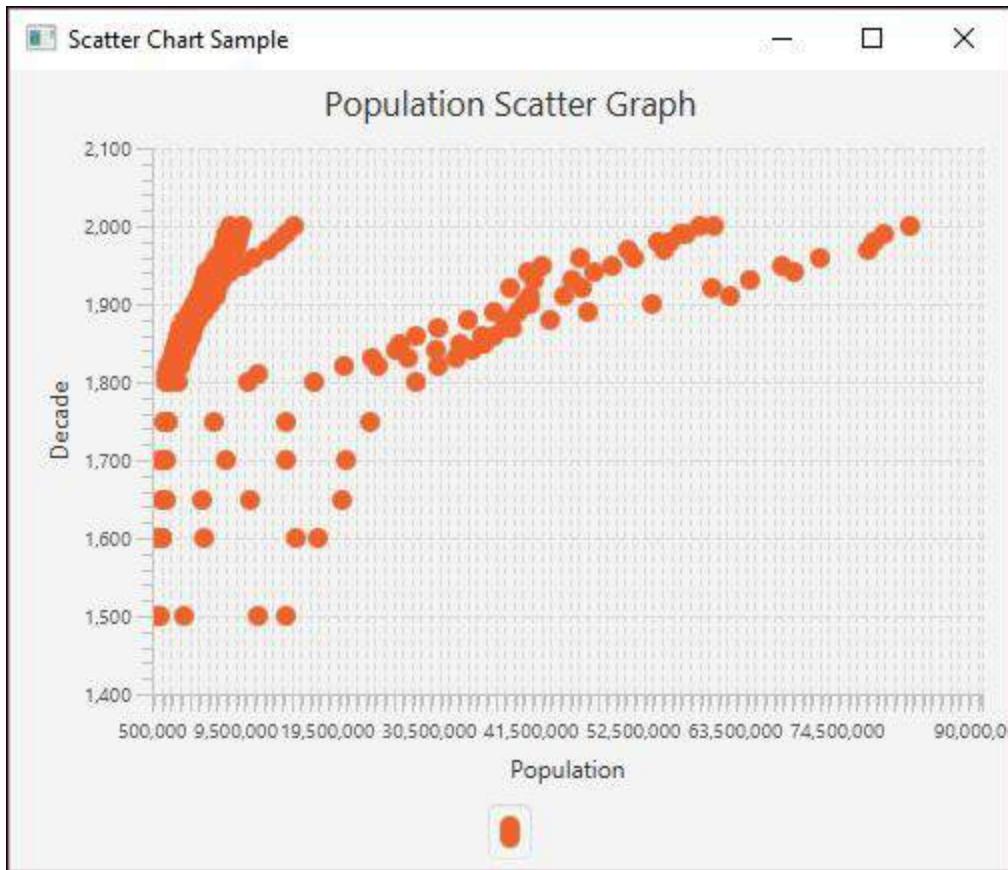
Hour	Minute
06	:15 :20 :25 :30 :40 :45 :55
07	:15 :20 :25 :30 :40 :45 :55
08	:00 :12 :24 :36 :48
09	:00 :12 :24 :36 :48
10	:00 :12 :24 :36 :48

This is much easier to read and process.

A very popular form of visualization in statistical analysis is the **histogram**. Histograms allow you to display frequencies within data using bars, similar to a bar chart. The main difference is that histograms are used to identify frequencies and trends within a dataset while bar charts are used to compare specific data values within a dataset. The following is an example of a histogram we will create in the *Creating histograms* section:



A **scatter plot** is simply collections of points, and analysis techniques, such as correlation or regression, can be used to identify trends within these types of graph. In the following scatter chart, as developed in *Creating scatter charts*, the population along the *X* axis is plotted against the decade along the *Y* axis:



Visual analysis goals

Each type of visual expression lends itself to different types of data and data analysis purposes. One common purpose of data analysis is **data classification**. This involves determining which subset within a dataset a particular data value belongs to. This process may occur early in the data analysis process because breaking data apart into manageable and related pieces simplifies the analysis process. Often, classification is not the end goal but rather an important intermediary step before further analysis can be undertaken.

Regression analysis is a complex and important form of data analysis. It involves studying relationships between independent and dependent variables, as well as multiple independent variables. This type of statistical analysis allows the analyst to identify ranges of acceptable or expected values and determine how individual values may fit into a larger dataset. Regression analysis is a significant part of machine learning, and we will discuss it in more detail in [Chapter 5, Statisitcal Data Analysis Techniques](#).

Clustering allows us to identify groups of data points within a particular set or class. While classification sorts data into similar types of datasets, clustering is concerned with the data within the set. For example, we may have a large dataset containing all feline species in the world, in the family Felidae. We could then classify these cats into two groups, Pantherinae (containing most larger cats) and Felinae (all other cats). Clustering would involve grouping subsets of similar cats within one of these classifications. For example, all tigers could be a cluster within the Pantherinae group.

Sometimes, our data analysis requires that we extract specific types of information from our dataset. The process of selecting the data to extract is known as **attribute selection** or **feature selection**. This process helps analysts simplify the data models and allows us to overcome issues with redundant or irrelevant information within our dataset.

With this introduction to basic plot and chart types, we will discuss Java support for creating these plots and charts.

Creating index charts

An index chart is a line chart that shows the percentage change of something over time. Frequently, such a chart is based on a single data attribute. In the following example, we will be using the Belgian population for six decades. The data is a subset of population data found at <https://ourworldindata.org/grapher/population-by-country?tab=data>:

Decade	Population
1950	8639369
1960	9118700
1970	9637800
1980	9846800
1990	9969310
2000	10263618

We start by creating the `MainApp` class, which extends `Application`. We create a series of instance variables. The `XYChart.Series` class represents a series of data points for some plot. In our case, this will be for the decades and population, which we will initialize shortly. The next declaration is for the `CategoryAxis` and `NumberAxis` instances. These represent the *X* and *Y* axes respectively. The declaration for the *Y* axis includes range and increment values for the population. This makes the chart a bit more readable. The last declaration is a string variable for the country:

```
public class MainApp extends Application {
    final XYChart.Series<String, Number> series =
        new XYChart.Series<>();
    final CategoryAxis xAxis = new CategoryAxis();
    final NumberAxis yAxis =
        new NumberAxis(8000000, 11000000, 1000000);
    final static String belgium = "Belgium";
    ...
}
```

In JavaFX, the `main` method usually launches the application using the base class `launch` method. Eventually, the `start` method is called, which we override. In this example, we call the `simpleLineChart` method where the user interface is created:

```
public static void main(String[] args) {
    launch(args);
}

public void start(Stage stage) {
    simpleIndexChart (stage);
}
```

The `simpleLineChart` follows and is passed an instance of the `Stage` class. This represents the client area of the application's window. We start by setting a title for the application and the line chart proper. The label of the *Y* axis is set. An instance of the `LineChart` class is initialized using the *X* and *Y* axis instances. This class represents the line chart:

```
public void simpleIndexChart (Stage stage) {
    stage.setTitle("Index Chart");
    lineChart.setTitle("Belgium Population");
    yAxis.setLabel("Population");
    final LineChart<String, Number> lineChart
        = new LineChart<>(xAxis, yAxis);

    ...
}
```

The series is given a name, and then the population for each decade is added to the series using the `addDataItem` helper method:

```
series.setName("Population");
addDataItem(series, "1950", 8639369);
addDataItem(series, "1960", 9118700);
addDataItem(series, "1970", 9637800);
addDataItem(series, "1980", 9846800);
addDataItem(series, "1990", 9969310);
addDataItem(series, "2000", 10263618);
```

The `addDataItem` method follows, which creates an `XYChart.Data` class instance using the `String` and `Number` values passed to it. It then adds the instance to the series:

```
public void addDataItem(XYChart.Series<String, Number> series,
    String x, Number y) {
```

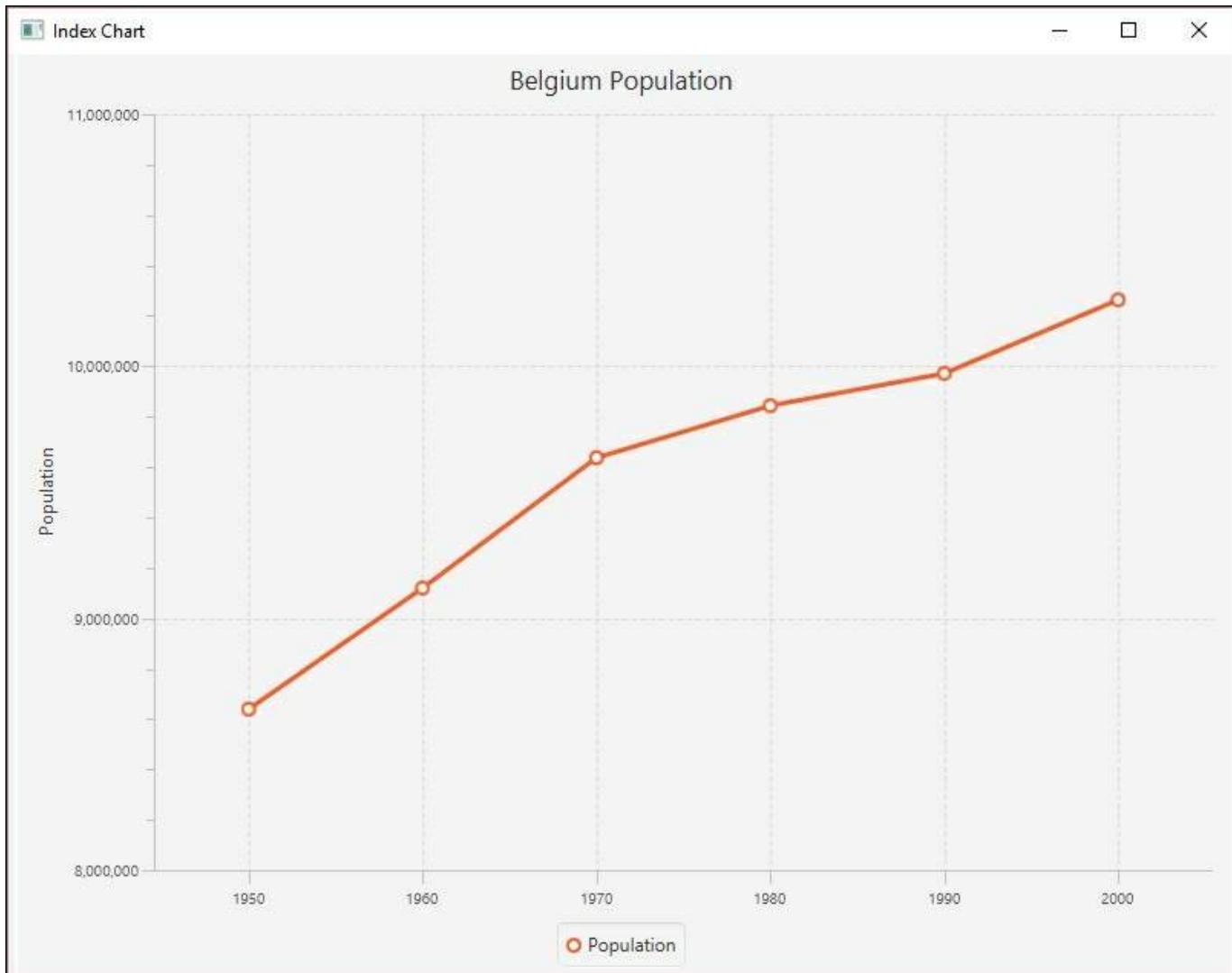
```
    series.getData().add(new XYChart.Data<>(x, y));  
}
```

The last part of the `simpleLineChart` method creates a `Scene` class instance that represents the content of the `stage`. JavaFX uses the concept of a stage and scene to deal with the internals of the application's GUI.

The `scene` is created using a line chart, and the application's size is set to 800 by 600 pixels. The series is then added to the line chart and `scene` is added to `stage`. The `show` method displays the application:

```
Scene scene = new Scene(lineChart, 800, 600);  
lineChart.getData().add(series);  
stage.setScene(scene);  
stage.show();
```

When the application executes the following window will be displayed:



Creating bar charts

A bar chart uses two axes with rectangular bars that can be either positioned either vertically or horizontally. The length of a bar is proportional to the value it represents. A bar chart can be used to show time series data.

In the following series of examples, we will be using a set of European country populations for three decades, as listed in the following table. The data is a subset of population data found at <https://ourworldindata.org/grapher/population-by-country?tab=data>:

Country	1950	1960	1970
Belgium	8,639,369	9,118,700	9,637,800
France	42,518,000	46,584,000	51,918,000
Germany	68,374,572	72,480,869	77,783,164
Netherlands	10,113,527	11,486,000	13,032,335
Sweden	7,014,005	7,480,395	8,042,803
United Kingdom	50,127,000	52,372,000	55,632,000

The first of three bar charts will be constructed using JavaFX. We start with a series of declarations for the countries as part of a class that extends the `Application` class:

```
public class MainApp extends Application {  
    final static String belgium = "Belgium";  
    final static String france = "France";  
    final static String germany = "Germany";  
    final static String netherlands = "Netherlands";  
    final static String sweden = "Sweden";  
    final static String unitedKingdom = "United Kingdom";  
  
    ...  
}
```

Next, we declared a series of instance variables that represent the parts of a graph. The first are `CategoryAxis` and `NumberAxis` instances:

```
final CategoryAxis xAxis = new CategoryAxis();
final NumberAxis yAxis = new NumberAxis();
```

The population and country data is stored in a series of `XYChart.Series` instances. Here, we have declared six different series, which use a string and number pair. The first example does not use all six series, but later examples will. We will initially assign a country string and its corresponding population to three series. These series will represent the populations for the decades 1950, 1960, and 1970:

```
final XYChart.Series<String, Number> series1 =
    new XYChart.Series<>();
final XYChart.Series<String, Number> series2
    new XYChart.Series<>();
final XYChart.Series<String, Number> series3 =
    new XYChart.Series<>();
final XYChart.Series<String, Number> series4 =
    new XYChart.Series<>();
final XYChart.Series<String, Number> series5 =
    new XYChart.Series<>();
final XYChart.Series<String, Number> series6 =
    new XYChart.Series<>();
```

We will start with two simple bar charts. The first one will show the countries as categories where the year changes occur within the category on the *X* axis and the population along the *Y* axis. The second shows the decades as categories containing the counties. The last example is a stacked bar chart.

Using country as the category

The elements of the bar chart are set up in the `simpleBarChartByCountry` method. The title of the chart is set and a `BarChart` class instance is created using the two axes. The chart, its *X* axis, and its *Y* axis also have labels that are initialized here:

```
public void simpleBarChartByCountry(Stage stage) {  
    stage.setTitle("Bar Chart");  
    final BarChart<String, Number> barChart  
        = new BarChart<>(xAxis, yAxis);  
    barChart.setTitle("Country Summary");  
    xAxis.setLabel("Country");  
    yAxis.setLabel("Population");  
    ...  
}
```

Next, the first three series are initialized with a name, and then the country and population data for that series. A helper method, `addDataItem`, as introduced in the previous section, is used to add the data to each series:

```
series1.setName("1950");  
addDataItem(series1, belgium, 8639369);  
addDataItem(series1, france, 42518000);  
addDataItem(series1, germany, 68374572);  
addDataItem(series1, netherlands, 10113527);  
addDataItem(series1, sweden, 7014005);  
addDataItem(series1, unitedKingdom, 50127000);  
  
series2.setName("1960");  
addDataItem(series2, belgium, 9118700);  
addDataItem(series2, france, 46584000);  
addDataItem(series2, germany, 72480869);  
addDataItem(series2, netherlands, 11486000);  
addDataItem(series2, sweden, 7480395);  
addDataItem(series2, unitedKingdom, 52372000);  
  
series3.setName("1970");  
addDataItem(series3, belgium, 9637800);  
addDataItem(series3, france, 51918000);  
addDataItem(series3, germany, 77783164);  
addDataItem(series3, netherlands, 13032335);  
addDataItem(series3, sweden, 8042803);  
addDataItem(series3, unitedKingdom, 55632000);
```

The last part of the method creates a `scene` instance. The three series are added to the

scene and the scene is attached to the stage using the setScene method. A stage is a class that essentially represents the client area of a window:

```
Scene scene = new Scene(barChart, 800, 600);
barChart.getData().addAll(series1, series2, series3);
stage.setScene(scene);
stage.show();
```

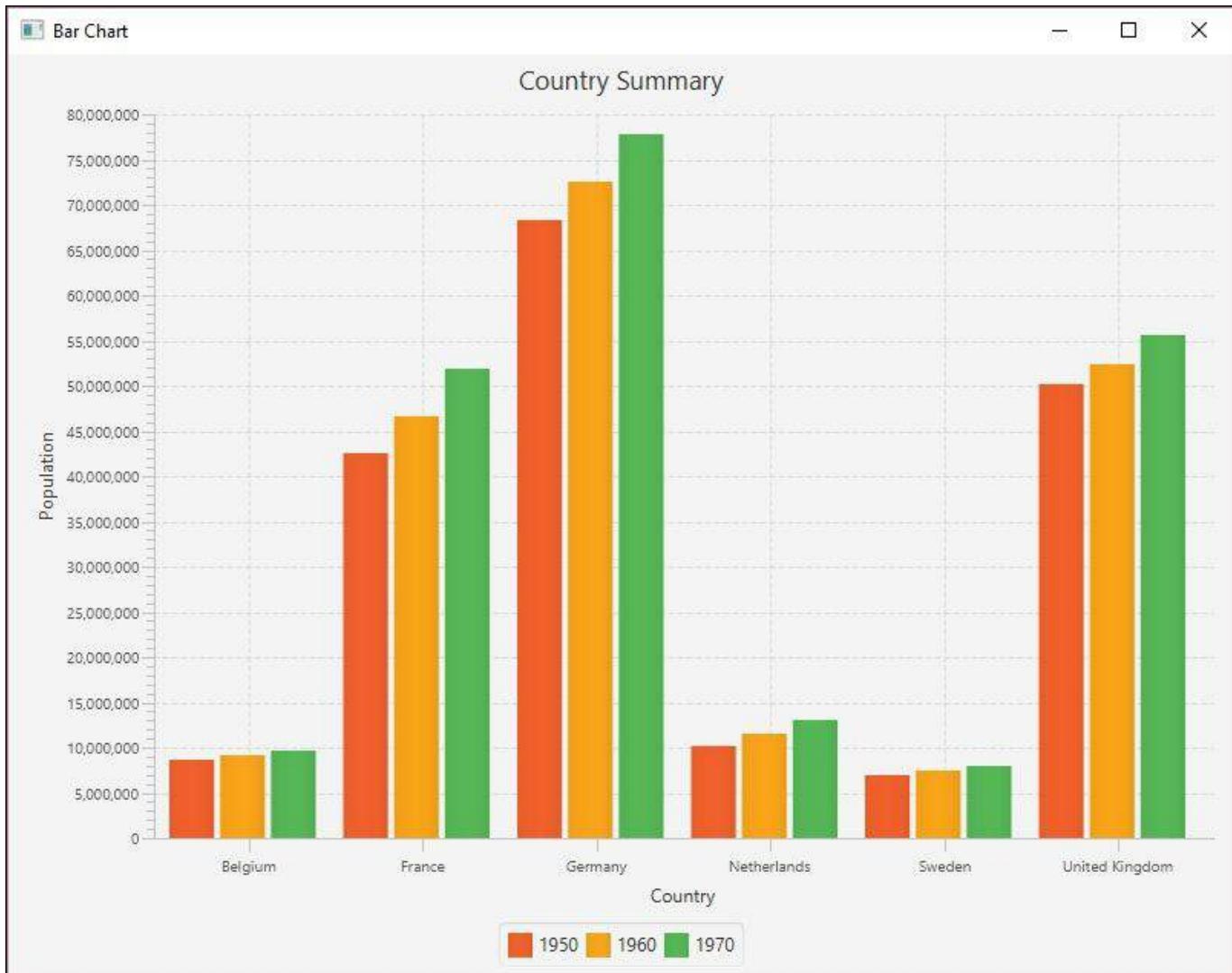
The last of the two methods is the start method, which is called automatically when the window is displayed. It is passed the Stage instance. Here, we call the simpleBarChartByCountry method:

```
public void start(Stage stage) {
    simpleBarChartByCountry(stage);
}
```

The main method consists of a call to the Application class's launch method:

```
public static void main(String[] args) {
    launch(args);
}
```

When the application is executed, the following graph is displayed:



Using decade as the category

In the following example, we will demonstrate how to display the same information, but we will organize the *X* axis categories by year. We will use the `simpleBarChartByYear` method, as shown next. The axis and titles are set up in the same way as before, but with different values for the title and labels:

```
public void simpleBarChartByYear(Stage stage) {  
    stage.setTitle("Bar Chart");  
    final BarChart<String, Number> barChart  
        = new BarChart<>(xAxis, yAxis);  
    barChart.setTitle("Year Summary");  
    xAxis.setLabel("Year");  
    yAxis.setLabel("Population");  
    ...  
}
```

The following string variables are declared for the three decades:

```
String year1950 = "1950";  
String year1960 = "1960";  
String year1970 = "1970";
```

The data series are created in the same way as before, except the country name is used for the series name and the year is used for the category. In addition, six series are used, one for each country:

```
series1.setName(belgium);  
addDataItem(series1, year1950, 8639369);  
addDataItem(series1, year1960, 9118700);  
addDataItem(series1, year1970, 9637800);  
  
series2.setName(france);  
addDataItem(series2, year1950, 42518000);  
addDataItem(series2, year1960, 46584000);  
addDataItem(series2, year1970, 51918000);  
  
series3.setName(germany);  
addDataItem(series3, year1950, 68374572);  
addDataItem(series3, year1960, 72480869);  
addDataItem(series3, year1970, 77783164);  
  
series4.setName(netherlands);  
addDataItem(series4, year1950, 10113527);  
addDataItem(series4, year1960, 11486000);
```

```
addDataItem(series4, year1970, 13032335);

series5.setName(sweden);
addDataItem(series5, year1950, 7014005);
addDataItem(series5, year1960, 7480395);
addDataItem(series5, year1970, 8042803);

series6.setName(unitedKingdom);
addDataItem(series6, year1950, 50127000);
addDataItem(series6, year1960, 52372000);
addDataItem(series6, year1970, 55632000);
```

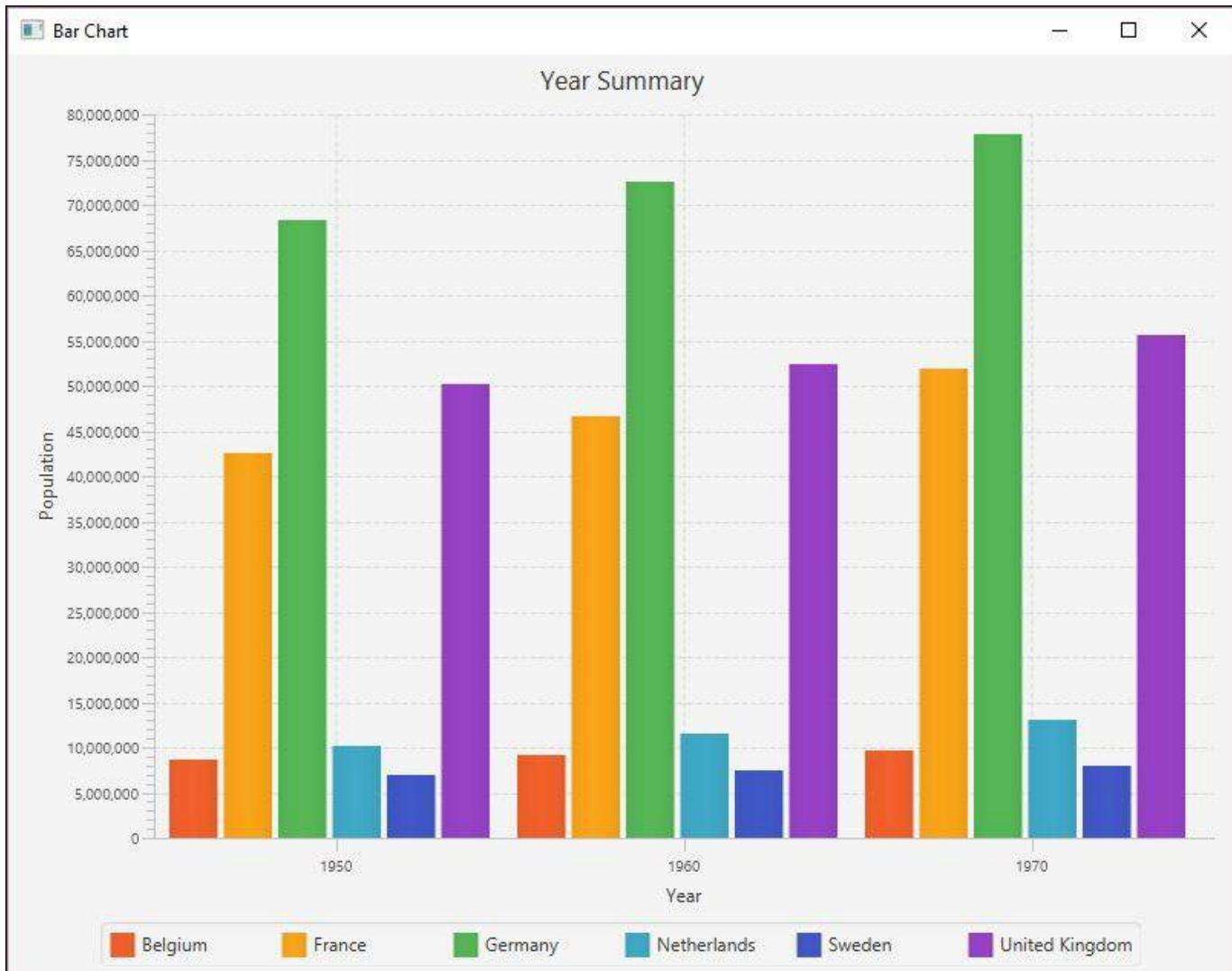
The `scene` is created and attached to the stage:

```
Scene scene = new Scene(barChart, 800, 600);
barChart.getData().addAll(series1, series2,
    series3, series4, series5, series6);
stage.setScene(scene);
stage.show();
```

The `main` method is unchanged, but the `start` method calls the `simpleBarChartByYear` method instead:

```
public void start(Stage stage) {
    simpleBarChartByYear(stage);
}
```

When the application is executed, the following graph is displayed:



Creating stacked graphs

An area chart depicts information by allocating more space for larger values. By stacking area charts on top of each other we create a stacked graph, sometimes called a stream graph. However, stacked graphs do not work well with negative values and cannot be used for data where summation does not make sense, such as with temperatures. If too many graphs are stacked, then it can become difficult to interpret.

Next, we will show how to create a stacked bar chart. The `stackedGraphExample` method contains the code to create the bar chart. We start with familiar code to set the title and labels. However, for the *X* axis, the `setCategories` method

`FXCollections.<String>observableArrayList` instance is used to set the categories. The argument of this constructor is an array of strings created by the `Arrays` class's `asList` method and the names of the countries:

```
public void stackedGraphExample(Stage stage) {  
    stage.setTitle("Stacked Bar Chart");  
    final StackedBarChart<String, Number> stackedBarChart  
        = new StackedBarChart<>(xAxis, yAxis);  
    stackedBarChart.setTitle("Country Population");  
    xAxis.setLabel("Country");  
    xAxis.setCategories(  
        FXCollections.<String>observableArrayList(  
            Arrays.asList(belgium, germany, france,  
                netherlands, sweden, unitedKingdom)));  
    yAxis.setLabel("Population");  
    ...  
}
```

The series are initialized with the year being used for the series name and the country, and their population being added using the helper method `addDataItem`. The scene is then created:

```
series1.setName("1950");  
addDataItem(series1, belgium, 8639369);  
addDataItem(series1, france, 42518000);  
addDataItem(series1, germany, 68374572);  
addDataItem(series1, netherlands, 10113527);  
addDataItem(series1, sweden, 7014005);  
addDataItem(series1, unitedKingdom, 50127000);  
  
series2.setName("1960");
```

```

addDataItem(series2, belgium, 9118700);
addDataItem(series2, france, 46584000);
addDataItem(series2, germany, 72480869);
addDataItem(series2, netherlands, 11486000);
addDataItem(series2, sweden, 7480395);
addDataItem(series2, unitedKingdom, 52372000);

series3.setName("1970");
addDataItem(series3, belgium, 9637800);
addDataItem(series3, france, 51918000);
addDataItem(series3, germany, 77783164);
addDataItem(series3, netherlands, 13032335);
addDataItem(series3, sweden, 8042803);
addDataItem(series3, unitedKingdom, 55632000);

Scene scene = new Scene(stackedBarChart, 800, 600);
stackedBarChart.getData().addAll(series1, series2, series3);
stage.setScene(scene);
stage.show();

```

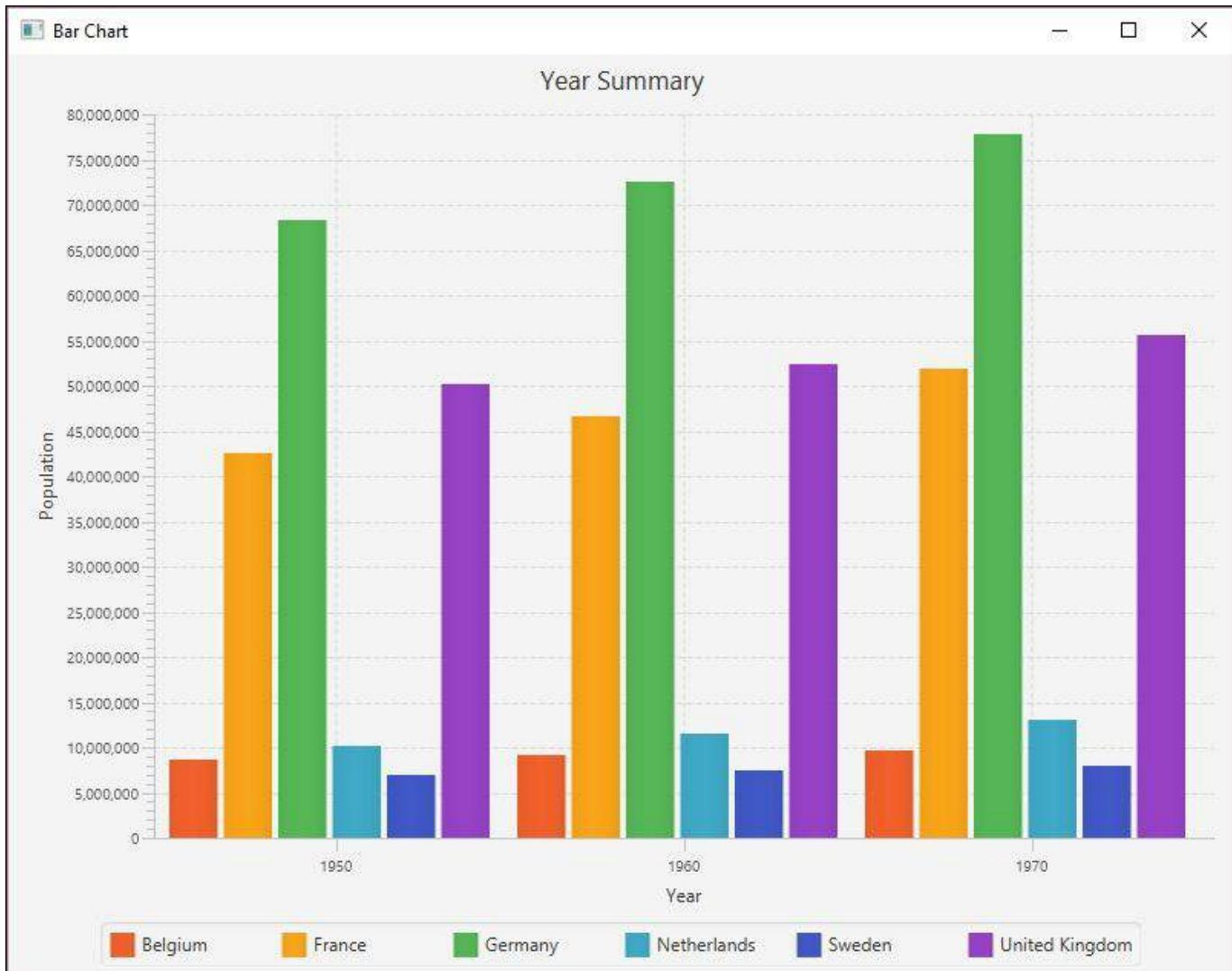
The `main` method is unchanged, but the `start` method calls the `stackedGraphExample` method instead:

```

public void start(Stage stage) {
    stackedGraphExample(stage);
}

```

When the application is executed, the following graph is displayed:



Creating pie charts

The following pie chart example is based on the 2000 population of selected European countries as summarized here:

Country	Population	Percentage
Belgium	10,263,618	3
France	61,137,000	26
Germany	82,187,909	35
Netherlands	15,907,853	7
Sweden	8,872,000	4
United Kingdom	59,522,468	25

The JavaFX implementation uses the same `Application` base class and `main` method as used in the previous examples. We will not use a separate method for creating the GUI, but instead place this code in the `start` method, as shown here:

```
public class PieChartSample extends Application {

    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("European Country Population");
        stage.setWidth(500);
        stage.setHeight(500);
        ...
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

A pie chart is represented by the `PieChart` class. We can create and initialize the pie chart in the constructor by using an `ObservableList` of pie chart data. This data

consists of a series of `PieChart.Data` instances, each containing a text label and a percentage value.

The next sequence creates an `ObservableList` instance based on the European population data presented earlier. The `FXCollections` class's `observableArrayList` method returns an `ObservableList` instance with a list of pie chart data:

```
ObservableList<PieChart.Data> pieChartData =  
    FXCollections.observableArrayList(  
        new PieChart.Data("Belgium", 3),  
        new PieChart.Data("France", 26),  
        new PieChart.Data("Germany", 35),  
        new PieChart.Data("Netherlands", 7),  
        new PieChart.Data("Sweden", 4),  
        new PieChart.Data("United Kingdom", 25));
```

We then create the pie chart and set its title. The pie chart is then added to the `scene`, the `scene` is associated with the `stage`, and then the window is displayed:

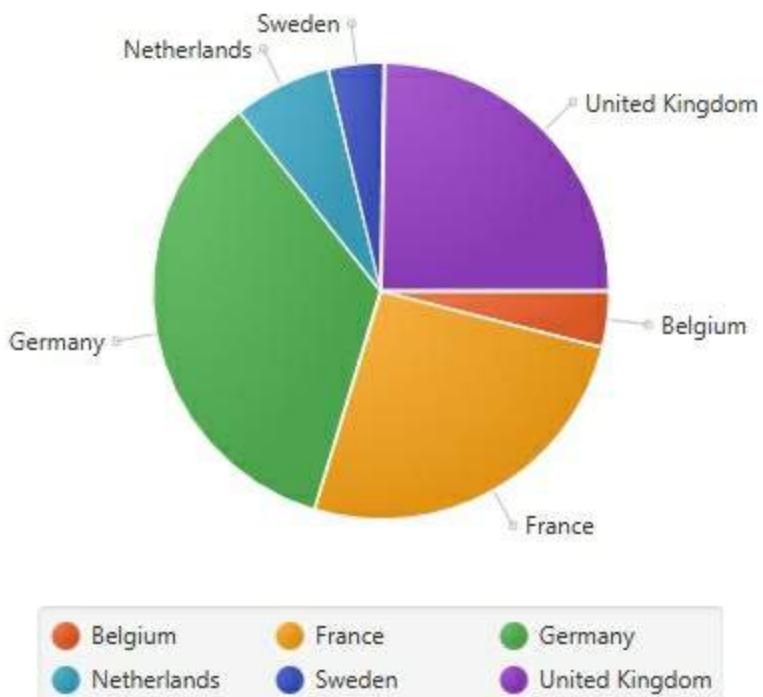
```
final PieChart pieChart = new PieChart(pieChartData);  
pieChart.setTitle("Country Population");  
((Group) scene.getRoot()).getChildren().add(pieChart);  
stage.setScene(scene);  
stage.show();
```

When the application is executed, the following graph is displayed:

European Country Population

- □ X

Country Population



Creating scatter charts

Scatter charts also use the `XYChart.Series` class in JavaFX. For this example, we will use a set of European data that includes the previous Europeans countries and their population data for the decades 1500 through 2000. This information is stored in a file called `EuropeanScatterData.csv`. The first part of this file is shown here:

```
1500 1400000
1600 1600000
1650 1500000
1700 2000000
1750 2250000
1800 3250000
1820 3434000
1830 3750000
1840 4080000
...
...
```

We start with the declaration of the JavaFX `MainApp` class, as shown next. The `main` method launches the application and the `start` method creates the user interface:

```
public class MainApp extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        ...
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Within the `start` method we set the title, create the axes, and create an instance of the `ScatterChart` that represents the scatter plot. The `NumberAxis` class's constructors used values that better match the data range than the default values used by its default constructor:

```
stage.setTitle("Scatter Chart Sample");
final NumberAxis yAxis = new NumberAxis(1400, 2100, 100);
final NumberAxis xAxis = new NumberAxis(500000, 90000000,
    1000000);
final ScatterChart<Number, Number> scatterChart = new
    ScatterChart<>(xAxis, yAxis);
```

Next, the axes' labels are set along with the scatter chart's title:

```
xAxis.setLabel("Population");
yAxis.setLabel("Decade");
scatterChart.setTitle("Population Scatter Graph");
```

An instance of the `XYChart.Series` class is created and named:

```
XYChart.Series series = new XYChart.Series();
```

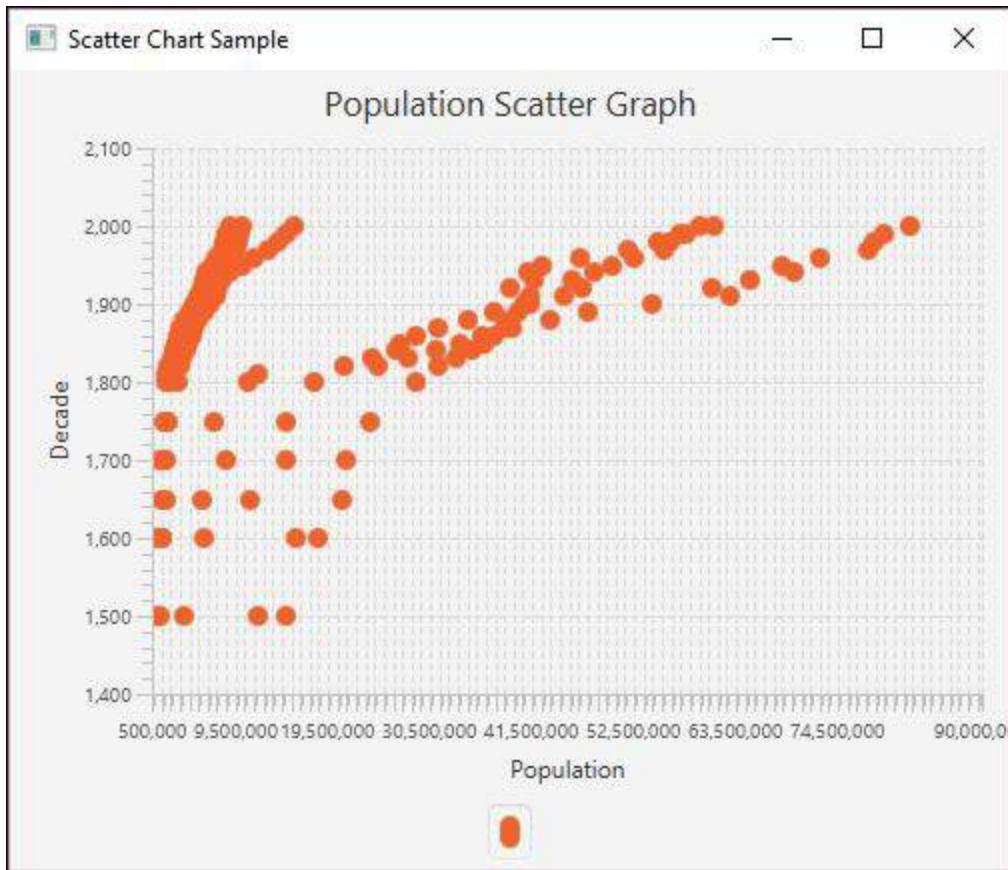
The series is populated using a `CSVReader` class instance and the file `EuropeanScatterData.csv`. This process was discussed in [Chapter 3, Data Cleaning](#):

```
try (CSVReader dataReader = new CSVReader(new
FileReader("EuropeanScatterData.csv"), ',', ',')) {
    String[] nextLine;
    while ((nextLine = dataReader.readNext()) != null) {
        int decade = Integer.parseInt(nextLine[0]);
        int population = Integer.parseInt(nextLine[1]);
        series.getData().add(new XYChart.Data(
            population, decade));
        out.println("Decade: " + decade +
                    " Population: " + population);
    }
}
scatterChart.getData().addAll(series);
```

The JavaFX scene and stage are created, and then the plot is displayed:

```
Scene scene = new Scene(scatterChart, 500, 400);
stage.setScene(scene);
stage.show();
```

When the application is executed, the following graph is displayed:



Creating histograms

Histograms, though similar in appearance to bar charts, are used to display the frequency of data items in relation to other items within the dataset. Each of the following examples using GRAL will use the `DataTable` class to initially hold the data to be displayed. In this example, we will read data from a sample file called `AgeofMarriage.csv`. This comma-separated file holds a list of ages at which people were first married.

We will create a new class, called `HistogramExample`, which extends the `JFrame` class and contains the following code within its constructor. We first create a `DataReader` object to specify that the data is in CSV format. We then use a try-catch block to handle IO exceptions and call the `DataReader` class's `read` method to place the data directly into a `DataTable` object. The first parameter of the `read` method is a `FileInputStream` object, and the second specifies the type of data expected from within the file:

```
DataReader readType=
    DataReaderFactory.getInstance().get("text/csv");
String fileName = "C://AgeofMarriage.csv";
try {
    DataTable histData = (DataTable) readType.read(
        New FileInputStream(fileName), Integer.class);
    ...
}
```

Next, we create a `Number` array to specify the ages for which we expect to have data. In this case, we expect the ages of marriage will range from 19 to 30. We use this array to create our `Histogram` object. We include our `DataTable` from earlier and specify the orientation as well. Then we create our `DataSource`, specify our starting age, and specify the spacing along our *X* axis:

```
Number ageRange[] = {19,20,21,22,23,24,25,26,27,28,29,30};
Histogram sampleHisto = new Histogram1D(
    histData, Orientation.VERTICAL, ageRange);
DataSource sampleHistData = new EnumeratedData(sampleHisto, 19,
    1.0);
```

We use the `BarPlot` class to create our histogram from the data we read in earlier:

```
BarPlot testPlot = new BarPlot(sampleHistData);
```

The next few steps serve to format various aspects of our histogram. We use the `setInsets` method to specify how much space to place around each side of the graph within the window. We can provide a title for our graph and specify the bar width:

```
testPlot.setInsets(new Insets2D.Double(20.0, 50.0, 50.0, 20.0));
testPlot.getTitle().setText("Average Age of Marriage");
testPlot.setBarWidth(0.7);
```

We also need to format our *X* and *Y* axes. We have chosen to set our range for the *X* axis to closely match our expected age range but to provide some space on the side of the graph. Because we know the amount of sample data, we set our *Y* axis to range from 0 to 10. In a business application, these ranges would be calculated by examining the actual dataset. We can also specify whether we want tick marks to show and where we would like the axes to intersect:

```
testPlot.getAxis(BarPlot.AXIS_X).setRange(18, 30.0);
testPlot.getAxisRenderer(BarPlot.AXIS_X).setTickAlignment(0.0);
testPlot.getAxisRenderer(BarPlot.AXIS_X).setTickSpacing(1);
testPlot.getAxisRenderer(BarPlot.AXIS_X).setMinorTicksVisible(false);

testPlot.getAxis(BarPlot.AXIS_Y).setRange(0.0, 10.0);
testPlot.getAxisRenderer(BarPlot.AXIS_Y).setTickAlignment(0.0);
testPlot.getAxisRenderer(BarPlot.AXIS_Y).setMinorTicksVisible(false);
testPlot.getAxisRenderer(BarPlot.AXIS_Y).setIntersection(0);
```

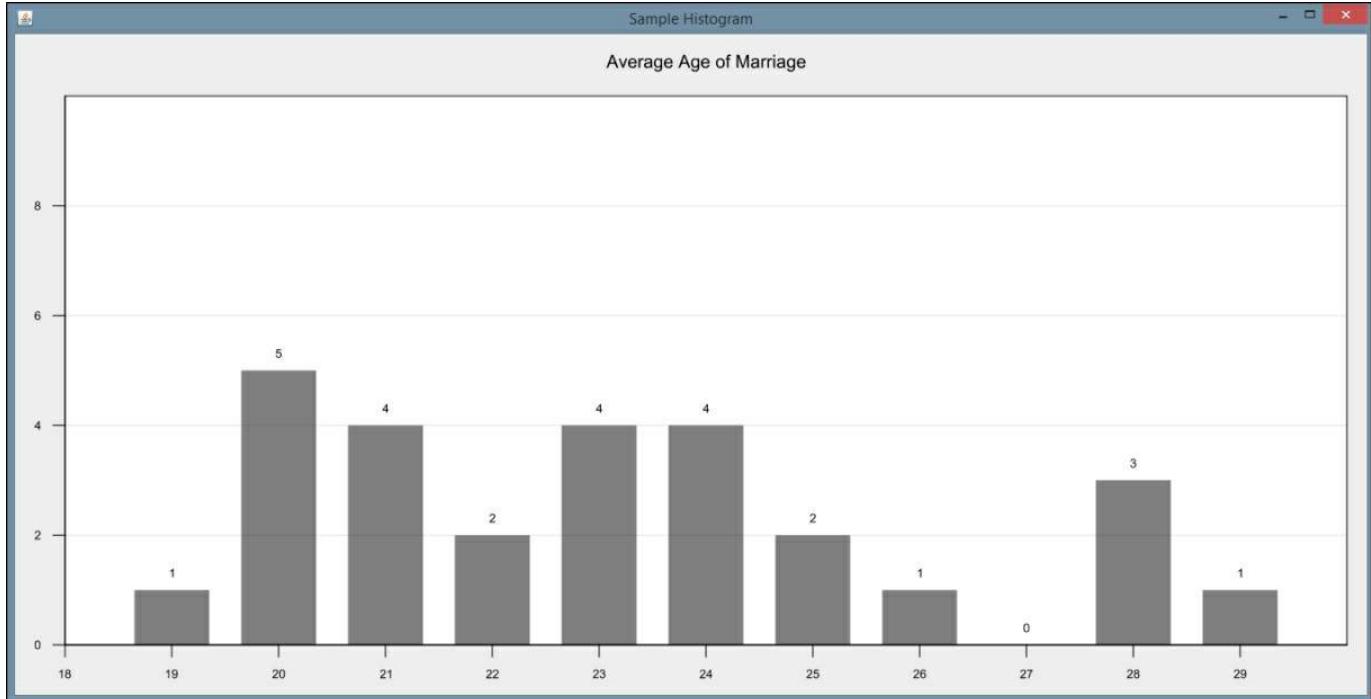
We also have a lot of flexibility with the color and values displayed on the graph. In this example, we have chosen to display the frequency value for each age and set our graph color to `black`:

```
PointRenderer renderHist =
    testPlot.getPointRenderers(sampleHistData).get(0);
renderHist.setColor(GraphicsUtils.deriveWithAlpha(Color.black,
    128));
renderHist.setValueVisible(true);
```

Finally, we set several properties for how we want our window to display:

```
InteractivePanel pan = new InteractivePanel(testPlot);
pan.setPannable(false);
pan.setZoomable(false);
add(pan);
setSize(1500, 700);
this.setVisible(true);
```

When the application is executed, the following graph is displayed:



Creating donut charts

Donut charts are similar to pie charts, but they are missing the middle section (hence the name donut). Some analysts prefer donut charts to pie charts because they do not emphasize the size of each piece within the chart and are easier to compare to other donut charts. They also provide the added advantage of taking up less space, allowing for more formatting options in the display.

In this example, we will assume our data is already populated in a two-dimensional array called `ageCount`. The first row of the array contains the possible age values, ranging again from 19 to 30 (inclusive). The second row contains the number of data values equal to each age. For example, in our dataset, there are six data values equal to 19, so `ageCount [0] [1]` contains the number six.

We create a `DataTable` and use the `add` method to add our values from the array. Notice we are testing to see if the value of a particular age is zero. In our test case, there will be zero data values equal to 23. We are opting to add a blank space in our donut chart if there are no data values for that point. This is accomplished by using a negative number as the first parameter in the `add` method. This will set an empty space of size 3:

```
DataTable donutData = new DataTable(Integer.class, Integer.class);
for(int Y = 0; Y < ageCount[0].length; y++) {
    if(ageCount[1][y] == 0) {
        donutData.add(-3, ageCount[0][y]);
    }else{
        donutData.add(ageCount[1][y], ageCount[0][y]);
    }
}
```

Next, we create our donut plot using the `PiePlot` class. We set basic properties of the plot, including specifying the values for the legend. In this case, we want our legend to reflect our age possibilities, so we use the `setLabelColumn` method to change the default labels. We also set our insets as we did in the previous example:

```
PiePlot testPlot = new PiePlot(donutData);
((ValueLegend) testPlot.getLegend()).setLabelColumn(1);
testPlot.getTitle().setText("Donut Plot Example");
testPlot.setRadius(0.9);
testPlot.setLegendVisible(true);
```

```
testPlot.setInsets(new Insets2D.Double(20.0, 20.0, 20.0, 20.0));
```

Next, we create a `PieSliceRenderer` object to set more advanced properties.

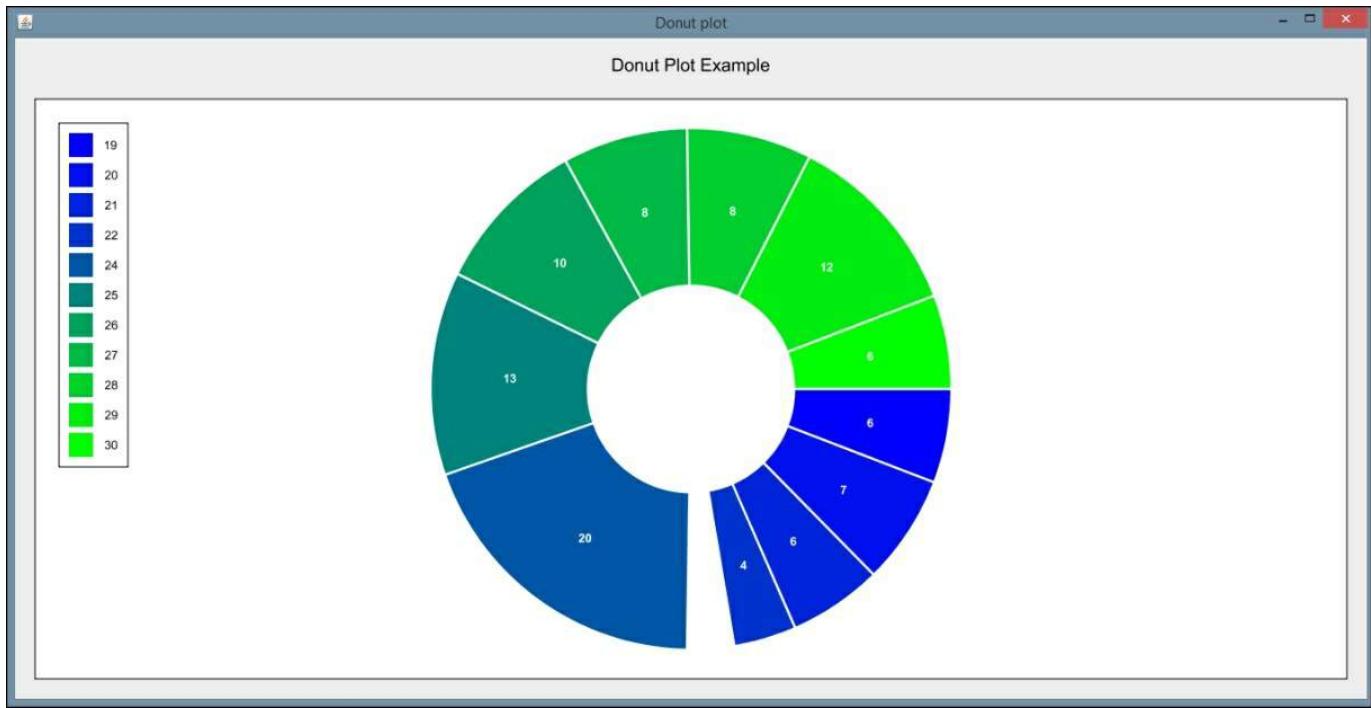
Because a donut plot is basically a pie plot in essence, we will render a donut plot by calling the `setInnerRadius` method. We also specify the gap between the pie slices, the colors used, and the style of the labels:

```
PieSliceRenderer renderPie = (PieSliceRenderer)
testPlot.getPointRenderer(donutData);
renderPie.setInnerRadius(0.4);
renderPie.setGap(0.2);
LinearGradient colors = new LinearGradient(
    Color.blue, Color.green);
renderPie.setColor(colors);
renderPie.setValueVisible(true);
renderPie.setValueColor(Color.WHITE);
renderPie.setValueFont(Font.decode(null).deriveFont(Font.BOLD));
```

Finally, we create our panel and set its size:

```
add(new InteractivePanel(testPlot), BorderLayout.CENTER);
setSize(1500, 700);
setVisible(true);
```

When the application is executed, the following graph is displayed:



Creating bubble charts

Bubble charts are similar to scatter plots except they represent data with three dimensions. The first two dimensions are expressed on the *X* and *Y* axes and the third is represented by the size of the point plotted. This can be helpful in determining relationships between data values.

We will again use the `DataTable` class to initially hold the data to be displayed. In this example, we will read data from a sample file called `MarriageByYears.csv`. This is also a CSV file, and contains one column representing the year a marriage occurred, a second column holding the age at which a person was married, and a third column holding integers representing marital satisfaction on a scale from 1 (least satisfied) to 10 (most satisfied). We create a `DataSeries` to represent our type of desired data plot and then create a `XYPlot` object:

```
DataReader readType =
    DataReaderFactory.getInstance().get("text/csv");
String fileName = "C://MarriageByYears.csv";
try {
    DataTable bubbleData = (DataTable) readType.read(
        new FileInputStream(fileName), Integer.class,
        Integer.class, Integer.class);
    DataSeries bubbleSeries = new DataSeries("Bubble", bubbleData);
    XYPlot testPlot = new XYPlot(bubbleSeries);
```

Next, we set basic property information about our chart. We will set the color and turn off the vertical and horizontal grids in this example. We will also make our *X* and *Y* axes invisible in this example. Notice that we still set a range for the axes, even though they are not displayed:

```
testPlot.setInsets(new Insets2D.Double(30.0));
testPlot.setBackground(new Color(0.75f, 0.75f, 0.75f));
XYPlotArea2D areaProp = (XYPlotArea2D) testPlot.getPlotArea();
areaProp.setBorderColor(null);
areaProp.setMajorGridX(false);
areaProp.setMajorGridY(false);
areaProp.setClippingArea(null);

testPlot.getAxisRenderer(XYPlot.AXIS_X).setShapeVisible(false);
testPlot.getAxisRenderer(XYPlot.AXIS_X).setTicksVisible(false);
testPlot.getAxisRenderer(XYPlot.AXIS_Y).setShapeVisible(false);
```

```
testPlot.getAxisRenderer(XYPlot.AXIS_Y).setTicksVisible(false);  
testPlot.getAxis(XYPlot.AXIS_X).setRange(1940, 2020);  
testPlot.getAxis(XYPlot.AXIS_Y).setRange(17, 30);
```

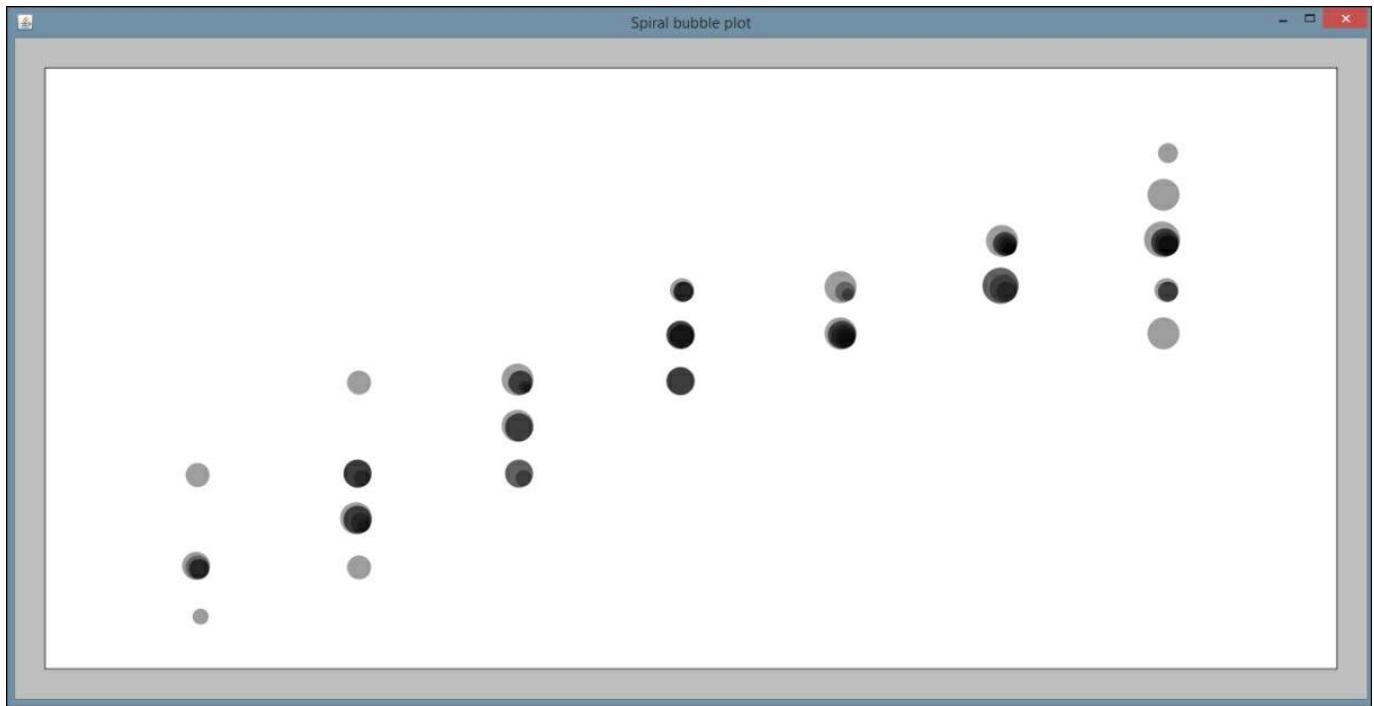
We can also set properties related to the bubbles drawn on the chart. Here, we set the color and shape, and specify which column of the data will be used to scale the shapes. In this case, the third column, with the marital satisfaction rating, will be used. We set it using the `setColumn` method:

```
Color color = GraphicsUtils.deriveWithAlpha(Color.black, 96);  
SizeablePointRenderer renderBubble = new SizeablePointRenderer();  
renderBubble.setShape(new Ellipse2D.Double(-3.5, -3.5, 4.0, 4.0));  
renderBubble.setColor(color);  
renderBubble.setColumn(2);  
testPlot.setPointRenderers(bubbleSeries, renderBubble);
```

Finally, we create our panel and set its size:

```
add(new InteractivePanel(testPlot), BorderLayout.CENTER);  
setSize(new Dimension(1500, 700));  
setVisible(true);
```

When the application is executed, the following graph is displayed. Notice both the size and color of the points changes depending upon the frequency of that particular data point:



Summary

In this chapter, we introduce basic graphs, plots, and charts used to visualize data. The process of visualization enables an analyst to graphically examine the data under review. This is more intuitive, and often facilitates the rapid identification of anomalies in the data that can be hard to extract from the raw data.

Several visual representations were examined, including line charts, a variety of bar charts, pie charts, scatterplots, histograms, donut charts, and bubble charts. Each of these graphical depictions of data provides a different perspective of the data being analyzed. The most appropriate technique depends on the nature of the data being used. While we have not covered all of the possible graphical techniques, this sample provides a good overview of what is available.

We were also concerned with how Java is used to draw these graphics. Many of the examples used JavaFX. This is a readily available tool that is bundled with Java SE. However, there are several other libraries available. We used GRAL to illustrate how to generate some graphs.

With the overview of visualization techniques, we are ready to move on to other topics, where visualization will be used to better convey the essence of data science techniques. In the next chapter, we will introduce basic statistical processes, including linear regression, and we will use the techniques introduced in this chapter.

Chapter 5. Statistical Data Analysis Techniques

The intent of this chapter is not to make the reader an expert in statistical techniques. Rather, it is to familiarize the reader with the basic statistical techniques in use and demonstrate how Java can support statistical analysis. While there are quite a variety of data analysis techniques, in this chapter, we will focus on the more common tasks.

These techniques range from the relatively simple mean calculation to sophisticated regression analysis models. Statistical analysis can be a very complicated process and requires significant study to be conducted properly. We will start with an introduction to basic statistical analysis techniques, including calculating the mean, median, mode, and standard deviation of a dataset. There are numerous approaches used to calculate these values, which we will demonstrate using standard Java and third-party APIs. We will also briefly discuss sample size and hypothesis testing.

Regression analysis is an important technique for analyzing data. The technique creates a line that tries to match the dataset. The equation representing the line can be used to predict future behavior. There are several types of regression analysis. In this chapter, we will focus on simple linear regression and multiple regression. With simple linear regression, a single factor such as age is used to predict some behavior such as the likelihood of eating out. With multiple regression, multiple factors such as age, income level, and marital status may be used to predict how often a person eats out.

Predictive analytics, or analysis, is concerned with predicting future events. Many of the techniques used in this book are concerned with making predictions. Specifically, the regression analysis part of this chapter predicts future behavior.

Before we see how Java supports regression analysis, we need to discuss basic statistical techniques. We begin with mean, mode, and median.

In this chapter, we will cover the following topics:

- Working with mean, mode, and median
- Standard deviation and sample size determination
- Hypothesis testing
- Regression analysis

Working with mean, mode, and median

The mean, median, and mode are basic ways to describe characteristics or summarize information from a dataset. When a new, large dataset is first encountered, it can be helpful to know basic information about it to direct further analysis. These values are often used in later analysis to generate more complex measurements and conclusions. This can occur when we use the mean of a dataset to calculate the standard deviation, which we will demonstrate in the *Standard deviation* section of this chapter.

Calculating the mean

The term **mean**, also called the average, is computed by adding values in a list and then dividing the sum by the number of values. This technique is useful for determining the general trend for a set of numbers. It can also be used to fill in missing data elements. We are going to examine several ways to calculate the mean for a given set of data using standard Java libraries as well as third-party APIs.

Using simple Java techniques to find mean

In our first example, we will demonstrate a basic way to calculate mean using standard Java capabilities. We will use an array of `double` values called `testData`:

```
double[] testData = {12.5, 18.7, 11.2, 19.0, 22.1, 14.3, 16.9, 12.5,  
17.8, 16.9};
```

We create a `double` variable to hold the sum of all of the values and a `double` variable to hold the `mean`. A loop is used to iterate through the data and add values together. Next, the sum is divided by the `length` of our array (the total number of elements) to calculate the `mean`:

```
double total = 0;  
for (double element : testData) {  
    total += element;  
}  
double mean = total / testData.length;  
out.println("The mean is " + mean);
```

Our output is as follows:

The mean is 16.19

Using Java 8 techniques to find mean

Java 8 provided additional capabilities with the introduction of optional classes. We are going to use the `OptionalDouble` class in conjunction with the `Arrays` class's `stream` method in this example. We will use the same array of doubles we used in the previous example to create an `OptionalDouble` object. If any of the numbers in the array, or the sum of the numbers in the array, is not a real number, the value of the `OptionalDouble` object will also not be a real number:

```
OptionalDouble mean = Arrays.stream(testData).average();
```

We use the `isPresent` method to determine whether we calculated a valid number for our mean. If we do not get a good result, the `isPresent` method will return `false` and we can handle any exceptions:

```
if (mean.isPresent()) {  
    out.println("The mean is " + mean.getAsDouble());  
} else {  
    out.println("The stream was empty");  
}
```

Our output is the following:

The mean is 16.19

Another, more succinct, technique using the `OptionalDouble` class involves lambda expressions and the `ifPresent` method. This method executes its argument if `mean` is a valid `OptionalDouble` object:

```
OptionalDouble mean = Arrays.stream(testData).average();  
mean.ifPresent(x-> out.println("The mean is " + x));
```

Our output is as follows:

The mean is 16.19

Finally, we can use the `orElse` method to either print the mean or an alternate value if `mean` is not a valid `OptionalDouble` object:

```
OptionalDouble mean = Arrays.stream(testData).average();  
out.println("The mean is " + mean.orElse(0));
```

Our output is the same:

The mean is 16.19

For our next two mean examples, we will use third-party libraries and continue using the array of doubles, `testData`.

Using Google Guava to find mean

In this example, we will use Google Guava libraries, introduced in [Chapter 3](#), *Data Cleaning*. The `Stats` class provides functionalities for handling numeric data, including finding mean and standard deviation, which we will demonstrate later. To calculate the `mean`, we first create a `Stats` object using our `testData` array and then execute the `mean` method:

```
Stats testStat = Stats.of(testData);
double mean = testStat.mean();
out.println("The mean is " + mean);
```

Notice the difference between the default format of the output in this example.

Using Apache Commons to find mean

In our final mean examples, we use Apache Commons libraries, also introduced in [Chapter 3](#), *Data Cleaning*. We first create a `Mean` object and then execute the `evaluate` method using our `testData`. This method returns a `double`, representing the mean of the values in the array:

```
Mean mean = new Mean();
double average = mean.evaluate(testData);
out.println("The mean is " + average);
```

Our output is the following:

The mean is 16.19

Apache Commons also provides a helpful `DescriptiveStatistics` class. We will use this later to demonstrate median and standard deviation, but first we will begin by calculating the mean. Using the `SynchronizedDescriptiveStatistics` class is advantageous as it is synchronized and therefore thread safe.

We start by creating our `DescriptiveStatistics` object, `statTest`. We then loop through our double array and add each item to `statTest`. We can then invoke the `getMean` method to calculate the `mean`:

```
DescriptiveStatistics statTest =
    new SynchronizedDescriptiveStatistics();
for(double num : testData) {
    statTest.addValue(num);
}
out.println("The mean is " + statTest.getMean());
```

Our output is as follows:

The mean is 16.19

Next, we will cover the related topic: median.

Calculating the median

The mean can be misleading if the dataset contains a large number of outlying values or is otherwise skewed. When this happens, the mode and median can be useful. The term **median** is the value in the middle of a range of values. For an odd number of values, this is easy to compute. For an even number of values, the median is calculated as the average of the middle two values.

Using simple Java techniques to find median

In our first example, we will use a basic Java approach to calculate the median. For these examples, we have modified our `testData` array slightly:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2, 12.5,  
17.8, 16.5};
```

First, we use the `Arrays` class to sort our data because finding the median is simplified when the data is in numeric order:

```
Arrays.sort(testData);
```

We then handle three possibilities:

- Our list is empty
- Our list has an even number of values
- Our list has an odd number of values

The following code could be shortened, but we have been explicit to help clarify the process. If our list has an even number of values, we divide the length of the list by 2. The first variable, `mid1`, will hold the first of two middle values. The second variable, `mid2`, will hold the second middle value. The average of these two numbers is our median value. The process for finding the median index of a list with an odd number of values is simpler and requires only that we divide the length by 2 and add 1:

```
if(testData.length==0){      // Empty list  
    out.println("No median. Length is 0");  
}else if(testData.length%2==0){ // Even number of elements  
    double mid1 = testData[(testData.length/2)-1];  
    double mid2 = testData[testData.length/2];  
    double med = (mid1 + mid2)/2;  
    out.println("The median is " + med);
```

```
 }else{ // Odd number of elements  
    double mid = testData[(testData.length/2)+1];  
    out.println("The median is " + mid);  
}
```

Using the preceding array, which contains an even number of values, our output is:

The median is 16.35

To test our code for an odd number of elements, we will add the double 12.5 to the end of the array. Our new output is as follows:

The median is 16.5

Using Apache Commons to find the median

We can also calculate the median using the Apache Commons `DescriptiveStatistics` class demonstrated in the *Calculating the mean* section. We will continue using the `testData` array with the following values:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2, 12.5,  
17.8, 16.5, 12.5};
```

Our code is very similar to what we used to calculate the mean. We simply create our `DescriptiveStatistics` object and call the `getPercentile` method, which returns an estimate of the value stored at the percentile specified in its argument. To find the median, we use the value of 50:

```
DescriptiveStatistics statTest =  
    new SynchronizedDescriptiveStatistics();  
for(double num : testData){  
    statTest.addValue(num);  
}  
out.println("The median is " + statTest.getPercentile(50));
```

Our output is as follows:

The median is 16.2

Calculating the mode

The term **mode** is used for the most frequently occurring value in a dataset. This can be thought of as the most popular result, or the highest bar in a histogram. It can be a useful piece of information when conducting statistical analysis but it can be more complicated to calculate than it first appears. To begin, we will demonstrate a simple Java technique using the following `testData` array:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2, 12.5,  
17.8, 16.5, 12.5};
```

We start off by initializing variables to hold the mode, the number of times the mode appears in the list, and a `tempCnt` variable. The `mode` and `modeCount` variables are used to hold the mode value and the number of times this value occurs in the list respectively. The variable `tempCnt` is used to count the number of times an element occurs in the list:

```
int modeCount = 0;  
double mode = 0;  
int tempCnt = 0;
```

We then use nested for loops to compare each value of the array to the other values within the array. When we find matching values, we increment our `tempCnt`. After comparing each value, we test to see whether `tempCnt` is greater than `modeCount`, and if so, we change our `modeCount` and `mode` to reflect the new values:

```
for (double testValue : testData) {  
    tempCnt = 0;  
    for (double value : testData) {  
        if (testValue == value) {  
            tempCnt++;  
        }  
    }  
  
    if (tempCnt > modeCount) {  
        modeCount = tempCnt;  
        mode = testValue;  
    }  
}  
out.println("Mode" + mode + " appears " + modeCount + " times.");
```

Using this example, our output is as follows:

The mode is 12.5 and appears 3 times.

While our preceding example seems straightforward, it poses potential problems. Modify the `testData` array as shown here, where the last entry is changed to 11.2:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2, 12.5,  
17.8, 16.5, 11.2};
```

When we execute our code this time, our output is as follows:

The mode is 12.5 and appears 2 times.

The problem is that our `testData` array now contains two values that appear two times each, 12.5 and 11.2. This is known as a multimodal set of data. We can address this through basic Java code and through third-party libraries, as we will show in a moment.

However, first we will show two approaches using simple Java. The first approach will use two `ArrayList` instances and the second will use an `ArrayList` and a `HashMap` instance.

Using ArrayLists to find multiple modes

In the first approach, we modify the code used in the last example to use an `ArrayList` class. We will create two `ArrayLists`, one to hold the unique numbers within the dataset and one to hold the count of each number. We also need a `tempMode` variable, which we use next:

```
ArrayList<Integer> modeCount = new ArrayList<Integer>();  
ArrayList<Double> mode = new ArrayList<Double>();  
int tempMode = 0;
```

Next, we will loop through the array and test for each value in our mode list. If the value is not found in the list, we add it to `mode` and set the same position in `modeCount` to 1. If the value is found, we increment the same position in `modeCount` by 1:

```
for (double testValue : testData) {  
    int loc = mode.indexOf(testValue);  
    if(loc == -1){  
        mode.add(testValue);  
        modeCount.add(1);  
    } else {  
        modeCount.set(loc, modeCount.get(loc) + 1);  
    }  
}
```

```

    }else{
        modeCount.set(loc, modeCount.get(loc)+1);
    }
}

```

Next, we loop through our `modeCount` list to find the largest value. This represents the mode, or the frequency of the most common value in the dataset. This allows us to select multiple modes:

```

for(int cnt = 0; cnt < modeCount.size(); cnt++) {
    if (tempMode < modeCount.get(cnt)) {
        tempMode = modeCount.get(cnt);
    }
}

```

Finally, we loop through our `modeCount` array again and print out any elements in `mode` that correspond to elements in `modeCount` containing the largest value, or mode:

```

for(int cnt = 0; cnt < modeCount.size(); cnt++) {
    if (tempMode == modeCount.get(cnt)) {
        out.println(mode.get(cnt) + " is a mode and appears " +
                    modeCount.get(cnt) + " times.");
    }
}

```

When our code is executed, our output reflects our multimodal dataset:

```

12.5 is a mode and appears 2 times.
11.2 is a mode and appears 2 times.

```

Using a HashMap to find multiple modes

The second approach uses `HashMap`. First, we create `ArrayList` to hold possible modes, as in the previous example. We also create our `HashMap` and a variable to hold the mode:

```

ArrayList<Double> modes = new ArrayList<Double>();
HashMap<Double, Integer> modeMap = new HashMap<Double, Integer>();
int maxMode = 0;

```

Next, we loop through our `testData` array and count the number of occurrences of each value in the array. We then add the count of each value and the value itself to the `HashMap`. If the count for the value is larger than our `maxMode` variable, we set

`maxMode` to our new largest number:

```
for (double value : testData) {  
    int modeCnt = 0;  
    if (modeMap.containsKey(value)) {  
        modeCnt = modeMap.get(value) + 1;  
    } else {  
        modeCnt = 1;  
    }  
    modeMap.put(value, modeCnt);  
    if (modeCnt > maxMode) {  
        maxMode = modeCnt;  
    }  
}
```

Finally, we loop through our `HashMap` and retrieve our modes, or all values with a count equal to our `maxMode`:

```
for (Map.Entry<Double, Integer> multiModes : modeMap.entrySet()) {  
    if (multiModes.getValue() == maxMode) {  
        modes.add(multiModes.getKey());  
    }  
}  
for (double mode : modes) {  
    out.println(mode + " is a mode and appears " + maxMode + " times.");  
}
```

When we execute our code, we get the same output as in the previous example:

```
12.5 is a mode and appears 2 times.  
11.2 is a mode and appears 2 times.
```

Using Apache Commons to find multiple modes

Another option uses the Apache Commons `StatUtils` class. This class contains several methods for statistical analysis, including multiple methods for the mean, but we will only examine the mode here. The method is named `mode` and takes an array of doubles as its parameter. It returns an array of doubles containing all modes of the dataset:

```
double[] modes = StatUtils.mode(testData);  
for (double mode : modes) {  
    out.println(mode + " is a mode.");  
}
```

One disadvantage is that we are not able to count the number of times our mode

appears within this method. We simply know what the mode is, not how many times it appears. When we execute our code, we get a similar output to our previous example:

```
12.5 is a mode.  
11.2 is a mode.
```

Standard deviation

Standard deviation is a measurement of how values are spread around the mean. A high deviation means that there is a wide spread, whereas a low deviation means that the values are more tightly grouped around the mean. This measurement can be misleading if there is not a single focus point or there are numerous outliers.

We begin by showing a simple example using basic Java techniques. We are using our testData array from previous examples, duplicated here:

```
double[] testData = {12.5, 18.3, 11.2, 19.0, 22.1, 14.3, 16.2, 12.5,  
17.8, 16.5, 11.2};
```

Before we can calculate the standard deviation, we need to find the average. We could use any of our techniques listed in the *Calculating the mean* section, but we will add up our values and divide by the length of testData for simplicity's sake:

```
int sum = 0;  
for(double value : testData){  
    sum += value;  
}  
double mean = sum/testData.length;
```

Next, we create a variable, sdSum, to help us calculate the standard deviation. As we loop through our array, we subtract the mean from each data value, square that value, and add it to sdSum. Finally, we divide sdSum by the length of the array and square that result:

```
int sdSum = 0;  
for (double value : testData) {  
    sdSum += Math.pow((value - mean), 2);  
}  
out.println("The standard deviation is " +  
Math.sqrt( sdSum / ( testData.length ) ));
```

Our output is our standard deviation:

```
The standard deviation is 3.3166247903554
```

Our next technique uses Google Guava's Stats class to calculate the standard

deviation. We start by creating a `Stats` object with our `testData`. We then call the `populationStandardDeviation` method:

```
Stats testStats = Stats.of(testData);
double sd = testStats.populationStandardDeviation();
out.println("The standard deviation is " + sd);
```

The output is as follows:

The standard deviation is 3.3943803826056653

This example calculates the standard deviation of an entire population. Sometimes it is preferable to calculate the standard deviation of a sample subset of a population, to correct possible bias. To accomplish this, we use essentially the same code as before but replace the `populationStandardDeviation` method with `sampleStandardDeviation`:

```
Stats testStats = Stats.of(testData);
double sd = testStats.sampleStandardDeviation();
out.println("The standard deviation is " + sd);
```

In this case, our output is:

The sample standard deviation is 3.560056179332006

Our next example uses the Apache Commons `DescriptiveStatistics` class, which we used to calculate the mean and median in previous examples. Remember, this technique has the advantage of being thread safe and synchronized. After we create a `SynchronizedDescriptiveStatistics` object, we add each value from the array.

We then call the `getStandardDeviation` method.

```
DescriptiveStatistics statTest =
    new SynchronizedDescriptiveStatistics();
for(double num : testData){
    statTest.addValue(num);
}
out.println("The standard deviation is " +
statTest.getStandardDeviation());
```

Notice the output matches our output from our previous example. The `getStandardDeviation` method by default returns the standard deviation adjusted for a sample:

```
The standard deviation is 3.5600561793320065
```

We can, however, continue using Apache Commons to calculate the standard deviation in either form. The `StandardDeviation` class allows you to calculate the population standard deviation or subset standard deviation. To demonstrate the differences, replace the previous code example with the following:

```
StandardDeviation sdSubset = new StandardDeviation(false);  
out.println("The population standard deviation is " +  
sdSubset.evaluate(testData));
```

```
StandardDeviation sdPopulation = new StandardDeviation(true);  
out.println("The sample standard deviation is " +  
sdPopulation.evaluate(testData));
```

On the first line, we created a new `StandardDeviation` object and set our constructor's parameter to `false`, which will produce the standard deviation of a population. The second section uses a value of `true`, which produces the standard deviation of a sample. In our example, we used the same test dataset. This means we were first treating it as though it were a subset of a population of data. In our second example we assumed that our dataset was the entire population of data. In reality, you would might not use the same set of data with each of those methods. The output is as follows:

```
The population standard deviation is 3.3943803826056653  
The sample standard deviation is 3.560056179332006
```

The preferred option will depend upon your sample and particular analyzation needs.

Sample size determination

Sample size determination involves identifying the quantity of data required to conduct accurate statistical analysis. When working with large datasets it is not always necessary to use the entire set. We use sample size determination to ensure we choose a sample small enough to manipulate and analyze easily, but large enough to represent our population of data accurately.

It is not uncommon to use a subset of data to train a model and another subset is used to test the model. This can be helpful for verifying accuracy and reliability of data. Some common consequences for a poorly determined sample size include false-positive results, false-negative results, identifying statistical significance where none exists, or suggesting a lack of significance where it is actually present. Many tools exist online for determining appropriate sample sizes, each with varying levels of complexity. One simple example is available at

<https://www.surveymonkey.com/mp/sample-size-calculator/>.

Hypothesis testing

Hypothesis testing is used to test whether certain assumptions, or premises, about a dataset could not happen by chance. If this is the case, then the results of the test are considered to be statistically significant.

Performing hypothesis testing is not a simple task. There are many different pitfalls to avoid such as the placebo effect or the observer effect. In the former, a participant will attain a result that they think is expected. In the observer effect, also called the **Hawthorne effect**, the results are skewed because the participants know they are being watched. Due to the complex nature of human behavior analysis, some types of statistical analysis are particularly subject to skewing or corruption.

The specific methods for performing hypothesis testing are outside the scope of this book and require a solid background in statistical processes and best practices.

Apache Commons provides a package,

`org.apache.commons.math3.stat.inference`, with tools for performing hypothesis testing. This includes tools to perform a student's T-test, chi square, and calculating p values.

Regression analysis

Regression analysis is useful for determining trends in data. It indicates the relationship between dependent and independent variables. The independent variables determine the value of a dependent variable. Each independent variable can have either a strong or a weak effect on the value of the dependent variable. Linear regression uses a line in a scatterplot to show the trend. Non-linear regression uses some sort of curve to depict the relationships.

For example, there is a relationship between blood pressure and various factors such as age, salt intake, and **Body Mass Index (BMI)**. The blood pressure can be treated as the dependent variable and the other factors as independent variables. Given a dataset containing these factors for a group of individuals we can perform regression analysis to see trends.

There are several types of regression analysis supported by Java. We will be examining simple linear regression and multiple linear regression. Both approaches take a dataset and derive a linear equation that best fits the data. Simple linear regression uses a single dependent and a single independent variable. Multiple linear regression uses multiple dependent variables.

There are several APIs that support simple linear regression including:

- **Apache Commons** - <http://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/index.html>
- **Weka** -
<http://weka.sourceforge.net/doc.dev/weka/core/matrix/LinearRegression.html>
- **JFree** -
<http://www.jfree.org/jfreechart/api/javadoc/org/jfree/data/statistics/Regression.html>
- **Michael Thomas Flanagan's Java Scientific Library** -
<http://www.ee.ucl.ac.uk/~mflanaga/java/Regression.html>

Nonlinear Java support can be found at:

- **odinsbane/least-squares-in-java** - <https://github.com/odinsbane/least-squares-in-java>
- **NonLinearLeastSquares (Parallel Java Library Documentation)** -
<https://www.cs.rit.edu/~ark/pj/doc/edu/rit/numeric/NonLinearLeastSquares.html>

There are several statistics that evaluate the effectiveness of an analysis. We will focus on basic statistics.

Residuals are the difference between the actual data values and the predicted values. The **Residual Sum of Squares (RSS)** is the sum of the squares of residuals. Essentially it measures the discrepancy between the data and a regression model. A small RSS indicates the model closely matches the data. RSS is also known as the **Sum of Squared Residuals (SSR)** or the **Sum of Squared Errors (SSE)** of prediction.

The **Mean Square Error (MSE)** is the sum of squared residuals divided by the degrees of freedom. The number of degrees of freedom is the number of independent observations (N) minus the number of estimates of population parameters. For simple linear regression this $N - 2$ because there are two parameters. For multiple linear regression it depends on the number of independent variables used.

A small MSE also indicates that the model fits the dataset well. You will see both of these statistics used when discussing linear regression models.

The correlation coefficient measures the association between two variables of a regression model. The correlation coefficient ranges from -1 to $+1$. A value of $+1$ means that two variables are perfectly related. When one increases, so does the other. A correlation coefficient of -1 means that two variables are negatively related. When one increases, the other decreases. A value of 0 means there is no correlation between the variables. The coefficient is frequently designated as R . It will often be squared, thus ignoring the sign of the relation. The Pearson's product moment correlation coefficient is normally used.

Using simple linear regression

Simple linear regression uses a least squares approach where a line is computed that minimizes the sum of squared of the distances between the points and the line. Sometimes the line is calculated without using the Y intercept term. The regression line is an estimate. We can use the line's equation to predict other data points. This is useful when we want to predict future events based on past performance.

In the following example we use the Apache Commons SimpleRegression class with the Belgium population dataset used in [Chapter 4, Data Visualization](#). The data is duplicated here for your convenience:

Decade	Population
1950	8639369
1960	9118700
1970	9637800
1980	9846800
1990	9969310
2000	10263618

While the application that we will demonstrate is a JavaFX application, we will focus on the linear regression aspects of the application. We used a JavaFX program to generate a chart to show the regression results.

The body of the `start` method follows. The input data is stored in a two-dimension array as shown here:

```
double[][] input = {{1950, 8639369}, {1960, 9118700},  
    {1970, 9637800}, {1980, 9846800}, {1990, 9969310},  
    {2000, 10263618}};
```

An instance of the `SimpleRegression` class is created and the data is added using the `addData` method:

```
SimpleRegression regression = new SimpleRegression();
```

```
regression.addData(input);
```

We will use the model to predict behavior for several years as declared in the array that follows:

```
double[] predictionYears = {1950, 1960, 1970, 1980, 1990, 2000,  
    2010, 2020, 2030, 2040};
```

We will also format our output using the following `NumberFormat` instances. One is used for the year where the `setGroupingUsed` method with a parameter of false suppresses commas.

```
NumberFormat yearFormat = NumberFormat.getNumberInstance();  
yearFormat.setMaximumFractionDigits(0);  
yearFormat.setGroupingUsed(false);  
NumberFormat populationFormat = NumberFormat.getNumberInstance();  
populationFormat.setMaximumFractionDigits(0);
```

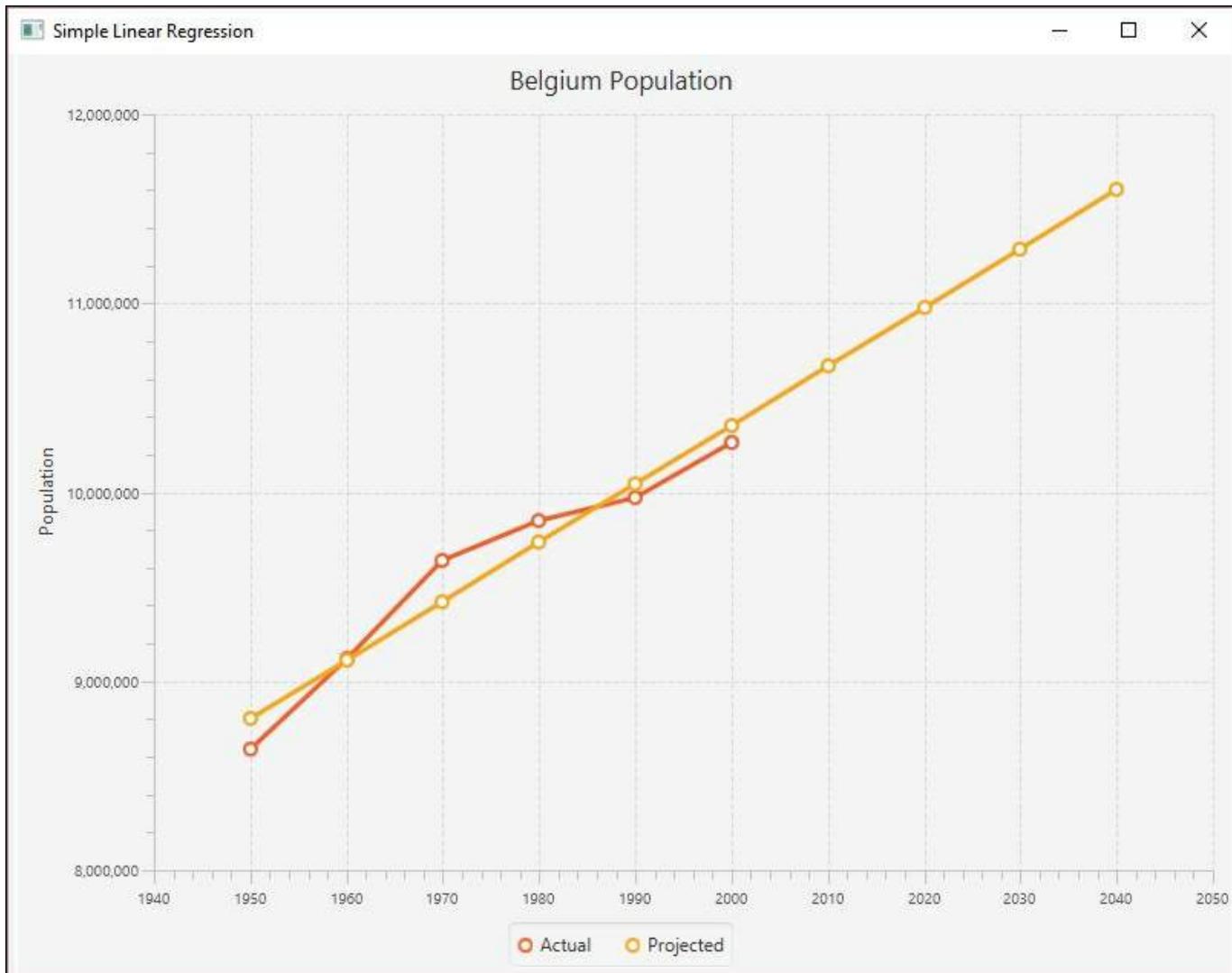
The `SimpleRegression` class possesses a `predict` method that is passed a value, a year in this case, and returns the estimated population. We use this method in a loop and call the method for each year:

```
for (int i = 0; i < predictionYears.length; i++) {  
    out.println(nf.format(predictionYears[i]) + "-"  
        + nf.format(regression.predict(predictionYears[i])));  
}
```

When the program is executed, we get the following output:

```
1950-8,801,975  
1960-9,112,892  
1970-9,423,808  
1980-9,734,724  
1990-10,045,641  
2000-10,356,557  
2010-10,667,474  
2020-10,978,390  
2030-11,289,307  
2040-11,600,223
```

To see the results graphically, we generated the following index chart. The line matches the actual population values fairly well and shows the projected populations in the future.



Simple Linear Regression

The `SimpleRegression` class supports a number of methods that provide additional information about the regression. These methods are summarized next:

Method	Meaning
<code>getR</code>	Returns Pearson's product moment correlation coefficient
<code>getRSquare</code>	Returns the coefficient of determination (R-square)
<code>getMeanSquareError</code>	Returns the MSE

getSlope	The slope of the line
getIntercept	The intercept

We used the helper method, `displayAttribute`, to display various attribute values as shown here:

```
displayAttribute(String attribute, double value) {
    NumberFormat numberFormat = NumberFormat.getNumberInstance();
    numberFormat.setMaximumFractionDigits(2);
    out.println(attribute + ": " + numberFormat.format(value));
}
```

We called the previous methods for our model as shown next:

```
displayAttribute("Slope", regression.getSlope());
displayAttribute("Intercept", regression.getIntercept());
displayAttribute("MeanSquareError",
    regression.getMeanSquareError());
displayAttribute("R", + regression.getR());
displayAttribute("RSquare", regression.getRSquare());
```

The output follows:

```
Slope: 31,091.64
Intercept: -51,826,728.48
MeanSquareError: 24,823,028,973.4
R: 0.97
RSquare: 0.94
```

As you can see, the model fits the data nicely.

Using multiple regression

Our intent is not to provide a detailed explanation of multiple linear regression as that would be beyond the scope of this section. A more thorough treatment can be found at http://www.biddle.com/documents/bcg_comp_chapter4.pdf. Instead, we will explain the basics of the approach and show how we can use Java to perform multiple regression.

Multiple regression works with data where multiple independent variables exist. This happens quite often. Consider that the fuel efficiency of a car can be dependent on the octane level of the gas being used, the size of the engine, the average cruising speed, and the ambient temperature. All of these factors can influence the fuel efficiency, some to a larger degree than others.

The independent variable is normally represented as Y where there are multiple dependent variables represented using different X s. A simplified equation for a regression using three dependent variables follows where each variable has a coefficient. The first term is the intercept. These coefficients are not intended to represent real values but are only used for illustrative purposes.

$$Y = 11 + 0.75 X1 + 0.25 X2 - 2 X3$$

The intercept and coefficients are generated using a multiple regression model based on sample data. Once we have these values, we can create an equation to predict other values.

We will use the Apache Commons `OLSMultipleLinearRegression` class to perform multiple regression using cigarette data. The data has been adapted from <http://www.amstat.org/publications/jse/v2n1/datasets.mcintyre.html>. The data consists of 25 entries for different brands of cigarettes with the following information:

- Brand name
- Tar content (mg)
- Nicotine content (mg)
- Weight (g)
- Carbon monoxide content (mg)

The data has been stored in a file called `data.csv` as shown in the following partial

listing of its contents where the columns values match the order of the previous list:

```
Alpine,14.1,.86,.9853,13.6  
Benson&Hedges,16.0,1.06,1.0938,16.6  
BullDurham,29.8,2.03,1.1650,23.5  
CamelLights,8.0,.67,.9280,10.2  
...
```

The following is a scatter plot chart showing the relationship of the data:



Multiple Regression Scatter plot

We will use a JavaFX program to create the scatter plot and to perform the analysis. We start with the `MainApp` class as shown next. In this example we will focus on the multiple regression code and we do not include the JavaFX code used to create the scatter plot. The complete program can be downloaded from <http://www.packtpub.com/support>.

The data is held in a one-dimensional array and a `NumberFormat` instance will be

used to format the values. The array size reflects the 25 entries and the 4 values per entry. We will not be using the brand name in this example.

```
public class MainApp extends Application {  
    private final double[] data = new double[100];  
    private final NumberFormat numberFormat =  
        NumberFormat.getNumberInstance();  
    ...  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

The data is read into the array using a CSVReader instance as shown next:

```
int i = 0;  
try (CSVReader dataReader = new CSVReader(  
        new FileReader("data.csv"), ',', ',')) {  
    String[] nextLine;  
    while ((nextLine = dataReader.readNext()) != null) {  
        String brandName = nextLine[0];  
        double tarContent = Double.parseDouble(nextLine[1]);  
        double nicotineContent = Double.parseDouble(nextLine[2]);  
        double weight = Double.parseDouble(nextLine[3]);  
        double carbonMonoxideContent =  
            Double.parseDouble(nextLine[4]);  
        data[i++] = carbonMonoxideContent;  
        data[i++] = tarContent;  
        data[i++] = nicotineContent;  
        data[i++] = weight;  
        ...  
    }  
}
```

Apache Commons possesses two classes that perform multiple regression:

- OLSMultipleLinearRegression- **Ordinary Least Square (OLS)** regression
- GLSMultipleLinearRegression- **Generalized Least Squared (GLS)** regression

When the latter technique is used, the correlation within elements of the model impacts the results addversely. We will use the

OLSMultipleLinearRegression class and start with its instantiation:

```
OLSMultipleLinearRegression ols =  
new OLSMultipleLinearRegression();
```

We will use the `newSampleData` method to initialize the model. This method needs the number of observations in the dataset and the number of independent variables. It may throw an `IllegalArgumentException` exception which needs to be handled.

```
int numberOfObservations = 25;
int numberOfIndependentVariables = 3;
try {
    ols.newSampleData(data, numberOfObservations,
                      numberOfIndependentVariables);
} catch (IllegalArgumentException e) {
    // Handle exceptions
}
```

Next, we set the number of digits that will follow the decimal point to two and invoke the `estimateRegressionParameters` method. This returns an array of values for our equation, which are then displayed:

```
numberFormat.setMaximumFractionDigits(2);
double[] parameters = ols.estimateRegressionParameters();
for (int i = 0; i < parameters.length; i++) {
    out.println("Parameter " + i +": " +
                numberFormat.format(parameters[i]));
}
```

When executed we will get the following output which gives us the parameters needed for our regression equation:

```
Parameter 0: 3.2
Parameter 1: 0.96
Parameter 2: -2.63
Parameter 3: -0.13
```

To predict a new dependent value based on a set of independent variables, the `getY` method is declared, as shown next. The `parameters` parameter contains the generated equation coefficients. The `arguments` parameter contains the value for the dependent variables. These are used to calculate the new dependent value which is returned:

```
public double getY(double[] parameters, double[] arguments) {
    double result = 0;
    for(int i=0; i<parameters.length; i++) {
        result += parameters[i] * arguments[i];
    }
    return result;
}
```

We can test this method by creating a series of independent values. Here we used the same values as used for the SalemUltra entry in the data file:

```
double arguments1[] = {1, 4.5, 0.42, 0.9106};  
out.println("X: " + 4.9 + " y: " +  
    numberFormat.format(getY(parameters, arguments1)));
```

This will give us the following values:

x: 4.9 y: 6.31

The return value of 6.31 is different from the actual value of 4.9. However, using the values for VirginiaSlims:

```
double arguments2[] = {1, 15.2, 1.02, 0.9496};  
out.println("X: " + 13.9 + " y: " +  
    numberFormat.format(getY(parameters, arguments2)));
```

We get the following result:

x: 13.9 y: 15.03

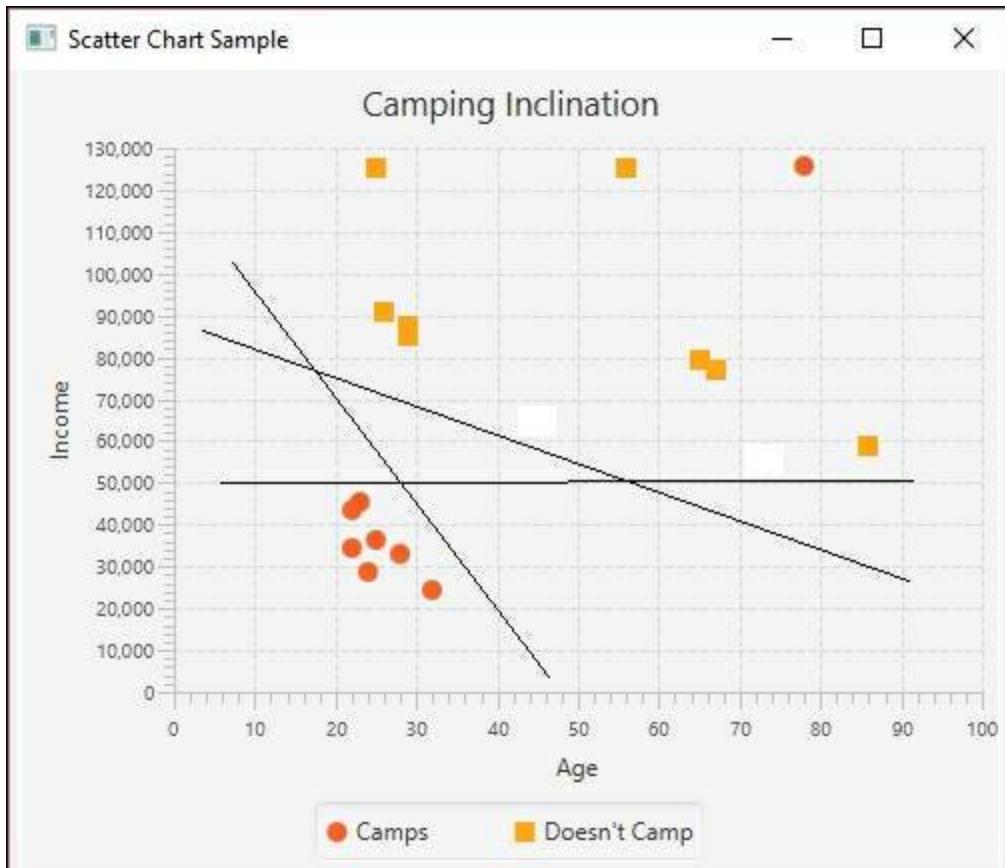
This is close to the actual value of 13.9. Next, we use a different set of values than found in the dataset:

```
double arguments3[] = {1, 12.2, 1.65, 0.86};  
out.println("X: " + 9.9 + " y: " +  
    numberFormat.format(getY(parameters, arguments3)));
```

The result follows:

x: 9.9 y: 10.49

The values differ but are still close. The following figure shows the predicted data in relation to the original data:



Multiple regression projected

The `OLSMultipleLinearRegression` class also possesses several methods to evaluate how well the models fits the data. However, due to the complex nature of multiple regression, we have not discussed them here.

Summary

In this chapter, we provided a brief introduction to the basic statistical analysis techniques you may encounter in data science applications. We started with simple techniques to calculate the mean, median, and mode of a set of numerical data. Both standard Java and third-party Java APIs were used to show how these attributes are calculated. While these techniques are relatively simple, there are some issues that need to be considered when calculating them.

Next, we examined linear regression. This technique is more predictive in nature and attempts to calculate other values in the future or past based on a sample dataset. We examine both simple linear regression and multiple regression and used Apache Commons classes to perform the regression and JavaFX to draw graphs.

Simple linear regression uses a single independent variable to predict a dependent variable. Multiple regression uses more than one independent variable. Both of these techniques have statistical attributes used to assess how well they match the data.

We demonstrated the use of the Apache Commons `OLSMultipleLinearRegression` class to perform multiple regression using cigarette data. We were able to use multiple attributes to create an equation that predicts the carbon monoxide output.

With these statistical techniques behind us, we can now examine basic machine learning techniques in the next chapter. This will include a detailed discussion of multilayer perceptrons and various other neural networks.

Chapter 6. Machine Learning

Machine learning is a broad topic with many different supporting algorithms. It is generally concerned with developing techniques that allow applications to learn without having to be explicitly programmed to solve a problem. Typically, a model is built to solve a class of problems and then is trained using sample data from the problem domain. In this chapter, we will address a few of the more common problems and models used in data science.

Many of these techniques use training data to teach a model. The data consists of various representative elements of the problem space. Once the model has been trained, it is tested and evaluated using testing data. The model is then used with input data to make predictions.

For example, the purchases made by customers of a store can be used to train a model. Subsequently, predictions can be made about customers with similar characteristics. Due to the ability to predict customer behavior, it is possible to offer special deals or services to entice customers to return or facilitate their visit.

There are several ways of classifying machine learning techniques. One approach is to classify them according to the learning style:

- **Supervised learning:** With supervised learning, the model is trained with data that matches input characteristic values to the correct output values
- **Unsupervised learning:** In unsupervised learning, the data does not contain results, but the model is expected to determine relationships on its own.
- **Semi-supervised:** This technique uses a small amount of labeled data containing the correct response with a larger amount of unlabeled data. The combination can lead to improved results.
- **Reinforcement learning:** This is similar to supervised learning but a reward is provided for good results.
- **Deep learning:** This approach models high-level abstractions using a graph that contains multiple processing levels.

In this chapter, we will only be able to touch on a few of these techniques. Specifically, we will illustrate three techniques that use supervised learning:

- **Decision trees:** A tree is constructed using the features of the problem as internal nodes and the results as leaves

- **Support vector machines:** Generally used for classification by creating a hyperplane that separates the dataset and then making predictions
- **Bayesian networks:** Models used to depict probabilistic relationships between events within an environment

For unsupervised learning, we will show how **association rule learning** can be used to find relationships between elements of a dataset. However, we will not address unsupervised learning in this chapter.

We will discuss the elements of reinforcement learning and discuss a few specific variations of this technique. We will also provide links to resources for further exploration.

The discussion of deep learning is postponed to [Chapter 8, Deep Learning](#). This technique builds upon neural networks, which will be discussed in [Chapter 7, Neural Networks](#).

In this chapter, we will cover the following specific topics:

- Decision trees
- Support vector machines
- Bayesian networks
- Association rule learning
- Reinforcement learning

Supervised learning techniques

There are a large number of supervised machine learning algorithms available. We will examine three of them: decision trees, support vector machines, and Bayesian networks. They all use annotated datasets that contain attributes and a correct response. Typically, a training and a testing dataset is used.

We start with a discussion of decision trees.

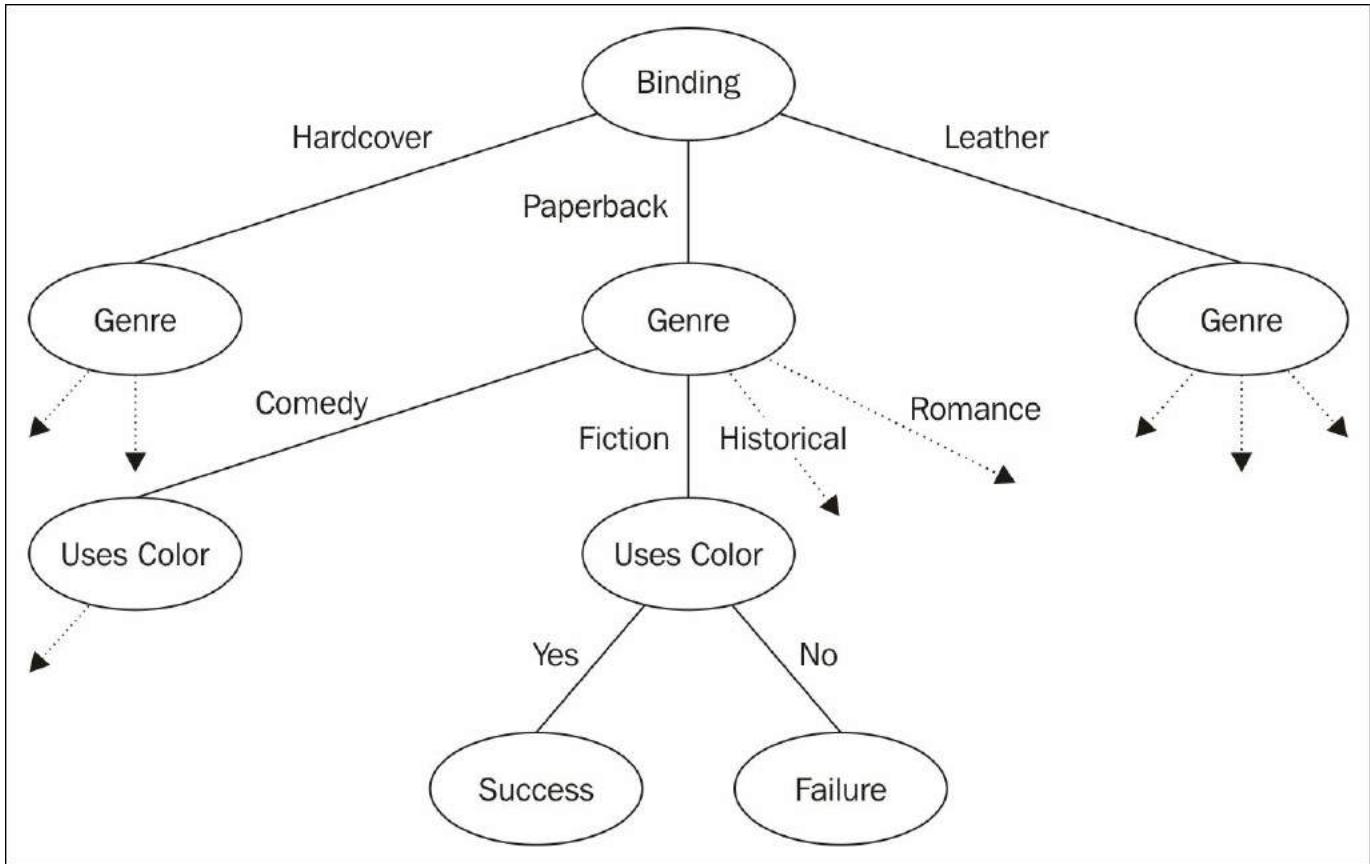
Decision trees

A machine learning **decision tree** is a model used to make predictions. It effectively maps certain observations to conclusions about a target. The term **tree** comes from the branches that reflect different states or values. The leaves of a tree represent results and the branches represent features that lead to the results. In data mining, a decision tree is a description of data used for classification. For example, we can use a decision tree to determine whether an individual is likely to buy an item based on certain attributes such as income level and postal code.

We want to create a decision tree that predicts results based on other variables. When the target variable takes on continuous values such as real numbers, the tree is called a **regression tree**.

A tree consists of internal nodes and leaves. Each internal node represents a feature of the model such as the number of years of education or whether a book is paperback or hardcover. The edges leading out of an internal node represent the values of these features. Each leaf is called a **class** and has an **associated probability distribution**.

For example, we will be using a dataset that deals with the success of a book based on its binding type, use of color, and genre. One possible decision tree based on this dataset follows:



Decision tree

Decision trees are useful and easy to understand. Preparing data for a model is straightforward even for large datasets.

Decision tree types

A tree can be *taught* by dividing an input dataset by the features. This is often done in a recursive fashion and is called **recursive partitioning** or **Top-Down Induction of Decision Trees (TDIDT)**. The recursion is bounded when node's values are all of the same type as the target or the recursion no longer adds value.

Classification and Regression Tree (CART) analysis refers to two different types of decision tree types:

- **Classification tree analysis:** The leaf corresponds to a target feature
- **Regression tree analysis:** The leaf possesses a real number representing a feature

During the process of analysis, multiple trees may be created. There are several techniques used to create trees. The techniques are called **ensemble methods**:

- **Bagging decision trees:** The data is resampled and frequently used to obtain a prediction based on consensus
- **Random forest classifier:** Used to improve the classification rate
- **Boosted trees:** This can be used for regression or classification problems
- **Rotation forest:** Uses a technique called **Principal Component Analysis (PCA)**

With a given set of data, it is possible that more than one tree models the data. For example, the root of a tree may specify whether a bank has an ATM machine and a subsequent internal node may specify the number of tellers. However, the tree could be created where the number of tellers is at the root and the existence of an ATM is an internal node. The difference in the structure of the tree can determine how efficient the tree is.

There are a number of ways of determining the order of the nodes of a tree. One technique is to select an attribute that provides the most information gain; that is, choose an attribute that better helps narrow down the possible decisions fastest.

Decision tree libraries

There are several Java libraries that support decision trees:

- **Weka:** <http://www.cs.waikato.ac.nz/ml/weka/>
- **Apache Spark:** <https://spark.apache.org/docs/1.2.0/mllib-decision-tree.html>
- **JBoost:** <http://jboost.sourceforge.net>
- **MAchine Learning for LanguagE Toolkit (MALLET):** <http://mallet.cs.umass.edu>

We will use **Waikato Environment for Knowledge Analysis (Weka)** to demonstrate how to create a decision tree in Java. Weka is a tool that has a GUI interface that permits analysis of data. It can also be invoked from the command line or through a Java API, which we will use.

As a tree is being built, a variable is selected to split the tree. There are several techniques used to select a variable. The one we use is determined by how much information is gained by choosing a variable. Specifically, we will use the **C4.5 algorithm** as supported by Weka's `J48` class.

Weka uses an `.arff` file to hold a dataset. This file is human readable and consists of two sections. The first is a header section; it describes the data in the file. This section uses the ampersand to specify the relation and attributes of the data. The second section is the data section; it consists of a comma-delimited set of data.

Using a decision tree with a book dataset

For this example, we will use a file called `books.arff`. It is shown next and uses four features called attributes. The features specify how a book is bound, whether it uses multiple colors, its genre, and a result indicating whether the book was purchased or not. The header section is shown here:

```
@RELATION book_purchases
@ATTRIBUTE Binding {Hardcover, Paperback, Leather}
@ATTRIBUTE Multicolor {yes, no}
@ATTRIBUTE Genre {fiction, comedy, romance, historical}
@ATTRIBUTE Result {Success, Failure}
```

The data section follows and consists of 13 book entries:

```
@DATA
Hardcover,yes,fiction,Success
Hardcover,no,comedy,Failure
Hardcover,yes,comedy,Success
Leather,no,comedy,Success
Leather,yes,historical,Success
Paperback,yes,fiction,Failure
Paperback,yes,romance,Failure
Leather,yes,comedy,Failure
Paperback,no,fiction,Failure
Paperback,yes,historical,Failure
Hardcover,yes,historical,Success
Paperback,yes,comedy,Success
Hardcover,yes,comedy,Success
```

We will use the `BookDecisionTree` class as defined next to process this file. It uses one constructor and three methods:

- `BookDecisionTree`: Reads in the trainer data and creates an `Instance` object used to process the data
- `main`: Drives the application
- `performTraining`: Trains the model using the dataset
- `getTestInstance`: Creates a test case

The `Instances` class holds elements representing the individual dataset elements:

```
public class BookDecisionTree {  
    private Instances trainingData;  
  
    public static void main(String[] args) {  
        ...  
    }  
  
    public BookDecisionTree(String fileName) {  
        ...  
    }  
  
    private J48 performTraining() {  
        ...  
    }  
  
    private Instance getTestInstance()  
    ...  
}  
}
```

The constructor opens a file and uses the `BufferedReader` instance to create an instance of the `Instances` class. Each element of the dataset will be either a feature or a result. The `setClassIndex` method specifies the index of the result class. In this case, it is the last index of the dataset and corresponds to success or failure:

```
public BookDecisionTree(String fileName) {  
    try {  
        BufferedReader reader = new BufferedReader(  
            new FileReader(fileName));  
        trainingData = new Instances(reader);  
        trainingData.setClassIndex(  
            trainingData.numAttributes() - 1);  
    } catch (IOException ex) {  
        // Handle exceptions  
    }  
}
```

We will use the `J48` class to generate a decision tree. This class uses the C4.5 decision tree algorithm for generating a pruned or unpruned tree. The `setOptions` method specifies that an unpruned tree be used. The `buildClassifier` method actually creates the classifier based on the dataset used:

```
private J48 performTraining() {  
    J48 j48 = new J48();
```

```

        String[] options = {"-U"};
        try {
            j48.setOptions(options);
            j48.buildClassifier(trainingData);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return j48;
    }
}

```

We will want to test the model, so we will create an object that implements the `Instance` interface for each test case. A `getTestInstance` helper method is passed three arguments representing the three features of a data element. The `DenseInstance` class is a class that implements the `Instance` interface. The values passed are assigned to the instance and the instance is returned:

```

private Instance getTestInstance(
    String binding, String multicolor, String genre) {
    Instance instance = new DenseInstance(3);
    instance.setDataset(trainingData);
    instance.setValue(trainingData.attribute(0), binding);
    instance.setValue(trainingData.attribute(1), multicolor);
    instance.setValue(trainingData.attribute(2), genre);
    return instance;
}

```

The `main` method uses all the previous methods to process and test our book dataset. First, a `BookDecisionTree` instance is created using the name of the book dataset file:

```

public static void main(String[] args) {
    try {
        BookDecisionTree decisionTree =
            new BookDecisionTree("books.arff");
        ...
    } catch (Exception ex) {
        // Handle exceptions
    }
}

```

Next, the `performTraining` method is invoked to train the model. We also display the tree:

```

J48 tree = decisionTree.performTraining();
System.out.println(tree.toString());

```

When executed, the following will be displayed:

```
J48 unpruned tree
-----
Binding = Hardcover: Success (5.0/1.0)
Binding = Paperback: Failure (5.0/1.0)
Binding = Leather: Success (3.0/1.0)
Number of Leaves : 3
Size of the tree : 4
```

Testing the book decision tree

We will test the model with two different test cases. Both use identical code to set up the instance. We use the `getTestInstance` method with test-case-specific values and then use the instance with `classifyInstance` to get results. To get something that is more readable, we generate a string, which is then displayed:

```
Instance testInstance = decisionTree.
    getTestInstance("Leather", "yes", "historical");
int result = (int) tree.classifyInstance(testInstance);
String results = decisionTree.trainingData.attribute(3).value(result);
System.out.println(
    "Test with: " + testInstance + " Result: " + results);

testInstance = decisionTree.
    getTestInstance("Paperback", "no", "historical");
result = (int) tree.classifyInstance(testInstance);
results = decisionTree.trainingData.attribute(3).value(result);
System.out.println(
    "Test with: " + testInstance + " Result: " + results);
```

The result of executing this code is as follows:

```
Test with: Leather,yes,historical Result: Success
Test with: Paperback,no,historical Result: Failure
```

This matches our expectations. This technique is based on the amount of information gain before and after an ordering decision has been made. This can be measured based on the entropy as calculated here:

```
Entropy = -portionPos * log2(portionPos) - portionNeg* log2(portionNeg)
```

In this example, `portionPos` is the portion of the data that is positive and `portionNeg` is the portion of the data that is negative. Based on the books file, we can calculate the entropy for the binding as shown in the following table. The

information gain is calculated by subtracting the entropy for binding from *1.0*:

Binding	Portion		Entropy
	Positive	Negative	
Hardcover	0.8	0.2	0.72
Leather	0.66	0.33	0.92
Paperback	0.2	0.8	0.72
Entropy for Binding			0.76
Information Gain			0.24

We can calculate the entropy for the use of color and genre in a similar manner. The information gain for color is *0.05*, and it is *0.15* for the genre. Thus, it makes more sense to use the binding type for the first level of the tree.

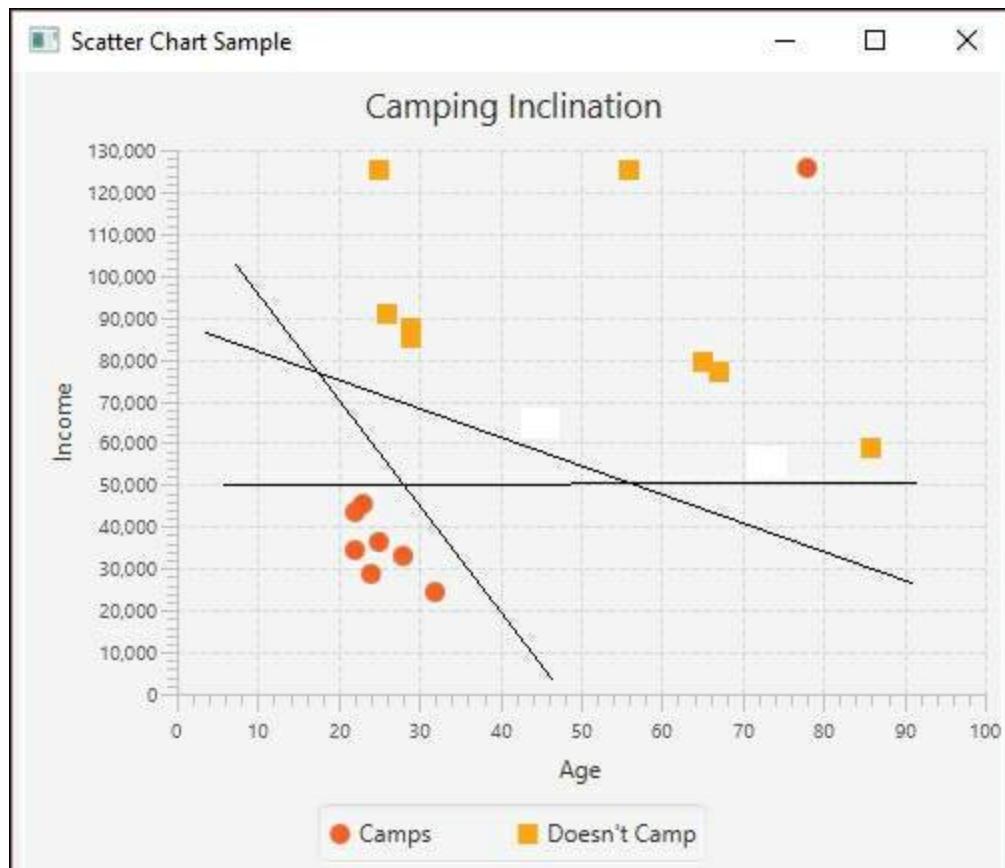
The resulting tree from the example consists of two levels, because the C4.5 algorithm determines that the remaining features do not provide any additional information gain.

Information gain can be problematic when a feature that has a large number of values is chosen, such as a customer's credit card number. Using this type of attribute will quickly narrow down the field, but it is too selective to be of much value.

Support vector machines

A **Support Vector Machine (SVM)** is a supervised machine learning algorithm used for both classification and regression problems. It is mostly used for classification problems. The approach creates a hyperplane to categorize the training data. A hyperplane can be envisioned as a geometric plane that separates two regions. In a two-dimensional space, it will be a line. In a three-dimensional space, it will be a two-dimensional plane. For higher dimensions, it is harder to conceptualize, but they do exist.

Consider the following figure depicting a distribution of two types of data points. The lines represent possible hyperplanes that separate these points. Part of the SVM process is to find the best hyperplane for the problem dataset. We will elaborate on this figure in the coding example.



Hyperplane example

Support vectors are data points that lie near the hyperplane. An SVM model uses the concept of a kernel to map input data to a higher order dimensional space to make the data more easily structured. A mapping function for doing this could lead to an infinite-dimensional space; that is, there could be an unbounded number of possible mappings.

However, what is known as the **kernel trick**, a kernel function is an approach that avoids this mapping and avoids possibly infeasible computations that might otherwise occur. SVMs support different types of kernels. A list of kernels can be found at <http://crsouza.com/2010/03/kernel-functions-for-machine-learning-applications/>. Choosing the right kernel depends on the problem. Commonly used kernels include:

- **Linear:** Uses a linear hyperplane
- **Polynomial:** Uses a polynomial equation for the hyperplane
- **Radial Basis Function (RBF):** Uses a non-linear hyperplane
- **Sigmoid:** The sigmoid kernel, also known as the **Hyperbolic Tangent kernel**, comes from the neural networks field and is equivalent to a two-layer perceptron neural network

These kernels support different algorithms for analyzing data.

SVMs are useful for higher dimensional spaces that humans have a harder time visualizing. In the previous figure, two attributes were used to predict a third. An SVM can be used when many more attributes are present. The SVM needs to be trained and this can take longer with larger datasets.

We will use the Weka class `SMO` to demonstrate SVM analysis. The class supports John Platt's sequential minimal optimization algorithm. More information about this algorithm is found at <https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization/>.

The `SMO` class supports the following kernels, which can be specified when using the class:

- **Puk:** The Pearson VII-function-based universal kernel
- **PolyKernel:** The polynomial kernel
- **RBFKernel:** The RBF kernel

The algorithm uses training data to create a classification model. Test data can then

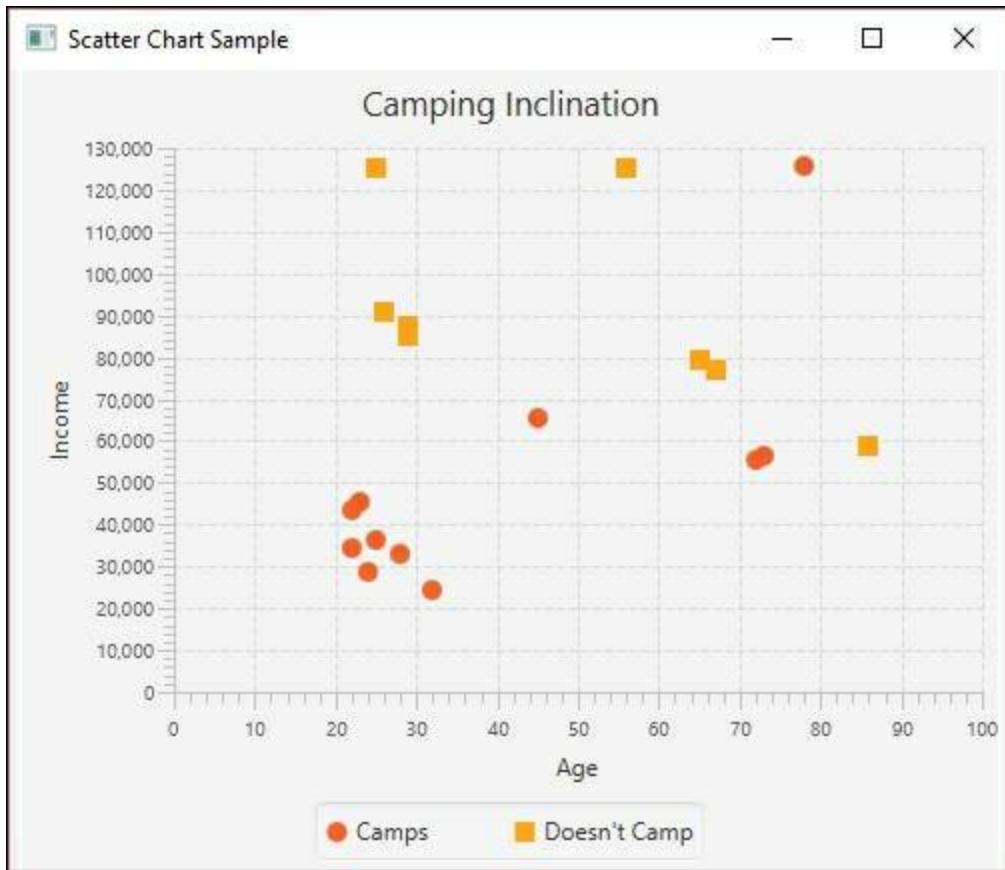
be used to evaluate the model. We can also evaluate individual data elements.

Using an SVM for camping data

For illustration purposes, we will be using a dataset consisting of age, income, and whether someone camps. We would like to be able to predict whether someone is inclined to camp based on their age and income. The data we use is stored in .arff format and is not based on a survey but has been created to explain the SVM process. The input data is found in the `camping.txt` file, as shown next. The file extension does not need to be .arff:

```
@relation camping
@attribute age numeric
@attribute income numeric
@attribute camps {1, 0}
@data
23,45600,1
45,65700,1
72,55600,1
24,28700,1
22,34200,1
28,32800,1
32,24600,1
25,36500,1
26,91000,0
29,85300,0
67,76800,0
86,58900,0
56,125300,0
25,125000,0
22,43600,1
78,125700,1
73,56500,1
29,87600,0
65,79300,0
```

The following shows how the data is distributed graphically. Notice the outlier found in the upper-right corner. The JavaFX code that produces this graph is found at <http://www.packtpub.com/support>:



Camping Graph

We will start by reading in the data and handling exceptions:

```
try {
    BufferedReader datafile;
    datafile = readDataFile("camping.txt");
    ...
} catch (Exception ex) {
    // Handle exceptions
}
```

The `readDataFile` method follows:

```
public BufferedReader readDataFile(String filename) {
    BufferedReader inputReader = null;
    try {
        inputReader = new BufferedReader(
            new FileReader(filename));
    } catch (FileNotFoundException ex) {
```

```

        // Handle exceptions
    }
    return inputReader;
}

```

The `Instances` class holds a series of data instances, where each instance is an age, income, and camping value. The `setClassIndex` method indicates which of the attributes is to be predicted. In this case, it is the `camps` attribute:

```

Instances data = new Instances(datafile);
data.setClassIndex(data.numAttributes() - 1);

```

To train the model, we will split the dataset into two sets. The first 14 instances are used to train the model and the last 5 instances are used to test the model. The second argument of the `Instances` constructor specifies the beginning index in the dataset, and the last argument specifies how many instances to include:

```

Instances trainingData = new Instances(data, 0, 14);
Instances testingData = new Instances(data, 14, 5);

```

An `Evaluation` class instance is created to evaluate the model. An instance of the `SMO` class is also created. The `SMO` class's `buildClassifier` method builds the classifier using the dataset:

```

Evaluation evaluation = new Evaluation(trainingData);
Classifier smo = new SMO();
smo.buildClassifier(data);

```

The `evaluateModel` method evaluates the model using the testing data. The results are then displayed:

```

evaluation.evaluateModel(smo, testingData);
System.out.println(evaluation.toSummaryString());

```

The output follows. Notice one incorrectly classified instance. This corresponds to the outlier mentioned earlier:

```

Correctly Classified Instances 4 80 %
Incorrectly Classified Instances 1 20 %
Kappa statistic 0.6154
Mean absolute error 0.2
Root mean squared error 0.4472
Relative absolute error 41.0256 %
Root relative squared error 91.0208 %
Coverage of cases (0.95 level) 80 %

```

```
Mean rel. region size (0.95 level) 50 %
Total Number of Instances 5
```

Testing individual instances

We can also test an individual instance using the `classifyInstance` method. In the following sequence, we create a new instance using the `DenseInstance` class. It is then populated using the attributes of the camping dataset:

```
Instance instance = new DenseInstance(3);
instance.setValue(data.attribute("age"), 78);
instance.setValue(data.attribute("income"), 125700);
instance.setValue(data.attribute("camps"), 1);
```

The instance needs to be associated with the dataset using the `setDataset` method:

```
instance.setDataset(data);
```

The `classifyInstance` method is then applied to the `smo` instance and the results are displayed:

```
System.out.println(smo.classifyInstance(instance));
```

When executed, we get the following output:

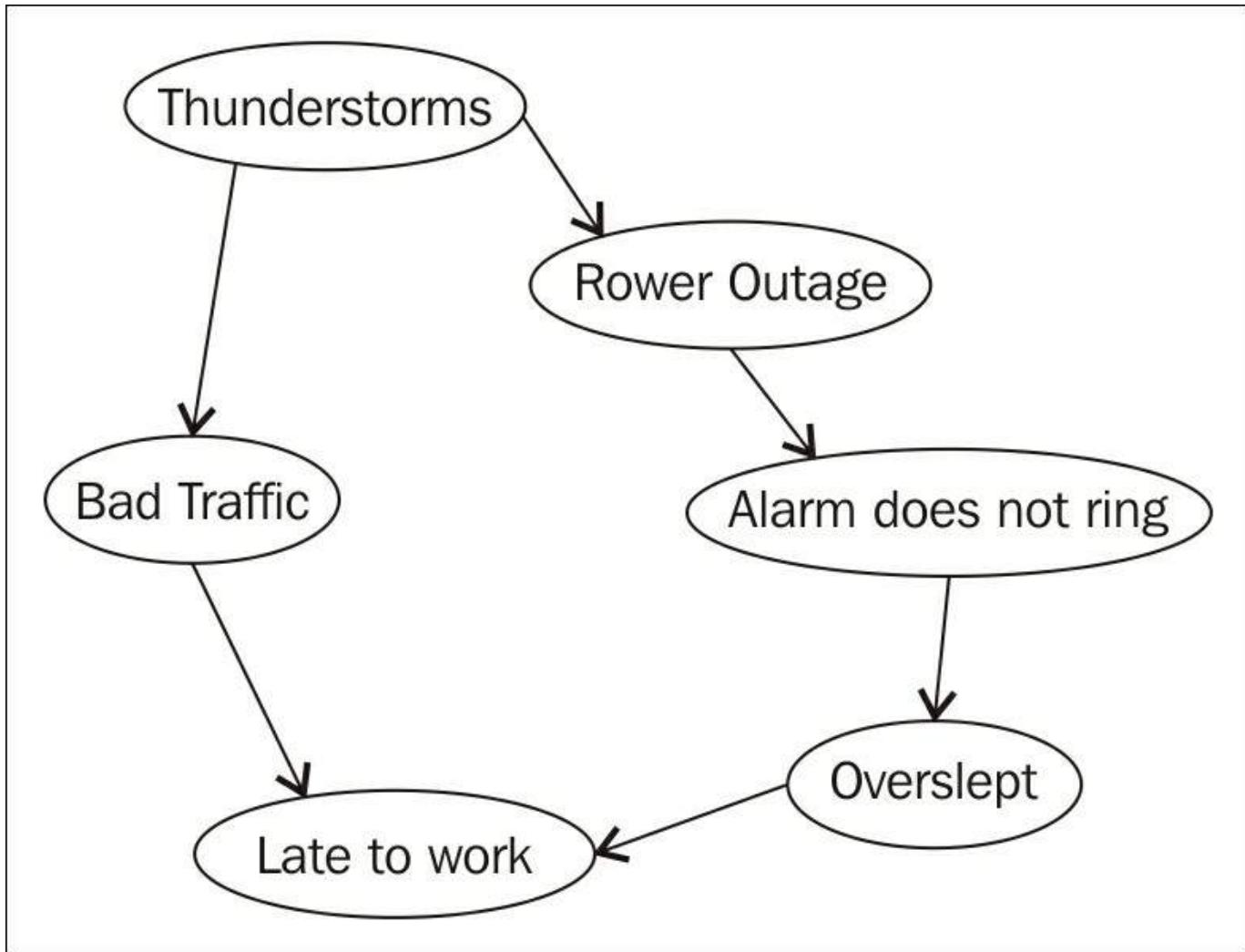
```
1.0
```

There are also alternate testing approaches. A common one is called **cross-validation folds**. This approach divides the dataset into *folds*, which are partitions of the dataset. Frequently, 10 partitions are created. Nine of the partitions are used for training and one for testing. This is repeated 10 times using a different partition of the dataset each time, and the average of the results is used. This technique is described at [https://weka.wikispaces.com/Generating+cross-validation+folders+\(Java+approach\).](https://weka.wikispaces.com/Generating+cross-validation+folders+(Java+approach).)

We will now examine the purpose and use of Bayesian networks.

Bayesian networks

Bayesian networks, also known as **Bayes nets** or **belief networks**, are models designed to reflect a particular world or environment by depicting the states of different attributes of the world and their statistical relationships. The models can be used to show a wide variety of real-world scenarios. In the following diagram, we model a system depicting the relationship between various factors and our likelihood of being late to work:



Bayesian network

Each circle on the diagram represents a node or part of the system, which can have various values and probabilities for each value. For example, **Power Outage** might

be true or false — either the power went out or it did not. The probability of the power going out affects the probability that your alarm will not ring, you might oversleep, and thus be late to work.

The nodes at the top of the diagram tend to imply a higher level of causality than those at the bottom. The higher nodes are called **parent nodes**, and they may have one or more child nodes. Bayesian networks only relate nodes that have a causal dependency and therefore allow more efficient computation of probabilities. Unlike other models, we would not have to store and analyze every possible combination of states of each node. Instead, we can calculate and store probabilities of related nodes. Additionally, Bayesian networks are easily adaptable and can grow as more knowledge about a particular world is acquired.

Using a Bayesian network

To model this type of network using Java, we will create a network using JBayes (<https://github.com/vangi/jbayes>). JBayes is an open source library for creating a simple **Bayesian Belief Network (BBN)**. It is available at no cost for personal or commercial use. In our next example, we will perform approximate inference, a technique considered less accurate but allowing for decreased computational time. This technique is often used when working with big data because it produces a reliable model in a reasonable amount of time. We conduct approximate inference by using weight sampling of each node. JBayes also provides support for exact inference. Exact inference is most often used with smaller datasets or in situations where accuracy is of most importance. JBayes performs exact inference using the junction tree algorithm.

To begin our approximate inference model, we will first create our nodes. We will use the preceding diagram depicting attributes affecting on-time arrival to work to build our network. In the following code example, we use method chaining to create our nodes. Three of the methods take a `String` parameter. The `name` method is the name associated with each node. For brevity, we are only using the first initial, so `s` represents `storms`, `t` represents `traffic`, and so on. The `value` method allows us to set values for the node. In each case, our nodes can only have two values: `t` for true or `f` for false:

```
Node storms =
Node.newBuilder().name("s").value("t").value("f").build();
Node traffic =
Node.newBuilder().name("t").value("t").value("f").build();
```

```

Node powerOut =
Node.newBuilder().name("p").value("t").value("f").build();
Node alarm = Node.newBuilder().name("a").value("t").value("f").build();
Node overslept =
Node.newBuilder().name("o").value("t").value("f").build();
Node lateToWork =
Node.newBuilder().name("l").value("t").value("f").build();

```

Next, we assign parents to each of our child nodes. Notice that `storms` is a parent node to both `traffic` and `powerOut`. The `lateToWork` node has two parent nodes, `traffic` and `overslept`:

```

traffic.addParent(storms);
powerOut.addParent(storms);
lateToWork.addParent(traffic);
alarm.addParent(powerOut);
overslept.addParent(alarm);
lateToWork.addParent(overslept);

```

Then we define the **conditional probability tables (CPTs)** for each of our nodes. These tables are basically two-dimensional arrays representing the probabilities of each attribute of each node. If we have more than one parent node, as in the case of the `lateToWork` node, we need a row for each. We have used arbitrary values for our probabilities in this example, but note that each row must sum to 1.0:

```

storms.setCpt(new double[][] {{0.7, 0.3}});
traffic.setCpt(new double[][] {{0.8, 0.2}});
powerOut.setCpt(new double[][] {{0.5, 0.5}});
alarm.setCpt(new double[][] {{0.7, 0.3}});
overslept.setCpt(new double[][] {{0.5, 0.5}});
lateToWork.setCpt(new double[][] {
    {0.5, 0.5},
    {0.5, 0.5}
});

```

Finally, we create a `Graph` object and add each node to our graph structure. We then use this graph to perform our sampling:

```

Graph bayesGraph = new Graph();
bayesGraph.addNode(storms);
bayesGraph.addNode(traffic);
bayesGraph.addNode(powerOut);
bayesGraph.addNode(alarm);
bayesGraph.addNode(overslept);
bayesGraph.addNode(lateToWork);

```

```
bayesGraph.sample(1000);
```

At this point, we may be interested in the probabilities of each event. We can use the prob method to check the probabilities of a True or False value for each node:

```
double[] stormProb = storms.probs();
double[] trafProb = traffic.probs();
double[] powerProb = powerOut.probs();
double[] alarmProb = alarm.probs();
double[] overProb = overslept.probs();
double[] lateProb = lateToWork.probs();

out.println("nStorm Probabilities");
out.println("True: " + stormProb[0] + " False: " + stormProb[1]);
out.println("nTraffic Probabilities");
out.println("True: " + trafProb[0] + " False: " + trafProb[1]);
out.println("nPower Outage Probabilities");
out.println("True: " + powerProb[0] + " False: " + powerProb[1]);
out.println("vAlarm Probabilities");
out.println("True: " + alarmProb[0] + " False: " + alarmProb[1]);
out.println("nOverslept Probabilities");
out.println("True: " + overProb[0] + " False: " + overProb[1]);
out.println("nLate to Work Probabilities");
out.println("True: " + lateProb[0] + " False: " + lateProb[1]);
```

Our output contains the probabilities of each value for each node. For example, the probability of a storm occurring is 71% while the probability of a storm not occurring is 29%:

```
Storm Probabilities
True: 0.71 False: 0.29
Traffic Probabilities
True: 0.726 False: 0.274
Power Outage Probabilities
True: 0.442 False: 0.558
Alarm Probabilities
True: 0.543 False: 0.457
Overslept Probabilities
True: 0.556 False: 0.444
Late to Work Probabilities
True: 0.469 False: 0.531
```

Note

Notice in this example that we have used numbers that produce a very high

likelihood of being late to work, roughly 47%. This is due to the fact that we have set the probabilities of our parent nodes fairly high as well. This data would vary dramatically if the chances of a storm were lower or if we changed some of the other child nodes as well.

If we would like to save information about our sample, we can save the data to a CSV file using the following code:

```
try {
    CsvUtil.saveSamples(bayesGraph, new FileWriter(
        new File("C://JBayesInfo.csv")));
} catch (IOException e) {
    // Handle exceptions
}
```

With this discussion of supervised learning complete, we will now move on to unsupervised learning.

Unsupervised machine learning

Unsupervised machine learning does not use annotated data; that is, the dataset does not contain anticipated results. While there are several unsupervised learning algorithms, we will demonstrate the use of association rule learning to illustrate this learning approach.

Association rule learning

Association rule learning is a technique that identifies relationships between data items. It is part of what is called **market basket analysis**. When a shopper makes purchases, these purchases are likely to consist of more than one item, and when it does, there are certain items that tend to be bought together. Association rule learning is one approach for identifying these related items. When an association is found, a rule can be formulated for it.

For example, if a customer buys diapers and lotion, they are also likely to buy baby wipes. An analysis can find these associations and a rule stating the observation can be formed. The rule would be expressed as $\{diapers, lotion\} \Rightarrow \{wipes\}$. Being able to identify these purchasing patterns allows a store to offer special coupons, arrange their products to be easier to get, or effect any number of other market-related activities.

One of the problems with this technique is that there are a large number of possible associations. One efficient method that is commonly used is the **apriori** algorithm. This algorithm works on a collection of transactions defined by a set of items. These items can be thought of as purchases and a transaction as a set of items bought together. The collection is often referred to as a database.

Consider the following set of transactions where a *1* indicates that the item was purchased as part of a transaction and *0* means that it was not purchased:

Transaction ID	Diapers	Lotion	Wipes	Formula
1	1	1	1	0
2	1	1	1	1
3	0	1	1	0
4	1	0	0	0
5	0	1	1	1

There are several analysis terms used with the apriori model:

- **Support:** This is the proportion of the items in a database that contain a subset of items. In the previous database, the item $\{diapers, lotion\}$ occurs 2/5 times or 20%.
- **Confidence:** This is a measure of the frequency of the rule being true. It is calculated as $conf(X \rightarrow Y) = sup(X \cup Y) / sup(X)$.
- **Lift:** This measures the degree to which the items are dependent upon each other. It is defined as $lift(X \rightarrow Y) = sup(X \cup Y) / (sup(X) * sup(Y))$.
- **Leverage:** Leverage is a measure of the number of transactions that are covered by both X and Y if X and Y are independent of each other. A value above 0 is a good indicator. It is calculated as $lev(X \rightarrow Y) = sup(X, Y) - sup(X) * sup(Y)$.
- **Conviction:** A measure of how often the rule makes an incorrect decision. It is defined as $conv(X \rightarrow Y) = 1 - sup(Y) / (1 - conf(X \rightarrow Y))$.

These definitions and sample values can be found at
https://en.wikipedia.org/wiki/Association_rule_learning.

Using association rule learning to find buying relationships

We will be using the `Apriori` Weka class to demonstrate Java support for the algorithm using two datasets. The first is the data discussed previously and the second deals with what a person may take on a hike.

The following is the data file, `babies.arff`, for baby information:

```
@relation TEST_ITEM_TRANS
@attribute Diapers {1, 0}
@attribute Lotion {1, 0}
@attribute Wipes {1, 0}
@attribute Formula {1, 0}
@data
1,1,1,0
1,1,1,1
0,1,1,0
1,0,0,0
0,1,1,1
```

We start by reading in the file using a `BufferedReader` instance. This object is used as the argument of the `Instances` class, which will hold the data:

```
try {
    BufferedReader br;
    br = new BufferedReader(new FileReader("babies.arff"));
    Instances data = new Instances(br);
```

```

        br.close();
        ...
    } catch (Exception ex) {
        // Handle exceptions
    }
}

```

Next, an `Apriori` instance is created. We set the number of rules to be generated and a minimum confidence for the rules:

```

Apriori apriori = new Apriori();
apriori.setNumRules(100);
apriori.setMinMetric(0.5);

```

The `buildAssociations` method generates the associations using the `Instances` variable. The associations are then displayed:

```

apriori.buildAssociations(data);
System.out.println(apriori);

```

There will be 100 rules displayed. The following is the abbreviated output. Each rule is followed by various measures of the rule:

Note

Note that rule 8 and 100 reflect the previous examples.

Apriori

```
=====
```

```

Minimum support: 0.3 (1 instances)
Minimum metric <confidence>: 0.5
Number of cycles performed: 14
Generated sets of large itemsets:
Size of set of large itemsets L(1): 8
Size of set of large itemsets L(2): 18
Size of set of large itemsets L(3): 16
Size of set of large itemsets L(4): 5
Best rules found:
1. Wipes=1 4 ==> Lotion=1 4 <conf:(1)> lift:(1.25) lev:(0.16) [0] conv:(0.8)
2. Lotion=1 4 ==> Wipes=1 4 <conf:(1)> lift:(1.25) lev:(0.16) [0] conv:(0.8)
3. Diapers=0 2 ==> Lotion=1 2 <conf:(1)> lift:(1.25) lev:(0.08) [0] conv:(0.4)
4. Diapers=0 2 ==> Wipes=1 2 <conf:(1)> lift:(1.25) lev:(0.08) [0] conv:(0.4)
5. Formula=1 2 ==> Lotion=1 2 <conf:(1)> lift:(1.25) lev:(0.08) [0]

```

```
conv: (0.4)
6. Formula=1 2 ==> Wipes=1 2 <conf: (1)> lift: (1.25) lev: (0.08) [0]
conv: (0.4)
7. Diapers=1 Wipes=1 2 ==> Lotion=1 2 <conf: (1)> lift: (1.25) lev: (0.08)
[0] conv: (0.4)
8. Diapers=1 Lotion=1 2 ==> Wipes=1 2 <conf: (1)> lift: (1.25) lev: (0.08)
[0] conv: (0.4)
...
62. Diapers=0 Lotion=1 Formula=1 1 ==> Wipes=1 1 <conf: (1)> lift: (1.25)
lev: (0.04) [0] conv: (0.2)
...
99. Lotion=1 Formula=1 2 ==> Diapers=1 1 <conf: (0.5)> lift: (0.83) lev:
(-0.04) [0] conv: (0.4)
100. Diapers=1 Lotion=1 2 ==> Formula=1 1 <conf: (0.5)> lift: (1.25) lev:
(0.04) [0] conv: (0.6)
```

This provides us with a list of relationships, which we can use to identify patterns in activities such as purchasing behavior.

Reinforcement learning

Reinforcement learning is a type of learning at the cutting edge of current research into neural networks and machine learning. Unlike unsupervised and supervised learning, reinforcement learning makes decisions based upon the results of an action. It is a goal-oriented learning process, similar to that used by many parents and teachers across the world. We teach children to study and perform well on tests so that they receive high grades as a reward. Likewise, reinforcement learning can be used to teach machines to make choices that will result in the highest reward.

There are four main components to reinforcement learning: the actor or agent, the state or scenario, the chosen action, and the reward. The actor is the object or vehicle making the decisions within the application. The state is the world the actor exists within. Any decision the actor makes occurs within the parameters of the state. The action is simply the choice the actor makes when given a set of options. The reward is the result of each action and influences the likelihood of choosing that particular action in the future.

It is essential to note that the action and the state where the action occurred are not independent. In fact, the correct, or highest rewarding, action can often depend upon the state in which the action occurs. If the actor is trying to decide how to cross a body of water, swimming might be a good option if the body of water is calm and rather small. Swimming would be a terrible choice for the actor to choose if he wanted to cross the Pacific Ocean.

To handle this problem, we can consider the **Q** function. This function results from the mapping of a particular state to an action within the state. The Q function would link a lower reward to swimming across the Pacific than it might for swimming across a small river. Rather than saying swimming is a low reward activity, the Q function allows for swimming to sometimes have a low reward and other times a higher reward.

Reinforcement learning always begins with a blank slate. The actor does not know the best path or sequence of decisions when the iteration first begins. However, after many iterations through a given problem, considering the results of each particular state-action pair choice, the algorithm improves and learns to make the highest rewarding choices.

The algorithms used to achieve reinforcement learning involve maximization of reward amidst a complex set of processes and choices. Though currently being tested in video games and other discrete environments, the ultimate goal is success of these algorithms in unpredictable, real-world scenarios. Within the topic of reinforcement learning, there are three main flavors or types: temporal difference learning, Q'-learning, and **State-Action-Reward-State-Action (SARSA)**.

Temporal difference learning takes into account previously learned information to inform future decisions. This type of learning assumes a correlation between past and future decisions. Before an action is taken, a prediction is made. This prediction is compared to other known information about the environment and similar decisions before an action is chosen. This process is known as bootstrapping and is thought to create more accurate and useful results.

Q-learning uses the Q function mentioned above to select not only the best action for one particular step in a given state, but the action that will lead to the highest reward *from that point forward*. This is known as the optimum policy. One great advantage Q-learning offers is the ability to make decisions without requiring a full model of the state. This allows it to function in states with random changes in actions and rewards.

SARSA is another algorithm used in reinforcement learning. Its name is fairly self-explanatory: the Q value depends upon the current **State**, the current chosen **Action**, the **Reward** for that action, the State the agent will exist in after the action is completed, and the subsequent Action taken in the new state. This algorithm looks further ahead one step to make the best possible decision.

There are limited tools currently available for performing reinforcement learning using Java. One popular tool is **Platform for Implementing Q-Learning Experiments (Piqle)**. This Java framework aims to provide tools for fast designing and testing of reinforcement learning experiments. Piqle can be downloaded from <http://piqle.sourceforge.net>. Another robust tool is called **Brown-UMBC Reinforcement Learning and Planning (BURPLAP)**. Found at <http://burlap.cs.brown.edu>, this library is also designed for development of algorithms and domains for reinforcement learning. This particular resource boasts in the flexibility of states and actions and supports a wide range of planning and learning algorithms. BURLAP also includes analysis tools for visualization purposes.

Summary

Machine learning is concerned with developing techniques that allow the applications to learn without having to be explicitly programmed to solve a problem. This flexibility allows such applications to be used in more varied settings with little to no modifications.

We saw how training data is used to create a model. Once the model has been trained, the model is evaluated using testing data. Both the training data and testing data come from the problem domain. Once trained, the model is used with other input data to make predictions.

We learned how the Weka Java API is used to create decision trees. This tree consists of internal nodes that represent different attributes of the problem. The leaves of the tree represent results. Since there are many ways of constructing a tree, part of the job of a decision tree is to create the best tree.

Support vector machines divide a dataset into sections thus classifying elements in the dataset. This classification is based on the attributes of the data such as age, hair color, or weight. With the model, it is possible to predict outcomes based on attributes of a data instance.

Bayesian networks are used to make predictions based on parent-child relationships between nodes. The probability of one event directly affects the probability of a child event, and we can use this information to predict outcomes of complex real-world environments.

In the section on association rule learning, we learned how the relationships between elements of a dataset can be identified. The more significant relationships allow us to create rules that are applied to solve various problems.

In our discussion of reinforcement learning, we discussed the elements of agent, state, action, and reward and their relationship to one another. We also discussed specific types of reinforcement learning and provided resources for further inquiry.

Having introduced the elements of machine learning, we are now ready to explore neural networks, found in the next chapter.

Chapter 7. Neural Networks

While neural networks have been around for a number of years, they have grown in popularity due to improved algorithms and more powerful machines. Some companies are building hardware systems that explicitly mimic neural networks (<https://www.wired.com/2016/05/google-tpu-custom-chips/>). The time has come to use this versatile technology to address data science problems.

In this chapter, we will explore the ideas and concepts behind neural networks and then demonstrate their use. Specifically, we will:

- Define and illustrate neural networks
- Describe how they are trained
- Examine various neural network architectures
- Discuss and demonstrate several different neural networks, including:
 - A simple Java example
 - A **Multi Layer Perceptron (MLP)** network
 - The **k-Nearest Neighbor (k-NN)** algorithm and others

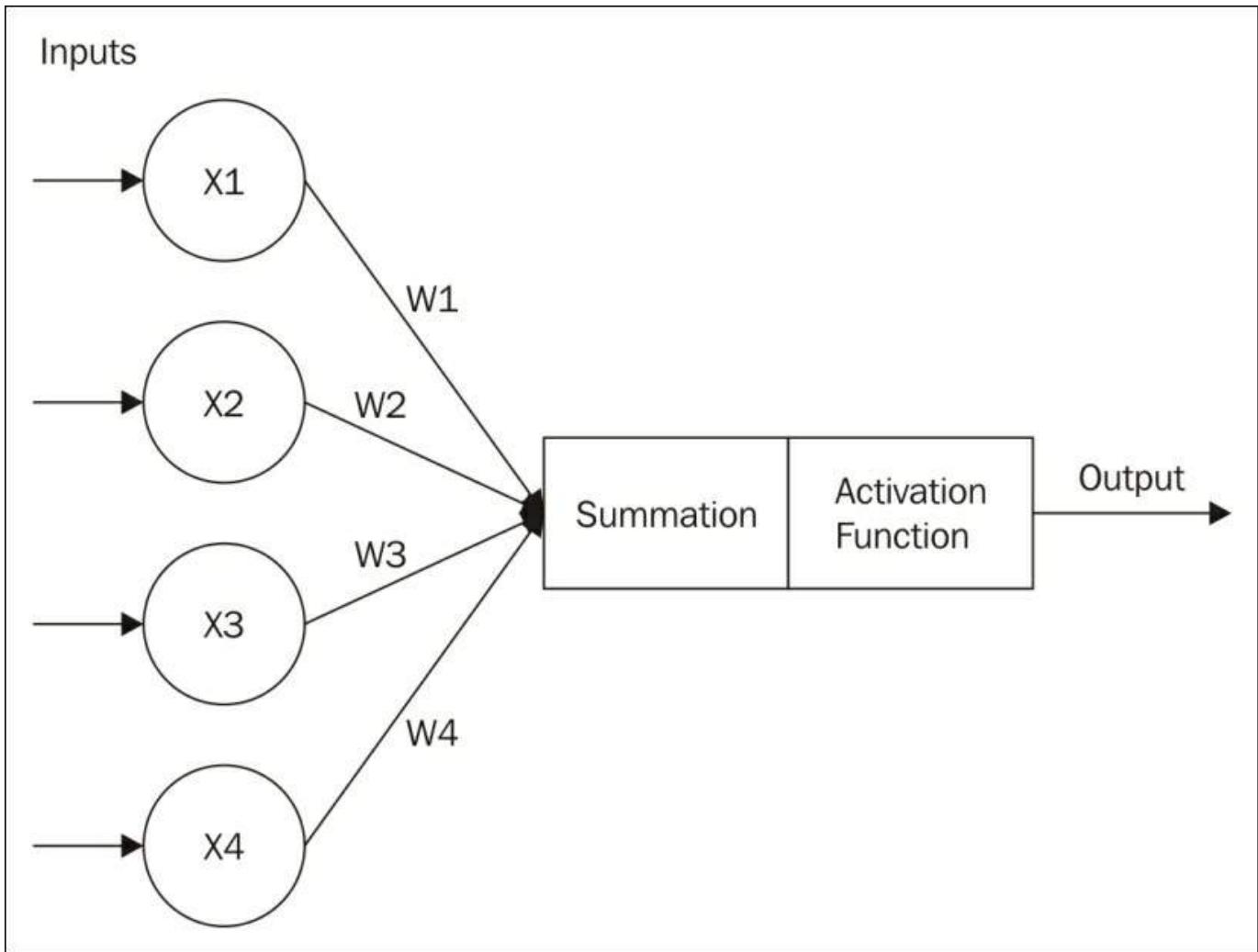
The idea for an **Artificial Neural Network (ANN)**, which we will call a neural network, originates from the neuron found in the brain. A **neuron** is a cell that has **dendrites** connecting it to input sources and other neurons. It receives stimulus from multiple sources through the **dendrites**. Depending on the source, the weight allocated to a source, the neuron is activated and **fires** a signal down a dendrite to another neuron. A collection of neurons can be trained and will respond to a particular set of input signals.

An artificial neuron is a node that has one or more inputs and a single output. Each input has a weight associated with it. By weighting inputs, we can amplify or de-amplify an input.

Note

Artificial neurons are alternately called **perceptrons**.

This is depicted in the following diagram, where the weights are summed and then sent to an **Activation Function** that determines the **Output**.



The neuron, and ultimately a collection of neurons, operate in one of two modes:

- **Training mode** - The neuron is trained to fire when a certain set of inputs are received
- **Testing mode** - Input is provided to the neuron, which responds as trained to a known set of inputs

A dataset is frequently split into two parts. A larger part is used to train a model. The second part is used to test and verify the model.

The output of a neuron is determined by the sum of the weighted inputs. Whether a neuron fires or not is determined by an **activation function**. There are several different types of activations functions, including:

- **Step function** - This linear function is computed using the summation of the

weighted inputs as follows:

$$Net_i = \sum W_i X_i$$

The $f(Net)$ designates the output of a function. It is 1 , if the Net input is greater than the activation threshold. When this happens the neuron fires. Otherwise it returns 0 and doesn't fire. The value is calculated based on all of the dendrite inputs.

- **Sigmoid** - This is a nonlinear function and is computed as follows:

$$f(Net) = \frac{1}{1 + e^{-Net_i}}$$

As the neuron is trained, the weights with each input can be adjusted.

In contrast to the step function, the sigmoid function is non-linear. This better matches some problem domains. We will find the sigmoid function used in multi-layer neural networks.

Training a neural network

There are three basic training approaches:

- **Supervised learning** - With supervised learning the model is trained with data that matches input sets to output values
- **Unsupervised learning** - In unsupervised learning, the data does not contain results, but the model is expected to determine relationships on its own
- **Reinforcement learning** - Similar to supervised learning, but a reward is provided for good results

These datasets differ in the information they contain. Supervised and reinforcement learning contain correct output for a set of input. The unsupervised learning does not contain correct results.

A neural network learns (at least with supervised learning) by feeding an input into a network and comparing the results, using the activation function, to the expected outcome. If they match, then the network has been trained correctly. If they don't match then the network is modified.

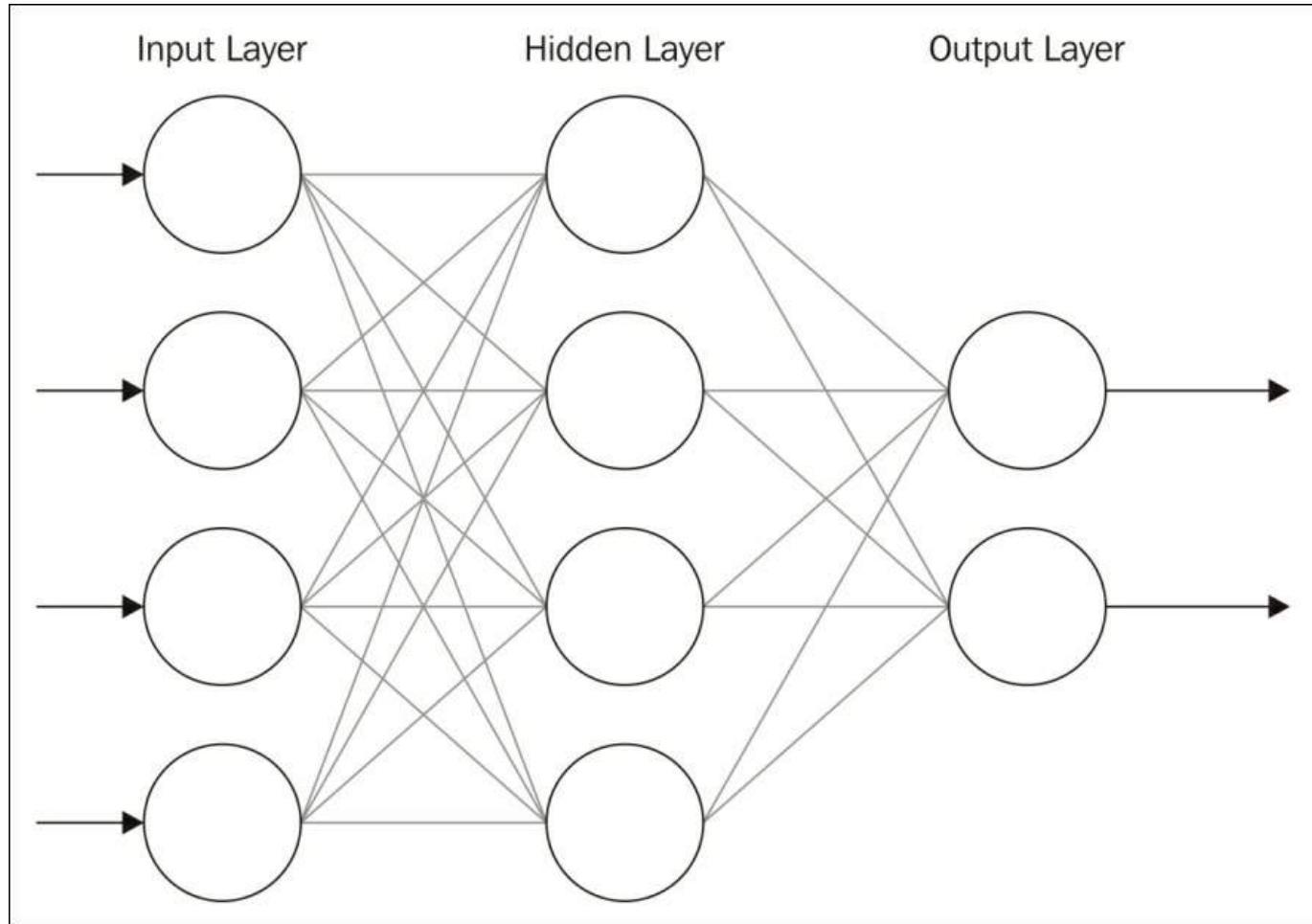
When we modify the weights we need to be careful not to change them too drastically. If the change is too large, then the results may change too much and we may miss the desired output. If the change is too little, then training the model will take too long. There are times when we may not want to change some weights.

A **bias** unit is a neuron that has a constant output. It is always one and is sometimes referred to as a fake node. This neuron is similar to an offset and is essential for most networks to function properly. You could compare the bias neuron to the y -intercept of a linear function in slope-intercept form. Just as adjusting the y -intercept value changes the location of the line, but not the shape/slope, the bias neuron can change the output values without adjusting the shape or function of the network. You can adjust the outputs to fit the particular needs of your problem.

Getting started with neural network architectures

Neural networks are usually created using a series of layers of neurons. There is typically an **Input Layer**, one or more middle layers (**Hidden Layer**), and an **Output Layer**.

The following is the depiction of a feedforward network:



The number of nodes and layers will vary. A feedforward network moves the information forward. There are also feedback networks where information is passed backwards. Multiple hidden layers are needed to handle the more complicated processing required for most analysis.

We will discuss several architectures and algorithms related to different types of neural networks throughout this chapter. Due to the complexity and length of explanation required, we will only provide in-depth analysis of a few key network types. Specifically, we will demonstrate a simple neural network, MLPs, and **Self-Organizing Maps (SOMs)**.

We will, however, provide an overview of many different options. The type of neural network and algorithm implementation appropriate for any particular model will depend upon the problem being addressed.

Understanding static neural networks

Static neural networks are ANNs that undergo a training or learning phase and then do not change when they are used. They differ from dynamic neural networks, which learn constantly and may undergo structural changes after the initial training period. Static neural networks are useful when the results of a model are relatively easy to reproduce or are more predictable. We will look at dynamic neural networks in a moment, but we will begin by creating our own basic static neural network.

A basic Java example

Before we examine various libraries and tools available for constructing neural networks, we will implement our own basic neural network using standard Java libraries. The next example is an adaptation of work done by Jeff Heaton (<http://www.informit.com/articles/article.aspx?p=30596>). We will construct a feed-forward backpropagation neural network and train it to recognize the XOR operator pattern. Here is the basic truth table for XOR:

X	Y	Result
0	0	0
0	1	1
1	0	1
1	1	0

This network needs only two input neurons and one output neuron corresponding to the X and Y input and the result. The number of input and output neurons needed for models is dependent upon the problem at hand. The number of hidden neurons is often the sum of the number of input and output neurons, but the exact number may need to be changed as training progresses.

We are going to demonstrate how to create and train the network next. We first provide the network with an input and observe the output. The output is compared to the expected output and then the weight matrix, called `weightChanges`, is adjusted. This adjustment ensures that the subsequent output will be closer to the expected output. This process is repeated until we are satisfied that the network can produce results significantly close enough to the expected output. In this example, we present the input and output as arrays of doubles where each input or output neuron is an element of the array.

Note

The input and output are sometimes referred to as **patterns**.

First, we will create a `SampleNeuralNetwork` class to implement the network. Begin

by adding the variables listed underneath to the class. We will discuss and demonstrate their purposes later in this section. Our class contains the following instance variables:

```
double errors;
int inputNeurons;
int outputNeurons;
int hiddenNeurons;
int totalNeurons;
int weights;
double learningRate;
double outputResults[];
double resultsMatrix[];
double lastErrors[];
double changes[];
double thresholds[];
double weightChanges[];
double allThresholds[];
double threshChanges[];
double momentum;
double errorChanges[];
```

Next, let's take a look at our constructor. We have four parameters, representing the number of inputs to our network, the number of neurons in hidden layers, the number of output neurons, and the rate and momentum at which we wish for learning to occur. The `learningRate` is a parameter that specifies the magnitude of changes in weight and bias during the training process. The `momentum` parameter specifies what fraction of a previous weight should be added to create a new weight. It is useful to prevent convergence at **local minimums or saddle points**. A high momentum increases the speed of convergence in a system, but can lead to an unstable system if it is too high. Both the momentum and learning rate should be values between 0 and 1:

```
public SampleNeuralNetwork(int inputCount,
    int hiddenCount,
    int outputCount,
    double learnRate,
    double momentum) {
    ...
}
```

Within our constructor we initialize all private instance variables. Notice that `totalNeurons` is set to the sum of all inputs, outputs, and hidden neurons. This sum

is then used to set several other variables. Also notice that the `weights` variable is calculated by finding the product of the number of inputs and hidden neurons, the product of the hidden neurons and the outputs, and adding these two products together. This is then used to create new arrays of length `weight`:

```

learningRate = learnRate;
momentum = momentum;

inputNeurons = inputCount;
hiddenNeurons = hiddenCount;
outputNeurons = outputCount;
totalNeurons = inputCount + hiddenCount + outputCount;
weights = (inputCount * hiddenCount)
    + (hiddenCount * outputCount);

outputResults      = new double[totalNeurons];
resultsMatrix     = new double[weights];
weightChanges     = new double[weights];
thresholds        = new double[totalNeurons];
errorChanges      = new double[totalNeurons];
lastErrors        = new double[totalNeurons];
allThresholds    = new double[totalNeurons];
changes           = new double[weights];
threshChanges     = new double[totalNeurons];
reset();

```

Notice that we call the `reset` method at the end of the constructor. This method resets the network to begin training with a random weight matrix. It initializes the thresholds and results matrices to random values. It also ensures that all matrices used for tracking changes are set back to zero. Using random values ensures that different results can be obtained:

```

public void reset() {
    int loc;
    for (loc = 0; loc < totalNeurons; loc++) {
        thresholds[loc] = 0.5 - (Math.random());
        threshChanges[loc] = 0;
        allThresholds[loc] = 0;
    }
    for (loc = 0; loc < resultsMatrix.length; loc++) {
        resultsMatrix[loc] = 0.5 - (Math.random());
        weightChanges[loc] = 0;
        changes[loc] = 0;
    }
}

```

We also need a method called `calcThreshold`. The **threshold** value specifies how close a value has to be to the actual activation threshold before the neuron will fire. For example, a neuron may have an activation threshold of 1. The threshold value specifies whether a number such as 0.999 counts as 1. This method will be used in subsequent methods to calculate the thresholds for individual values:

```
public double threshold(double sum) {
    return 1.0 / (1 + Math.exp(-1.0 * sum));
}
```

Next, we will add a method to calculate the output using a given set of inputs. Both our input parameter and the data returned by the method are arrays of `double` values. First, we need two position variables to use in our loops, `loc` and `pos`. We also want to keep track of our position within arrays based upon the number of input and hidden neurons. The index for our hidden neurons will start after our input neurons, so its position is the same as the number of input neurons. The position of our output neurons is the sum of our input neurons and hidden neurons. We also need to initialize our `outputResults` array:

```
public double[] calcOutput(double input[]) {
    int loc, pos;
    final int hiddenIndex = inputNeurons;
    final int outIndex = inputNeurons + hiddenNeurons;

    for (loc = 0; loc < inputNeurons; loc++) {
        outputResults[loc] = input[loc];
    }
    ...
}
```

Then we calculate outputs based upon our input neurons for the first layer of our network. Notice our use of the `threshold` method within this section. Before we can place our sum in the `outputResults` array, we need to utilize the `threshold` method:

```
int rLoc = 0;
for (loc = hiddenIndex; loc < outIndex; loc++) {
    double sum = thresholds[loc];
    for (pos = 0; pos < inputNeurons; pos++) {
        sum += outputResults[pos] * resultsMatrix[rLoc++];
    }
    outputResults[loc] = threshold(sum);
}
```

Now we take into account our hidden neurons. Notice the process is similar to the previous section, but we are calculating outputs for the hidden layer rather than the input layer. At the end, we return our result. This result is in the form of an array of doubles containing the values of each output neuron. In our example, there is only one output neuron:

```

double result[] = new double[outputNeurons];
for (loc = outIndex; loc < totalNeurons; loc++) {
    double sum = thresholds[loc];

    for (pos = hiddenIndex; pos < outIndex; pos++) {
        sum += outputResults[pos] * resultsMatrix[rLoc++];
    }
    outputResults[loc] = threshold(sum);
    result[loc-outIndex] = outputResults[loc];
}

return result;

```

It is quite likely that the output does not match the expected output, given our XOR table. To handle this, we use error calculation methods to adjust the weights of our network to produce better output. The first method we will discuss is the `calcError` method. This method will be called every time a set of outputs is returned by the `calcOutput` method. It does not return data, but rather modifies arrays containing weight and threshold values. The method takes an array of doubles representing the ideal value for each output neuron. Notice we begin as we did in the `calcOutput` method and set up indexes to use throughout the method. Then we clear out any existing hidden layer errors:

```

public void calcError(double ideal[]) {
    int loc, pos;
    final int hiddenIndex = inputNeurons;
    final int outputIndex = inputNeurons + hiddenNeurons;

    for (loc = inputNeurons; loc < totalNeurons; loc++) {
        lastErrors[loc] = 0;
    }
}

```

Next we calculate the difference between our expected output and our actual output. This allows us to determine how to adjust the weights for further training. To do this, we loop through our arrays containing the expected outputs, `ideal`, and the actual

outputs, outputResults. We also adjust our errors and change in errors in this section:

```
for (loc = outputIndex; loc < totalNeurons; loc++) {  
    lastErrors[loc] = ideal[loc - outputIndex] -  
        outputResults[loc];  
    errors += lastErrors[loc] * lastErrors[loc];  
    errorChanges[loc] = lastErrors[loc] * outputResults[loc]  
        *(1 - outputResults[loc]);  
}  
  
int locx = inputNeurons * hiddenNeurons;  
for (loc = outputIndex; loc < totalNeurons; loc++) {  
    for (pos = hiddenIndex; pos < outputIndex; pos++) {  
        changes[locx] += errorChanges[loc] *  
            outputResults[pos];  
        lastErrors[pos] += resultsMatrix[locx] *  
            errorChanges[loc];  
        locx++;  
    }  
    allThresholds[loc] += errorChanges[loc];  
}
```

Next we calculate and store the change in errors for each neuron. We use the lastErrors array to modify the errorChanges array, which contains total errors:

```
for (loc = hiddenIndex; loc < outputIndex; loc++) {  
    errorChanges[loc] = lastErrors[loc] *outputResults[loc]  
        * (1 - outputResults[loc]);  
}
```

We also fine tune our system by making changes to the allThresholds array. It is important to monitor the changes in errors and thresholds so the network can improve its ability to produce correct output:

```
locx = 0;  
for (loc = hiddenIndex; loc < outputIndex; loc++) {  
    for (pos = 0; pos < hiddenIndex; pos++) {  
        changes[locx] += errorChanges[loc] *  
            outputResults[pos];  
        lastErrors[pos] += resultsMatrix[locx] *  
            errorChanges[loc];  
        locx++;  
    }
```

```

        allThresholds[loc] += errorChanges[loc];
    }
}

```

We have one other method used for calculating errors in our network. The `getError` method calculates the root mean square for our entire set of training data. This allows us to identify our average error rate for the data:

```

public double getError(int len) {
    double err = Math.sqrt(errors / (len * outputNeurons));
    errors = 0;
    return err;
}

```

Now that we can initialize our network, compute outputs, and calculate errors, we are ready to train our network. We accomplish this through the use of the `train` method. This method makes adjustments first to the weights based upon the errors calculated in the previous method, and then adjusts the thresholds:

```

public void train() {
    int loc;
    for (loc = 0; loc < resultsMatrix.length; loc++) {
        weightChanges[loc] = (learningRate * changes[loc]) +
            (momentum * weightChanges[loc]);
        resultsMatrix[loc] += weightChanges[loc];
        changes[loc] = 0;
    }
    for (loc = inputNeurons; loc < totalNeurons; loc++) {
        threshChanges[loc] = learningRate * allThresholds[loc] +
            (momentum * threshChanges[loc]);
        thresholds[loc] += threshChanges[loc];
        allThresholds[loc] = 0;
    }
}

```

Finally, we can create a new class to test our neural network. Within the `main` method of another class, add the following code to represent the XOR problem:

```

double xorIN[][] = {
    {0.0,0.0},
    {1.0,0.0},
    {0.0,1.0},
    {1.0,1.0}};

double xorEXPECTED[][] = { {0.0},{1.0},{1.0},{0.0} };

```

Next we want to create our new `SampleNeuralNetwork` object. In the following example, we have two input neurons, three hidden neurons, one output neuron (the XOR result), a learn rate of 0.7, and a momentum of 0.9. The number of hidden neurons is often best determined by trial and error. In subsequent executions, consider adjusting the values in this constructor and examine the difference in results:

```
SampleNeuralNetwork network = new  
    SampleNeuralNetwork(2,3,1,0.7,0.9);
```

Note

The learning rate and momentum should usually fall between zero and one.

We then repeatedly call our `calcOutput`, `calcError`, and `train` methods, in that order. This allows us to test our output, calculate the error rate, adjust our network weights, and then try again. Our network should display increasingly accurate results:

```
for (int runCnt=0;runCnt<10000;runCnt++) {  
    for (int loc=0;loc<xorIN.length;loc++) {  
        network.calcOutput(xorIN[loc]);  
        network.calcError(xorEXPECTED[loc]);  
        network.train();  
    }  
    System.out.println("Trial #" + runCnt + ",Error:" +  
        network.getError(xorIN.length));  
}
```

Execute the application and notice that the error rate changes with each iteration of the loop. The acceptable error rate will depend upon the particular network and its purpose. The following is some sample output from the preceding code. For brevity we have included the first and the last training output. Notice that the error rate is initially above 50%, but falls to close to 1% by the last run:

```
Trial #0,Error:0.5338334002845255  
Trial #1,Error:0.5233475199946769  
Trial #2,Error:0.5229843653785426  
Trial #3,Error:0.5226263062497853  
Trial #4,Error:0.5226916275713371  
...  
Trial #994,Error:0.014457034704806316  
Trial #995,Error:0.01444865096401158
```

```
Trial #996,Error:0.01444028142777395  
Trial #997,Error:0.014431926056394229  
Trial #998,Error:0.01442358481032747  
Trial #999,Error:0.014415257650182488
```

In this example, we have used a small scale problem and we were able to train our network rather quickly. In a larger scale problem, we would start with a training set of data and then use additional datasets for further analysis. Because we really only have four inputs in this scenario, we will not test it with any additional data.

This example demonstrates some of the inner workings of a neural network, including details about how errors and output can be calculated. By exploring a relatively simple problem we are able to examine the mechanics of a neural network. In our next examples, however, we will use tools that hide these details from us, but allow us to conduct robust analysis.

Understanding dynamic neural networks

Dynamic neural networks differ from static networks in that they continue learning after the training phase. They can make adjustments to their structure independently of external modification. A **feedforward neural network (FNN)** is one of the earliest and simplest dynamic neural networks. This type of network, as its name implies, only feeds information forward and does not form any cycles. This type of network formed the foundation for much of the later work in dynamic ANNs. We will show in-depth examples of two types of dynamic networks in this section, MLP networks and SOMs.

Multilayer perceptron networks

A MLP network is a FNN with multiple layers. The network uses supervised learning with backpropagation where feedback is sent to early layers to assist in the learning process. Some of the neurons use a nonlinear activation function mimicking biological neurons. Every nodes of one layer is fully connected to the following layer.

We will use a dataset called `dermatology.arff` that can be downloaded from <http://repository.seasr.org/Datasets/UCI/arff/>. This dataset contains 366 instances used to diagnosis erythematous-squamous diseases. It uses 34 attributes to classify the disease into one of five different categories. The following is a sample instance:

The last field represents the disease category. This dataset has been partitioned into two files: `dermatologyTrainingSet.arff` and `dermatologyTestingSet.arff`. The training set uses the first 80% (292 instances) of the original set and ends with line 456. The testing set is the last 20% (74 instances) and starts with line 457 of the original set (lines 457-530).

Building the model

Before we can make any predictions, it is necessary that we train the model on a representative set of data. We will use the Weka class, `MultilayerPerceptron`, for training and eventually to make predictions. First, we declare strings for the training and testing of filenames and the corresponding `FileReader` instances for them. The instances are created and the last field is specified as the field to use for classification:

```
String trainingFileName = "dermatologyTrainingSet.arff";
String testingFileName = "dermatologyTestingSet.arff";

try (FileReader trainingReader = new FileReader(trainingFileName);
     FileReader testingReader =
         new FileReader(testingFileName)) {
    Instances trainingInstances = new Instances(trainingReader);
    trainingInstances.setClassIndex(
        trainingInstances.numAttributes() - 1);
    Instances testingInstances = new Instances(testingReader);
    testingInstances.setClassIndex(
        testingInstances.numAttributes() - 1);
```

```

    ...
} catch (Exception ex) {
    // Handle exceptions
}

```

An instance of the `MultilayerPerceptron` class is then created:

```
MultilayerPerceptron mlp = new MultilayerPerceptron();
```

There are several model parameters that we can set, as shown here:

Parameter	Method	Description
Learning rate	<code>setLearningRate</code>	Affects the training speed
Momentum	<code>setMomentum</code>	Affects the training speed
Training time	<code>setTrainingTime</code>	The number of training epochs used to train the model
Hidden layers	<code>setHiddenLayers</code>	The number of hidden layers and perceptrons to use

As mentioned previously, the learning rate will affect the speed in which your model is trained. A large value can increase the training speed. If the learning rate is too small, then the training time may take too long. If the learning rate is too large, then the model may move past a local minimum and become divergent. That is, if the increase is too large, we might skip over a meaningful value. You can think of this a graph where a small dip in a plot along the Y axis is missed because we incremented our X value too much.

Momentum also affects the training speed by effectively increasing the rate of learning. It is used in addition to the learning rate to add momentum to the search for the optimal value. In the case of a local minimum, the momentum helps get out of the minimum in its quest for a global minimum.

When the model is learning it performs operations iteratively. The term, **epoch** is used to refer to the number of iterations. Hopefully, the total error encounter with each epoch will decrease to a point where further epochs are not useful. It is ideal to avoid too many epochs.

A neural network will have one or more hidden layers. Each of these layers will have a specific number of perceptrons. The `setHiddenLayers` method specifies the

number of layers and perceptrons using a string. For example, 3,5 would specify two hidden layers with three and five perceptrons per layer, respectively.

For this example, we will use the following values:

```
mlp.setLearningRate(0.1);  
mlp.setMomentum(0.2);  
mlp.setTrainingTime(2000);  
mlp.setHiddenLayers("3");
```

The `buildClassifier` method uses the training data to build the model:

```
mlp.buildClassifier(trainingInstances);
```

Evaluating the model

The next step is to evaluate the model. The `Evaluation` class is used for this purpose. Its constructor takes the training set as input and the `evaluateModel` method performs the actual evaluation. The following code illustrates this using the testing dataset:

```
Evaluation evaluation = new Evaluation(trainingInstances);  
evaluation.evaluateModel(mlp, testingInstances);
```

One simple way of displaying the results of the evaluation is using the `toSummaryString` method:

```
System.out.println(evaluation.toSummaryString());
```

This will display the following output:

```
Correctly Classified Instances 73 98.6486 %  
Incorrectly Classified Instances 1 1.3514 %  
Kappa statistic 0.9824  
Mean absolute error 0.0177  
Root mean squared error 0.076  
Relative absolute error 6.6173 %  
Root relative squared error 20.7173 %  
Coverage of cases (0.95 level) 98.6486 %  
Mean rel. region size (0.95 level) 18.018 %  
Total Number of Instances 74
```

Frequently, it will be necessary to experiment with these parameters to get the best results. The following are the results of varying the number of perceptrons:

Number of perceptrons	Correctly classified instances		Incorrectly classified instances	
	Number	Percentage	Number	Percentage
2	55	74.3243%	19	25.6757%
3	73	98.6486%	1	1.3514%
4	72	97.2973%	2	2.7027%
5	72	97.2973%	2	2.7027%

Predicting other values

Once we have a model trained, we can use it to evaluate other data. In the previous testing dataset there was one instance which failed. In the following code sequence, this instance is identified and the predicted and actual results are displayed.

Each instance of the testing dataset is used as input to the `classifyInstance` method. This method tries to predict the correct result. This result is compared to the last field of the instance that contains the actual value. For mismatches, the predicted and actual values are displayed:

```
for (int i = 0; i < testingInstances.numInstances(); i++) {
    double result = mlp.classifyInstance(
        testingInstances.instance(i));
    if (result != testingInstances
        .instance(i)
        .value(testingInstances.numAttributes() - 1)) {
        out.println("Classify result: " + result
            + " Correct: " + testingInstances.instance(i)
            .value(testingInstances.numAttributes() - 1));
        ...
    }
}
```

For the testing set we get the following output:

```
Classify result: 1.0 Correct: 3.0
```

We can get the likelihood of the prediction being correct using the

MultilayerPerceptron class' distributionForInstance method. Place the following code into the previous loop. It will capture the incorrect instance, which is easier than instantiating an instance based on the 34 attributes used by the dataset. The distributionForInstance method takes this instance and returns a two element array of doubles. The first element is the probability of the result being positive and the second is the probability of it being negative:

```
Instance incorrectInstance = testingInstances.instance(i);
incorrectInstance.setDataset(trainingInstances);
double[] distribution = mlp.distributionForInstance(incorrectInstance);
out.println("Probability of being positive: " + distribution[0]);
out.println("Probability of being negative: " + distribution[1]);
```

The output for this instance is as follows:

```
Probability of being positive: 0.00350515156929017
Probability of being negative: 0.9683660500711128
```

This can provide a more quantitative feel for the reliability of the prediction.

Saving and retrieving the model

We can also save and retrieve a model for later use. To save the model, build the model and then use the `SerializationHelper` class' static method `write`, as shown in the following code snippet. The first argument is the name of the file to hold the model:

```
SerializationHelper.write("mlpModel", mlp);
```

To retrieve the model, use the corresponding `read` method as shown here:

```
mlp = (MultilayerPerceptron) SerializationHelper.read("mlpModel");
```

Next, we will learn how to use another useful neural network approach, SOMs.

Learning vector quantization

Learning Vector Quantization (LVQ) is another special type of a dynamic ANN. SOMs, which we will discuss in a moment, are a by-product of LVQ networks. This type of network implements a competitive type of algorithm in which the winning neuron gains the weight. These types of networks are used in many different applications and are considered to be more natural and intuitive than some other ANNs. In particular, LVQ is effective for classification of text-based data.

The basic algorithm begins by setting the number of neurons, the weight for each neuron, how fast the neurons can learn, and a list of input vectors. In this context, a vector is similar to a vector in physics and represents the values provided to the input layer neurons. As the network is trained, a vector is used as input, a winning neuron is selected, and the weight of the winning neuron is updated. This model is iterative and will continue to run until a solution is found.

Self-Organizing Maps

SOMs is a technique that takes multidimensional data and reducing it to one or two dimensions. This compression technique is called **vector quantization**. The technique usually involves a visual component that allows a human to better see how the data has been categorized. SOM learns without supervision.

The SOM is good for finding clusters, which is not to be confused with classification. With classification we are interested in finding the best fit for a data instance among predefined categories. With clustering we are interested in grouping instances where the categories are unknown.

A SOM uses a lattice of neurons, usually a two-dimensional array or a hexagonal grid, representing neurons that are assigned weights. The input sources are connected to each of these neurons. The technique then adjusts the weights assigned to each lattice member through several iterations until the best fit is found. When finished, the lattice members will have grouped the input dataset into categories. The SOM results can be viewed to identify categories and map new input to one of the identified categories.

Using a SOM

We will use the Weka to demonstrate SOM. However, it does not come installed with standard Weka. Instead, we will need to download a set of Weka classification algorithms from <https://sourceforge.net/projects/weka/classalgos/files/> and the actual SOM class from http://www.cis.hut.fi/research/som_pak/. The classification algorithms include support for LVQ. More details about the classification algorithms can be found at <http://weka.classalgos.sourceforge.net/>.

To use the SOM class, called `SelfOrganizingMap`, the source code needs to be in your project. The Javadoc for this class is found at <http://jsalatas.ictpro.gr/weka/doc/SelfOrganizingMap/>.

We start with the creation of an instance of the `SelfOrganizingMap` class. This is followed by code to read in data and create an `Instances` object to hold the data. In this example, we will use the `iris.arff` file, which can be found in the Weka data directory. Notice that once the `Instances` object is created we do not specify the class index as we did with previous Weka examples since SOM uses unsupervised

learning:

```
SelfOrganizingMap som = new SelfOrganizingMap();
String trainingFileName = "iris.arff";
try (FileReader trainingReader =
    new FileReader(trainingFileName)) {
    Instances trainingInstances = new Instances(trainingReader);
    ...
} catch (IOException ex) {
    // Handle exceptions
} catch (Exception ex) {
    // Handle exceptions
}
```

The `buildClusterer` method will execute the SOM algorithm using the training dataset:

```
som.buildClusterer(trainingInstances);
```

Displaying the SOM results

We can now display the results of the operation as follows:

```
out.println(som.toString());
```

The `iris` dataset uses five attributes: `sepallength`, `sepalwidth`, `petallength`, `petalwidth`, and `class`. The first four attributes are numeric and the fifth has three possible values: `Iris-setosa`, `Iris-versicolor`, and `Iris-virginica`. The first part of the abbreviated output that follows identified four clusters and the number of instances in each cluster. This is followed by statistics for each of the attributes:

```
Self Organized Map
=====
Number of clusters: 4
Cluster
Attribute 0 1 2 3
(50) (42) (29) (29)
=====
sepallength
value 5.0036 6.2365 5.5823 6.9513
min 4.3 5.6 4.9 6.2
max 5.8 7 6.3 7.9
mean 5.006 6.25 5.5828 6.9586
std. dev. 0.3525 0.3536 0.3675 0.5046
...
class
```

```
value 0 1.5048 1.0787 2
min 0 1 1 2
max 0 2 2 2
mean 0 1.4524 1.069 2
std. dev. 0 0.5038 0.2579 0
```

These statistics can provide insight into the dataset. If we are interested in determining which dataset instance is found in a cluster, we can use the `getClusterInstances` method to return the array that groups the instances by cluster. As shown next, this method is used to list the instance by cluster:

```
Instances[] clusters = som.getClusterInstances();
int index = 0;
for (Instances instances : clusters) {
    out.println("-----Custer " + index);
    for (Instance instance : instances) {
        out.println(instance);
    }
    out.println();
    index++;
}
```

As we can see with the abbreviated output of this sequence, different `iris` classes are grouped into the different clusters:

```
-----Custer 0
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
...
5.3,3.7,1.5,0.2,Iris-setosa
5,3.3,1.4,0.2,Iris-setosa
-----Custer 1
7,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
...
6.5,3,5.2,2,Iris-virginica
5.9,3,5.1,1.8,Iris-virginica

-----Custer 2
5.5,2.3,4,1.3,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
4.9,2.4,3.3,1,Iris-versicolor
...
4.9,2.5,4.5,1.7,Iris-virginica
```

6,2.2,5,1.5,Iris-virginica

-----Cluster 3

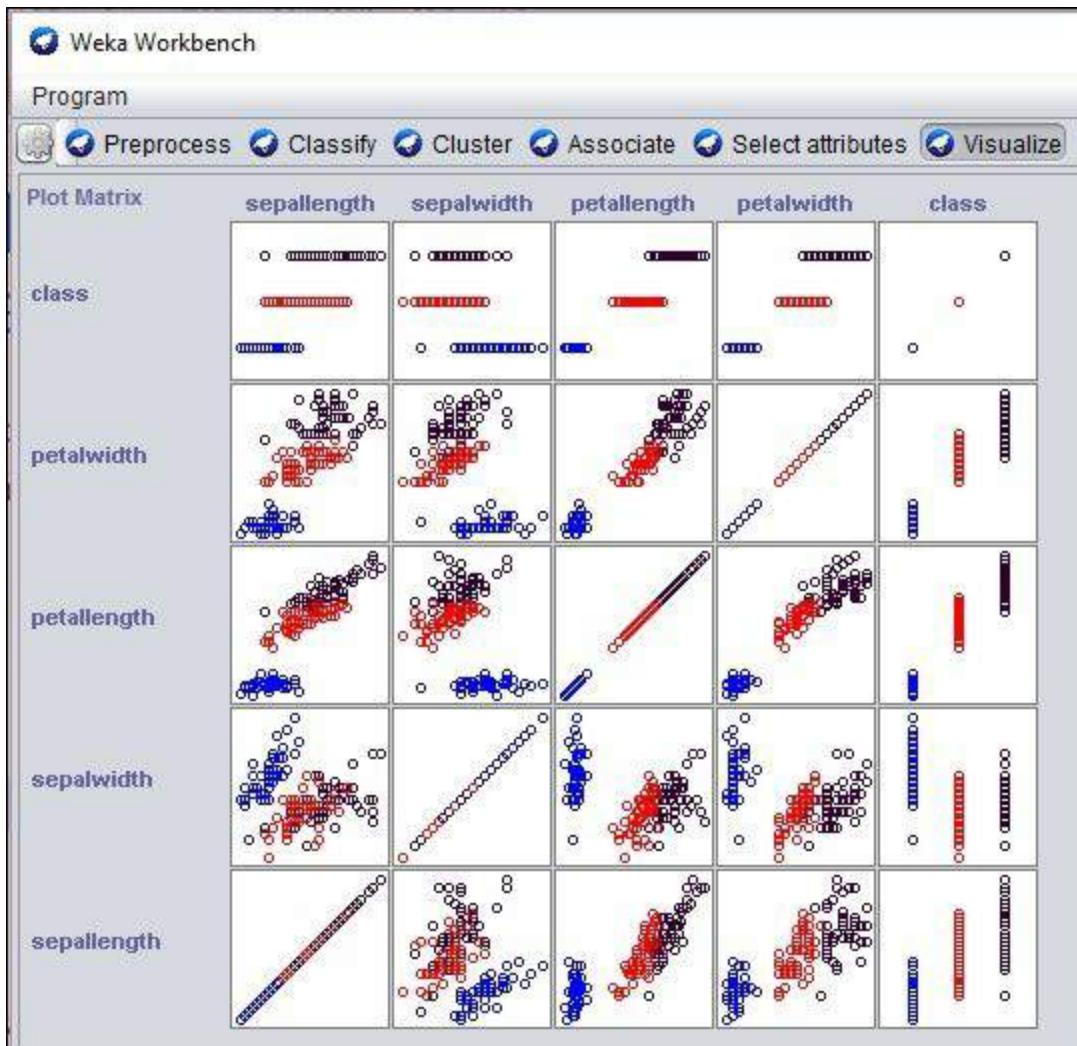
6.3,3.3,6,2.5,Iris-virginica

7.1,3,5.9,2.1,Iris-virginica

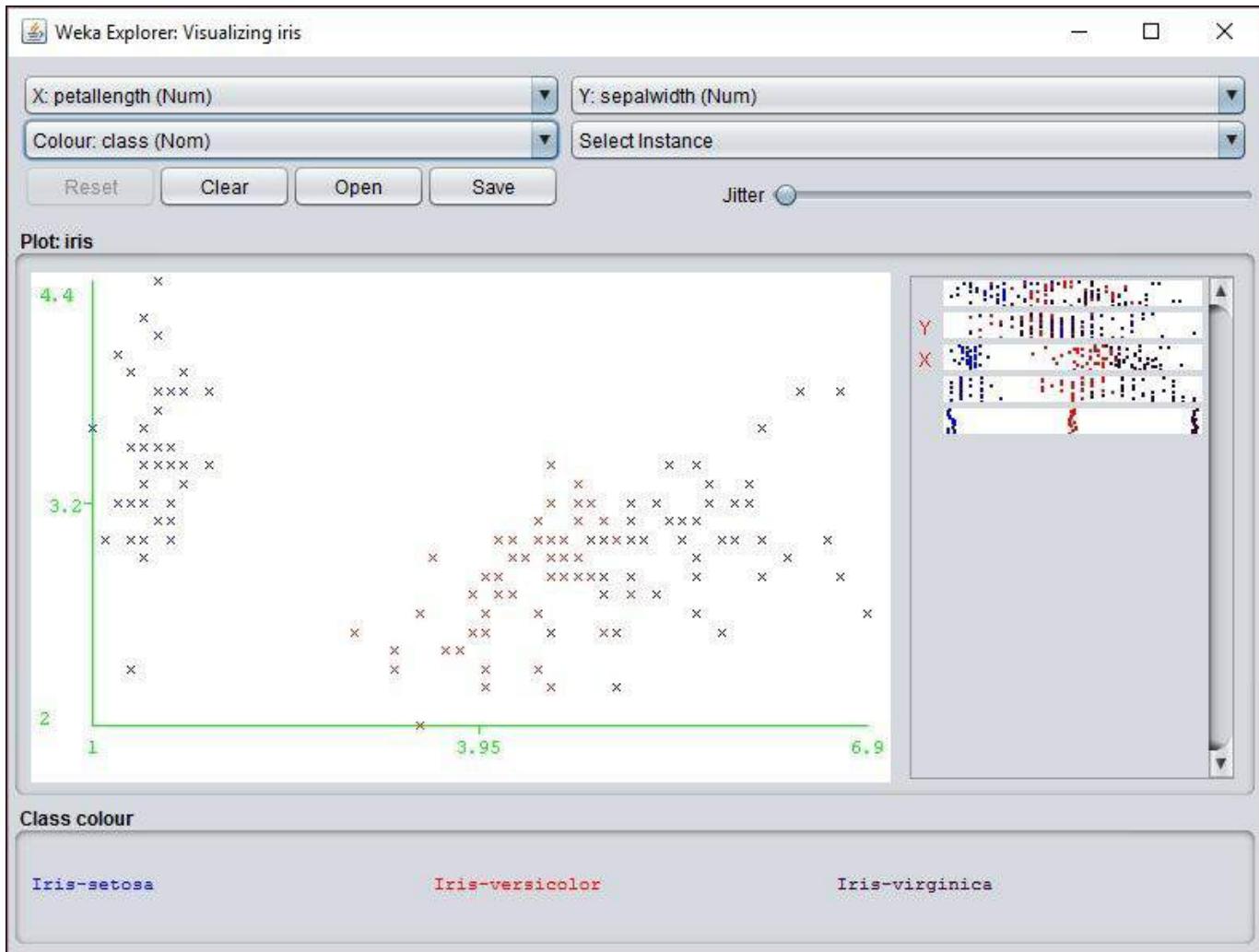
6.5,3,5.8,2.2,Iris-virginica

...

The cluster results can be displayed visually using the Weka GUI interface. In the following screenshot, we have used the **Weka Workbench** to analyze and visualize the result of the SOM analysis:



An individual section of the graph can be selected, customized, and analyzed as follows:



However, before you can use the `SOM` class, the `WekaPackageManager` must be used to add the `SOM` package to Weka. This process is discussed at <https://weka.wikispaces.com/How+do+I+use+the+package+manager%3F>.

If a new instance needs to be mapped to a cluster, the `distributionForInstance` method can be used as illustrated in *Predicting other values* section.

Additional network architectures and algorithms

We have discussed a few of the most common and practical neural networks. At this point, we would also like to consider some specialized neural networks and their application in various fields of study. These types of networks do not fit neatly into one particular category, but may still be of interest.

The k-Nearest Neighbors algorithm

An artificial neural network implementing the k-NN algorithm is similar to MLP networks, but it provides significant reduction in time compared to the winner takes all strategy. This type of network does not require a training algorithm after the initial weights are set and has fewer connections among its neurons. We have chosen not to provide an example of this algorithm's implementation because its use in Weka is very similar to the MLP example.

This type of network is best suited to classification tasks. Because it utilizes lazy learning techniques, reserving all computation until after information has been classified, it is considered to be one of the simpler models. In this model, the neurons are weighted based upon their distance from their neighbors. The classification of the neighbors is already known and therefore no specific training is required.

Instantaneously trained networks

Instantaneously Trained Neural Networks (ITNNs) are feedforward ANNs. They are special because they add a new hidden neuron for every unique set of training data. The main advantage to this type of network is the ability to provide generalization to other problems.

ITNNs are especially useful in short term learning situations. In particular, this type of network is useful for web searches and other pattern recognition functions with large datasets. These networks are suited for time series prediction and other deep learning purposes.

Spiking neural networks

A **Spiking Neural Network (SNN)** is a more complex ANN due to the fact it takes into account not only the neuron and information propagation, but also the timing of each event. In these networks, every neuron does not fire during every propagation of information, but rather only when the **membrane potential** for a particular neuron reaches a specific threshold. The membrane potential refers to the activation level of a neuron and closely resembles the way biological neurons fire.

Due to the close mimicry of biological neural networks, SNNs are especially suited to biological study and application. They have been used to model the nervous system of animals and insects and are useful for predicting the outcome of various stimuli. These networks have the ability to create very complex models with significant detail, but sacrifice time to accomplish this goal.

Cascading neural networks

Cascading Neural Networks (CNNs) is a specialized supervised learning algorithm. In this type, the network is initially very small and simplistic. As the network learns, it gradually adds new hidden units. Once the node is added, its input weight is constant and cannot be changed or removed.

This type of neural network is praised for its quick learning rate and ability to dynamically build itself. The user of such a network does not have to worry about topological design. Additionally, these networks do not require backpropagation of error information to make adjustments.

Holographic associative memory

Holographic Associative Memory (HAM) is a special type of complex neural network. This is a specialized type of network related to natural human memory and visual analysis. This network is especially useful for pattern recognition and associative memory tasks and can be applied to optical computations.

HAM attempts to closely mimic human visualization and pattern recognition. In this network, stimulus-response patterns are learned without iteration and backpropagation of errors is not required. Unlike other networks discussed in this chapter, HAM does not exhibit the same type of connected behaviour. Instead, the stimulus-response patterns can be stored within a single neuron.

Backpropagation and neural networks

Backpropagation algorithms are another supervised learning techniques used to train neural networks. As the name suggests, this algorithm calculates the computed output error and then changes the weights of each neuron in a backwards manner. Backpropagation is primarily used with MLP networks. It is important to note forward propagation must occur before backward propagation can be used.

In its most basic form, this algorithm consists of four steps:

1. Perform forward propagation for a given set of inputs.
2. Calculate the error value for each output.
3. Change the weights based upon the calculated error for each node.
4. Perform forward propagation again.

This algorithm completes when the output matches the expected output.

Summary

In this chapter, we have provided a broad overview of artificial neural networks, as well as a detailed examination of a few specific implementations. We began with a discussion of the basic properties of neural networks, training algorithms, and neural network architectures.

Next we provided an example of a simple static neural network implementing the XOR problem using Java. This example provided detailed explanation of the code used to build and train the network, including some of the math behind the weight adjustments during the training process. We then discussed dynamic neural networks and provided two in-depth examples, the MLP and SOM networks. These used the Weka tools to create and train the networks.

Finally, we concluded our chapter with a discussion of additional network architectures and algorithms. We chose some of the more popular networks to summarize and explored situations where each type would be most useful. We also included a discussion of backpropagation in this section.

In the next chapter, we will expand upon this introduction and take a look at deep learning with neural networks.

Chapter 8. Deep Learning

In this chapter, we will focus on neural networks, often referred to as **Deep Learning Networks (DLNs)**. This type of network is characterized as a multiple-layer neural network. Each of these layers are trained on the output of the previous layer, potentially identifying features and sub-features of the dataset. A feature hierarchy is created in this manner.

DLNs typically work with unstructured and unlabeled data, which constitute the vast bulk of data found in the world today. DLN will take this unstructured data, identify features, and try to reconstruct the original input. This approach is illustrated with **Restricted Boltzmann Machines (RBMs)** in *Restricted Boltzmann Machines* and with autoencoders in *Deep autoencoders*. An autoencoder takes a dataset and effectively compresses it. It then decompresses it to reconstruct the original dataset.

DLN can also be used for predictive analysis. The last step of a DLN will use an activation function to generate output represented by one of several categories. When used with new data, the model will attempt to classify the input based on the previously trained model.

An important DLN task is ensuring that the model is accurate and minimizes error. As with simple neural networks, weights and biases are used at each layer. As weight values are adjusted, errors can be introduced. A technique to adjust weights uses **gradient descent**. This can be thought of as the slope of the change. The idea is to modify the weight so as to minimize the error. It is an optimization technique that speeds up the learning process.

Later in the chapter, we will examine **Convolutional Neural Networks (CNNs)** and briefly discuss **Recurrent Neural Networks (RNN)**. Convolution networks mimic the visual cortex in that each neuron can interact with and make decisions **based** on a region of information. Recurrent networks process information based on not only the output of the previous layer but also the calculations performed in previous layers.

There are several libraries that support deep learning, including these:

- **N-Dimensional Arrays for Java (ND4J)** (<http://nd4j.org/>): A scientific computing library intended for production use
- **Deeplearning4j** (<http://deeplearnin4j.org/>): An open source, distributed deep-learning library

- **Encog** (<http://www.heatonresearch.com/encog/>): This library supports several deep learning algorithms

ND4J is a lower level library that is actually used in other projects, including DL4J. Encog is perhaps not as well supported as DL4J, but does provide support for deep learning.

The examples used in this chapter are all based on the **Deep Learning for Java (DL4J)** (<http://deeplearning4j.org>) API with support from ND4J. This library provides good support for many of the algorithms associated with deep learning. As a result, the next section explains the basic tasks found in common with many of the deep learning algorithms, such as loading data, training a model, and testing the model.

Deeplearning4j architecture

In this section, we will discuss its architecture and address several of the common tasks performed when using the API. DLN typically starts with the creation of a `MultiLayerConfiguration` instance, which defines the network, or model. The network is composed of multiple layers. **Hyperparameters** are used to configure the network and are variables that affect such things as learning speed, activation functions to use for a layer, and how weights are to be initialized.

As with neural networks, the basic DLN process consists of:

- Acquiring and manipulating data
- Configuring and building a model
- Training the model
- Testing the model

We will investigate each of these tasks in the next sections.

Note

The code examples in this section are not intended to be entered and executed here. Instead, these examples are snippets out of later models that we will be using.

Acquiring and manipulating data

The DL4J API has a number of techniques for acquiring data. We will focus on those specific techniques that we will use in our examples. The dataset used by a DL4J project is often modified using either **binarization** or **normalization**. Binarization converts data to ones and zeroes. Normalization converts data to a value between *1* and *0*.

Data feed to DLN is transformed to a set of numbers. These numbers are referred to as **vectors**. These vectors consist of a one-column matrix with a variable number of rows. The process of creating a vector is called **vectorization**.

Canova (<http://deeplearning4j.org/canova.html>) is a DL4J library that supports vectorization. It works with many different types of datasets. It has been merged with **DataVec** (<http://deeplearning4j.org/datavec>), a vectorization and **Extract, Transform, and Load (ETL)** library.

In this section, we will focus on how to read in CSV data.

Reading in a CSV file

ND4J provides the `CSVRecordReader` class, which is useful for reading CSV data. It has three overloaded constructors. The one we will demonstrate is passed two arguments. The first is the number of lines to skip when first reading a file and the second is a string holding the delimiters used to parse the text.

In the following code, we create a new instance of the class, where we do not skip any lines and use only a comma for a delimiter:

```
RecordReader recordReader = new CSVRecordReader(0, ",");
```

The class implements the `RecordReader` interface. It has an `initialize` method that is passed an instance of the `FileSplit` class. One of its constructors is passed an instance of a `File` object that references a dataset. The `FileSplit` class assists in splitting the data for training and testing. In this example, we initialize the reader for a file called `car.txt` that we will use in the *Preparing the data* section:

```
recordReader.initialize(new FileSplit(new File("car.txt")));
```

To process the data, we need an iterator such as the `DataSetIterator` instance

shown next. This class possesses a multitude of overloaded constructors. In the following example, the first argument is the `RecordReader` instance. This is followed by three arguments. The first is the batch size, which is the number of records to retrieve at a time. The next one is the index of the last attribute of the record. The last argument is the number of classes represented by the dataset:

```
DataSetIterator iterator =
    new RecordReaderDataSetIterator(recordReader, 1728, 6, 4);
```

The file's record's last attribute will hold a class value if we use a dataset for regression. This is precisely how we will use it later. The number of the class's parameter is only used with regression.

In the next code sequence, we will split the dataset into two sets: one for training and one for testing. Starting with the `next` method, this method returns the next dataset from the source. The size of the dataset is dependent on the batch size used earlier. The `shuffle` method randomizes the input while the `splitTestAndTrain` method returns an instance of the `SplitTestAndTrain` class, which we use to get the training and testing datasets. The `splitTestAndTrain` method's argument specifies the percentage of the data to be used for training.

```
DataSet dataset = iterator.next();
dataset.shuffle();
SplitTestAndTrain testAndTrain = dataset.splitTestAndTrain(0.65);
DataSet trainingData = testAndTrain.getTrain();
DataSet testData = testAndTrain.getTest();
```

We can then use these datasets with a model.

Configuring and building a model

Frequently, DL4J uses the `MultiLayerConfiguration` class to define the configuration of the model and the `MultiLayerNetwork` class to represent a model. These classes provide a flexible way of building models.

In the following example, we will demonstrate the use of these classes. Starting with the `MultiLayerConfiguration` class, we find that several methods are used in a fluent style. We will provide more details about these methods shortly. However, notice that two layers are defined for this model:

```
MultiLayerConfiguration conf =
    new NeuralNetConfiguration.Builder()
        .iterations(1000)
        .activation("relu")
        .weightInit(WeightInit.XAVIER)
        .learningRate(0.4)
        .list()
        .layer(0, new DenseLayer.Builder()
            .nIn(6).nOut(3)
            .build())
        .layer(1, new OutputLayer
            .Builder(LossFunctions.LossFunction
                .NEGATIVELOGLIKELIHOOD)
            .activation("softmax")
            .nIn(3).nOut(4).build())
        .backprop(true).pretrain(false)
        .build();
```

The `nIn` and `nOut` methods specify the number of inputs and outputs for a layer.

Using hyperparameters in ND4J

Builder classes are common in DL4J. In the previous example, the `NeuralNetConfiguration.Builder` class is used. The methods used here are but a few of the many that are available. In the following table, we describe several of them:

Method	Usage
<code>iterations</code>	Controls the number of optimization iterations performed
<code>activation</code>	This is the activation function used

weightInit	Used to initialize the initial weights for the model
learningRate	Controls the speed the model learns
List	Creates an instance of the <code>NeuralNetConfiguration.ListBuilder</code> class so that we can add layers
Layer	Creates a new layer
backprop	When set to true, it enables backpropagation
pretrain	When set to true, it will pretrain the model
Build	Performs the actual build process

Let's examine how a layer is created more closely. In the example, the `list` method returns a `NeuralNetConfiguration.ListBuilder` instance. Its `layer` method takes two arguments. The first is the number of the layer, a zero-based numbering scheme. The second is the `Layer` instance.

There are two different layers used here with two different builders: a `DenseLayer.Builder` and an `OutputLayer.Builder` instance. There are several types of layers available in DL4J. The argument of a builder's constructor may be a **loss function**, as is the case with the output layer, and is explained next.

In a feedback network, the neural network's guess is compared to what is called the **ground truth**, which is the error. This error is used to update the network through the modification of weights and biases. The loss function, also called an **objective** or **cost function**, measures the difference.

There are several loss functions supported by DL4J:

- **MSE**: In linear regression MSE stands for mean squared error
- **EXPLL**: In poisson regression EXPLL stands for exponential log likelihood
- **XENT**: In binary classification XENT stands for cross entropy
- **MCXENT**: This stands for multiclass cross entropy
- **RMSE_XENT**: This stands for RMSE cross entropy
- **SQUARED_LOSS**: This stands for squared loss
- **RECONSTRUCTION_CROSSENTROPY**: This stands for reconstruction cross entropy

- NEGATIVELOGLIKELIHOOD: This stands for negative log likelihood
- CUSTOM: Define your own loss function

The remaining methods used with the builder instance are the activation function, the number of inputs and outputs for the layer, and the `build` method, which creates the layer.

Each layer of a multi-layer network requires the following:

- **Input:** Usually in the form of an input vector
- **Weights:** Also called coefficients
- **Bias:** Used to ensure that at least some nodes in a layer are activated
- **Activation function:** Determines whether a node fires

There are many different types of activation functions, each of which can address a particular type of problem.

The activation function is used to determine whether the neuron fires. There are several functions supported, including `relu` (rectified linear), `tanh`, `sigmoid`, `softmax`, `hardtanh`, `leakyrelu`, `maxout`, `softsign`, and `softplus`.

Note

An interesting list of activation functions along with graphs is found at <http://stats.stackexchange.com/questions/115258/comprehensive-list-of-activation-functions-in-neural-networks-with-pros-cons> and https://en.wikipedia.org/wiki/Activation_function.

Instantiating the network model

Next, a `MultiLayerNetwork` instance is created using the defined configuration. The model is initialized, and its listeners are set. The `ScoreIterationListener` instance will display information as the model trains, which we will see shortly. Its constructor's argument specifies how often that information should be displayed:

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(100));
```

We are now ready to train the model.

Training a model

This is actually a fairly simple step. The `fit` method performs the training:

```
model.fit(trainingData);
```

When executed, the output will be generated using any listeners associated with the model, as is the preceding case, where a `ScoreIterationListener` instance is used.

Another example of how the `fit` method is used is through the process of iterating through a dataset, as shown next. In this example, a sequence of datasets is used. This is the part of an autoencoder where the output is intended to match the input, as explained in *Deep autoencoders* section. The dataset used as the argument to the `fit` method uses both the input and the expected output. In this case, they are the same as provided by the `getFeatureMatrix` method:

```
while (iterator.hasNext()) {  
    DataSet dataSet = iterator.next();  
    model.fit(new DataSet(dataSet.getFeatureMatrix(),  
        dataSet.getFeatureMatrix()));  
}
```

For larger datasets, it is necessary to pretrain the model several times to get accurate results. This is often performed in parallel to reduce training time. This option is set with a layer class's `pretrain` method.

Testing a model

The evaluation of a model is performed using the `Evaluation` class and the training dataset. An `Evaluation` instance is created using an argument specifying the number of classes. The test data is fed into the model using the `output` method. The `eval` method takes the output of the model and compares it against the test data classes to generate statistics:

```
Evaluation evaluation = new Evaluation(4);  
INDArray output = model.output(testData.getFeatureMatrix());  
evaluation.eval(testData.getLabels(), output);  
out.println(evaluation.stats());
```

The output will look similar to the following:

```
=====Scores=====  
Accuracy: 0.9273  
Precision: 0.854  
Recall: 0.8323  
F1 Score: 0.843
```

These statistics are detailed here:

- **Accuracy:** This is a measure of how often the correct answer was returned.
- **Precision:** This is a measure of the probability that a positive response is correct.
- **Recall:** This measures how likely the result will be classified correctly if given a positive example.
- **F1 Score:** This is the probability that the network's results are correct. It is the harmonic mean of recall and precision. It is calculated by dividing the number of true positives by the sum of true positives and false negatives.

We will use the `Evaluation` class to determine the quality of our model. A measure called **f1** is used, whose values range from *0* to *1*, where *1* represents the best quality.

Deep learning and regression analysis

Neural networks can be used to perform regression analysis. However, other techniques (see the early chapters) may offer a more effective solution. With regression analysis, we want to predict a result based on several input variables.

We can perform regression analysis using an output layer that consists of a single neuron that sums the weighted input plus bias of the previous hidden layer. Thus, the result is a single value representing the regression.

Preparing the data

We will use a car evaluation database to demonstrate how to predict the acceptability of a car based on a series of attributes. The file containing the data we will be using can be downloaded from: <http://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data>. It consists of car data such as price, number of passengers, and safety information, and an assessment of its overall quality. It is this latter element, quality, that we will try to predict. The comma-delimited values in each attribute are shown next, along with substitutions. The substitutions are needed because the model expects numeric data:

Attribute	Original value	Substituted value
Buying price	vhigh, high, med, low	3,2,1,0
Maintenance price	vhigh, high, med, low	3,2,1,0
Number of doors	2, 3, 4, 5-more	2,3,4,5
Seating	2, 4, more	2,4,5
Cargo space	small, med, big	0,1,2
Safety	low, med, high	0,1,2

There are 1,728 instances in the file. The cars are marked with four classes:

Class	Number of instances	Percentage of instances	Original value	Substituted value
Unacceptable	1210	70.023%	unacc	0
Acceptable	384	22.222%	acc	1
Good	69	3.99%	good	2
Very good	65	3.76%	v-good	3

Setting up the class

We start with the definition of a `CarRegressionExample` class, as shown next, where an instance of the class is created and where the work is performed within its default constructor:

```
public class CarRegressionExample {  
  
    public CarRegressionExample() {  
        try {  
            ...  
        } catch (IOException | InterruptedException ex) {  
            // Handle exceptions  
        }  
    }  
  
    public static void main(String[] args) {  
        new CarRegressionExample();  
    }  
}
```

Reading and preparing the data

The first task is to read in the data. We will use the `CSVRecordReader` class to get the data, as explained in *Reading in a CSV file*:

```
RecordReader recordReader = new CSVRecordReader(0, ",");
recordReader.initialize(new FileSplit(new File("car.txt")));
DataSetIterator iterator = new
    RecordReaderDataSetIterator(recordReader, 1728, 6, 4);
```

With this dataset, we will split the data into two sets. Sixty five percent of the data is used for training and the rest for testing:

```
DataSet dataset = iterator.next();
dataset.shuffle();
SplitTestAndTrain testAndTrain = dataset.splitTestAndTrain(0.65);
DataSet trainingData = testAndTrain.getTrain();
DataSet testData = testAndTrain.getTest();
```

The data now needs to be normalized:

```
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainingData);
normalizer.transform(trainingData);
normalizer.transform(testData);
```

We are now ready to build the model.

Building the model

A `MultiLayerConfiguration` instance is created using a series of `NeuralNetConfiguration.Builder` methods. The following is the code used. We will discuss the individual methods following the code. Note that this configuration uses two layers. The last layer uses the `softmax` activation function, which is used for regression analysis:

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .iterations(1000)
    .activation("relu")
    .weightInit(WeightInit.XAVIER)
    .learningRate(0.4)
    .list()
    .layer(0, new DenseLayer.Builder()
        .nIn(6).nOut(3)
        .build())
    .layer(1, new OutputLayer
        .Builder(LossFunctions.LossFunction
            .NEGATIVELOGLIKELIHOOD)
        .activation("softmax")
        .nIn(3).nOut(4).build())
    .backprop(true).pretrain(false)
    .build();
```

Two layers are created. The first is the input layer. The `DenseLayer.Builder` class is used to create this layer. The `DenseLayer` class is a feed-forward and fully connected layer. The created layer uses the six car attributes as input. The output consists of three neurons that are fed into the output layer and is duplicated here for your convenience:

```
.layer(0, new DenseLayer.Builder()
    .nIn(6).nOut(3)
    .build())
```

The second layer is the output layer created with the `OutputLayer.Builder` class. It uses a loss function as the argument of its constructor. The `softmax` activation function is used since we are performing regression as shown here:

```
.layer(1, new OutputLayer
    .Builder(LossFunctions.LossFunction
        .NEGATIVELOGLIKELIHOOD)
    .activation("softmax")
    .nIn(3).nOut(4).build())
```

Next, a `MultiLayerNetwork` instance is created using the configuration. The model is initialized, its listeners are set, and then the `fit` method is invoked to perform the actual training. The `ScoreIterationListener` instance will display information as the model trains which we will see shortly in the output of this example. The `ScoreIterationListener` constructor's argument specifies the frequency that information is displayed:

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(100));
model.fit(trainingData);
```

We are now ready to evaluate the model.

Evaluating the model

In the next sequence of code, we evaluate the model against the training dataset. An `Evaluation` instance is created using an argument specifying that there are four classes. The test data is fed into the model using the `output` method. The `eval` method takes the output of the model and compares it against the test data classes to generate statistics. The `getLabels` method returns the expected values:

```
Evaluation evaluation = new Evaluation(4);
INDArray output = model.output(testData.getFeatureMatrix());
evaluation.eval(testData.getLabels(), output);
out.println(evaluation.stats());
```

The output of the training follows, which is produced by the `ScoreIterationListener` class. However, the values you get may differ due to how the data is selected and analyzed. Notice that the score improves with the iterations but levels out after about 500 iterations:

```
12:43:35.685 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 0 is 1.443480901811554
12:43:36.094 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 100 is 0.3259061845624861
12:43:36.390 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 200 is 0.2630572026049783
12:43:36.676 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 300 is 0.24061281470878784
12:43:36.977 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 400 is 0.22955121170274934
12:43:37.292 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 500 is 0.22249920540161677
12:43:37.575 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 600 is 0.2169898450109222
12:43:37.872 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 700 is 0.21271599814600958
12:43:38.161 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 800 is 0.2075677126088741
12:43:38.451 [main] INFO o.d.o.l.ScoreIterationListener - Score at
iteration 900 is 0.20047317735870715
```

This is followed by the results of the `stats` method as shown next. The first part reports on how examples are classified and the second part displays various statistics:

```
Examples labeled as 0 classified by model as 0: 397 times
```

```
Examples labeled as 0 classified by model as 1: 10 times
Examples labeled as 0 classified by model as 2: 1 times
Examples labeled as 1 classified by model as 0: 8 times
Examples labeled as 1 classified by model as 1: 113 times
Examples labeled as 1 classified by model as 2: 1 times
Examples labeled as 1 classified by model as 3: 1 times
Examples labeled as 2 classified by model as 1: 7 times
Examples labeled as 2 classified by model as 2: 21 times
Examples labeled as 2 classified by model as 3: 14 times
Examples labeled as 3 classified by model as 1: 2 times
Examples labeled as 3 classified by model as 3: 30 times
=====Scores=====Accu
racy: 0.9273
Precision: 0.854
Recall: 0.8323
F1 Score: 0.843
=====
```

The regression model does a reasonable job with this dataset.

Restricted Boltzmann Machines

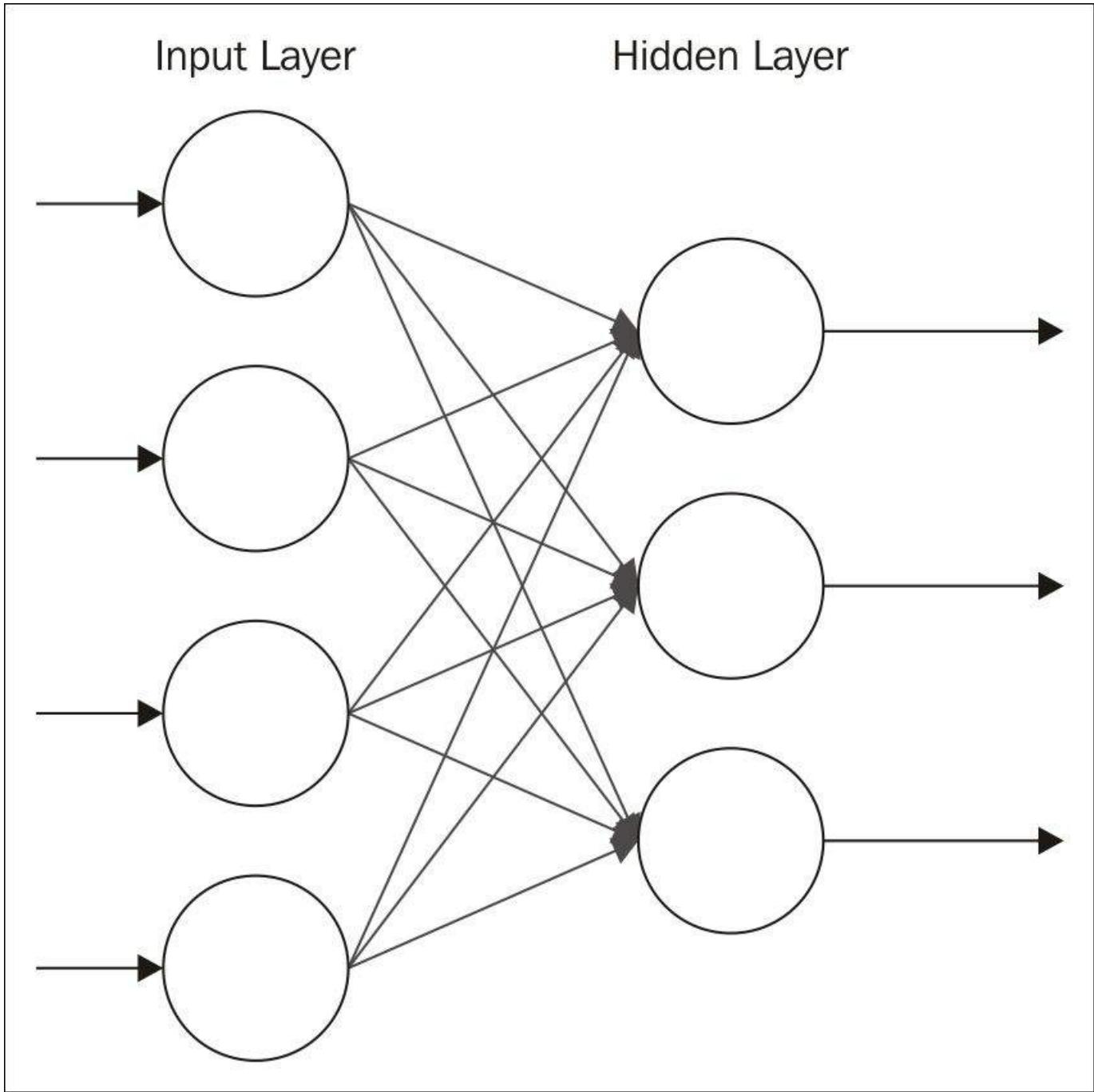
RBM is often used as part of a multi-layer deep belief network. The output of the RBM is used as an input to another layer. The use of the RBM is repeated until the final layer is reached.

Note

Deep Belief Networks (DBNs) consist of several RBMs stacked together. Each hidden layer provides the input for the subsequent layer. Within each layer, the nodes cannot communicate laterally and it becomes essentially a network of other single-layer networks. DBNs are especially helpful for classifying, clustering, and recognizing image data.

The term, **continuous restricted Boltzmann machine**, refers an RBM that uses values other than integers. Input data is normalized to values between zero and one.

Each node of the input layer is connected to each node of the second layer. No nodes of the same layer are connected to each other. That is, there is no intra-layer communication. This is what restricted means.



The number of input nodes for the visible layer is dependent on the problem being solved. For example, if we are looking at an image with 256 pixels, then we will need 256 input nodes. For an image, this is the number of rows times the number of columns for the image.

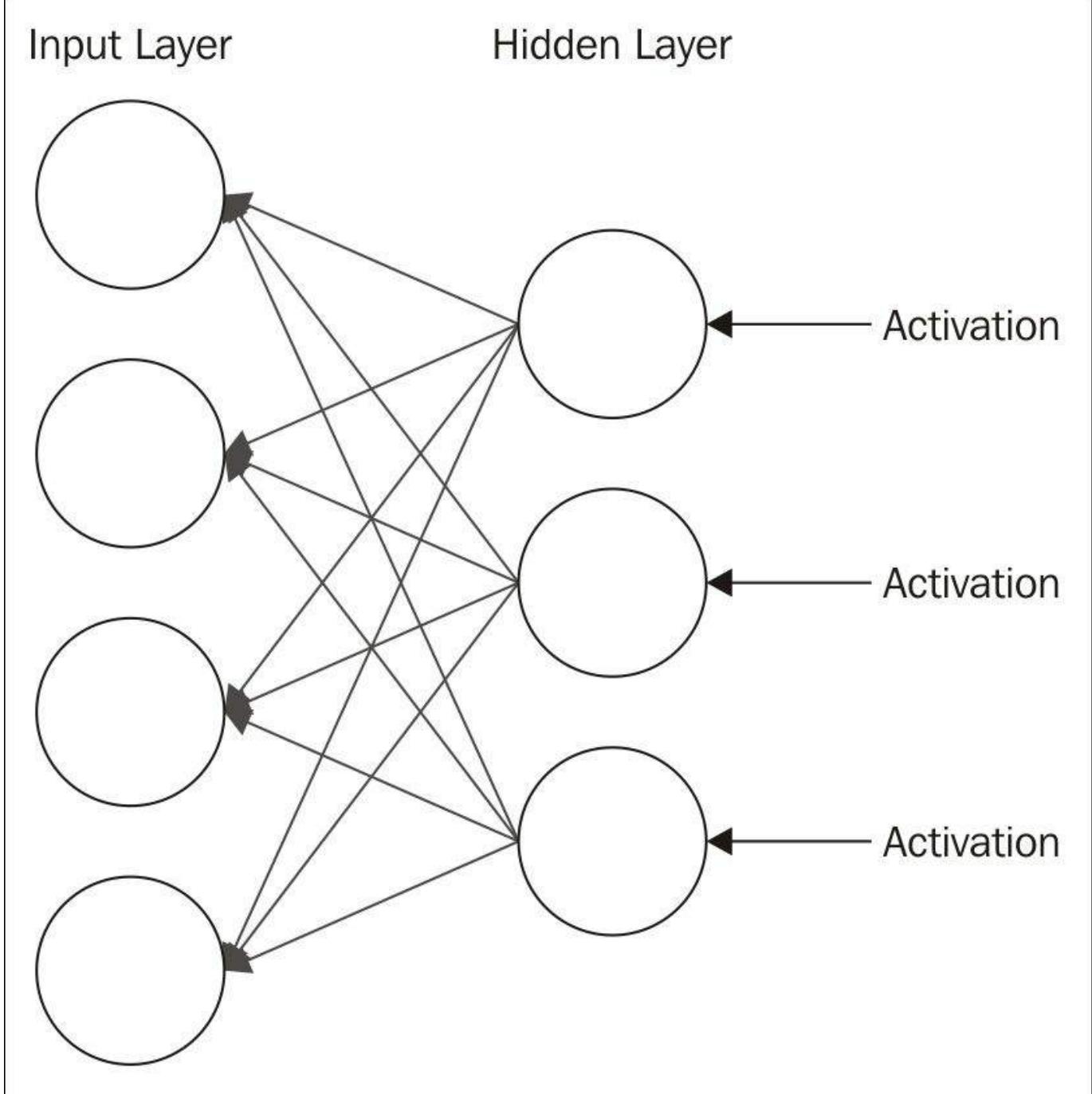
The **Hidden Layer** should contain fewer neurons than the **Input Layer**. Using close

to the same number of neurons will sometimes result in the construction of an identity function. Too many neurons may result in overfitting. This means that datasets with a large number of inputs will require multiple layers. Smaller input sizes result in the need for fewer layers.

Stochastic, that is, random, values are assigned to each node's weights. The value for a node is multiplied by its weight and then added to a bias. This value, combined with the combined input from the other input nodes, is then fed into the activation function, where an output value is generated.

Reconstruction in an RBM

The RBM technique goes through a reconstruction phase. This is where the activations are fed back to the first layer and multiplied by the same weights used for the input. The sum of these values from each node of the second layer, plus another bias, represents an approximation of the original input. The idea is to train the model to minimize the difference between the original input values and the feedback values.



The difference in values is treated as an error. The process is repeated until an error minimum is reached. You can think of the reconstruction as guesses about the original input. These guesses are essentially a probability distribution of the original input. This is called generative learning, in contrast to discriminative learning, which occurs with classification techniques.

In a multi-layer model, each layer can be used to essentially identify a feature. In subsequent layers, a combination of features may be identified or generated. In this way, a seemingly random set of pixel values may be analyzed to identify the veins of a leaf, a leaf, a trunk, and then a tree.

The output of an RBM is a value that essentially represents a percentage. If it is not zero, then the machine has learned something about the input.

Configuring an RBM

We will examine two different RBM configurations. The first one is minimal and we will see it again in *Deep autoencoders*. The second uses several additional methods and provides more insights into the various ways it can be configured.

The following statement creates a new layer using the `RBM.Builder` class. The input is computed based on the number of rows and columns of an image. The output is large, containing 1000 neurons. The loss function is `RMSE_XENT`. This loss function works better for some classification problems:

```
.layer(0, new RBM.Builder()
    .nIn(numRows * numColumns).nOut(1000)
    .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
    .build())
```

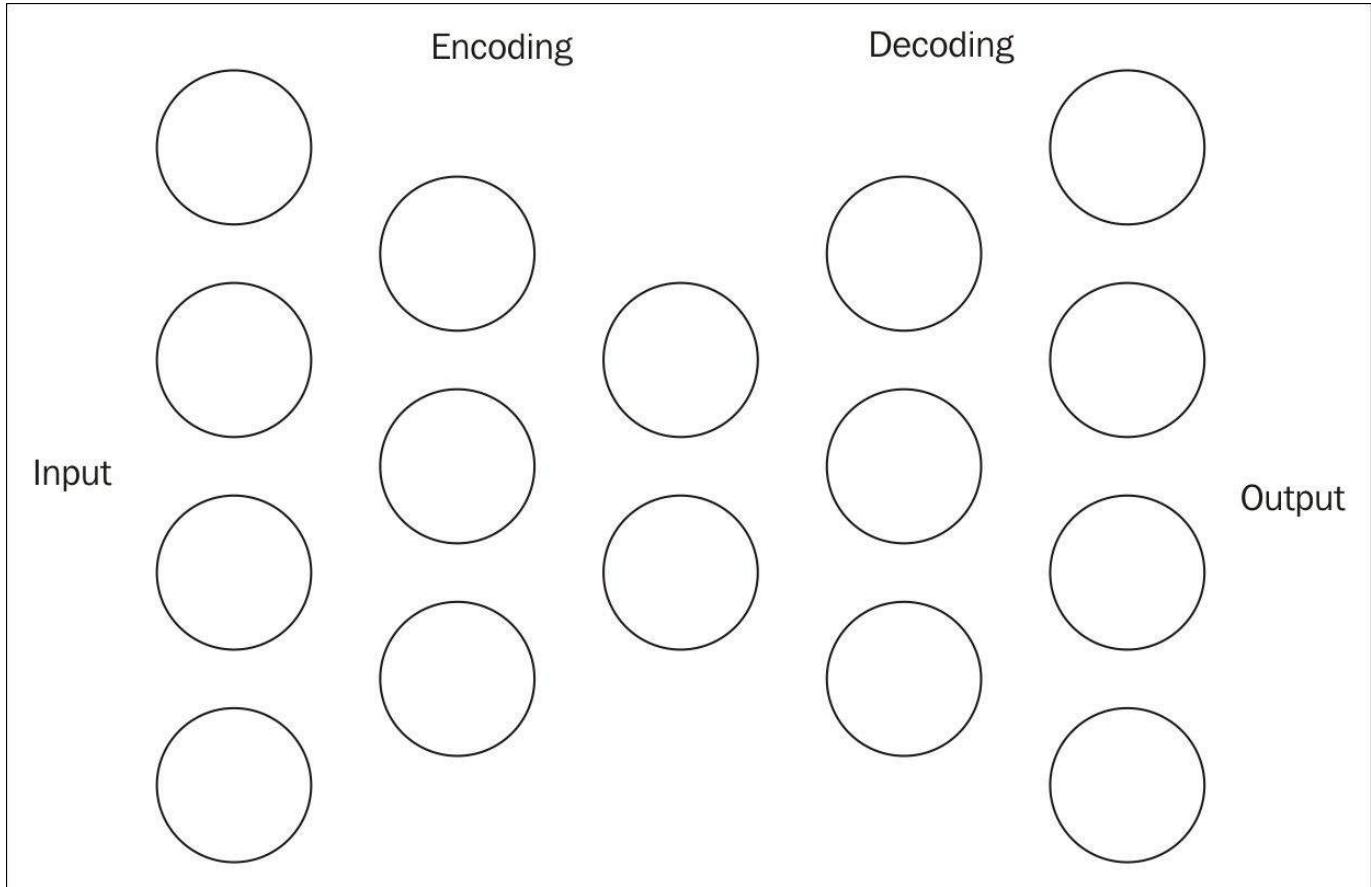
Next is a more complex RBM. We will not detail each of these methods here but will see them used in later examples:

```
.layer(new RBM.Builder()
    .l2(1e-1).l1(1e-3)
    .nIn(numRows * numColumns
    .nOut(outputNum)
    .activation("relu")
    .weightInit(WeightInit.RELU)
    .lossFunction(LossFunctions.LossFunction
        .RECONSTRUCTION_CROSSENTROPY).k(3)
    .hiddenUnit(HiddenUnit.RECTIFIED)
    .visibleUnit(VisibleUnit.GAUSSIAN)
    .updater(Updater.ADAGRAD)
        .gradientNormalization(
            GradientNormalization.ClipL2PerLayer)
    .build())
```

A single-layer `RBM` is not always useful. A multi-layer autoencoder is often required. We will look at this option in the next section.

Deep autoencoders

An autoencoder is used for feature selection and extraction. It consists of two symmetrical DBNs. The first half of the network is composed of several layers, which performs encoding. The second part of the network performs decoding. Each layer of the autoencoder is an RBM. This is illustrated in the following figure:



The purpose of the encoding sequence is to compress the original input into a smaller vector space. The middle layer of the previous figure is this compressed layer. These intermediate vectors can be thought of as possible features of the dataset. The encoding is also referred to as the pre-training half. It is the output of the intermediate RBM layer and does not perform classification.

The encoder's first layer will use more inputs than used by the dataset. This has the effect of expanding the features of the dataset. A sigmoid-belief unit is a form of non-linear transformation used with each layer. This unit is not able to accurately

represent information as real values. However, using more inputs, it is able to do a better job.

The second half of the network performs decoding, effectively reconstructing the input. This is a forward-feed network, using the same weights as the corresponding layers in the encoding half. However, the weights are transposed and are not initialized randomly. The training rate needs to be set lower for the second half.

An autoencoder is useful for data compression and searching. The output of the first half of the model is compressed, thus making it useful for storage and transmission usage. Later, it can be decompressed, as we will demonstrate in Chapter 10, *Visual and Audio Analysis*. This is sometimes referred to as semantic hashing.

If a series of inputs, such as images or sounds, have been compressed and stored, then new input can be compressed and matched with the stored values to find the best fit. An autoencoder can also be used for other information retrieval tasks.

Building an autoencoder in DL4J

This example is adapted from <http://deeplearning4j.org/deepautoencoder>. We start with a try-catch block to handle errors that may crop up and with a few variable declarations. This example uses the Mnist (<http://yann.lecun.com/exdb/mnist/>) dataset, which is a set of images containing hand-written numbers. Each image consists of 28 by 28 pixels. An iterator is declared to access the data:

```
try {
    final int numberOfRows = 28;
    final int numberOfColumns = 28;
    int seed = 123;
    int numberOfIterations = 1;

    iterator = new MnistDataSetIterator(
        1000, MnistDataFetcher.NUM_EXAMPLES, true);
    ...
} catch (IOException ex) {
    // Handle exceptions
}
```

Configuring the network

The configuration of the network is created using the `NeuralNetConfiguration.Builder()` class. Ten layers are created where the input layer consists of 1000 neurons. This is larger than the 28 by 28 pixel input and is used to compensate for the sigmoid-belief units used in each layer.

Each of the subsequent layers gets smaller until layer four is reached. This layer represents the last step of the encoding process. With layer five, the decoding process starts and the subsequent layers get bigger. The last layer uses 1000 neurons.

Each layer of the model uses an RBM instance except the last layer, which is constructed using the `OutputLayer.Builder` class. The configuration code follows:

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(numberOfIterations)
    .optimizationAlgo(
        OptimizationAlgorithm.LINE_GRADIENT_DESCENT)
    .list()
    .layer(0, new RBM.Builder()
        .nIn(numberOfRows * numberOfColumns).nOut(1000)
```

```

        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(1, new RBM.Builder().nIn(1000).nOut(500)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(2, new RBM.Builder().nIn(500).nOut(250)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(3, new RBM.Builder().nIn(250).nOut(100)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(4, new RBM.Builder().nIn(100).nOut(30)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build()) //encoding stops
    .layer(5, new RBM.Builder().nIn(30).nOut(100)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build()) //decoding starts
    .layer(6, new RBM.Builder().nIn(100).nOut(250)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(7, new RBM.Builder().nIn(250).nOut(500)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(8, new RBM.Builder().nIn(500).nOut(1000)
        .lossFunction(LossFunctions.LossFunction.RMSE_XENT)
        .build())
    .layer(9, new OutputLayer.Builder(
        LossFunctions.LossFunction.RMSE_XENT).nIn(1000)
        .nOut(numberOfRows * numberOfColumns).build())
    .pretrain(true).backprop(true)
    .build();
}

```

Building and training the network

The model is then created and initialized, and score iteration listeners are set up:

```

model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(Collections.singletonList(
    (IterationListener) new ScoreIterationListener()));

```

The model is trained using the `fit` method:

```

while (iterator.hasNext()) {
    DataSet dataSet = iterator.next();
    model.fit(new DataSet(dataSet.getFeatureMatrix(),
        dataSet.getFeatureMatrix()));
}

```

Saving and retrieving a network

It is useful to save the model so that it can be used for later analysis. This is accomplished using the `ModelSerializer` class's `writeModel` method. It takes the `model` instance and `modelFile` instance, along with a `boolean` parameter specifying whether the model's updater should be saved. An updater is a learning algorithm used for adjusting certain model parameters:

```
modelFile = new File("savedModel");
ModelSerializer.writeModel(model, modelFile, true);
```

The model can be retrieved using the following code:

```
modelFile = new File("savedModel");
MultiLayerNetwork model =
ModelSerializer.restoreMultiLayerNetwork(modelFile);
```

Specialized autoencoders

There are specialized versions of autoencoders. When an autoencoder uses more hidden layers than inputs, it may learn the identity function, which is a function that always returns the same value used as input to the function. To avoid this problem, an extension to the **autoencoder**, **denoising autoencoder**, is used; it randomly modifies the input introducing noise. The amount of noise introduced varies depending on the input dataset. A **Stacked Denoising Autoencoder (SdA)** is a series of denoising autoencoders strung together.

Convolutional networks

CNNs are feed-forward networks modeled after the visual cortex found in animals. The visual cortex is arranged with overlapping neurons, and so in this type of network, the neurons are also arranged in overlapping sections, known as receptive fields. Due to their design model, they function with minimal preprocessing or prior knowledge, and this lack of human intervention makes them especially useful.

This type of network is used frequently in image and video recognition applications. They can be used for classification, clustering, and object recognition. CNNs can also be applied to text analysis by implementing **Optical Character Recognition (OCR)**. CNNs have been a driving force in the machine learning movement in part due to their wide applicability in practical situations.

We are going to demonstrate a CNN using DL4J. The process will closely mirror the process we used in the *Building an autoencoder in DL4J* section. We will again use the `Mnist` dataset. This dataset contains image data, so it is well-suited to a convolutional network.

Building the model

First, we need to create a new `DataSetIterator` to process the data. The parameters for the `MnistDataSetIterator` constructor are the batch size, `1000` in this case, and the total number of samples to process. We then get our next dataset, shuffle the data to randomize, and split our data to be tested and trained. As we discussed earlier in the chapter, we typically use `65%` of the data to train the data and the remaining `35%` is used for testing:

```
DataSetIterator iter = new MnistDataSetIterator(1000,
MnistDataFetcher.NUM_EXAMPLES);
DataSet dataset = iter.next();
dataset.shuffle();
SplitTestAndTrain testAndTrain = dataset.splitTestAndTrain(0.65);
DataSet trainingData = testAndTrain.getTrain();
DataSet testData = testAndTrain.getTest();
```

We then normalize both sets of data:

```
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainingData);
normalizer.transform(trainingData);
normalizer.transform(testData);
```

Next, we can build our network. As shown earlier, we will again use a `MultiLayerConfiguration` instance with a series of `NeuralNetConfiguration.Builder` methods. We will discuss the individual methods after the following code sequence. Notice that the last layer again uses the `softmax` activation function for regression analysis:

```
MultiLayerConfiguration.Builder builder = new
    NeuralNetConfiguration.Builder()
.seed(123)
.iterations(1)
.regularization(true).l2(0.0005)
.weightInit(WeightInit.XAVIER)
.optimizationAlgo(OptimizationAlgorithm
    .STOCHASTIC_GRADIENT_DESCENT)
.updater(Updater.NESTEROVS).momentum(0.9)
.list()
.layer(0, new ConvolutionLayer.Builder(5, 5)
    .nIn(6)
    .stride(1, 1)
    .nOut(20)
```

```

.activation("identity")
.build())
.layer(1, new SubsamplingLayer.Builder(
    SubsamplingLayer.PoolingType.MAX)
.kernelSize(2, 2)
.stride(2, 2)
.build())
.layer(2, new ConvolutionLayer.Builder(5, 5)
.stride(1, 1)
.nOut(50)
.activation("identity")
.build())
.layer(3, new SubsamplingLayer.Builder(
    SubsamplingLayer.PoolingType.MAX)
.kernelSize(2, 2)
.stride(2, 2)
.build())
.layer(4, new DenseLayer.Builder().activation("relu")
.nOut(500).build())
.layer(5, new OutputLayer.Builder(
    LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
.nOut(10)
.activation("softmax")
.build())
.backprop(true).pretrain(false);

```

The first layer, layer 0, which is duplicated next for your convenience, uses the `ConvolutionLayer.Builder` method. The input to a convolution layer is the product of the image height, width, and number of channels. In a standard RGB image, there are three channels. The `nIn` method takes the number of channels. The `nOut` method specifies that 20 outputs are expected:

```

.layer(0, new ConvolutionLayer.Builder(5, 5)
.nIn(6)
.stride(1, 1)
.nOut(20)
.activation("identity")
.build())

```

Layers 1 and 3 are both subsampling layers. These layers follow convolution layers and do no real convolution themselves. They return a single value, the maximum value for that input region:

```

.layer(1, new SubsamplingLayer.Builder(
    SubsamplingLayer.PoolingType.MAX)
.kernelSize(2, 2)

```

```

    .stride(2, 2)
    .build())
    ...
.layer(3, new SubsamplingLayer.Builder(
    SubsamplingLayer.PoolingType.MAX)
    .kernelSize(2, 2)
    .stride(2, 2)
    .build())

```

Layer 2 is also a convolution layer like layer 0. Notice that we do not specify the number of channels in this layer:

```

.layer(2, new ConvolutionLayer.Builder(5, 5)
    .nOut(50)
    .activation("identity")
    .build())

```

The fourth layer uses the `DenseLayer.Builder` class, as in our earlier example. As mentioned previously, the `DenseLayer` class is a feed-forward and fully connected layer:

```

.layer(4, new DenseLayer.Builder().activation("relu")
    .nOut(500).build())

```

The layer 5 is an `OutputLayer` instance and uses softmax automation:

```

.layer(5, new OutputLayer.Builder(
    LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
    .nOut(10)
    .activation("softmax")
    .build())
    .backprop(true).pretrain(false);

```

Finally, we create a new instance of the `ConvolutionalLayerSetup` class. We pass the builder object and the dimensions of our image (28 x 28). We also pass the number of channels, in this case, 1:

```
new ConvolutionLayerSetup(builder, 28, 28, 1);
```

We can now configure and fit our model. We once again use the `MultiLayerConfiguration` and `MultiLayerNetwork` classes to build our network. We set up listeners and then iterate through our data. For each `DataSet`, we execute the `fit` method:

```
MultiLayerConfiguration conf = builder.build();
```

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(Collections.singletonList((IterationListener)
    new ScoreIterationListener(1/5)));

while (iter.hasNext()) {
    DataSet next = iter.next();
    model.fit(new DataSet(next.getFeatureMatrix(), next.getLabels()));
}
```

We are now ready to evaluate our model.

Evaluating the model

To evaluate our model, we use the `Evaluation` class. We get the output from our model and send it, along with the labels for our dataset, to the `eval` method. We then execute the `stats` method to get the statistical information on our network:

```
Evaluation evaluation = new Evaluation(4);
INDArray output = model.output(testData.getFeatureMatrix());
evaluation.eval(testData.getLabels(), output);
out.println(evaluation.stats());
```

The following is a sample output from the execution of this code, for we are only showing the results of the `stats` method. The first part reports on how examples are classified and the second part displays various statistics:

```
Examples labeled as 0 classified by model as 0: 19 times
Examples labeled as 1 classified by model as 1: 41 times
Examples labeled as 2 classified by model as 1: 4 times
Examples labeled as 2 classified by model as 2: 30 times
Examples labeled as 2 classified by model as 3: 1 times
Examples labeled as 3 classified by model as 2: 1 times
Examples labeled as 3 classified by model as 3: 28 times
=====Scores=====Accuracy: 0.3371
Precision: 0.8481
Recall: 0.8475
F1 Score: 0.8478
=====
```

As in our previous model, the evaluation demonstrates decent accuracy and success with our network.

Recurrent Neural Networks

RNN differ from feed-forward networks in that their input includes the input from the previous iteration or step. They still process the current input but use a feedback loop to take into consideration the inputs to the prior step, also called the recent past, for context. This step effectively gives the network memory. One popular type of recurrent network involves **Long Short-Term Memory (LSTM)**. This type of memory improves the processing power of the network.

RNNs are designed to process sequential data and are especially useful for analysis and prediction with text data. Given a sequence of words, an RNN can predict the probability of each word being the next in the sequence. This also allows for text generation by the network. RNNs are versatile and also process image data well, especially image labeling applications. The flexibility in design and purpose and ease in training make RNNs popular choices for many data science applications. DL4J also provides support for LSTM networks and other RNNs.

Summary

In this chapter, we examined deep learning techniques for neural networks. All API support in this chapter was provided by Deeplearning4j. We began by demonstrating how to acquire and prepare data for use with deep learning networks. We discussed how to configure and build a model. This was followed by an explanation of how to train and test a model by splitting the dataset into training and testing segments.

Our discussion continued with an examination of deep learning and regression analysis. We showed how to prepare the data and class, build the model, and evaluate the model. We used sample data and displayed output statistics to demonstrate the relative effectiveness of our model.

RBM and DBNs were then examined. DBNs are comprised of RBMs stacked together and are especially useful for classification and clustering applications. Deep autoencoders are also built using RBMs, with two symmetrical DBNs. The autoencoders are especially useful for feature selection and extraction.

Finally, we demonstrated a convolutional network. This network is modeled after the visual cortex and allows the network to use regions of information for classification. As in previous examples, we built, trained, and then evaluated the model for effectiveness. We then concluded the chapter with a brief introduction to recurrent neural networks.

We will expand upon these topics as we move into next chapter and examine text analysis techniques.

Chapter 9. Text Analysis

Text analysis is a broad topic and is typically referred to as **Natural Language Processing (NLP)**. It is used for many different tasks, including text searching, language translation, sentiment analysis, speech recognition, and classification, to mention a few. The process of analyzing can be difficult due to the particularities and ambiguity found in natural languages. However, there has been a considerable amount of work in this area and there are several Java APIs supporting this effort.

We will start with an introduction to the basic concepts and tasks used in NLP. These include the following:

- **Tokenization**: The process of splitting text into individual tokens or words.
- **Stop words**: These are words that are common and may not be necessary for processing. They include such words as the, a, and to.
- **Name Entity Recognition (NER)**: This is the process of identifying elements of text such as people's name, locations, or things.
- **Parts of Speech (POS)**: This identifies the grammatical parts of a sentence such as noun, verb, adjective, and so on.
- **Relationships**: Here, we are concerned with identifying how parts of text are related to each other, such as the subject and object of a sentence.

The concepts of words, sentences, and paragraphs are well known. However, extracting and analyzing these components is not always that straightforward. The term **corpus** frequently refers to a collection of text.

As with most data science problems, it is important to preprocess text. Frequently, this involves handling such tasks as these:

- Handling Unicode
- Converting text to uppercase or lowercase
- Removing stop words

We examined several techniques for tokenization and removing stop words in [Chapter 3, Data Cleaning](#). In this chapter, we will focus on POS, NER, extracting relationships from sentence, text classification, and sentiment analysis.

There are several NLP APIs available, including these:

- **OpenNLP** (<https://opennlp.apache.org/>): An open source Apache project

- **StanfordNLP** (<http://nlp.stanford.edu/software/>) : Another open source library
- **UIMA** (<https://uima.apache.org/>): An Apache project supporting pipelines
- **LingPipe** (<http://alias-i.com/lingpipe/>): A library that uses pipelines extensively
- **DL4J** (<http://deeplearning4j.org/>): The Deep Learning for Java library supports various classes for deep learning neural networks including support for NLP

We will use OpenNLP and DL4J to demonstrate text analysis in this chapter. We chose these because they are both well-known and have good published resources for additional support.

We will use the Google **Word2Vec** and **Doc2Vec** neural networks to perform text classification. This includes feature vectors based on other words as well as using labeled information to classify documents. Finally, we will discuss sentiment analysis. This type of analysis seeks to assign meaning to text and also uses the Word2Vec network.

We start our discussion with NER.

Implementing named entity recognition

This is sometimes referred to as finding people and things. Given a text segment, we may want to identify all the names of people present. However, this is not always easy because a name such as Rob may also be used as a verb.

In this section, we will demonstrate how to use OpenNLP's `TokenNameFinderModel` class to find names and locations in text. While there are other entities we may want to find, this example will demonstrate the basics of the technique. We begin with names.

Most names occur within a single line. We do not want to use multiple lines because an entity such as a state might inadvertently be identified incorrectly. Consider the following sentences:

Jim headed north. Dakota headed south.

If we ignored the period, then the state of North Dakota might be identified as a location, when in fact it is not present.

Using OpenNLP to perform NER

We start our example with a try-catch block to handle exceptions. OpenNLP uses models that have been trained on different sets of data. In this example, the `en-token.bin` and `en-ner-person.bin` files contain the models for the tokenization of English text and for English name elements, respectively. These files can be downloaded from <http://opennlp.sourceforge.net/models-1.5/>. However, the IO stream used here is standard Java:

```
try (InputStream tokenStream =
        new FileInputStream(new File("en-token.bin"));
      InputStream personModelStream = new FileInputStream(
        new File("en-ner-person.bin"))) {
    ...
} catch (Exception ex) {
    // Handle exceptions
}
```

An instance of the `TokenizerModel` class is initialized using the token stream. This instance is then used to create the actual `TokenizerME` tokenizer. We will use this instance to tokenize our sentence:

```
TokenizerModel tm = new TokenizerModel(tokenStream);
TokenizerME tokenizer = new TokenizerME(tm);
```

The `TokenNameFinderModel` class is used to hold a model for name entities. It is initialized using the person model stream. An instance of the `NameFinderME` class is created using this model since we are looking for names:

```
TokenNameFinderModel tnfM = new
    TokenNameFinderModel(personModelStream);
NameFinderME nf = new NameFinderME(tnfM);
```

To demonstrate the process, we will use the following sentence. We then convert it to a series of tokens using the tokenizer and `tokenizer` method:

```
String sentence = "Mrs. Wilson went to Mary's house for dinner.";
String[] tokens = tokenizer.tokenize(sentence);
```

The `Span` class holds information regarding the positions of entities. The `find` method will return the position information, as shown here:

```
Span[] spans = nf.find(tokens);
```

This array holds information about person entities found in the sentence. We then display this information as shown here:

```
for (int i = 0; i < spans.length; i++) {  
    out.println(spans[i] + " - " + tokens[spans[i].getStart()]);  
}
```

The output for this sequence is as follows. Notice that it identifies the last name of Mrs. Wilson but not the "Mrs.":

```
[1..2) person - Wilson  
[4..5) person - Mary
```

Once these entities have been extracted, we can use them for specialized analysis.

Identifying location entities

We can also find other types of entities such as dates and locations. In the following example, we find locations in a sentence. It is very similar to the previous person example, except that an `en-ner-location.bin` file is used for the model:

```
try (InputStream tokenStream =
      new FileInputStream("en-token.bin");
      InputStream locationModelStream = new FileInputStream(
          new File("en-ner-location.bin"))); {

    TokenizerModel tm = new TokenizerModel(tokenStream);
    TokenizerME tokenizer = new TokenizerME(tm);

    TokenNameFinderModel tnfm =
        new TokenNameFinderModel(locationModelStream);
    NameFinderME nf = new NameFinderME(tnfm);

    sentence = "Enid is located north of Oklahoma City.";
    String tokens[] = tokenizer.tokenize(sentence);

    Span spans[] = nf.find(tokens);

    for (int i = 0; i < spans.length; i++) {
        out.println(spans[i] + " - " +
                    tokens[spans[i].getStart()]);
    }
} catch (Exception ex) {
    // Handle exceptions
}
```

With the sentence defined previously, the model was only able to find the second city, as shown here. This likely due to the confusion that arises with the name `Enid` which is both the name of a city and a person's name:

[5..7) location - Oklahoma

Suppose we use the following sentence:

```
sentence = "Pond Creek is located north of Oklahoma City.;"
```

Then we get this output:

[1..2) location - Creek

[6..8) location - Oklahoma

Unfortunately, it has missed the town of Pond Creek. NER is a useful tool for many applications, but like many techniques, it is not always foolproof. The accuracy of the NER approach presented, and many of the other NLP examples, will vary depending on factors such as the accuracy of the model, the language being used, and the type of entity.

We may also be interested in how text can be classified. We will examine one approach in the next section.

Classifying text

Classifying text is an important part of machine learning and data science. We have to be able to classify text for a variety of applications, including document retrieval and web searches. It is often important to assign specific labels to the data before we can determine its usefulness for a particular application or search result.

In this chapter, we are going to demonstrate a technique involving the use of paragraph vectors and labeled data with DL4J classes. This example allows us to read in documents and, based on the text inside of the document, assign a label (or classification) to the document. We are also going to show an example of classifying text by similarity. This means we will match phrases and words that have similar structure. This example will also use DL4J.

Word2Vec and Doc2Vec

We will be using Word2Vec and Doc2Vec in a few examples in this chapter.

Word2Vec is a neural network with two layers used for text processing. Given a body of text, the network will provide feature vectors for the words contained in the text. These vectors are simply mathematical representations of the word features and can be numerically compared to other vectors. This comparison is often referred to as the distance between two words.

Word2Vec operates with the understanding that words can be classified by determining the probability that two words will occur together. Because of this methodology, Word2Vec can be used for more than classification of sentences. Any object or data that can be represented by text labels can be classified with this network.

Doc2Vec is an extension of Word2Vec. Rather than building vectors representing the features of individual words compared to other words, as Word2Vec does, this network compares words to given labels. The vectors are set up to represent the theme or overall meaning of a document. Our next example shows how these feature vectors are then associated with specific documents.

Classifying text by labels

In our first example using Doc2Vec, we will associate our documents with three labels: health, finance, and science. But before we can associate the data with labels, we have to define those labels and train our model to recognize the labels. Each label represents the meaning or classification of a particular piece of text.

In this example we will use sample documents, each pre-labelled with our categories: health, finance, or science. We will use these paragraphs to train our model and then, as in previous examples, use a set of test data to test our model. We will be using the files found at <https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-examples/src/main/resources/paravec>. We have based this example upon sample code written for DL4J, which can be found at <https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/nlp/paragraphvectors/ParagraphV>

First we need to set up some instance variables to use later in our code. We will be using a `ParagraphVectors` object to create our vectors, a `LabelAwareIterator` object to iterate through our data, and a `TokenizerFactory` object to tokenize our data:

```
ParagraphVectors pVect;
LabelAwareIterator iter;
TokenizerFactory tFact;
```

Then we will set up our `ClassPathResource`. This specifies the directory within our project that contains the data files to be classified. The first resource contains our labeled data used for training purposes. We then direct our iterator and tokenizer to use the resources specified as the `ClassPathResource`. We also specify that we will use the `CommonPreprocessor` to preprocess our data:

```
ClassPathResource resource = new
    ClassPathResource("paravec/labeled");

iter = new FileLabelAwareIterator.Builder()
    .addSourceFolder(resource.getFile())
    .build();

tFact = new DefaultTokenizerFactory();
tFact.setTokenPreProcessor(new CommonPreprocessor());
```

Next, we build our `ParagraphVectors`. This is where we specify the learning rate, batch size, and number of training epochs. We include our iterator and tokenizer in the setup process as well. Once we've built our `ParagraphVectors`, we call the `fit` method to train our model using the training data in the `paravec/labeled` directory:

```
pVect = new ParagraphVectors.Builder()
    .learningRate(0.025)
    .minLearningRate(0.001)
    .batchSize(1000)
    .epochs(20)
    .iterate(iter)
    .trainWordVectors(true)
    .tokenizerFactory(tFact)
    .build();

pVect.fit();
```

Now that we have trained our model, we can use our unlabeled data to test. We create a new `ClassPathResource` for our unlabeled data and create a new `FileLabelAwareIterator` as well:

```
ClassPathResource unlabeledText =
    new ClassPathResource("paravec/unlabeled");
FileLabelAwareIterator unlabeledIter =
    new FileLabelAwareIterator.Builder()
        .addSourceFolder(unlabeledText.getFile())
        .build();
```

The next step involves iterating through our unlabeled data and identifying the correct label for each document. We can generally expect that each document will fall into multiple labels but have a different weight, or percent match, for each. So, while one article may be mostly classified as a health article, it likely has enough information to be also classified, to a lesser degree, as a science article.

Next, we set up a `MeansBuilder` and `LabelSeeker` object. These classes access tables containing the relationships between words and labels, which we will use in our `ParagraphVectors`. The `InMemoryLookupTable` class provides access to a default table for word lookup:

```
MeansBuilder mBuilder =
    new MeansBuilder((InMemoryLookupTable<VocabWord>)
        pVect.getLookupTable(), tFact);
LabelSeeker lSeeker =
    new LabelSeeker(iter.getLabelsSource().getLabels(),
```

```
(InMemoryLookupTable<VocabWord>)
pVect.getLookupTable());
```

Finally, we iterate through our unlabeled documents. For each document, we will change the document into a vector and use our `LabelSeeker` to get the scores for each document. We log the scores for each document and print out the score with the appropriate labels:

```
while (unlabeledIter.hasNextDocument()) {
    LabelledDocument doc = unlabeledIter.nextDocument();
    INDArray docCentroid = mBuilder.documentAsVector(doc);
    List<Pair<String, Double>> scores =
        lSeeker.getScores(docCentroid);
    out.println("Document '" + doc.getLabel() +
        "' falls into the following categories: ");
    for (Pair<String, Double> score : scores) {
        out.println ("          " + score.getFirst() + ":" + " "
            + score.getSecond());
    }
}
```

The output from our preceding print statements is as follows:

```
Document 'finance' falls into the following categories:
finance: 0.2889593541622162
health: 0.11753179132938385
science: 0.021202782168984413
Document 'health' falls into the following categories:
finance: 0.059537000954151154
health: 0.27373185753822327
science: 0.07699354737997055
```

In each instance, our documents were classified properly, as demonstrated by the higher percentage assigned to the correct label category. This classification can be used in conjunction with other data analysis techniques to draw additional conclusions about the data contained in the files. Often text classification is an initial or early step in a data analysis process as documents are classified into groups for further analysis.

Classifying text by similarity

In this next example, we will match different text samples based on their structure and similarity. We will still be using the `ParagraphVectors` class we used in the previous example. To begin, download the `raw_sentences.txt` file from GitHub (<https://github.com/deeplearning4j/dl4j-examples/tree/master/dl4j-examples/src/main/resources>) and add it to your project. This file contains a list of sentences which we will read in, label, and then compare.

First, we set up our `ClassPathResource` and assign an iterator to handle our file data. We have used a `SentenceIterator` for this example:

```
ClassPathResource srcFile = new  
    ClassPathResource("/raw_sentences.txt");  
File file = srcFile.getFile();  
SentenceIterator iter = new BasicLineIterator(file);
```

Next, we will again use `TokenizerFactory` to tokenize our data. We also want to create a new `LabelsSource` object. This allows us to define the format of our sentence labels. We have chosen to prefix each line with `LINE_`:

```
TokenizerFactory tFact = new DefaultTokenizerFactory();  
tFact.setTokenPreProcessor(new CommonPreprocessor());  
LabelsSource labelFormat = new LabelsSource("LINE_");
```

Now we are ready to build our `ParagraphVectors`. Our setup process includes these methods: `minWordFrequency`, which specifies the minimum word frequency to use in the training corpus, and `iterations`, which specifies the number of iterations for each mini batch. We also set the number of epochs, the layer size, and the learning rate. Additionally, we include our `LabelsSource`, defined before, and our iterator and tokenizer. The `trainWordVectors` method specifies whether word and document representations should be built together. Finally, `sampling` determines whether subsampling should occur or not. We then call our `build` and `fit` methods:

```
ParagraphVectors vec = new ParagraphVectors.Builder()  
    .minWordFrequency(1)  
    .iterations(5)  
    .epochs(1)  
    .layerSize(100)  
    .learningRate(0.025)  
    .labelsSource(labelFormat)
```

```

>windowSize(5)
.iterate(iter)
.trainWordVectors(false)
.tokenizerFactory(tFact)
.sampling(0)
.build();

vec.fit();

```

Next, we will include some statements to evaluate the accuracy of our classifications. It is important to note that while the document itself starts at 1, the indexing process begins at 0. So, for example, line 9836 in the document will be associated with the label LINE_9835. We will first compare three sentences that should be classified as somewhat similar, and then two examples comparing dissimilar sentences. The `similarity` method takes two labels and returns the relative distance between them in the form of `double`:

```

double similar1 = vec.similarity("LINE_9835", "LINE_12492");
out.println("Comparing lines 9836 & 12493
('This is my house .')/'('This is my world .')
Similarity = " + similar1);

double similar2 = vec.similarity("LINE_3720", "LINE_16392");
out.println("Comparing lines 3721 & 16393
('This is my way .')/'('This is my work .')
Similarity = " + similar2);

double similar3 = vec.similarity("LINE_6347", "LINE_3720");
out.println("Comparing lines 6348 & 3721
('This is my case .')/'('This is my way .')
Similarity = " + similar3);

double dissimilar1 = vec.similarity("LINE_3720", "LINE_9852");
out.println("Comparing lines 3721 & 9853
('This is my way .')/'('We now have one .')
Similarity = " + dissimilar1);

double dissimilar2 = vec.similarity("LINE_3720", "LINE_3719");
out.println("Comparing lines 3721 & 3720
('This is my way .')/'('At first he says no .')
Similarity = " + dissimilar2);

```

The output of our print statements is shown as follows. Compare the result of the `similarity` method for the three similar sentences and the two dissimilar sentences.

Of particular note, the `similarity` method result for the last example, two very dissimilar sentences, returned a negative number. This implies a more significant disparity:

```
16:56:15.423 [main] INFO o.d.m.s.SequenceVectors - Epoch: [1]; Words
vectorized so far: [3171540]; Lines vectorized so far: [485810];
learningRate: [1.0E-4]
Comparing lines 9836 & 12493 ('This is my house .')/'This is my world .
') Similarity = 0.7641470432281494
Comparing lines 3721 & 16393 ('This is my way .')/'This is my work .')
Similarity = 0.7246013879776001
Comparing lines 6348 & 3721 ('This is my case .')/'This is my way .')
Similarity = 0.8988922834396362
Comparing lines 3721 & 9853 ('This is my way .')/'We now have one .')
Similarity = 0.5840312242507935
Comparing lines 3721 & 3720 ('This is my way .')/'At first he says no .
') Similarity = -0.6491150259971619
```

Although this example uses `ParagraphVectors` like our first classification example, this demonstrates flexibility in our approach. We can use these DL4J libraries to classify data in more than one manner.

Understanding tagging and POS

POS is concerned with identifying the types of components found in a sentence. For example, this sentence has several elements, including the verb "has", several nouns such as "example" and "elements", and adjectives such as "several". Tagging, or more specifically **POS tagging**, is the process of associating element types to words.

POS tagging is useful as it adds more information about the sentence. We can ascertain the relationship between words and often their relative importance. The results of tagging are often used in later processing steps.

This task can be difficult as we are unable to rely upon a simple dictionary of words to determine their type. For example, the word `lead` can be used as both a noun and as a verb. We might use it in either of the following two sentences:

`He took the lead in the play.`
`Lead the way!`

POS tagging will attempt to associate the proper label to each word of a sentence.

Using OpenNLP to identify POS

To illustrate this process, we will be using OpenNLP (<https://opennlp.apache.org/>). This is an open source Apache project which supports many other NLP processing tasks.

We will be using the `POSModel` class, which can be trained to recognize POS elements. In this example, we will use it with a previously trained model based on the **Penn TreeBank tag-set**

(<http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html>). Various pretrained models are found at <http://opennlp.sourceforge.net/models-1.5/>. We will be using the `en-pos-maxent.bin` model. This has been trained on English text using what is called maximum entropy.

Maximum entropy refers to the amount of uncertainty in the model which it maximizes. For a given problem there is a set of probabilities describing what is known about the data set. These probabilities are used to build a model. For example, we may know that there is a 23 percent chance that one specific event may follow a certain condition. We do not want to make any assumptions about unknown probabilities so we avoid adding unjustified information. A maximum entropy approach attempts to preserve as much uncertainty as possible; hence it attempts to maximize entropy.

We will also use the `POSTaggerME` class, which is a maximum entropy tagger. This is the class that will make tag predictions. With any sentence, there may be more than one way of classifying, or tagging, its components.

We start with code to acquire the previously trained English tagger model and a simple sentence to be tagged:

```
try (InputStream input = new FileInputStream(
        new File("en-pos-maxent.bin")));
{
    String sentence = "Let's parse this sentence.";
    ...
} catch (IOException ex) {
    // Handle exceptions
}
```

The tagger uses an array of strings, where each string is a word. The following sequence takes the previous sentence and creates an array called `words`. The first part

uses the `Scanner` class to parse the sentence string. We could have used other code to read the data from a file if needed. After that, the `List` class's `toArray` method is used to create the array of strings:

```
List<String> list = new ArrayList<>();
Scanner scanner = new Scanner(sentence);
while(scanner.hasNext()) {
    list.add(scanner.next());
}
String[] words = new String[1];
words = list.toArray(words);
```

The model is then built using the file containing the model:

```
POSModel posModel = new POSModel(input);
```

The tagger is then created based on the model:

```
POSTaggerME postagger = new POSTaggerME(posModel);
```

The `tag` method does the actual work. It is passed an array of words and returns an array of tags. The words and tags are then displayed:

```
String[] postags = postagger.tag(words);
for(int i=0; i<postags.length; i++) {
    out.println(words[i] + " - " + postags[i]);
}
```

The output for this example follows:

```
Let's - NNP
parse - NN
this - DT
sentence. - NN
```

The analysis has determined that the word `let's` is a singular proper noun while the words `parse` and `sentence` are singular nouns. The word `this` is a determiner, that is, it is a word that modifies another and helps identify a phrase as general or specific. A list of tags is provided in the next section.

Understanding POS tags

The POS elements returned abbreviations. A list of **Penn TreeBankPOS** tags can be found at

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

The following is a shortened version of this list:

Tag	Description	Tag	Description
DT	Determiner	RB	Adverb
JJ	Adjective	RBR	Adverb, comparative
JJR	Adjective, comparative	RBS	Adverb, superlative
JJS	Adjective, superlative	RP	Particle
NN	Noun, singular or mass	SYM	Symbol
NNS	Noun, plural	TOP	Top of the parse tree
NNP	Proper noun, singular	VB	Verb, base form
NNPS	Proper noun, plural	VBD	Verb, past tense
POS	Possessive ending	VBG	Verb, gerund or present participle
PRP	Personal pronoun	VBN	Verb, past participle
PRP\$	Possessive pronoun	VBP	Verb, non-3rd person singular present
S	Simple declarative clause	VBZ	Verb, 3rd person singular present

As mentioned earlier, there may be more than one possible set of POS assignments for a sentence. The `topKSequences` method, as shown next, will return various assignment possibilities along with a score. The method returns an array of `Sequence` objects whose `toString` method returns the score and POS list:

```
Sequence sequences[] = postagger.topKSequences(words);
for(Sequence sequence : sequences) {
```

```
        out.println(sequence);
    }
```

The output for the previous sentence follows, where the last sequence is considered to be the most probable alternative:

```
-2.3264880694837213 [NNP, NN, DT, NN]
-2.6610271245387853 [NNP, VBD, DT, NN]
-2.6630142638557217 [NNP, VB, DT, NN]
```

Each line of output assigns possible tags to each word of the sentence. We can see that only the second word, `parse`, is determined to have other possible tags.

Next, we will demonstrate how to extract relationships from text.

Extracting relationships from sentences

Knowing the relationship between elements of a sentence is important in many analysis tasks. It is useful for assessing the important content of a sentence and providing insight into the meaning of a sentence. This type of analysis has been used for tasks ranging from grammar checking to speech recognition to language translations.

In the previous section, we demonstrated one approach used to extract the parts of speech. Using this technique, we were able to identify the sentence element types present in a sentence. However, the relationships between these elements is missing. We need to parse the sentence to extract these relationships between sentence elements.

Using OpenNLP to extract relationships

There are several techniques and APIs that can be used to extract this type of information. In this section we will use OpenNLP to demonstrate one way of extracting the structure of a sentence. The demonstration is centered around the `ParserTool` class, which uses a previously trained model. The parsing process will return the probabilities that the sentence's elements extracted are correct. As will many NLP tasks, there are often multiple answers possible.

We start with a try-with-resource block to open an input stream for the model. The `en-parser-chunking.bin` file contains a model that uses parses text into its POS. In this case, it is trained for English:

```
try (InputStream modelInputStream = new FileInputStream(
        new File("en-parser-chunking.bin")));
{
    ...
} catch (Exception ex) {
    // Handle exceptions
}
```

Within the try block an instance of the `ParserModel` class is created using the input stream. The actual parser is created next using the `ParserFactory` class's `create` method:

```
ParserModel parserModel = new ParserModel(modelInputStream);
Parser parser = ParserFactory.create(parserModel);
```

We will use the following sentence to test the parser. The `ParserTool` class's `parseLine` method does the actual parsing and returns an array of `Parse` objects. Each of these objects holds one parsing alternative. The last argument of the `parseLine` method specifies how many alternatives to return:

```
String sentence = "Let's parse this sentence.";
Parse[] parseTrees = ParserTool.parseLine(sentence, parser, 3);
```

The next sequence displays each of the possibilities:

```
for(Parse tree : parseTrees) {
    tree.show();
}
```

The output of the `show` method for this example follows. The tags were previously

defined in *Understanding POS tags* section:

```
(TOP (NP (NP (NNP Let's) (NN parse)) (NP (DT this) (NN sentence.))))  
(TOP (S (NP (NNP Let's)) (VP (VB parse) (NP (DT this) (NN  
sentence.))))  
(TOP (S (NP (NNP Let's)) (VP (VBD parse) (NP (DT this) (NN  
sentence.))))
```

The following example reformats the last two outputs to better show the relationships. They differ in how they classify the verb parse:

```
(TOP  
(S  
(NP (NNP Let's))  
(VP (VB parse)  
(NP (DT this) (NN sentence.))  
)  
)  
)  
(TOP  
(S  
(NP (NNP Let's))  
(VP (VBD parse)  
(NP (DT this) (NN sentence.))  
)  
)  
)
```

When there are multiple parse alternatives, the `Parse` class's `getProb` returns a probability that reflects the model's confidence in the alternatives. The following sequence demonstrates this method:

```
for(Parse tree : parseTrees) {  
    out.println("Probability: " + tree.getProb());  
}
```

The output follows:

```
Probability: -3.6810244423259078  
Probability: -3.742475884515823  
Probability: -4.16148634555491
```

Another interesting NLP task is sentiment analysis, which we will demonstrate next.

Sentiment analysis

Sentiment analysis involves the evaluation and classification of words based on their context, meaning, and emotional implications. Typically, if we were to look up a word in a dictionary we will find a meaning or definition for the word but, taken out of the context of a sentence, we may not be able to ascribe detailed and precise meaning to the word.

For example, the word *toast* could be defined as simply a slice of heated and browned bread. But in the context of the sentence *He's toast!*, the meaning changes completely. Sentiment analysis seeks to derive meanings of words based on their context and usage.

It is important to note that advanced sentiment analysis will expand beyond simple positive or negative classification and ascribe detailed emotional meaning to words. It is far simpler to classify words as positive or negative but far more useful to classify them as happy, furious, indifferent, or anxious.

This type of analysis falls into the category of effective computing, a type of computing interested in the emotional implications and uses of technological tools. This type of computing is especially significant given the growing amount of emotionally influenced data readily available for analysis on social media sites today.

Being able to determine the emotional content of text enables a more targeted, and appropriate response. For example, being able to judge the emotional response in a chat session between a customer and technical representative can allow the representative to do a better job. This can be especially important when there is a cultural or language gap between them.

This type of analysis can also be applied to visual images. It could be used to gauge someone's response to a new product, such as when conducting a taste test, or to judge how people react to scenes of a movie or commercial.

As part of our example we will be using a bag-of-words model. Bag-of-words models simplify word representation for natural language processing by containing a set, known as the **bag**, of words irrespective of grammar or word order. The words have features used for classification, most importantly the frequency of each word.

Because some words such as the, a, or and will naturally have a higher frequency in any text, the words are given a weight as well. Common words with less contextual significance will have a smaller weight and factor less into the text analysis.

Downloading and extracting the Word2Vec model

To demonstrate sentiment analysis, we will use Google's Word2Vec models in conjunction with DL4J to simply classify movie reviews as either positive or negative based upon the words used in the review. This example is adapted from work done by Alex Black (<https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/recurrent/word2vecsentiment/Word2VecSentiment.java>). As discussed previously in this chapter, Word2Vec consists of two-layer neural networks trained to build meaning from the context of words. We will also be using a large set of movie reviews from <http://ai.stanford.edu/~amaas/data/sentiment/>.

Before we begin, you will need to download the Word2Vec data from <https://code.google.com/p/word2vec/>. The basic process includes:

- Downloading and extracting the movie reviews
- Loading the Word2Vec Google News vectors
- Loading each movie review

The words within the reviews are then broken into vectors and used to train the network. We will train the network across five epochs and evaluate the network's performance after each epoch.

To begin, we first declare three final variables. The first is the URL to retrieve the training data, the second is the location to store our extracted data, and the third is the location of the Google News vectors on the local machine. Modify this third variable to reflect the location on your local machine:

```
public static final String TRAINING_DATA_URL =
    "http://ai.stanford.edu/~amaas/" +
    "data/sentiment/aclImdb_v1.tar.gz";
public static final String EXTRACT_DATA_PATH =
    FilenameUtils.concat(System.getProperty(
        "java.io.tmpdir"), "dl4j_w2vSentiment/");
public static final String GNEWS_VECTORS_PATH =
    "C:/YOUR_PATH/GoogleNews-vectors-negative300.bin" +
    "/GoogleNews-vectors-negative300.bin";
```

Next we download and extract our model data. The next two methods are modelled

after the code found in the DL4J example. We first create a new method, `getModelData`. The method is shown next in its entirety.

First we create a new `File` using the `EXTRACT_DATA_PATH` we defined previously. If the file does not already exist, we create a new directory. Next, we create two more `File` objects, one for the path to the archived TAR file and one for the path to the extracted data. Before we attempt to extract the data, we check whether these two files exist. If the archive path does not exist, we download the data from the `TRAINING_DATA_URL` and then extract the data. If the extracted file does not exist, we then extract the data:

```
private static void getModelData() throws Exception {
    File modelDir = new File(EXTRACT_DATA_PATH);
    if (!modelDir.exists()) {
        modelDir.mkdir();
    }
    String archivePath = EXTRACT_DATA_PATH + "aclImdb_v1.tar.gz";
    File archiveName = new File(archivePath);
    String extractPath = EXTRACT_DATA_PATH + "aclImdb";
    File extractName = new File(extractPath);
    if (!archiveName.exists()) {
        FileUtils.copyURLToFile(new URL(TRAINING_DATA_URL),
                               archiveName);
        extractTar(archivePath, EXTRACT_DATA_PATH);
    } else if (!extractName.exists()) {
        extractTar(archivePath, EXTRACT_DATA_PATH);
    }
}
```

To extract our data, we will create another method called `extractTar`. We will provide two inputs to the method, the `archivePath` and the `EXTRACT_DATA_PATH` defined before. We also need to define our buffer size to use in the extraction process:

```
private static final int BUFFER_SIZE = 4096;
```

We first create a new `TarArchiveInputStream`. We use the `GzipCompressorInputStream` because it provides support for extracting `.gz` files. We also use the `BufferedInputStream` to improve performance in our extraction process. The compressed file is very large and may take some time to download and extract.

Next we create a `TarArchiveEntry` and begin reading in data using the `TarArchiveInputStream getNextEntry` method. As we process entry in the compressed file, we first check whether the entry is a directory. If it is, we create a new directory in our extraction location. Finally we create a new `FileOutputStream` and `BufferedOutputStream` and use the `write` method to write our data in the extracted location:

```
private static void extractTar(String dataIn, String dataOut)
    throws IOException {
    try (TarArchiveInputStream inStream =
        new TarArchiveInputStream(
            new GzipCompressorInputStream(
                new BufferedInputStream(
                    new FileInputStream(dataIn))))) {
        TarArchiveEntry tarFile;
        while ((tarFile = (TarArchiveEntry)
inStream.getNextEntry())
            != null) {
            if (tarFile.isDirectory()) {
                new File(dataOut + tarFile.getName()).mkdirs();
            } else {
                int count;
                byte data[] = new byte[BUFFER_SIZE];
                FileOutputStream fileInStream =
                    new FileOutputStream(dataOut +
tarFile.getName());
                BufferedOutputStream outStream =
                    new BufferedOutputStream(fileInStream,
                        BUFFER_SIZE);
                while ((count = inStream.read(data, 0,
BUFFER_SIZE))
                    != -1) {
                    outStream.write(data, 0, count);
                }
            }
        }
    }
}
```

Building our model and classifying text

Now that we have created methods to download and extract our data, we need to declare and initialize variables used to control the execution of our model. Our `batchSize` refers to the amount of words we process in each example, in this case 50. Our `vectorSize` determines the size of the vectors. The Google News model has word vectors of size 300. `nEpochs` refers to the number of times we attempt to run through our training data. Finally, `truncateReviewsToLength` specifies whether, for memory utilization purposes, we should truncate the movie reviews if they exceed a specific length. We have chosen to truncate reviews longer than 300 words:

```
int batchSize = 50;
int vectorSize = 300;
int nEpochs = 5;
int truncateReviewsToLength = 300;
```

Now we can set up our neural network. We will use a `MultiLayerConfiguration` network, as discussed in [Chapter 8, Deep Learning](#). In fact, our example here is very similar to the model built in configuring and building a model, with a few differences. In particular, in this model we will use a faster learning rate and a `GravesLSTM` recurrent network in layer 0. We will have the same number of input neurons as we have words in our vector, in this case, 300. We also use `gradientNormalization`, a technique used to help our algorithm find the optimal solution. Notice we are using the `softmax` activation function, which was discussed in [Chapter 8, Deep Learning](#). This function uses regression and is especially suited for classification algorithms:

```
MultiLayerConfiguration sentimentNN =
    new NeuralNetConfiguration.Builder()
        .optimizationAlgo(OptimizationAlgorithm
            .STOCHASTIC_GRADIENT_DESCENT).iterations(1)
        .updater(Updater.RMSPROP)
        .regularization(true).l2(1e-5)
        .weightInit(WeightInit.XAVIER)
        .gradientNormalization(GradientNormalization
            .ClipElementWiseAbsoluteValue)
            .gradientNormalizationThreshold(1.0)
        .learningRate(0.0018)
        .list()
        .layer(0, new GravesLSTM.Builder()
            .nIn(vectorSize).nOut(200)
            .activation("softsign").build())
```

```

    .layer(1, new RnnOutputLayer.Builder()
        .activation("softmax")
        .lossFunction(LossFunctions.LossFunction.MCXENT)
        .nIn(200).nOut(2).build())
    .pretrain(false).backprop(true).build();
}

```

We can then create our `MultiLayerNetwork`, initialize the network, and set listeners.

```

MultiLayerNetwork net = new MultiLayerNetwork(sentimentNN);
net.init();
net.setListeners(new ScoreIterationListener(1));

```

Next we create a `WordVectors` object to load our Google data. We use a `DataSetIterator` to test and train our data. The `AsyncDataSetIterator` allows us to load our data in a separate thread, to improve performance. This process requires a large amount of memory and so improvements such as this are essential for optimal performance:

```

WordVectors wordVectors = WordVectorSerializer
DataSetIterator trainData = new AsyncDataSetIterator(
    new SentimentExampleIterator(EXTRACT_DATA_PATH, wordVectors,
        batchSize, truncateReviewsToLength, true), 1);
DataSetIterator testData = new AsyncDataSetIterator(
    new SentimentExampleIterator(EXTRACT_DATA_PATH, wordVectors,
        100, truncateReviewsToLength, false), 1);

```

Finally, we are ready to train and evaluate our data. We run through our data `nEpochs` times; in this case, we have five iterations. Each iteration executes the `fit` method against our training data and then creates a new `Evaluation` object to evaluate our model using `testData`. The evaluation is based on around 25,000 movie reviews and can take a significant amount of time to run. As we evaluate the data, we create `INDArray` to store information, including the feature matrix and labels from our data. This data is used later in the `evalTimeSeries` method for evaluation. Finally, we print out our evaluation statistics:

```

for (int i = 0; i < nEpochs; i++) {
    net.fit(trainData);
    trainData.reset();

    Evaluation evaluation = new Evaluation();
    while (testData.hasNext()) {
        DataSet t = testData.next();
        INDArray dataFeatures = t.getFeatureMatrix();

```

```

INDArray dataLabels = t.getLabels();
INDArray inMask = t.getFeaturesMaskArray();
INDArray outMask = t.getLabelsMaskArray();
INDArray predicted = net.output(dataFeatures, false,
    inMask, outMask);

evaluation.evalTimeSeries(dataLabels, predicted, outMask);
}
testData.reset();

out.println(evaluation.stats());
}

```

The output from the final iteration is shown next. Our examples classified as 0 are considered negative reviews and the ones classified as 1 are considered positive reviews:

```

Epoch 4 complete. Starting evaluation:
Examples labeled as 0 classified by model as 0: 11122 times
Examples labeled as 0 classified by model as 1: 1378 times
Examples labeled as 1 classified by model as 0: 3193 times
Examples labeled as 1 classified by model as 1: 9307 times
=====Scores=====Accu
racy: 0.8172
Precision: 0.824
Recall: 0.8172
F1 Score: 0.8206
=====

```

If compared with previous iterations, you should notice the score and accuracy improving with each evaluation. With each iteration, our model improves its accuracy in classifying movie reviews as either negative or positive.

Summary

In this chapter, we introduced a number of NLP tasks and showed how they are supported. In particular, we used OpenNLP and DL4J to illustrate how they are performed. While there are a number of other libraries available, these examples provide a good introduction to the techniques.

We started with an introduction to basic NLP terms and concepts such as named entity recognition, POS, and relationships between elements of a sentence. Named entity recognition is concerned with finding and labeling the parts of a sentence such as people, locations, and things. POS associates labels with elements of a sentence. For example, `NN` refers to a noun and `VB` to a verb.

We then included a discussion of the Word2Vec and Doc2Vec neural networks. These were used to classify text, both with labels and by similarity with other words. We demonstrated the use of DL4J resources to create feature vectors for document association with labels.

While the identification of these associations is interesting, a more useful analysis is performed when relationships are extracted from a sentence. We demonstrated how relationships are found using OpenNLP. The POS are associated with each word and the relationships between the words are shown using a set of tags and parentheses. This type of analysis can be used for more sophisticated analyses such as language translation and grammar checking.

Finally, we discussed and showed examples of sentiment analysis. This process involves classifying text based on its tone or contextual meaning. We examined a process for classifying movie reviews as positive or negative.

In this chapter, we demonstrated various techniques for text analysis and classification. In our next chapter, we will examine techniques designed for video and audio analysis.

Chapter 10. Visual and Audio Analysis

The use of sound, images, and videos is becoming a more important aspect of our day-to-day lives. Phone conversations and devices reliant on voice commands are increasingly common. People regularly conduct video chats with other people around the world. There has been a rapid proliferation of photo and video sharing sites. Applications that utilize images, video, and sound from a variety of sources are becoming more common.

In this chapter, we will demonstrate several techniques available to Java to process sounds and images. The first part of the chapter addresses sound processing. Both speech recognition and **Text-To-Speech (TTS)** APIs will be demonstrated.

Specifically, we will use the FreeTTS (<http://freetts.sourceforge.net/docs/index.php>) API to convert text to speech, followed with a demonstration of the CMU Sphinx toolkit for speech recognition.

The **Java Speech API (JSAPI)** (<http://www.oracle.com/technetwork/java/index-140170.html>) provides access to speech technology. It is not part of the standard JDK but is supported by third-party vendors. Its intent is to support speech recognition and speech synthesizers. There are several vendors that support JSAPI, including FreeTTS and Festival (<http://www.cstr.ed.ac.uk/projects/festival/>).

In addition, there are several cloud-based speech APIs, including IBM's support through **Watson Cloud** speech-to-text capabilities.

Next, we will examine image processing techniques, including facial recognition. This involves identifying faces within an image. This technique is easy to accomplish using OpenCV (<http://opencv.org/>) which we will demonstrate in the Identifying faces section.

We will end the chapter with a discussion of Neuroph Studio, a neural network Java-based editor, to classify images and perform image recognition. We will continue to use faces here and attempt to train a network to recognize images of human faces.

Text-to-speech

Speech synthesis generates human speech. TTS converts text to speech and is useful for a number of different applications. It is used in many places, including phone help desk systems and ordering systems. The TTS process typically consists of two parts. The first part tokenizes and otherwise processes the text into speech units. The second part converts these units into speech.

The two primary approaches for TTS uses **concatenation synthesis** and **formant synthesis**. Concatenation synthesis frequently combines prerecorded human speech to create the desired output. Formant synthesis does not use human speech but generates speech by creating electronic waveforms.

We will be using FreeTTS (<http://freetts.sourceforge.net/docs/index.php>) to demonstrate TTS. The latest version can be downloaded from <https://sourceforge.net/projects/freetts/files/>. This approach uses concatenation to generate speech.

There are several important terms used in TTS/FreeTTS:

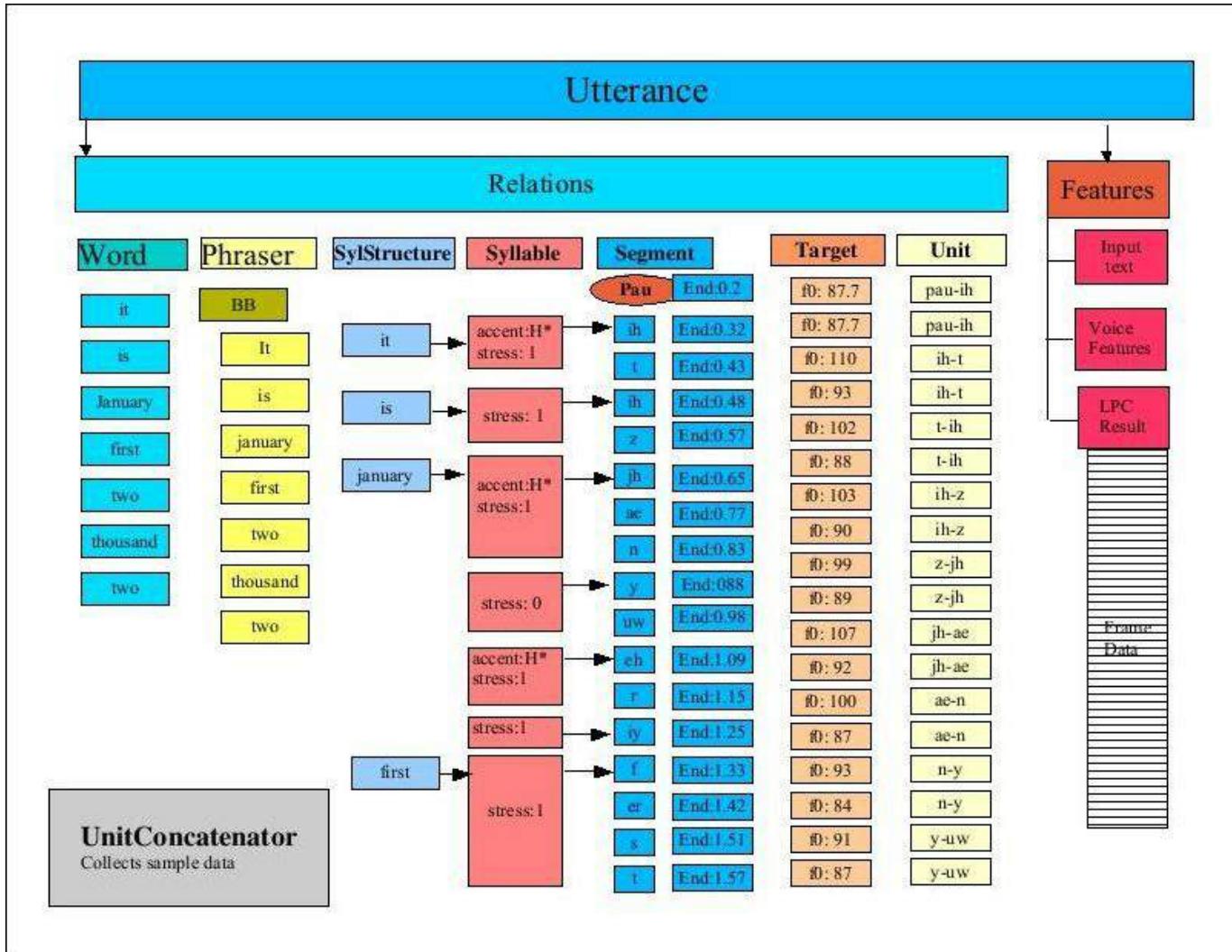
- **Utterance** - This concept corresponds roughly to the vocal sounds that make up a word or phrase
- **Items** - Sets of features (name/value pairs) that represent parts of an utterance
- **Relationship** - A list of items, used by FreeTTS to iterate back and forward through an utterance
- **Phone** - A distinct sound
- **Diphone** - A pair of adjacent phones

The FreeTTS Programmer's Guide (<http://freetts.sourceforge.net/docs/ProgrammerGuide.html>) details the process of converting text to speech. This is a multi-step process whose major steps include the following:

- **Tokenization** - Extracting the tokens from the text
- **TokenToWords** - Converting certain words, such as 1910 to nineteen ten
- **PartOfSpeechTagger** - This step currently does nothing, but is intended to identify the parts of speech
- **Phraser** - Creates a phrase relationship for the utterance
- **Segmenter** - Determines where syllable breaks occur

- **PauseGenerator** - This step inserts pauses within speech, such as before utterances
- **Intonator** - Determines the accents and tones
- **PostLexicalAnalyzer** - This step fixes problems such as a mismatch between the available diphones and the one that needs to be spoken
- **Durator** - Determines the duration of the syllables
- **ContourGenerator** - Calculates the fundamental frequency curve for an utterance, which maps the frequency against time and contributes to the generation of tones
- **UnitSelector** - Groups related diphones into a unit
- **PitchMarkGenerator** - Determines the pitches for an utterance
- **UnitConcatenator** - Concatenates the diphone data together

The following figure is from the *FreeTTS Programmer's Guide, Figure 11*: The **Utterance** after **UnitConcatenator** processing, and depicts the process. This high-level overview of the TTS process provides a hint at the complexity of the process:



Using FreeTTS

TTS system facilitates the use of different voices. For example, these differences may be in the language, the sex of the speaker, or the age of the speaker.

The **MBROLA Project's** (<http://tcts.fpms.ac.be/synthesis/mbrola.html>) objective is to support voice synthesizers for as many languages as possible. MBROLA is a speech synthesizer that can be used with a TTS system such as FreeTTS to support TTS synthesis.

Download MBROLA for the appropriate platform binary from <http://tcts.fpms.ac.be/synthesis/mbrola.html>. From the same page, download any desired MBROLA voices found at the bottom of the page. For our examples we will use `usa1`, `usa2`, and `usa3`. Further details about the setup are found at <http://freetts.sourceforge.net/mbrola/README.html>.

The following statement illustrates the code needed to access the MBROLA voices. The `setProperty` method assigns the path where the MBROLA resources are found:

```
System.setProperty("mbrola.base", "path-to-mbrola-directory");
```

To demonstrate how to use TTS, we use the following statement. We obtain an instance of the `VoiceManager` class, which will provide access to various voices:

```
VoiceManager voiceManager = VoiceManager.getInstance();
```

To use a specific voice the `getVoice` method is passed the name of the voice and returns an instance of the `Voice` class. In this example, we used `mbrola_us1`, which is a US English, young, female voice:

```
Voice voice = voiceManager.getVoice("mbrola_us1");
```

Once we have obtained the `Voice` instance, use the `allocate` method to load the voice. The `speak` method is then used to synthesize the words passed to the method as a string, as illustrated here:

```
voice.allocate();
voice.speak("Hello World");
```

When executed, the words "Hello World" should be heard. Try this with other voices, as described in the next section, and text to see which combination is best

suited for an application.

Getting information about voices

The `VoiceManager` class' `getVoices` method is used to obtain an array of the voices currently available. This can be useful to provide users with a list of voices to choose from. We will use the method here to illustrate some of the voices available. In the next code sequence, the method returns the array, whose elements are then displayed:

```
Voice[] voices = voiceManager.getVoices();  
for (Voice v : voices) {  
    out.println(v);  
}
```

The output will be similar to the following:

```
CMUClusterUnitVoice  
CMUDiphoneVoice  
CMUDiphoneVoice  
MbrolaVoice  
MbrolaVoice  
MbrolaVoice
```

The `getVoiceInfo` method provides potentially more useful information, though it is somewhat verbose:

```
out.println(voiceManager.getVoiceInfo());
```

The first part of the output follows; the `VoiceDirectory` directory is displayed followed by the details of the voice. Notice that the directory name contains the name of the voice. The `KevinVoiceDirectory` contains two voices: `kevin` and `kevin16`:

```
VoiceDirectory  
'com.sun.speech.freetts.en.us.cmu_time_awb.AlanVoiceDirectory'  
Name: alan  
Description: default time-domain cluster unit voice  
Organization: cmu  
Domain: time  
Locale: en_US  
Style: standard  
Gender: MALE  
Age: YOUNGER_ADULT  
Pitch: 100.0  
Pitch Range: 12.0
```

```
Pitch Shift: 1.0
Rate: 150.0
Volume: 1.0
VoiceDirectory
'com.sun.speech.freetts.en.us.cmu_us_kal.KevinVoiceDirectory'
Name: kevin
Description: default 8-bit diphone voice
Organization: cmu
Domain: general
Locale: en_US
Style: standard
Gender: MALE
Age: YOUNGER_ADULT
Pitch: 100.0
Pitch Range: 11.0
Pitch Shift: 1.0
Rate: 150.0
Volume: 1.0
Name: kevin16
Description: default 16-bit diphone voice
Organization: cmu
Domain: general
Locale: en_US
Style: standard
Gender: MALE
Age: YOUNGER_ADULT
Pitch: 100.0
Pitch Range: 11.0
Pitch Shift: 1.0
Rate: 150.0
Volume: 1.0
...
Using voices from a JAR file
```

Voices can be stored in JAR files. The `VoiceDirectory` class provides access to voices stored in this manner. The voice directories available to FreeTTs are found in the `lib` directory and include the following:

- `cmu_time_awb.jar`
- `cmu_us_kal.jar`

The name of a voice directory can be obtained from the command prompt:

```
java -jar fileName.jar
```

For example, execute the following command:

```
java -jar cmu_time_awb.jar
```

It generates the following output:

```
VoiceDirectory
'com.sun.speech.freetts.en.us.cmu_time_awb.AlanVoiceDirectory'
Name: alan
Description: default time-domain cluster unit voice
Organization: cmu
Domain: time
Locale: en_US
Style: standard
Gender: MALE
Age: YOUNGER_ADULT
Pitch: 100.0
Pitch Range: 12.0
Pitch Shift: 1.0
Rate: 150.0
Volume: 1.0
```

Gathering voice information

The `Voice` class provides a number of methods that permit the extraction or setting of speech characteristics. As we demonstrated earlier, the `VoiceManager` class' `getVoiceInfo` method provided information about the voices currently available. However, we can use the `Voice` class to get information about a specific voice.

In the following example, we will display information about the voice `kevin16`. We start by getting an instance of this voice using the `getVoice` method:

```
VoiceManager vm = VoiceManager.getInstance();
Voice voice = vm.getVoice("kevin16");
voice.allocate();
```

Next, we call a number of the `Voice` class' `get` method to obtain specific information about the voice. This includes previous information provided by the `getVoiceInfo` method and other information that is not otherwise available:

```
out.println("Name: " + voice.getName());
out.println("Description: " + voice.getDescription());
out.println("Organization: " + voice.getOrganization());
out.println("Age: " + voice.getAge());
out.println("Gender: " + voice.getGender());
out.println("Rate: " + voice.getRate());
out.println("Pitch: " + voice.getPitch());
out.println("Style: " + voice.getStyle());
```

The output of this example follows:

```
Name: kevin16
Description: default 16-bit diphone voice
Organization: cmu
Age: YOUNGER_ADULT
Gender: MALE
Rate: 150.0
Pitch: 100.0
Style: standard
```

These results are self-explanatory and give you an idea of the type of information available. There are additional methods that give you access to details regarding the TTS process that are not normally of interest. This includes information such as the audio player being used, utterance-specific data, and features of a specific phone.

Having demonstrated how text can be converted to speech, we will now examine how we can convert speech to text.

Understanding speech recognition

Converting speech to text is an important application feature. This ability is increasingly being used in a wide variety of contexts. Voice input is used to control smart phones, automatically handle input as part of help desk applications, and to assist people with disabilities, to mention a few examples.

Speech consists of an audio stream that is complex. Sounds can be split into **phones**, which are sound sequences that are similar. Pairs of these phones are called **diphones**. **Utterances** consist of words and various types of pauses between them.

The essence of the conversion process involves splitting sounds by silences between utterances. These utterances are then matched to the words that most closely sound like the utterance. However, this can be difficult due to many factors. For example, these differences may be in the form of variances in how words are pronounced due to the context of the word, regional dialects, the quality of the sound, and other factors.

The matching process is quite involved and often uses multiple models. A model may be used to match acoustic features with a sound. A phonetic model can be used to match phones to words. Another model is used to restrict word searches to a given language. These models are never entirely accurate and contribute to inaccuracies found in the recognition process.

We will be using CMUSphinx 4 to illustrate this process.

Using CMUPhinx to convert speech to text

Audio processed by CMUSphinx must be in **Pulse Code Modulation (PCM)** format. PCM is a technique that samples analog data, such as an analog wave representing speech, and produces a digital version of the signal. FFmpeg (<https://ffmpeg.org/>) is a free tool that can convert between audio formats if needed.

You will need to create sample audio files using the PCM format. These files should be fairly short and can contain numbers or words. It is recommended that you run the examples with different files to see how well the speech recognition works.

First, we set up the basic framework for the conversion by creating a try-catch block to handle exceptions. First, create an instance of the `Configuration` class. It is used to configure the recognizer to recognize standard English. The configuration models and dictionary need to be changed to handle other languages:

```
try {
    Configuration configuration = new Configuration();
    String prefix = "resource:/edu/cmu/sphinx/models/en-us/";
    configuration
        .setAcousticModelPath(prefix + "en-us");
    configuration
        .setDictionaryPath(prefix + "cmudict-en-us.dict");
    configuration
        .setLanguageModelPath(prefix + "en-us.lm.bin");
    ...
} catch (IOException ex) {
    // Handle exceptions
}
```

The `StreamSpeechRecognizer` class is then created using `configuration`. This class processes the speech based on an input stream. In the following code, we create an instance of the `StreamSpeechRecognizer` class and an `InputStream` from the speech file:

```
StreamSpeechRecognizer recognizer = new StreamSpeechRecognizer(
    configuration);
InputStream stream = new FileInputStream(new File("filename"));
```

To start speech processing, the `startRecognition` method is invoked. The `getResult` method returns a `SpeechResult` instance that holds the result of the processing. We then use the `SpeechResult` method to get the best results. We stop

the processing using the `stopRecognition` method:

```
recognizer.startRecognition(stream);
SpeechResult result;
while ((result = recognizer.getResult()) != null) {
    out.println("Hypothesis: " + result.getHypothesis());
}
recognizer.stopRecognition();
```

When this is executed, we get the following, assuming the speech file contained this sentence:

```
Hypothesis: mary had a little lamb
```

When speech is interpreted there may be more than one possible word sequence. We can obtain the best ones using the `getNbest` method, whose argument specifies how many possibilities should be returned. The following demonstrates this method:

```
Collection<String> results = result.getNbest(3);
for (String sentence : results) {
    out.println(sentence);
}
```

One possible output follows:

```
<s> mary had a little lamb </s>
<s> marry had a little lamb </s>
<s> mary had a a little lamb </s>
```

This gives us the basic results. However, we will probably want to do something with the actual words. The technique for getting the words is explained next.

Obtaining more detail about the words

The individual words of the results can be extracted using the `getWords` method, as shown next. The method returns a list of `WordResult` instance, each of which represents one word:

```
List<WordResult> words = result.getWords();  
for (WordResult wordResult : words) {  
    out.print(wordResult.getWord() + " ");  
}
```

The output for this code sequence follows <sil> reflects a silence found at the beginning of the speech:

```
<sil> mary had a little lamb
```

We can extract more information about the words using various methods of the `WordResult` class. In this sequence that follows, we will return the confidence and time frame associated with each word.

The `getConfidence` method returns the confidence expressed as a log. We use the `SpeechResult` class' `getResult` method to get an instance of the `Result` class. Its `getLogMath` method is then used to get a `LogMath` instance. The `logToLinear` method is passed the confidence value and the value returned is a real number between 0 and 1.0 inclusive. More confidence is reflected by a larger value.

The `getTimeFrame` method returns a `TimeFrame` instance. Its `toString` method returns two integer values, separated by a colon, reflecting the beginning and end times of the word:

```
for (WordResult wordResult : words) {  
    out.printf("%s\n\tConfidence: %.3f\n\tTime Frame: %s\n",  
              wordResult.getWord(), result  
                  .getResult()  
                  .getLogMath()  
                  .logToLinear((float)wordResult  
                               .getConfidence()),  
              wordResult.getTimeFrame());  
}
```

One possible output follows:

```
<sil>
```

```
Confidence: 0.998
Time Frame: 0:430
mary
Confidence: 0.998
Time Frame: 440:900
had
Confidence: 0.998
Time Frame: 910:1200
a
Confidence: 0.998
Time Frame: 1210:1340
little
Confidence: 0.998
Time Frame: 1350:1680
lamb
Confidence: 0.997
Time Frame: 1690:2170
```

Now that we have examined how sound can be processed, we will turn our attention to image processing.

Extracting text from an image

The process of extracting text from an image is called **O ptical Character Recognition (OCR)**. This can be very useful when the text data that needs to be processed is embedded in an image. For example, the information contained in license plates, road signs, and directions can be very useful at times.

We can perform OCR using Tess4j (<http://tess4j.sourceforge.net/>), a Java JNA wrapper for Tesseract OCR API. We will demonstrate how to use the API using an image captured from the Wikipedia article on OCR (https://en.wikipedia.org/wiki/Optical_character_recognition#Applications). The Javadoc for the API is found at <http://tess4j.sourceforge.net/docs/docs-3.0/>. The image we use is shown here:

OCR engines have been developed into many kinds of object-oriented OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR. They can be used for:

- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- Automatic insurance documents key information extraction
- Extracting business card information into a contact list^[9]
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg
- Make electronic images of printed documents searchable, e.g. Google Books
- Converting handwriting in real time to control a computer (pen computing)
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR^{[10][11][12]}
- Assistive technology for blind and visually impaired users

Using Tess4j to extract text

The `ITesseract` interface contains numerous OCR methods. The `doOCR` method takes a file and returns a string containing the words found in the file, as shown here:

```
ITesseract instance = new Tesseract();
try {
    String result = instance.doOCR(new File("OCRExample.png"));
    out.println(result);
} catch (TesseractException e) {
    // Handle exceptions
}
```

Part of the output is shown next:

OCR engines have been developed into many kinds of object-oriented OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR
They can be used for
- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- Automatic number plate recognition

As you can see, there are numerous errors in this example. Often the quality of an image needs to be improved before it can be processed correctly. Techniques for improving the quality of the output can be found at <https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality>. For example, we can use the `setLanguage` method to specify the language processed. Also, the method often works better on TIFF images.

In the next example, we used an enlarged portion of the previous image, as shown here:

OCR engines have been developed into many kinds of object-oriented OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- Automatic number plate recognition

The output is much better, as shown here:

OCR engines have been developed into many kinds of object-oriented OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- . Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt
- . Automatic number plate recognition

These examples highlight the need for the careful cleaning of data.

Identifying faces

Identifying faces within an image is useful in many situations. It can potentially classify an image as one containing people or find people in an image for further processing. We will use OpenCV 3.1 (<http://opencv.org/opencv-3-1.html>) for our examples.

OpenCV (<http://opencv.org/>) is an open source computer vision library that supports several programming languages, including Java. It supports a number of techniques, including machine learning algorithms, to perform computer vision tasks. The library supports such operations as face detection, tracking camera movements, extracting 3D models, and removing red eye from images. In this section, we will demonstrate face detection.

Using OpenCV to detect faces

The example that follows was adapted from

http://docs.opencv.org/trunk/d9/d52/tutorial_java_dev_intro.html. Start by loading the native libraries added to your system when OpenCV was installed. On Windows, this requires that appropriate DLL files are available:

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

We used a base string to specify the location of needed OpenCV files. Using an absolute path works better with many methods:

```
String base = "PathToResources";
```

The `CascadeClassifier` class is used for object classification. In this case, we will use it for face detection. An XML file is used to initialize the class. In the following code, we use the `lbpcascade_frontalface.xml` file, which provides information to assist in the identification of objects. In the OpenCV download are several files, as listed here, that can be used for specific face recognition scenarios:

- `lbpcascade_frontalcatface.xml`
- `lbpcascade_frontalface.xml`
- `lbpcascade_frontalprofileface.xml`
- `lbpcascade_silverware.xml`

The following statement initializes the class to detect faces:

```
CascadeClassifier faceDetector =
    new CascadeClassifier(base +
        "/lbpcascade_frontalface.xml");
```

The image to be processed is loaded, as shown here:

```
Mat image = Imgcodecs.imread(base + "/images.jpg");
```

For this example, we used the following image:



To find this image, perform a Google search using the term **people**. Select the **Images** category and then filter for **Labeled for reuse**. The image has the label: **Closeup portrait of a group of business people laughing by Lynda Sanchez**.

When faces are detected, the location within the image is stored in a `MatOfRect` instance. This class is intended to hold vectors and matrixes for any faces found:

```
MatOfRect faceVectors = new MatOfRect();
```

At this point, we are ready to detect faces. The `detectMultiScale` method performs this task. The image and the `MatOfRect` instance to hold the locations of any images are passed to the method:

```
faceDetector.detectMultiScale(image, faceVectors);
```

The next statement shows how many faces were detected:

```
out.println(faceVectors.toArray().length + " faces found");
```

We need to use this information to augment the image. This process will draw boxes around each face found, as shown next. To do this, the `Imgproc` class' `rectangle` method is used. The method is called once for each face detected. It is passed the image to be modified and the points represented the boundaries of the face:

```
for (Rect rect : faceVectors.toArray()) {  
    Imgproc.rectangle(image, new Point(rect.x, rect.y),  
                      new Point(rect.x + rect.width, rect.y + rect.height),  
                      new Scalar(0, 255, 0));  
}
```

The last step writes this image to a file using the `Imgcodecs` class' `imwrite` method:

```
Imgcodecs.imwrite("faceDetection.png", image);
```

As shown in the following image, it was able to identify four images:



Using different configuration files will work better for other facial profiles.

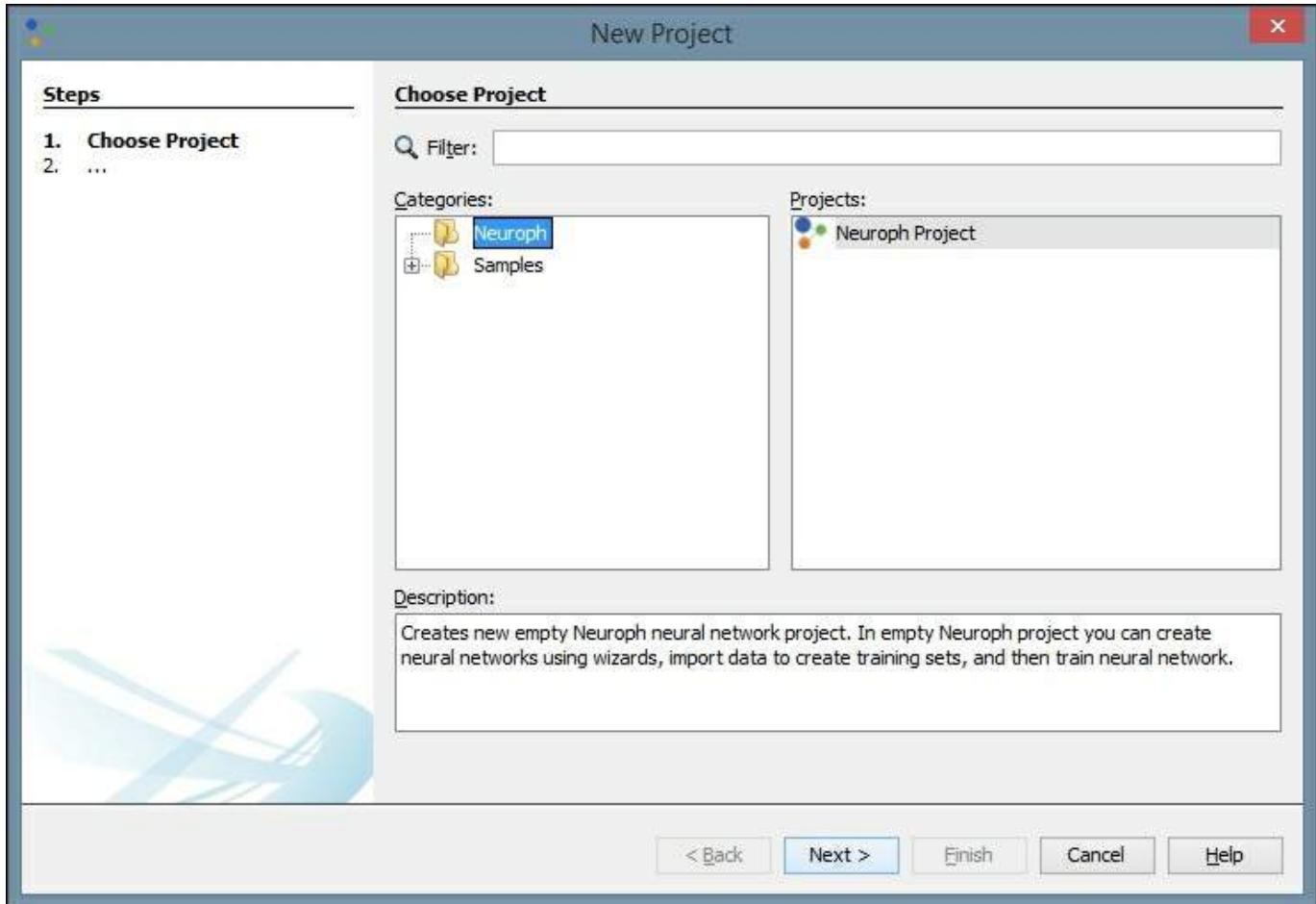
Classifying visual data

In this section, we will demonstrate one technique for classifying visual data. We will use Neuroph to accomplish this. Neuroph is a Java-based neural network framework that supports a variety of neural network architectures. Its open source library provides support and plugins for other applications. In this example, we will use its neural network editor, Neuroph Studio, to create a network. This network can be saved and used in other applications. Neuroph Studio is available for download here: <http://neuroph.sourceforge.net/download.html>. We are building upon the process shown here: http://neuroph.sourceforge.net/image_recognition.htm.

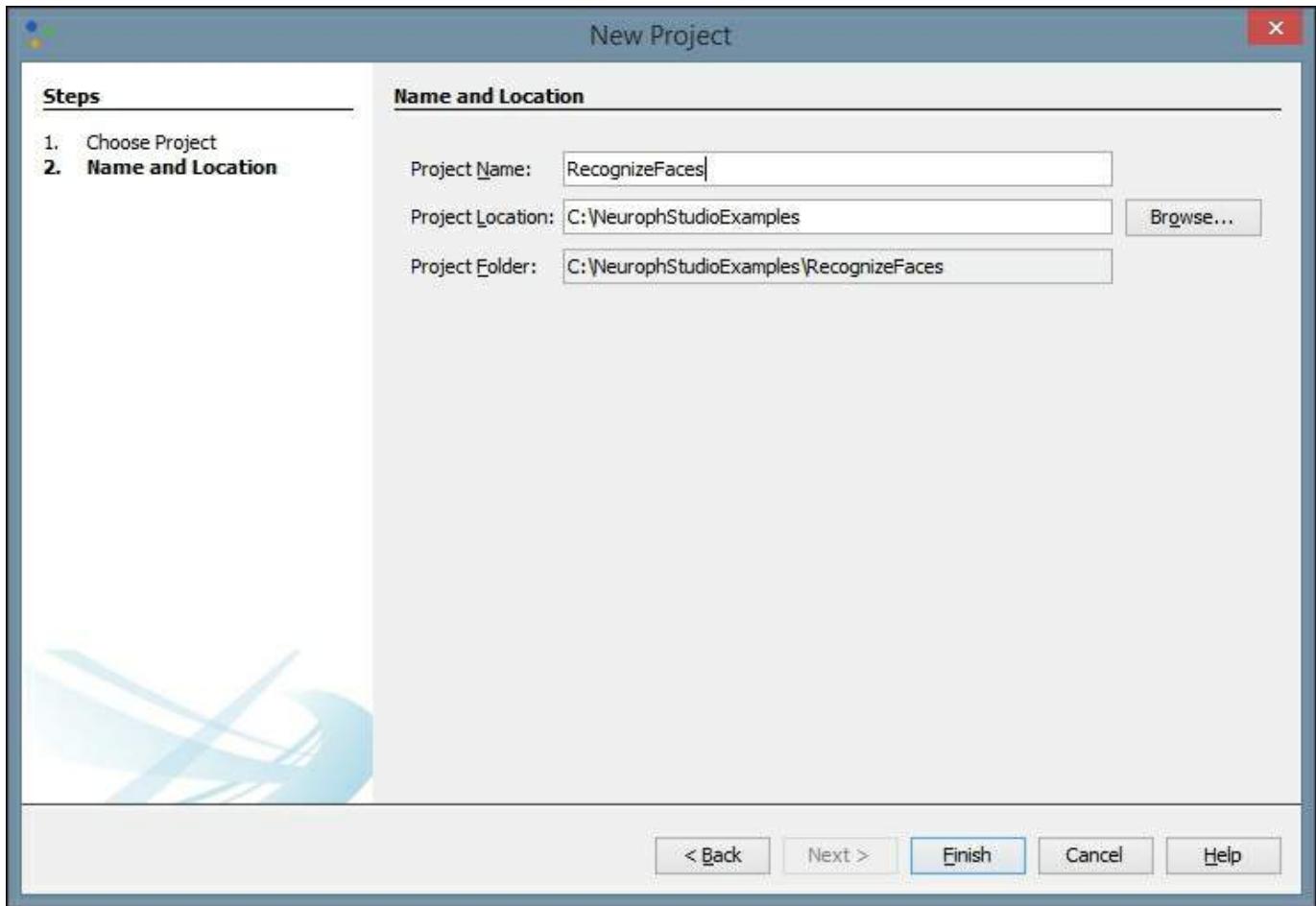
For our example, we will create a **Multi Layer Perceptron (MLP)** network. We will then train our network to recognize images. We can both train and test our network using Neuroph Studio. It is important to understand how MLP networks recognize and interpret image data. Every image is basically represented by three two-dimensional arrays. Each array contains information about the color components: one array contains information about the color red, one about the color green, and one about the color blue. Every element of the array holds information about one specific pixel in the image. These arrays are then flattened into a one-dimensional array to be used as an input by the neural network.

Creating a Neuroph Studio project for classifying visual images

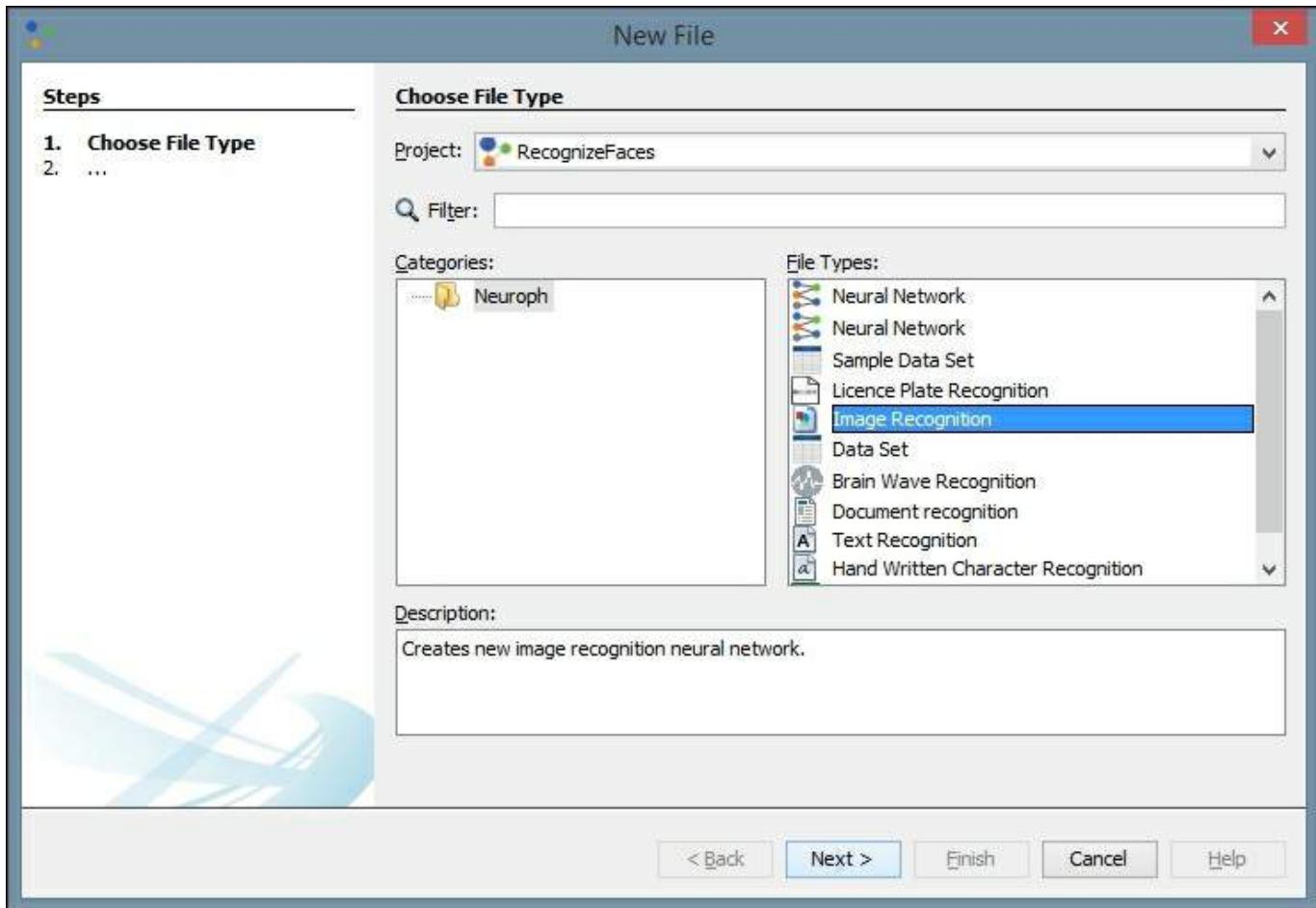
To begin, first create a new Neuroph Studio project:



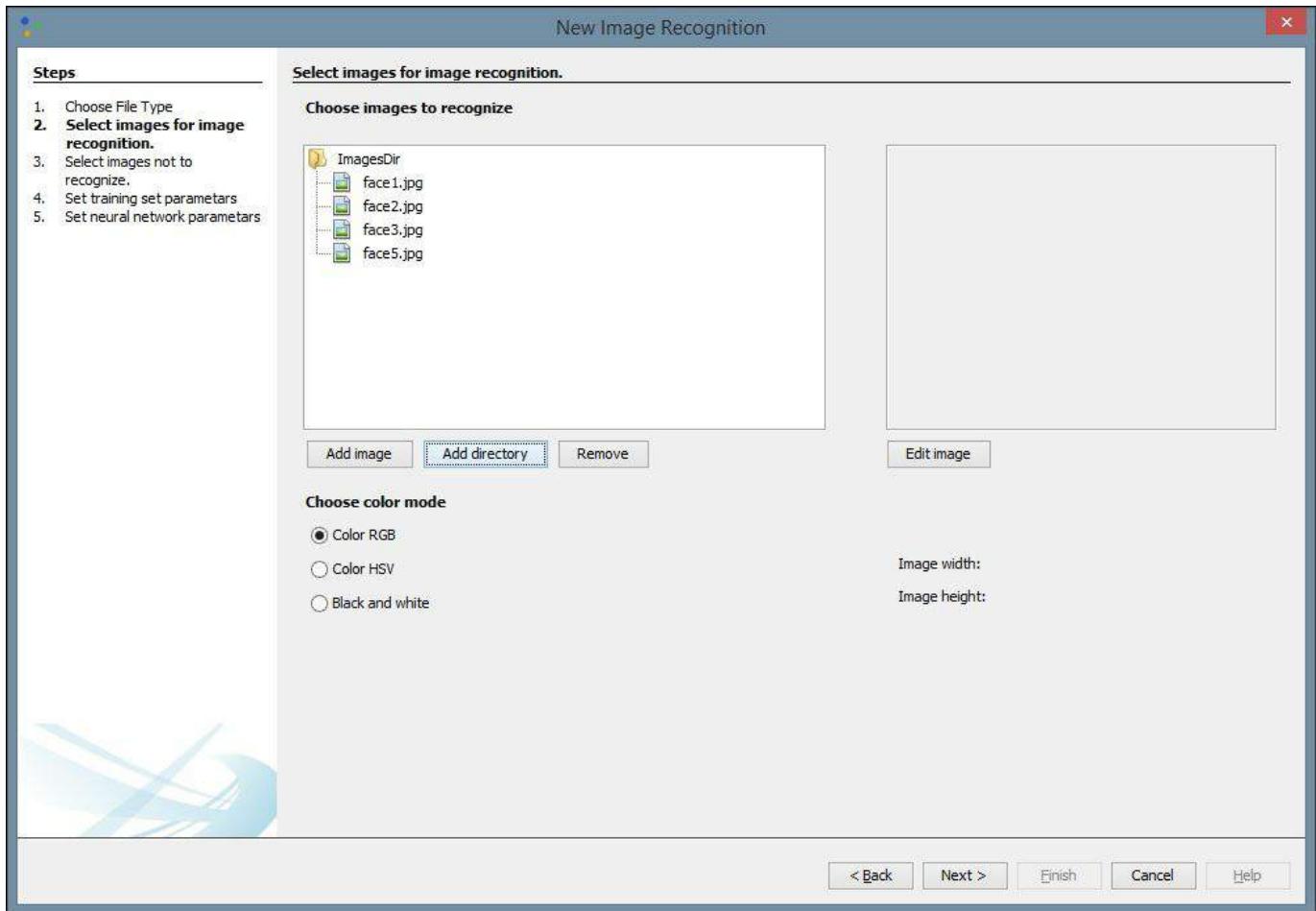
We will name our project `RecognizeFaces` because we are going to train the neural network to recognize images of human faces:



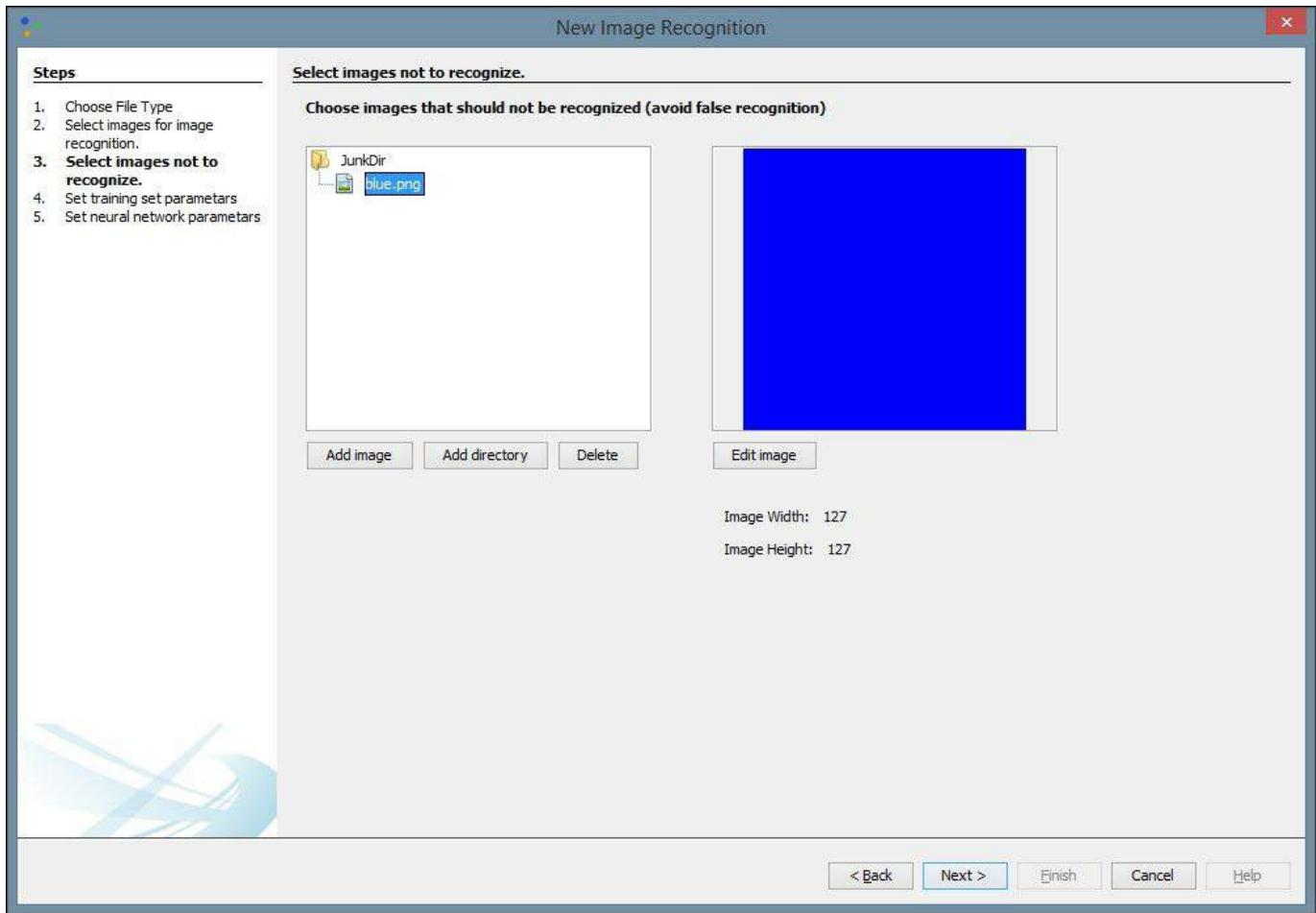
Next, we create a new file in our project. There are many types of project to choose from, but we will choose an **Image Recognition** type:



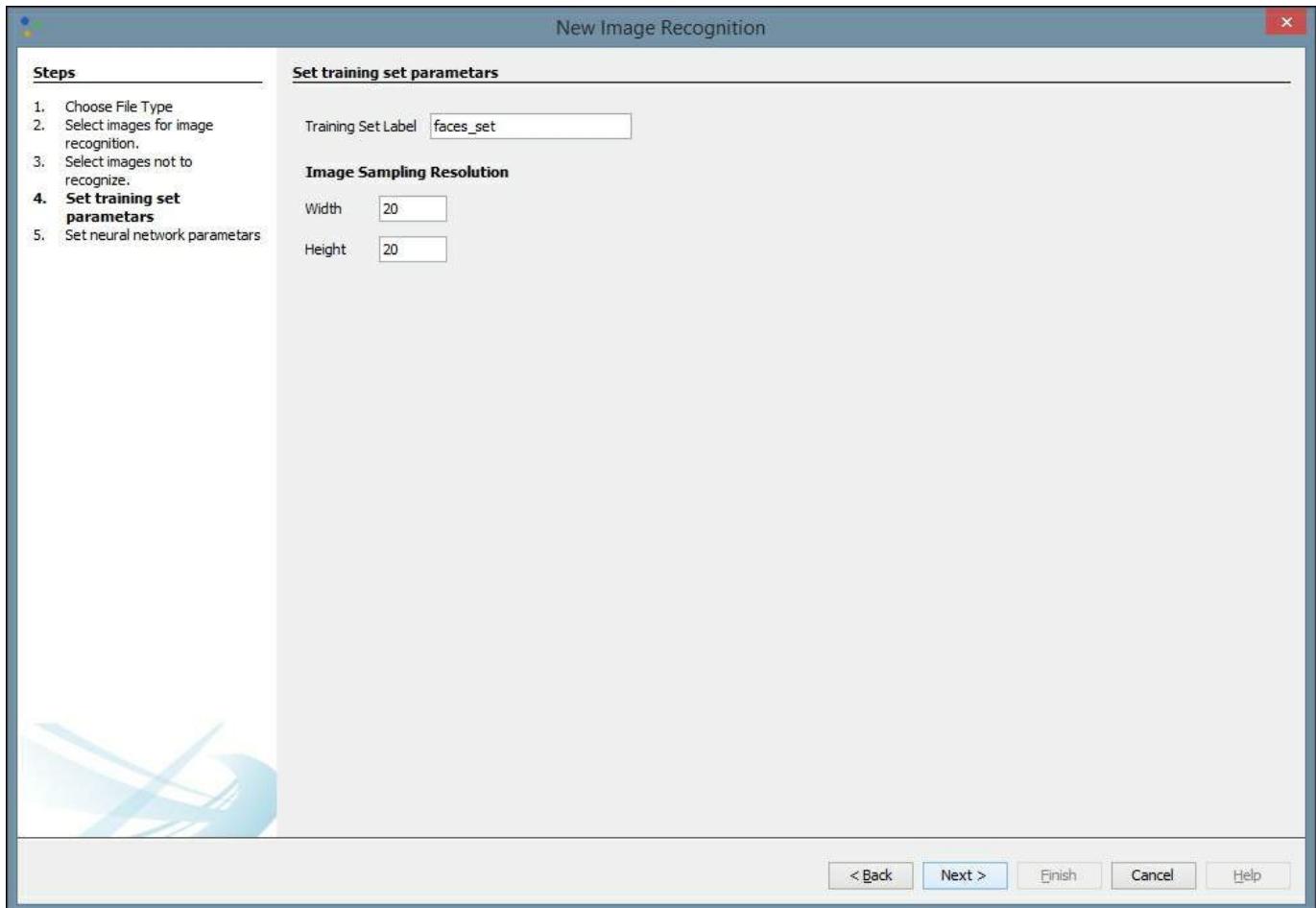
Click **Next** and then click **Add directory**. We have created a directory on our local machine and added several different black and white images of faces to use for training. These can be found by searching Google images or another search engine. The more quality images you have to train with, theoretically, the better your network will be:



After you click **Next**, you will be directed to select an image to not recognize. You may need to try different images based upon the images you want to recognize. The image you select here will prevent false recognitions. We have chosen a simple blue square from another directory on our local machine, but if you are using other types of image for training, other color blocks may work better:



Next, we need to provide network training parameters. We also need to label our training dataset and set our resolution. A height and width of 20 is a good place to start, but you may want to change these values to improve your results. Some trial and error may be involved. The purpose of providing this information is to allow for image scaling. When we scale images to a smaller size, our network can process them and learn faster:



Finally, we can create our network. We assign a label to our network and define our **Transfer function**. The default function, **Sigmoid**, will work for most networks, but if your results are not optimal you may want to try **Tanh**. The default number of **Hidden Layers Neuron Counts** is 12, and that is a good place to start. Be aware that increasing the number of neurons increases the time it takes to train the network and decreases your ability to generalize the network to other images. As with some of our previous values, some trial and error may be necessary to find the optimal settings for a given network. Select **Finish** when you are done:

New Image Recognition

Steps

1. Choose File Type
2. Select images for image recognition.
3. Select images not to recognize.
4. Set training set parameters
5. Set neural network parameters

Set neural network parameters

Network label: faces_net

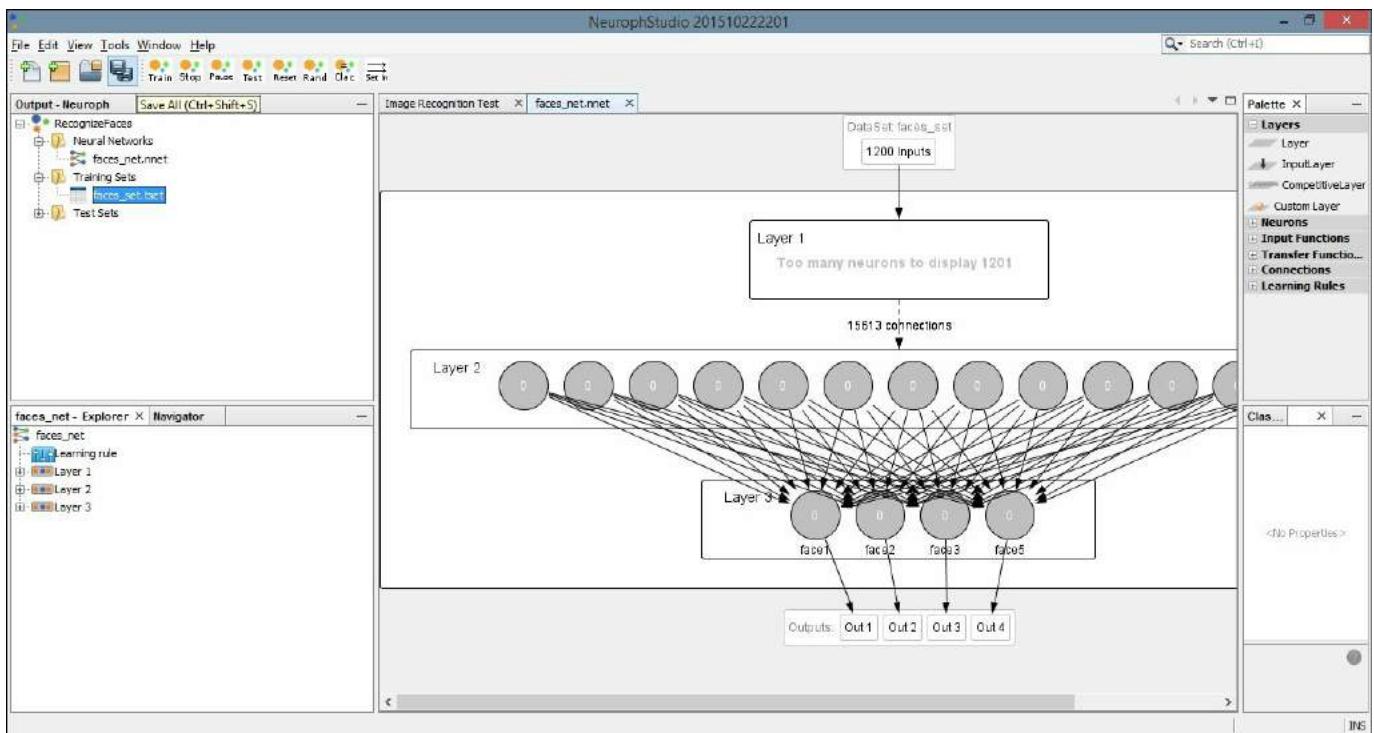
Transfer function: Sigmoid

Hidden Layers Neuron Counts: 12

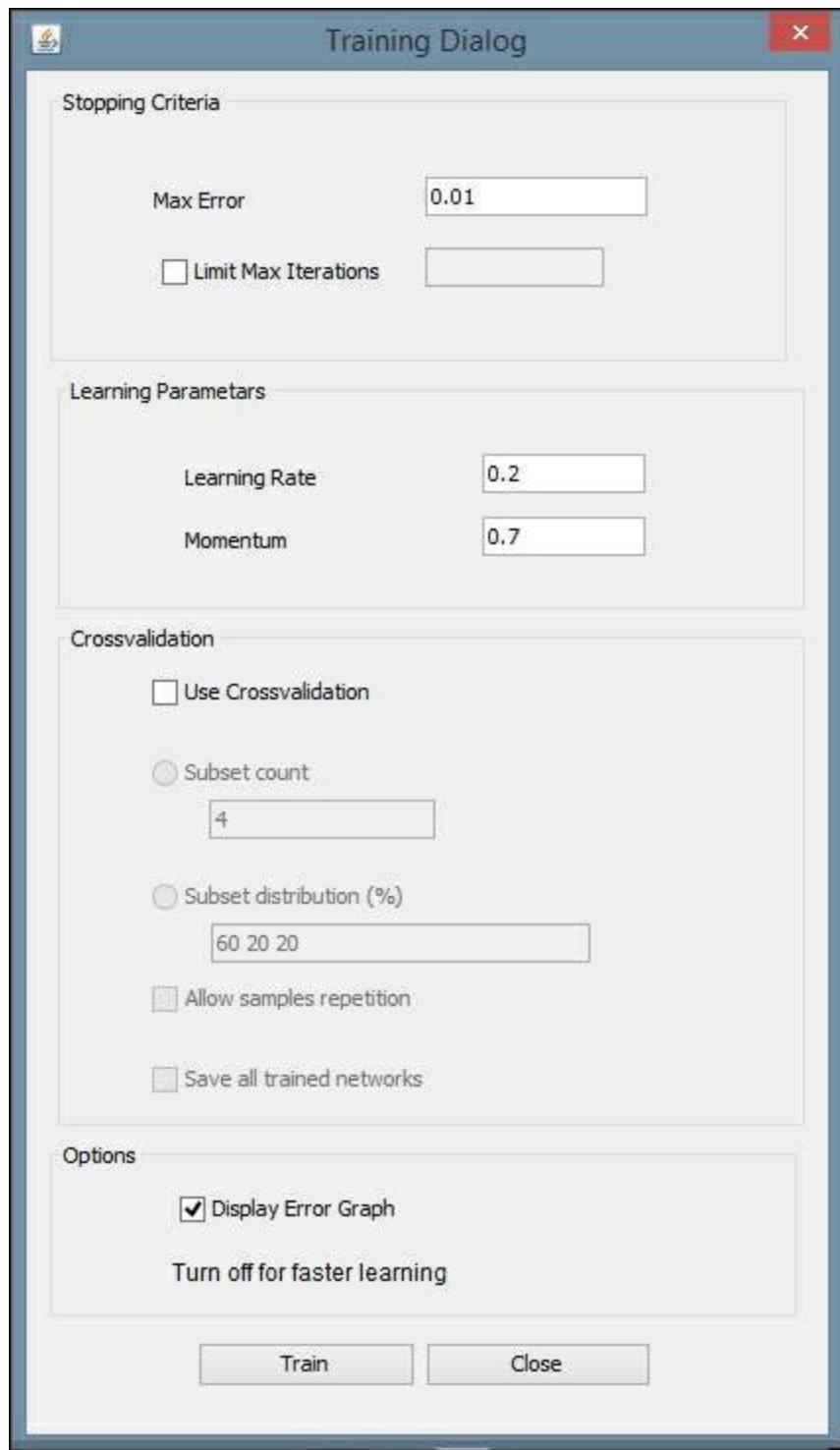
< Back Next > Finish Cancel Help

Training the model

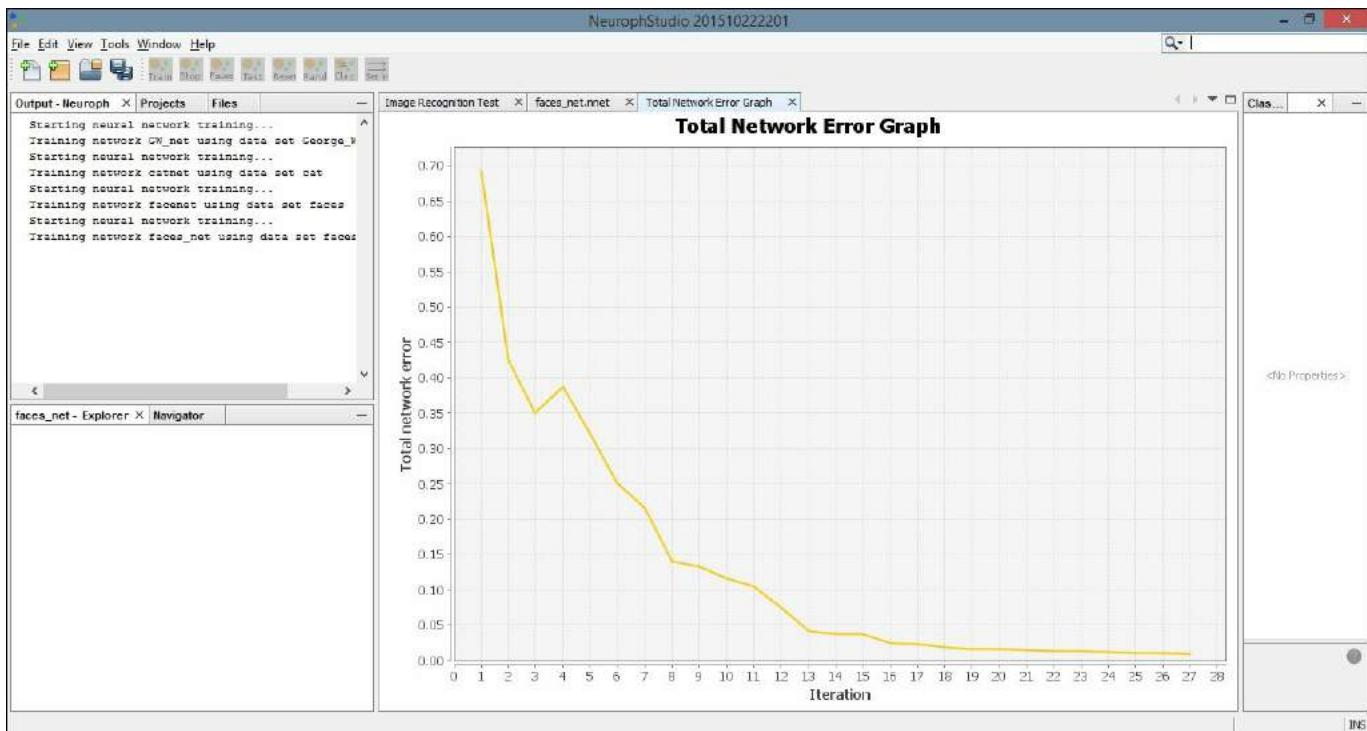
Once we have created our network, we need to train it. Begin by double-clicking on your neural network in the left pane. This is the file with the .nnet extension. When you do this, you will open a visual representation of the network in the main window. Then drag the dataset, with the file extension .tsest, from the left pane to the top node of the neural network. You will notice the description on the node change to the name of your dataset. Next, click the **Train** button, located in the top-left part of the window:



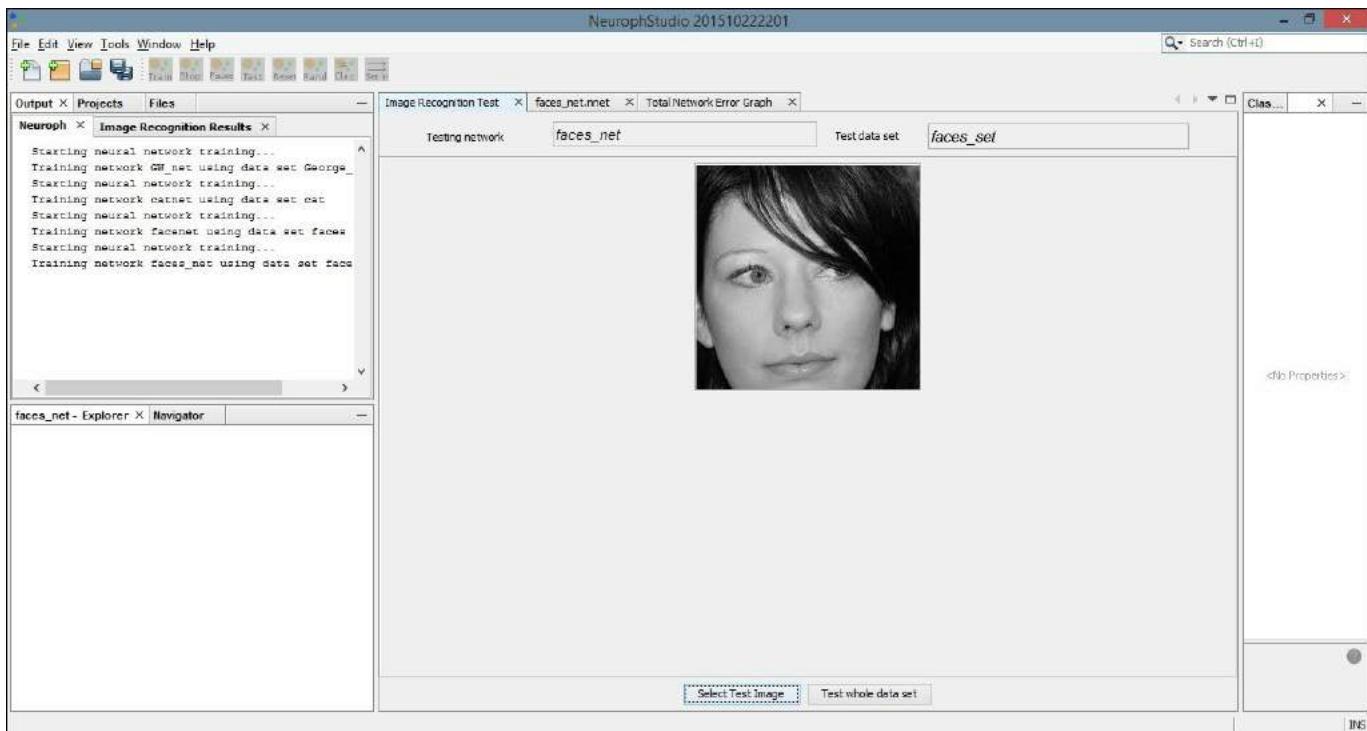
This will open a dialog box with settings for the training. You can leave the default values for **Max Error**, **Learning Rate**, and **Momentum**. Make sure the **Display Error Graph** box is checked. This allows you to see the improvement in the error rate as the training process continues:



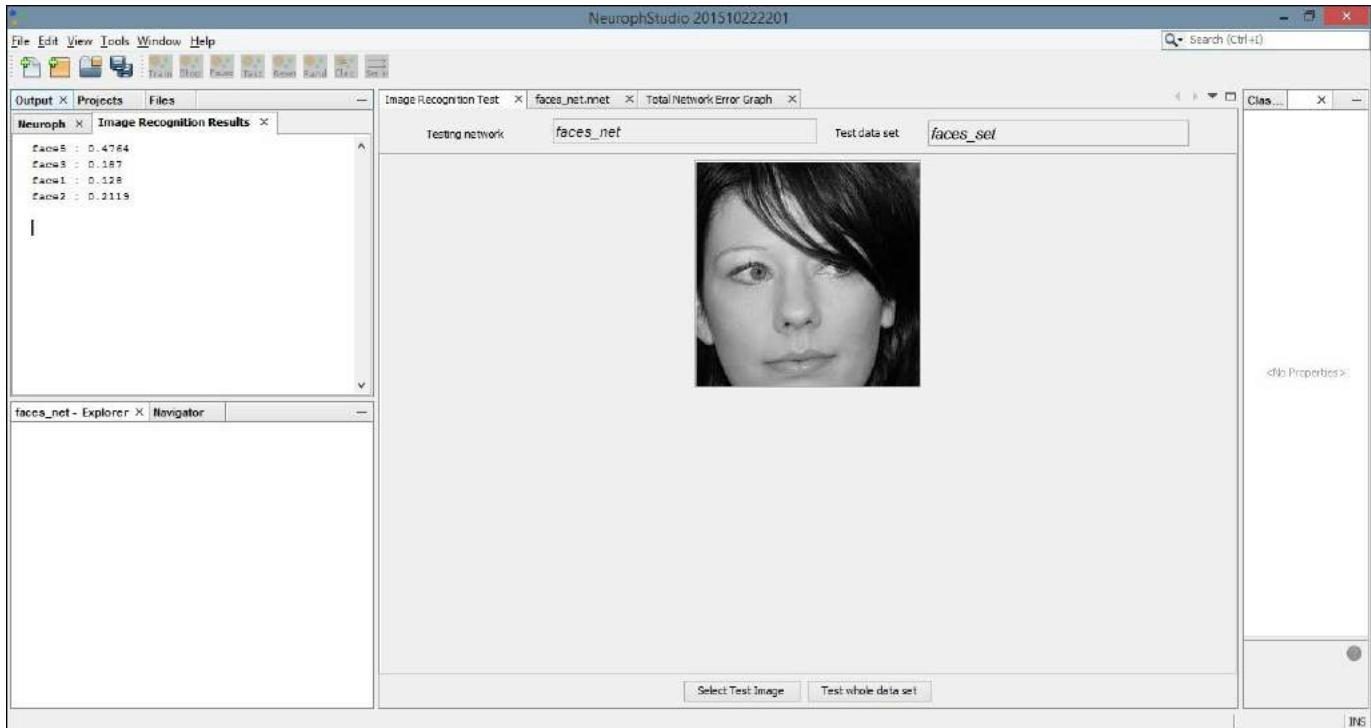
After you click the **Train** button, you should see an error graph similar to the following one:



Select the tab titled **Image Recognition Test**. Then click on the **Select Test Image** button. We have loaded a simple image of a face that was not included in our original dataset:



Locate the **Output** tab. It will be in the bottom or left pane and will display the results of comparing our test image with each image in the training set. The greater the number, the more closely our test image matches the image from our training set. The last image results in a greater output number than the first few comparisons. If we compare these images, they are more similar than the others in the dataset, and thus the network was able to create a more positive recognition of our test image:



We can now save our network for later use. Select **Save** from the **File** menu and then you can use the **.nnet** file in external applications. The following code example shows a simple technique for running test data through your pre-built neural network. The `NeuralNetwork` class is part of the Neuroph core package, and the `load` method allows you to load the trained network into your project. Notice we used our neural network name, `faces_net`. We then retrieve the plugin for our image recognition file. Next, we call the `recognizeImage` method with a new image, which must handle an `IOException`. Our results are stored in a `HashMap` and printed to the console:

```
NeuralNetwork iRNet = NeuralNetwork.load("faces_net.nnet");
ImageRecognitionPlugin iRFile
```

```
= (ImageRecognitionPlugin) iRNet.getPlugin(  
    ImageRecognitionPlugin.class);  
try {  
    HashMap<String, Double> newFaceMap  
        = imageRecognition.recognizeImage(  
            new File("testFace.jpg"));  
    out.println(newFaceMap.toString());  
} catch(IOException e) {  
    // Handle exceptions  
}
```

This process allows us to use a GUI editor application to create our network in a more visual environment, but then embed the trained network into our own applications.

Summary

In this chapter, we demonstrated many techniques for processing speech and images. This capability is becoming important, as electronic devices are increasingly embracing these communication mediums.

TTS was demonstrated using FreeTSS. This technique allows a computer to present results as speech as opposed to text. We learned how we can control the attributes of the voice used, such as its gender and age.

Recognizing speech is useful and helps bridge the human-computer interface gap. We demonstrated how CMUSphinx is used to recognize human speech. As there is often more than one way speech can be interpreted, we learned how the API can return various options. We also demonstrated how individual words are extracted, along with the relative confidence that the right word was identified.

Image processing is a critical aspect of many applications. We started our discussion of image processing by use Tess4J to extract text from an image. This process is sometimes referred to as OCR. We learned, as with many visual and audio data files, that the quality of the results is related to the quality of the image.

We also learned how to use OpenCV to identify faces within an image. Information about specific view of faces, such as frontal or profile views, are contained in XML files. These files were used to outline faces within an image. More than one face can be detected at a time.

It can be helpful to classify images and sometimes external tools are useful for this purpose. We examined Neuroph Studio and created a neural network designed to recognize and classify images. We then tested our network with images of human faces.

In the next chapter, we will learn how to speed up common data science applications using multiple processors.

Chapter 11. Mathematical and Parallel Techniques for Data Analysis

The concurrent execution of a program can result in significant performance improvements. In this chapter, we will address the various techniques that can be used in data science applications. These can range from low-level mathematical calculations to higher-level API-specific options.

Always keep in mind that performance enhancement starts with ensuring that the correct set of application functionality is implemented. If the application does not do what a user expects, then the enhancements are for nought. The architecture of the application and the algorithms used are also more important than code enhancements. Always use the most efficient algorithm. Code enhancement should then be considered. We are not able to address the higher-level optimization issues in this chapter; instead, we will focus on code enhancements.

Many data science applications and supporting APIs use matrix operations to accomplish their tasks. Often these operations are buried within an API, but there are times when we may need to use these directly. Regardless, it can be beneficial to understand how these operations are supported. To this end, we will explain how matrix multiplication is handled using several different approaches.

Concurrent processing can be implemented using Java threads. A developer can use threads and thread pools to improve an application's response time. Many APIs will use threads when multiple CPUs or GPUs are not available, as is the case with **Aparapi**. We will not illustrate the use of threads here. However, the reader is assumed to have a basic knowledge of threads and thread pools.

The map-reduce algorithm is used extensively for data science applications. We will present a technique for achieving this type of parallel processing using Apache's Hadoop. Hadoop is a framework supporting the manipulation of large datasets, and can greatly decrease the required processing time for large data science projects. We will demonstrate a technique for calculating an average value for a sample set of data.

There are several well-known APIs that support multiple processors, including CUDA and OpenCL. CUDA is supported using **Java bindings for CUDA (JCuda)**

(<http://jcuda.org/>). We will not demonstrate this technique directly here. However, many of the APIs we will use do support CUDA if it is available, such as DL4J. We will briefly discuss OpenCL and how it is supported in Java. It is worth noting that the Aparapi API provides higher-level support, which may use either multiple CPUs or GPUs. A demonstration of Aparapi in support of matrix multiplication will be illustrated.

In this chapter, we will examine how multiple CPUs and GPUs can be harnessed to speed up data mining tasks. Many of the APIs we have used already take advantage of multiple processors or at least provide a means to enable GPU usage. We will introduce a number of these options in this chapter.

Concurrent processing is also supported extensively in the cloud. Many of the techniques discussed here are used in the cloud. As a result, we will not explicitly address how to conduct parallel processing in the cloud.

Implementing basic matrix operations

There are several different types of matrix operations, including simple addition, subtraction, scalar multiplication, and various forms of multiplication. To illustrate the matrix operations, we will focus on what is known as **matrix product**. This is a common approach that involves the multiplication of two matrixes to produce a third matrix.

Consider two matrices, A and B , where matrix A has n rows and m columns. Matrix B will have m rows and p columns. The product of A and B , written as AB , is an n row and p column matrix. The m entries of the rows of A are multiplied by the m entries of the columns of matrix B . This is more explicitly shown here, where:

$$A = \begin{vmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{vmatrix} \quad B = \begin{vmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{vmatrix}$$

Where the product is defined as follows:

$$(AB)_{ij} = \sum_{k=1}^m A_{ik} B_{kj}$$

We start with the declaration and initialization of the matrices. The variables n , m , p represent the dimensions of the matrices. The A matrix is n by m , the B matrix is m by

p , and the C matrix representing the product is n by p :

```
int n = 4;
int m = 2;
int p = 3;

double A[][] = {
    {0.1950, 0.0311},
    {0.3588, 0.2203},
    {0.1716, 0.5931},
    {0.2105, 0.3242}};
double B[][] = {
    {0.0502, 0.9823, 0.9472},
    {0.5732, 0.2694, 0.916}};
double C[][] = new double[n][p];
```

The following code sequence illustrates the multiplication operation using nested `for` loops:

```
for (int i = 0; i < n; i++) {
    for (int k = 0; k < m; k++) {
        for (int j = 0; j < p; j++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

This following code sequence formats output to display our matrix:

```
out.println("\nResult");
for (int i = 0; i < n; i++) {
    for (int j = 0; j < p; j++) {
        out.printf("%.4f ", C[i][j]);
    }
    out.println();
}
```

The result appears as follows:

```
Result
0.0276 0.1999 0.2132
0.1443 0.4118 0.5417
0.3486 0.3283 0.7058
0.1964 0.2941 0.4964
```

Later, we will demonstrate several alternative techniques for performing the same operation. Next, we will discuss how to tailor support for multiple processors using

DL4J.

Using GPUs with DeepLearning4j

DeepLearning4j works with GPUs such as those provided by NVIDIA. There are options available that enable the use of GPUs, specify how many GPUs should be used, and control the use of GPU memory. In this section, we will show how you can use these options. This type of control is often available with other high-level APIs.

DL4J uses **n-dimensional arrays for Java (ND4J)** (<http://nd4j.org/>) to perform numerical computations. This is a library that supports n-dimensional array objects and other numerical computations, such as linear algebra and signal processing. It includes support for GPUs and is also integrated with Hadoop and Spark.

A **vector** is a one-dimensional array of numbers and is used extensively with neural networks. A vector is a type of mathematical structure called a **tensor**. A tensor is essentially a multidimensional array. We can think of a tensor as an array with three or more dimensions, and each dimension is called a **rank**.

There is often a need to map a multidimensional set of numbers to a one-dimensional array. This is done by flattening the array using a defined order. For example, with a two-dimensional array many systems will allocate the members of the array in row-column order. This means the first row is added to the vector, followed by the second vector, and then the third, and so forth. We will use this approach in the *Using the ND4J API* section.

To enable GPU use, the project's POM file needs to be modified. In the properties section of the POM file, the `nd4j.backend` tag needs to be added or modified, as shown here:

```
<nd4j.backend>nd4j-cuda-7.5-platform</nd4j.backend>
```

Models can be trained in parallel using the `ParallelWrapper` class. The training task is automatically distributed among available CPUs/GPUs. The model is used as an argument to the `ParallelWrapper` class' `Builder` constructor, as shown here:

```
ParallelWrapper parallelWrapper =  
    new ParallelWrapper.Builder(aModel)  
        // Builder methods...  
        .build();
```

When executed, a copy of the model is used on each GPU. After the number of

iterations is specified by the `averagingFrequency` method, the models are averaged and then the training process continues.

There are various methods that can be used to configure the class, as summarized in the following table:

Method	Purpose
<code>prefetchBuffer</code>	Specifies the size of a buffer used to pre-fetch data
<code>workers</code>	Specifies the number of workers to be used
<code>averageUpdaters</code> <code>averagingFrequency</code> <code>reportScoreAfterAveraging</code> <code>useLegacyAveraging</code>	Various methods to control how averaging is achieved

The number of workers should be greater than the number of GPUs available.

As with most computations, using a lower precision value will speed up the processing. This can be controlled using the `setDTypeForContext` method, as shown next. In this case, half precision is specified:

```
DataTypeUtil.setDTypeForContext(DataBuffer.Type.HALF);
```

This support and more details regarding optimization techniques can be found at <http://deeplearning4j.org/gpu>.

Using map-reduce

Map-reduce is a model for processing large sets of data in a parallel, distributed manner. This model consists of a `map` method for filtering and sorting data, and a `reduce` method for summarizing data. The map-reduce framework is effective because it distributes the processing of a dataset across multiple servers, performing mapping and reduction simultaneously on smaller pieces of the data. Map-reduce provides significant performance improvements when implemented in a multi-threaded manner. In this section, we will demonstrate a technique using Apache's Hadoop implementation. In the *Using Java 8 to perform map-reduce* section, we will discuss techniques for performing map-reduce using Java 8 streams.

Hadoop is a software ecosystem providing support for parallel computing. Map-reduce jobs can be run on Hadoop servers, generally set up as clusters, to significantly improve processing speeds. Hadoop has trackers that run map-reduce operations on nodes within a Hadoop cluster. Each node operates independently and the trackers monitor the progress and integrate the output of each node to generate the final output. The following image can be found at <http://www.developer.com/java/data/big-data-tool-map-reduce.html> and demonstrates the basic map-reduce model with trackers.

Using Apache's Hadoop to perform map-reduce

We are going to show you a very simple example of a map-reduce application here. Before we can use Hadoop, we need to download and extract Hadoop application files. The latest versions can be found at <http://hadoop.apache.org/releases.html>. We are using version 2.7.3 for this demonstration.

You will need to set your `JAVA_HOME` environment variable. Additionally, Hadoop is intolerant of long file paths and spaces within paths, so make sure you extract Hadoop to the simplest directory structure possible.

We will be working with a sample text file containing information about books. Each line of our tab-delimited file has the book title, author, and page count:

```
Moby Dick Herman Melville 822  
  
Charlotte's Web E.B. White 189  
The Grapes of Wrath John Steinbeck 212  
Jane Eyre Charlotte Bronte 299  
A Tale of Two Cities Charles Dickens 673  
War and Peace Leo Tolstoy 1032  
The Great Gatsby F. Scott Fitzgerald 275
```

We are going to use a `map` function to extract the title and page count information and then a `reduce` function to calculate the average page count of the books in our dataset. To begin, create a new class, `AveragePageCount`. We will create two static classes within `AveragePageCount`, one to handle the `map` procedure and one to handle the reduction.

Writing the map method

First, we will create the `TextMapper` class, which will implement the `map` method. This class inherits from the `Mapper` class and has two private instance variables, `pages` and `bookTitle`. `pages` is an `IntWritable` object and `bookTitle` is a `Text` object. `IntWritable` and `Text` are used because these objects will need to be serialized to a byte stream before they can be transmitted to the servers for processing. These objects take up less space and transfer faster than the comparable `int` or `String` objects:

```
public static class TextMapper
    extends Mapper<Object, Text, Text, IntWritable> {

    private final IntWritable pages = new IntWritable();
    private final Text bookTitle = new Text();

}
```

Within our `TextMapper` class we create the `map` method. This method takes three parameters: the `key` object, a `Text` object, `bookInfo`, and the `Context`. The `key` allows the tracker to map each particular object back to the correct job. The `bookInfo` object contains the text or string data about each book. `Context` holds information about the entire system and allows the method to report on progress and update values within the system.

Within the `map` method, we use the `split` method to break each piece of book information into an array of `String` objects. We set our `bookTitle` variable to position `0` of the array and set `pages` to the value stored in position `2`, after parsing it as an integer. We can then write out our book title and page count information through the context and update our entire system:

```
public void map(Object key, Text bookInfo, Context context)
    throws IOException, InterruptedException {
    String[] book = bookInfo.toString().split("\t");
    bookTitle.set(book[0]);
    pages.set(Integer.parseInt(book[2]));
    context.write(bookTitle, pages);
}
```

Writing the reduce method

Next, we will write our `AverageReduce` class. This class extends the `Reducer` class and will perform the reduction processes to calculate our average page count. We have created four variables for this class: a `FloatWritable` object to store our average page count, a float `average` to hold our temporary average, a float `count` to count how many books exist in our dataset, and an integer `sum` to add up the page counts:

```
public static class AverageReduce
    extends Reducer<Text, IntWritable, Text, FloatWritable> {

    private final FloatWritable finalAvg = new FloatWritable();
    float average = 0f;
    float count = 0f;
    int sum = 0;

}
```

Within our `AverageReduce` class we will create the `reduce` method. This method takes as input a `Text` key, an `Iterable` object holding writeable integers representing the page counts, and the `Context`. We use our iterator to process the page counts and add each to our sum. We then calculate the average and set the value of `finalAvg`. This information is paired with a `Text` object label and written to the `Context`:

```
public void reduce(Text key, Iterable<IntWritable> pageCnts,
    Context context)
    throws IOException, InterruptedException {

    for (IntWritable cnt : pageCnts) {
        sum += cnt.get();
    }
    count += 1;
    average = sum / count;
    finalAvg.set(average);
    context.write(new Text("Average Page Count = "), finalAvg);
}
```

Creating and executing a new Hadoop job

We are now ready to create our `main` method in the same class and execute our map-reduce processes. To do this, we need to create a new `Configuration` object and a new `Job`. We then set up the significant classes to use in our application.

```
public static void main(String[] args) throws Exception {  
    Configuration con = new Configuration();  
    Job bookJob = Job.getInstance(con, "Average Page Count");  
    ...  
}
```

We set our main class, `AveragePageCount`, in the `setJarByClass` method. We specify our `TextMapper` and `AverageReduce` classes using the `setMapperClass` and `setReducerClass` methods, respectively. We also specify that our output will have a text-based key and a writeable integer using the `setOutputKeyClass` and `setOutputValueClass` methods:

```
bookJob.setJarByClass(AveragePageCount.class);  
bookJob.setMapperClass(TextMapper.class);  
bookJob.setReducerClass(AverageReduce.class);  
bookJob.setOutputKeyClass(Text.class);  
bookJob.setOutputValueClass(IntWritable.class);
```

Finally, we create new input and output paths using the `addInputPath` and `setOutputPath` methods. These methods both take our `Job` object as the first parameter and a `Path` object representing our input and output file locations as the second parameter. We then call `waitForCompletion`. Our application exits once this call returns true:

```
FileInputFormat.addInputPath(bookJob, new  
Path("C:/Hadoop/books.txt"));  
FileOutputFormat.setOutputPath(bookJob, new  
Path("C:/Hadoop/BookOutput"));  
if (bookJob.waitForCompletion(true)) {  
    System.exit(0);  
}
```

To execute the application, open a command prompt and navigate to the directory containing our `AveragePageCount.class` file. We then use the following command to execute our sample application:

```
hadoop AveragePageCount
```

While our task is running, we see updated information about our process output to the screen. A sample of our output is shown as follows:

```
...
File System Counters
FILE: Number of bytes read=1132
FILE: Number of bytes written=569686
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
Map-Reduce Framework
Map input records=7
Map output records=7
Map output bytes=136
Map output materialized bytes=156
Input split bytes=90
Combine input records=0
Combine output records=0
Reduce input groups=7
Reduce shuffle bytes=156
Reduce input records=7
Reduce output records=7
Spilled Records=14
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=11
Total committed heap usage (bytes)=536870912
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=249
File Output Format Counters
Bytes Written=216
```

If we open the `BookOutput` directory created on our local machine, we find four new files. Use a text editor to open `part-r-00000`. This file contains information about the average page count as it was calculated using parallel processes. A sample of this output follows:

```
Average Page Count = 673.0
Average Page Count = 431.0
```

```
Average Page Count = 387.0
Average Page Count = 495.75
Average Page Count = 439.0
Average Page Count = 411.66666
Average Page Count = 500.2857
```

Notice how the average changes as each individual process is combined with the other reduction processes. This has the same effect as calculating the average of the first two books first, then adding in the third book, then the fourth, and so on. The advantage here of course is that the averaging is done in a parallel manner. If we had a huge dataset, we should expect to see a noticeable advantage in execution time. The last line of `BookOutput` reflects the correct and final average of all seven page counts.

Various mathematical libraries

There are numerous mathematical libraries available for Java use. In this section, we will provide a quick and high-level overview of several libraries. These libraries do not necessarily automatically support multiple processors. In addition, the intent of this section is to provide some insight into how these libraries can be used. In most cases, they are relatively easy to use.

A list of Java mathematical libraries is found at

https://en.wikipedia.org/wiki/List_of_numerical_libraries#Java and <https://javamatrix.org/>. We will demonstrate the use of the jblas, Apache Commons Math, and the ND4J libraries.

Using the jblas API

The jblas API (<http://jblas.org/>) is a math library supporting Java. It is based on **Basic Linear Algebra Subprograms (BLAS)** (<http://www.netlib.org/blas/>) and **Linear Algebra Package (LAPACK)** (<http://www.netlib.org/lapack/>), which are standard libraries for fast arithmetic calculation. The jblas API provides a wrapper around these libraries.

The following is a demonstration of how matrix multiplication is performed. We start with the matrix definitions:

```
DoubleMatrix A = new DoubleMatrix(new double[][]{  
    {0.1950, 0.0311},  
    {0.3588, 0.2203},  
    {0.1716, 0.5931},  
    {0.2105, 0.3242}});  
  
DoubleMatrix B = new DoubleMatrix(new double[][]{  
    {0.0502, 0.9823, 0.9472},  
    {0.5732, 0.2694, 0.916}});  
DoubleMatrix C;
```

The actual statement to perform multiplication is quite short, as shown next. The `mmul` method is executed against the `A` matrix, where the `B` array is passed as an argument:

```
C = A.mmul(B);
```

The resulting `C` matrix is then displayed:

```
for(int i=0; i<C.getRows(); i++) {  
    out.println(C.getRow(i));  
}
```

The output should be as follows:

```
[0.027616, 0.199927, 0.213192]  
[0.144288, 0.411798, 0.541650]  
[0.348579, 0.328344, 0.705819]  
[0.196399, 0.294114, 0.496353]
```

This library is fairly easy to use and supports an extensive set of arithmetic operations.

Using the Apache Commons math API

The Apache Commons math API (<http://commons.apache.org/proper/commons-math/>) supports a large number of mathematical and statistical operations. The following example illustrates how to perform matrix multiplication.

We start with the declaration and initialization of the `A` and `B` matrices:

```
double[][] A = {  
    {0.1950, 0.0311},  
    {0.3588, 0.2203},  
    {0.1716, 0.5931},  
    {0.2105, 0.3242}};  
  
double[][] B = {  
    {0.0502, 0.9823, 0.9472},  
    {0.5732, 0.2694, 0.916}};
```

Apache Commons uses the `RealMatrix` class to hold a matrix. In the following code sequence, the corresponding matrices for the `A` and `B` matrices are created using the `Array2DRowRealMatrix` constructor:

```
RealMatrix aRealMatrix = new Array2DRowRealMatrix(A);  
RealMatrix bRealMatrix = new Array2DRowRealMatrix(B);
```

The multiplication is straightforward using the `multiply` method, as shown next:

```
RealMatrix cRealMatrix = aRealMatrix.multiply(bRealMatrix);
```

The next `for` loop will display the following results:

```
for (int i = 0; i < cRealMatrix.getRowDimension(); i++) {  
    out.println(cRealMatrix.getRowVector(i));  
}
```

The output should be as follows:

```
{0.02761552; 0.19992684; 0.2131916}  
{0.14428772; 0.41179806; 0.54165016}  
{0.34857924; 0.32834382; 0.70581912}  
{0.19639854; 0.29411363; 0.4963528}
```

Using the ND4J API

ND4J (<http://nd4j.org/>) is the library used by DL4J to perform arithmetic operations. The library is also available for direct use. In this section, we will demonstrate how matrix multiplication is performed using the `A` and `B` matrices.

Before we can perform the multiplication, we need to flatten the matrices to vectors. The following declares and initializes these vectors:

```
double[] A = {  
    0.1950, 0.0311,  
    0.3588, 0.2203,  
    0.1716, 0.5931,  
    0.2105, 0.3242};  
  
double[] B = {  
    0.0502, 0.9823, 0.9472,  
    0.5732, 0.2694, 0.916};
```

The `Nd4j` class' `create` method creates an `INDArray` instance given a vector and dimension information. The first argument of the method is the vector. The second argument specifies the dimensions of the matrix. The last argument specifies the order the rows and columns are laid out. This order is either row-column major as exemplified by `c`, or column-row major order as used by FORTRAN. Row-column order means the first row is allocated to the vector, followed by the second row, and so forth.

In the following code sequence `INDArray` instances are created using the `A` and `B` vectors. The first is a 4 row, 2 column matrix using row-major order as specified by the third argument, `c`. The second `INDArray` instance represents the `B` matrix. If we wanted to use column-row ordering, we would use an `f` instead.

```
INDArray aINDArray = Nd4j.create(A, new int[]{4,2}, 'c');  
INDArray bINDArray = Nd4j.create(B, new int[]{2,3}, 'c');
```

The `c` array, represented by `cINDArray`, is then declared and assigned the result of the multiplication. The `mmul` performs the operation:

```
INDArray cINDArray;  
cINDArray = aINDArray.mmul(bINDArray);
```

The following sequence displays the results using the `getRow` method:

```
for(int i=0; i<cINDArray.rows(); i++) {  
    out.println(cINDArray.getRow(i));  
}
```

The output should be as follows:

```
[0.03, 0.20, 0.21]  
[0.14, 0.41, 0.54]  
[0.35, 0.33, 0.71]  
[0.20, 0.29, 0.50]
```

Next, we will provide an overview of the OpenCL API that provide supports for concurrent operations on a number of platforms.

Using OpenCL

Open Computing Language (OpenCL) (<https://www.khronos.org/opencl/>)

supports programs that execute across heterogeneous platforms, that is, platforms potentially using different vendors and architectures. The platforms can use different processing units, including **Central Processing Unit (CPU)**, **Graphical Processing Unit (GPU)**, **Digital Signal Processor (DSP)**, **Field-Programmable Gate Array (FPGA)**, and other types of processors.

OpenCL uses a C99-based language to program the devices, providing a standard interface for programming concurrent behavior. OpenCL supports an API that allows code to be written in different languages. For Java, there are several APIs that support the development of OpenCL based languages:

- **Java bindings for OpenCL (JOCL)** (<http://www.jocl.org/>) - This is a binding to the original OpenCL C implementation and can be verbose.
- **JavaCl** (<https://code.google.com/archive/p/javacl/>) - Provides an object-oriented interface to JOCL.
- **Java OpenCL** (<http://jogamp.org/jocl/www/>) - Also provides an object-oriented abstraction of JOCL. It is not intended for client use.
- **The Lightweight Java Game Library (LWJGL)** (<https://www.lwjgl.org/>) - Also provides support for OpenCL and is oriented toward GUI applications.

In addition, Aparapi provides higher-level access to OpenCL, thus avoiding some of the complexity involved in creating OpenCL applications.

Code that runs on a processor is encapsulated in a kernel. Multiple kernels will execute in parallel on different computing devices. There are different levels of memory supported by OpenCL. A specific device may not support each level. The levels include:

- **Global memory** - Shared by all computing units
- **Read-only memory** - Generally not writable
- **Local memory** - Shared by a group of computing units
- **Per-element private memory** - Often a register

OpenCL applications require a considerable amount of initial code to be useful. This complexity does not permit us to provide a detailed example of its use. However, the Aparapi section does provide some feel for how OpenCL applications are structured.

Using Aparapi

Aparapi (<https://github.com/aparapi/aparapi>) is a Java library that supports concurrent operations. The API supports code running on GPUs or CPUs. GPU operations are executed using OpenCL, while CPU operations use Java threads. The user can specify which computing resource to use. However, if GPU support is not available, Aparapi will revert to Java threads.

The API will convert Java byte codes to OpenCL at runtime. This makes the API largely independent from the graphics card used. The API was initially developed by AMD but has been released as open source. This is reflected in the basic package name, `com.amd.aparapi`. Aparapi offers a higher level of abstraction than provided by OpenCL.

Aparapi code is located in a class derived from the `Kernel` class. Its `execute` method will start the operations. This will result in an internal call to a `run` method, which needs to be overridden. It is within the `run` method that concurrent code is placed. The `run` method is executed multiple times on different processors.

Due to OpenCL limitations, we are unable to use inheritance or method overloading. In addition, it does not like `println` in the `run` method, since the code may be running on a GPU. Aparapi only supports one-dimensional arrays. Arrays using two or more dimensions need to be flattened to a one dimension array. The support for double values is dependent on the OpenCL version and GPU configuration.

When a Java thread pool is used, it allocates one thread per CPU core. The kernel containing the Java code is cloned, one copy per thread. This avoids the need to access data across a thread. Each thread has access to information, such as a global ID, to assist in the code execution. The kernel will wait for all of the threads to complete.

Aparapi downloads can be found at <https://github.com/aparapi/aparapi/releases>.

Creating an Aparapi application

The basic framework for an Aparapi application is shown next. It consists of a `Kernel` derived class where the `run` method is overridden. In this example, the `run` method will perform scalar multiplication. This operation involves multiplying each element of a vector by some value.

The `ScalarMultiplicationKernel` extends the `Kernel` class. It possesses two instance variables used to hold the matrices for input and output. The constructor will initialize the matrices. The `run` method will perform the actual computations, and the `displayResult` method will show the results of the multiplication:

```
public class ScalarMultiplicationKernel extends Kernel {  
    float[] inputMatrix;  
    float outputMatrix [];  
  
    public ScalarMultiplicationKernel(float inputMatrix[]) {  
        ...  
    }  
  
    @Override  
    public void run() {  
        ...  
    }  
  
    public void displayResult() {  
        ...  
    }  
}
```

The constructor is shown here:

```
public ScalarMultiplicationKernel(float inputMatrix[]) {  
    this.inputMatrix = inputMatrix;  
    outputMatrix = new float[this.inputMatrix.length];  
}
```

In the `run` method, we use a global ID to index into the matrix. This code is executed on each computation unit, for example, a GPU or thread. A unique global ID is provided to each computational unit, allowing the code to access a specific element of the matrix. In this example, each element of the input matrix is multiplied by 2 and then assigned to the corresponding element of the output matrix:

```

public void run() {
    int globalID = this.getGlobalID();
    outputMatrix[globalID] = 2.0f * inputMatrix[globalID];
}

```

The `displayResult` method simply displays the contents of the `outputMatrix` array:

```

public void displayResult() {
    out.println("Result");
    for (float element : outputMatrix) {
        out.printf("%.4f ", element);
    }
    out.println();
}

```

To use this kernel, we need to declare variables for the `inputMatrix` and its size. The size will be used to control how many kernels to execute:

```

float inputMatrix[] = {3, 4, 5, 6, 7, 8, 9};
int size = inputMatrix.length;

```

The kernel is then created using the input matrix followed by the invocation of the `execute` method. This method starts the process and will eventually invoke the `Kernel` class' `run` method based on the `execute` method's argument. This argument is referred to as the pass ID. While not used in this example, we will use it in the next section. When the process is complete, the resulting output matrix is displayed and the `dispose` method is called to stop the process:

```

ScalarMultiplicationKernel kernel =
    new ScalarMultiplicationKernel(inputMatrix);
kernel.execute(size);
kernel.displayResult();
kernel.dispose();

```

When this application is executed we will get the following output:

```
6.0000 8.0000 10.0000 12.0000 14.0000 16.0000 18.000
```

We can specify the execution mode using the `Kernel` class' `setExecutionMode` method, as shown here:

```
kernel.setExecutionMode(Kernel.EXECUTION_MODE.GPU);
```

However, it is best to let Aparapi determine the execution mode. The following table summarizes the execution modes available:

Execution mode	Meaning
Kernel.EXECUTION_MODE.NONE	Does not specify mode
Kernel.EXECUTION_MODE.CPU	Use CPU
Kernel.EXECUTION_MODE.GPU	Use GPU
Kernel.EXECUTION_MODE.JTP	Use Java threads
Kernel.EXECUTION_MODE.SEQ	Use single loop (for debugging purposes)

Next, we will demonstrate how we can use Aparapi to perform dot product matrix multiplication.

Using Aparapi for matrix multiplication

We will use the matrices as used in the *Implementing basic matrix operations* section. We start with the declaration of the `MatrixMultiplicationKernel` class, which contains the vector declarations, a constructor, the `run` method, and a `displayResults` method. The vectors for matrices `A` and `B` have been flattened to one-dimensional arrays by allocating the matrices in row-column order:

```
class MatrixMultiplicationKernel extends Kernel {  
    float[] vectorA = {  
        0.1950f, 0.0311f, 0.3588f,  
        0.2203f, 0.1716f, 0.5931f,  
        0.2105f, 0.3242f};  
    float[] vectorB = {  
        0.0502f, 0.9823f, 0.9472f,  
        0.5732f, 0.2694f, 0.916f};  
    float[] vectorC;  
    int n;  
    int m;  
    int p;  
  
    @Override  
    public void run() {  
        ...  
    }  
  
    public MatrixMultiplicationKernel(int n, int m, int p) {  
        ...  
    }  
  
    public void displayResults () {  
        ...  
    }  
}
```

The `MatrixMultiplicationKernel` constructor assigns values for the matrices' dimensions and allocates memory for the result stored in `vectorC`, as shown here:

```
public MatrixMultiplicationKernel(int n, int m, int p) {  
    this.n = n;  
    this.p = p;  
    this.m = m;  
    vectorC = new float[n * p];  
}
```

The run method uses a global ID and a pass ID to perform the matrix multiplication. The pass ID is specified as the second argument of the Kernel class' execute method, as we will see shortly. This value allows us to advance the column index for vectorC. The vector indexes map to the corresponding row and column positions of the original matrices:

```
public void run() {
    int i = getGlobalId();
    int j = this.getPassId();
    float value = 0;
    for (int k = 0; k < p; k++) {
        value += vectorA[k + i * m] * vectorB[k * p + j];
    }
    vectorC[i * p + j] = value;
}
```

The displayResults method is shown as follows:

```
public void displayResults() {
    out.println("Result");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < p; j++) {
            out.printf("%.4f ", vectorC[i * p + j]);
        }
        out.println();
    }
}
```

The kernel is started in the same way as in the previous section. The execute method is passed the number of kernels that should be created and an integer indicating the number of passes to make. The number of passes is used to control the index into the vectorA and vectorB arrays:

```
MatrixMultiplicationKernel kernel = new MatrixMultiplicationKernel(n,
m,
p); kernel.execute(6, 3); kernel.displayResults();
kernel.dispose();
```

When this example is executed, you will get the following output:

```
Result
0.0276  0.1999  0.2132
0.1443  0.4118  0.5417
0.3486  0.3283  0.7058
0.1964  0.2941  0.4964
```

Next, we will see how Java 8 additions can contribute to solving math-intensive problems in a parallel manner.

Using Java 8 streams

The release of Java 8 came with a number of important enhancements to the language. The two enhancements of interest to us include lambda expressions and streams. A lambda expression is essentially an anonymous function that adds a functional programming dimension to Java. The concept of streams, as introduced in Java 8, does not refer to IO streams. Instead, you can think of it as a sequence of objects that can be generated and manipulated using a fluent style of programming. This style will be demonstrated shortly.

As with most APIs, programmers must be careful to consider the actual execution performance of their code using realistic test cases and environments. If not used properly, streams may not actually provide performance improvements. In particular, parallel streams, if not crafted carefully, can produce incorrect results.

We will start with a quick introduction to lambda expressions and streams. If you are familiar with these concepts you may want to skip over the next section.

Understanding Java 8 lambda expressions and streams

A lambda expression can be expressed in several different forms. The following illustrates a simple lambda expression where the symbol, `->`, is the lambda operator. This will take some value, `e`, and return the value multiplied by two. There is nothing special about the name `e`. Any valid Java variable name can be used:

```
e -> 2 * e
```

It can also be expressed in other forms, such as the following:

```
(int e) -> 2 * e  
(double e) -> 2 * e  
(int e) -> {return 2 * e;}
```

The form used depends on the intended value of `e`. Lambda expressions are frequently used as arguments to a method, as we will see shortly.

A stream can be created using a number of techniques. In the following example, a stream is created from an array. The `IntStream` interface is a type of stream that uses integers. The `Arrays` class' `stream` method converts an array into a stream:

```
IntStream stream = Arrays.stream(numbers);
```

We can then apply various `stream` methods to perform an operation. In the following statement, the `forEach` method will simply display each integer in the stream:

```
stream.forEach(e -> out.printf("%d ", e));
```

There are a variety of `stream` methods that can be applied to a stream. In the following example, the `mapToDouble` method will take an integer, multiply it by 2, and then return it as a `double`. The `forEach` method will then display these values:

```
stream  
    .mapToDouble(e-> 2 * e)  
    .forEach(e -> out.printf("%.4f ", e));
```

The cascading of method invocations is referred to as **fluent programming**.

Using Java 8 to perform matrix multiplication

Here, we will illustrate how streams can be used to perform matrix multiplication. The definitions of the `A`, `B`, and `C` matrices are the same as declared in the *Implementing basic matrix operations* section. They are duplicated here for your convenience:

```
double A[][] = {  
    {0.1950, 0.0311},  
    {0.3588, 0.2203},  
    {0.1716, 0.5931},  
    {0.2105, 0.3242}};  
double B[][] = {  
    {0.0502, 0.9823, 0.9472},  
    {0.5732, 0.2694, 0.916}};  
double C[][] = new double[n][p];
```

The following sequence is a stream implementation of matrix multiplication. A detailed explanation of the code follows:

```
C = Arrays.stream(A)  
    .parallel()  
    .map(AMatrixRow -> IntStream.range(0, B[0].length)  
        .mapToDouble(i -> IntStream.range(0, B.length)  
            .mapToDouble(j -> AMatrixRow[j] * B[j][i]))  
        .sum())  
    .toArray()).toArray(double[][]::new);
```

The first `map` method, shown as follows, creates a stream of double vectors representing the 4 rows of the `A` matrix. The `range` method will return a list of stream elements ranging from its first argument to the second argument.

```
.map(AMatrixRow -> IntStream.range(0, B[0].length))
```

The variable `i` corresponds to the numbers generated by the second `range` method, which corresponds to the number of rows in the `B` matrix (2). The variable `j` corresponds to the numbers generated by the third `range` method, representing the number of columns of the `B` matrix (3).

At the heart of the statement is the matrix multiplication, where the `sum` method calculates the sum:

```
.mapToDouble(j -> AMatrixRow[j] * B[j][i])
```

```
.sum()
```

The last part of the expression creates the two-dimensional array for the C matrix. The operator, `::new`, is called a method reference and is a shorter way of invoking the new operator to create a new object:

```
).toArray()).toArray(double[][]::new);
```

The `displayResult` method is as follows:

```
public void displayResult() {  
    out.println("Result");  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < p; j++) {  
            out.printf("%.4f ", C[i][j]);  
        }  
        out.println();  
    }  
}
```

The output of this sequence follows:

```
Result  
0.0276 0.1999 0.2132  
0.1443 0.4118 0.5417  
0.3486 0.3283 0.7058  
0.1964 0.2941 0.4964
```

Using Java 8 to perform map-reduce

In the next section, we will use Java 8 streams to perform a map-reduce operation similar to the one demonstrated using Hadoop in the *Using map-reduce* section. In this example, we will use a Stream of Book objects. We will then demonstrate how to use the Java 8 `reduce` and `average` methods to get our total page count and average page count.

Rather than begin with a text file, as we did in the Hadoop example, we have created a `Book` class with title, author, and page-count fields. In the `main` method of the driver class, we have created new instances of `Book` and added them to an `ArrayList` called `books`. We have also created a `double` value `average` to hold our average, and initialized our variable `totalPg` to zero:

```
ArrayList<Book> books = new ArrayList<>();
double average;
int totalPg = 0;

books.add(new Book("Moby Dick", "Herman Melville", 822));
books.add(new Book("Charlotte's Web", "E.B. White", 189));
books.add(new Book("The Grapes of Wrath", "John Steinbeck", 212));
books.add(new Book("Jane Eyre", "Charlotte Bronte", 299));
books.add(new Book("A Tale of Two Cities", "Charles Dickens", 673));
books.add(new Book("War and Peace", "Leo Tolstoy", 1032));
books.add(new Book("The Great Gatsby", "F. Scott Fitzgerald", 275));
```

Next, we perform a map and reduce operation to calculate the total number of pages in our set of books. To accomplish this in a parallel manner, we use the `stream` and `parallel` methods. We then use the `map` method with a lambda expression to accumulate all of the page counts from each `Book` object. Finally, we use the `reduce` method to merge our page counts into one final value, which is to be assigned to `totalPg`:

```
totalPg = books
    .stream()
    .parallel()
    .map((b) -> b.pgCnt)
    .reduce(totalPg, (accumulator, _item) -> {
        out.println(accumulator + " " + _item);
        return accumulator + _item;
    });
```

Notice in the preceding `reduce` method we have chosen to print out information about the reduction operation's cumulative value and individual items. The `accumulator` represents the aggregation of our page counts. The `_item` represents the individual task within the map-reduce process undergoing reduction at any given moment.

In the output that follows, we will first see the `accumulator` value stay at zero as each individual book item is processed. Gradually, the `accumulator` value increases. The final operation is the reduction of the values 1223 and 2279. The sum of these two numbers is 3502, or the total page count for all of our books:

```
0 822
0 189
0 299
0 673
0 212
299 673
0 1032

0 275
1032 275
972 1307
189 212
822 401
1223 2279
```

Next, we will add code to calculate the average page count of our set of books. We multiply our `totalPg` value, determined using map-reduce, by `1.0` to prevent truncation when we divide by the integer returned by the `size` method. We then print out `average`.

```
average = 1.0 * totalPg / books.size();
out.printf("Average Page Count: %.4f\n", average);
```

Our output is as follows:

Average Page Count: 500.2857

We could have used Java 8 streams to calculate the average directly using the `map` method. Add the following code to the `main` method. We use `parallelStream` with our `map` method to simultaneously get the page count for each of our books. We then use `mapToDouble` to ensure our data is of the correct type to calculate our average. Finally, we use the `average` and `getAsDouble` methods to calculate our average page

count:

```
average = books
    .parallelStream()
    .map(b -> b.pgCnt)
    .mapToDouble(s -> s)
    .average()
    .getAsDouble();
out.printf("Average Page Count: %.4f\n", average);
```

Then we print out our average. Our output, identical to our previous example, is as follows:

```
Average Page Count: 500.2857
```

These techniques made use of Java 8 capabilities related to the map-reduce framework to solve numeric problems. This type of process can also be applied to other types of data, including text-based data. The true benefit is seen when these processes handle extremely large datasets within a greatly reduced time frame.

Summary

Data science uses math extensively to analyze problems. There are numerous Java math libraries available, many of which support concurrent operations. In this chapter, we introduced a number of libraries and techniques to provide some insight into how they can be used to support and improve the performance of applications.

We started with a discussion of how simple matrix multiplication is performed. A basic Java implementation was presented. In later sections, we duplicated the implementation using other APIs and technologies.

Many higher level APIs, such as DL4J, support a number of useful data analysis techniques. Beneath these APIs often lies concurrent support for multiple CPUs and GPUs. Sometimes this support is configurable, as is the case for DL4J. We briefly discussed how we can configure ND4J to support multiple processors.

The map-reduce algorithm has found extensive use in the data science community. We took advantage of the parallel processing power of this framework to calculate the average of a given set of values, the page counts for a set of books. This technique used Apache's Hadoop to perform the map and reduce functions.

Mathematical techniques are supported by a large number of libraries. Many of these libraries do not directly support parallel operations. However, understanding what is available and how they can be used is important. To that end, we demonstrated how three different Java APIs can be used: jblas, Apache Commons Math, and ND4J.

OpenCL is an API that supports parallel operations on a variety of hardware platforms, processor types, and languages. This support is fairly low level. There are a number of Java bindings for OpenCL, which we reviewed.

Aparapi is a higher level of support for Java that can use CPUs, CUDA, or OpenCL to effect parallel operations. We demonstrated this support using the matrix multiplication example.

We wrapped up our discussion with an introduction to Java 8 streams and lambda expressions. These language elements can support parallel operations to improve an application's performance. In addition, this can often provide a more elegant and more maintainable implementation once the programmer becomes familiar with the

techniques. We also demonstrated techniques for performing map-reduce using Java 8.

In the next chapter, we will conclude the book by illustrating how many of the techniques introduced can be used to build a complete application.

Chapter 12. Bringing It All Together

While we have demonstrated many aspects of using Java to support data science tasks, the need to combine and use these techniques in an integrated manner exists. It is one thing to use the techniques in isolation and another to use them in a cohesive fashion. In this chapter, we will provide you with additional experience with these technologies and insights into how they can be used together.

Specifically, we will create a console-based application that analyzes tweets related to a user-defined topic. Using a console-based application allows us to focus on data-science-specific technologies and avoids having to choose a specific GUI technology that may not be relevant to us. It provides a common base from which a GUI implementation can be created if needed.

The application performs and illustrates the following high-level tasks:

- Data acquisition
- Data cleaning, including:
 - Removing stop words
 - Cleaning the text
 - Sentiment analysis
 - Basic data statistic collection
 - Display of results

More than one type of analysis can be used with many of these steps. We will show the more relevant approaches and allude to other possibilities as appropriate. We will use Java 8's features whenever possible.

Defining the purpose and scope of our application

The application will prompt the user for a set of selection criteria, which include topic and sub-topic areas, and the number of tweets to process. The analysis performed will simply compute and display the number of positive and negative tweets for a topic and sub-topic. We used a generic sentiment analysis model, which will affect the quality of the sentiment analysis. However, other models and more analysis can be added.

We will use a Java 8 stream to structure the processing of tweet data. It is a stream of `TweetHandler` objects, as we will describe shortly.

We use several classes in this application. They are summarized here:

- `TweetHandler`: This class holds the raw tweet text and specific fields needed for the processing including the actual tweet, username, and similar attributes.
- `TwitterStream`: This is used to acquire the application's data. Using a specific class separates the acquisition of the data from its processing. The class possesses a few fields that control how the data is acquired.
- `ApplicationDriver`: This contains the `main` method, user prompts, and the `TweetHandler` stream that controls the analysis.

Each of these classes will be detailed in later sections. However, we will present `ApplicationDriver` next to provide an overview of the analysis process and how the user interacts with the application.

Understanding the application's architecture

Every application has its own unique structure, or architecture. This architecture provides the overarching organization or framework for the application. For this application, we combine the three classes using a Java 8 stream in the `ApplicationDriver` class. This class consists of three methods:

- `ApplicationDriver`: Contains the applications' user input
- `performAnalysis`: Performs the analysis
- `main`: Creates the `ApplicationDriver` instance

The class structure is shown next. The three instance variables are used to control the processing:

```
public class ApplicationDriver {  
    private String topic;  
    private String subTopic;  
    private int numberOfTweets;  
  
    public ApplicationDriver() { ... }  
    public void performAnalysis() { ... }  
  
    public static void main(String[] args) {  
        new ApplicationDriver();  
    }  
}
```

The `ApplicationDriver` constructor follows. A `Scanner` instance is created and the sentiment analysis model is built:

```
public ApplicationDriver() {  
    Scanner scanner = new Scanner(System.in);  
    TweetHandler swt = new TweetHandler();  
    swt.buildSentimentAnalysisModel();  
    ...  
}
```

The remainder of the method prompts the user for input and then calls the `performAnalysis` method:

```
out.println("Welcome to the Tweet Analysis Application");
```

```

out.print("Enter a topic: ");
this.topic = scanner.nextLine();
out.print("Enter a sub-topic: ");
this.subTopic = scanner.nextLine().toLowerCase();
out.print("Enter number of tweets: ");
this.numberOfTweets = scanner.nextInt();
performAnalysis();

```

The `performAnalysis` method uses a Java 8 `Stream` instance obtained from the `TwitterStream` instance. The `TwitterStream` class constructor uses the number of tweets and `topic` as input. This class is discussed in the *Data acquisition using Twitter* section:

```

public void performAnalysis() {
    Stream<TweetHandler> stream = new TwitterStream(
        this.numberOfTweets, this.topic).stream();
    ...
}

```

The stream uses a series of `map`, `filter`, and a `forEach` method to perform the processing. The `map` method modifies the stream's elements. The `filter` methods remove elements from the stream. The `forEach` method will terminate the stream and generate the output.

The individual methods of the stream are executed in order. When acquired from a public Twitter stream, the Twitter information arrives as a JSON document, which we process first. This allows us to extract relevant tweet information and set the data to fields of the `TweetHandler` instance. Next, the text of the tweet is converted to lowercase. Only English tweets are processed and only those tweets that contain the sub-topic will be processed. The tweet is then processed. The last step computes the statistics:

```

stream
    .map(s -> s.processJSON())
    .map(s -> s.toLowerCase())
    .filter(s -> s.isEnglish())
    .map(s -> s.removeStopWords())
    .filter(s -> s.containsCharacter(this.subTopic))
    .map(s -> s.performSentimentAnalysis())
    .forEach((TweetHandler s) -> {
        s.computeStats();
        out.println(s);
    });

```

The results of the processing are then displayed:

```
out.println();
out.println("Positive Reviews: "
+ TweetHandler.getNumberOfPositiveReviews());
out.println("Negative Reviews: "
+ TweetHandler.getNumberOfNegativeReviews());
```

We tested our application on a Monday night during a Monday-night football game and used the topic #MNF. The # symbol is called a **hashtag** and is used to categorize tweets. By selecting a popular category of tweets, we ensured that we would have plenty of Twitter data to work with. For simplicity, we chose the football subtopic. We also chose to only analyze 50 tweets for this example. The following is an abbreviated sample of our prompts, input, and output:

```
Building Sentiment Model
Welcome to the Tweet Analysis Application
Enter a topic: #MNF
Enter a sub-topic: football
Enter number of tweets: 50
Creating Twitter Stream
51 messages processed!
Text: rt @ bleacherreport : touchdown , broncos ! c . j . anderson
punches ! lead , 7 - 6 # mnf # denvshou
Date: Mon Oct 24 20:28:20 CDT 2016
Category: neg
...
Text: i cannot emphasize enough how big td drive . @ broncos offense .
needed confidence booster & ; just got . # mnf # denvshou
Date: Mon Oct 24 20:28:52 CDT 2016
Category: pos
Text: least touchdown game . # mnf
Date: Mon Oct 24 20:28:52 CDT 2016
Category: neg
Positive Reviews: 13
Negative Reviews: 27
```

We print out the text of each tweet, along with a timestamp and category. Notice that the text of the tweet does not always make sense. This may be due to the abbreviated nature of Twitter data, but it is partially due to the fact this text has been cleaned and stop words have been removed. We should still see our topic, #MNF, although it will be lowercase due to our text cleaning. At the end, we print out the total number of tweets classified as positive and negative.

The classification of tweets is done by the `performSentimentAnalysis` method.

Notice the process of classification using sentiment analysis is not always precise. The following tweet mentions a touchdown by a Denver Broncos player. This tweet could be construed as positive or negative depending on an individual's personal feelings about that team, but our model classified it as positive:

```
Text: cj anderson td run @ broncos . broncos now lead 7 - 6 . # mnf  
Date: Mon Oct 24 20:28:42 CDT 2016  
Category: pos
```

Additionally, some tweets may have a neutral tone, such as the one shown next, but still be classified as either positive or negative. The following tweet is a retweet of a popular sports news twitter handle, @bleacherreport:

```
Text: rt @ bleacherreport : touchdown , broncos ! c . j . anderson  
punches ! lead , 7 - 6 # mnf # denvshou  
Date: Mon Oct 24 20:28:37 CDT 2016  
Category: neg
```

This tweet has been classified as negative but perhaps could be considered neutral. The contents of the tweet simply provide information about a score in a football game. Whether this is a positive or negative event will depend upon which team a person may be rooting for. When we examine the entire set of tweet data analysed, we notice that this same @bleacherreport tweet has been retweeted a number of times and classified as negative each time. This could skew our analysis when we consider that we may have a large number of improperly classified tweets. Using incorrect data decreases the accuracy of the results.

One option, depending on the purpose of analysis, may be to exclude tweets by news outlets or other popular Twitter users. Additionally we could exclude tweets with *RT*, an abbreviation denoting that the tweet is a retweet of another user.

There are additional issues to consider when performing this type of analysis, including the sub-topic used. If we were to analyze the popularity of a Star Wars character, then we would need to be careful which names we use. For example, when choosing a character name such as Han Solo, the tweet may use an alias. Aliases for Han Solo include Vykk Draygo, Rysto, Jenos Idanian, Solo Jaxal, Master Marksman, and Jobekk Jonn, to mention a few (http://starwars.wikia.com/wiki/Category:Han_Solo_aliases). The actor's name may be used instead of the actual character, which is Harrison Ford in the case of Han Solo. We may also want to consider the actor's nickname, such as Harry for Harrison.

Data acquisition using Twitter

The Twitter API is used in conjunction with HBC's HTTP client to acquire tweets, as previously illustrated in the Handling Twitter section of [Chapter 2, Data Acquisition](#). This process involves using the public stream API at the default access level to pull a sample of public tweets currently streaming on Twitter. We will refine the data based on user-selected keywords.

To begin, we declare the `TwitterStream` class. It consists of two instance variables, (`numberOfTweets` and `topic`), two constructors, and a `stream` method. The `numberOfTweets` variable contains the number of tweets to select and process, and `topic` allows the user to search for tweets related to a specific topic. We have set our default constructor to pull 100 tweets related to Star Wars:

```
public class TwitterStream {  
    private int numberOfTweets;  
    private String topic;  
  
    public TwitterStream() {  
        this(100, "Stars Wars");  
    }  
  
    public TwitterStream(int numberOfTweets, String topic) { ... }  
}
```

The heart of our `TwitterStream` class is the `stream` method. We start by performing authentication using the information provided by Twitter when we created our Twitter application. We then create a `BlockingQueue` object to hold our streaming data. In this example, we will set a default capacity of 1000. We use our `topic` variable in the `trackTerms` method to specify the types of tweets we are searching for. Finally, we specify our `endpoint` and turn off stall warnings:

```
String myKey = "mySecretKey";  
String mySecret = "mySecret";  
String myToken = "myToKen";  
String myAccess = "myAccess";  
  
out.println("Creating Twitter Stream");  
BlockingQueue<String> statusQueue = new  
LinkedBlockingQueue<>(1000);  
StatusesFilterEndpoint endpoint = new StatusesFilterEndpoint();
```

```
endpoint.trackTerms(Lists.newArrayList("twitterapi", this.topic));
endpoint.stallWarnings(false);
```

Now we can create an `Authentication` object using `OAuth1`, a variation of the `OAuth` class. This allows us to build our connection client and complete the HTTP connection:

```
Authentication twitterAuth = new OAuth1(myKey, mySecret, myToken,
                                         myAccess);

BasicClient twitterClient = new ClientBuilder()
    .name("Twitter client")
    .hosts(Constants.STREAM_HOST)
    .endpoint(endpoint)
    .authentication(twitterAuth)
    .processor(new StringDelimitedProcessor(statusQueue))
    .build();

twitterClient.connect();
```

Next, we create two `ArrayLists`, `list` to hold our `TweetHandler` objects and `twitterList` to hold the JSON data streamed from Twitter. We will discuss the `TweetHandler` object in the next section. We use the `drainTo` method in place of the `poll` method demonstrated in [Chapter 2, Data Acquisition](#), because it can be more efficient for large amounts of data:

```
List<TweetHandler> list = new ArrayList();
List<String> twitterList = new ArrayList();
```

Next we loop through our retrieved messages. We call the `take` method to remove each string message from the `BlockingQueue` instance. We then create a new `TweetHandler` object using the message and place it in our `list`. After we have handled all of our messages and the for loop completes, we stop the HTTP client, display the number of messages, and return our stream of `TweetHandler` objects:

```
statusQueue.drainTo(twitterList);
for(int i=0; i<numberOfTweets; i++) {
    String message;
    try {
        message = statusQueue.take();
        list.add(new TweetHandler(message));
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

```
twitterClient.stop();
out.printf("%d messages processed!\n",
    twitterClient.getStatsTracker().getNumMessages()));

return list.stream();
}
```

We are now ready to clean and analyze our data.

Understanding the TweetHandler class

The `TweetHandler` class holds information about a specific tweet. It takes the raw JSON tweet and extracts those parts that are relevant to the application's needs. It also possesses the methods to process the tweet's text such as converting the text to lowercase and removing tweets that are not relevant. The first part of the class is shown next:

```
public class TweetHandler {  
    private String jsonText;  
    private String text;  
    private Date date;  
    private String language;  
    private String category;  
    private String userName;  
    ...  
    public TweetHandler processJSON() { ... }  
    public TweetHandler toLowerCase(){ ... }  
    public TweetHandler removeStopWords(){ ... }  
    public boolean isEnglish(){ ... }  
    public boolean containsCharacter(String character) { ... }  
    public void computeStats(){ ... }  
    public void buildSentimentAnalysisModel{ ... }  
    public TweetHandler performSentimentAnalysis(){ ... }  
}
```

The instance variables show the type of data retrieved from a tweet and processed, as detailed here:

- `jsonText`: The raw JSON text
- `text`: The text of the processed tweet
- `date`: The date of the tweet
- `language`: The language of the tweet
- `category`: The tweet classification, which is positive or negative
- `userName`: The name of the Twitter user

There are several other instance variables used by the class. The following are used to create and use a sentiment analysis model. The `classifier` static variable refers to the model:

```
private static String[] labels = {"neg", "pos"};  
private static int nGramSize = 8;  
private static DynamicLMClassifier<NGramProcessLM>
```

```
classifier = DynamicLMClassifier.createNGramProcess(  
    labels, nGramSize);
```

The default constructor is used to provide an instance to build the sentiment model. The single argument constructor creates a `TweetHandler` object using the raw JSON text:

```
public TweetHandler() {  
    this.jsonText = "";  
}  
  
public TweetHandler(String jsonText) {  
    this.jsonText = jsonText;  
}
```

The remainder of the methods are discussed in the following sections.

Extracting data for a sentiment analysis model

In [Chapter 9, Text Analysis](#), we used DL4J to perform sentiment analysis. We will use LingPipe in this example as an alternative to our previous approach. Because we want to classify Twitter data, we chose a dataset with pre-classified tweets, available at <http://thinknook.com/wp-content/uploads/2012/09/Sentiment-Analysis-Dataset.zip>. We must complete a one-time process of extracting this data into a format we can use with our model before we continue with our application development.

This dataset exists in a large `.csv` file with one tweet and classification per line. The tweets are classified as either `0` (negative) or `1` (positive). The following is an example of one line of this data file:

```
95,0,Sentiment140, - Longest night ever.. ugh!  
http://tumblr.com/xwp1yxhi6
```

The first element represents a unique ID number which is part of the original data set and which we will use for the filename. The second element is the classification, the third is a data set label (effectively ignored for the purposes of this project), and the last element is the actual tweet text. Before we can use this data with our LingPipe model, we must write each tweet into an individual file. To do this, we created three string variables. The `filename` variable will be assigned either `pos` or `neg` depending on each tweet's classification and will be used in the write operation. We also use the `file` variable to hold the name of the individual tweet file and the `text` variable to hold the individual tweet text. Next, we use the `readAllLines` method with the `Paths` class's `get` method to store our data in a `List` object. We need to specify the `charset`, `StandardCharsets.ISO_8859_1`, as well:

```
try {  
    String filename;  
    String file;  
    String text;  
    List<String> lines = Files.readAllLines(  
        Paths.get("\\path-to-file\\SentimentAnalysisDataset.csv"),  
        StandardCharsets.ISO_8859_1);  
    ...  
  
} catch (IOException ex) {  
    // Handle exceptions  
}
```

Now we can loop through our list and use the `split` method to store our `.csv` data in a string array. We convert the element at position `1` to an integer and determine whether it is a `1`. Tweets classified with a `1` are considered positive tweets and we set `filename` to `pos`. All other tweets set the `filename` to `neg`. We extract the output `filename` from the element at position `0` and the text from element `3`. We ignore the label in position `2` for the purposes of this project. Finally, we write out our data:

```
for(String s : lines) {  
    String[] oneLine = s.split(",");  
    if(Integer.parseInt(oneLine[1])==1) {  
        filename = "pos";  
    } else {  
        filename = "neg";  
    }  
    file = oneLine[0]+".txt";  
    text = oneLine[3];  
    Files.write(Paths.get(  
        path-to-file\\txt_sentoken"+filename+""+file),  
        text.getBytes());  
}
```

Notice that we created the `neg` and `pos` directories within the `txt_sentoken` directory. This location is important when we read the files to build our model.

Building the sentiment model

Now we are ready to build our model. We loop through the `labels` array, which contains `pos` and `neg`, and for each label we create a new `Classification` object. We then create a new file using this label and use the `listFiles` method to create an array of filenames. Next, we will traverse these filenames using a `for` loop:

```
public void buildSentimentAnalysisModel() {  
    out.println("Building Sentiment Model");  
  
    File trainingDir = new File("\\path to file\\txt_sentoken");  
    for (int i = 0; i < labels.length; i++) {  
        Classification classification =  
            new Classification(labels[i]);  
        File file = new File(trainingDir, labels[i]);  
        File[] trainingFiles = file.listFiles();  
        ...  
    }  
}
```

Within the `for` loop, we extract the tweet data and store it in our string, `review`. We then create a new `Classified` object using `review` and `classification`. Finally we can call the `handle` method to classify this particular text:

```
for (int j = 0; j < trainingFiles.length; j++) {  
    try {  
        String review = Files.readFromfile(trainingFiles[j],  
            "ISO-8859-1");  
        Classified<CharSequence> classified = new  
            Classified<>(review, classification);  
        classifier.handle(classified);  
    } catch (IOException ex) {  
        // Handle exceptions  
    }  
}
```

For the dataset discussed in the previous section, this process may take a substantial amount of time. However, we consider this time trade-off to be worth the quality of analysis made possible by this training data.

Processing the JSON input

The Twitter data is retrieved using JSON format. We will use Twitter4J (<http://twitter4j.org>) to extract the relevant parts of the tweet and store in the corresponding field of the TweetHandler class.

The TweetHandler class's processJSON method does the actual data extraction. An instance of the JSONObject is created based on the JSON text. The class possesses several methods to extract specific types of data from an object. We use the getString method to get the fields we need.

The start of the processJSON method is shown next, where we start by obtaining the JSONObject instance, which we will use to extract the relevant parts of the tweet:

```
public TweetHandler processJSON() {  
    try {  
        JSONObject jsonObject = new JSONObject(this.jsonText);  
        ...  
    } catch (JSONException ex) {  
        // Handle exceptions  
    }  
    return this;  
}
```

First, we extract the tweet's text as shown here:

```
this.text = jsonObject.getString("text");
```

Next, we extract the tweet's date. We use the SimpleDateFormat class to convert the date string to a Date object. Its constructor is passed a string that specifies the format of the date string. We used the string "EEE MMM d HH:mm:ss z yyyy", whose parts are detailed next. The order of the string elements corresponds to the order found in the JSON entity:

- EEE: Day of the week specified using three characters
- MMM: Month, using three characters
- d: Day of the month
- HH:mm:ss: Hours, minutes, and seconds
- z: Time zone
- yyyy: Year

The code follows:

```
SimpleDateFormat sdf = new SimpleDateFormat(  
    "EEE MMM d HH:mm:ss Z yyyy");  
try {  
    this.date = sdf.parse(jsonObject.getString("created_at"));  
} catch (ParseException ex) {  
    // Handle exceptions  
}
```

The remaining fields are extracted as shown next. We had to extract an intermediate JSON object to extract the name field:

```
this.language = jsonObject.getString("lang");  
JSONObject user = jsonObject.getJSONObject("user");  
this.userName = user.getString("name");
```

Having acquired and extracted the text, we are now ready to perform the important task of cleaning the data.

Cleaning data to improve our results

Data cleaning is a critical step in most data science problems. Data that is not properly cleaned may have errors such as misspellings, inconsistent representation of elements such as dates, and extraneous words.

There are numerous data cleaning options that we can apply to Twitter data. For this application, we perform simple cleaning. In addition, we will filter out certain tweets.

The conversion of the text to lowercase letters is easily achieved as shown here:

```
public TweetHandler toLowerCase() {  
    this.text = this.text.toLowerCase().trim();  
    return this;  
}
```

Part of the process is to remove certain tweets that are not needed. For example, the following code illustrates how to detect whether the tweet is in English and whether it contains a sub-topic of interest to the user. The `boolean` return value is used by the `filter` method in the Java 8 stream, which performs the actual removal:

```
public boolean isEnglish() {  
    return this.language.equalsIgnoreCase("en");  
}  
  
public boolean containsCharacter(String character) {  
    return this.text.contains(character);  
}
```

Numerous other cleaning operations can be easily added to the process such as removing leading and trailing white space, replacing tabs, and validating dates and email addresses.

Removing stop words

Stop words are those words that do not contribute to the understanding or processing of data. Typical stop words include the 0, and, a, and or. When they do not contribute to the data process, they can be removed to simplify processing and make it more efficient.

There are several techniques for removing stop words, as discussed in [Chapter 9, Text Analysis](#). For this application, we will use LingPipe (<http://alias-i.com/lingpipe/>) to remove stop words. We use the `EnglishStopTokenizerFactory` class to obtain a model for our stop words based on an `IndoEuropeanTokenizerFactory` instance:

```
public TweetHandler removeStopWords() {  
    TokenizerFactory tokenizerFactory  
        = IndoEuropeanTokenizerFactory.INSTANCE;  
    tokenizerFactory =  
        new EnglishStopTokenizerFactory(tokenizerFactory);  
    ...  
    return this;  
}
```

A series of tokens that do not contain stop words are extracted, and a `StringBuilder` instance is used to create a string to replace the original text:

```
Tokenizer tokens = tokenizerFactory.tokenizer(  
    this.text.toCharArray(), 0, this.text.length());  
StringBuilder buffer = new StringBuilder();  
for (String word : tokens) {  
    buffer.append(word + " ");  
}  
this.text = buffer.toString();
```

The LingPipe model we used may not be the best suited for all tweets. In addition, it has been suggested that removing stop words from tweets may not be productive (<http://oro.open.ac.uk/40666/>). Options to select various stop words and whether stop words should even be removed can be added to the stream process.

Performing sentiment analysis

We can now perform sentiment analysis using the model built in the Building the sentiment model section of this chapter. We create a new `Classification` object by passing our cleaned text to the `classify` method. We then use the `bestCategory` method to classify our text as either positive or negative. Finally, we set `category` to the result and return the `TweetHandler` object:

```
public TweetHandler performSentimentAnalysis() {  
    Classification classification =  
        classifier.classify(this.text);  
    String bestCategory = classification.bestCategory();  
    this.category = bestCategory;  
    return this;  
}
```

We are now ready to analyze the results of our application.

Analysing the results

The analysis performed in this application is fairly simple. Once the tweets have been classified as either positive or negative, a total is computed. We used two static variables for this purpose:

```
private static int numberOfPositiveReviews = 0;  
private static int numberOfNegativeReviews = 0;
```

The `computeStats` method is called from the Java 8 stream and increments the appropriate variable:

```
public void computeStats() {  
    if(this.category.equalsIgnoreCase("pos")) {  
        numberOfPositiveReviews++;  
    } else {  
        numberOfNegativeReviews++;  
    }  
}
```

Two static methods provide access to the number of reviews:

```
public static int getNumberOfPositiveReviews() {  
    return numberOfPositiveReviews;  
}  
  
public static int getNumberOfNegativeReviews() {  
    return numberOfNegativeReviews;  
}
```

In addition, a simple `toString` method is provided to display basic tweet information:

```
public String toString() {  
    return "\nText: " + this.text  
        + "\nDate: " + this.date  
        + "\nCategory: " + this.category;  
}
```

More sophisticated analysis can be added as required. The intent of this application was to demonstrate a technique for combining the various data processing tasks.

Other optional enhancements

There are numerous improvements that can be made to the application. Many of these are user preferences and others relate to improving the results of the application. A GUI interface would be useful in many situations. Among the user options, we may want add support for:

- Displaying individual tweets
- Allowing null sub-topics
- Processing other tweet fields
- Providing list of topics or sub-topics the user can choose from
- Generating additional statistics and supporting charts

With regard to process result improvements, the following should be considered:

- Correct user entries for misspelling
- Remove spacing around punctuation
- Use alternate stop word removal techniques
- Use alternate sentiment analysis techniques

The details of many of these enhancements are dependent on the GUI interface used and the purpose and scope of the application.

Summary

The intent of this chapter was to illustrate how various data science tasks can be integrated into an application. We chose an application that processes tweets because it is a popular social medium and allows us to apply many of the techniques discussed in earlier chapters.

A simple console-based interface was used to avoid cluttering the discussion with specific but possibly irrelevant GUI details. The application prompted the user for a Twitter topic, a sub-topic, and the number of tweets to process. The analysis consisted of determining the sentiments of the tweets, with simple statistics regarding the positive or negative nature of the tweets.

The first step in the process was to build a sentiment model. We used LingPipe classes to build a model and perform the analysis. A Java 8 stream was used and supported a fluent style of programming where the individual processing steps could be easily added and removed.

Once the stream was created, the JSON raw text was processed and used to initialize a `TweetHandler` class. Instances of this class were subsequently modified, including converting the text to lowercase, removing non-English tweets, removing stop words, and selecting only those tweets that contain the sub-topic. Sentiment analysis was then performed, followed by the computation of the statistics.

Data science is a broad topic that utilizes a wide range of statistical and computer science topics. In this book, we provided a brief introduction to many of these topics and how they are supported by Java.

Part 2. Module 2

Machine Learning in Java

Design, build, and deploy your own machine learning applications by leveraging key Java machine learning libraries

Chapter 1. Applied Machine Learning Quick Start

This chapter introduces the basics of machine learning, laying down the common themes and concepts and making it easy to follow the logic and familiarize yourself with the topic. The goal is to quickly learn the step-by-step process of **applied machine learning** and grasp the main machine learning principles. In this chapter, we will cover the following topics:

- Introducing machine learning and its relation to data science
- Discussing the basic steps in applied machine learning
- Discussing the kind of data we are dealing with and its importance
- Discussing approaches of collecting and preprocessing the data
- Making sense of data using machine learning
- Using machine learning to extract insights from data and build predictors

If you are already familiar with machine learning and are eager to start coding, then quickly jump to the following chapters. However, if you need to refresh your memory or clarify some concepts, then it is strongly recommend to revisit the topics presented in this chapter.

Machine learning and data science

Nowadays, everyone talks about machine learning and data science. So, what exactly is machine learning anyway? How does it relate to data science? These two terms are commonly confused, as they often employ the same methods and overlap significantly. Therefore, let's first clarify what they are.

Josh Wills tweeted:

"Data scientist is a person who is better at statistics than any software engineer and better at software engineering than any statistician".

--(Josh Wills)

More specifically, data science encompasses the entire process of obtaining knowledge from data by integrating methods from statistics, computer science, and other fields to gain insight from data. In practice, data science encompasses an iterative process of data harvesting, cleaning, analysis and visualization, and deployment.

Machine learning, on the other hand, is mainly concerned with fairly generic algorithms and techniques that are used in analysis and modeling phases of data science process. Arthur Samuel proposed the following definition back in 1995:

"Machine Learning relates with the study, design and development of the algorithms that give computers the capability to learn without being explicitly programmed."

--Arthur Samuel

What kind of problems can machine learning solve?

Among the different machine learning approaches, there are three main ways of learning, as shown in the following list:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Given a set of example inputs, X , and their outcomes, Y , supervised learning aims to learn a general mapping function, f , that transforms inputs to outputs, as $f: X \rightarrow Y$

An example of supervised learning is credit card fraud detection, where the learning algorithm is presented with credit card transactions (matrix X) marked as normal or suspicious. (vector Y). The learning algorithm produces a decision model that marks unseen transactions as normal or suspicious (that is the f function).

In contrast, unsupervised learning algorithms do not assume given outcome labels, Y as they focus on learning the structure of the data, such as grouping similar inputs into clusters. Unsupervised learning can, hence, discover hidden patterns in the data. An example of unsupervised learning is an item-based recommendation system, where the learning algorithm discovers similar items bought together, for example, *people who bought book A also bought book B*.

Reinforcement learning addresses the learning process from a completely different angle. It assumes that an agent, which can be a robot, bot, or computer program, interacts with a dynamic environment to achieve a specific goal. The environment is described with a set of states and the agent can take different actions to move from one state to another. Some states are marked as goal states and if the agent achieves this state, it receives a large reward. In other states, the reward is smaller, non-existing, or even negative. The goal of reinforcement learning is to find an optimal policy, that is, a mapping function that specifies the action to take in each of the states without a teacher explicitly telling whether this leads to the goal state or not. An example of reinforcement learning is a program for driving a vehicle, where the states correspond to the driving conditions—for example, current speed, road segment information, surrounding traffic, speed limits, and obstacles on the road—and the actions can be driving maneuvers such as turn left or right, stop, accelerate,

and continue. The learning algorithm produces a policy that specifies the action that is to be taken in specific configuration of driving conditions.

In this book, we will focus on supervised and unsupervised learning only, as they share many concepts. If reinforcement learning sparked your interest, a good book to start with is *Reinforcement Learning: An Introduction* by Richard S. Sutton and Andrew Barto.

Applied machine learning workflow

This book's emphasis is on applied machine learning. We want to provide you with the practical skills needed to get learning algorithms to work in different settings. Instead of math and theory of machine learning, we will spend more time on the practical, hands-on skills (and dirty tricks) to get this stuff to work well on an application. We will focus on supervised and unsupervised machine learning and cover the essential steps from data science to build the applied machine learning workflow.

A typical workflow in applied machine learning applications consists of answering a series of questions that can be summarized in the following five steps:

1. **Data and problem definition:** The first step is to ask interesting questions. What is the problem you are trying solve? Why is it important? Which format of result answers your question? Is this a simple yes/no answer? Do you need to pick one of the available questions?
2. **Data collection:** Once you have a problem to tackle, you will need the data. Ask yourself what kind of data will help you answer the question. Can you get the data from the available sources? Will you have to combine multiple sources? Do you have to generate the data? Are there any sampling biases? How much data will be required?
3. **Data preprocessing:** The first data preprocessing task is data cleaning. For example, filling missing values, smoothing noisy data, removing outliers, and resolving inconsistencies. This is usually followed by integration of multiple data sources and data transformation to a specific range (normalization), to value bins (discretized intervals), and to reduce the number of dimensions.
4. **Data analysis and modeling with unsupervised and supervised learning:** Data analysis and modeling includes unsupervised and supervised machine learning, statistical inference, and prediction. A wide variety of machine learning algorithms are available, including k-nearest neighbors, naïve Bayes, decision trees, support vector machines, logistic regression, k-means, and so on. The choice of method to be deployed depends on the problem definition discussed in the first step and the type of collected data. The final product of this step is a model inferred from the data.
5. **Evaluation:** The last step is devoted to model assessment. The main issue models built with machine learning face is how well they model the underlying data—if a model is too specific, that is, it **overfits** to the data used for training,

it is quite possible that it will not perform well on a new data. The model can be too generic, meaning that it **underfits** the training data. For example, when asked how the weather is in California, it always answers sunny, which is indeed correct most of the time. However, such a model is not really useful for making valid predictions. The goal of this step is to correctly evaluate the model and make sure it will work on new data as well. Evaluation methods include separate test and train set, cross-validation, and leave-one-out validation.

In the following sections, we will take a closer look at each of the steps. We will try to understand the type of questions we must answer during the applied machine learning workflow and also look at the accompanying concepts of data analysis and evaluation.

Data and problem definition

Data is simply a collection of measurements in the form of numbers, words, measurements, observations, descriptions of things, images, and so on.

Measurement scales

The most common way to represent the data is using a set of attribute-value pairs. Consider the following example:

```
Bob = {  
height: 185cm,  
eye color: blue,  
hobbies: climbing, sky diving  
}
```

For example, Bob has attributes named height, eye color, and hobbies with values 185cm, blue, climbing, sky diving, respectively.

A set of data can be simply presented as a table, where columns correspond to attributes or features and rows correspond to particular data examples or instances. In supervised machine learning, the attribute whose value we want to predict the outcome, Y , from the values of the other attributes, X , is denoted as class or the target variable, as follows:

Name	Height [cm]	Eye color	Hobbies
Bob	185.0	Blue	Climbing, sky diving
Anna	163.0	Brown	Reading
...

The first thing we notice is how varying the attribute values are. For instance, height is a number, eye color is text, and hobbies are a list. To gain a better understanding of the value types, let's take a closer look at the different types of data or measurement scales. Stevens (1946) defined the following four scales with increasingly more expressive properties:

- **Nominal data** are mutually exclusive, but not ordered. Their examples include eye color, martial status, type of car owned, and so on.
- **Ordinal data** correspond to categories where order matters, but not the difference between the values, such as pain level, student letter grade, service quality rating, IMDB movie rating, and so on.
- **Interval data** where the difference between two values is meaningful, but there

is no concept of zero. For instance, standardized exam score, temperature in Fahrenheit, and so on.

- **Ratio data** has all the properties of an interval variable and also a clear definition of zero; when the variable equals to zero, there is none of this variable. Variables such as height, age, stock price, and weekly food spending are ratio variables.

Why should we care about measurement scales? Well, machine learning heavily depends on the statistical properties of the data; hence, we should be aware of the limitations each data type possesses. Some machine learning algorithms can only be applied to a subset of measurement scales.

The following table summarizes the main operations and statistics properties for each of the measurement types:

Property	Nominal	Ordinal	Interval	Ratio
Frequency of distribution	✓	✓	✓	✓
Mode and median		✓	✓	✓
Order of values is known		✓	✓	✓
Can quantify difference between each value			✓	✓
Can add or subtract values			✓	✓
Can multiply and divide values				✓
Has true zero				✓

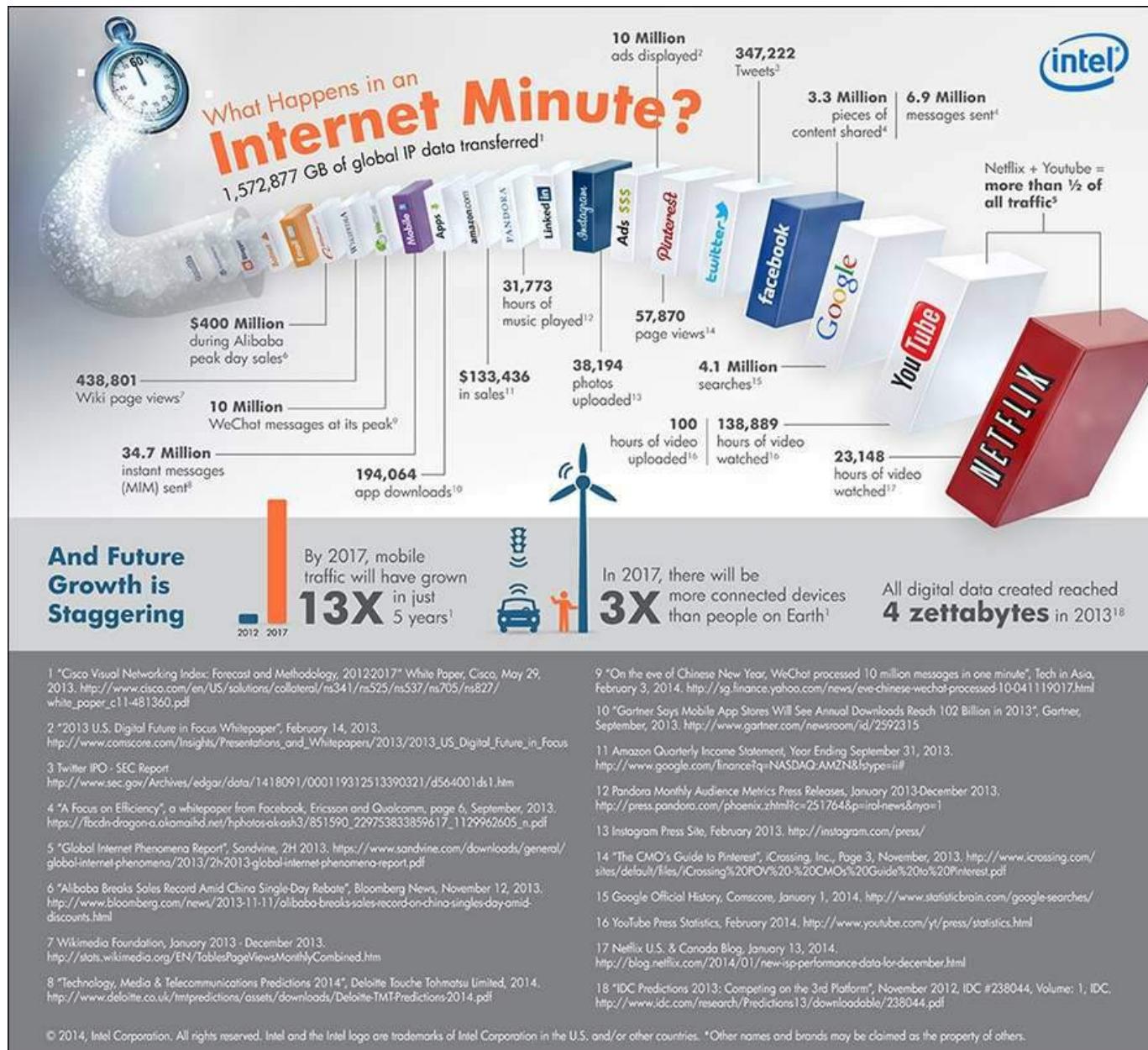
Furthermore, nominal and ordinal data correspond to discrete values, while interval and ratio data can correspond to continuous values as well. In supervised learning, the measurement scale of the attribute values that we want to predict dictates the kind of machine algorithm that can be used. For instance, predicting discrete values from a limited list is called classification and can be achieved using decision trees; while predicting continuous values is called regression, which can be achieved using model trees.

Data collection

So, where does the data come from? We have two choices: observe the data from existing sources or generate the data via surveys, simulations, and experiments. Let's take a closer look at both the approaches.

Find or observe data

Data can be found or observed at many places. An obvious data source is the Internet. Intel (2013) presented the following iconographic, showing the massive amount of data collected by different Internet services. In 2013, digital devices created four zettabytes (10^{21} = billion terabytes) of data. In 2017, it is expected that the number of connected devices will reach three times the number of people on earth; hence, the amount of data generated and collected will increase even further:



To get the data from the Internet, there are multiple options, as shown in the following:

- Bulk downloads from websites such as Wikipedia, IMDb, and Million Song database.
- Accessing the data through API (NY Times, Twitter, Facebook, Foursquare).
- Web scraping—It is OK to scrape public, non-sensitive, and anonymized data. Be sure to check terms of conditions and to fully reference information.

The main drawbacks of found data are that it takes time and space to accumulate the data; they cover only what happened, for instance, intentions, motivations, or internal motivations are not collected. Finally, such data might be noisy, incomplete, inconsistent, and may even change over time.

Another option is to collect measurements from sensors such as inertial and location sensors in mobile devices, environmental sensors, and software agents monitoring key performance indicators.

Generate data

An alternative approach is to generate the data by yourself, for example, with a survey. In survey design, we have to pay attention to data sampling, that is, who are the respondents answering the survey. We only get data from the respondents who are accessible and willing to respond. Also, respondents can provide answers that are in line with their self-image and researcher's expectations.

Next, the data can be collected with simulations, where a domain expert specifies behavior model of users at a micro level. For instance, crowd simulation requires specifying how different types of users will behave in crowd, for example, following the crowd, looking for an escape, and so on. The simulation can be then run under different conditions to see what happens (Tsai et al. 2011). Simulations are appropriate for studying macro phenomena and emergent behavior; however, they are typically hard to validate empirically.

Furthermore, you can design experiments to thoroughly cover all the possible outcomes, where you keep all the variables constant and only manipulate one variable at a time. This is the most costly approach, but usually provides the best quality of data.

Sampling traps

Data collection may involve many traps. To demonstrate one, let me share a story. There is supposed to be a global, unwritten rule for sending regular mail between students for free. If you write *student to student* to the place where the stamp should be, the mail is delivered to the recipient for free. Now, suppose Jacob sends a set of postcards to Emma, and given that Emma indeed receives some of the postcards, she concludes that all the postcards are delivered and that the rule indeed holds true. Emma reasons that as she received the postcards, all the postcards are delivered. However, she does not possess the information about the postcards that were sent by Jacob, but were undelivered; hence, she is unable to account this into her inference. What Emma experienced is **survivorship bias**, that is, she drew the conclusion based on the survived data only. For your information, the postcards that are being sent with *student to student* stamp get a circled black letter *T* stamp on them, which means *postage is due* and that receiver should pay it, including a small fine. However, mail services often have higher costs on applying such fee and hence do not do it (Magalhães, 2010).

Another example is a study, which found that the profession with the lowest average age of death was student. Being a student does not cause you to die at an early age, being a student means you are young. This is what makes the average of those that die so low (Gelman and Nolan, 2002).

Furthermore, a study that found that only 1.5% of drivers in accidents reported they were using a cell phone, whereas 10.9% reported another occupant in the car distracted them. Can we conclude that using a cell phone is safer than speaking with another occupant (Uts, 2003)? To answer this question, we need to know the prevalence of the cell phone use. It is likely that a higher number of people talked to another occupant in the car while driving than talking on the cell during the period when the data was collected.

Data pre-processing

The goal of data pre-processing tasks is to prepare the data for a machine learning algorithm in the best possible way as not all algorithms are capable of addressing issues with missing data, extra attributes, or denormalized values.

Data cleaning

Data cleaning, also known as data cleansing or data scrubbing, is the process of the following:

- Identifying inaccurate, incomplete, irrelevant, or corrupted data to remove it from further processing
- Parsing data, extracting information of interest, or validating whether a string of data is in an acceptable format
- Transforming data into a common encoding format, for example, utf-8 or int32, time scale, or normalized range
- Transforming data into a common data schema, for instance, if we collect temperature measurements from different types of sensors, we might want them to have the same structure

Now, let's look at some more concrete pre-processing steps.

Fill missing values

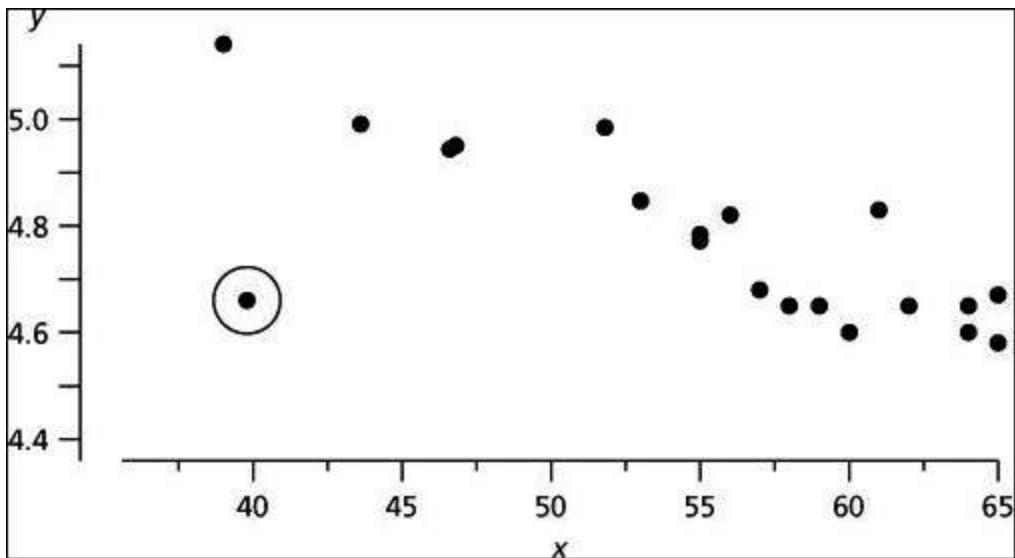
Machine learning algorithms generally do not work well with missing values. Rare exceptions include decision trees, naïve Bayes classifier, and some rule-based learners. It is very important to understand why a value is missing. It can be missing due to many reasons such as random error, systematic error, and sensor noise. Once we identified the reason, there are multiple ways to deal with the missing values, as shown in the following list:

- **Remove the instance:** If there is enough data, and only a couple of non-relevant instances have some missing values, then it is safe to remove these instances.
- **Remove the attribute:** Removing an attribute makes sense when most of the values are missing, values are constant, or attribute is strongly correlated with another attribute.
- **Assign a special value N/A:** Sometimes a value is missing due to valid reasons such as the value is out of scope discrete attribute value is not defined, or it is not possible to obtain or measure the value, which can be an indicator as well. For example, a person never rates a movie, so his rating on this movie is nonexistent.
- **Take the average attribute value:** In case we have a limited number of instances, we might not be able to afford removing instances or attributes. In that case, we can estimate the missing values, for example, by assigning the average attribute value or the average value over similar instances.
- **Predict the value from other attributes:** Predict the value from the previous entries if the attribute possesses time dependencies.

As we have seen, the value can be missing for many reasons, and hence, it is important to understand why the value is missing, absent, or corrupted.

Remove outliers

Outliers in data are values that are unlike any other values in the series and affect all learning methods to various degrees. These can be extreme values, which could be detected with confidence intervals and removed by threshold. The best approach is to visualize the data and inspect the visualization to detect irregularities. An example is shown in the following diagram. Visualization applies to low-dimensional data only:



Data transformation

Data transformation techniques tame the dataset to a format that a machine learning algorithm expects as an input, and may even help the algorithm to learn faster and achieve better performance. Standardization, for instance, assumes that data follows Gaussian distribution and transforms the values in such a way that the mean value is zero and the deviation is 1, as follows:

$$X = \frac{X - \text{mean}(X)}{\text{st.dev}}$$

Normalization, on the other hand, scales the values of attributes to a small, specified range, usually between 0 and 1:

$$X = \frac{X - \text{min}}{\text{max} - \text{min}}$$

Many machine learning toolboxes automatically normalize and standardize the data for you.

The last transformation technique is discretization, which divides the range of a continuous attribute into intervals. Why should we care? Some algorithms, such as decision trees and naïve Bayes prefer discrete attributes. The most common ways to select the intervals are shown in the following:

- **Equal width:** The interval of continuous variable is divided into k equal-width intervals
- **Equal frequency:** Suppose there are N instances, each of the k intervals contains approximately N/k instances
- **Min entropy:** The approach recursively splits the intervals until the entropy, which measures disorder, decreases more than the entropy increase, introduced by the interval split (Fayyad and Irani, 1993)

The first two methods require us to specify the number of intervals, while the last method sets the number of intervals automatically; however, it requires the class

variable, which means, it won't work for unsupervised machine learning tasks.

Data reduction

Data reduction deals with abundant attributes and instances. The number of attributes corresponds to the number of dimensions in our dataset. Dimensions with low prediction power do not only contribute very little to the overall model, but also cause a lot of harm. For instance, an attribute with random values can introduce some random patterns that will be picked up by a machine learning algorithm.

To deal with this problem, the first set of techniques removes such attributes, or in other words, selects the most promising ones. This process is known as feature selection or attribute selection and includes methods such as ReliefF, information gain, and Gini index. These methods are mainly focused on discrete attributes.

Another set of tools, focused on continuous attributes, transforms the dataset from the original dimensions into a lower-dimensional space. For example, if we have a set of points in three-dimensional space, we can make a projection into a two-dimensional space. Some information is lost, but in case the third dimension is irrelevant, we don't lose much as the data structure and relationships are almost perfectly preserved. This can be performed by the following methods:

- **Singular value decomposition (SVD)**
- **Principal Component Analysis (PCA)**
- Neural nets auto encoders

The second problem in data reduction is related to too many instances; for example, they can be duplicates or coming from a very frequent data stream. The main idea is to select a subset of instances in such a way that distribution of the selected data still resembles the original data distribution, and more importantly, the observed process. Techniques to reduce the number of instances involve random data sampling, stratification, and others. Once the data is prepared, we can start with the data analysis and modeling.

Unsupervised learning

Unsupervised learning is about analyzing the data and discovering hidden structures in unlabeled data. As no notion of the right labels is given, there is also no error measure to evaluate a learned model; however, unsupervised learning is an extremely powerful tool. Have you ever wondered how Amazon can predict what books you'll like? How Netflix knows what you want to watch before you do? The answer can be found in unsupervised learning. The following is one such example.

Find similar items

Many problems can be formulated as finding similar sets of elements, for example, customers who purchased similar products, web pages with similar content, images with similar objects, users who visited similar websites, and so on.

Two items are considered similar if they are a *small distance* apart. The main questions are how each item is represented and how is the distance between the items defined. There are two main classes of distance measures: Euclidean distances and non-Euclidean distances.

Euclidean distances

In the Euclidean space, with the n dimension, the distance between two elements is based on the locations of the elements in such a space, which is expressed as **p-norm distance**. Two commonly used distance measures are L_2 and L_1 norm distances.

L_2 norm, also known as Euclidean distance, is the most frequently applied distance measure that measures how far apart two items in a two-dimensional space are. It is calculated as a square root of the sum of the squares of the differences between elements a and b in each dimension, as follows:

$$L_2 \text{norm } d(a,b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

L_1 norm, also known as Manhattan distance, city block distance, and taxicab norm, simply sums the absolute differences in each dimension, as follows:

$$L_1 \text{norm } d(a,b) = \sum_{i=1}^n |a_i - b_i|$$

Non-Euclidean distances

A non-Euclidean distance is based on the properties of the elements, but not on their location in space. Some well-known distances are Jaccard distance, cosine distance, edit distance, and Hamming distance.

Jaccard distance is used to compute the distance between two sets. First, we compute the Jaccard similarity of two sets as the size of their intersection divided by the size of their union, as follows:

$$sim(A, B) = \frac{A \cap B}{A \cup B}$$

The Jaccard distance is then defined as 1 minus Jaccard similarity, as shown in the following:

$$d(A, B) = 1 - sim(A, B) = 1 - \frac{A \cap B}{A \cup B}$$

Cosine distance between two vectors focuses on the orientation and not magnitude, therefore, two vectors with the same orientation have cosine similarity 1 , while two perpendicular vectors have cosine similarity 0 . Suppose we have two multidimensional points, think of a point as a vector from origin $(0, 0, \dots, 0)$ to its location. Two vectors make an angle, whose cosine distance is a normalized dot-product of the vectors, as follows:

$$d(A, B) = \arccos \frac{A \cdot B}{\|A\| \|B\|}$$

Cosine distance is commonly used in a high-dimensional feature space; for instance, in text mining, where a text document represents an instance, features that correspond to different words, and their values corresponds to the number of times the word appears in the document. By computing cosine similarity, we can measure how likely two documents match in describing similar content.

Edit distance makes sense when we compare two strings. The distance between the $a=a_1a_2a_3\dots a_n$ and $b=b_1b_2b_3\dots b_n$ strings is the smallest number of the insert/delete operation of single characters required to convert the string from a to b . For example,

$a = abcd$ and $b = abbd$. To convert a to b , we have to delete the second b and insert c in its place. No smallest number of operations would convert a to b , thus the distance is $d(a, b)=2$.

Hamming distance compares two vectors of the same size and counts the number of dimensions in which they differ. In other words, it measures the number of substitutions required to convert one vector into another.

There are many distance measures focusing on various properties, for instance, correlation measures the linear relationship between two elements: **Mahalanobis distance** that measures the distance between a point and distribution of other points and **SimRank**, which is based on graph theory, measures similarity of the structure in which elements occur, and so on. As you can already imagine selecting and designing the right similarity measure for your problem is more than half of the battle. An impressive overview and evaluation of similarity measures is collected in [Chapter 2, Similarity and Dissimilarity Measures](#) in the book *Image Registration: Principles, Tools and Methods* by A. A. Goshtasby (2012).

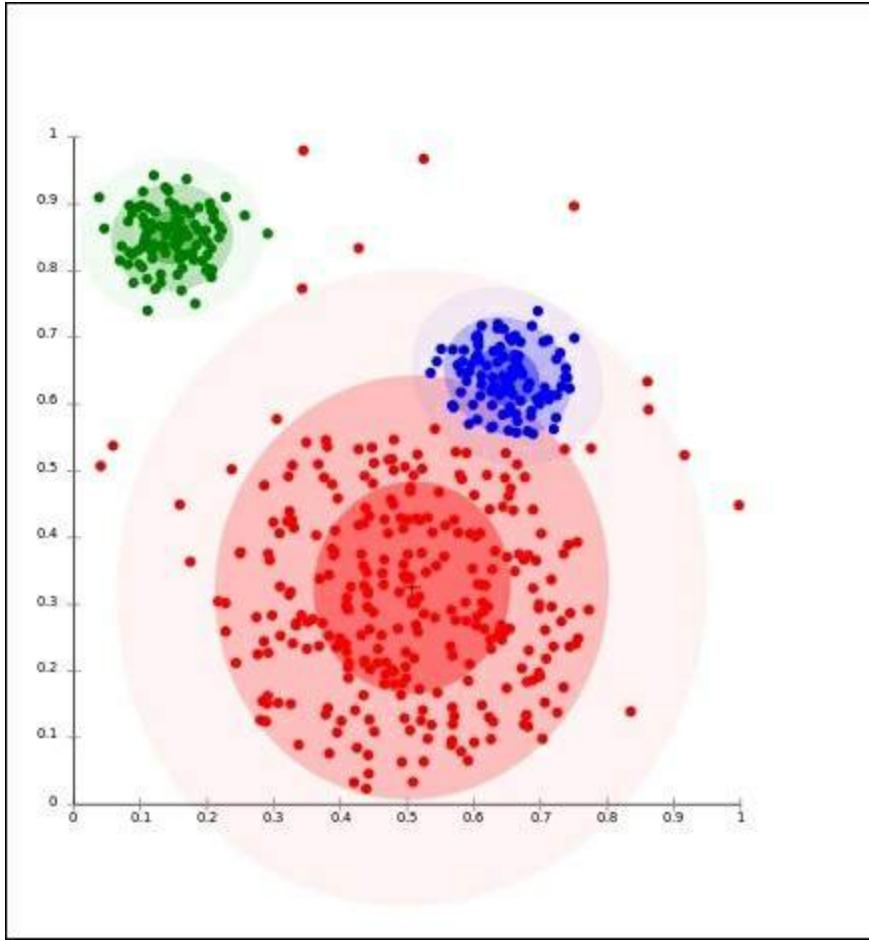
The curse of dimensionality

The curse of dimensionality refers to a situation where we have a large number of features, often hundreds or thousands, which lead to an extremely large space with sparse data and, consequently, to distance anomalies. For instance, in high dimensions, almost all pairs of points are equally distant from each other; in fact, almost all the pairs have distance close to the average distance. Another manifestation of the curse is that any two vectors are almost orthogonal, which means all the angles are close to 90 degrees. This practically makes any distance measure useless.

A cure for the curse of dimensionality might be found in one of the data reduction techniques, where we want to reduce the number of features; for instance, we can run a feature selection algorithm such as ReliefF or feature extraction/reduction algorithm such as PCA.

Clustering

Clustering is a technique for grouping similar instances into clusters according to some distance measure. The main idea is to put instances that are similar (that is, close to each other) into the same cluster, while keeping the dissimilar points (that is, the ones further apart from each other) in different clusters. An example of how clusters might look is shown in the following diagram:



The clustering algorithms follow two fundamentally different approaches. The first is a **hierarchical** or **agglomerative** approach that first considers each point as its own cluster, and then iteratively merges the most similar clusters together. It stops when further merging reaches a predefined number of clusters or if the clusters to be merged are spread over a large region.

The other approach is based on point assignment. First, initial cluster centers (that is,

centroids) are estimated—for instance, randomly—and then, each point is assigned to the closest cluster, until all the points are assigned. The most well-known algorithm in this group is **k-means clustering**.

The k-means clustering picks initial cluster centers either as points that are as far as possible from one another or (hierarchically) clusters a sample of data and picks a point that is the closest to the center of each of the k clusters.

Supervised learning

Supervised learning is the key concept behind amazing things such as voice recognition, e-mail spam filtering, face recognition in photos, and detecting credit card frauds. More formally, given a set D of learning examples described with features, X , the goal of supervised learning is to find a function that predicts a target variable, Y . The function f that describes the relation between features X and class Y is called a model:

$$f(X) \rightarrow Y$$

The general structure of supervised learning algorithms is defined by the following decisions (Hand et al., 2001):

1. Define the task.
2. Decide on the machine learning algorithm, which introduces specific inductive bias, that is, apriori assumptions that it makes regarding the target concept.
3. Decide on the **score** or **cost** function, for instance, information gain, root mean square error, and so on.
4. Decide on the optimization/search method to optimize the score function.
5. Find a function that describes the relation between X and Y .

Many decisions are already made for us by the type of the task and dataset that we have. In the following sections, we will take a closer look at the classification and regression methods and the corresponding score functions.

Classification

Classification can be applied when we deal with a discrete class, and the goal is to predict one of the mutually-exclusive values in the target variable. An example would be credit scoring, where the final prediction is whether the person is credit liable or not. The most popular algorithms include decision tree, naïve Bayes classifier, support vector machines, neural networks, and ensemble methods.

Decision tree learning

Decision tree learning builds a classification tree, where each node corresponds to one of the attributes, edges correspond to a possible value (or intervals) of the attribute from which the node originates, and each leaf corresponds to a class label. A decision tree can be used to visually and explicitly represent the prediction model, which makes it a very transparent (white box) classifier. Notable algorithms are ID3 and C4.5, although many alternative implementations and improvements (for example, J48 in Weka) exist.

Probabilistic classifiers

Given a set of attribute values, a probabilistic classifier is able to predict a distribution over a set of classes, rather than an exact class. This can be used as a degree of certainty, that is, how sure the classifier is in its prediction. The most basic classifier is naïve Bayes, which happens to be the optimal classifier if, and only if, the attributes are conditionally independent. Unfortunately, this is extremely rare in practice.

There is really an enormous subfield denoted as probabilistic graphical models, comprising of hundreds of algorithms; for example, Bayesian network, dynamic Bayesian networks, hidden Markov models, and conditional random fields that can handle not only specific relationships between attributes, but also temporal dependencies. Karkera (2014) wrote an excellent introductory book on this topic, *Building Probabilistic Graphical Models with Python*, while Koller and Friedman (2009) published a comprehensive theory bible, *Probabilistic Graphical Models*.

Kernel methods

Any linear model can be turned into a non-linear model by applying the kernel trick to the model—replacing its features (predictors) by a kernel function. In other words,

the kernel implicitly transforms our dataset into higher dimensions. The kernel trick leverages the fact that it is often easier to separate the instances in more dimensions. Algorithms capable of operating with kernels include the kernel perceptron, **Support Vector Machines (SVM)**, Gaussian processes, PCA, canonical correlation analysis, ridge regression, spectral clustering, linear adaptive filters, and many others.

Artificial neural networks

Artificial neural networks are inspired by the structure of biological neural networks and are capable of machine learning, as well as pattern recognition. They are commonly used for both regression and classification problems, comprising a wide variety of algorithms and variations for all manner of problem types. Some popular classification methods are **perceptron**, **restricted Boltzmann machine (RBM)**, and **deep belief networks**.

Ensemble learning

Ensemble methods compose of a set of diverse weaker models to obtain better predictive performance. The individual models are trained separately and their predictions are then combined in some way to make the overall prediction.

Ensembles, hence, contain multiple ways of modeling the data, which hopefully leads to better results. This is a very powerful class of techniques, and as such, it is very popular; for instance, boosting, bagging, AdaBoost, and Random Forest. The main differences among them are the type of weak learners that are to be combined and the ways in which to combine them.

Evaluating classification

Is our classifier doing well? Is this better than the other one? In classification, we count how many times we classify something right and wrong. Suppose there are two possible classification labels—yes and no—then there are four possible outcomes, as shown in the next figure:

- True positive—hit: This indicates a *yes* instance correctly predicted as *yes*
- True negative—correct rejection: This indicates a *no* instance correctly predicted as *no*
- False positive—false alarm: This indicates a *no* instance predicted as *yes*
- False negative—miss: This indicates a *yes* instance predicted as *no*

	Predicted as positive?
"	"

	Yes	No
Really positive?	TP—true positive	FN—false negative
No	FP—false positive	TN—true negative

The basic two performance measures of a classifier are classification error and accuracy, as shown in the following image:

$$\text{Classification error} = \frac{\text{errors}}{\text{totals}} = \frac{FP + FN}{FP + FN + TP + TN}$$

$$\text{Classification accuracy} = 1 - \text{error} = \frac{\text{correct}}{\text{totals}} = \frac{TP + TN}{FP + FN + TP + TN}$$

The main problem with these two measures is that they cannot handle unbalanced classes. Classifying whether a credit card transaction is an abuse or not is an example of a problem with unbalanced classes, there are 99.99% normal transactions and just a tiny percentage of abuses. Classifier that says that every transaction is a normal one is 99.99% accurate, but we are mainly interested in those few classifications that occur very rarely.

Precision and recall

The solution is to use measures that don't involve TN (correct rejections). Two such measures are as follows:

- *Precision*: This is the proportion of positive examples correctly predicted as positive (TP) out of all examples predicted as positive ($TP + FP$):

$$\text{Precision} = \frac{TP}{TP + FP}$$

- *Recall*: This is the proportion of positives examples correctly predicted as positive (TP) out of all positive examples ($TP + FN$):

$$Recall = \frac{TP}{TP + FN}$$

It is common to combine the two and report the *F-measure*, which considers both precision and recall to calculate the score as a weighted average, where the score reaches its best value at 1 and worst at 0, as follows:

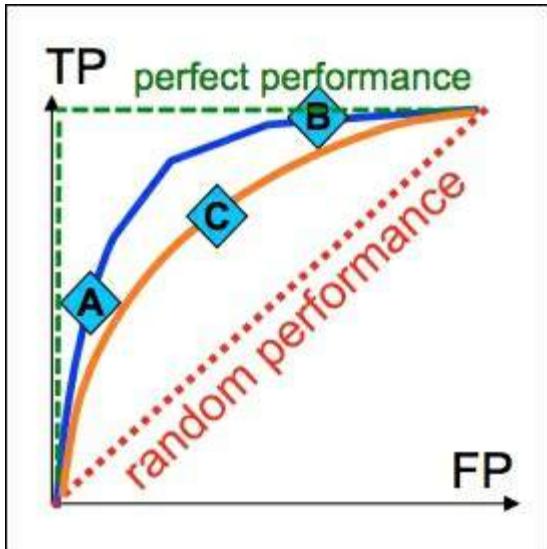
$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

Roc curves

Most classification algorithms return a classification confidence denoted as $f(X)$, which is, in turn, used to calculate the prediction. Following the credit card abuse example, a rule might look similar to the following:

$$F(X) = \begin{cases} \text{abuse, if } f(X) > \text{threshold} \\ \text{not abuse, else} \end{cases}$$

The threshold determines the error rate and the true positive rate. The outcomes for all the possible threshold values can be plotted as a **Receiver Operating Characteristics (ROC)** as shown in the following diagram:



A random predictor is plotted with a red dashed line and a perfect predictor is plotted with a green dashed line. To compare whether the **A** classifier is better than **C**, we compare the area under the curve.

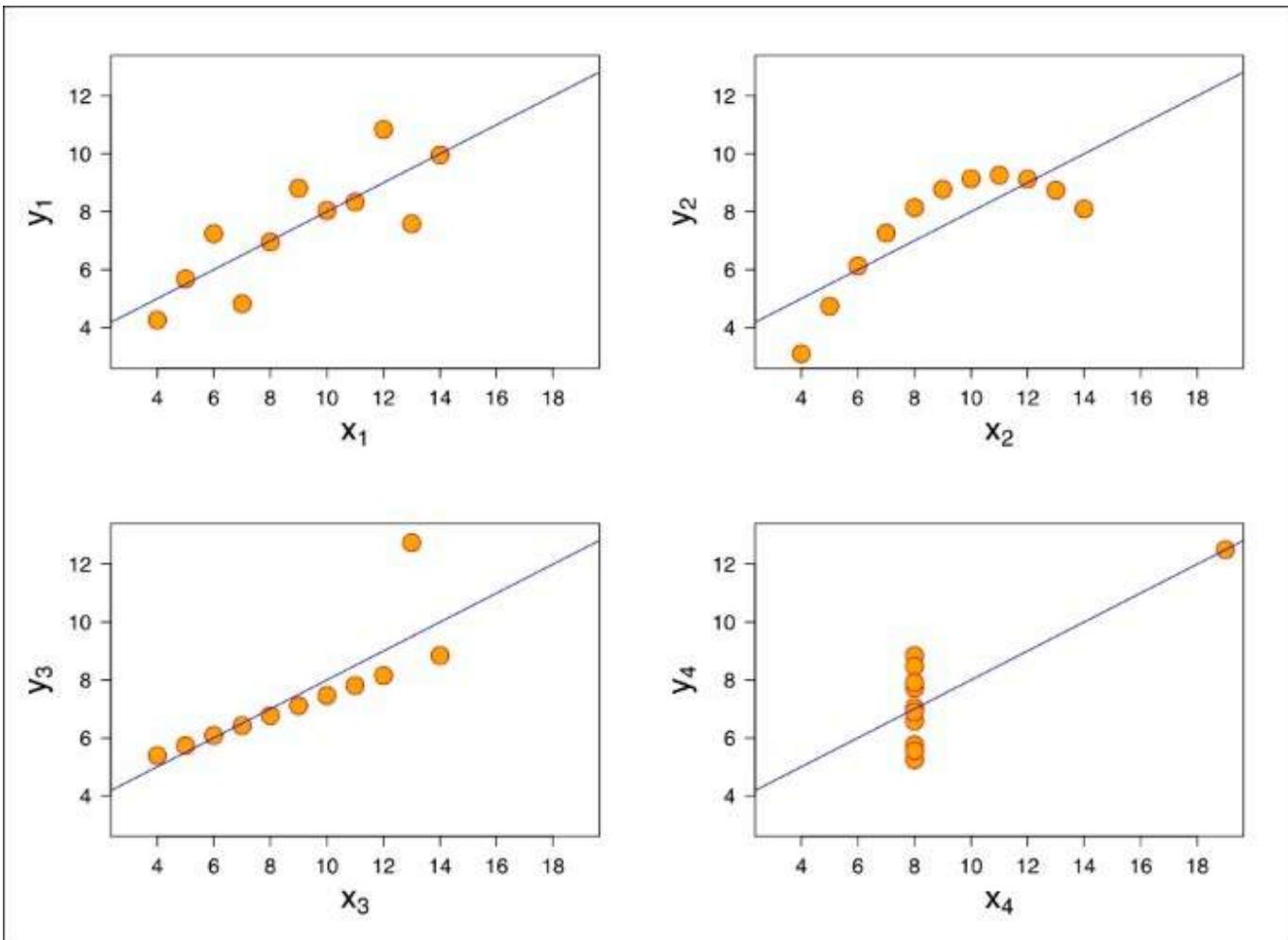
Most of the toolboxes provide all of the previous measures out-of-the-box.

Regression

Regression deals with continuous target variable, unlike classification, which works with a discrete target variable. For example, in order to forecast the outside temperature of the following few days, we will use regression; while classification will be used to predict whether it will rain or not. Generally speaking, regression is a process that estimates the relationship among features, that is, how varying a feature changes the target variable.

Linear regression

The most basic regression model assumes linear dependency between features and target variable. The model is often fitted using least squares approach, that is, the best model minimizes the squares of the errors. In many cases, linear regression is not able to model complex relations, for example, the next figure shows four different sets of points having the same linear regression line: the upper-left model captures the general trend and can be considered as a proper model, the bottom-left model fits points much better, except an outlier—this should be carefully checked—and the upper and lower-right side linear models completely miss the underlying structure of the data and cannot be considered as proper models.



Evaluating regression

In regression, we predict numbers Y from inputs X and the predictions are usually wrong and not exact. The main question that we ask is by how much? In other words, we want to measure the distance between the predicted and true values.

Mean squared error

Mean squared error is an average of the squared difference between the predicted and true values, as follows:

$$MSE(X, Y) = \sqrt{\frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2}$$

The measure is very sensitive to the outliers, for example, 99 exact predictions and *one prediction* off by 10 is scored the same as all predictions wrong by 1. Moreover, the measure is sensitive to the mean. Therefore, relative squared error, which compares the MSE of our predictor to the MSE of the mean predictor (which always predicts the mean value) is often used instead.

Mean absolute error

Mean absolute error is an average of the absolute difference between the predicted and the true values, as follows:

$$MAS(X, Y) = \frac{1}{n} \sum_{i=1}^n |f(X_i) - Y_i|$$

The MAS is less sensitive to the outliers, but it is also sensitive to the mean and scale.

Correlation coefficient

Correlation coefficient compares the average of prediction relative to the mean multiplied by training values relative to the mean. If the number is negative, it means weak correlation, positive number means strong correlation, and zero means no correlation. The correlation between true values X and predictions Y is defined as follows:

$$CC_{xy} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

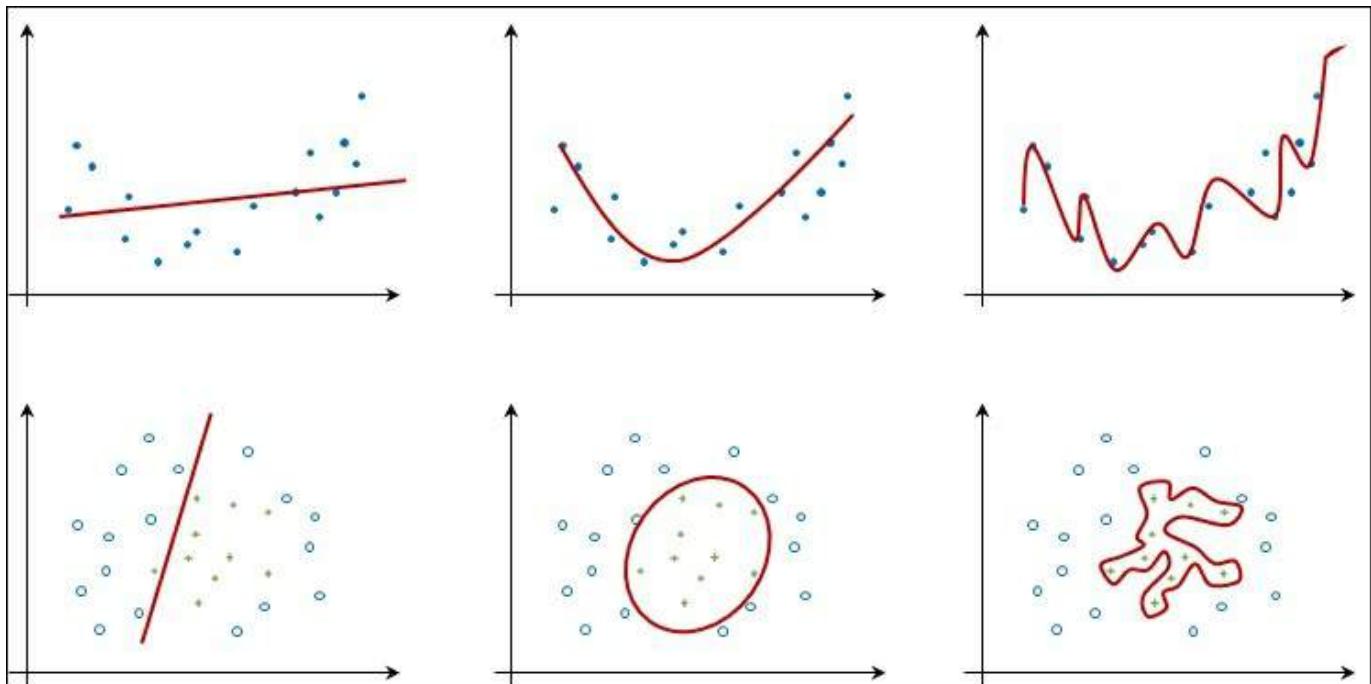
The CC measure is completely insensitive to the mean and scale, and less sensitive to the outliers. It is able to capture the relative ordering, which makes it useful to rank the tasks such as document relevance and gene expression.

Generalization and evaluation

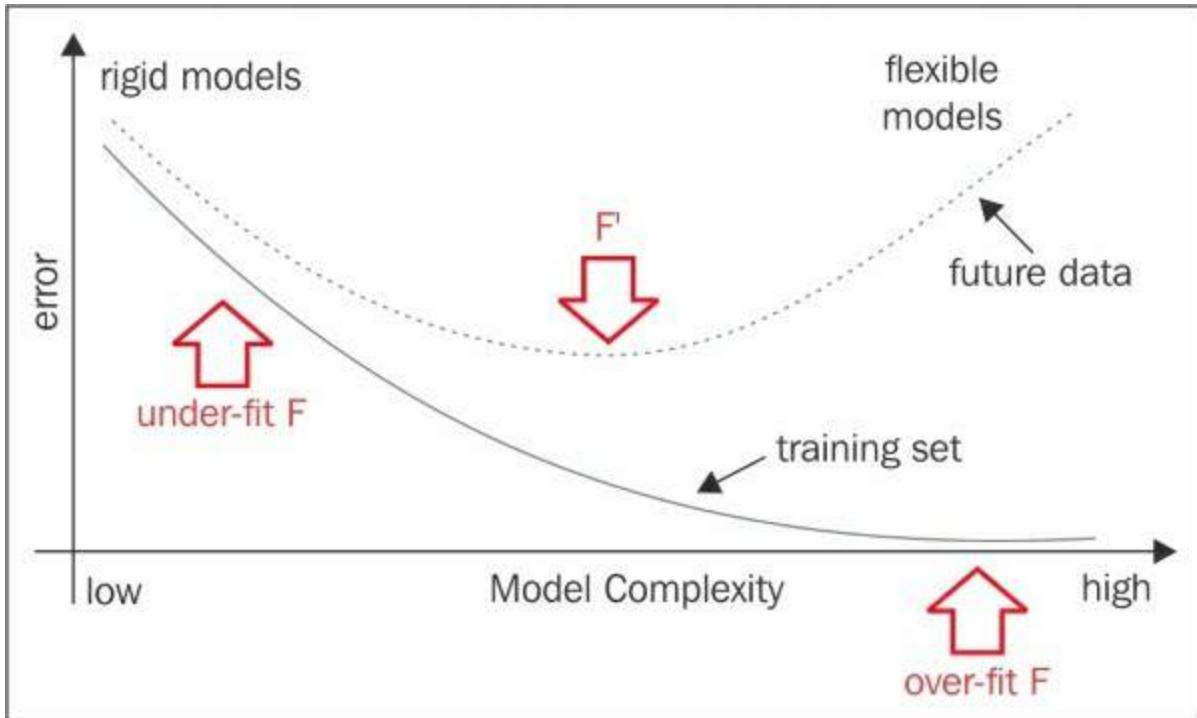
Once the model is built, how do we know it will perform on new data? Is this model any good? To answer these questions, we'll first look into the model generalization and then, see how to get an estimate of the model performance on new data.

Underfitting and overfitting

Predictor training can lead to models that are too complex or too simple. The model with low complexity (the leftmost models) can be as simple as predicting the most frequent or mean class value, while the model with high complexity (the rightmost models) can represent the training instances. Too rigid modes, which are shown on the left-hand side, cannot capture complex patterns; while too flexible models, shown on the right-hand side, fit to the noise in the training data. The main challenge is to select the appropriate learning algorithm and its parameters, so that the learned model will perform well on the new data (for example, the middle column):



The following figure shows how the error in the training set decreases with the model complexity. Simple rigid models underfit the data and have large errors. As model complexity increases, it describes the underlying structure of the training data better, and consequentially, the error decreases. If the model is too complex, it overfits the training data and its prediction error increases again:



Depending on the task complexity and data availability, we want to tune our classifiers towards less or more complex structures. Most learning algorithms allow such tuning, as follows:

- **Regression:** This is the order of the polynomial
- **Naive Bayes:** This is the number of the attributes
- **Decision trees:** This is the number of nodes in the tree, pruning confidence
- **k-nearest neighbors:** This is the number of neighbors, distance-based neighbor weights
- **SVM:** This is the kernel type, cost parameter
- **Neural network:** This is the number of neurons and hidden layers

With tuning, we want to minimize the generalization error, that is, how well the classifier performs on future data. Unfortunately, we can never compute the true generalization error; however, we can estimate it. Nevertheless, if the model performs well on the training data, but performance is much worse on the test data, the model most likely overfits.

Train and test sets

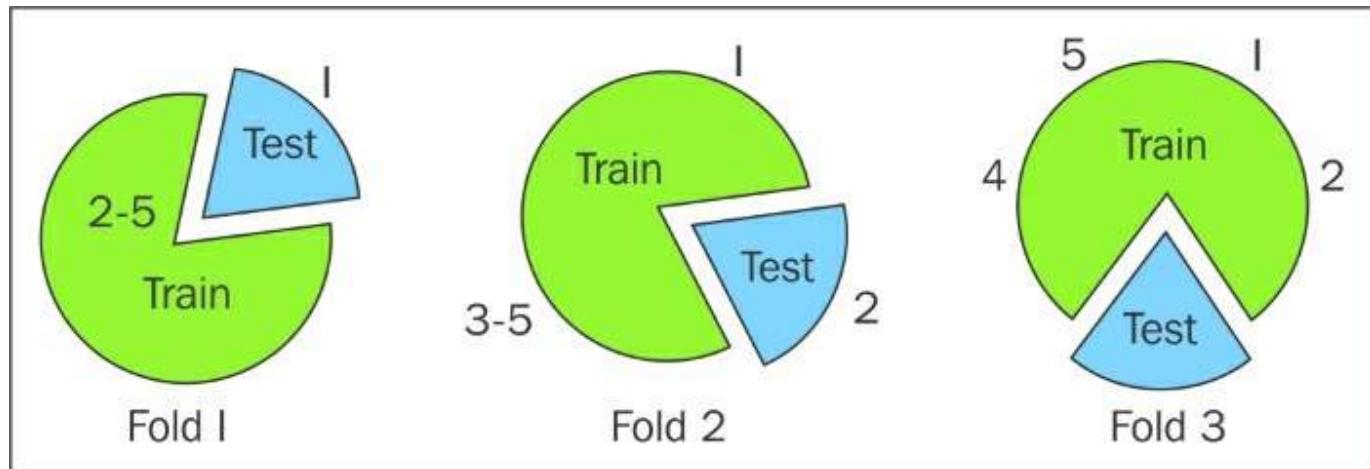
To estimate the generalization error, we split our data into two parts: training data

and testing data. A general rule of thumb is to split them in the *training:testing* ratio, that is, 70:30. We first train the predictor on the training data, then predict the values for the test data, and finally, compute the error—the difference between the predicted and the true values. This gives us an estimate of the true generalization error.

The estimation is based on the following two assumptions: first, we assume that the test set is an unbiased sample from our dataset; and second, we assume that the actual new data will reassemble the distribution as our training and testing examples. The first assumption can be mitigated by cross-validation and stratification. Also, if it is scarce one can't afford to leave out a considerable amount of data for separate test set as learning algorithms do not perform well if they don't receive enough data. In such cases, cross-validation is used instead.

Cross-validation

Cross-validation splits the dataset into k sets of approximately the same size, for example, to five sets as shown in the following figure. First, we use the 2-5 sets for learning and set 1 for training. We then repeat the procedure five times, leaving out one set at a time for testing, and average the error over the five repetitions.



This way, we used all the data for learning and testing as well, while we avoided using the same data to train and test a model.

Leave-one-out validation

An extreme example of cross-validation is the leave-one-out validation. In this case, the number of folds is equal to the number of instances; we learn on all but one

instance, and then test the model on the omitted instance. We repeat this for all instances, so that each instance is used exactly once for the validation. This approach is recommended when we have a limited set of learning examples, for example, less than 50.

Stratification

Stratification is a procedure to select a subset of instances in such a way that each fold roughly contains the same proportion of class values. When a class is continuous, the folds are selected so that the mean response value is approximately equal in all the folds. Stratification can be applied along with cross-validation or separate training and test sets.

Summary

In this chapter, we refreshed the machine learning basics. We revisited the workflow of applied machine learning and clarified the main tasks, methods, and algorithms. In the next chapter, we will review the kind of Java libraries that are available and the kind of tasks they can perform.

Chapter 2. Java Libraries and Platforms for Machine Learning

Implementing machine learning algorithms by yourself is probably the best way to learn machine learning, but you can progress much faster if you step on the shoulders of the giants and leverage one of the existing open source libraries.

This chapter reviews various libraries and platforms for machine learning in Java. The goal is to understand what each library brings to the table and what kind of problems is it able to solve?

In this chapter, we will cover the following topics:

- The requirement of Java to implement a machine learning app
- Weka, a general purpose machine learning platform
- Java machine learning library, a collection of machine learning algorithms
- Apache Mahout, a scalable machine learning platform
- Apache Spark, a distributed machine learning library
- Deeplearning4j, a deep learning library
- MALLET, a text mining library

We'll also discuss how to architect the complete machine learning app stack for both single-machine and big data apps using these libraries with other components.

The need for Java

New machine learning algorithms are often first scripted at university labs, gluing together several languages such as shell scripting, Python, R, MATLAB, Java, Scala, or C++ to prove a new concept and theoretically analyze its properties. An algorithm might then take a long path of refactoring before it lands in a library with standardized input/output and interfaces. While Python, R, and MATLAB are quite popular, they are mainly used for scripting, research, and experimenting. Java, on the other hand, is the de-facto enterprise language, which could be attributed to static typing, robust IDE support, good maintainability, as well as decent threading model, and high-performance concurrent data structure libraries. Moreover, there are already many Java libraries available for machine learning, which make it really convenient to interface them in existing Java applications and leverage powerful machine learning capabilities.

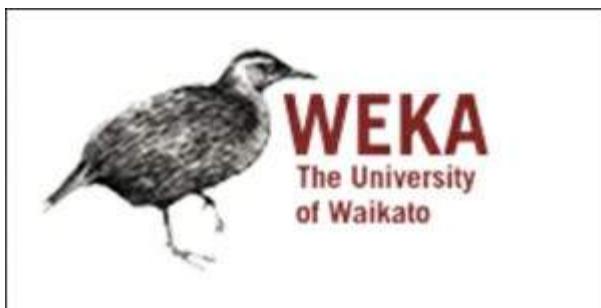
Machine learning libraries

There are over 70 Java-based open source machine learning projects listed on the MLOSS.org website and probably many more unlisted projects live at university servers, GitHub, or Bitbucket. In this section, we will review the major libraries and platforms, the kind of problems they can solve, the algorithms they support, and the kind of data they can work with.

Weka

Weka, which is short for **Waikato Environment for Knowledge Analysis**, is a machine learning library developed at the University of Waikato, New Zealand, and is probably the most well-known Java library. It is a general-purpose library that is able to solve a wide variety of machine learning tasks, such as classification, regression, and clustering. It features a rich graphical user interface, command-line interface, and Java API. You can check out Weka at <http://www.cs.waikato.ac.nz/ml/weka/>.

At the time of writing this book, Weka contains 267 algorithms in total: data pre-processing (82), attribute selection (33), classification and regression (133), clustering (12), and association rules mining (7). Graphical interfaces are well-suited for exploring your data, while Java API allows you to develop new machine learning schemes and use the algorithms in your applications.



Weka is distributed under **GNU General Public License (GNU GPL)**, which means that you can copy, distribute, and modify it as long as you track changes in source files and keep it under GNU GPL. You can even distribute it commercially, but you must disclose the source code or obtain a commercial license.

In addition to several supported file formats, Weka features its own default data format, ARFF, to describe data by attribute-data pairs. It consists of two parts. The first part contains header, which specifies all the attributes (that is, features) and their type; for instance, nominal, numeric, date, and string. The second part contains data, where each line corresponds to an instance. The last attribute in the header is implicitly considered as the target variable, missing data are marked with a question mark. For example, returning to the example from [Chapter 1, Applied Machine Learning Quick Start](#), the Bob instance written in an ARFF file format would be as

follows:

```
@RELATION person_dataset

@ATTRIBUTE `Name` STRING
@ATTRIBUTE `Height` NUMERIC
@ATTRIBUTE `Eye color`{blue, brown, green}
@ATTRIBUTE `Hobbies` STRING

@DATA
'Bob', 185.0, blue, 'climbing, sky diving'
'Anna', 163.0, brown, 'reading'
'Jane', 168.0, ?, ?
```

The file consists of three sections. The first section starts with the @RELATION <string> keyword, specifying the dataset name. The next section starts with the @ATTRIBUTE keyword, followed by the attribute name and type. The available types are STRING, NUMERIC, DATE, and a set of categorical values. The last attribute is implicitly assumed to be the target variable that we want to predict. The last section starts with the @DATA keyword, followed by one instance per line. Instance values are separated by comma and must follow the same order as attributes in the second section.

More Weka examples will be demonstrated in [Chapter 3, Basic Algorithms – Classification, Regression, and Clustering](#), and [Chapter 4, Customer Relationship Prediction with Ensembles](#).

Note

To learn more about Weka, pick up a quick-start book, *Weka How-to* by Kaluza (*Packt Publishing*) to start coding or look into *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations* by Witten and Frank (*Morgan Kaufmann Publishers*) for theoretical background and in-depth explanations.

Weka's Java API is organized in the following top-level packages:

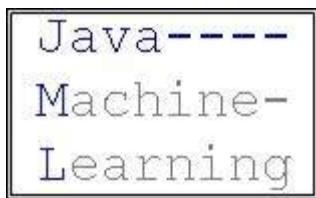
- `weka.associations`: These are data structures and algorithms for association rules learning, including **Apriori**, **predictive apriori**, `FilteredAssociator`, **FP-Growth**, **Generalized Sequential Patterns (GSP)**, **Hotspot**, and **Tertius**.
- `weka.classifiers`: These are supervised learning algorithms, evaluators, and data structures. The package is further split into the following components:

- weka.classifiers.bayes: This implements Bayesian methods, including naive Bayes, Bayes net, Bayesian logistic regression, and so on
 - weka.classifiers.evaluation: These are supervised evaluation algorithms for nominal and numerical prediction, such as evaluation statistics, confusion matrix, ROC curve, and so on
 - weka.classifiers.functions: These are regression algorithms, including linear regression, isotonic regression, Gaussian processes, support vector machine, multilayer perceptron, voted perceptron, and others
 - weka.classifiers.lazy: These are instance-based algorithms such as k-nearest neighbors, K*, and lazy Bayesian rules
 - weka.classifiers.meta: These are supervised learning meta-algorithms, including AdaBoost, bagging, additive regression, random committee, and so on
 - weka.classifiers.mi: These are multiple-instance learning algorithms, such as citation k-nn, diverse density, MI AdaBoost, and others
 - weka.classifiers.rules: These are decision tables and decision rules based on the separate-and-conquer approach, Ripper, Part, Prism, and so on
 - weka.classifiers.trees: These are various decision trees algorithms, including ID3, C4.5, M5, functional tree, logistic tree, random forest, and so on
- weka.clusterers: These are clustering algorithms, including k-means, Clope, Cobweb, DBSCAN hierarchical clustering, and farthest.
- weka.core: These are various utility classes, data presentations, configuration files, and so on.
- weka.datagenerators: These are data generators for classification, regression, and clustering algorithms.
- weka.estimators: These are various data distribution estimators for discrete/nominal domains, conditional probability estimations, and so on.
- weka.experiment: These are a set of classes supporting necessary configuration, datasets, model setups, and statistics to run experiments.
- weka.filters: These are attribute-based and instance-based selection algorithms for both supervised and unsupervised data preprocessing.
- weka.gui: These are graphical interface implementing **explorer**, **experimenter**, and **knowledge flow** applications. Explorer allows you to investigate dataset, algorithms, as well as their parameters, and visualize dataset with scatter plots and other visualizations. Experimenter is used to design batches of experiment,

but it can only be used for classification and regression problems. Knowledge flows implements a visual drag-and-drop user interface to build data flows, for example, load data, apply filter, build classifier, and evaluate.

Java machine learning

Java machine learning library, or Java-ML, is a collection of machine learning algorithms with a common interface for algorithms of the same type. It only features Java API, therefore, it is primarily aimed at software engineers and programmers. Java-ML contains algorithms for data preprocessing, feature selection, classification, and clustering. In addition, it features several Weka bridges to access Weka's algorithms directly through the Java-ML API. It can be downloaded from <http://javaml.sourceforge.net>; where, the latest release was in 2012 (at the time of writing this book).



Java-ML is also a general-purpose machine learning library. Compared to Weka, it offers more consistent interfaces and implementations of recent algorithms that are not present in other packages, such as an extensive set of state-of-the-art similarity measures and feature-selection techniques, for example, dynamic time warping, random forest attribute evaluation, and so on. Java-ML is also available under the GNU GPL license.

Java-ML supports any type of file as long as it contains one data sample per line and the features are separated by a symbol such as comma, semi-colon, and tab.

The library is organized around the following top-level packages:

- `net.sf.javaml.classification`: These are classification algorithms, including naive Bayes, random forests, bagging, self-organizing maps, k-nearest neighbors, and so on
- `net.sf.javaml.clustering`: These are clustering algorithms such as k-means, self-organizing maps, spatial clustering, Cobweb, AQBC, and others
- `net.sf.javaml.core`: These are classes representing instances and datasets
- `net.sf.javaml.distance`: These are algorithms that measure instance distance and similarity, for example, **Chebyshev distance**, cosine distance/similarity, Euclidian distance, Jaccard distance/similarity, **Mahalanobis distance**,

Manhattan distance, Minkowski distance, Pearson correlation coefficient, Spearman's footrule distance, dynamic time wrapping (DTW), and so on

- `net.sf.javaml.featureselection`: These are algorithms for feature evaluation, scoring, selection, and ranking, for instance, gain ratio, ReliefF, Kullback-Liebler divergence, symmetrical uncertainty, and so on
- `net.sf.javaml.filter`: These are methods for manipulating instances by filtering, removing attributes, setting classes or attribute values, and so on
- `net.sf.javaml.matrix`: This implements in-memory or file-based array
- `net.sf.javaml.sampling`: This implements sampling algorithms to select a subset of dataset
- `net.sf.javaml.tools`: These are utility methods on dataset, instance manipulation, serialization, Weka API interface, and so on
- `net.sf.javaml.utils`: These are utility methods for algorithms, for example, statistics, math methods, contingency tables, and others

Apache Mahout

The Apache Mahout project aims to build a scalable machine learning library. It is built atop scalable, distributed architectures, such as Hadoop, using the MapReduce paradigm, which is an approach for processing and generating large datasets with a parallel, distributed algorithm using a cluster of servers.



Mahout features console interface and Java API to scalable algorithms for clustering, classification, and collaborative filtering. It is able to solve three business problems: item recommendation, for example, recommending items such as *people who liked this movie also liked...*; clustering, for example, of text documents into groups of topically-related documents; and classification, for example, learning which topic to assign to an unlabeled document.

Mahout is distributed under a commercially-friendly Apache License, which means that you can use it as long as you keep the Apache license included and display it in your program's copyright notice.

Mahout features the following libraries:

- `org.apache.mahout.cf.taste`: These are collaborative filtering algorithms based on user-based and item-based collaborative filtering and matrix factorization with ALS
- `org.apache.mahout.classifier`: These are in-memory and distributed implementations, including logistic regression, naive Bayes, random forest, **hidden Markov models (HMM)**, and multilayer perceptron
- `org.apache.mahout.clustering`: These are clustering algorithms such as canopy clustering, k-means, fuzzy k-means, streaming k-means, and spectral clustering
- `org.apache.mahout.common`: These are utility methods for algorithms, including distances, MapReduce operations, iterators, and so on

- `org.apache.mahout.driver`: This implements a general-purpose driver to run main methods of other classes
- `org.apache.mahout.ep`: This is the evolutionary optimization using the recorded-step mutation
- `org.apache.mahout.math`: These are various math utility methods and implementations in Hadoop
- `org.apache.mahout.vectorizer`: These are classes for data presentation, manipulation, and MapReduce jobs

Apache Spark

Apache Spark, or simply Spark, is a platform for large-scale data processing built atop Hadoop, but, in contrast to Mahout, it is not tied to the MapReduce paradigm. Instead, it uses in-memory caches to extract a working set of data, process it, and repeat the query. This is reported to be up to ten times as fast as a Mahout implementation that works directly with disk-stored data. It can be grabbed from <https://spark.apache.org>.



There are many modules built atop Spark, for instance, **GraphX** for graph processing, **Spark Streaming** for processing real-time data streams, and MLlib for machine learning library featuring classification, regression, collaborative filtering, clustering, dimensionality reduction, and optimization.

Spark's MLlib can use a Hadoop-based data source, for example, **Hadoop Distributed File System (HDFS)** or HBase, as well as local files. The supported data types include the following:

- **Local vector** is stored on a single machine. Dense vectors are presented as an array of double-typed values, for example, $(2.0, 0.0, 1.0, 0.0)$; while sparse vector is presented by the size of the vector, an array of indices, and an array of values, for example, $[4, (0, 2), (2.0, 1.0)]$.
- **Labeled point** is used for supervised learning algorithms and consists of a local vector labeled with a double-typed class values. Label can be class index, binary outcome, or a list of multiple class indices (multiclass classification). For example, a labeled dense vector is presented as $[1.0, (2.0, 0.0, 1.0, 0.0)]$.
- **Local matrix** stores a dense matrix on a single machine. It is defined by matrix dimensions and a single double-array arranged in a column-major order.
- **Distributed matrix** operates on data stored in Spark's **Resilient Distributed Dataset (RDD)**, which represents a collection of elements that can be operated on in parallel. There are three presentations: row matrix, where each row is a local vector that can be stored on a single machine, row indices are

meaningless; and indexed row matrix, which is similar to row matrix, but the row indices are meaningful, that is, rows can be identified and joins can be executed; and coordinate matrix, which is used when a row cannot be stored on a single machine and the matrix is very sparse.

Spark's MLlib API library provides interfaces to various learning algorithms and utilities as outlined in the following list:

- `org.apache.spark.mllib.classification`: These are binary and multiclass classification algorithms, including linear SVMs, logistic regression, decision trees, and naive Bayes
- `org.apache.spark.mllib.clustering`: These are k-means clustering
- `org.apache.spark.mllib.linalg`: These are data presentations, including dense vectors, sparse vectors, and matrices
- `org.apache.spark.mllib.optimization`: These are the various optimization algorithms used as low-level primitives in MLlib, including gradient descent, stochastic gradient descent, update schemes for distributed SGD, and limited-memory BFGS
- `org.apache.spark.mllib.recommendation`: These are model-based collaborative filtering implemented with alternating least squares matrix factorization
- `org.apache.spark.mllib.regression`: These are regression learning algorithms, such as linear least squares, decision trees, Lasso, and Ridge regression
- `org.apache.spark.mllib.stat`: These are statistical functions for samples in sparse or dense vector format to compute the mean, variance, minimum, maximum, counts, and nonzero counts
- `org.apache.spark.mllib.tree`: This implements classification and regression decision tree-learning algorithms
- `org.apache.spark.mllib.util`: These are a collection of methods to load, save, preprocess, generate, and validate the data

Deeplearning4j

Deeplearning4j, or DL4J, is a deep-learning library written in Java. It features a distributed as well as a single-machine deep-learning framework that includes and supports various neural network structures such as feedforward neural networks, **RBM**, convolutional neural nets, deep belief networks, autoencoders, and others. DL4J can solve distinct problems, such as identifying faces, voices, spam or e-commerce fraud.

Deeplearning4j is also distributed under Apache 2.0 license and can be downloaded from <http://deeplearning4j.org>. The library is organized as follows:

- org.deeplearning4j.base: These are loading classes
- org.deeplearning4j.berkeley: These are math utility methods
- org.deeplearning4j.clustering: This is the implementation of k-means clustering
- org.deeplearning4j.datasets: This is dataset manipulation, including import, creation, iterating, and so on
- org.deeplearning4j.distributions: These are utility methods for distributions
- org.deeplearning4j.eval: These are evaluation classes, including the confusion matrix
- org.deeplearning4j.exceptions: This implements exception handlers
- org.deeplearning4j.models: These are supervised learning algorithms, including deep belief network, stacked autoencoder, stacked denoising autoencoder, and RBM
- org.deeplearning4j.nn: These are the implementation of components and algorithms based on neural networks, such as neural network, multi-layer network, convolutional multi-layer network, and so on
- org.deeplearning4j.optimize: These are neural net optimization algorithms, including back propagation, multi-layer optimization, output layer optimization, and so on
- org.deeplearning4j.plot: These are various methods for rendering data
- org.deeplearning4j.rng: This is a random data generator
- org.deeplearning4j.util: These are helper and utility methods

MALLET

Machine Learning for Language Toolkit (MALLET), is a large library of natural language processing algorithms and utilities. It can be used in a variety of tasks such as document classification, document clustering, information extraction, and topic modeling. It features command-line interface as well as Java API for several algorithms such as naive Bayes, HMM, **Latent Dirichlet** topic models, logistic regression, and conditional random fields.



MALLET is available under Common Public License 1.0, which means that you can even use it in commercial applications. It can be downloaded from <http://mallet.cs.umass.edu>. Mallet instance is represented by name, label, data, and source. However, there are two methods to import data into the Mallet format, as shown in the following list:

- Instance per file: Each file, that is, document, corresponds to an instance and Mallet accepts the directory name for the input.
- Instance per line: Each line corresponds to an instance, where the following format is assumed: the `instance_name` label token. Data will be a feature vector, consisting of distinct words that appear as tokens and their occurrence count.

The library comprises the following packages:

- `cc.mallet.classify`: These are algorithms for training and classifying instances, including AdaBoost, bagging, C4.5, as well as other decision tree models, multivariate logistic regression, naive Bayes, and Winnow2.
- `cc.mallet.cluster`: These are unsupervised clustering algorithms, including greedy agglomerative, hill climbing, k-best, and k-means clustering.
- `cc.mallet.extract`: This implements tokenizers, document extractors, document viewers, cleaners, and so on.
- `cc.mallet.fst`: This implements sequence models, including conditional

random fields, HMM, maximum entropy Markov models, and corresponding algorithms and evaluators.

- `cc.mallet.grmm`: This implements graphical models and factor graphs such as inference algorithms, learning, and testing. For example, loopy belief propagation, Gibbs sampling, and so on.
- `cc.mallet.optimize`: These are optimization algorithms for finding the maximum of a function, such as gradient ascent, limited-memory BFGS, stochastic meta ascent, and so on.
- `cc.mallet.pipe`: These are methods as pipelines to process data into MALLET instances.
- `cc.mallet.topics`: These are topics modeling algorithms, such as Latent Dirichlet allocation, four-level pachinko allocation, hierarchical PAM, DMRT, and so on.
- `cc.mallet.types`: This implements fundamental data types such as dataset, feature vector, instance, and label.
- `cc.mallet.util`: These are miscellaneous utility functions such as command-line processing, search, math, test, and so on.

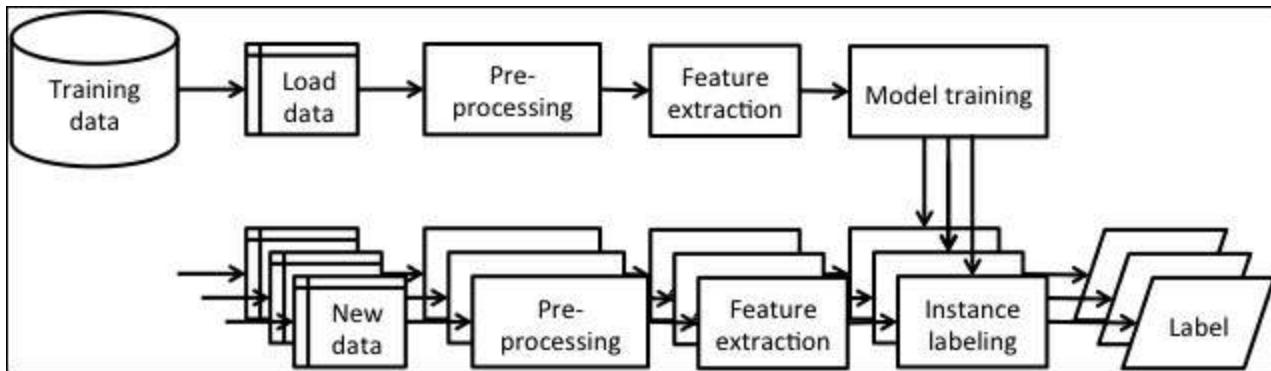
Comparing libraries

The following table summarizes all the presented libraries. The table is, by no means, exhaustive—there are many more libraries covering the specific-problem domains. This review should serve as an overview of the big names in the Java machine learning world:

	Problem domains	License	Architecture	Algorithms
Weka	General purpose	GNU GPL	Single machine	Decision trees, naive Bayes, neural network, random forest, AdaBoost, hierarchical clustering, and so on
Java-ML	General purpose	GNU GPL	Single machine	k-means clustering, self-organizing maps, Markov chain clustering, Cobweb, random forest, decision trees, bagging, distance measures, and so on
Mahout	Classification, recommendation, and clustering	Apache 2.0 License	Distributed, single machine	Logistic regression, naive Bayes, random forest, HMM, multilayer perceptron, k-means clustering, and so on
Spark	General purpose	Apache 2.0 License	Distributed	SVM, logistic regression, decision trees, naive Bayes, k-means clustering, linear least squares, LASSO, ridge regression, and so on
DL4J	Deep learning	Apache 2.0 License	Distributed, single machine	RBM, deep belief networks, deep autoencoders, recursive neural tensor networks, convolutional neural network, and stacked denoising autoencoders
MALLET	Text mining	Common Public License 1.0	Single machine	Naive Bayes, decision trees, maximum entropy, hidden Markov models, and conditional random fields

Building a machine learning application

Machine learning applications, especially those focused on classification, usually follow the same high-level workflow as shown in the following diagram. The workflow comprises two phases: training the classifier and classification of new instances. Both phases share common steps as you can see in the following diagram:



First, we use a set of **Training data**, select a representative subset as the training set, preprocess missing data, and extract features. A selected supervised learning algorithm is used to train a model, which is deployed in the second phase. The second phase puts a new data instance through the same **Pre-processing** and **Feature extraction** procedure and applies the learned model to obtain the instance label. If you are able to collect new labeled data, periodically rerun the learning phase to retrain the model, and replace the old one with the retrained one in the classification phase.

Traditional machine learning architecture

Structured data, such as transactional, customer, analytical, and market data, usually resides within a local relational database. Given a query language, such as SQL, we can query the data used for processing, as shown in the workflow in the previous diagram. Usually, all the data can be stored in the memory and further processed with a machine learning library such as Weka, Java-ML, or Mallet.

A common practice in the architecture design is to create data pipelines, where different steps in the workflow are split. For instance, in order to create a client data record, we might have to scrap the data from different data sources. The record can be then saved in an intermediate database for further processing.

To understand how the high-level aspects of big data architecture differ, let's first clarify when is the data considered big?

Dealing with big data

Big data existed long before the phrase was invented, for instance, banks and stock exchanges have been processing billions of transactions daily for years, and airline companies have worldwide real-time infrastructure for operational management of passenger booking, and so on. So what is big data really? Doug Laney (2001) suggested that big data is defined by three *Vs*: volume, velocity, and variety. Therefore, to answer the question whether your data is big, we can translate this into the following three subquestions:

- **Volume:** Can you store your data in memory?
- **Velocity:** Can you process new incoming data with a single machine?
- **Variety:** Is your data from a single source?

If you answered all the questions with yes, then your data is probably not big, do not worry, you have just simplified your application architecture.

If your answer to all the questions was no, then your data is big! However, if you have mixed answers, then it's complicated. Some may argue that a *V* is important, other may say the other *Vs*. From the machine learning point of view, there is a fundamental difference in algorithm implementation to process the data in memory or from distributed storage. Therefore, a rule of thumb is as follows: if you cannot store your data in the memory, then you should look into a big data machine learning library.

The exact answer depends on the problem that you are trying to solve. If you're starting a new project, I'd suggest you start off with a single-machine library and prototype your algorithm, possibly with a subset of your data if the entire data does not fit into the memory. Once you've established good initial results, consider moving to something more heavy duty such as Mahout or Spark.

Big data application architecture

Big data, such as documents, weblogs, social networks, sensor data, and others, are stored in a NoSQL database, such as MongoDB, or a distributed filesystem, such as HDFS. In case we deal with structured data, we can deploy database capabilities using systems such as Cassandra or HBase built atop Hadoop. Data processing follows the MapReduce paradigm, which breaks data processing problems into smaller subproblems and distributes tasks across processing nodes. Machine learning

models are finally trained with machine learning libraries such as Mahout and Spark.

Note

MongoDB is a NoSQL database, which stores documents in a JSON-like format.

Read more about it at <https://www.mongodb.org>.

Hadoop is a framework for distributed processing of large datasets across a cluster of computers. It includes its own filesystem format HDFS, job scheduling framework YARD, and implements the MapReduce approach for parallel data processing. More about Hadoop is available at <http://hadoop.apache.org/>.

Cassandra is a distributed database management system build to provide fault-tolerant, scalable, and decentralized storage. More information is available at <http://cassandra.apache.org/>.

HBase is another database that focuses on random read/write access to distributed storage. More information is available at <https://hbase.apache.org/>.

Summary

Selecting a machine learning library has an important impact on your application architecture. The key is to consider your project requirements. What kind of data do you have? What kind of problem are you trying to solve? Is your data big? Do you need distributed storage? What kind of algorithm are you planning to use? Once you figure out what you need to solve your problem, pick a library that best fits your needs.

In the next chapter, we will cover how to complete basic machine learning tasks such as classification, regression, and clustering using some of the presented libraries.

Chapter 3. Basic Algorithms – Classification, Regression, and Clustering

In the previous chapter, we reviewed the key Java libraries for machine learning and what they bring to the table. In this chapter, we will finally get our hands dirty. We will take a closer look at the basic machine learning tasks such as classification, regression, and clustering. Each of the topics will introduce basic algorithms for classification, regression, and clustering. The example datasets will be small, simple, and easy to understand.

The following is the list of topics that will be covered in this chapter:

- Loading data
- Filtering attributes
- Building classification, regression, and clustering models
- Evaluating models

Before you start

Download the latest version of Weka 3.6 from

<http://www.cs.waikato.ac.nz/ml/weka/downloading.html>.

There are multiple download options available. We'll want to use Weka as a library in our source code, so make sure you skip the self-extracting executables and pick the ZIP archive as shown at the following image. Unzip the archive and locate `weka.jar` within the extracted archive:

- **Other platforms (Linux, etc.)**

Click **here** to download a zip archive containing Weka
(weka-3-7-11.zip; 33.2 MB)

First unzip the zip file. This will create a new directory called weka-3-7-11. To run Weka, change into that directory and type

```
java -Xmx1000M -jar weka.jar
```

Note that Java needs to be installed on your system for this to work. Also note, that using `-jar` will override your current CLASSPATH variable and only use the `weka.jar`.

We'll use the Eclipse IDE to show examples, as follows:

1. Start a new Java project.
2. Right-click on the project properties, select **Java Build Path**, click on the **Libraries** tab, and select **Add External JARs**.
3. Navigate to extract the Weka archive and select the `weka.jar` file.

That's it, we are ready to implement the basic machine learning techniques!

Classification

We will start with the most commonly used machine learning technique, that is, classification. As we reviewed in the first chapter, the main idea is to automatically build a mapping between the input variables and the outcome. In the following sections, we will look at how to load the data, select features, implement a basic classifier in Weka, and evaluate the classifier performance.

Data

For this task, we will have a look at the `zoo` database [ref]. The database contains 101 data entries of the animals described with 18 attributes as shown in the following table:

animal	aquatic	fins
hair	predator	legs
feathers	toothed	tail
eggs	backbone	domestic
milk	breathes	cat size
airborne	venomous	type

An example entry in the dataset set is a lion with the following attributes:

- animal: lion
- hair: true
- feathers: false
- eggs: false
- milk: true
- airbone: false
- aquatic: false
- predator: true
- toothed: true
- backbone: true
- breaths: true
- venomous: false
- fins: false
- legs: 4
- tail: true
- domestic: false
- catsize: true
- type: mammal

Our task will be to build a model to predict the outcome variable, `animal`, given all the other attributes as input.

Loading data

Before we start with the analysis, we will load the data in Weka's ARFF format and print the total number of loaded instances. Each data sample is held within an `Instances` object, while the complete dataset accompanied with meta-information is handled by the `Instances` object.

To load the input data, we will use the `DataSource` object that accepts a variety of file formats and converts them to `Instances`:

```
DataSource source = new DataSource(args[0]);
Instances data = source.getDataSet();
System.out.println(data.numInstances() + " instances loaded.");
// System.out.println(data.toString());
```

This outputs the number of loaded `instances`, as follows:

```
101 instances loaded.
```

We can also print the complete dataset by calling the `data.toString()` method.

Our task is to learn a model that is able to predict the `animal` attribute in the future examples for which we know the other attributes but do not know the `animal` label. Hence, we remove the `animal` attribute from the training set. We accomplish this by filtering out the `animal` attribute using the `Remove` filter.

First, we set a string table of parameters, specifying that the first attribute must be removed. The remaining attributes are used as our dataset for training a classifier:

```
Remove remove = new Remove();
String[] opts = new String[]{"-R", "1"};
```

Finally, we call the `Filter.useFilter(Instances, Filter)` static method to apply the filter on the selected dataset:

```
remove.setOptions(opts);
remove.setInputFormat(data);
data = Filter.useFilter(data, remove);
```

Feature selection

As introduced in [Chapter 1, Applied Machine Learning Quick Start](#), one of the pre-processing steps is focused on feature selection, also known as attribute selection. The goal is to select a subset of relevant attributes that will be used in a learned model. Why is feature selection important? A smaller set of attributes simplifies the models and makes them easier to interpret by users, this usually requires shorter training and reduces overfitting.

Attribute selection can take into account the class value or not. In the first case, an attribute selection algorithm evaluates the different subsets of features and calculates a score that indicates the quality of selected attributes. We can use different searching algorithms such as exhaustive search, best first search, and different quality scores such as information gain, Gini index, and so on.

Weka supports this process by an `AttributeSelection` object, which requires two additional parameters: evaluator, which computes how informative an attribute is and a ranker, which sorts the attributes according to the score assigned by the evaluator.

In this example, we will use information gain as an evaluator and rank the features by their information gain score:

```
InfoGainAttributeEval eval = new InfoGainAttributeEval();
Ranker search = new Ranker();
```

Next, we initialize an `AttributeSelection` object and set the evaluator, ranker, and data:

```
AttributeSelection attSelect = new AttributeSelection();
attSelect.setEvaluator(eval);
attSelect.setSearch(search);
attSelect.SelectAttributes(data);
```

Finally, we can print an order list of attribute indices, as follows:

```
int[] indices = attSelect.selectedAttributes();
System.out.println(Utils.arrayToString(indices));
```

The method outputs the following result:

```
12,3,7,2,0,1,8,9,13,4,11,5,15,10,6,14,16
```

The top three most informative attributes are 12 (fins), 3 (eggs), 7 (aquatic), 2 (hair), and so on. Based on this list, we can remove additional, non-informative features in order to help learning algorithms achieve more accurate and faster learning models.

What would make the final decision about the number of attributes to keep? There's no rule of thumb related to an exact number—the number of attributes depends on the data and problem. The purpose of attribute selection is choosing attributes that serve your model better, so it is better to focus whether the attributes are improving the model.

Learning algorithms

We have loaded our data, selected the best features, and are ready to learn some classification models. Let's begin with the basic decision trees.

Decision tree in Weka is implemented within the `J48` class, which is a re-implementation of Quinlan's famous C4.5 decision tree learner [Quinlan, 1993].

First, we initialize a new `J48` decision tree learner. We can pass additional parameters with a string table, for instance, tree pruning that controls the model complexity (refer to [Chapter 1](#), *Applied Machine Learning Quick Start*). In our case, we will build an un-pruned tree, hence we will pass a single `-U` parameter:

```
J48 tree = new J48();
String[] options = new String[1];
options[0] = "-U";

tree.setOptions(options);
```

Next, we call the `buildClassifier(Instances)` method to initialize the learning process:

```
tree.buildClassifier(data);
```

The built model is now stored in a `tree` object. We can output the entire `J48` unpruned tree calling the `toString()` method:

```
System.out.println(tree);
```

The output is as follows:

```
J48 unpruned tree
-----
feathers = false
|   milk = false
|   |   backbone = false
|   |   |   airborne = false
|   |   |   |   predator = false
|   |   |   |   |   legs <= 2: invertebrate (2.0)
|   |   |   |   |   |   legs > 2: insect (2.0)
|   |   |   |   |   predator = true: invertebrate (8.0)
|   |   |   |   |   |   airborne = true: insect (6.0)
|   |   |   |   backbone = true
```

```
|   |   |   fins = false
|   |   |   |   tail = false: amphibian (3.0)
|   |   |   |   tail = true: reptile (6.0/1.0)
|   |   |   fins = true: fish (13.0) |   milk = true: mammal (41.0)
feathers = true: bird (20.0)
```

Number of Leaves : .9

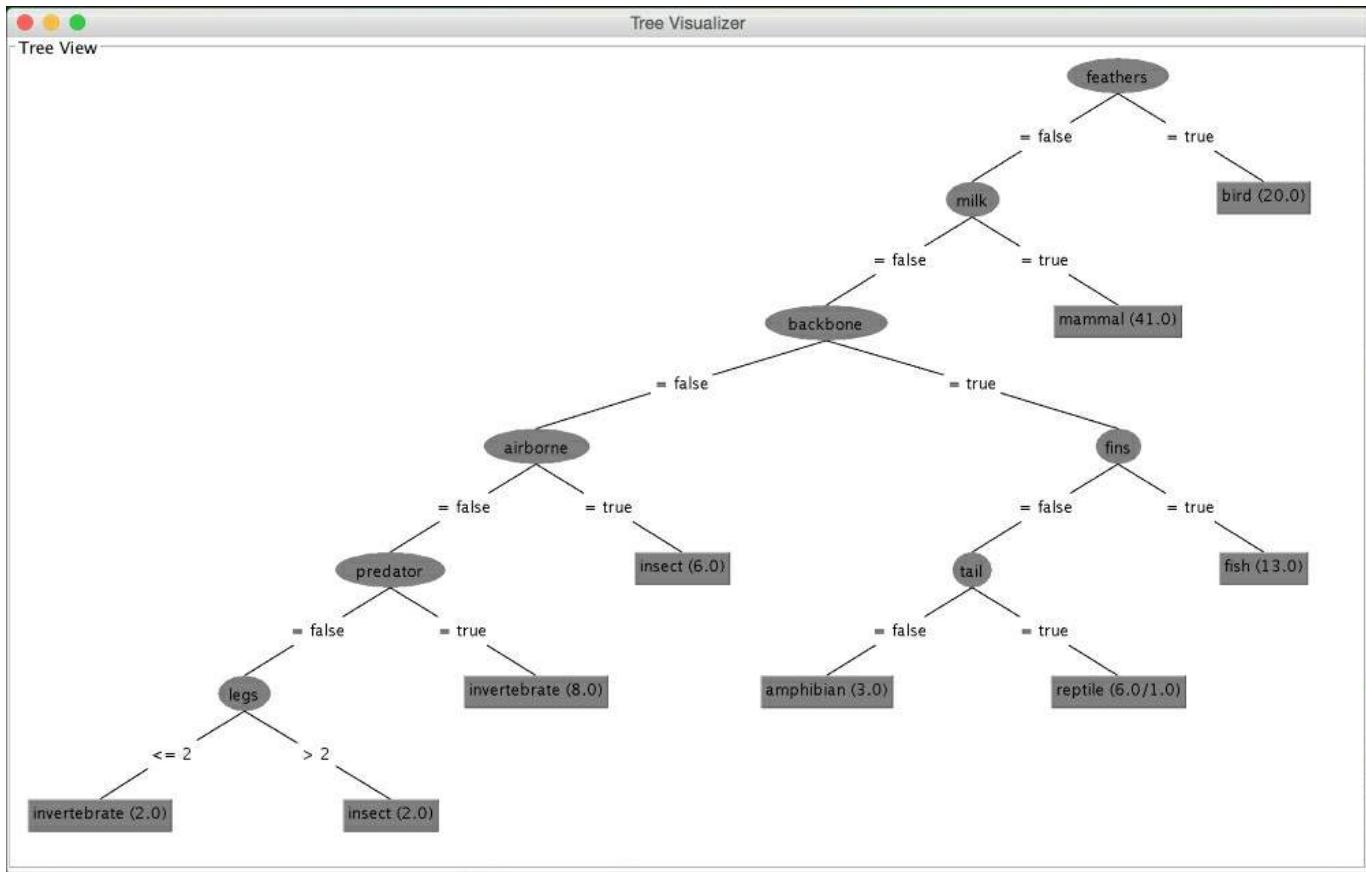
Size of the tree : ..17

The outputted tree has 17 nodes in total, 9 of these are terminal (**Leaves**).

Another way to present the tree is to leverage the built-in `TreeVisualizer` tree viewer, as follows:

```
TreeVisualizer tv = new TreeVisualizer(null, tree.graph(), new
PlaceNode2());
JFrame frame = new javax.swing.JFrame("Tree Visualizer");
frame.setSize(800, 500);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().add(tv);
frame.setVisible(true);
tv.fitToScreen();
```

The code results in the following frame:



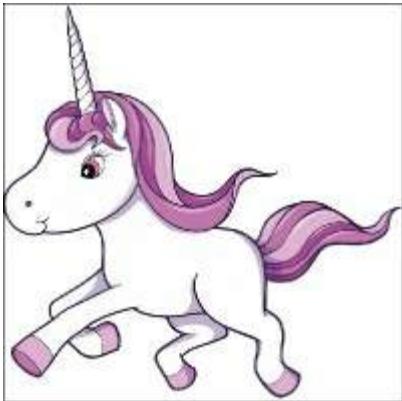
The decision process starts at the top node, also known as the root node. The node label specifies the attribute value that will be checked. In our example, we first check the value of the `feathers` attribute. If the feather is present, we follow the right-hand branch, which leads us to the leaf labeled `bird`, indicating there are 20 examples supporting this outcome. If the feather is not present, we follow the left-hand branch, which leads us to the next `milk` attribute. We check the value of the attribute again and follow the branch that matches the attribute value. We repeat the process until we reach a leaf node.

We can build other classifiers by following the same steps: initialize a classifier, pass the parameters controlling the model complexity, and call the `buildClassifier(Instances)` method.

In the next section, we will learn how to use a trained model to assign a class label to a new example whose label is unknown.

Classify new data

Suppose we record attributes for an animal whose label we do not know, we can predict its label from the learned classification model:



We first construct a feature vector describing the new specimen, as follows:

```
double[] vals = new double[data.numAttributes()];  
vals[0] = 1.0; //hair {false, true}  
vals[1] = 0.0; //feathers {false, true}  
vals[2] = 0.0; //eggs {false, true}  
vals[3] = 1.0; //milk {false, true}  
vals[4] = 0.0; //airborne {false, true}  
vals[5] = 0.0; //aquatic {false, true}  
vals[6] = 0.0; //predator {false, true}  
vals[7] = 1.0; //toothed {false, true}  
vals[8] = 1.0; //backbone {false, true}  
vals[9] = 1.0; //breathes {false, true}  
vals[10] = 1.0; //venomous {false, true}  
vals[11] = 0.0; //fins {false, true}  
vals[12] = 4.0; //legs INTEGER [0,9]  
vals[13] = 1.0; //tail {false, true}  
vals[14] = 1.0; //domestic {false, true}  
vals[15] = 0.0; //catsize {false, true}  
Instance myUnicorn = new Instance(1.0, vals);
```

Finally, we call the `classify(Instance)` method on the model to obtain the class value. The method returns label index, as follows:

```
double result = tree.classifyInstance(myUnicorn);  
System.out.println(data.classAttribute().value((int) result));
```

This outputs the `mammal` class label.

Evaluation and prediction error metrics

We built a model, but we do not know if it can be trusted. To estimate its performance, we can apply a cross-validation technique explained in [Chapter 1, Applied Machine Learning Quick Start](#).

Weka offers an `Evaluation` class implementing cross validation. We pass the model, data, number of folds, and an initial random seed, as follows:

```
Classifier cl = new J48();
Evaluation eval_roc = new Evaluation(data);
eval_roc.crossValidateModel(cl, data, 10, new Random(1), new Object[]
{});
System.out.println(eval_roc.toSummaryString());
```

The evaluation results are stored in the `Evaluation` object.

A mix of the most common metrics can be invoked by calling the `toString()` method. Note that the output does not differentiate between regression and classification, so pay attention to the metrics that make sense, as follows:

Correctly Classified Instances	93	92.0792 %
Incorrectly Classified Instances	8	7.9208 %
Kappa statistic	0.8955	
Mean absolute error	0.0225	
Root mean squared error	0.14	
Relative absolute error	10.2478 %	
Root relative squared error	42.4398 %	
Coverage of cases (0.95 level)	96.0396 %	
Mean rel. region size (0.95 level)	15.4173 %	
Total Number of Instances	101	

In the classification, we are interested in the number of correctly/incorrectly classified instances.

Confusion matrix

Furthermore, we can inspect where a particular misclassification has been made by examining the confusion matrix. Confusion matrix shows how a specific class value was predicted:

```
double[][] confusionMatrix = eval_roc.confusionMatrix();  
System.out.println(eval_roc.toMatrixString());
```

The resulting confusion matrix is as follows:

```
==== Confusion Matrix ===
```

	a	b	c	d	e	f	g	<-- classified as
41	0	0	0	0	0	0	0	a = mammal
0	20	0	0	0	0	0	0	b = bird
0	0	3	1	0	1	0	0	c = reptile
0	0	0	13	0	0	0	0	d = fish
0	0	1	0	3	0	0	0	e = amphibian
0	0	0	0	0	5	3	1	f = insect
0	0	0	0	0	2	8	1	g = invertebrate

The first column names in the first row correspond to labels assigned by the classification mode. Each additional row then corresponds to an actual true class value. For instance, the second row corresponds instances with the `mammal` true class label. In the column line, we read that all mammals were correctly classified as mammals. In the fourth row, reptiles, we notice that three were correctly classified as reptiles, while one was classified as `fish` and one as an `insect`. Confusion matrix hence, gives us an insight into the kind of errors that our classification model makes.

Choosing a classification algorithm

Naive Bayes is one of the most simple, efficient, and effective inductive algorithms in machine learning. When features are independent, which is rarely true in real world, it is theoretically optimal, and even with dependent features, its performance is amazingly competitive (Zhang, 2004). The main disadvantage is that it cannot learn how features interact with each other, for example, despite the fact that you like your tea with lemon or milk, you hate a tea having both of them at the same time.

Decision tree's main advantage is a model, that is, a tree, which is easy to interpret and explain as we studied in our example. It can handle both nominal and numeric features and you don't have to worry about whether the data is linearly separable.

Some other examples of classification algorithms are as follows:

- `weka.classifiers.rules.ZeroR`: This predicts the majority class and is considered as a baseline, that is, if your classifier's performance is worse than the average value predictor, it is not worth considering it.
- `weka.classifiers.trees.RandomTree`: This constructs a tree that considers K randomly chosen attributes at each node.
- `weka.classifiers.trees.RandomForest`: This constructs a set (that is, forest) of random trees and uses majority voting to classify a new instance.
- `weka.classifiers.lazy.IBk`: This is the k-nearest neighbor's classifier that is able to select an appropriate value of neighbors based on cross-validation.
- `weka.classifiers.functions.MultilayerPerceptron`: This is a classifier based on neural networks that use back-propagation to classify instances. The network can be built by hand, or created by an algorithm, or both.
- `weka.classifiers.bayes.NaiveBayes`: This is a naive Bayes classifier that uses estimator classes, where numeric estimator precision values are chosen based on the analysis of the training data.
- `weka.classifiers.meta.AdaBoostM1`: This is the class for boosting a nominal class classifier using the **AdaBoost M1** method. Only nominal class problems can be tackled. This often dramatically improves the performance, but sometimes it overfits.
- `weka.classifiers.meta.Bagging`: This is the class for bagging a classifier to reduce the variance. This can perform classification and regression, depending on the base learner.

Regression

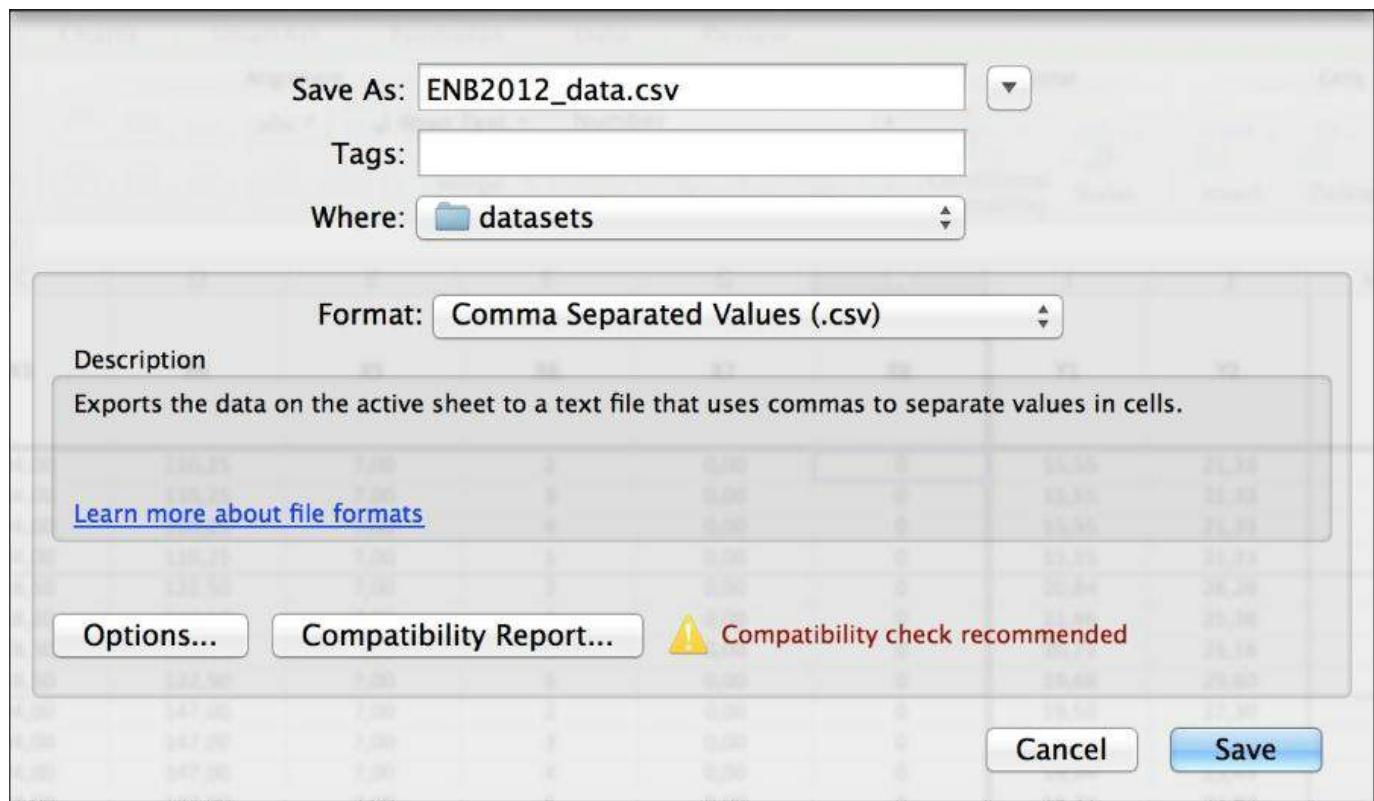
We will explore basic regression algorithms through analysis of energy efficiency dataset (Tsanas and Xifara, 2012). We will investigate the heating and cooling load requirements of the buildings based on their construction characteristics such as surface, wall and roof area, height, hazing area, and compactness. The researchers used a simulator to design 12 different house configurations while varying 18 building characteristics. In total, 768 different buildings were simulated.

Our first goal is to systematically analyze the impact each building characterizes has on the target variable, that is, heating or cooling load. The second goal is to compare the performance of a classical linear regression model against other methods, such as SVM regression, random forests, and neural networks. For this task, we will use the Weka library.

Loading the data

Download the energy efficiency dataset from
<https://archive.ics.uci.edu/ml/datasets/Energy+efficiency>.

The dataset is in Excel's XLSX format, which cannot be read by Weka. We can convert it to a **Comma Separated Value (CSV)** format by clicking **File | Save As...** and picking CSV in the saving dialog as shown in the following screenshot. Confirm to save only the active sheet (since all others are empty) and confirm to continue to lose some formatting features. Now, the file is ready to be loaded by Weka:



Open the file in a text editor and inspect if the file was indeed correctly transformed. There might be some minor issues that may be potentially causing problems. For instance, in my export, each line ended with a double semicolon, as follows:

```
x1;x2;x3;x4;x5;x6;x7;x8;Y1;Y2;;  
0,98;514,50;294,00;110,25;7,00;2;0,00;0;15,55;21,33;;  
0,98;514,50;294,00;110,25;7,00;3;0,00;0;15,55;21,33;;
```

To remove the doubled semicolon, we can use the **Find and Replace** function: find

" ; ; " and replace it with " ; ".

The second problem was that my file had a long list of empty lines at the end of the document, which can be simply deleted:

```
0,62;808,50;367,50;220,50;3,50;5;0,40;5;16,64;16,03;;  
;;;;;;;  
;;;;;;;
```

Now, we are ready to load the data. Let's open a new file and write a simple data import function using Weka's converter for reading files in CSV format:

```
import weka.core.Instances;  
import weka.core.converters.CSVLoader;  
import java.io.File;  
import java.io.IOException;  
  
public class EnergyLoad {  
  
    public static void main(String[] args) throws IOException {  
  
        // load CSV  
        CSVLoader loader = new CSVLoader();  
        loader.setSource(new File(args[0]));  
        Instances data = loader.getDataSet();  
  
        System.out.println(data);  
    }  
}
```

The data is loaded. Let's move on.

Analyzing attributes

Before we analyze attributes, let's first try to understand what we are dealing with. In total, there are eight attributes describing building characteristic and two target variables, heating and cooling load, as shown in the following table:

Attribute	Attribute name
x1	Relative compactness
x2	Surface area
x3	Wall area
x4	Roof area
x5	Overall height
x6	Orientation
x7	Glazing area
x8	Glazing area distribution
y1	Heating load
y2	Cooling load

Building and evaluating regression model

We will start with learning a model for heating load by setting the class attribute at the feature position:

```
data.setClassIndex(data.numAttributes() - 2);
```

The second target variable—cooling load—can be now removed:

```
//remove last attribute Y2
Remove remove = new Remove();
remove.setOptions(new String[] {"-R", data.numAttributes()+""});
remove.setInputFormat(data);
data = Filter.useFilter(data, remove);
```

Linear regression

We will start with a basic linear regression model implemented with the `LinearRegression` class. Similarly as in the classification example, we will initialize a new model instance, pass parameters and data, and invoke the `buildClassifier(Instances)` method, as follows:

```
import weka.classifiers.functions.LinearRegression;
...
data.setClassIndex(data.numAttributes() - 2);
LinearRegression model = new LinearRegression();
model.buildClassifier(data);
System.out.println(model);
```

The learned model, which is stored in the object, can be outputted by calling the `toString()` method, as follows:

```
Y1 =
```

```
-64.774 * x1 +
-0.0428 * x2 +
0.0163 * x3 +
-0.089 * x4 +
4.1699 * x5 +
19.9327 * x7 +
0.2038 * x8 +
83.9329
```

Linear regression model constructed a function that linearly combines the input

variables to estimate the heating load. The number in front of the feature explains the feature's impact on the target variable: sign corresponds to positive/negative impact, while magnitude corresponds to its significance. For instance, feature x_1 —relative compactness is negatively correlated with heating load, while glazing area is positively correlated. These two features also significantly impact the final heating load estimation.

The model performance can be similarly evaluated with cross-validation technique.

The 10-fold cross-validation is as follows:

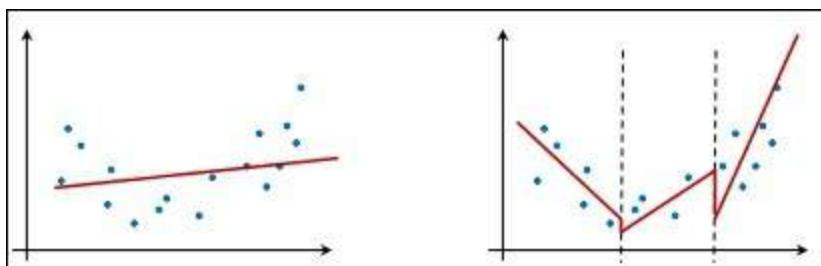
```
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(
model, data, 10, new Random(1), new String[]{} );
System.out.println(eval.toSummaryString());
```

We can output the common evaluation metrics including correlation, mean absolute error, relative absolute error, and so on, as follows:

Correlation coefficient	0.956
Mean absolute error	2.0923
Root mean squared error	2.9569
Relative absolute error	22.8555 %
Root relative squared error	29.282 %
Total Number of Instances	768

Regression trees

Another approach is to construct a set of regression models, each on its own part of the data. The following diagram shows the main difference between a regression model and a regression tree. Regression model constructs a single model that best fits all the data. Regression tree, on the other hand, constructs a set of regression models, each modeling a part of the data as shown on the right-hand side. Compared to the regression model, the regression tree can better fit the data, but the function is a piece-wise linear with jumps between modeled regions:



Regression tree in Weka is implemented within the `M5` class. Model construction follows the same paradigm: initialize model, pass parameters and data, and invoke the `buildClassifier(Instances)` method.

```
import weka.classifiers.trees.M5P;
...
M5P md5 = new M5P();
md5.setOptions(new String[]{""});
md5.buildClassifier(data);
System.out.println(md5);
```

The induced model is a tree with equations in the leaf nodes, as follows:

```
M5 pruned model tree:
(using smoothed linear models)

x1 <= 0.75 :
|   x7 <= 0.175 :
|   |   x1 <= 0.65 : LM1 (48/12.841%)
|   |   x1 >  0.65 : LM2 (96/3.201%)
|   x7 >  0.175 :
|   |   x1 <= 0.65 : LM3 (80/3.652%)
|   |   x1 >  0.65 : LM4 (160/3.502%)
x1 >  0.75 :
|   x1 <= 0.805 : LM5 (128/13.302%)
|   x1 >  0.805 :
|   |   x7 <= 0.175 :
|   |   |   x8 <= 1.5 : LM6 (32/20.992%)
|   |   |   x8 >  1.5 :
|   |   |   |   x1 <= 0.94 : LM7 (48/5.693%)
|   |   |   |   x1 >  0.94 : LM8 (16/1.119%)
|   |   x7 >  0.175 :
|   |   |   x1 <= 0.84 :
|   |   |   |   x7 <= 0.325 : LM9 (20/5.451%)
|   |   |   |   x7 >  0.325 : LM10 (20/5.632%)
|   |   |   x1 >  0.84 :
|   |   |   |   x7 <= 0.325 : LM11 (60/4.548%)
|   |   |   |   x7 >  0.325 :
|   |   |   |   |   x3 <= 306.25 : LM12 (40/4.504%)
|   |   |   |   |   x3 >  306.25 : LM13 (20/6.934%)

LM num: 1
Y1 =
 72.2602 * x1
 + 0.0053 * x3
```

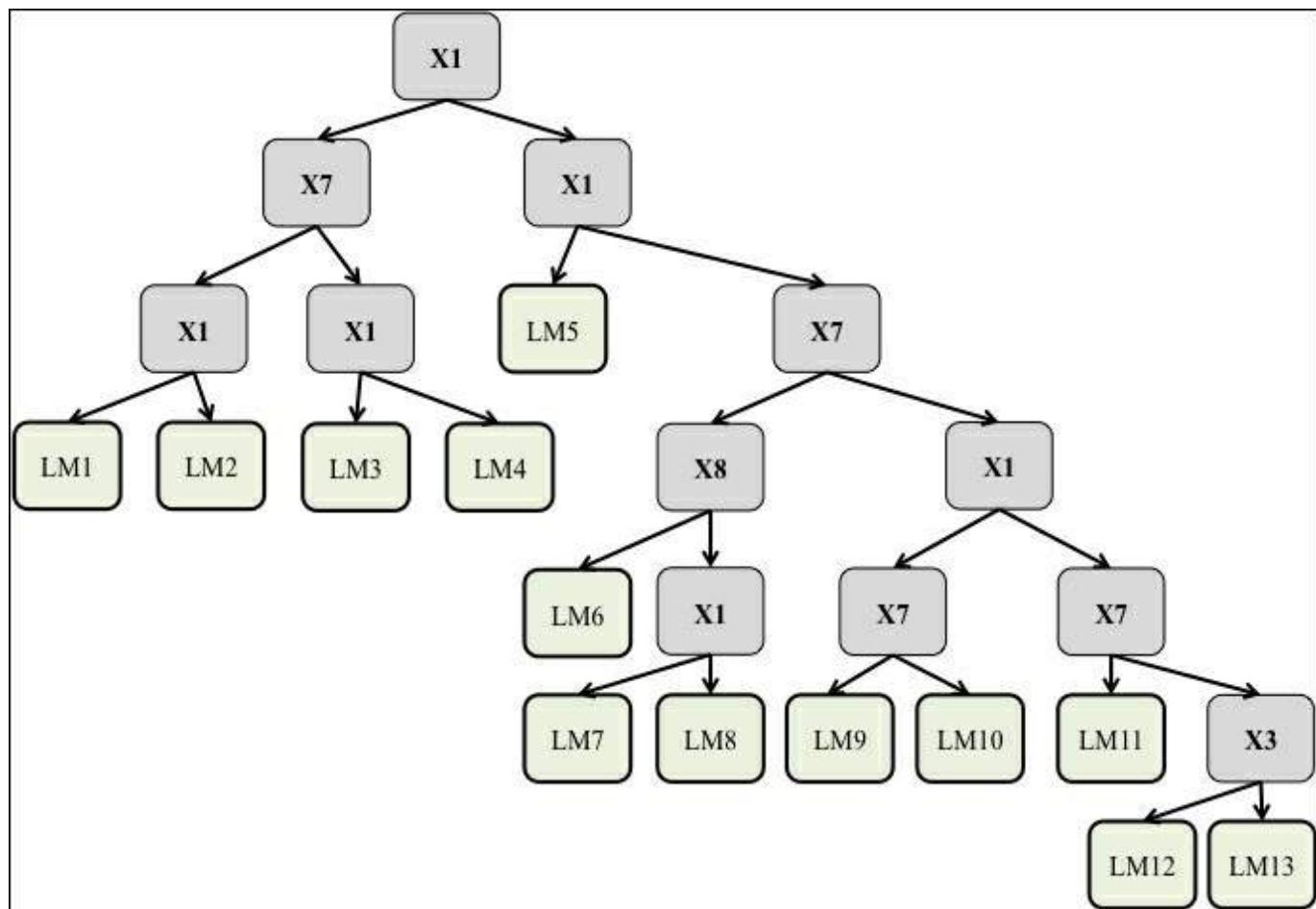
```
+ 11.1924 * x7  
+ 0.429 * x8  
- 36.2224
```

...

```
LM num: 13  
Y1 =  
5.8829 * x1  
+ 0.0761 * x3  
+ 9.5464 * x7  
- 0.0805 * x8  
+ 2.1492
```

Number of Rules : 13

The tree has 13 leaves, each corresponding to a linear equation. The preceding output is visualized in the following diagram:



The tree can be read similar to a classification tree. The most important features are at the top of the tree. The terminal node, leaf, contains a linear regression model explaining the data that reach this part of the tree.

Evaluation outputs the following results:

Correlation coefficient	0.9943
Mean absolute error	0.7446
Root mean squared error	1.0804
Relative absolute error	8.1342 %
Root relative squared error	10.6995 %
Total Number of Instances	768

Tips to avoid common regression problems

First, use prior studies and domain knowledge to figure out which features to include in regression. Check literature, reports, and previous studies on what kind of features work and reasonable variables for modeling your problem. Suppose you have a large set of features with random data, it is highly likely that several features will be correlated to the target variable (even though the data is random).

Keep the model simple to avoid overfitting. The Occam's razor principle states that you should select a model that best explains your data with the fewest assumptions. In practice, the model can be as simple as 2-4 predictor features.

Clustering

Compared to a supervised classifier, the goal of clustering is to identify intrinsic groups in a set of unlabeled data. It could be applied in identifying representative examples of homogeneous groups, finding useful and suitable groupings, or finding unusual examples, such as outliers.

We'll demonstrate how to implement clustering by analyzing the Bank dataset. The dataset consist of 11 attributes, describing 600 instances with age, sex, region, income, marriage status, children, car ownership status, saving activity, current activity, mortgage status, and PEP. In our analysis, we will try to identify the common groups of clients by applying the **Expectation Maximization (EM)** clustering.

EM works as follows: given a set of clusters, EM first assigns each instance with a probability distribution of belonging to a particular cluster. For example, if we start with three clusters— A , B , and C —an instance might get the probability distribution of 0.70, 0.10, and 0.20, belonging to the A , B , and C clusters, respectively. In the second step, EM re-estimates the parameter vector of the probability distribution of each class. The algorithm iterates these two steps until the parameters converge or the maximum number of iterations is reached.

The number of clusters to be used in EM can be set either manually or automatically by cross validation. Another approach to determining the number of clusters in a dataset includes the **elbow** method. The method looks at the percentage of variance that is explained with a specific number of clusters. The method suggests increasing the number of clusters until the additional cluster does not add much information, that is, explains little additional variance.

Clustering algorithms

The process of building a cluster model is quite similar to the process of building a classification model, that is, load the data and build a model. Clustering algorithms are implemented in the `weka.clusterers` package, as follows:

```
import java.io.BufferedReader;
import java.io.FileReader;

import weka.core.Instances;
import weka.clusterers.EM;

public class Clustering {

    public static void main(String args[]) throws Exception{

        //load data
        Instances data = new Instances(new BufferedReader(new
FileReader(args[0])));

        // new instance of clusterer
        EM model = new EM();
        // build the clusterer
        model.buildClusterer(data);
        System.out.println(model);

    }
}
```

The model identified the following six clusters:

```
EM
==
```

```
Number of clusters selected by cross validation: 6
```

Attribute	Cluster					
	0 (0.1)	1 (0.13)	2 (0.26)	3 (0.25)	4 (0.12)	5 (0.14)
<hr/>						
age						
0_34	10.0535	51.8472	122.2815	12.6207	3.1023	1.0948
35_51	38.6282	24.4056	29.6252	89.4447	34.5208	3.3755
52_max	13.4293	6.693	6.3459	50.8984	37.861	81.7724
[total]	62.1111	82.9457	158.2526	152.9638	75.4841	86.2428
sex						

FEMALE	27.1812	32.2338	77.9304	83.5129	40.3199	44.8218
MALE	33.9299	49.7119	79.3222	68.4509	34.1642	40.421
[total]	61.1111	81.9457	157.2526	151.9638	74.4841	85.2428
region						
INNER_CITY	26.1651	46.7431	73.874	60.1973	33.3759	34.6445
TOWN	24.6991	13.0716	48.4446	53.1731	21.617	
17.9946						
...						

The table can be read as follows: the first line indicates six clusters, while the first column shows attributes and their ranges. For example, the attribute `age` is split into three ranges: 0-34, 35-51, and 52-max. The columns on the left indicate how many instances fall into the specific range in each cluster, for example, clients in the 0-34 years age group are mostly in cluster #2 (122 instances).

Evaluation

A clustering algorithm's quality can be estimated using the log likelihood measure, which measures how consistent the identified clusters are. The dataset is split into multiple folds and clustering is run with each fold. The motivation here is that if the clustering algorithm assigns high probability to similar data that wasn't used to fit parameters, then it has probably done a good job of capturing the data structure. Weka offers the `ClusterEvaluation` class to estimate it, as follows:

```
double logLikelihood = ClusterEvaluation.crossValidateModel(model,  
data, 10, new Random(1));  
System.out.println(logLikelihood);
```

It has the following output:

```
-8.773410259774291
```

Summary

In this chapter, you learned how to implement basic machine learning tasks with Weka: classification, regression, and clustering. We briefly discussed attribute selection process, trained models, and evaluated their performance.

The next chapter will focus on how to apply these techniques to solve real-life problems, such as customer retention.

Chapter 4. Customer Relationship Prediction with Ensembles

Any type of company offering a service, product, or experience needs a solid understanding of relationship with their customers; therefore, **Customer Relationship Management (CRM)** is a key element of modern marketing strategies. One of the biggest challenges that businesses face is the need to understand exactly what causes a customer to buy new products.

In this chapter, we will work on a real-world marketing database provided by the French telecom company, Orange. The task will be to estimate the following likelihoods for customer actions:

- Switch provider (churn)
- Buy new products or services (appetency)
- Buy upgrades or add-ons proposed to them to make the sale more profitable (upselling)

We will tackle the **Knowledge Discovery and Data Mining (KDD) Cup 2009** challenge (KDD Cup, 2009) and show the steps to process the data using Weka. First, we will parse and load the data and implement the basic baseline models. Later, we will address advanced modeling techniques, including data pre-processing, attribute selection, model selection, and evaluation.

Note

KDD Cup is the leading data mining competition in the world. It is organized annually by **ACM Special Interest Group on Knowledge Discovery and Data Mining**. The winners are announced at the **Conference on Knowledge Discovery and Data Mining**, which is usually held in August.

Yearly archives, including all the corresponding datasets, are available here:
<http://www.kdd.org/kdd-cup>.

Customer relationship database

The most practical way to build knowledge on customer behavior is to produce scores that explain a target variable such as churn, appetency, or upselling. The score is computed by a model using input variables describing customers, for example, current subscription, purchased devices, consumed minutes, and so on. The scores are then used by the information system, for example, to provide relevant personalized marketing actions.

In 2009, the conference on KDD organized a machine learning challenge on customer-relationship prediction (KDD Cup, 2009).

Challenge

Given a large set of customer attributes, the task was to estimate the following three target variables (KDD Cup, 2009):

- **Churn probability**, in our context, is the likelihood a customer will switch providers:

Churn rate is also sometimes called attrition rate. It is one of two primary factors that determine the steady-state level of customers a business will support. In its broadest sense, churn rate is a measure of the number of individuals or items moving into or out of a collection over a specific period of time. The term is used in many contexts, but is most widely applied in business with respect to a contractual customer base. For instance, it is an important factor for any business with a subscriber-based service model, including mobile telephone networks and pay TV operators. The term is also used to refer to participant turnover in peer-to-peer networks.

- **Appetency probability**, in our context, is the propensity to buy a service or product
- **Upselling probability** is the likelihood that a customer will buy an add-on or upgrade:

Upselling is a sales technique whereby a salesman attempts to have the customer purchase more expensive items, upgrades, or other add-ons in an attempt to make a more profitable sale. Upselling usually involves marketing more profitable services or products, but upselling can also be simply exposing the customer to other options he or she may not have considered previously. Upselling can imply selling something additional, or selling something that is more profitable or otherwise preferable for the seller instead of the original sale.

The challenge was to beat the in-house system developed by Orange Labs. This was an opportunity for the participants to prove that they could handle a large database, including heterogeneous noisy data and unbalanced class distributions.

Dataset

For the challenge, the company Orange released a large dataset of customer data, containing about one million customers, described in ten tables with hundreds of fields. In the first step, they resampled the data to select a less unbalanced subset containing 100,000 customers. In the second step, they used an automatic feature construction tool that generated 20,000 features describing customers, which was then narrowed down to 15,000 features. In the third step, the dataset was anonymized by randomizing the order of features, discarding attribute names, replacing nominal variables with randomly generated strings, and multiplying continuous attributes by a random factor. Finally, all the instances were split randomly into a train and test dataset.

The KDD Cup provided two sets of data: large set and small set, corresponding to fast and slow challenge, respectively. They are described at the KDD Cup site as follows:

Both training and test sets contain 50,000 examples. The data are split similarly for the small and large versions, but the samples are ordered differently within the training and within the test sets. Both small and large datasets have numerical and categorical variables. For the large dataset, the first 14,740 variables are numerical and the last 260 are categorical. For the small dataset, the first 190 variables are numerical and the last 40 are categorical.

In this chapter, we will work with the small dataset consisting of 50,000 instances described with 230 variables each. Each of the 50,000 rows of data correspond to a client and are associated with three binary outcomes—one for each of the three challenges (upsell, churn, and appetency).

To make this clearer, the following image illustrates the dataset in a table format:

230 numeric and nominal attributes										Three binary classes		
Var85	Var123	Var125	Var126	Var132	Var133	Var134	Var225	Var229	Var230	Label Churn	Label Appetency	Label Upselling
12	6	720	8	0	1212385	69134				-1	-1	-1
2	72	0		8	4136430	357038				1	-1	-1
58	114	5967	-28	0	3478905	248932	kG3k	am7c		-1	-1	-1
0	0	0	-14	0	0	0				-1	-1	-1
0	0	15111	58	0	150650	66046	kG3k	mj86		-1	-1	-1
10	0	1935		8	641020	43684		am7c		-1	-1	-1
16	24	13194	-24	0	1664450	104978	kG3k	am7c		-1	-1	-1
2	12	0	-8	8	3839825	1284128				-1	-1	-1
2	90	2754		0	3830510	203586	kG3k	am7c		-1	-1	-1
24	66	6561		32	2577245	210014	kG3k			-1	-1	-1
6	12	5823	58	0	0	7134	kG3k	mj86		-1	-1	-1
28	24	66825	52	8	134105	15166	kG3k			-1	-1	-1
0	0	44154	10	0	0	0		mj86		-1	-1	-1
22	54	5202		0	2772010	1095062	xG3x			-1	-1	-1
0	102	31104	8	0	2170355	57596				-1	-1	1
0	0	2574		0	0	0	ELof	ojmt		-1	-1	-1
14	186	8019		48	3571845	587392	kG3k	am7c		-1	-1	-1
0	30	5319		8	500295	31436		am7c		-1	-1	-1
2	0	13788	4	0	918350	0	kG3k			-1	-1	-1
14	0	7110		0	2055150	392138				1	-1	-1
8	66	0	-8	0	3258940	1121306				-1	-1	-1
0	18	0	-10	0	0	0				-1	-1	-1
12	0	531	36	0	491345	56742	ELof	mj86		-1	-1	-1
0	12	16803	12	0	201110	1693090				1	-1	-1
14	0	25740		0	2932660	313200	xG3x			-1	-1	1

The table depicts the first 25 instances, that is, customers, each described with 250 attributes. For this example, only a selected subset of 10 attributes is shown. The dataset contains many missing values and even empty or constant attributes. The last three columns of the table correspond to the three distinct class labels corresponding to the ground truth, that is, if the customer indeed switched the provider (churn), bought a service (appetency), or bought an upgrade (upsell). However, note that the labels are provided separately from the data in three distinct files, hence, it is essential to retain the order of the instances and corresponding class labels to ensure proper correspondence.

Evaluation

The submissions were evaluated according to the arithmetic mean of the area under the ROC curve (AUC) for the three tasks, that is, churn, appetency, and upselling. ROC curve shows the performance of model as a curve obtained by plotting sensitivity against specificity for various threshold values used to determine the classification result (refer to [Chapter 1](#), *Applied Machine Learning Quick Start*, section *ROC curves*). Now, the AUC is related to the area under this curve, meaning larger the area, better the classifier. Most toolboxes, including Weka, provide an API to calculate AUC score.

Basic naive Bayes classifier baseline

As per the rules of the challenge, the participants had to outperform the basic naive Bayes classifier to qualify for prizes, which makes an assumption that features are independent (refer to [Chapter 1](#), *Applied Machine Learning Quick Start*).

The KDD Cup organizers run the vanilla naive Bayes classifier, without any feature selection or hyperparameter adjustments. For the large dataset, the overall scores of the naive Bayes on the test set were as follows:

- **Churn problem:** $AUC = 0.6468$
- **Appetency problem:** $AUC = 0.6453$
- **Upselling problem:** $AUC=0.7211$

Note that the baseline results are reported for large dataset only. Moreover, while both training and test datasets are provided at the KDD Cup site, the actual true labels for the test set are not provided. Therefore, when we process the data with our models, there is no way to know how well the models will perform on the test set. What we will do is use only the training data and evaluate our models with cross validation. The results will not be directly comparable, but, nevertheless, we have an idea for what a reasonable magnitude of the AUC score is.

Getting the data

At the KDD Cup web page (<http://kdd.org/kdd-cup/view/kdd-cup-2009/Data>), you should see a page that looks like the following screenshot. First, under the **Small version (230 var.)** header, download `orange_small_train.data.zip`. Next, download the three sets of true labels associated with this training data. The following files are found under the **Real binary targets (small)** header:

- `orange_small_train_appency.labels`
- `orange_small_train_churn.labels`
- `orange_small_train_upselling.labels`

Save and unzip all the files marked in the red boxes, as shown in the following screenshot:

SIGKDD : KDD Cup 2009

kdd.org/kdd-cup/view/kdd-cup-2009/Data

KDD

HOME CONFERENCES AWARDS PUBLICATIONS NEWS ABOUT SIGKDD CONTACT

Intro Tasks Rules Data Results FAQ Contacts

KDD Cup 2009

SEARCH KDD CUP ARCHIVES

Data

KDD Cup 2009: Customer relationship prediction

Data Download

Training and test data matrices and practice target values

The large dataset archives are available since the onset of the challenge. The small dataset will be made available at the end of the fast challenge. Both training and test sets contain **50,000 examples**. The data are split similarly for the small and large versions, but the samples are ordered differently within the training and within the test sets. Both small and large datasets have numerical and categorical variables. For the large dataset, the first **14,740 variables are numerical** and the last **260 are categorical**. For the small dataset, the first **190 variables are numerical** and the last **40 are categorical**. Toy target values are available only for practice purpose. The prediction of the toy target values will not be part of the final evaluation.

Small version (230 var.):

- [orange_small_train.data.zip \(8.2 Mbytes\)](#)
- [orange_small_test.data.zip \(8.2 Mbytes\)](#)

Large version (15,000 var.):

- [orange_large_train.data.chunk1.zip \(52.7 Mbytes\)](#)
- [orange_large_train.data.chunk2.zip \(52.7 Mbytes\)](#)
- [orange_large_train.data.chunk3.zip \(52.6 Mbytes\)](#)
- [orange_large_train.data.chunk4.zip \(52.5 Mbytes\)](#)
- [orange_large_train.data.chunk5.zip \(52.6 Mbytes\)](#)
- [orange_large_test.data.chunk1.zip \(52.8 Mbytes\)](#)
- [orange_large_test.data.chunk2.zip \(52.5 Mbytes\)](#)
- [orange_large_test.data.chunk3.zip \(52.6 Mbytes\)](#)
- [orange_large_test.data.chunk4.zip \(52.6 Mbytes\)](#)
- [orange_large_test.data.chunk5.zip \(52.6 Mbytes\)](#)

Toy targets (large):

- [orange_large_train_toy.labels](#)

True task labels

Real binary targets (small):

- [orange_small_train_appency.labels](#)
- [orange_small_train_churn.labels](#)
- [orange_small_train_upselling.labels](#)

KDD Cup Archive

- KDD Cup 2016
- KDD Cup 2014
- KDD Cup 2013 (Track 2)
- KDD Cup 2013 (Track 1)
- KDD Cup 2012 (Track 2)
- KDD Cup 2012 (Track 1)
- KDD Cup 2011
- KDD Cup 2010
- KDD CUP 2009**
- KDD Cup 2008
- KDD Cup 2007
- KDD Cup 2006
- KDD Cup 2005
- KDD Cup 2004
- KDD Cup 2003
- KDD Cup 2002
- KDD Cup 2001
- KDD Cup 2000
- KDD Cup 1999
- KDD Cup 1998
- KDD Cup 1997

In the following sections, we will first load the data into Weka and apply basic modeling with the naive Bayes to obtain our own baseline AUC scores. Later, we will look into more advanced modeling techniques and tricks.

Loading the data

We will load the data to Weka directly from the `.cvs` format. For this purpose, we will write a function that accepts the path to the data file and the true labels file. The function will load and merge both datasets and remove empty attributes:

```
public static Instances loadData(String pathData, String pathLabels)
throws Exception {
```

First, we load the data using the `CSVLoader()` class. Additionally, we specify the `\t` tab as a field separator and force the last 40 attributes to be parsed as nominal:

```
// Load data
CSVLoader loader = new CSVLoader();
loader.setFieldSeparator("\t");
loader.setNominalAttributes("191-last");
loader.setSource(new File(pathData));
Instances data = loader.getDataSet();
```

Note

The `CSVLoader` class accepts many additional parameters specifying column separator, string enclosures, whether a header row is present or not, and so on. Complete documentation is available here:

<http://weka.sourceforge.net/doc.dev/weka/core/converter/CSVLoader.html>

Next, some of the attributes do not contain a single value and Weka automatically recognizes them as the `String` attributes. We actually do not need them, so we can safely remove them using the `RemoveType` filter. Additionally, we specify the `-T` parameters, which means remove attribute of specific type and the attribute type that we want to remove:

```
// remove empty attributes identified as String attribute
RemoveType removeString = new RemoveType();
removeString.setOptions(new String[] {"-T", "string"});
removeString.setInputFormat(data);
Instances filteredData = Filter.useFilter(data, removeString);
```

Alternatively, we could use the `void deleteStringAttributes()` method implemented within the `Instances` class, which has the same effect, for example, `data.removeStringAttributes()`.

Now, we will load and assign class labels to the data. We will again utilize `CVSLoader`, where we specify that the file does not have any header line, that is, `setNoHeaderRowPresent(true)`:

```
// Load labels
loader = new CSVLoader();
loader.setFieldSeparator("\t");
loader.setNoHeaderRowPresent(true);
loader.setNominalAttributes("first-last");
loader.setSource(new File(pathLabelles));
Instances labels = loader.getDataSet();
```

Once we have loaded both files, we can merge them together by calling the `Instances.mergeInstances(Instances, Instances)` static method. The method returns a new dataset that has all the attributes from the first dataset plus the attributes from the second set. Note that the number of instances in both datasets must be the same:

```
// Append label as class value
Instances labeledData = Instances.mergeInstances(filteredData,
labelles);
```

Finally, we set the last attribute, that is, the label attribute that we have just added, as a target variable and return the resulting dataset:

```
// set the label attribute as class
labeledData.setClassIndex(labeledData.numAttributes() - 1);

System.out.println(labeledData.toSummaryString());
return labeledData;
}
```

The function outputs a summary as shown in the following and returns the labeled dataset:

```
Relation Name: orange_small_train.data-
weka.filters.unsupervised.attribute.RemoveType-
Tstring_orange_small_train_churn.labels.txt
Num Instances: 50000
Num Attributes: 215
```

Name	Type	Nom	Int	Real	Missing	Unique	Dist
1 Var1	Num	0%	1%	0%	49298 / 99%	8 / 0%	18
2 Var2	Num	0%	2%	0%	48759 / 98%	1 / 0%	2
3 Var3	Num	0%	2%	0%	48760 / 98%	104 / 0%	146

4 Var4 Num 0% 3% 0% **48421 / 97%** 1 / 0% **4**

...

Basic modeling

In this section, we will implement our own baseline model by following the approach that the KDD Cup organizers took. However, before we go to the model, let's first implement the evaluation engine that will return AUC on all three problems.

Evaluating models

Now, let's take a closer look at the evaluation function. The evaluation function accepts an initialized model, cross-validates the model on all three problems, and reports the results as an area under the ROC curve (AUC), as follows:

```
public static double[] evaluate(Classifier model) throws Exception {  
  
    double results[] = new double[4];  
  
    String[] labelFiles = new String[]{  
        "churn", "appetency", "upselling"};  
  
    double overallScore = 0.0;  
    for (int i = 0; i < labelFiles.length; i++) {
```

First, we call the `Instance loadData(String, String)` function that we implemented earlier to load the train data and merge it with the selected labels:

```
// Load data  
Instances train_data = loadData(  
    path + "orange_small_train.data",  
    path+"orange_small_train_"+labelFiles[i]+".labels.txt");
```

Next, we initialize the `weka.classifiers.Evaluation` class and pass our dataset (the dataset is used only to extract data properties, the actual data are not considered). We call the `void crossValidateModel(Classifier, Instances, int, Random)` method to begin cross validation and select to create five folds. As validation is done on random subsets of the data, we need to pass a random seed as well:

```
// cross-validate the data  
Evaluation eval = new Evaluation(train_data);  
eval.crossValidateModel(model, train_data, 5,  
    new Random(1));
```

After the evaluation completes, we read the results by calling the `double areUnderROC(int)` method. As the metric depends on the target value that we are interested in, the method expects a class value index, which can be extracted by searching the index of the "1" value in the class attribute:

```
// Save results  
results[i] = eval.areaUnderROC(
```

```
    train_data.classAttribute().indexOfValue("1"));
overallScore += results[i];
}
```

Finally, the results are averaged and returned:

```
// Get average results over all three problems
results[3] = overallScore / 3;
return results;
}
```

Implementing naive Bayes baseline

Now, when we have all the ingredients, we can replicate the naive Bayes approach that we are expected to outperform. This approach will not include any additional data pre-processing, attribute selection, and model selection. As we do not have true labels for test data, we will apply the five-fold cross validation to evaluate the model on a small dataset.

First, we initialize a naive Bayes classifier, as follows:

```
Classifier baselineNB = new NaiveBayes();
```

Next, we pass the classifier to our evaluation function, which loads the data and applies cross validation. The function returns an area under the ROC curve score for all three problems and overall results:

```
double resNB[] = evaluate(baselineNB);
System.out.println("Naive Bayes\n" +
"\tchurn: " + resNB[0] + "\n" +
"\tappetency: " + resNB[1] + "\n" +
"\tup-sell: " + resNB[2] + "\n" +
"\toverall: " + resNB[3] + "\n");
```

In our case, the model achieves the following results:

```
Naive Bayes
churn:    0.5897891153549814
appetency: 0.630778394752436
up-sell:   0.6686116692438094
overall:   0.6297263931170756
```

These results will serve as a baseline when we tackle the challenge with more advanced modeling. If we process the data with significantly more sophisticated, time-consuming, and complex techniques, we expect the results to be much better. Otherwise, we are simply wasting the resources. In general, when solving machine learning problems, it is always a good idea to create a simple baseline classifier that serves us as an orientation point.

Advanced modeling with ensembles

In the previous section, we implemented an orientation baseline, so let's focus on heavy machinery. We will follow the approach taken by the KDD Cup 2009 winning solution developed by the **IBM Research** team (Niculescu-Mizil and others, 2009).

Their strategy to address the challenge was using the **Ensemble Selection** algorithm (Caruana and Niculescu-Mizil, 2004). This is an ensemble method, which means it constructs a series of models and combines their output in a specific way to provide the final classification. It has several desirable properties as shown in the following list that make it a good fit for this challenge:

- It was proven to be robust, yielding excellent performance
- It can be optimized for a specific performance metric, including AUC
- It allows different classifiers to be added to the library
- It is an anytime method, meaning that, if we run out of time, we have a solution available

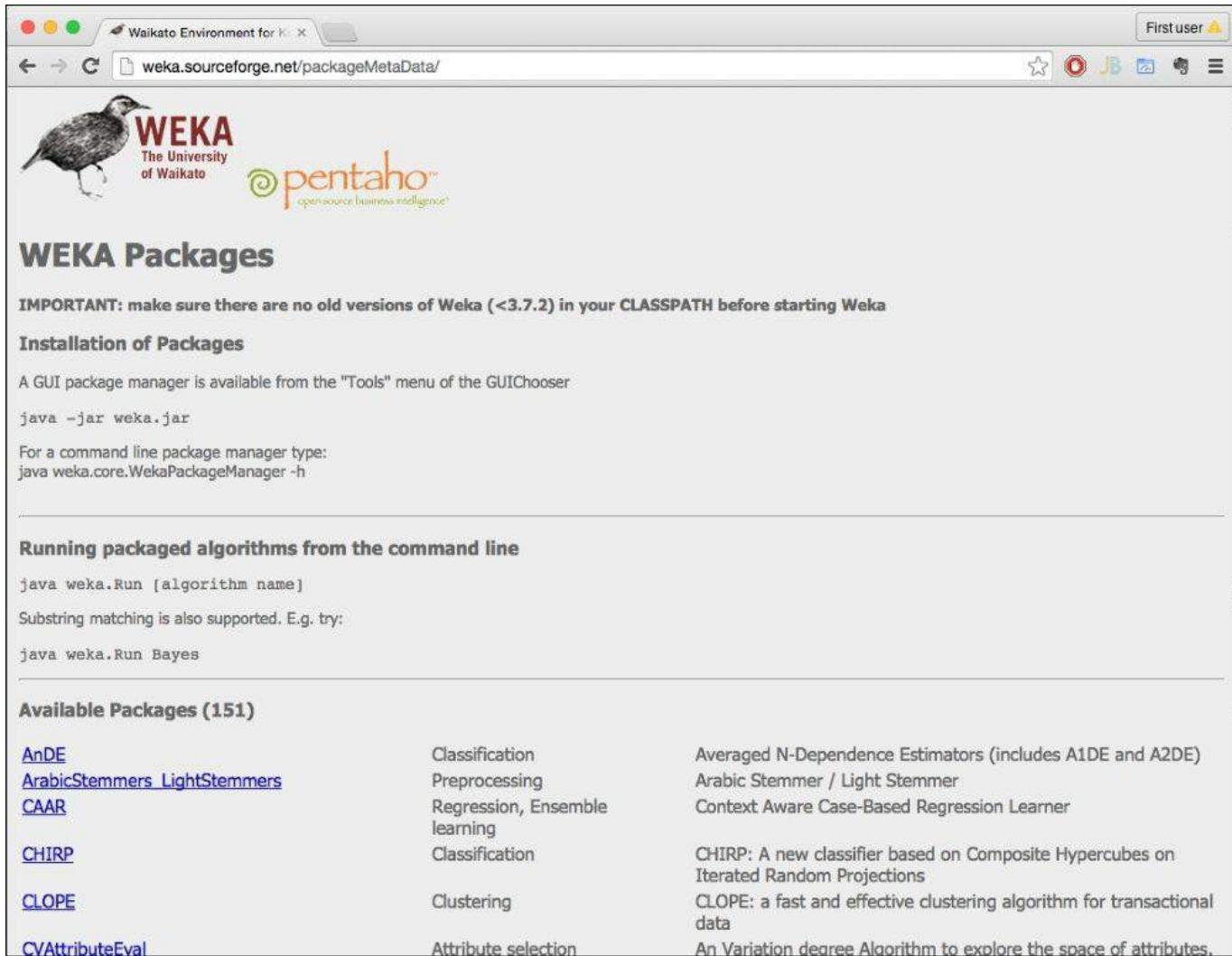
In this section, we will loosely follow the steps as described in their report. Note, this is not an exact implementation of their approach, but rather a solution overview that will include the necessary steps to dive deeper.

The general overview of steps is as follows:

1. First, we will preprocess the data by removing attributes that clearly do not bring any value, for example, all the missing or constant values; fixing missing values in order to help machine learning algorithms, which cannot deal with them; and converting categorical attributes to numerical.
2. Next, we will run attributes selection algorithm to select only a subset of attribute that can help in prediction of tasks.
3. In the third step, we will instantiate the Ensemble Selection algorithms with a wide variety of models, and, finally, evaluate the performance.

Before we start

For this task, we will need an additional Weka package, `ensembleLibrary`. Weka 3.7.2 or higher versions support external packages developed mainly by the academic community. A list of **WEKA Packages** is available at <http://weka.sourceforge.net/packageMetaData> as shown at the following screenshot:



The screenshot shows a web browser window titled "Waikato Environment for Knowledge Analysis" with the URL "weka.sourceforge.net/packageMetaData/". The page features the WEKA logo (a bird) and the Pentaho logo (a stylized orange 'p'). The main content area is titled "WEKA Packages". It includes a warning: "IMPORTANT: make sure there are no old versions of Weka (<3.7.2) in your CLASSPATH before starting Weka". Below this, there are sections for "Installation of Packages" (describing GUI and command-line package managers) and "Running packaged algorithms from the command line" (with examples like "java weka.Run [algorithm name]"). The final section, "Available Packages (151)", lists 151 packages with their descriptions, such as AnDE, ArabicStemmers, LightStemmers, CAAR, CHIRP, CLOPE, and CVAttributeEval.

Packages	Description
AnDE	Classification
ArabicStemmers	Preprocessing
LightStemmers	Regression, Ensemble learning
CAAR	Classification
CHIRP	Clustering
CLOPE	Attribute selection
CVAttributeEval	Averaged N-Dependence Estimators (includes A1DE and A2DE) Arabic Stemmer / Light Stemmer Context Aware Case-Based Regression Learner CHIRP: A new classifier based on Composite Hypercubes on Iterated Random Projections CLOPE: a fast and effective clustering algorithm for transactional data An Variation degreee Alorithm to explore the space of attributes.

Find and download the latest available version of the `ensembleLibrary` package at <http://prdownloads.sourceforge.net/weka/ensembleLibrary1.0.5.zip?download>.

After you unzip the package, locate `ensembleLibrary.jar` and import it to your code, as follows:

```
import weka.classifiers.meta.EnsembleSelection;
```

Data pre-processing

First, we will utilize Weka's built-in

`weka.filters.unsupervised.attribute.RemoveUseless` filter, which works exactly as its name suggests. It removes the attributes that do not vary much, for instance, all constant attributes are removed, and attributes that vary too much, almost at random. The maximum variance, which is applied only to nominal attributes, is specified with the `-M` parameter. The default parameter is 99%, which means that if more than 99% of all instances have unique attribute values, the attribute is removed, as follows:

```
RemoveUseless removeUseless = new RemoveUseless();
removeUseless.setOptions(new String[] { "-M", "99" });// threshold
removeUseless.setInputFormat(data);
data = Filter.useFilter(data, removeUseless);
```

Next, we will replace all the missing values in the dataset with the modes (nominal attributes) and means (numeric attributes) from the training data by using the `weka.filters.unsupervised.attribute.ReplaceMissingValues` filter. In general, missing values replacement should be proceeded with caution while taking into consideration the meaning and context of the attributes:

```
ReplaceMissingValues fixMissing = new ReplaceMissingValues();
fixMissing.setInputFormat(data);
data = Filter.useFilter(data, fixMissing);
```

Finally, we will discretize numeric attributes, that is, we transform numeric attributes into intervals using the `weka.filters.unsupervised.attribute.Discretize` filter. With the `-B` option, we set to split numeric attributes into four intervals, and the `-R` option will specify the range of attributes (only numeric attributes will be discretized):

```
Discretize discretizeNumeric = new Discretize();
discretizeNumeric.setOptions(new String[] {
    "-B", "4", // no of bins
    "-R", "first-last")); //range of attributes
fixMissing.setInputFormat(data);
data = Filter.useFilter(data, fixMissing);
```

Attribute selection

In the next step, we will select only informative attributes, that is, attributes that more likely help with prediction. A standard approach to this problem is to check the information gain carried by each attribute. We will use the

`weka.attributeSelection.AttributeSelection` filter, which requires two additional methods: evaluator, that is, how attribute usefulness is calculated, and search algorithms, that is, how to select a subset of attributes.

In our case, we first initialize `weka.attributeSelection.InfoGainAttributeEval` that implements calculation of information gain:

```
InfoGainAttributeEval eval = new InfoGainAttributeEval();
Ranker search = new Ranker();
```

To select only top attributes above some threshold, we initialize `weka.attributeSelection.Ranker` to rank the attributes with information gain above a specific threshold. We specify this with the `-T` parameter, while keeping the value of the threshold low to keep the attributes with at least some information:

```
search.setOptions(new String[] { "-T", "0.001" });
```

Tip

The general rule for settings this threshold is to sort the attributes by information gain and pick the threshold where the information gain drops to negligible value.

Next, we can initialize the `AttributeSelection` class, set the evaluator and ranker, and apply the attribute selection to our dataset:

```
AttributeSelection attSelect = new AttributeSelection();
attSelect.setEvaluator(eval);
attSelect.setSearch(search);

// apply attribute selection
attSelect.SelectAttributes(data);
```

Finally, we remove the attributes that were not selected in the last run by calling the `reduceDimensionality(Instances)` method.

```
// remove the attributes not selected in the last run
data = attSelect.reduceDimensionality(data);
```

At the end, we are left with 214 out of 230 attributes.

Model selection

Over the years, practitioners in the field of machine learning have developed a wide variety of learning algorithms and improvements to the existing ones. There are so many unique supervised learning methods that it is challenging to keep track of all of them. As characteristics of the datasets vary, no one method is the best in all the cases, but different algorithms are able to take advantage of different characteristics and relationships of a given dataset. The property the Ensemble Selection algorithm is to try to leverage (Jung, 2005):

Intuitively, the goal of ensemble selection algorithm is to automatically detect and combine the strengths of these unique algorithms to create a sum that is greater than the parts. This is accomplished by creating a library that is intended to be as diverse as possible to capitalize on a large number of unique learning approaches. This paradigm of overproducing a huge number of models is very different from more traditional ensemble approaches. Thus far, our results have been very encouraging.

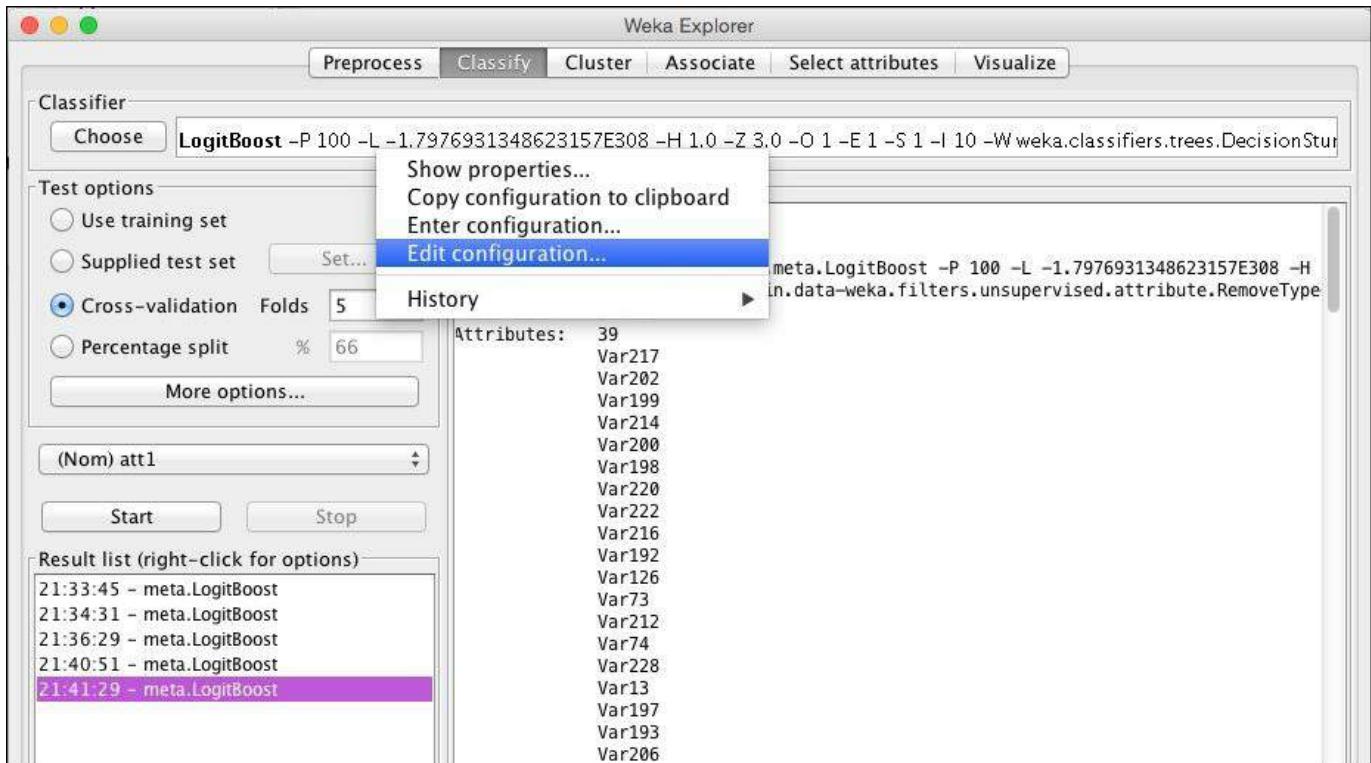
First, we need to create the model library by initializing the `weka.classifiers.EnsembleLibrary` class, which will help us define the models:

```
EnsembleLibrary ensembleLib = new EnsembleLibrary();
```

Next, we add the models and their parameters as strings to the library as string values, for example, we can add three decision tree learners with different parameters, as follows:

```
ensembleLib.addModel("weka.classifiers.trees.J48 -S -C 0.25 -B -M 2");  
ensembleLib.addModel("weka.classifiers.trees.J48 -S -C 0.25 -B -M 2 -A");
```

If you are familiar with the Weka graphical interface, you can also explore the algorithms and their configurations there and copy the configuration as shown in the following screenshot: right-click on the algorithm name and navigate to **Edit configuration | Copy configuration string**:



To complete the example, we added the following algorithms and their parameters:

- Naive Bayes that was used as default baseline:

```
ensembleLib.addModel("weka.classifiers.bayes.NaiveBayes");
```

- k-nearest neighbors based on lazy models?:

```
ensembleLib.addModel("weka.classifiers.lazy.IBk");
```

- Logistic regression as simple logistic with default parameters:

```
ensembleLib.addModel("weka.classifiers.functions.SimpleLogistic");
```

- Support vector machines with default parameters:

```
ensembleLib.addModel("weka.classifiers.functions.SMO");
```

- AdaBoost, which is an ensemble method itself:

```
ensembleLib.addModel("weka.classifiers.meta.AdaBoostM1");
```

- LogitBoost, an ensemble method based on logistic regression:

```
ensembleLib.addModel("weka.classifiers.meta.LogitBoost");
```

- Decision stump, an ensemble method based on one-level decision trees:

```
ensembleLib.addModel("classifiers.trees.DecisionStump");
```

As the EnsembleLibrary implementation is primarily focused on GUI and console users, we have to save the models into a file by calling the `saveLibrary(File, EnsembleLibrary, JComponent)` method, as follows:

```
EnsembleLibrary.saveLibrary(new File(path+"ensembleLib.model.xml"),  
ensembleLib, null);  
System.out.println(ensembleLib.getModels());
```

Next, we can initialize the Ensemble Selection algorithm by instantiating the `weka.classifiers.meta.EnsembleSelection` class. Let's first review the following method options:

- `-L </path/to/modelLibrary>`: This specifies the modelLibrary file, continuing the list of all models.
- `-W </path/to/working/directory>`: This specifies the working directory, where all models will be stored.
- `-B <numModelBags>`: This sets the number of bags, that is, the number of iterations to run the Ensemble Selection algorithm.
- `-E <modelRatio>`: This sets the ratio of library models that will be randomly chosen to populate each bag of models.
- `-V <validationRatio>`: This sets the ratio of the training data set that will be reserved for validation.
- `-H <hillClimbIterations>`: This sets the number of hill climbing iterations to be performed on each model bag.
- `-I <sortInitialization>`: This sets the ratio of the ensemble library that the sort initialization algorithm will be able to choose from, while initializing the ensemble for each model bag.
- `-X <numFolds>`: This sets the number of cross validation folds.
- `-P <hillclimbMetric>`: This specifies the metric that will be used for model selection during the hill climbing algorithm. Valid metrics are accuracy, rmse, roc, precision, recall, fscore, and all.
- `-A <algorithm>`: This specifies the algorithm to be used for ensemble selection. Valid algorithms are forward (default) for forward selection, backward for backward elimination, both for both forward and backward elimination, best to simply print the top performer from the ensemble library, and library to only train the models in the ensemble library.

- **-R**: This flags whether or not the models can be selected more than once for an ensemble.
- **-G**: This states whether the sort initialization greedily stops adding models when the performance degrades.
- **-O**: This is a flag for verbose output. This prints the performance of all the selected models.
- **-S <num>**: This is a random number seed (default 1).
- **-D**: If set, the classifier is run in the debug mode and may output additional information to the console.

We initialize the algorithm with the following initial parameters, where we specified optimizing the ROC metric:

```
EnsembleSelection ensambleSel = new EnsembleSelection();
ensambleSel.setOptions(new String[]{
    "-L", path+"ensembleLib.model.xml", // </path/to/modelLibrary>"-W",
    path+"esTmp", // </path/to/working/directory> -
    "-B", "10", // <numModelBags>
    "-E", "1.0", // <modelRatio>.
    "-V", "0.25", // <validationRatio>
    "-H", "100", // <hillClimbIterations>
    "-I", "1.0", // <sortInitialization>
    "-X", "2", // <numFolds>
    "-P", "roc", // <hillclimbMettric>
    "-A", "forward", // <algorithm>
    "-R", "true", // - Flag to be selected more than once
    "-G", "true", // - stops adding models when performance degrades
    "-O", "true", // - verbose output.
    "-S", "1", // <num> - Random number seed.
    "-D", "true" // - run in debug mode
});
```

Performance evaluation

The evaluation is heavy both computationally and memory-wise, so make sure that you initialize the JVM with extra heap space—for instance, **java -Xmx16g**—while the computation can take a couple of hours or days, depending on the number of algorithms you include in the model library. This example took 4 hours and 22 minutes on 12-core Intel Xeon E5-2420 CPU with 32 GB of memory, utilizing 10% CPU and 6 GB of memory on average.

We call our evaluation method and output the results, as follows:

```
double resES[] = evaluate(ensambleSel);
System.out.println("Ensemble Selection\n"
+ "\tchurn:      " + resES[0] + "\n"
+ "\tappetency:  " + resES[1] + "\n"
+ "\tup-sell:    " + resES[2] + "\n"
+ "\toverall:   " + resES[3] + "\n");
```

The specific set of classifiers in the model library achieved the following result:

```
Ensamble
churn:      0.7109874158176481
appetency:  0.786325687118347
up-sell:    0.8521363243575182
overall:   0.7831498090978378
```

Overall, the approach has brought us to a significant improvement of more than 15 percentage points compared to the initial baseline that we designed at the beginning of the chapter. While it is hard to give a definite answer, the improvement was mainly due to three factors: data pre-processing and attribute selection, exploration of a large variety of learning methods, and use of an ensemble-building technique that is able to take advantage of the variety of base classifiers without overfitting. However, the improvement requires a significant increase in processing time, as well as working memory.

Summary

In this chapter, we tackled the KDD Cup 2009 challenge on customer-relationship prediction, where we implemented the data pre-processing steps, addressing the missing values and redundant attributes. We followed the winning KDD Cup solution, studying how to leverage ensemble methods using a basket of learning algorithms, which can significantly boost the classification performance.

In the next chapter, we will tackle another problem addressing the customer behavior, that is, the analysis of purchasing behavior, where you will learn how to use algorithms that detect frequently occurring patterns.

Chapter 5. Affinity Analysis

Affinity analysis is the heart of **Market basket analysis (MBA)**. It can discover co-occurrence relationships among activities performed by specific users or groups. In retail, affinity analysis can help you understand the purchasing behavior of customers. These insights can drive revenue through smart cross-selling and upselling strategies and can assist you in developing loyalty programs, sales promotions, and discount plans.

In this chapter, we will look into the following topics:

- Market basket analysis
- Association rule learning
- Other applications in various domains

First, we will revise the core association rule learning concepts and algorithms, such as support, lift, **Apriori algorithm**, and **FP-growth algorithm**. Next, we will use Weka to perform our first affinity analysis on supermarket dataset and study how to interpret the resulting rules. We will conclude the chapter by analyzing how association rule learning can be applied in other domains, such as **IT Operations Analytics**, medicine, and others.

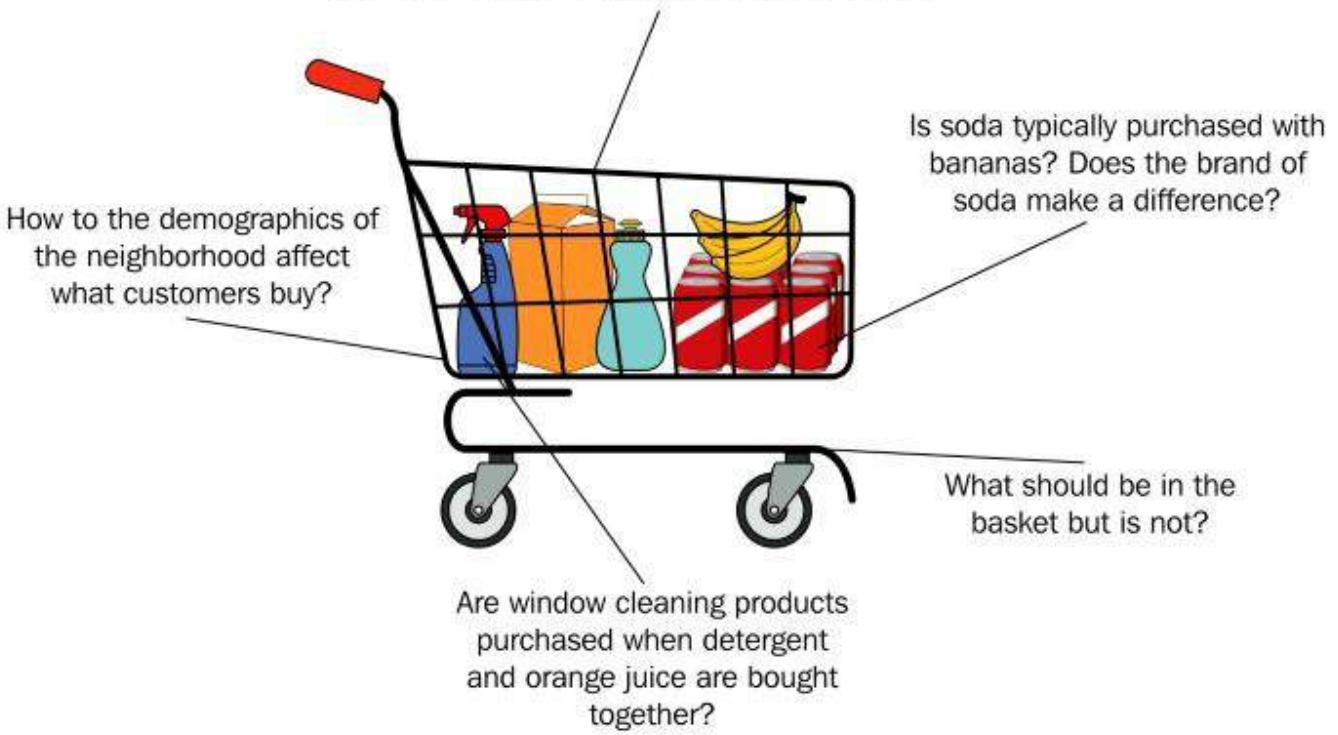
Market basket analysis

Since the introduction of electronic point of sale, retailers have been collecting an incredible amount of data. To leverage this data in order to produce business value, they first developed a way to consolidate and aggregate the data to understand the basics of the business. What are they selling? How many units are moving? What is the sales amount?

Recently, the focus shifted to the lowest level of granularity—the market basket transaction. At this level of detail, the retailers have direct visibility into the market basket of each customer who shopped at their store, understanding not only the quantity of the purchased items in that particular basket, but also how these items were bought in conjunction with each other. This can be used to drive decisions about how to differentiate store assortment and merchandise, as well as effectively combine offers of multiple products, within and across categories, to drive higher sales and profits. These decisions can be implemented across an entire retail chain, by channel, at the local store level, and even for a specific customer with so-called personalized marketing, where a unique product offering is made for each customer.

Questions in a Shopping Cart

In this shopping basket, the shopper placed a quart of orange juice, some bananas, dish detergent, some window cleaner and a six-pack of soda.



MBA covers a wide variety of analysis:

- **Item affinity:** This defines the likelihood of two (or more) items being purchased together
- **Identification of driver items:** This enables the identification of the items that drive people to the store and always need to be in stock
- **Trip classification:** This analyzes the content of the basket and classifies the shopping trip into a category: weekly grocery trip, special occasion, and so on
- **Store-to-store comparison:** Understanding the number of baskets allows any metric to be divided by the total number of baskets, effectively creating a convenient and easy way to compare the stores with different characteristics (units sold per customer, revenue per transaction, number of items per basket, and so on)
- **Revenue optimization:** This helps in determining the magic price points for this store, increasing the size and value of the market basket

- **Marketing:** This helps in identifying more profitable advertising and promotions, targeting offers more precisely in order to improve ROI, generating better loyalty card promotions with longitudinal analysis, and attracting more traffic to the store
- **Operations optimization:** This helps in matching the inventory to the requirement by customizing the store and assortment to trade area demographics, optimizing store layout

Predictive models help retailers to direct the right offer to the right customer segments/profiles, as well as gain understanding of what is valid for which customer, predict the probability score of customers responding to this offer, and understand the customer value gain from the offer acceptance.

Affinity analysis

Affinity analysis is used to determine the likelihood that a set of items will be bought together. In retail, there are natural product affinities, for example, it is very typical for people who buy hamburger patties to buy hamburger rolls, along with ketchup, mustard, tomatoes, and other items that make up the burger experience.

While there are some product affinities that might seem trivial, there are some affinities that are not very obvious. A classic example is toothpaste and tuna. It seems that people who eat tuna are more prone to brush their teeth right after finishing their meal. So, why is it important for retailers to get a good grasp of the product affinities? This information is critical to appropriately plan promotions as reducing the price for some items may cause a spike on related high-affinity items without the need to further promote these related items.

In the following section, we'll look into the algorithms for association rule learning: Apriori and FP-growth.

Association rule learning

Association rule learning has been a popular approach to discover interesting relations hips between items in large databases. It is most commonly applied in retail to reveal regularities between products.

Asociation rule learning approaches find patterns as interesting strong rules in the database using different measures of interestingness. For example, the following rule would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat: $\{onions, potatoes\} \rightarrow \{burger\}$

Another classic story probably told in every machine learning class is the beer and diaper story. An analysis of supermarket shoppers' behavior showed that customers, presumably young men, who buy diapers tend also to buy beer. It immediately became a popular example of how an unexpected association rule might be found from everyday data; however, there are varying opinions as to how much of the story is true. Daniel Powers says (DSS News, 2002):

"In 1992, Thomas Blischok, manager of a retail consulting group at Teradata, and his staff prepared an analysis of 1.2 million market baskets from about 25 Osco Drug stores. Database queries were developed to identify affinities. The analysis "did discover that between 5:00 and 7:00 p.m. consumers bought beer and diapers". Osco managers did NOT exploit the beer and diapers relationship by moving the products closer together on the shelves."

In addition to the preceding example from MBA, association rules are today employed in many application areas, including web usage mining, intrusion detection, continuous production, and bioinformatics. We'll take a closer look at these areas later in this chapter.

Basic concepts

Before we dive into algorithms, let's first review the basic concepts.

Database of transactions

In association rule mining, the dataset is structured a bit differently than the approach presented in the first chapter. First, there is no class value, as this is not required for learning association rules. Next, the dataset is presented as a transactional table, where each supermarket item corresponds to a binary attribute. Hence, the feature vector could be extremely large.

Consider the following example. Suppose we have four receipts as shown in the following image. Each receipt corresponds to a purchasing transaction:

WWW.MACHINE-LEARNING-JAVA.COM			WWW.MACHINE-LEARNING-JAVA.COM			WWW.MACHINE-LEARNING-JAVA.COM			WWW.MACHINE-LEARNING-JAVA.COM		
GROCERY STORE 921 JAVA AVENUE NEW YORK NY 9999			GROCERY STORE 921 JAVA AVENUE NEW YORK NY 9999			GROCERY STORE 921 JAVA AVENUE NEW YORK NY 9999			GROCERY STORE 921 JAVA AVENUE NEW YORK NY 9999		
PURCHASE:			PURCHASE:			PURCHASE:			PURCHASE:		
POTATOES	\$4.12	POTATOES	\$4.12	BURGER	\$12.84	BURGER	\$12.84	ONIONS	\$8.14	DIPPERS	\$25.95
BURGER	\$12.84	ONIONS	\$8.14	BEER	\$27.65	BEER	\$27.65				
VAT +1%	TRX#	\$1.77	VAT +1%	TRX#	\$6.15	VAT +1%	TRX#	\$6.88	VAT +1%	TRX#	\$47.48
TOTAL: \$17.98			TOTAL: \$62.88			TOTAL: \$68.88			TOTAL: \$47.48		
PAYMENT METHOD: CREDIT CARD TRANSACTION #1456293867 -001 DATE:10/08/2016 9:29:27 AM			PAYMENT METHOD: CREDIT CARD TRANSACTION #145629842E -001 DATE:10/08/2016 9:30:28 AM			PAYMENT METHOD: CREDIT CARD TRANSACTION #1456293568 -001 DATE:10/08/2016 9:31:48 AM			PAYMENT METHOD: CREDIT CARD TRANSACTION #1456293459 -001 DATE:10/08/2016 9:30:59 AM		
THANK YOU			THANK YOU			THANK YOU			THANK YOU		

To write these receipts in the form of a transactional database, we first identify all the possible items that appear in the receipts. These items are onions, potatoes, burger, beer, and dippers. Each purchase, that is, transaction, is presented in a row, and there is *1* if an item was purchased within the transaction and *0* otherwise, as shown in the following table:

Transaction ID	Onions	Potatoes	Burger	Beer	Dippers
1	0	1	1	0	0
2	1	1	1	1	0

3	0	0	0	1	1	
4	1	0	1	1	0	

This example is really small. In practical applications, the dataset often contains thousands or millions of transactions, which allow learning algorithm the discovery of statistically significant patterns.

Itemset and rule

Itemset is simply a set of items, for example, $\{onions, potatoes, burger\}$. A rule consists of two itemsets, X and Y , in the following format $X \rightarrow Y$.

This indicates a pattern that when the X itemset is observed, Y is also observed. To select interesting rules, various measures of significance can be used.

Support

Support, for an itemset, is defined as the proportion of transactions that contain the itemset. The $\{potatoes, burger\}$ itemset in the previous table has the following support as it occurs in 50% of transactions (2 out of 4 transactions) $supp(\{potatoes, burger\}) = 2/4 = 0.5$.

Intuitively, it indicates the share of transactions that support the pattern.

Confidence

Confidence of a rule indicates its accuracy. It is defined as

$$Conf(X \rightarrow Y) = supp(X \cup Y) / supp(X)$$

.

For example, the $\{onions, burger\} \rightarrow \{beer\}$ rule has the confidence $0.5/0.5 = 1.0$ in the previous table, which means that 100% of the times when *onions* and *burger* are bought together, *beer* is bought as well.

Apriori algorithm

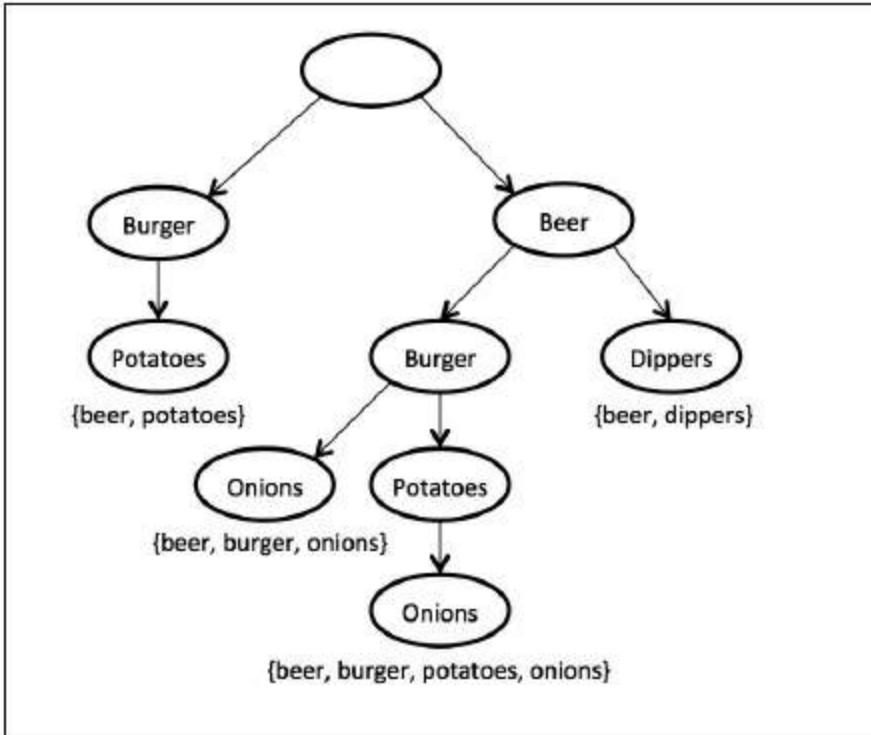
Apriori algorithm is a classic algorithm used for frequent pattern mining and association rule learning over transactional. By identifying the frequent individual items in a database and extending them to larger itemsets, Apriori can determine the association rules, which highlight general trends about a database.

Apriori algorithm constructs a set of itemsets, for example, $itemset1 = \{Item\ A, Item\ B\}$, and calculates support, which counts the number of occurrences in the database. Apriori then uses a bottom-up approach, where frequent itemsets are extended, one item at a time, and it works by eliminating the largest sets as candidates by first looking at the smaller sets and recognizing that a large set cannot be frequent unless all its subsets are. The algorithm terminates when no further successful extensions are found.

Although, Apriori algorithm is an important milestone in machine learning, it suffers from a number of inefficiencies and tradeoffs. In the following section, we'll look into a more recent FP-growth technique.

FP-growth algorithm

FP-growth, where **frequent pattern (FP)**, represents the transaction database as a prefix tree. First, the algorithm counts the occurrence of items in the dataset. In the second pass, it builds a prefix tree, an ordered tree data structure commonly used to store a string. An example of prefix tree based on the previous example is shown in the following diagram:



If many transactions share most frequent items, prefix tree provides high compression close to the tree root. Large itemsets are grown directly, instead of generating candidate items and testing them against the entire database. Growth starts at the bottom of the tree, by finding all the itemsets matching minimal support and confidence. Once the recursive process has completed, all large itemsets with minimum coverage have been found and association rule creation begins.

FP-growth algorithms have several advantages. First, it constructs an FP-tree, which encodes the original dataset in a substantially compact presentation. Second, it efficiently builds frequent itemsets, leveraging the **FP-tree structure** and **divide-and-conquer** strategy.

The supermarket dataset

The supermarket dataset, located in `datasets/chap5/supermarket.arff`, describes the shopping habits of supermarket customers. Most of the attributes stand for a particular item group, for example, diary foods, beef, potatoes; or department, for example, department 79, department 81, and so on. The following image shows an excerpt of the database, where the value is t if the customer had bought an item and missing otherwise. There is one instance per customer. The dataset contains no class attribute, as this is not required to learn association rules. A sample of data is shown in the following table:

	coffee	sauces-gravy-pickle	confectionary	puddings-deserts	dishcloths-scour	deod-disinfectant	frozen foods	razor blades	fuels-garden aids	spices	jams-spreads
1	1	1	1	0	1	0	1	1	0	0	0
0	0	1	0	0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	1	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	0	1	0	0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	1	1	1	1	1	0	1	0	0	0	1

Discover patterns

To discover shopping patterns, we will use the two algorithms that we have looked into before, Apriori and FP-growth.

Apriori

We will use the Apriori algorithm as implemented in Weka. It iteratively reduces the minimum support until it finds the required number of rules with the given minimum confidence:

```
import java.io.BufferedReader;
import java.io.FileReader;
import weka.core.Instances;
import weka.associations.Apriori;
```

First, we will load the supermarket dataset:

```
Instances data = new Instances(
new BufferedReader(
new FileReader("datasets/chap5/supermarket.arff")));
```

Next, we will initialize an Apriori instance and call the buildAssociations(Instances) function to start frequent pattern mining, as follows:

```
Apriori model = new Apriori();
model.buildAssociations(data);
```

Finally, we can output the discovered itemsets and rules, as shown in the following code:

```
System.out.println(model);
```

The output is as follows:

```
Apriori
=====
Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:
Size of set of large itemsets L(1): 44
Size of set of large itemsets L(2): 380
Size of set of large itemsets L(3): 910
Size of set of large itemsets L(4): 633
Size of set of large itemsets L(5): 105
Size of set of large itemsets L(6): 1
```

Best rules found:

```
1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and
cake=t 723      <conf: (0.92)> lift: (1.27) lev: (0.03) [155] conv: (3.35)
2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and
cake=t 696      <conf: (0.92)> lift: (1.27) lev: (0.03) [149] conv: (3.28)
3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and
cake=t 705      <conf: (0.92)> lift: (1.27) lev: (0.03) [150] conv: (3.27)
...
```

The algorithm outputs ten best rules according to confidence. Let's look the first rule and interpret the output, as follows:

```
biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t
723      <conf: (0.92)> lift: (1.27) lev: (0.03) [155] conv: (3.35)
```

It says that when biscuits, frozen foods, and fruits are bought together and the total purchase price is high, it is also very likely that bread and cake are purchased as well. The {biscuits, frozen foods, fruit, total high} itemset appears in 788 transactions, while the {bread, cake} itemset appears in 723 transactions. The confidence of this rule is 0.92, meaning that the rule holds true in 92% of transactions where the {biscuits, frozen foods, fruit, total high} itemset is present.

The output also reports additional measures such as lift, leverage, and conviction, which estimate the accuracy against our initial assumptions, for example, the 3.35 conviction value indicates that the rule would be incorrect 3.35 times as often if the association was purely a random chance. Lift measures the number of times X and Y occur together than expected if they were statistically independent ($\text{lift}=1$). The 2.16 lift in the $X \rightarrow Y$ rule means that the probability of X is 2.16 times greater than the probability of Y .

FP-growth

Now, let's try to get the same results with more efficient FP-growth algorithm. FP-growth is also implemented in the `weka.associations` package:

```
import weka.associations.FPGrowth;
```

The FP-growth is initialized similarly as we did earlier:

```
FPGrowth fpgModel = new FPGrowth();
fpgModel.buildAssociations(data);
System.out.println(fpgModel);
```

The output reveals that FP-growth discovered 16 rules:

```
FPGrowth found 16 rules (displaying top 10)

1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread
and cake=t]: 723  <conf: (0.92)> lift: (1.27) lev: (0.03) conv: (3.35)
2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread
and cake=t]: 696  <conf: (0.92)> lift: (1.27) lev: (0.03) conv: (3.28)
...
```

We can observe that FP-growth found the same set of rules as Apriori; however, the time required to process larger datasets can be significantly shorter.

Other applications in various areas

We looked into affinity analysis to demystify shopping behavior patterns in supermarkets. Although the roots of association rule learning are in analyzing point-of-sale transactions, they can be applied outside the retail industry to find relationships among other types of baskets. The notion of a basket can easily be extended to services and products, for example, to analyze items purchased using a credit card, such as rental cars and hotel rooms, and to analyze information on value-added services purchased by telecom customers (call waiting, call forwarding, DSL, speed call, and so on), which can help the operators determine the ways to improve their bundling of service packages.

Additionally, we will look into the following examples of potential cross-industry applications:

- Medical diagnosis
- Protein sequences
- Census data
- Customer relationship management
- IT Operations Analytics

Medical diagnosis

Applying association rules in medical diagnosis can be used to assist physicians while curing patients. The general problem of the induction of reliable diagnostic rules is hard as, theoretically, no induction process can guarantee the correctness of induced hypotheses by itself. Practically, diagnosis is not an easy process as it involves unreliable diagnosis tests and the presence of noise in training examples.

Nevertheless, association rules can be used to identify likely symptoms appearing together. A transaction, in this case, corresponds to a medical case, while symptoms correspond to items. When a patient is treated, a list of symptoms is recorded as one transaction.

Protein sequences

A lot of research has gone into understanding the composition and nature of proteins; yet many things remain to be understood satisfactorily. It is now generally believed that amino-acid sequences of proteins are not random.

With association rules, it is possible to identify associations between different amino acids that are present in a protein. A protein is a sequences made up of 20 types of amino acids. Each protein has a unique three-dimensional structure, which depends on the amino-acid sequence; slight change in the sequence may change the functioning of protein. To apply association rules, a protein corresponds to a transaction, while amino acids and their structure corespond to the items.

Such association rules are desirable for enhancing our understanding of protein composition and hold the potential to give clues regarding the global interactions amongst some particular sets of amino acids occurring in the proteins. Knowledge of these association rules or constraints is highly desirable for synthesis of artificial proteins.

Census data

Censuses make a huge variety of general statistical information about the society available to both researchers and general public. The information related to population and economic census can be forecasted in planning public services (education, health, transport, and funds) as well as in business (for setting up new factories, shopping malls, or banks and even marketing particular products).

To discover frequent patterns, each statistical area (for example, municipality, city, and neighborhood) corresponds to a transaction, and the collected indicators correspond to the items.

Customer relationship management

The **customer relationship management (CRM)**, as we briefly discussed in the previous chapters, is a rich source of data through which companies hope to identify the preference of different customer groups, products, and services in order to enhance the cohesion between their products and services and their customers.

Association rules can reinforce the knowledge management process and allow the marketing personnel to know their customers well in order to provide better quality services. For example, association rules can be applied to detect a change of customer behavior at different time snapshots from customer profiles and sales data. The basic idea is to discover changes from two datasets and generate rules from each dataset to carry out rule matching.

IT Operations Analytics

Based on records of a large number of transactions, association rule learning is well-suited to be applied to the data that is routinely collected in day-to-day IT operations, enabling IT Operations Analytics tools to detect frequent patterns and identify critical changes. IT specialists need to see the big picture and understand, for example, how a problem on a database could impact an application server.

For a specific day, IT operations may take in a variety of alerts, presenting them in a transactional database. Using an association rule learning algorithm, IT Operations Analytics tools can correlate and detect the frequent patterns of alerts appearing together. This can lead to a better understanding about how a component impacts another.

With identified alert patterns, it is possible to apply predictive analytics. For example, a particular database server hosts a web application and suddenly an alert about a database is triggered. By looking into frequent patterns identified by an association rule learning algorithm, this means that the IT staff needs to take action before the web application is impacted.

Association rule learning can also discover alert events originating from the same IT event. For example, every time a new user is added, six changes in the Windows operating system are detected. Next, in the **Application Portfolio Management (APM)**, IT may face multiple alerts, showing that the transactional time in a database is high. If all these issues originate from the same source (such as getting hundreds of alerts about changes that are all due to a Windows update), this frequent pattern mining can help to quickly cut through a number of alerts, allowing the IT operators to focus on truly critical changes.

Summary

In this chapter, you learned how to leverage association rule learning on transactional datasets to gain insight about frequent patterns. We performed an affinity analysis in Weka and learned that the hard work lies in the analysis of results—careful attention is required when interpreting rules, as association (that is, correlation) is not the same as causation.

In the next chapter, we'll look at how to take the problem of item recommendation to the next level using scalable machine learning library, Apache Mahout, which is able to handle big data.

Chapter 6. Recommendation Engine with Apache Mahout

Recommendation engines are probably one of the most applied data science approaches in startups today. There are two principal techniques for building a recommendation system: **content-based filtering** and **collaborative filtering**. The content-based algorithm uses the properties of the items to find items with similar properties. Collaborative filtering algorithms take user ratings or other user behavior and make recommendations based on what users with similar behavior liked or purchased.

This chapter will first explain the basic concepts required to understand recommendation engine principles and then demonstrate how to utilize Apache Mahout's implementation of various algorithms to quickly get a scalable recommendation engine. This chapter will cover the following topics:

- How to build a recommendation engine
- Getting Apache Mahout ready
- Content-based approach
- Collaborative filtering approach

By the end of the chapter, you will learn the kind of recommendation engine that is appropriate for our problem and how to quickly implement one.

Basic concepts

Recommendation engines aim to show user items of interest. What makes them different from search engines is that the relevant content usually appears on a website without requesting it and users don't have to build queries as recommendation engines observe user's actions and construct query for users without their knowledge.

Arguably, the most well-known example of recommendation engine is www.amazon.com, providing personalized recommendation in a number of ways. The following image shows an example of **Customers Who Bought This Item Also Bought**. As we will see later, this is an example of collaborative item-based recommendation, where items similar to a particular item are recommended:

The screenshot shows a grid of five book covers recommended by the system. Each book has a 'LOOK INSIDE!' button above it. Below each book cover, the title, author, price, and a star rating are displayed. A left arrow icon is visible on the far left of the grid.

Book Title	Author	Price	Rating
Data Mining: (The Morgan Kaufmann Series in...	Ian H. Witten	\$37.49	4.5★ 53 reviews
Machine Learning with Spark	Nick Pentreath	\$25.00	4.5★ 1 review
Learning Spark: Lightning-Fast Big...	Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia	\$25.61	4.5★ 17 reviews
Data Science for Business: What you need...	Foster Provost	\$25.61	4.5★ 104 reviews #1 Best Seller in Business Mathematics Skills
Practical Data Science Cookbook	Tony Ojeda	\$25.00	4.5★ 13 reviews

An example of recommendation engine from www.amazon.com.

In this section, we will introduce key concepts related to understanding and building recommendation engines.

Key concepts

Recommendation engine requires the following four inputs to make recommendations:

- Item information described with attributes
- User profile such as age range, gender, location, friends, and so on
- User interactions in form of ratings, browsing, tagging, comparing, saving, and emailing
- Context where the items will be displayed, for example, item category and item's geographical location

These inputs are then combined together by the recommendation engine to help us answer the following questions:

- Users who bought, watched, viewed, or bookmarked this item also bought, watched, viewed, or bookmarked...
- Items similar to this item...
- Other users you may know...
- Other users who are similar to you...

Now let's have a closer look at how this combining works.

User-based and item-based analysis

Building a recommendation engine depends on whether the engine searches for related items or users when trying to recommend a particular item.

In item-based analysis, the engine focuses on identifying items that are similar to a particular item; while in user-based analysis, users similar to the particular user are first determined. For example, users with the same profile information (age, gender, and so on) or actions history (bought, watched, viewed, and so on) are determined and then the same items are recommended to other similar users.

Both approaches require us to compute a similarity matrix, depending on whether we're analyzing item attributes or user actions. Let's take a deeper look at how this is done.

Approaches to calculate similarity

There are three fundamental approaches to calculate similarity, as follows:

- Collaborative filtering algorithms take user ratings or other user behavior and make recommendations based on what users with similar behavior liked or purchased
- The content-based algorithm uses the properties of the items to find items with similar properties
- A hybrid approach combining collaborative and content-based filtering

Let's take a look at each approach in detail.

Collaborative filtering

Collaborative filtering is based solely on user ratings or other user behavior, making recommendations based on what users with similar behavior liked or purchased.

A key advantage of collaborative filtering is that it does not rely on item content, and therefore, it is capable of accurately recommending complex items such as movies, without understanding the item itself. The underlying assumption is that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past.

A major disadvantage of this approach is the so-called cold start, meaning that if we want to build an accurate collaborative filtering system, the algorithm often needs a large amount of user ratings. This usually takes collaborative filtering out of the first version of the product and it is introduced later when a decent amount of data is collected.

Content-based filtering

Content-based filtering, on the other hand, is based on a description of items and a profile of user's preferences combined as follows. First, the items are described with attributes, and to find similar items, we measure the distance between items using a distance measure such as **cosine distance** or **Pearson coefficient** (more about distance measures is in [Chapter 1, Applied Machine Learning Quick Start](#)). Now, the user profile enters the equation. Given the feedback about the kind of items the user likes, we can introduce weights specifying the importance of a specific item attribute. For instance, Pandora Radio streaming service applies content-based

filtering to create stations using more than 400 attributes. A user initially picks a song with specific attributes, and by providing feedback, important song attributes are emphasized.

This approach initially needs very little information on user feedback, thus it effectively avoids the cold-start issue.

Hybrid approach

Now collaborative versus content-based to choose? Collaborative filtering is able to learn user preferences from user's actions regarding one content source and use them across other content types. Content-based filtering is limited to recommending content of the same type that the user is already using. This provides value to different use cases, for example, recommending news articles based on news browsing is useful, but it is much more useful if different sources such as books and movies can be recommended based on news browsing.

Collaborative filtering and content-based filtering are not mutually exclusive; they can be combined to be more effective in some cases. For example, **Netflix** uses collaborative filtering to analyze searching and watching patterns of similar users, as well as content-based filtering to offer movies that share characteristics with films that the user has rated highly.

There is a wide variety of hybridization techniques such as weighted, switching, mixed, feature combination, feature augmentation, cascade, meta-level, and so on. Recommendation systems are an active area in machine learning and data mining community with special tracks on data science conferences. A good overview of techniques is summarized in the paper *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions* by Adomavicius and Tuzhilin (2005), where the authors discuss different approaches and underlying algorithms and provide references to further papers. To get more technical and understand all the tiny details when a particular approach makes sense, you should look at the book edited by Ricci et al. (2010) *Recommender Systems Handbook* (1st ed.), Springer-Verlag New York.

Exploitation versus exploration

In recommendation system, there is always a tradeoff between recommending items that fall into the user's sweet spot based on what we already know about the user (**exploitation**) and recommending items that don't fall into user's sweet spot with the aim to expose user to some novelties (**exploration**). Recommendation systems with little exploration will only recommend items consistent with the previous user ratings, preventing showing items outside their current bubble. In practice, serendipity of getting new items out of user's sweet spot is often desirable, leading to pleasant surprise and potential discovery of new sweet spots.

In this section, we discussed the essential concepts required to start building recommendation engines. Now, let's take a look at how to actually build one with Apache Mahout.

Getting Apache Mahout

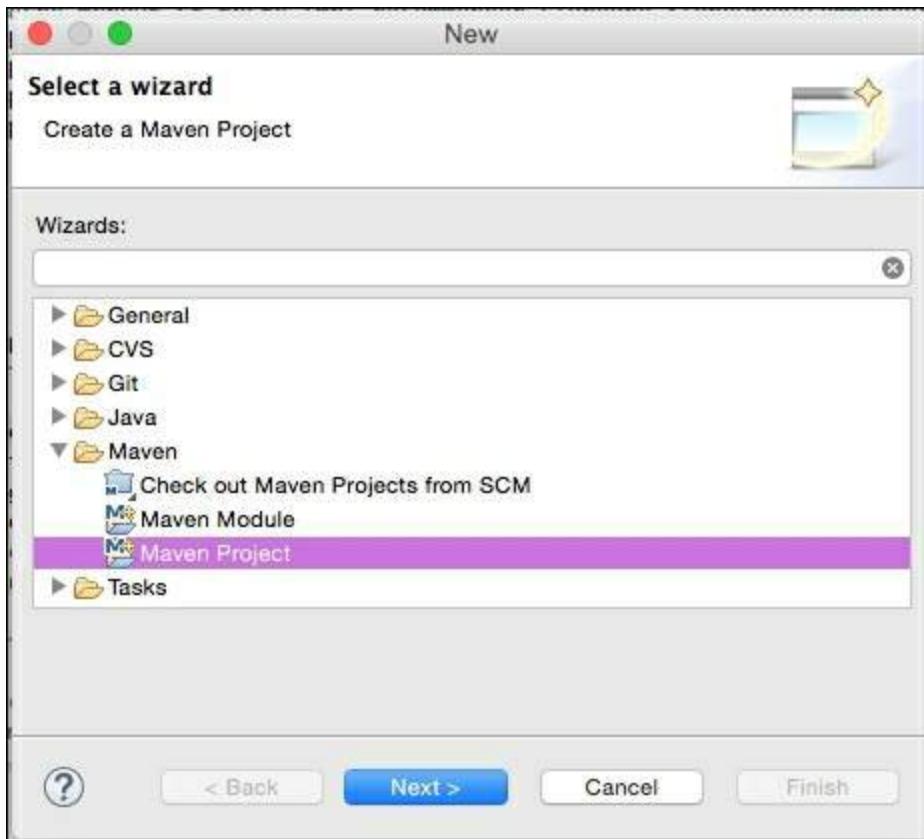
Mahout was introduced in [Chapter 2](#), *Java Tools and Libraries for Machine Learning*, as a scalable machine learning library. It provides a rich set of components with which you can construct a customized recommendation system from a selection of algorithms. The creators of Mahout say it is designed to be enterprise-ready; it's designed for performance, scalability, and flexibility.

Mahout can be configured to run in two flavors: with or without Hadoop for a single machine and distributed processing, correspondingly. We will focus on configuring Mahout without Hadoop. For more advanced configurations and further uses of Mahout, I would recommend two recent books: *Learning Apache Mahout* (Tiwary, 2015) and *Learning Apache Mahout Classification* (Gupta, 2015).

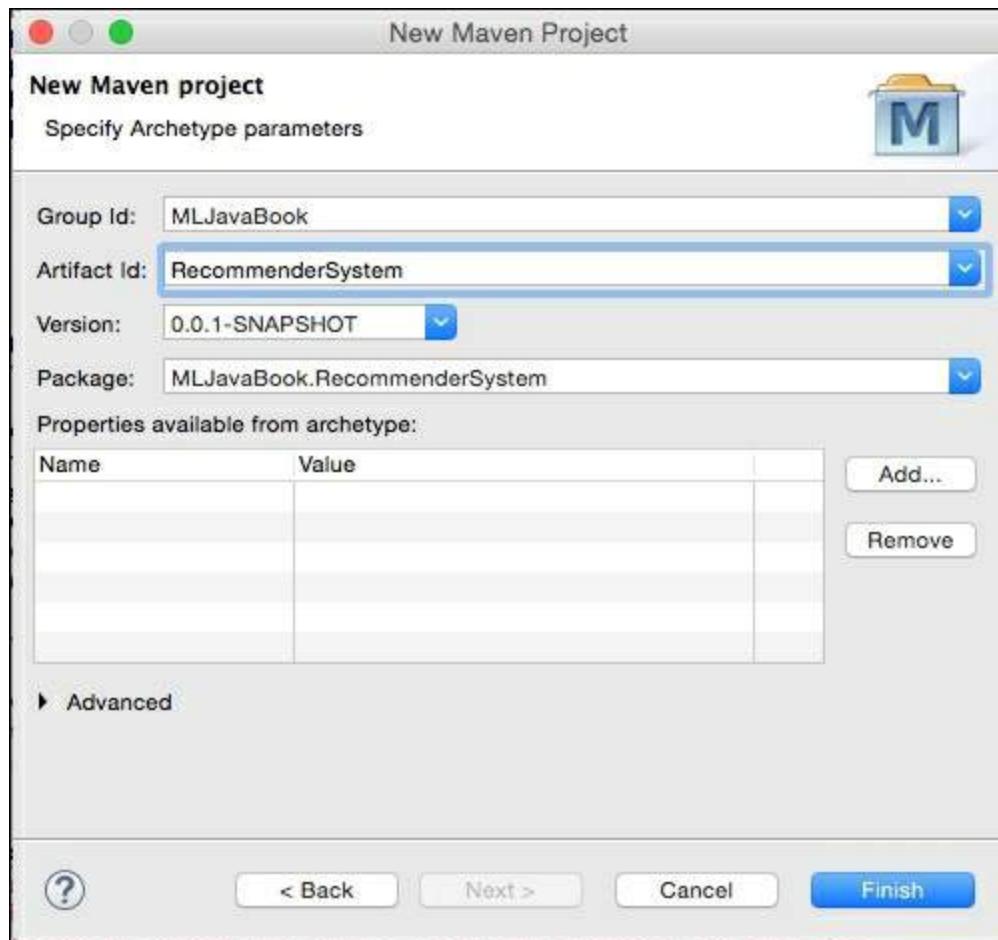
As Apache Mahout's build and release system is based on Maven, we will need to learn how to install it. We will look at the most convenient approach using Eclipse with Maven plugin.

Configuring Mahout in Eclipse with the Maven plugin

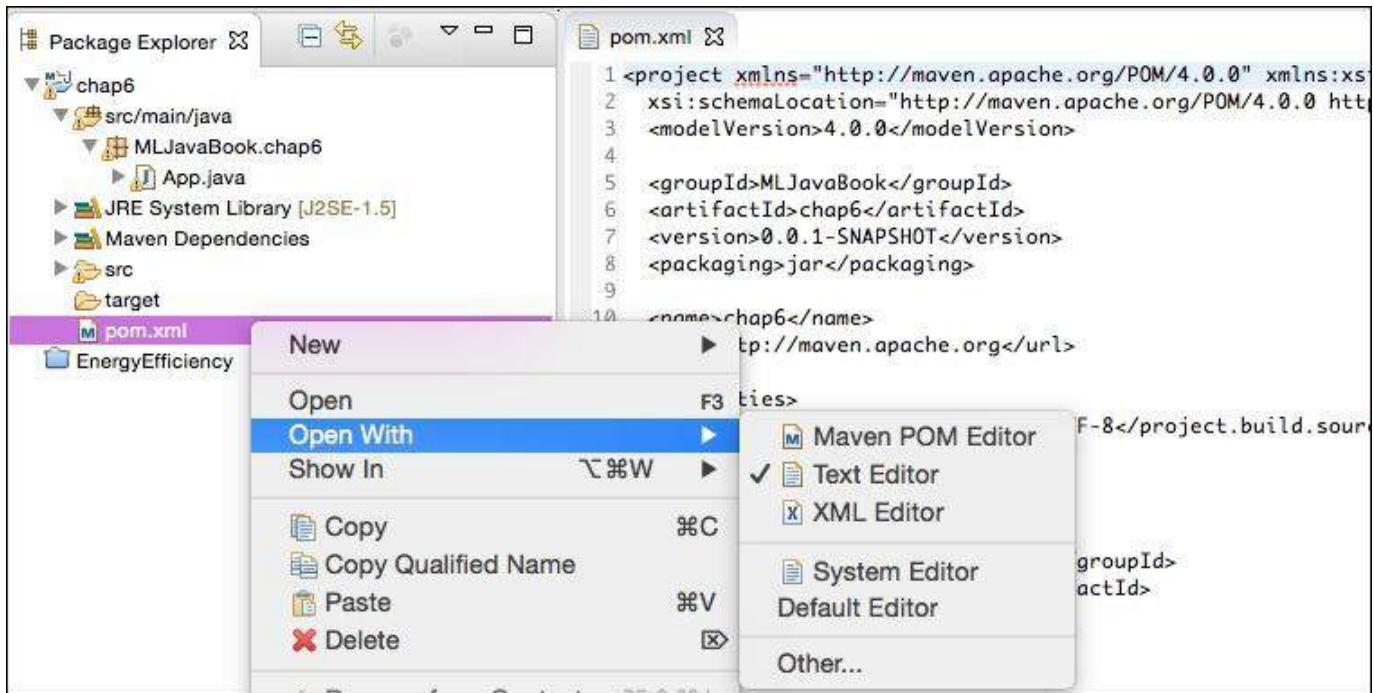
We will need a recent version of Eclipse, which can be downloaded from its home page. We use Eclipse Luna in this book. Open Eclipse and start a new **Maven Project** with default settings as shown in the following screenshot:



The **New Maven project** screen will appear as shown in the following image:



Now, we need to tell the project to add Mahout jar and its dependencies to the project. Locate the `pom.xml` file and open it with the text editor (left click on **Open With | Text Editor**), as shown in the following screenshot:



Locate the line starting with `<dependencies>` and add the following code in the next line:

```
<dependency>
  <groupId>org.apache.mahout</groupId>
  <artifactId>mahout-mr</artifactId>
  <version>0.10.0</version>
</dependency>
```

That's it, Mahout is added and we are ready to begin now.

Building a recommendation engine

To demonstrate both the content-based filtering and collaborative filtering approaches, we'll build a book-recommendation engine.

Book ratings dataset

In this chapter, we will work with book ratings dataset (Ziegler et al, 2005) collected in a four-week crawl. It contains data on 278,858 members of the **Book-Crossing** website and 1,157,112 ratings, both implicit and explicit, referring to 271,379 distinct ISBNs. User data is anonymized, but with demographic information. The dataset is available at:

<http://www2.informatik.uni-freiburg.de/~cziegler/BX/>.

The Book-Crossing dataset comprises three files described at their website as follows:

- **BX-Users:** This contains the users. Note that user IDs (User-ID) have been anonymized and mapped to integers. Demographic data is provided (Location and Age) if available. Otherwise, these fields contain NULL-values.
- **BX-Books:** Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (Book-Title, Book-Author, Year-Of-Publication, and Publisher), obtained from Amazon Web Services. Note that in case of several authors, only the first author is provided. URLs linking to cover images are also given, appearing in three different flavors (Image-URL-S, Image-URL-M, and Image-URL-L), that is, small, medium, and large. These URLs point to the Amazon website.
- **BX-Book-Ratings:** This contains the book rating information. Ratings (Book-Rating) are either explicit, expressed on a scale of 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

Loading the data

There are two approaches for loading the data according to where the data is stored: file or database. First, we will take a detailed look at how to load the data from the file, including how to deal with custom formats. At the end, we quickly take a look at how to load the data from a database.

Loading data from file

Loading data from file can be achieved with the `FileDataModel` class, expecting a comma-delimited file, where each line contains a `userID`, `itemID`, optional preference value, and optional `timestamp` in the same order, as follows:

```
userID, itemID[, preference[, timestamp]]
```

Optional preference accommodates applications with binary preference values, that is, user either expresses a preference for an item or not, without degree of preference, for example, with like/dislike.

A line that begins with hash, `#`, or an empty line will be ignored. It is also acceptable for the lines to contain additional fields, which will be ignored.

The `DataModel` class assumes the following types:

- `userID`, `itemID` can be parsed as `long`
- preference value can be parsed as `double`
- `timestamp` can be parsed as `long`

If you are able to provide the dataset in the preceding format, you can simply use the following line to load the data:

```
DataModel model = new FileDataModel(new File(path));
```

This class is not intended to be used for very large amounts of data, for example, tens of millions of rows. For that, a JDBC-backed `DataModel` and a database are more appropriate.

In real world, however, we cannot always ensure that the input data supplied to us contain only integer values for `userID` and `itemID`. For example, in our case, `itemID` correspond to ISBN book numbers uniquely identifying items, but these are not

integers and the `FileDataModel` default won't be suitable to process our data.

Now, let's consider how to deal with the case where our `itemID` is a string. We will define our custom data model by extending `FileDataModel` and overriding the long `readItemIDFromString(String)` method in order to read `itemID` as a string and convert the it into `long` and return a unique long value. To convert `String` to unique `long`, we'll extend another Mahout `AbstractIDMigrator` helper class, which is designed exactly for this task.

Now, let's first look at how `FileDataModel` is extended:

```
class StringItemIdFileDataModel extends FileDataModel {  
  
    //initialize migrator to covert String to unique long  
    public ItemMemIDMigrator memIdMigtr;  
  
    public StringItemIdFileDataModel(File dataFile, String regex) throws  
IOException {  
    super(dataFile, regex);  
}  
  
    @Override  
    protected long readItemIDFromString(String value) {  
  
        if (memIdMigtr == null) {  
            memIdMigtr = new ItemMemIDMigrator();  
        }  
  
        // convert to long  
        long retValue = memIdMigtr.toLongID(value);  
        //store it to cache  
        if (null == memIdMigtr.toStringID(retValue)) {  
            try {  
                memIdMigtr.singleInit(value);  
            } catch (TasteException e) {  
                e.printStackTrace();  
            }  
        }  
        return retValue;  
    }  
  
    // convert long back to String  
    String getItemIDAsString(long itemId) {  
        return memIdMigtr.toStringID(itemId);  
    }  
}
```

Other useful methods that can be overridden are as follows:

- `readUserIDFromString(String value)` if user IDs are not numeric
- `readTimestampFromString(String value)` to change how timestamp is parsed

Now, let's take a look how `AbstractIDMigrator` is extended:

```
class ItemMemIDMigrator extends AbstractIDMigrator {  
  
    private FastByIDMap<String> longToString;  
  
    public ItemMemIDMigrator() {  
        this.longToString = new FastByIDMap<String>(10000);  
    }  
  
    public void storeMapping(long longID, String stringID) {  
        longToString.put(longID, stringID);  
    }  
  
    public void singleInit(String stringID) throws TasteException {  
        storeMapping(toLongID(stringID), stringID);  
    }  
  
    public String toStringID(long longID) {  
        return longToString.get(longID);  
    }  
}
```

Now, we have everything in place and we can load our dataset with the following code:

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(  
    new File("datasets/chap6/BX-Book-Ratings.csv"), ";");  
System.out.println(  
    "Total items: " + model.getNumItems() +  
    "\nTotal users: " + model.getNumUsers());
```

This outputs the total number of users and items:

```
Total items: 340556  
Total users: 105283
```

We are ready to move on and start making recommendations.

Loading data from database

Alternately, we can also load the data from database using one of the JDBC data models. In this chapter, we will not dive into the detailed instructions about how to set up database, connection, and so on, but just give a sketch on how this can be done.

Database connectors have been moved to a separate package `mahout-integration`, hence we have to first add the package to our dependency list. Open the `pom.xml` file and add the following dependency:

```
<dependency>
  <groupId>org.apache.mahout</groupId>
  <artifactId>mahout-integration</artifactId>
  <version>0.7</version>
</dependency>
```

Consider that we want to connect a MySQL database. In this case, we will also need a package that handles database connections. Add the following to the `pom.xml` file:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.35</version>
</dependency>
```

Now, we have all the packages, so we can create a connection. First, let's initialize a `DataSource` class with connection details, as follows:

```
MysqlDataSource dbsource = new MysqlDataSource();
dbsource.setUser("user");
dbsource.setPassword("pass");
dbsource.setServerName("hostname.com");
dbsource.setDatabaseName("db");
```

Mahout integration implements `JDBCDataModel` to various databases that can be accessed via JDBC. By default, this class assumes that there is `DataSource` available under the JNDI name `jdbc/taste`, which gives access to a database with a `taste_preferences` table with the following schema:

```
CREATE TABLE taste_preferences (
  user_id BIGINT NOT NULL,
  item_id BIGINT NOT NULL,
  preference REAL NOT NULL,
  PRIMARY KEY (user_id, item_id)
)
```

```
CREATE INDEX taste_preferences_user_id_index ON taste_preferences
(user_id);
CREATE INDEX taste_preferences_item_id_index ON taste_preferences
(item_id);
```

A database-backed data model is initialized as follows. In addition to the DB connection object, we can also specify the custom table name and table column names, as follows:

```
DataModel dataModel = new MySQLJDBCDataModel(dbsource,
"taste_preferences",
"user_id", "item_id", "preference", "timestamp");
```

In-memory database

Last, but not least, the data model can be created on the fly and held in memory. A database can be created from an array of preferences holding user ratings for a set of items.

We can proceed as follows. First, we create a `FastByIDMap` hash map of preference arrays, `PreferenceArray`, which stores an array of preferences:

```
FastByIDMap<PreferenceArray> preferences = new FastByIDMap<PreferenceArray>();
```

Next, we can create a new preference array for a user that will hold their ratings. The array must be initialized with a size parameter that reserves that many slots in memory:

```
PreferenceArray prefsForUser1 =
new GenericUserPreferenceArray(10);
```

Next, we set user ID for current preference at position 0. This will actually set the user ID for all preferences:

```
prefsForUser1.setUserID(0, 1L);
```

Set item ID for current preference at position 0:

```
prefsForUser1.setItemID(0, 101L);
```

Set preference value for preference at 0:

```
prefsForUser1.setValue(0, 3.0f);
```

Continue for other item ratings:

```
prefsForUser1.setItemID (1, 102L);  
prefsForUser1.setValue (1, 4.5F);
```

Finally, add user preferences to the hash map:

```
preferences.put (1L, prefsForUser1); // use userID as the key
```

The preference hash map can be now used to initialize GenericDataModel:

```
DataModel dataModel = new GenericDataModel(preferences);
```

This code demonstrates how to add two preferences for a single user; while in practical application, you'll want to add multiple preferences for multiple users.

Collaborative filtering

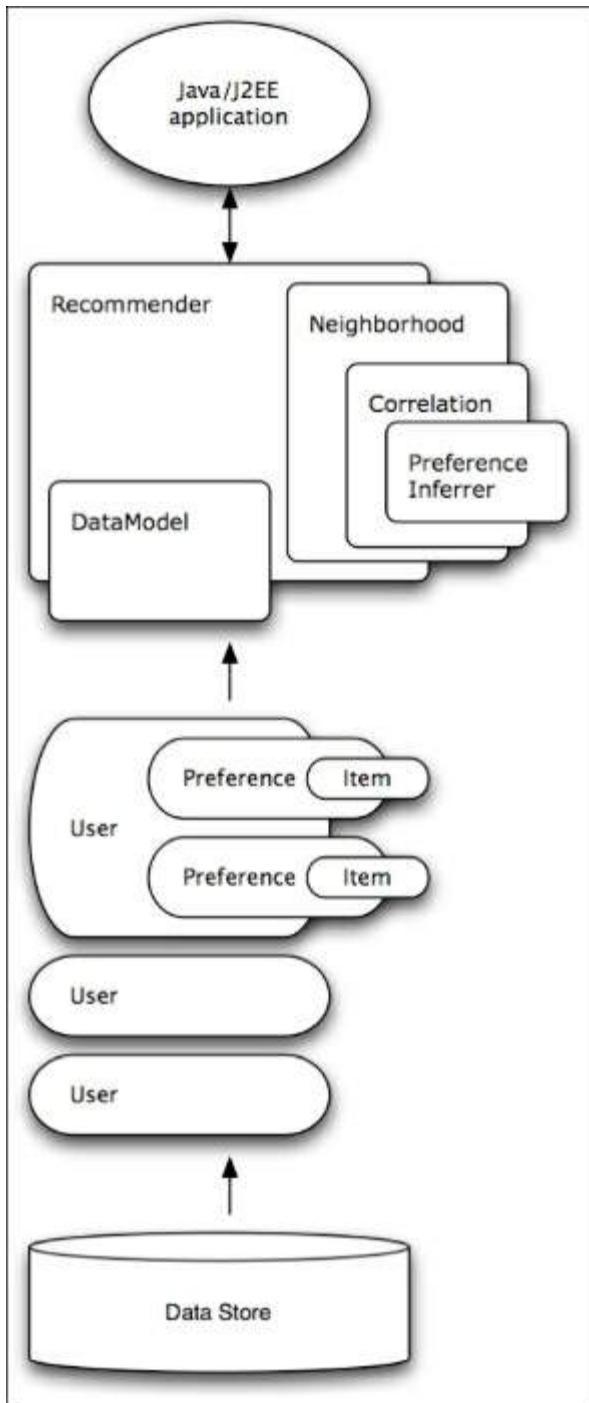
Recommendation engines in Mahout can be built with the `org.apache.mahout.cf.taste` package, which was formerly a separate project called `Taste` and has continued development in Mahout.

A Mahout-based collaborative filtering engine takes the users' preferences for items (tastes) and returns the estimated preferences for other items. For example, a site that sells books or CDs could easily use Mahout to figure out the CDs that a customer might be interested in listening to with the help of the previous purchase data.

Top-level packages define the Mahout interfaces to the following key abstractions:

- **DataModel**: This represents a repository of information about users and their preferences for items
- **UserSimilarity**: This defines a notion of similarity between two users
- **ItemSimilarity**: This defines a notion of similarity between two items
- **UserNeighborhood**: This computes neighborhood users for a given user
- **Recommender**: This recommends items for user

A general structure of the concepts is shown in the following diagram:



User-based filtering

The most basic user-based collaborative filtering can be implemented by initializing the previously described components as follows.

First, load the data model:

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(
    new File("/datasets/chap6/BX-Book-Ratings.csv", ";"));
```

Next, define how to calculate how the users are correlated, for example, using Pearson correlation:

```
UserSimilarity similarity =
    new PearsonCorrelationSimilarity(model);
```

Next, define how to tell which users are similar, that is, users that are close to each other according to their ratings:

```
UserNeighborhood neighborhood =
    new ThresholdUserNeighborhood(0.1, similarity, model);
```

Now, we can initialize a `GenericUserBasedRecommender` default engine with data model, neighborhood, and similar objects, as follows:

```
UserBasedRecommender recommender =
new GenericUserBasedRecommender(model, neighborhood, similarity);
```

That's it. Our first basic recommendation engine is ready. Let's discuss how to invoke recommendations. First, let's print the items that the user already rated along with ten recommendations for this user:

```
long userID = 80683;
int noItems = 10;

List<RecommendedItem> recommendations = recommender.recommend(
    userID, noItems);

System.out.println("Rated items by user:");
for(Preference preference : model.getPreferencesFromUser(userID)) {
    // convert long itemID back to ISBN
    String itemISBN = model.getItemIDAsString(
        preference.getItemId());
    System.out.println("Item: " + books.get(itemISBN) +
        " | Item id: " + itemISBN +
        " | Value: " + preference.getValue());
}

System.out.println("\nRecommended items:");
for (RecommendedItem item : recommendations) {
    String itemISBN = model.getItemIDAsString(item.getItemId());
    System.out.println("Item: " + books.get(itemISBN) +
        " | Item id: " + itemISBN +
```

```
    " | Value: " + item.getValue());
}
```

This outputs the following recommendations along with their scores:

Rated items:

```
Item: The Handmaid's Tale | Item id: 0395404258 | Value: 0.0
Item: Get Clark Smart : The Ultimate Guide for the Savvy Consumer |
Item id: 1563526298 | Value: 9.0
Item: Plum Island | Item id: 0446605409 | Value: 0.0
Item: Blessings | Item id: 0440206529 | Value: 0.0
Item: Edgar Cayce on the Akashic Records: The Book of Life | Item id:
0876044011 | Value: 0.0
Item: Winter Moon | Item id: 0345386108 | Value: 6.0
Item: Sarah Bishop | Item id: 059032120X | Value: 0.0
Item: Case of Lucy Bending | Item id: 0425060772 | Value: 0.0
Item: A Desert of Pure Feeling (Vintage Contemporaries) | Item id:
0679752714 | Value: 0.0
Item: White Abacus | Item id: 0380796155 | Value: 5.0
Item: The Land of Laughs : A Novel | Item id: 0312873115 | Value: 0.0
Item: Nobody's Son | Item id: 0152022597 | Value: 0.0
Item: Mirror Image | Item id: 0446353957 | Value: 0.0
Item: All I Really Need to Know | Item id: 080410526X | Value: 0.0
Item: Dreamcatcher | Item id: 0743211383 | Value: 7.0
Item: Perplexing Lateral Thinking Puzzles: Scholastic Edition | Item
id: 0806917695 | Value: 5.0
Item: Obsidian Butterfly | Item id: 0441007813 | Value: 0.0
```

Recommended items:

```
Item: Keeper of the Heart | Item id: 0380774933 | Value: 10.0
Item: Bleachers | Item id: 0385511612 | Value: 10.0
Item: Salem's Lot | Item id: 0451125452 | Value: 10.0
Item: The Girl Who Loved Tom Gordon | Item id: 0671042858 | Value: 10.0
Item: Mind Prey | Item id: 0425152898 | Value: 10.0
Item: It Came From The Far Side | Item id: 0836220730 | Value: 10.0
Item: Faith of the Fallen (Sword of Truth, Book 6) | Item id:
081257639X | Value: 10.0
Item: The Talisman | Item id: 0345444884 | Value: 9.86375
Item: Hamlet | Item id: 067172262X | Value: 9.708363
Item: Untamed | Item id: 0380769530 | Value: 9.708363
```

Item-based filtering

The `ItemSimilarity` is the most important point to discuss here. Item-based recommenders are useful as they can take advantage of something very fast: they base their computations on item similarity, not user similarity, and item similarity is relatively static. It can be precomputed, instead of recomputed in real time.

Thus, it's strongly recommended that you use `GenericItemSimilarity` with precomputed similarities if you're going to use this class. You can use `PearsonCorrelationSimilarity` too, which computes similarities in real time, but you will probably find this painfully slow for large amounts of data:

```
StringItemIdFileDataModel model = new StringItemIdFileDataModel(  
    new File("datasets/chap6/BX-Book-Ratings.csv"), ";");  
  
ItemSimilarity itemSimilarity = new  
PearsonCorrelationSimilarity(model);  
  
ItemBasedRecommender recommender = new  
GenericItemBasedRecommender(model, itemSimilarity);  
  
String itemISBN = "0395272238";  
long itemID = model.readItemIDFromString(itemISBN);  
int noItems = 10;  
List<RecommendedItem> recommendations =  
recommender.mostSimilarItems(itemID, noItems);  
  
System.out.println("Recommendations for item: "+books.get(itemISBN));  
  
System.out.println("\nMost similar items:");  
for (RecommendedItem item : recommendations) {  
    itemISBN = model.getItemIdAsString(item.getItemId());  
    System.out.println("Item: " + books.get(itemISBN) + " | Item id: " +  
itemISBN + " | Value: " + item.getValue());  
}
```

Recommendations for item: Close to the Bone

Most similar items:

```
Item: Private Screening | Item id: 0345311396 | Value: 1.0  
Item: Heartstone | Item id: 0553569783 | Value: 1.0  
Item: Clockers / Movie Tie In | Item id: 0380720817 | Value: 1.0  
Item: Rules of Prey | Item id: 0425121631 | Value: 1.0  
Item: The Next President | Item id: 0553576666 | Value: 1.0  
Item: Orchid Beach (Holly Barker Novels (Paperback)) | Item id:  
0061013412 | Value: 1.0  
Item: Winter Prey | Item id: 0425141233 | Value: 1.0  
Item: Night Prey | Item id: 0425146413 | Value: 1.0  
Item: Presumed Innocent | Item id: 0446359866 | Value: 1.0  
Item: Dirty Work (Stone Barrington Novels (Paperback)) | Item id:  
0451210158 | Value: 1.0
```

The resulting list returns a set of items similar to particular item that we selected.

Adding custom rules to recommendations

It often happens that some business rules require us to boost the score of the selected items. In the book dataset, for example, if a book is recent, we want to give it a higher score. That's possible using the `IDRescorer` interface implementing, as follows:

- `rescore(long, double)` that takes `itemId` and original score as an argument and returns a modified score
- `isFiltered(long)` that may return `true` to exclude a specific item from recommendation or `false` otherwise

Our example could be implemented as follows:

```
class MyRescorer implements IDRescorer {

    public double rescore(long itemId, double originalScore) {
        double newScore = originalScore;
        if(bookIsNew(itemId)){
            originalScore *= 1.3;
        }
        return newScore;
    }

    public boolean isFiltered(long arg0) {
        return false;
    }

}
```

An instance of `IDRescorer` is provided when invoking `recommender.recommend`:

```
IDRescorer rescorer = new MyRescorer();
List<RecommendedItem> recommendations =
recommender.recommend(userID, noItems, rescorer);
```

Evaluation

You might wonder how to make sure that the returned recommendations make any sense? The only way to be really sure about how effective recommendations are is to use A/B testing in a live system with real users. For example, the A group receives a random item as a recommendation, while the B group receives an item recommended by our engine.

As this is neither always possible nor practical, we can get an estimate with offline statistical evaluation. One way to proceed is to use the k-fold cross validation introduced in [Chapter 1](#), *Applied Machine Learning Quick Start*. We partition dataset into multiple sets, some are used to train our recommendation engine and the rest to test how well it recommends items to unknown users.

Mahout implements the `RecommenderEvaluator` class that splits a dataset in two parts. The first part, 90% by default, is used to produce recommendations, while the rest of the data is compared against estimated preference values to test the match. The class does not accept a `recommender` object directly, you need to build a class implementing the `RecommenderBuilder` interface instead, which builds a `recommender` object for a given `DataModel` object that is then used for testing. Let's take a look at how this is implemented.

First, we create a class that implements the `RecommenderBuilder` interface. We need to implement the `buildRecommender` method, which will return a `recommender`, as follows:

```
public class BookRecommender implements RecommenderBuilder {  
    public Recommender buildRecommender(DataModel dataModel) {  
        UserSimilarity similarity =  
            new PearsonCorrelationSimilarity(model);  
        UserNeighborhood neighborhood =  
            new ThresholdUserNeighborhood(0.1, similarity, model);  
        UserBasedRecommender recommender =  
            new GenericUserBasedRecommender(  
                model, neighborhood, similarity);  
        return recommender;  
    }  
}
```

Now that we have class that returns a `recommender` object, we can initialize a `RecommenderEvaluator` instance. Default implementation of this class is the `AverageAbsoluteDifferenceRecommenderEvaluator` class, which computes the average absolute difference between the predicted and actual ratings for users. The following code shows how to put the pieces together and run a hold-out test.

First, load a data model:

```
DataModel dataModel = new FileDataModel(  
    new File("/path/to/dataset.csv"));
```

Next, initialize an evaluator instance, as follows:

```
RecommenderEvaluator evaluator =  
    new AverageAbsoluteDifferenceRecommenderEvaluator();
```

Initialize the BookRecommender object, implementing the RecommenderBuilder interface:

```
RecommenderBuilder builder = new MyRecommenderBuilder();
```

Finally, call the evaluate() method, which accepts the following parameters:

- RecommenderBuilder: This is the object implementing RecommenderBuilder that can build recommender to test
- DataModelBuilder: DataModelBuilder to use, or if null, a default DataModel implementation will be used
- DataModel: This is the dataset that will be used for testing
- trainingPercentage: This indicates the percentage of each user's preferences to use to produce recommendations; the rest are compared to estimated preference values to evaluate the recommender performance
- evaluationPercentage: This is the percentage of users to be used in evaluation

The method is called as follows:

```
double result = evaluator.evaluate(builder, null, model, 0.9, 1.0);  
System.out.println(result);
```

The method returns a double, where 0 presents the best possible evaluation, meaning that the recommender perfectly matches user preferences. In general, lower the value, better the match.

Online learning engine

What about the online aspect? The above will work great for existing users; but what about new users which register in the service? For sure, we want to provide some reasonable recommendations for them as well. Creating a recommendation instance is expensive (it definitely takes longer than a usual network request), so we can't just create a new recommendation each time.

Luckily, Mahout has a possibility of adding temporary users to a data model. The general set up is as follows:

- Periodically recreate the whole recommendation using current data (for example, each day or hour, depending on how long it takes)
- When doing a recommendation, check whether the user exists in the system
- If yes, complete the recommendation as always
- If no, create a temporary user, fill in the preferences, and do the recommendation

The first part (periodically recreating the recommender) may be actually quite tricky if you are limited on memory: when creating the new recommender, you need to hold two copies of the data in memory (in order to still be able to serve requests from the old one). However, as this doesn't really have anything to do with recommendations, I won't go into details here.

As for the temporary users, we can wrap our data model with a `PlusAnonymousConcurrentUserDataModel` instance. This class allows us to obtain a temporary user ID; the ID must be later released so that it can be reused (there's a limited number of such IDs). After obtaining the ID, we have to fill in the preferences, and then, we can proceed with the recommendation as always:

```
class OnlineRecommendation{

    Recommender recommender;
    int concurrentUsers = 100;
    int noItems = 10;

    public OnlineRecommendation() throws IOException {

        DataModel model = new StringItemIdFileDataModel(
            new File "/chap6/BX-Book-Ratings.csv"), ";");
        PlusAnonymousConcurrentUserDataModel plusModel = new
        PlusAnonymousConcurrentUserDataModel(model, concurrentUsers);
        recommender = ...;

    }

    public List<RecommendedItem> recommend(long userId, PreferenceArray
preferences) {

        if(userExistsInDataModel(userId)) {
            return recommender.recommend(userId, noItems);
        }

        else{
```

```
    PlusAnonymousConcurrentUserDataModel plusModel =
        (PlusAnonymousConcurrentUserDataModel)
    recommender.getDataModel();

    // Take an available anonymous user from the poll
    Long anonymousUserID = plusModel.takeAvailableUser();

    // Set temporary preferences
    PreferenceArray tempPrefs = preferences;
    tempPrefs.setUserID(0, anonymousUserID);
    tempPrefs.setItemID(0, itemID);
    plusModel.setTempPrefs(tempPrefs, anonymousUserID);

    List<RecommendedItem> results =
recommender.recommend(anonymousUserID, noItems);

    // Release the user back to the poll
    plusModel.releaseUser(anonymousUserID);

    return results;
}

}
}
```

Content-based filtering

Content-based filtering is out of scope in the Mahout framework, mainly because it is up to you to decide how to define similar items. If we want to do a content-based item-item similarity, we need to implement our own `ItemSimilarity`. For instance, in our book's dataset, we might want to make up the following rule for book similarity:

- If genres are the same, add 0.15 to similarity
- If author is the same, add 0.50 to similarity

We could now implement our own similarity measure as follows:

```
class MyItemSimilarity implements ItemSimilarity {  
    ...  
    public double itemSimilarity(long itemID1, long itemID2) {  
        MyBook book1 = lookupMyBook (itemID1);  
        MyBook book2 = lookupMyBook (itemID2);  
        double similarity = 0.0;  
        if (book1.getGenre().equals(book2.getGenre()))  
            similarity += 0.15;  
        }  
        if (book1.getAuthor().equals(book2.getAuthor ())) {  
            similarity += 0.50;  
        }  
        return similarity;  
    }  
    ...  
}
```

We then use this `ItemSimilarity` instead of something like `LogLikelihoodSimilarity` or other implementations with a `GenericItemBasedRecommender`. That's about it. This is as far as we have to go to perform content-based recommendation in the Mahout framework.

What we saw here is one of the simplest forms of content-based recommendation. Another approach could be to create a content-based profile of users, based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually-rated content vectors.

Summary

In this chapter, you learned the basic concepts of recommendation engines, the difference between collaborative and content-based filtering, and how to use Apache Mahout, which is a great basis to create recommenders as it is very configurable and provides many extension points. We looked at how to pick the right configuration parameter values, set up rescoreing, and evaluate the recommendation results.

With this chapter, we completed data science techniques to analyze customer behavior that started with customer-relationship prediction in [Chapter 4, Customer Relationship Prediction with Ensembles](#), and continued with affinity analytics in [Chapter 5, Affinity Analysis](#). In the next chapter, we will move on to other topics, such as fraud and anomaly detection.

Chapter 7. Fraud and Anomaly Detection

Outlier detection is used to identify exceptions, rare events, or other anomalous situations. Such anomalies may be hard-to-find needles in a haystack, but their consequences may nonetheless be quite dramatic, for instance, credit card fraud detection, identifying network intrusion, faults in a manufacturing processes, clinical trials, voting activities, and criminal activities in e-commerce. Therefore, discovered anomalies represent high value when they are found or high costs if they are not found. Applying machine learning to outlier detection problems brings new insight and better detection of outlier events. Machine learning can take into account many disparate sources of data and find correlations that are too obscure for human analysis to identify.

Take the example of e-commerce fraud detection. With machine learning algorithm in place, the purchaser's online behavior, that is, website browsing history, becomes a part of the fraud detection algorithm rather than simply considering the history of purchases made by the cardholder. This involves analyzing a variety of data sources, but it is also a far more robust approach to e-commerce fraud detection.

In this chapter, we will cover the following topics:

- Problems and challenges
- Suspicious pattern detection
- Anomalous pattern detection
- Working with unbalanced datasets
- Anomaly detection in time series

Suspicious and anomalous behavior detection

The problem of learning patterns from sensor data arises in many applications, including e-commerce, smart environments, video surveillance, network analysis, human-robot interaction, ambient assisted living, and so on. We focus on detecting patterns that deviate from regular behaviors and might represent a security risk, health problem, or any other abnormal behavior contingency.

In other words, deviant behavior is a data pattern that either does not conform to the expected behavior (anomalous behavior) or matches a previously defined unwanted behavior (suspicious behavior). Deviant behavior patterns are also referred to as outliers, exceptions, peculiarities, surprise, misuse, and so on. Such patterns relatively occur infrequently; however, when they do occur, their consequences can be quite dramatic, and often negatively so. Typical examples include credit card fraud detection, cyber-intrusions, and industrial damage. In e-commerce, fraud is estimated to cost merchants more than \$200 billion a year; in healthcare, fraud is estimated to cost taxpayers \$60 billion a year; for banks, the cost is over \$12 billion.

Unknown-unknowns

When Donald Rumsfeld, US Secretary of Defense, had a news briefing on February 12, 2002, about the lack of evidence linking the government of Iraq to the supply of weapons of mass destruction to terrorist groups, it immediately became a subject of much commentary. Rumsfeld stated (DoD News, 2012):

"Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns—the ones we don't know we don't know. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones."

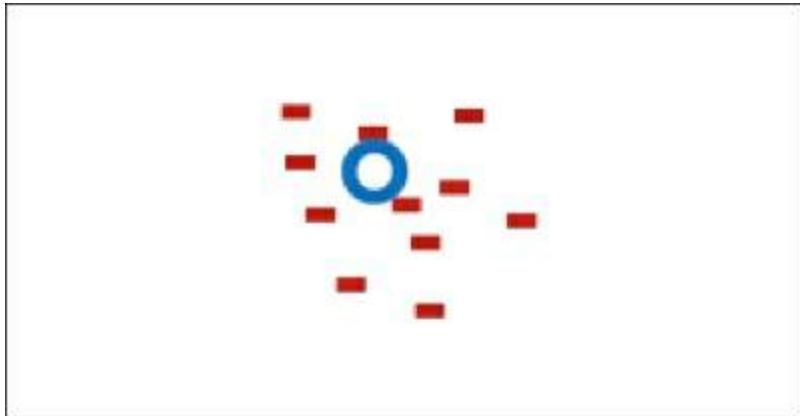
The statement might seem confusing at first, but the idea of unknown unknowns was well studied among scholars dealing with risk, NSA, and other intelligence agencies. What the statement basically says is the following:

- **Known-knowns:** These are well-known problems or issues we know how to recognize them and how deal with them
- **Known-unknowns:** These are expected or foreseeable problems, which can be reasonably anticipated, but have not occurred before
- **Unknown-unknowns:** These are unexpected and unforeseeable problems, which pose significant risk as they cannot be anticipated, based on previous experience

In the following sections, we will look into two fundamental approaches dealing with the first two types of knowns and unknowns: suspicious pattern detection dealing with known-knowns and anomalous pattern detection targeting known-unknowns.

Suspicious pattern detection

The first approach assumes a behavior library that encodes negative patterns shown as red minus signs in the following image, and thus recognizing that observed behavior corresponds to identifying a match in the library. If a new pattern (blue circle) can be matched against negative patterns, then it is considered suspicious:



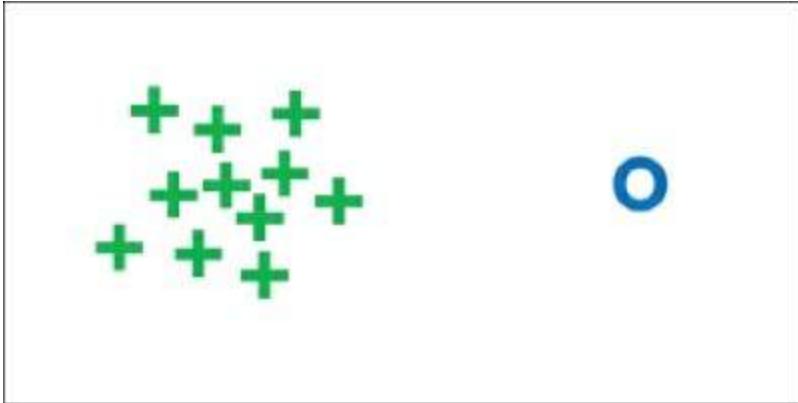
For example, when you visit a doctor, she inspects various health symptoms (body temperature, pain levels, affected areas, and so on) and matches the symptoms to a known disease. In machine learning terms, the doctor collects attributes and performs classifications.

An advantage of this approach is that we immediately know what is wrong; for example, assuming we know the disease, we can select appropriate treatment procedure.

A major disadvantage of this approach is that it can detect only suspicious patterns that are known in advance. If a pattern is not inserted into a negative pattern library, then we will not be able to recognize it. This approach is, therefore, appropriate for modeling known-knowns.

Anomalous pattern detection

The second approach uses the pattern library in an inverse fashion, meaning that the library encodes only positive patterns marked with green plus signs in the following image. When an observed behavior (blue circle) cannot be matched against the library, it is considered anomalous:



This approach requires us to model only what we have seen in the past, that is, normal patterns. If we return to the doctor example, the main reason we visited the doctor in the first place was because we did not feel fine. Our perceived state of feelings (for example, headache, sore skin) did not match our usual feelings, therefore, we decided to seek doctor. We don't know which disease caused this state nor do we know the treatment, but we were able to observe that it doesn't match the usual state.

A major advantage of this approach is that it does not require us to say anything about non-normal patterns; hence, it is appropriate for modeling known-unknowns and unknown-unknowns. On the other hand, it does not tell us what exactly is wrong.

Analysis types

Several approaches have been proposed to tackle the problem either way. We broadly classify anomalous and suspicious behavior detection in the following three categories: pattern analysis, transaction analysis, and plan recognition. In the following sections, we will quickly look into some real-life applications.

Pattern analysis

An active area of anomalous and suspicious behavior detection from patterns is based on visual modalities such as camera. Zhang et al (2007) proposed a system for a visual human motion analysis from a video sequence, which recognizes unusual behavior based on walking trajectories; Lin et al (2009) described a video surveillance system based on color features, distance features, and a count feature, where evolutionary techniques are used to measure observation similarity. The system tracks each person and classifies their behavior by analyzing their trajectory patterns. The system extracts a set of visual low-level features in different parts of the image, and performs a classification with SVMs to detect aggressive, cheerful, intoxicated, nervous, neutral, and tired behavior.

Transaction analysis

Transaction analysis assumes discrete states/transactions in contrast to continuous observations. A major research area is **Intrusion Detection (ID)** that aims at detecting attacks against information systems in general. There are two types of ID systems, signature-based and anomaly-based, that broadly follow the suspicious and anomalous pattern detection as described in the previous sections. A comprehensive review of ID approaches was published by Gyanchandani et al (2012).

Furthermore, applications in ambient-assisted living that are based on wearable sensors also fit to transaction analysis as sensing is typically event-based.

Lymberopoulos et al (2008) proposed a system for automatic extraction of the users' **spatio-temporal** patterns encoded as sensor activations from the sensor network deployed inside their home. The proposed method, based on location, time, and duration, was able to extract frequent patterns using the Apriori algorithm and encode the most frequent patterns in the form of a Markov chain. Another area of related work includes **Hidden Markov Models (HMMs)** (Rabiner, 1989) that are widely used in traditional activity recognition for modeling a sequence of actions, but these topics are already out of scope of this book.

Plan recognition

Plan recognition focuses on a mechanism for recognizing the unobservable state of an agent, given observations of its interaction with its environment (Avrahami-Zilberbrand, 2009). Most existing investigations assume discrete observations in the form of activities. To perform anomalous and suspicious behavior detection, plan recognition algorithms may use a hybrid approach, a symbolic plan recognizer is used to filter consistent hypotheses, passing them to an evaluation engine, which focuses on ranking.

These were advanced approaches applied to various real-life scenarios targeted at discovering anomalies. In the following sections, we'll dive into more basic approaches for suspicious and anomalous pattern detection.

Fraud detection of insurance claims

First, we'll take a look at suspicious behavior detection, where the goal is to learn known patterns of frauds, which correspond to modeling known-knowns.

Dataset

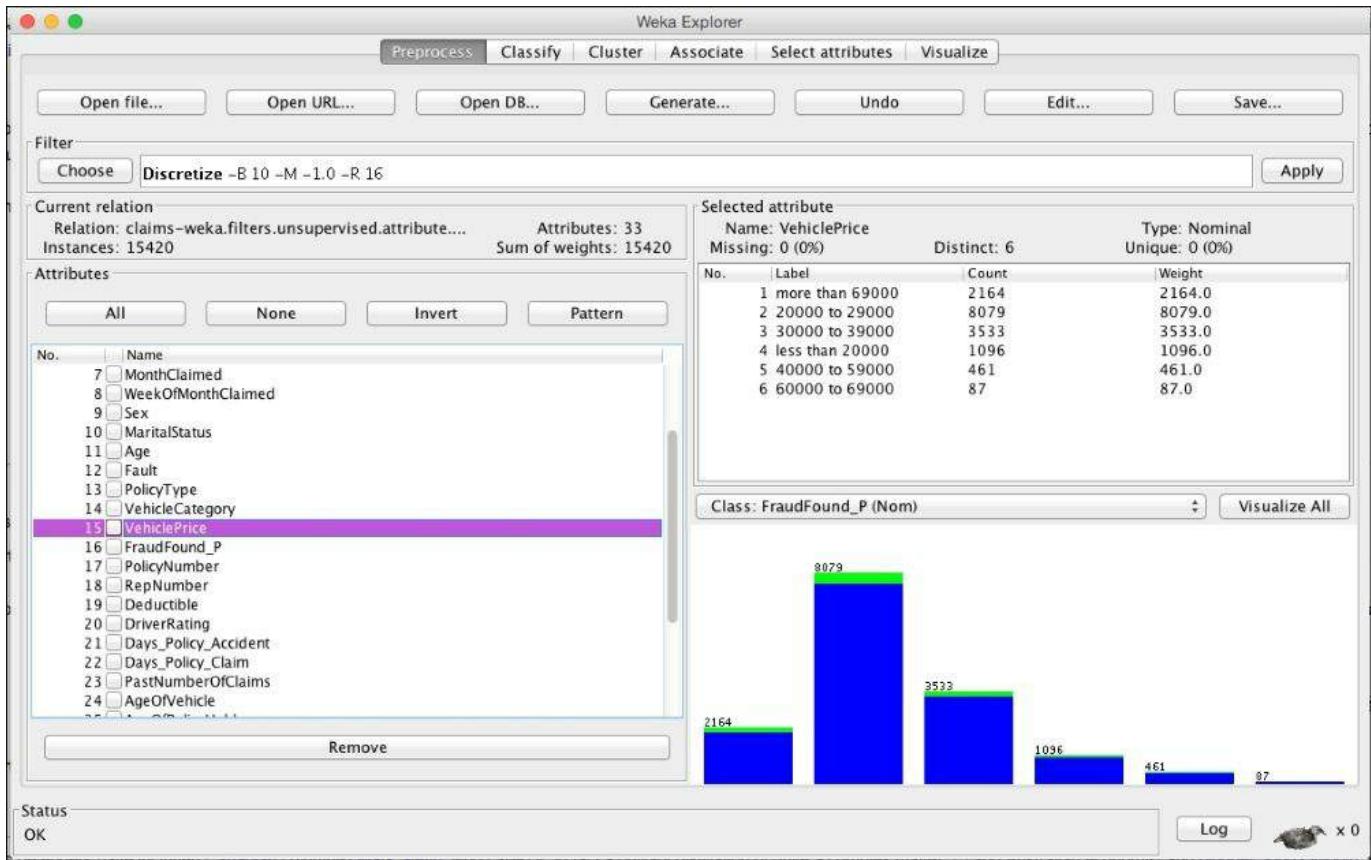
We'll work with a dataset describing insurance transactions publicly available at **Oracle Database Online Documentation** (2015), as follows:

http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/anomalies.htm

The dataset describes insurance vehicle incident claims for an undisclosed insurance company. It contains 15,430 claims; each claim comprises 33 attributes describing the following components:

- Customer demographic details (**Age**, **Sex**, **MartialStatus**, and so on)
- Purchased policy (**PolicyType**, **VehicleCategory**, number of supplements, agent type, and so on)
- Claim circumstances (day/month/week claimed, policy report filed, witness present, past days between incident-policy report, incident-claim, and so on)
- Other customer data (number of cars, previous claims, **DriverRating**, and so on)
- Fraud found (yes and no)

A sample of the database shown in the following screenshot depicts the data loaded into Weka:



Now the task is to create a model that will be able to identify suspicious claims in future. The challenging thing about this task is the fact that only 6% of claims are suspicious. If we create a dummy classifier saying no claim is suspicious, it will be accurate in 94% cases. Therefore, in this task, we will use different accuracy measures: precision and recall.

Recall the outcome table from [Chapter 1, Applied Machine Learning Quick Start](#), where there are four possible outcomes denoted as true positive, false positive, false negative, and true negative:

		Classified as	
		Fraud	No fraud
Actual	Fraud	TP—true positive	FN—false negative
	No fraud	FP—false positive	TN—true negative

Precision and recall are defined as follows:

- Precision is equal to the proportion of correctly raised alarms, as follows:

$$Pr = \frac{TP}{TP + FP}$$

- Recall is equal to the proportion of deviant signatures, which are correctly identified as such:

$$Re = \frac{TP}{TP + FN}$$

- With these measures, our dummy classifier scores $Pr = 0$ and $Re = 0$ as it never marks any instance as fraud ($TP = 0$). In practice, we want to compare classifiers by both numbers, hence we use *F-measure*. This is a de-facto measure that calculates a harmonic mean between precision and recall, as follows:

$$F\text{-measure} = \frac{2 * Pr * Re}{Pr + Re}$$

Now let's move on to designing a real classifier.

Modeling suspicious patterns

To design a classifier, we can follow the standard supervised learning steps as described in [Chapter 1, Applied Machine Learning Quick Start](#). In this recipe, we will include some additional steps to handle unbalanced dataset and evaluate classifiers based on precision and recall. The plan is as follows:

- Load the data in the `.csv` format
- Assign the class attribute
- Convert all the attributes from numeric to nominal in order to make sure there are no incorrectly loaded numerical values
- **Experiment 1:** Evaluate models with k-fold cross validation
- **Experiment 2:** Rebalance dataset to a more balanced class distribution and manually perform cross validation
- Compare classifiers by recall, precision, and f-measure

First, let's load the data using the `CSVLoader` class, as follows:

```
String filePath = "/Users/bostjan/Dropbox/ML Java  
Book/book/datasets/chap07/claims.csv";  
  
CSVLoader loader = new CSVLoader();  
loader.setFieldSeparator(",");  
loader.setSource(new File(filePath));  
Instances data = loader.getDataSet();
```

Next, we need to make sure all the attributes are nominal. During the data import, Weka applies some heuristics to guess the most probable attribute type, that is, numeric, nominal, string, or date. As heuristics cannot always guess the correct type, we can set types manually, as follows:

```
NumericToNominal toNominal = new NumericToNominal();  
toNominal.setInputFormat(data);  
data = Filter.useFilter(data, toNominal);
```

Before we continue, we need to specify the attribute that we will try to predict. We can achieve this by calling the `setClassIndex(int)` function:

```
int CLASS_INDEX = 15;  
data.setClassIndex(CLASS_INDEX);
```

Next, we need to remove an attribute describing the policy number as it has no

predictive value. We simply apply the `Remove` filter, as follows:

```
Remove remove = new Remove();
remove.setInputFormat(data);
remove.setOptions(new String[] {"-R", ""+POLICY_INDEX});
data = Filter.useFilter(data, remove);
```

Now we are ready to start modeling.

Vanilla approach

The vanilla approach is to directly apply the lesson as demonstrated in [Chapter 3, Basic Algorithms – Classification, Regression, Clustering](#), without any pre-processing and not taking into account dataset specifics. To demonstrate drawbacks of vanilla approach, we will simply build a model with default parameters and apply k-fold cross validation.

First, let's define some classifiers that we want to test:

```
ArrayList<Classifier>models = new ArrayList<Classifier>();
models.add(new J48());
models.add(new RandomForest());
models.add(new NaiveBayes());
models.add(new AdaBoostM1());
models.add(new Logistic());
```

Next, we create an `Evaluation` object and perform k-fold cross validation by calling the `crossValidate(Classifier, Instances, int, Random, String[])` method, outputting precision, recall, and fMeasure:

```
int FOLDS = 3;
Evaluation eval = new Evaluation(data);

for(Classifier model : models){
    eval.crossValidateModel(model, data, FOLDS,
    new Random(1), new String[] {});
    System.out.println(model.getClass().getName() + "\n"+
        "\tRecall: " + eval.recall(FRAUD) + "\n"+
        "\tPrecision: " + eval.precision(FRAUD) + "\n"+
        "\tF-measure: " + eval.fMeasure(FRAUD));
}
```

The evaluation outputs the following scores:

```
weka.classifiers.trees.J48
```

```

Recall: 0.03358613217768147
Precision: 0.9117647058823529
F-measure: 0.06478578892371996
...
weka.classifiers.functions.Logistic
Recall: 0.037486457204767065
Precision: 0.2521865889212828
F-measure: 0.06527070364082249

```

We can see the results are not very promising. Recall, that is, the share of discovered frauds among all frauds is only 1-3%, meaning that only 1-3/100 frauds are detected. On the other hand, precision, that is, the accuracy of alarms is 91%, meaning that in 9/10 cases, when a claim is marked as fraud, the model is correct.

Dataset rebalancing

As the number of negative examples, that is, frauds, is very small, compared to positive examples, the learning algorithms struggle with induction. We can help them by giving them a dataset, where the share of positive and negative examples is comparable. This can be achieved with dataset rebalancing.

Weka has a built-in filter, **Resample**, which produces a random subsample of a dataset using either sampling with replacement or without replacement. The filter can also bias distribution towards a uniform class distribution.

We will proceed by manually implementing k-fold cross validation. First, we will split the dataset into k equal folds. Fold k will be used for testing, while the other folds will be used for learning. To split dataset into folds, we'll use the `StratifiedRemoveFolds` filter, which maintains the class distribution within the folds, as follows:

```

StratifiedRemoveFolds kFold = new StratifiedRemoveFolds();
kFold.setInputFormat(data);

double measures[][] = new double[models.size()][3];

for(int k = 1; k <= FOLDS; k++) {

    // Split data to test and train folds
    kFold.setOptions(new String[]{
        "-N", "+FOLDS", "-F", "+k", "-S", "1"});
    Instances test = Filter.useFilter(data, kFold);

    kFold.setOptions(new String[] {

```

```

"-N", "+FOLDS, "-F", "+k, "-S", "1", "-V} );
// select inverse "-V"
Instances train = Filter.useFilter(data, kFold);

```

Next, we can rebalance train dataset, where the `-z` parameter specifies the percentage of dataset to be resampled, and `-B` bias the class distribution towards uniform distribution:

```

Resample resample = new Resample();
resample.setInputFormat(data);
resample.setOptions(new String[] {"-z", "100", "-B", "1"}); //with replacement
Instances balancedTrain = Filter.useFilter(train, resample);

```

Next, we can build classifiers and perform evaluation:

```

for(ListIterator<Classifier>it = models.listIterator(); it.hasNext();) {
    Classifier model = it.next();
    model.buildClassifier(balancedTrain);
    eval = new Evaluation(balancedTrain);
    eval.evaluateModel(model, test);

    // save results for average
    measures[it.previousIndex()][0] += eval.recall(FRAUD);
    measures[it.previousIndex()][1] += eval.precision(FRAUD);
    measures[it.previousIndex()][2] += eval.fMeasure(FRAUD);
}

```

Finally, we calculate the average and output the best model:

```

// calculate average
for(int i = 0; i < models.size(); i++) {
    measures[i][0] /= 1.0 * FOLDS;
    measures[i][1] /= 1.0 * FOLDS;
    measures[i][2] /= 1.0 * FOLDS;
}

// output results and select best model
Classifier bestModel = null; double bestScore = -1;
for(ListIterator<Classifier> it = models.listIterator(); it.hasNext();)
{
    Classifier model = it.next();
    double fMeasure = measures[it.previousIndex()][2];
    System.out.println(
        model.getClass().getName() + "\n"+
        "\tRecall: "+measures[it.previousIndex()][0] + "\n"+
        "\tPrecision: "+measures[it.previousIndex()][1] + "\n"+

```

```

"\tF-measure: "+fMeasure);
if(fMeasure > bestScore) {
    bestScore = fMeasure;
    bestModel = model;
}
System.out.println("Best model:"+bestModel.getClass().getName());

```

Now the performance of the models has significantly improved, as follows:

```

weka.classifiers.trees.J48
Recall: 0.44204845100610574
Precision: 0.14570766048577555
F-measure: 0.21912423640160392
...
weka.classifiers.functions.Logistic
Recall: 0.7670657247204478
Precision: 0.13507459756495374
F-measure: 0.22969038530557626
Best model: weka.classifiers.functions.Logistic

```

What we can see is that all the models have scored significantly better; for instance, the best model, Logistic Regression, correctly discovers 76% of frauds, while producing a reasonable amount of false alarms—only 13% of claims marked as fraud are indeed fraudulent. If an undetected fraud is significantly more expensive than investigation of false alarms, then it makes sense to deal with an increased number of false alarms.

The overall performance has most likely still some room for improvement; we could perform attribute selection and feature generation and apply more complex model learning that we discussed in [Chapter 3, Basic Algorithms – Classification, Regression, Clustering](#).

Anomaly detection in website traffic

In the second example, we'll focus on modeling the opposite of the previous example. Instead of discussing what typical fraud-less cases are, we'll discuss the normal expected behavior of the system. If something cannot be matched against our expected model, it will be considered anomalous.

Dataset

We'll work with a publicly available dataset released by Yahoo Labs that is useful for discussing how to detect anomalies in time series data. For Yahoo, the main use case is in detecting unusual traffic on Yahoo servers.

Even though Yahoo announced that their data is publicly available, you have to apply to use it, and it takes about 24 hours before the approval is granted. The dataset is available here:

<http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70>

The data set comprises real traffic to Yahoo services, along with some synthetic data. In total, the dataset contains 367 time series, each of which contain between 741 and 1680 observations, recorded at regular intervals. Each series is written in its own file, one observation per line. A series is accompanied by a second column indicator with a one if the observation was an anomaly, and zero otherwise. The anomalies in real data were determined by human judgment, while those in the synthetic data were generated algorithmically. A snippet of the synthetic times series data is shown in the following table:

timestamp	value	anomaly	change point	trend	noise	12 hour seasonality	daily seasonality	weekly seasonality
1422237600	4333.43	0	0	4599	1.81	-190.95	-128.86	52.44
1422241200	4316.14	0	0	4602	-14.65	-220.5	-105.21	54.51
1422244800	4403.20	0	0	4605	7.04	-190.95	-74.39	56.51
1422248400	4531.20	0	0	4608	13.52	-110.25	-38.51	58.43
1422252000	4967.50	1	0	4911	-3.77	-6.91	-2.33	60.27

Snippet of the synthetic time-series data

In the following section, we'll learn how to transform time series data to attribute presentation that allows us to apply machine learning algorithms.

Anomaly detection in time series data

Detecting anomalies in raw, streaming time series data requires some data transformations. The most obvious way is to select a time window and sample time series with fixed length. In the next step, we want to compare a new time series to our previously collected set to detect if something is out of the ordinary.

The comparison can be done with various techniques, as follows:

- Forecasting the most probable following value, as well as confidence intervals (for example, Holt-Winters exponential smoothing). If a new value is out of forecasted confidence interval, it is considered anomalous.
- Cross correlation compares new sample to library of positive samples, it looks for exact match. If the match is not found, it is marked as anomalous.
- Dynamic time wrapping is similar to cross correlation, but allows signal distortion in comparison.
- Discretizing signal to bands, where each band corresponds to a letter. For example, $A=[\min, \text{mean}/3]$, $B=[\text{mean}/3, \text{mean}^*2/3]$, and $C=[\text{mean}^*2/3, \max]$ transforms the signal to a sequence of letters such as $aAABAACAAABBA\dots$. This approach reduces the storage and allows us to apply text-mining algorithms that we will discuss in [Chapter 10, Text Mining with Mallet – Topic Modeling and Spam Detection](#).
- Distribution-based approach estimates distribution of values in a specific time window. When we observe a new sample, we can compare whether distribution matches to the previously observed one.

This list is, by no means, exhaustive. Different approaches are focused on detecting different anomalies (for example, in value, frequency, and distribution). We will focus on a version of distribution-based approaches.

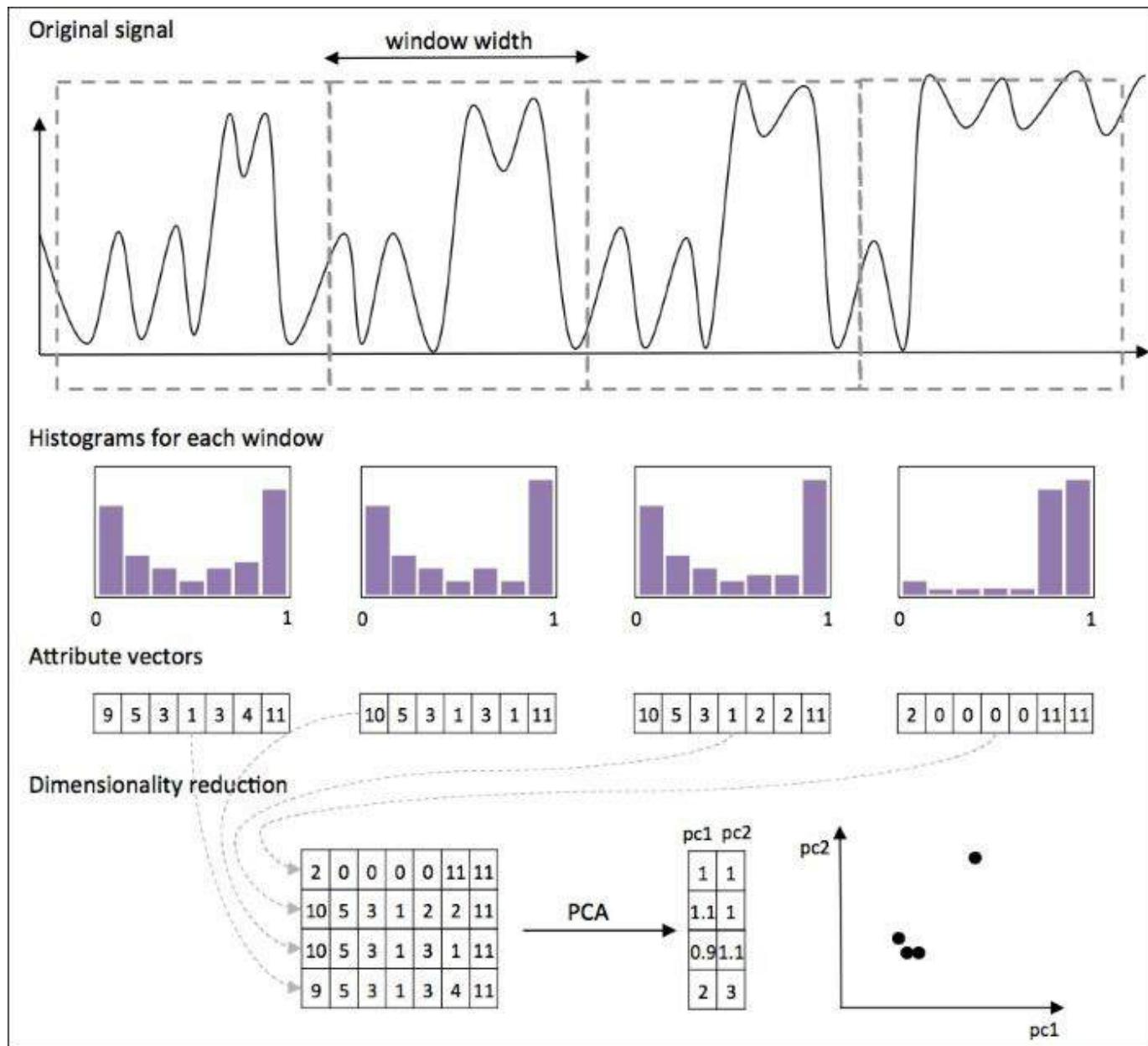
Histogram-based anomaly detection

In histogram-based anomaly detection, we split signals by some selected time window as shown in the following image.

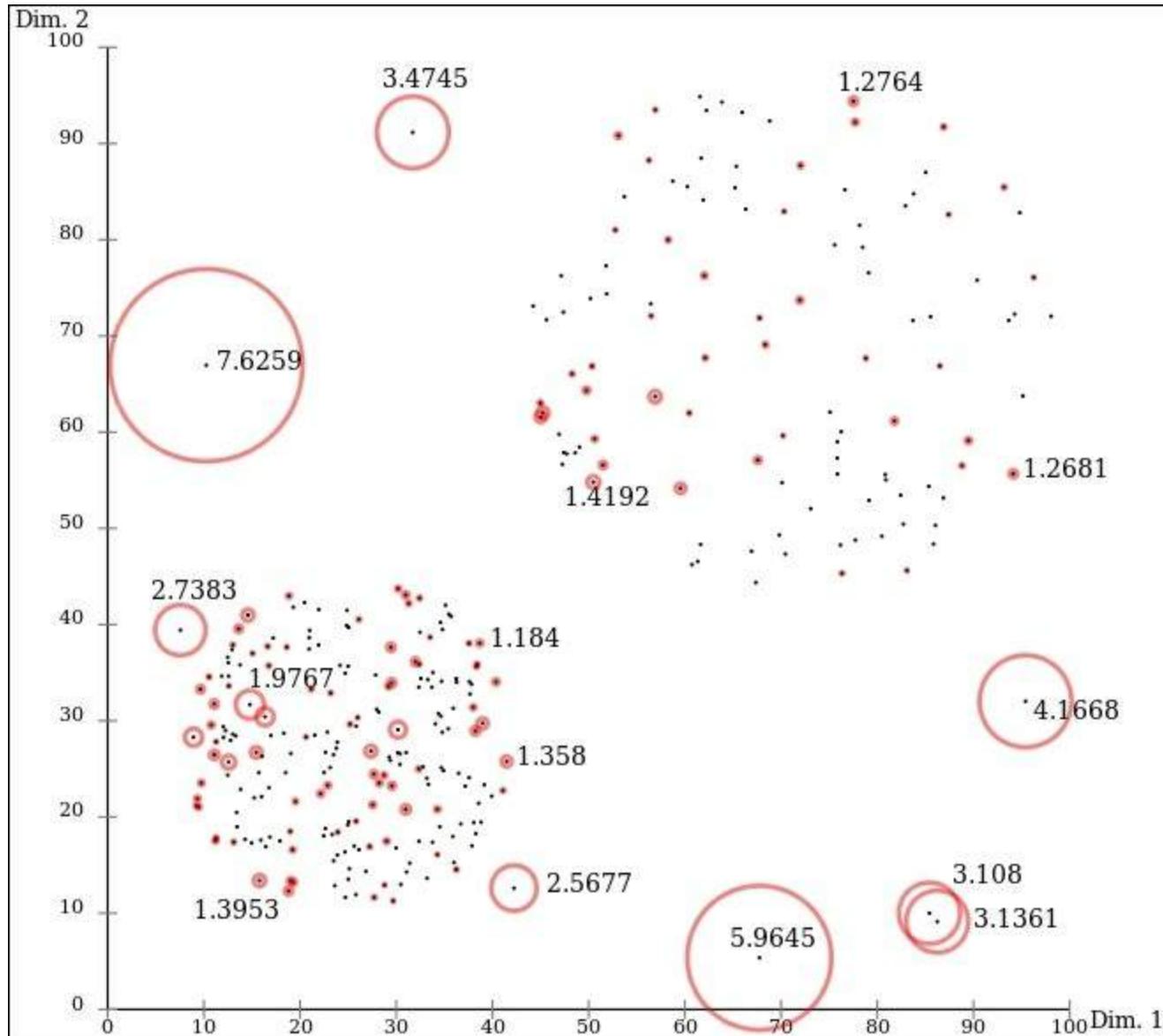
For each window, we calculate the histogram, that is, for a selected number of buckets, we count how many values fall into each bucket. The histogram captures basic distribution of values in a selected time window as shown at the center of the diagram.

Histograms can be then directly presented as instances, where each bin corresponds to an attribute. Further, we can reduce the number of attributes by applying a dimensionality-reduction technique such as **Principal Component Analysis (PCA)**, which allows us to visualize the reduced-dimension histograms in a plot as shown at the bottom-right of the diagram, where each dot corresponds to a histogram.

In our example, the idea is to observe website traffic for a couple of days and then to create histograms, for example, four-hour time windows to build a library of positive behavior. If a new time window histogram cannot be matched against positive library, we can mark it as an anomaly:



For comparing a new histogram to a set of existing histograms, we will use a density-based k-nearest neighbor algorithm, **Local Outlier Factor (LOF)** (Breunig et al, 2000). The algorithm is able to handle clusters with different densities as shown in the following image. For example, the upper-right cluster is large and widespread as compared to the bottom-left cluster, which is smaller and denser:



Let's get started!

Loading the data

In the first step, we'll need to load the data from text files to a Java object. The files are stored in a folder, each file contains one-time series with values per line. We'll load them into a list of `Double`:

```
String filePath = "chap07/ydata/A1Benchmark/real";
List<List<Double>> rawData = new ArrayList<List<Double>>();
```

We will need the `min` and `max` value for histogram normalization, so let's collect them in this data pass:

```
double max = Double.MIN_VALUE;
double min = Double.MAX_VALUE;

for(int i = 1; i<= 67; i++) {
    List<Double> sample = new ArrayList<Double>();
    BufferedReader reader = new BufferedReader(new
FileReader(filePath+i+".csv"));

    boolean isAnomaly = false;
    reader.readLine();
    while(reader.ready()) {
        String line[] = reader.readLine().split(",");
        double value = Double.parseDouble(line[1]);
        sample.add(value);

        max = Math.max(max, value);
        min = Double.min(min, value);

        if(line[2] == "1")
            isAnomaly = true;
    }
    System.out.println(isAnomaly);
    reader.close();

    rawData.add(sample);
}
```

The data is loaded, now let's move on to histograms.

Creating histograms

We will create a histogram for a selected time window with the `WIN_SIZE` width. The histogram will hold the `HIST_BINS` value buckets. The histograms consisting of list of doubles will be stored into an array list:

```

int WIN_SIZE = 500;
int HIST_BINS = 20;
int current = 0;

List<double[]> dataHist = new ArrayList<double[]>();
for(List<Double> sample : rawData){
    double[] histogram = new double[HIST_BINS];
    for(double value : sample){
        int bin = toBin(normalize(value, min, max), HIST_BINS);
        histogram[bin]++;
        current++;
        if(current == WIN_SIZE) {
            current = 0;
            dataHist.add(histogram);
            histogram = new double[HIST_BINS];
        }
    }
    dataHist.add(histogram);
}

```

Histograms are now completed. The last step is to transform them into Weka's `Instance` objects. Each histogram value will correspond to one Weka attribute, as follows:

```

ArrayList<Attribute> attributes = new ArrayList<Attribute>();
for(int i = 0; i < HIST_BINS; i++) {
    attributes.add(new Attribute("Hist-"+i));
}
Instances dataset = new Instances("My dataset", attributes,
dataHist.size());
for(double[] histogram: dataHist){
    dataset.add(new Instance(1.0, histogram));
}

```

The dataset is now loaded and ready to be plugged into an anomaly-detection algorithm.

Density based k-nearest neighbors

To demonstrate how LOF calculates scores, we'll first split the dataset into training and testing set using the `testCV(int, int)` function. The first parameter specifies the number of folds, while the second parameter specifies which fold to return.

```

// split data to train and test
Instances trainData = dataset.testCV(2, 0);
Instances testData = dataset.testCV(2, 1);

```

The LOF algorithm is not a part of the default Weka distribution, but it can be downloaded through Weka's package manager:

<http://weka.sourceforge.net/packageMetaData/localOutlierFactor/index.html>

LOF algorithm has two implemented interfaces: as an unsupervised filter that calculates LOF values (known-unknowns) and as a supervised k-nn classifier (known-knowns). In our case, we want to calculate the outlier-ness factor, therefore, we'll use the unsupervised filter interface:

```
import weka.filters.unsupervised.attribute.LOF;
```

The filter is initialized the same way as a usual filter. We can specify the `k` number of neighbors, for example, `k=3`, with `-min` and `-max` parameters. LOF allows us to specify two different `k` parameters, which are used internally as the upper and lower bound to find the minimal/maximal number `lof` values:

```
LOF lof = new LOF();
lof.setInputFormat(trainData);
lof.setOptions(new String[]{"-min", "3", "-max", "3"});
```

Next, we load training instances into the filter that will serve as a positive example library. After we complete the loading, we call the `batchFinished()` method to initialize internal calculations:

```
for(Instance inst : trainData) {
    lof.input(inst);
}
lof.batchFinished();
```

Finally, we can apply the filter to test data. Filter will process the instances and append an additional attribute at the end containing the LOF score. We can simply output the score on the console:

```
Instances testDataLofScore = Filter.useFilter(testData, lof);

for(Instance inst : testDataLofScore) {
    System.out.println(inst.value(inst.numAttributes()-1));
}
```

The LOF score of the first couple of test instances is as follows:

1.306740014927325
1.318239332210458

1.0294812291949587

1.1715039094530768

To understand the LOF values, we need some background on the LOF algorithm. It compares the density of an instance to the density of its nearest neighbors. The two scores are divided, producing the LOF score. The LOF score around 1 indicates that the density is approximately equal, while higher LOF values indicate that the density of the instance is substantially lower than the density of its neighbors. In such cases, the instance can be marked as anomalous.

Summary

In this chapter, we looked into detecting anomalous and suspicious patterns. We discussed the two fundamental approaches focusing on library encoding either positive or negative patterns. Next, we got our hands on two real-life datasets, where we discussed how to deal with unbalanced class distribution and perform anomaly detection in time series data.

In the next chapter, we'll dive deeper into patterns and more advanced approaches to build pattern-based classifier, discussing how to automatically assign labels to images with deep learning.

Chapter 8. Image Recognition with DeepLearning4j

Images have become ubiquitous in web services, social networks, and web stores. In contrast to humans, computers have great difficulty in understanding what is in the image and what does it represent. In this chapter, we'll first look at the challenges required to teach computers how to understand images, and then focus on an approach based on deep learning. We'll look at a high-level theory required to configure a deep learning model and discuss how to implement a model that is able to classify images using a Java library, DeepLearning4j.

This chapter will cover the following topics:

- Introducing image recognition
- Discussing deep learning fundamentals
- Building an image recognition model

Introducing image recognition

A typical goal of image recognition is to detect and identify an object in a digital image. Image recognition is applied in factory automation to monitor product quality; surveillance systems to identify potentially risky activities, such as moving persons or vehicles; security applications to provide biometric identification through fingerprints, iris, or facial features; autonomous vehicles to reconstruct conditions on the road and environment and so on.

Digital images are not presented in a structured way with attribute-based descriptions; instead, they are encoded as the amount of color in different channels, for instance, black-white and red-green-blue channels. The learning goal is to then identify patterns associated with a particular object. The traditional approach for image recognition consists of transforming an image into different forms, for instance, identify object corners, edges, same-color blobs, and basic shapes. Such patterns are then used to train a learner to distinguish between objects. Some notable examples of traditional algorithms are:

- Edge detection finds boundaries of objects within an image
- Corner detection identifies intersections of two edges or other interesting points, such as line endings, curvature maxima/minima, and so on
- Blob detection identifies regions that differ in a property, such as brightness or color, compared to its surrounding regions
- Ridge detection identifies additional interesting points at the image using smooth functions
- **Scale Invariant Feature Transform (SIFT)** is a robust algorithm that can match objects even if their scale or orientation differs from the representative samples in database
- Hough transform identifies particular patterns in the image

A more recent approach is based on deep learning. Deep learning is a form of neural network, which mimics how the brain processes information. The main advantage of deep learning is that it is possible to design neural networks that can automatically extract relevant patterns, which in turn, can be used to train a learner. With recent advances in neural networks, image recognition accuracy was significantly boosted. For instance, the **ImageNet** challenge (ImageNet, 2016), where competitors are provided more than 1.2 million images from 1,000 different object categories, reports that the error rate of the best algorithm was reduced from 28% in 2010, using SVM,

to only 7% in 2014, using deep neural network.

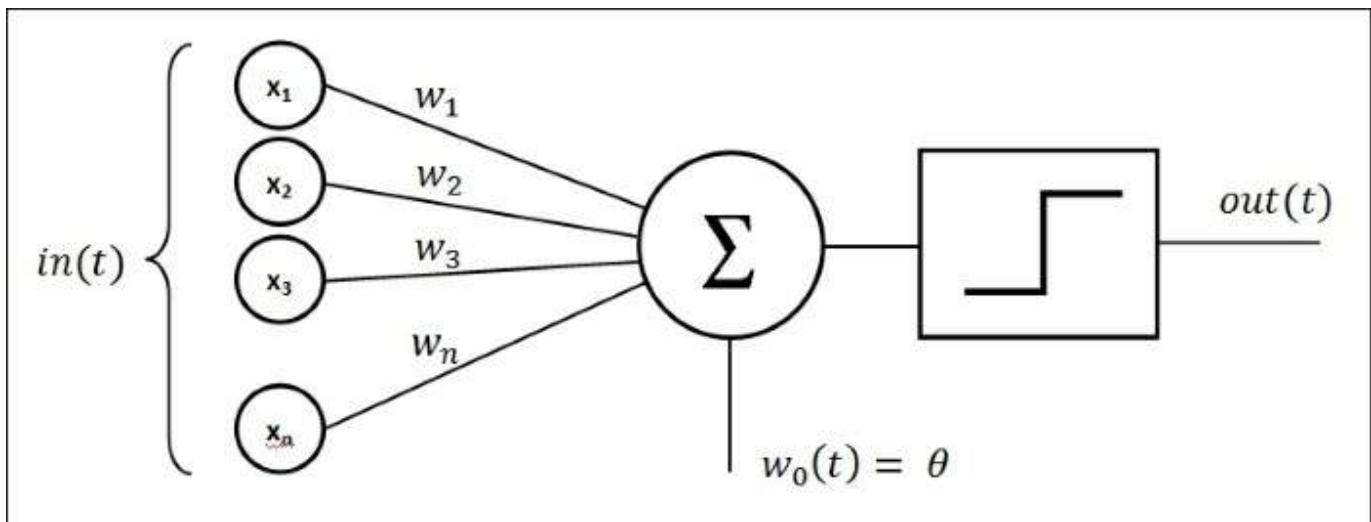
In this chapter, we'll take a quick look at neural networks, starting from the basic building block—**perceptron**—and gradually introducing more complex structures.

Neural networks

The first neural networks, introduced in the sixties, are inspired by biological neural networks. Recent advances in neural networks proved that deep neural networks fit very well in pattern recognition tasks, as they are able to automatically extract interesting features and learn the underlying presentation. In this section, we'll refresh the fundamental structures and components from a single perceptron to deep networks.

Perceptron

Perceptron is a basic neural network building block and one of the earliest supervised algorithms. It is defined as a sum of features, multiplied by corresponding weights and a bias. The function that sums all of this together is called **sum transfer function** and it is fed into an **activation function**. If the binary step activation function reaches a threshold, the output is 1 , otherwise 0 , which gives us a binary classifier. A schematic illustration is shown in the following diagram:



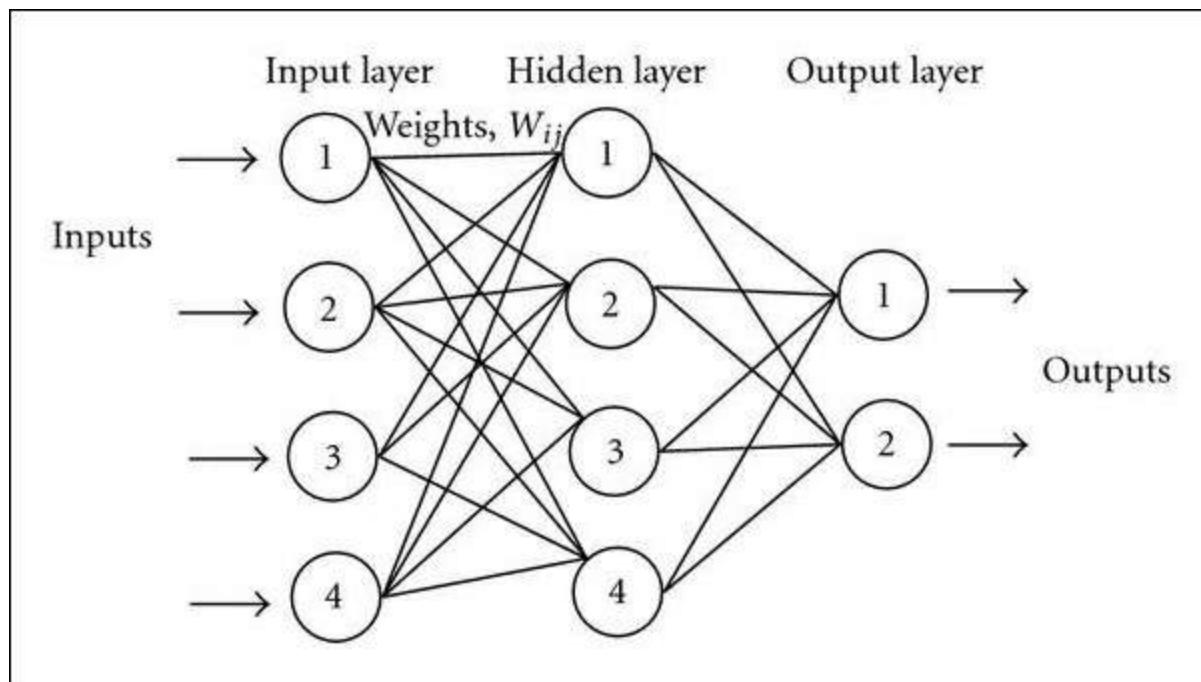
Training perceptrons involves a fairly simple learning algorithm that calculates the errors between the calculated output values and correct training output values, and uses this to create an adjustment to the weights; thus implementing a form of gradient descent. This algorithm is usually called the **delta rule**.

Single-layer perceptron is not very advanced, and nonlinearly separable functions, such as XOR, cannot be modeled using it. To address this issue, a structure with

multiple perceptrons was introduced, called **multilayer perceptron**, also known as **feedforward neural network**.

Feedforward neural networks

A feedforward neural network is an artificial neural network that consists of several perceptrons, which are organized by layers, as shown in the following diagram: input layer, output layer, and one or more hidden layers. Each layer perceptron, also known as neuron, has direct connections to the perceptrons in the next layer; whereas, connections between two neurons carry a weight similar to the perceptron weights. The diagram shows a network with a four-unit **Input layer**, corresponding to the size of feature vector of length 4, a four-unit **Hidden layer**, and a two-unit **Output layer**, where each unit corresponds to one class value:



The most popular approach to train multilayer networks is backpropagation. In backpropagation, the calculated output values are compared with the correct values in the same way as in delta rule. The error is then fed back through the network by various techniques, adjusting the weights of each connection in order to reduce the value of the error. The process is repeated for sufficiently large number of training cycles, until the error is under a certain threshold.

Feedforward neural network can have more than one hidden layer; whereas, each

additional hidden layer builds a new abstraction atop the previous layers. This often leads to more accurate models; however, increasing the number of hidden layers leads to the following two known issues:

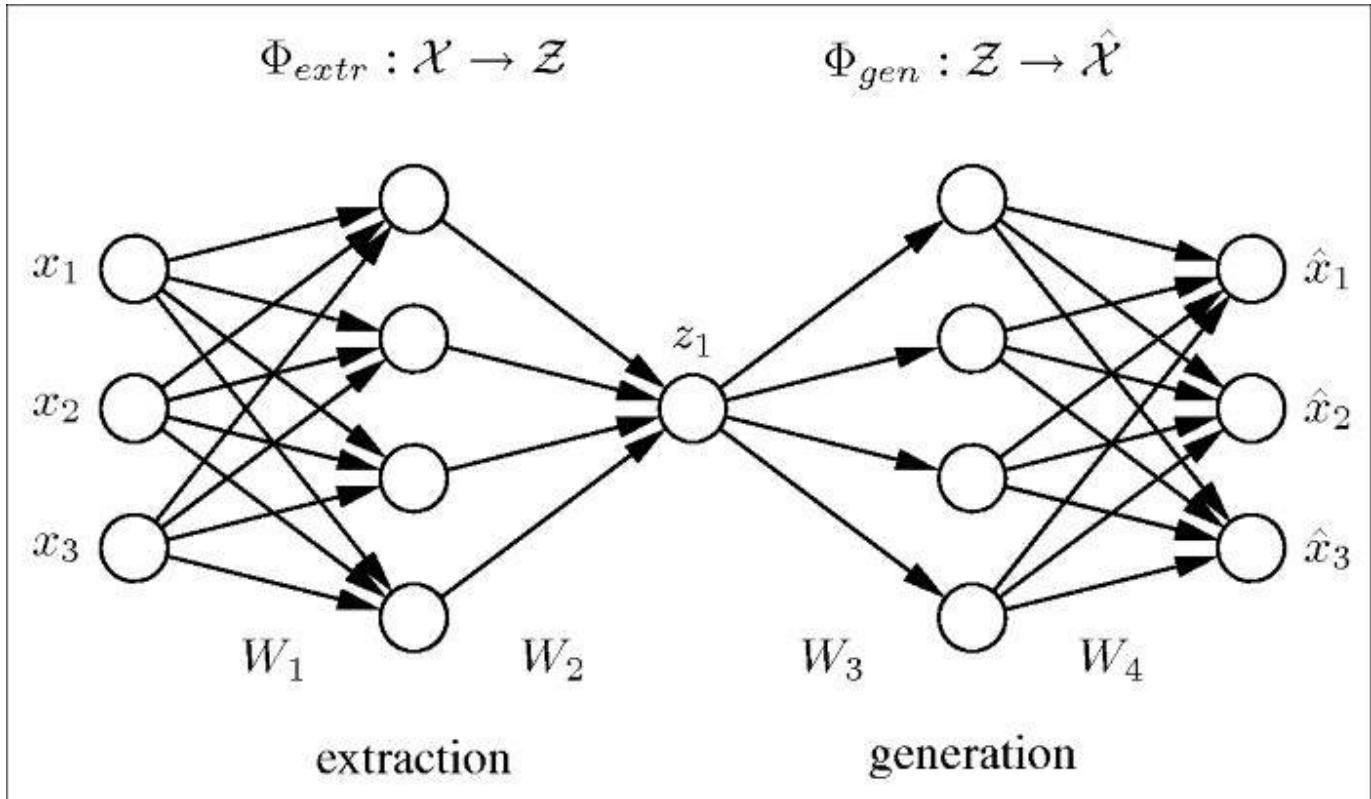
- **Vanishing gradients problem:** With more hidden layers, the training with backpropagation becomes less and less useful for passing information to the front layers, causing these layers to train very slowly
- **Overfitting:** The model fits the training data too well and performs poorly on real examples

Let's look at some other networks structures that address these issues.

Autoencoder

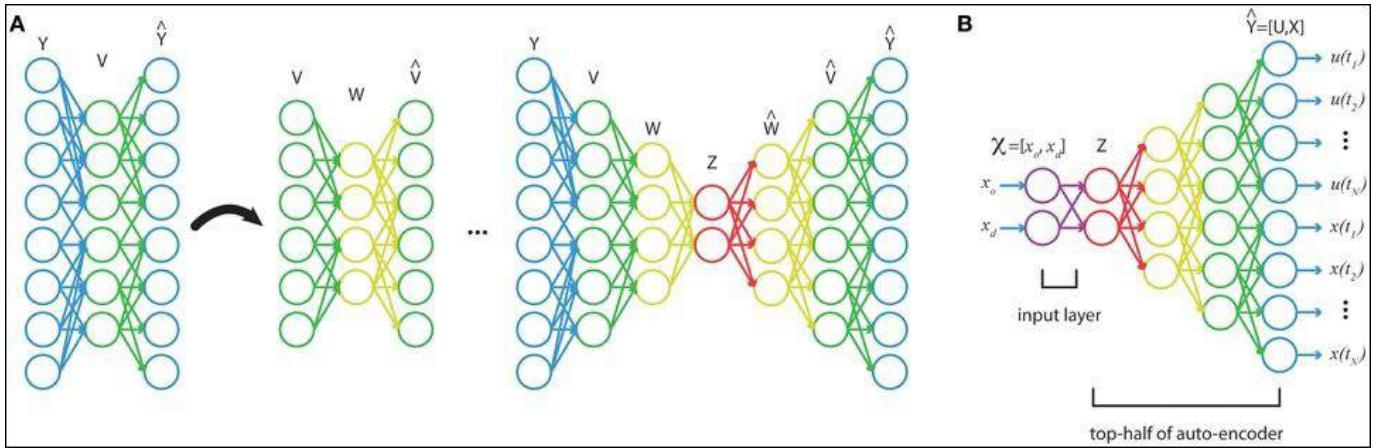
Autoencoder is a feedforward neural network that aims to learn how to compress the original dataset. Therefore, instead of mapping features to input layer and labels to output layer, we will map the features to both input and output layers. The number of units in hidden layers is usually different from the number of units in input layers, which forces the network to either expand or reduce the number of original features. This way the network will learn the important features, while effectively applying dimensionality reduction.

An example network is shown below. The three-unit input layer is first expanded into a four-unit layer and then compressed into a single-unit layer. The other side of the network restores the single-layer unit back to the four-unit layer, and then to the original three-input layer:



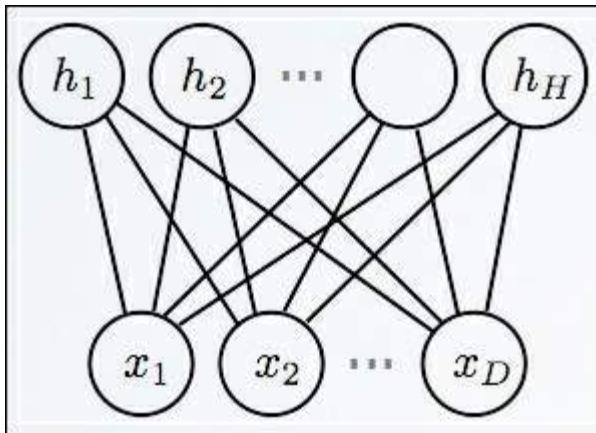
Once the network is trained, we can take the left-hand side to extract image features as we would with traditional image processing.

The autoencoders can be also combined into **stacked autoencoders**, as shown in the following image. First, we will discuss the hidden layer in a basic autoencoder, as described previously. Then we will take the learned hidden layer (green circles) and repeat the procedure, which in effect, learns a more abstract presentation. We can repeat the procedure multiple times, transforming the original features into increasingly reduced dimensions. At the end, we will take all the hidden layers and stack them into a regular feedforward network, as shown at the top-right part of the diagram:



Restricted Boltzmann machine

Restricted Boltzman machine is an undirected neural network, also denoted as **Generative Stochastic Networks (GSNs)**, and can learn probability distribution over its set of inputs. As the name suggests, they originate from Boltzman machine, a recurrent neural network introduced in the eighties. Restricted means that the neurons must form two fully connected layers—input layer and hidden layer—as show in the following diagram:

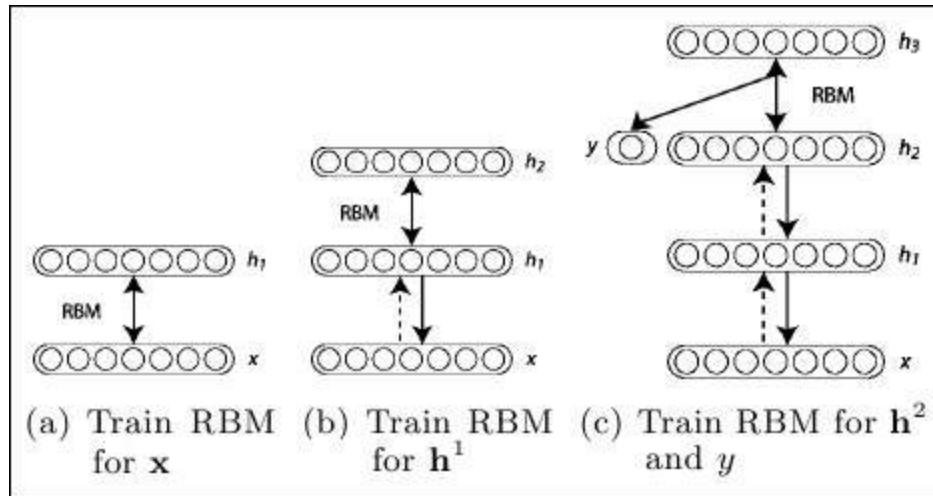


Unlike feedforward networks, the connections between the visible and hidden layers are undirected, hence the values can be propagated in both visible-to-hidden and hidden-to-visible directions.

Training Restricted Boltzman machines is based on the **Contrastive Divergence**

algorithm, which uses a gradient descent procedure, similar to backpropagation, to update weights, and **Gibbs sampling** is applied on the **Markov chain** to estimate the gradient—the direction on how to change the weights.

Restricted Boltzmann machines can also be stacked to create a class known as **Deep Belief Networks (DBNs)**. In this case, the hidden layer of RBM acts as a visible layer for the RBM layer, as shown in the following diagram:



The training, in this case, is incremental; training layer by layer.

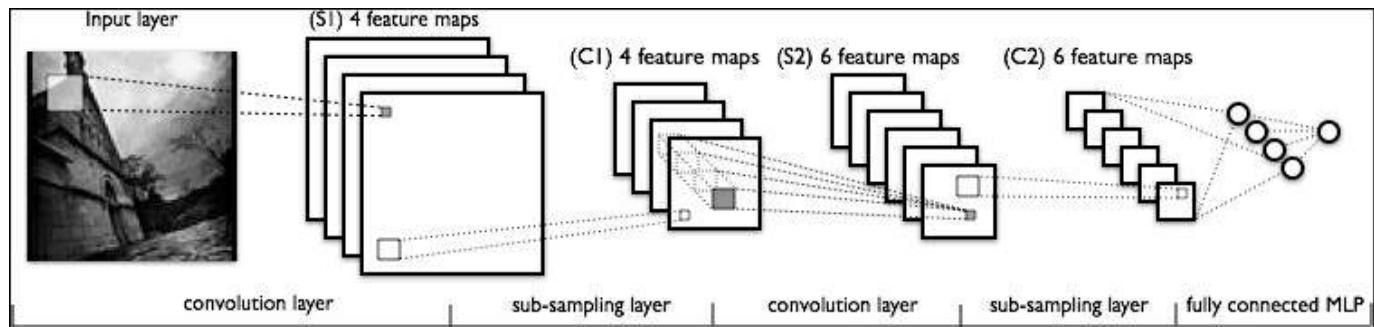
Deep convolutional networks

A network structure that recently achieves very good results at image recognition benchmarks is **Convolutional Neural Network (CNN)** or ConvNet. CNNs are a type of feedforward neural network that are structured in such a way that emulates behavior of the visual cortex, exploiting 2D structures of an input image, that is, patterns that exhibit spatially local correlation.

A CNN consists of a number of convolutional and subsampling layers, optionally followed by fully connected layers. An example is shown in the following image. The input layer reads all the pixels at an image and then we apply multiple filters. In the following image, four different filters are applied. Each filter is applied to the original image, for example, one pixel of a 6×6 filter is calculated as the weighted sum of a 6×6 square of input pixels and corresponding 6×6 weights. This effectively introduces filters similar to the standard image processing, such as smoothing, correlation, edge detection, and so on. The resulting image is called

feature map. In the example in the image, we have four feature maps, one for each filter.

The next layer is the subsampling layer, which reduces the size of the input. Each feature map is subsampled typically with mean or max pooling over a contiguous region of 2×2 (up to 5×5 for large images). For example, if the feature map size is 16×16 and the subsampling region is 2×2 , the reduced feature map size is 8×8 , where 4 pixels (2×2 square) are combined into a single pixel by calculating max, min, mean, or some other functions:



The network may contain several consecutive convolution and subsampling layers, as shown in the preceding diagram. A particular feature map is connected to the next reduced/convoluted feature map, while feature maps at the same layer are not connected to each other.

After the last subsampling or convolutional layer, there is usually a fully connected layer, identical to the layers in a standard multilayer neural network, which represents the target data.

CNN is trained using a modified backpropagation algorithm that takes the subsampling layers into account and updates the convolutional filter weights based on all the values where this filter is applied.

Note

Some good CNN designs can be found at the ImageNet competition results page:

<http://www.image-net.org/>

An example is AlexNet, described in the *ImageNet Classification with Deep*

Covolutional Neural Networks paper by *A. Krizhevsky et al.*

This concludes our review of main neural network structures. In the following section, we'll move to the actual implementation.

Image classification

In this section, we will discuss how to implement some of the neural network structures with the deeplearning4j library. Let's start.

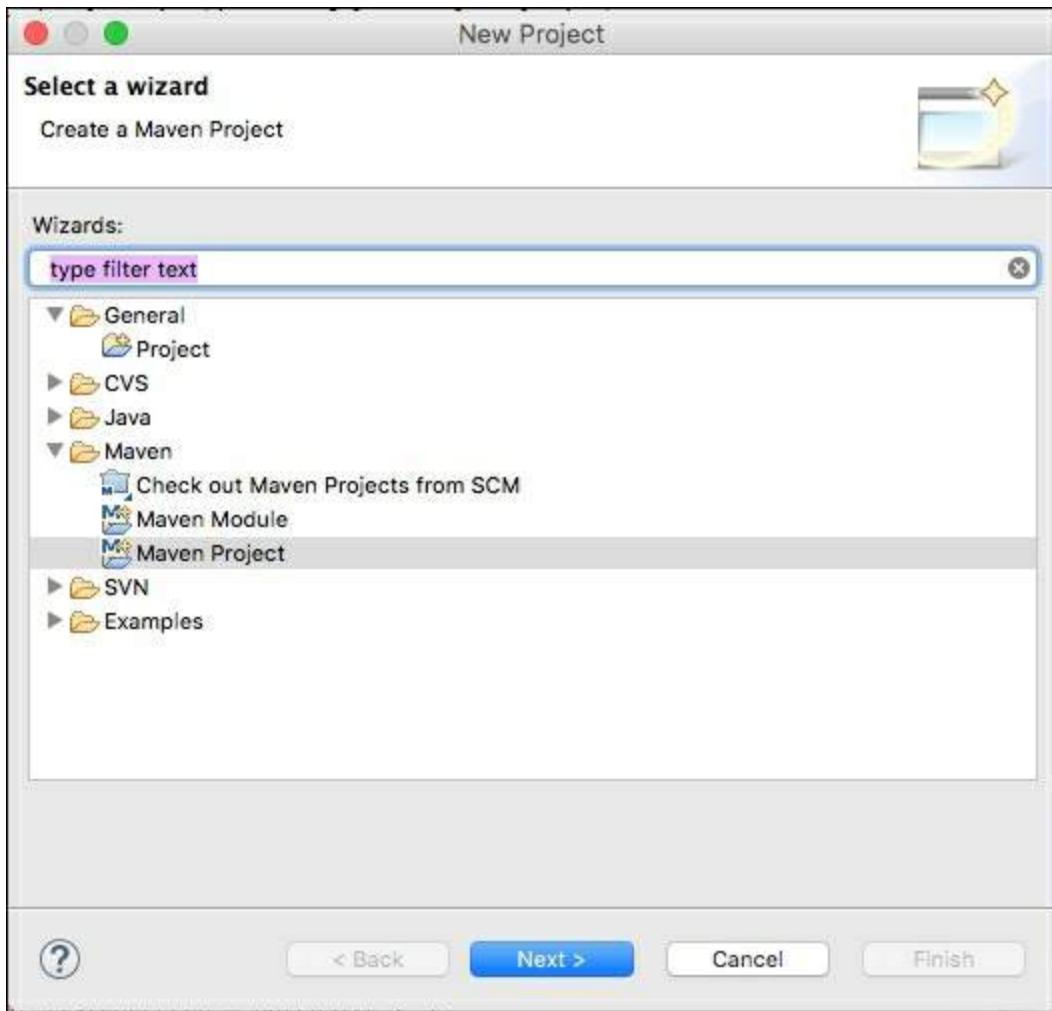
Deeplearning4j

As we discussed in [Chapter 2, Java Libraries and Platforms for Machine Learning](#), deeplearning4j is an open source, distributed deep learning project in Java and Scala. Deeplearning4j relies on Spark and Hadoop for MapReduce, trains models in parallel, and iteratively averages the parameters they produce in a central model. A detailed library summary is presented in [Chapter 2, Java Libraries and Platforms for Machine Learning](#).

Getting DL4J

The most convenient way to get deeplearning4j is through the Maven repository:

1. Start a new Eclipse project and pick **Maven Project**, as shown in the following screenshot:



2. Open the `pom.xml` file and add the following dependencies under the `<dependencies>` section:

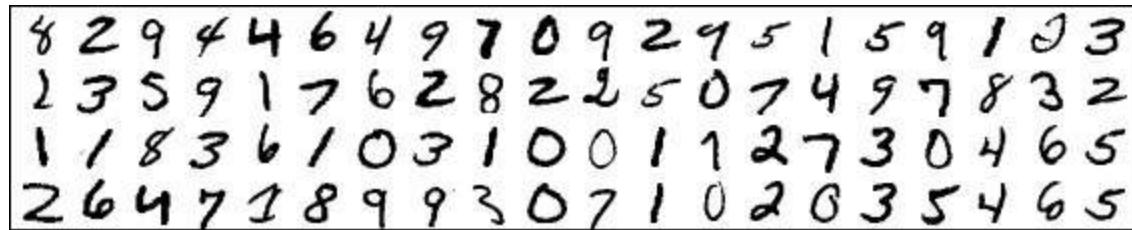
```
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-nlp</artifactId>
    <version>${dl4j.version}</version>
</dependency>

<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-core</artifactId>
    <version>${dl4j.version}</version>
</dependency>
```

3. Finally, right-click on **Project**, select **Maven**, and pick **Update project**.

MNIST dataset

One of the most famous datasets is MNIST dataset, which consists of handwritten digits, as shown in the following image. The dataset comprises 60,000 training and 10,000 testing images:



The dataset is commonly used in image recognition problems to benchmark algorithms. The worst recorded error rate is 12%, with no preprocessing and using a SVM in one-layer neural network. Currently, as of 2016, the lowest error rate is only 0.21%, using the **DropConnect** neural network, followed by **deep convolutional network** at 0.23%, and deep feedforward network at 0.35%.

Now, let's see how to load the dataset.

Loading the data

Deeplearning4j provides the MNIST dataset loader out of the box. The loader is initialized as `DataSetIterator`. Let's first import the `DataSetIterator` class and all the supported datasets that are part of the `impl` package, for example, Iris, MNIST, and others:

```
import org.deeplearning4j.datasets.iterator.DataSetIterator;
import org.deeplearning4j.datasets.iterator.impl.*;
```

Next, we'll define some constants, for instance, the images consist of 28 x 28 pixels and there are 10 target classes and 60,000 samples. Initialize a new `MnistDataSetIterator` class that will download the dataset and its labels. The parameters are iteration batch size, total number of examples, and whether the datasets should be binarized or not:

```
int numRows = 28;
int numColumns = 28;
int outputNum = 10;
int numSamples = 60000;
int batchSize = 100;
DataSetIterator iter = new MnistDataSetIterator(batchSize,
numSamples,true);
```

Having an already-implemented data importer is really convenient, but it won't work on your data. Let's take a quick look at how is it implemented and what needs to be modified to support your dataset. If you're eager to start implementing neural networks, you can safely skip the rest of this section and return to it when you need to import your own data.

Note

To load the custom data, you'll need to implement two classes: `DataSetIterator` that holds all the information about the dataset and `BaseDataFetcher` that actually pulls the data either from file, database, or web. Sample implementations are available on GitHub at

<https://github.com/deeplearning4j/deeplearning4j/tree/master/deeplearning4j-core/src/main/java/org/deeplearning4j/datasets/iterator/impl>.

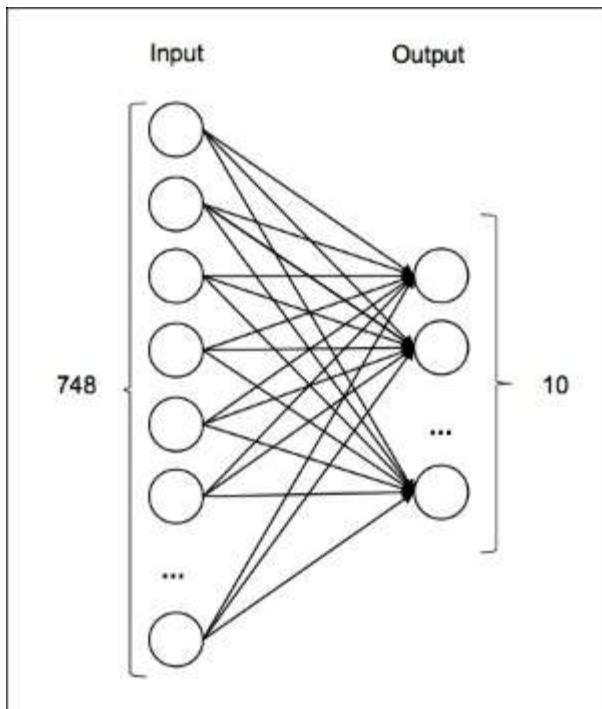
Another option is to use the **Canova** library, which is developed by the same authors, at <http://deeplearning4j.org/canovadoc/>.

Building models

In this section, we'll discuss how to build an actual neural network model. We'll start with a basic single-layer neural network to establish a benchmark and discuss the basic operations. Later, we'll improve this initial result with DBN and **Multilayer Convolutional Network**.

Building a single-layer regression model

Let's start by building a single-layer regression model based on the softmax activation function, as shown in the following diagram. As we have a single layer, **Input** to the neural network will be all the figure pixels, that is, $28 \times 28 = 748$ neurons. The number of **Output** neurons is **10**, one for each digit. The network layers are fully connected, as shown in the following diagram:



A neural network is defined through a `NeuralNetConfiguration` `Builder` object as follows:

```
MultiLayerConfiguration conf = new  
NeuralNetConfiguration.Builder()
```

We will define the parameters for gradient search in order to perform iterations with the **conjugate gradient optimization** algorithm. The momentum parameter determines how fast the optimization algorithm converges to an local optimum—the higher the momentum, the faster the training; but higher speed can lower model's accuracy, as follows:

```
.seed(seed)
.gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
.gradientNormalizationThreshold(1.0)
.iterations(iterations)
.momentum(0.5)
.momentumAfter(Collections.singletonMap(3, 0.9))
.optimizationAlgo(OptimizationAlgorithm.CONJUGATE_GRADIENT)
```

Next, we will specify that the network will have one layer and define the error function (`NEGATIVELOGLIKELIHOOD`), internal perceptron activation function (`softmax`), and the number of input and output layers that correspond to total image pixels and the number of target variables:

```
.list(1)
.layer(0, new
OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD)
.activation("softmax")
.nIn(numRows*numColumns).nOut(outputNum).build())
```

Finally, we will set the network to `pretrain`, disable backpropagation, and actually build the untrained network structure:

```
.pretrain(true).backprop(false)
.build();
```

Once the network structure is defined, we can use it to initialize a new `MultiLayerNetwork`, as follows:

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
```

Next, we will point the model to the training data by calling the `setListeners` method, as follows:

```
model.setListeners(Collections.singletonList((IterationListener) new
ScoreIterationListener(listenerFreq)));
```

We will also call the `fit(int)` method to trigger an end-to-end network training:

```
model.fit(iter);
```

To evaluate the model, we will initialize a new `Evaluation` object that will store batch results:

```
Evaluation eval = new Evaluation(outputNum);
```

We can then iterate over the dataset in batches in order to keep the memory consumption at a reasonable rate and store the results in an `eval` object:

```
DataSetIterator testIter = new MnistDataSetIterator(100, 10000);
while(testIter.hasNext()) {
    DataSet testMnist = testIter.next();
    INDArray predict2 =
        model.output(testMnist.getFeatureMatrix());
    eval.eval(testMnist.getLabels(), predict2);
}
```

Finally, we can get the results by calling the `stats()` function:

```
log.info(eval.stats());
```

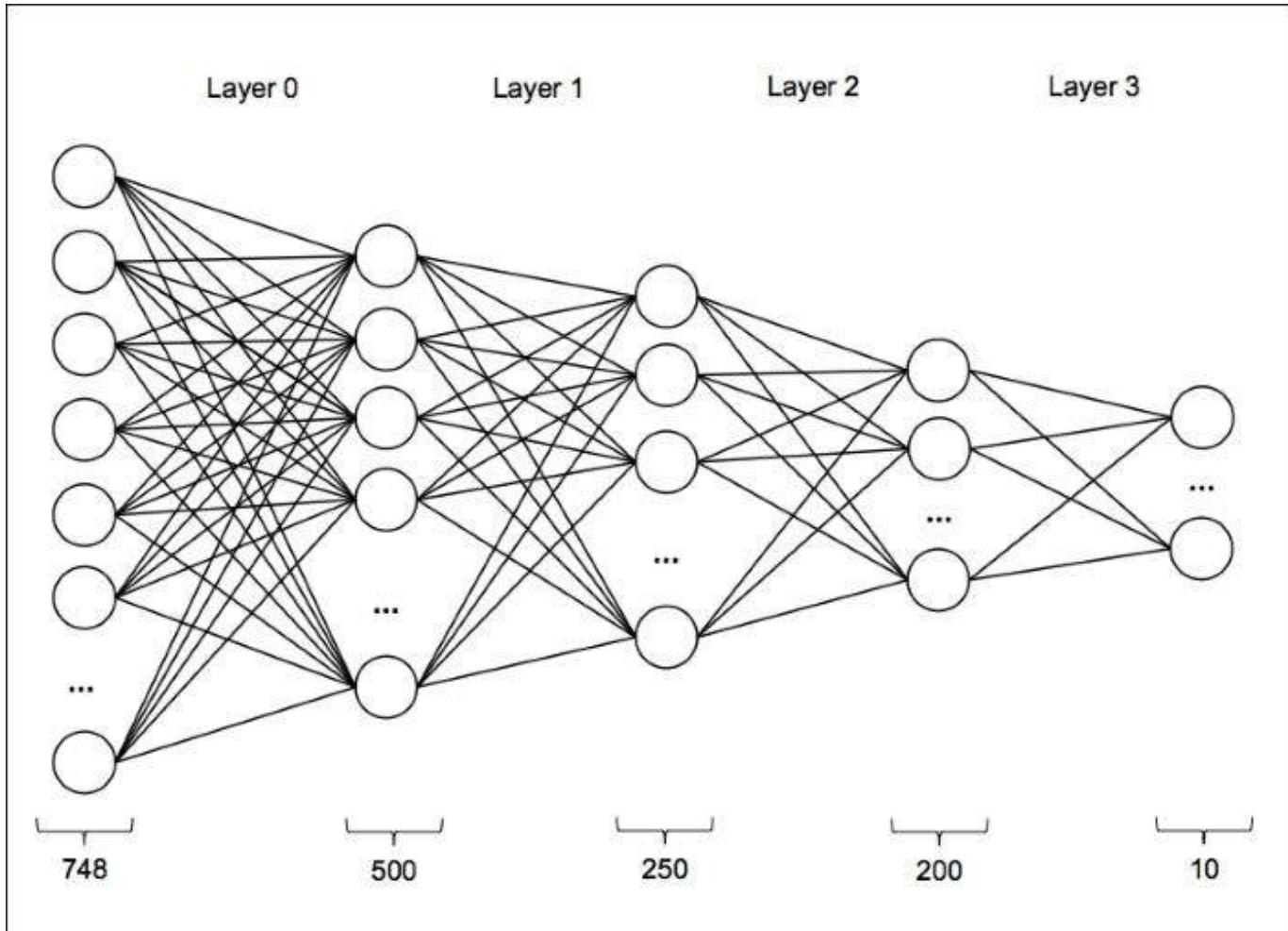
A basic one-layer model achieves the following accuracy:

```
Accuracy: 0.8945
Precision: 0.8985
Recall: 0.8922
F1 Score: 0.8953
```

Getting 89.22% accuracy, that is, 10.88% error rate, on MNIST dataset is quite bad. We'll improve that by going from a simple one-layer network to the moderately sophisticated deep belief network using Restricted Boltzmann machines and Multilayer Convolutional Network.

Building a deep belief network

In this section, we'll build a deep belief network based on Restricted Boltzmann machine, as shown in the following diagram. The network consists of four layers: the first layer receives the **748** inputs to **500** neurons, then to **250**, followed by **200**, and finally to the last **10** target values:



As the code is the same as in the previous example, let's take a look at how to configure such a network:

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
```

We defined the gradient optimization algorithm, as shown in the following code:

```
.seed(seed)
.gradientNormalization(
    GradientNormalization.ClipElementWiseAbsoluteValue)
.gradientNormalizationThreshold(1.0)
.iterations(iterations)
.momentum(0.5)
.momentumAfter(Collections.singletonMap(3, 0.9))
.optimizationAlgo(OptimizationAlgorithm.CONJUGATE_GRADIENT)
```

We will also specify that our network will have four layers:

```
.list(4)
```

The input to the first layer will be 748 neurons and the output will be 500 neurons. We'll use the root mean squared-error cross entropy, Xavier algorithm, to initialize weights by automatically determining the scale of initialization based on the number of input and output neurons, as follows:

```
.layer(0, new RBM.Builder()  
.nIn(numRows*numColumns)  
.nOut(500)  
.weightInit(WeightInit.XAVIER)  
.lossFunction(LossFunction.RMSE_XENT)  
.visibleUnit(RBM.VisibleUnit.BINARY)  
.hiddenUnit(RBM.HiddenUnit.BINARY)  
.build())
```

The next two layers will have the same parameters, except the number of input and output neurons:

```
.layer(1, new RBM.Builder()  
.nIn(500)  
.nOut(250)  
.weightInit(WeightInit.XAVIER)  
.lossFunction(LossFunction.RMSE_XENT)  
.visibleUnit(RBM.VisibleUnit.BINARY)  
.hiddenUnit(RBM.HiddenUnit.BINARY)  
.build())  
.layer(2, new RBM.Builder()  
.nIn(250)  
.nOut(200)  
.weightInit(WeightInit.XAVIER)  
.lossFunction(LossFunction.RMSE_XENT)  
.visibleUnit(RBM.VisibleUnit.BINARY)  
.hiddenUnit(RBM.HiddenUnit.BINARY)  
.build())
```

Now the last layer will map the neurons to outputs, where we'll use the softmax activation function, as follows:

```
.layer(3, new OutputLayer.Builder()  
.nIn(200)  
.nOut(outputNum)  
.lossFunction(LossFunction.NEGATIVELOGLIKELIHOOD)  
.activation("softmax")  
.build())  
.pretrain(true).backprop(false)
```

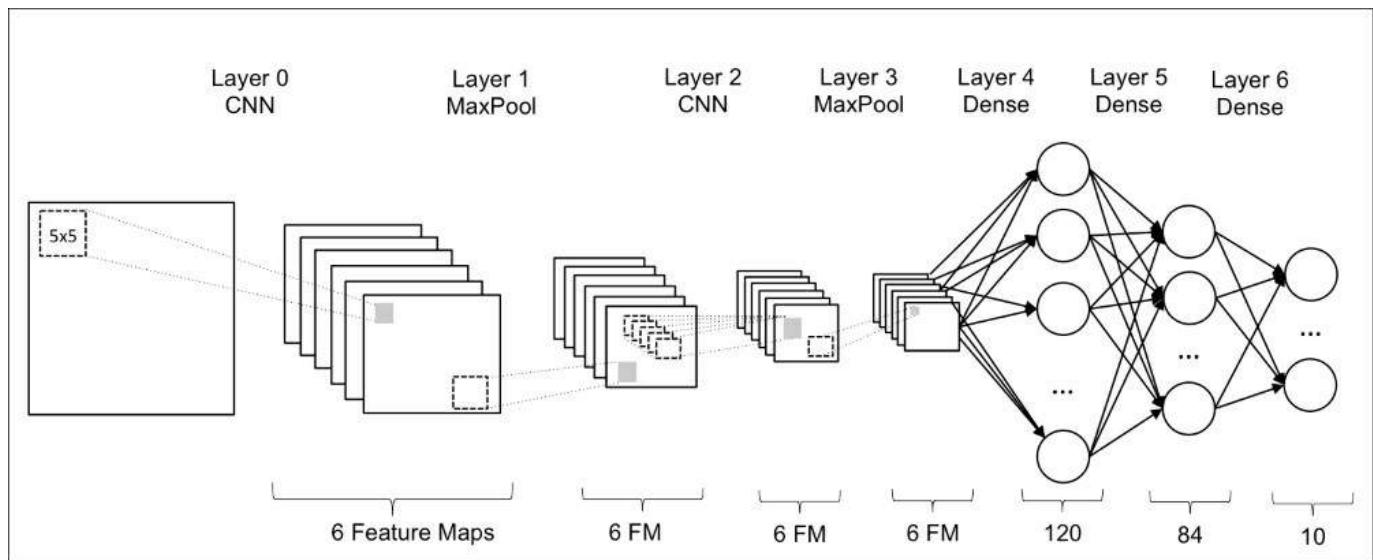
```
.build();
```

The rest of the training and evaluation is the same as in the single-layer network example. Note that training deep network might take significantly more time compared to a single-layer network. The accuracy should be around 93%.

Now let's take a look at another deep network.

Build a Multilayer Convolutional Network

In the final example, we'll discuss how to build a convolutional network, as shown in the following diagram. The network will consist of seven layers: first, we'll repeat two pairs of convolutional and subsampling layers with max pooling. The last subsampling layer is then connected to a densely connected feedforward neuronal network, comprising 120 neurons, 84 neurons, and 10 neurons in the last three layers, respectively. Such a network effectively forms the complete image recognition pipeline, where the first four layers correspond to feature extraction and the last three layers correspond to the learning model:



Network configuration is initialized as we did earlier:

```
MultiLayerConfiguration.Builder conf = new  
NeuralNetConfiguration.Builder()
```

We will specify the gradient descent algorithm and its parameters, as follows:

```

.seed(seed)
.iterations(iterations)
.activation("sigmoid")
.weightInit(WeightInit.DISTRIBUTION)
.dist(new NormalDistribution(0.0, 0.01))
.learningRate(1e-3)
.learningRateScoreBasedDecayRate(1e-1)
.optimizationAlgo(
OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)

```

We will also specify the seven network layers, as follows:

```
.list(7)
```

The input to the first convolutional layer is the complete image, while the output is six feature maps. The convolutional layer will apply a 5 x 5 filter, and the result will be stored in a 1 x 1 cell:

```

.layer(0, new ConvolutionLayer.Builder(
    new int[]{5, 5}, new int[]{1, 1})
    .name("cnn1")
    .nIn(numRows*numColumns)
    .nOut(6)
    .build())

```

The second layer is a subsampling layer that will take a 2 x 2 region and store the max result into a 2 x 2 element:

```

.layer(1, new SubsamplingLayer.Builder(
SubsamplingLayer.PoolingType.MAX,
new int[]{2, 2}, new int[]{2, 2})
.name("maxpool1")
.build())

```

The next two layers will repeat the previous two layers:

```

.layer(2, new ConvolutionLayer.Builder(new int[]{5, 5}, new int[]{1,
1})
    .name("cnn2")
    .nOut(16)
    .biasInit(1)
    .build())
.layer(3, new
SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX, new int[]{2,
2}, new int[]{2, 2})
    .name("maxpool2")
    .build())

```

Now we will wire the output of the subsampling layer into a dense feedforward network, consisting of 120 neurons, and then through another layer, into 84 neurons, as follows:

```
.layer(4, new DenseLayer.Builder()
    .name("ffn1")
    .nOut(120)
    .build())
.layer(5, new DenseLayer.Builder()
    .name("ffn2")
    .nOut(84)
    .build())
```

The final layer connects 84 neurons with 10 output neurons:

```
.layer(6, new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
    .name("output")      .nOut(outputNum)
    .activation("softmax") // radial basis function required
    .build())
.backprop(true)
.pretrain(false)
.cnnInputSize(numRows,numColumns,1);
```

To train this structure, we can reuse the code that we developed in the previous two examples. Again, the training might take some time. The network accuracy should be around 98%.

Note

As model training significantly relies on linear algebra, training can be significantly sped up by using **Graphics Processing Unit (GPU)** for an order of magnitude. As GPU backend is at the time of writing undergoing a rewrite, please check the latest documentation at <http://deeplearning4j.org/documentation>

As we saw in different examples, increasingly more complex neural networks allow us to extract relevant features automatically, thus completely avoiding traditional image processing. However, the price we pay for this is an increased processing time and a lot of learning examples to make this approach efficient.

Summary

In this chapter, we discussed how to recognize patterns in images in order to distinguish between different classes by covering fundamental principles of deep learning and discussing how to implement them with the deeplearning4j library. We started by refreshing the basic neural network structure and discussed how to implement them to solve handwritten digit recognition problem.

In the next chapter, we'll look further into patterns; however, instead of patterns in images, we'll tackle patterns with temporal dependencies that can be found in sensor data.

Chapter 9. Activity Recognition with Mobile Phone Sensors

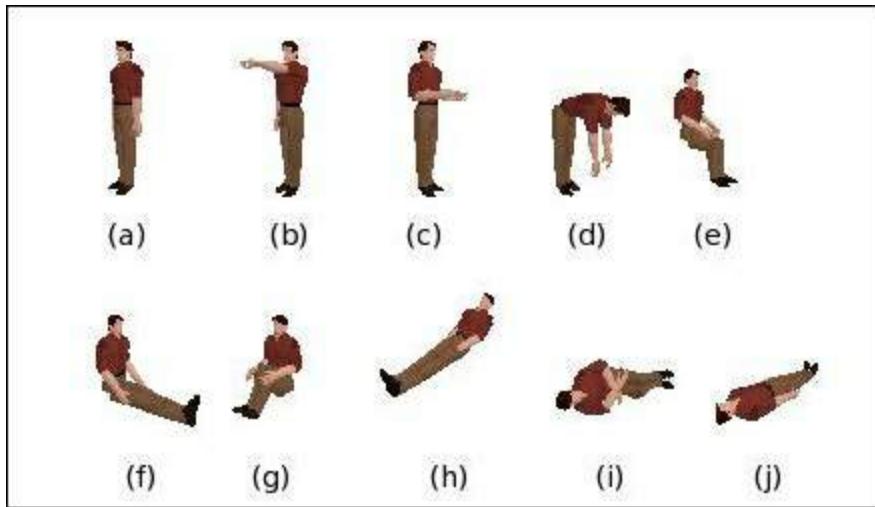
While the previous chapter focused on pattern recognition in images, this chapter is all about recognizing patterns in sensor data, which, in contrast to images, has temporal dependencies. We will discuss how to recognize granular daily activities such as walking, sitting, and running using mobile phone inertial sensors. The chapter also provides references to related research and emphasizes best practices in the activity recognition community.

The topics covered in this chapter will include the following:

- Introducing activity recognition, covering mobile phone sensors and activity recognition pipeline
- Collecting sensor data from mobile devices
- Discussing activity classification and model evaluation
- Deploying activity recognition model

Introducing activity recognition

Activity recognition is an underpinning step in behavior analysis, addressing healthy lifestyle, fitness tracking, remote assistance, security applications, elderly care, and so on. Activity recognition transforms low-level sensor data from sensors, such as accelerometer, gyroscope, pressure sensor, and GPS location, to a higher-level description of behavior primitives. In most cases, these are basic activities, for example, walking, sitting, lying, jumping, and so on, as shown in the following image, or they could be more complex behaviors, such as going to work, preparing breakfast, shopping, and so on:



In this chapter, we will discuss how to add the activity recognition functionality into a mobile application. We will first look at what does an activity recognition problem looks like, what kind of data do we need to collect, what are the main challenges are, and how to address them?

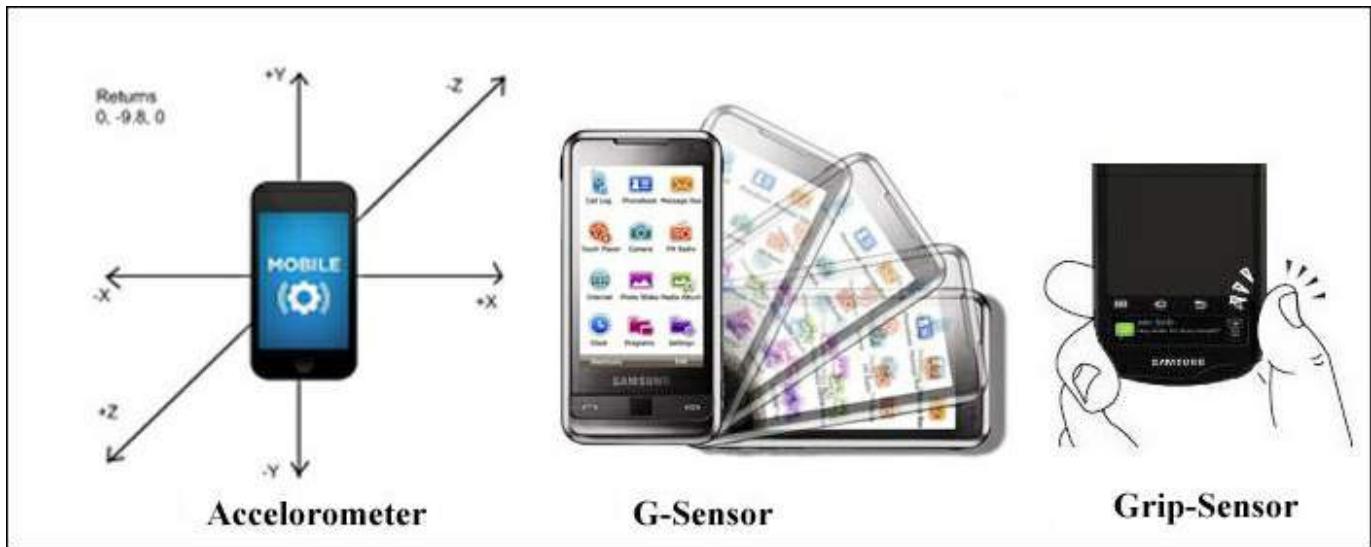
Later, we will follow an example to see how to actually implement activity recognition in an Android application, including data collection, data transformation, and building a classifier.

Let's start!

Mobile phone sensors

Let's first review what kinds of mobile phone sensors there are and what they report. Most smart devices are now equipped with a several built-in sensors that measure the motion, position, orientation, and conditions of the ambient environment. As sensors provide measurements with high precision, frequency, and accuracy, it is possible to reconstruct complex user motions, gestures, and movements. Sensors are often incorporated in various applications; for example, gyroscope readings are used to steer an object in a game, GPS data is used to locate the user, and accelerometer data is used to infer the activity that the user is performing, for example, cycling, running, or walking.

The next image shows a couple of examples what kind of interactions the sensors are able to detect:



Mobile phone sensors can be classified into the following three broad categories:

- **Motion sensors** measure acceleration and rotational forces along the three perpendicular axes. Examples of sensors in this category include accelerometers, gravity sensors, and gyroscopes.
- **Environmental sensors** measure a variety of environmental parameters, such as illumination, air temperature, pressure, and humidity. This category includes barometers, photometers, and thermometers.
- **Position sensors** measure the physical position of a device. This category

includes orientation sensors and magnetometers.

Note

More detailed descriptions for different mobile platforms are available at the following links:

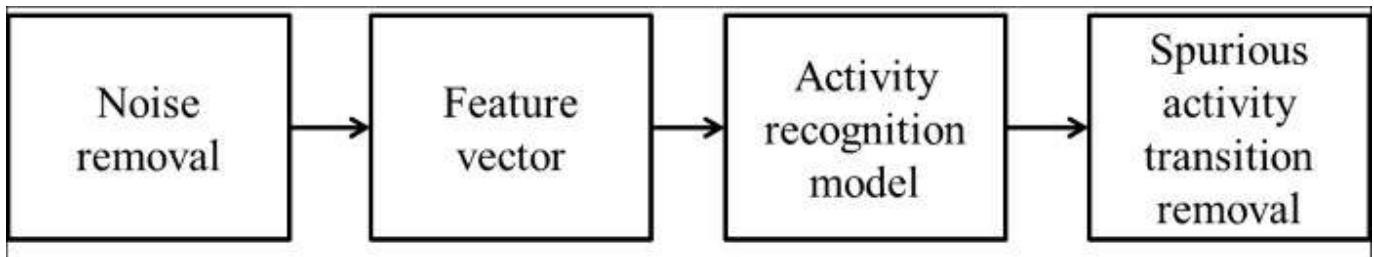
- Android sensors framework:
http://developer.android.com/guide/topics/sensors/sensors_overview.html
- iOS Core Motion framework:
<https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CM>
- Windows Phone: [https://msdn.microsoft.com/en-us/library/windows/apps/hh202968\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/hh202968(v=vs.105).aspx)

In this chapter, we will work only with Android's sensors framework.

Activity recognition pipeline

Classifying multidimensional time-series sensor data is inherently more complex compared to classifying traditional, nominal data as we saw in the previous chapters. First, each observation is temporally connected to the previous and following observations, making it very difficult to apply a straightforward classification of a single set of observations only. Second, the data obtained by sensors at different time points stochastic, that is unpredictable due to influence of sensor noise, environmental disturbances, and many other reasons. Moreover, an activity can comprise various sub-activities executed in different manner and each person performs the activity a bit differently, which results in high intraclass differences. Finally, all these reasons make an activity recognition model imprecise, resulting in new data being often misclassified. One of the highly desirable properties of an activity recognition classifier is to ensure continuity and consistency in the recognized activity sequence.

To deal with these challenges, activity recognition is applied to a pipeline as shown in the following:



In the first step, we attenuate as much noise as we can, for example, by reducing sensor sampling rate, removing outliers, applying high/low-pass filters, and so on. In the next phase, we construct a feature vector, for instance, we convert sensor data from time domain to frequency domain by applying **Discrete Fourier Transform (DFT)**. DFT is a method that takes a list of samples as an input and returns a list of sinusoid coefficients ordered by their frequencies. They represent a combination of frequencies that are present in the original list of samples.

Note

A gentle introduction of Fourier transform is written by Pete Bevelacqua at

<http://www.thefouriertransform.com/>.

If you want to get more technical and theoretical background on the Fourier transform, take a look at the eighth and ninth lectures in the class by Robert Gallanger and Lizhong Zheng at MIT open course:

<http://theopenacademy.com/content/principles-digital-communication>

Next, based on the feature vector and set of training data, we can build an activity recognition model that assigns an atomic action to each observation. Therefore, for each new sensor reading, the model will output the most probable activity label. However, models make mistakes. Hence, the last phase smooths the transitions between activities by removing transitions that cannot occur in reality, for example, it is not physically feasible that the transition between activities lying-standing-lying occurs in less than half a second, hence such transition between activities is smoothed as lying-lying-lying.

The activity recognition model is constructed with a supervised learning approach, which consists of training and classification steps. In the training step, a set of labeled data is provided to train the model. The second step is used to assign a label to the new unseen data by the trained model. The data in both phases must be pre-processed with the same set of tools, such as filtering and feature-vector computation.

The post-processing phase, that is, spurious activity removal, can also be a model itself and, hence, also requires a learning step. In this case, the pre-processing step also includes activity recognition, which makes such arrangement of classifiers a meta-learning problem. To avoid overfitting, it is important that the dataset used for training the post-processing phase is not the same as that used for training the activity recognition model.

We will roughly follow a lecture on smartphone programming by professor Andrew T. Campbell from Dartmouth University and leverage data collection mobile app that they developed in the class (Campbell, 2011).

The plan

The plan consists of training phase and deployment phase. Training phase shown in the following image boils down to the following steps:

1. Install Android Studio and import `MyRunsDataCollector.zip`.
2. Load the application in your Android phone.
3. Collect your data, for example, standing, walking, and running, and transform the data to a feature vector comprising of FFT transforms. Don't panic, low-level signal processing functions such as FFT will not be written from scratch as we will use existing code to do that. The data will be saved on your phone in a file called `features.arff`.
4. Create and evaluate an activity recognition classifier using exported data and implement filter for spurious activity transitions removal.
5. Plug the classifier back into the mobile application.

If you don't have an Android phone, or if you want to skip all the steps related to mobile application, just grab an already-collected dataset located in `data/features.arff` and jump directly to the *Building a classifier* section.

Collecting data from a mobile phone

This section describes the first three steps from the plan. If you want to directly work with the data, you can just skip this section and continue to the following *Building a classifier* section. There are many open source mobile apps for sensor data collection, including an app by Prof. Campbell that we will use in this chapter. The application implements the essentials to collect sensor data for different activity classes, for example, standing, walking, running, and others.

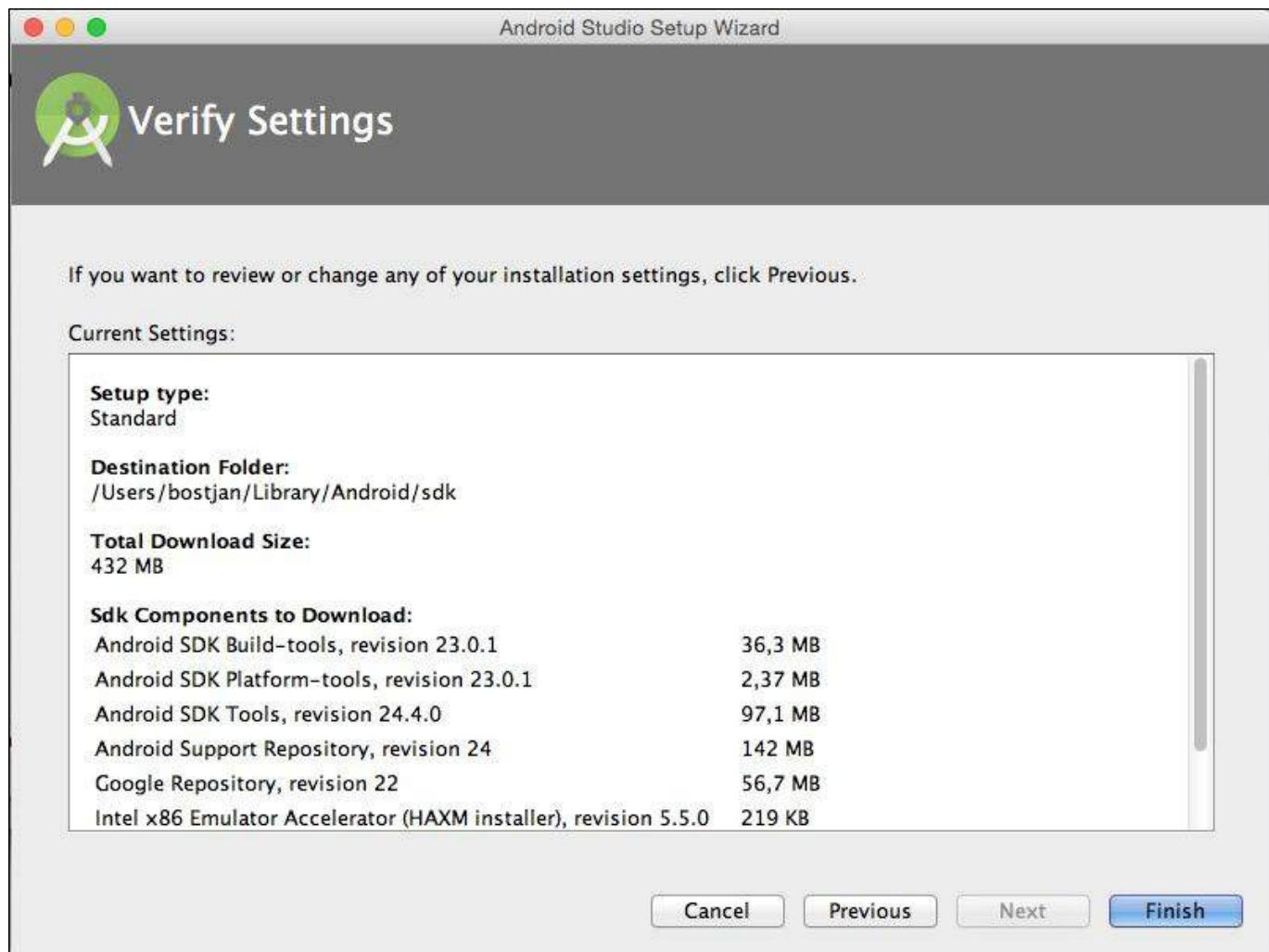
Let's start by preparing the Android development environment. If you have already installed it, jump to the *Loading the data collector* section.

Installing Android Studio

Android Studio is a development environment for Android platform. We will quickly review installation steps and basic configurations required to start the app on a mobile phone. For more detailed introduction to Android development, I would recommend an introductory book, *Android 5 Programming by Example* by *Kyle Mew*.

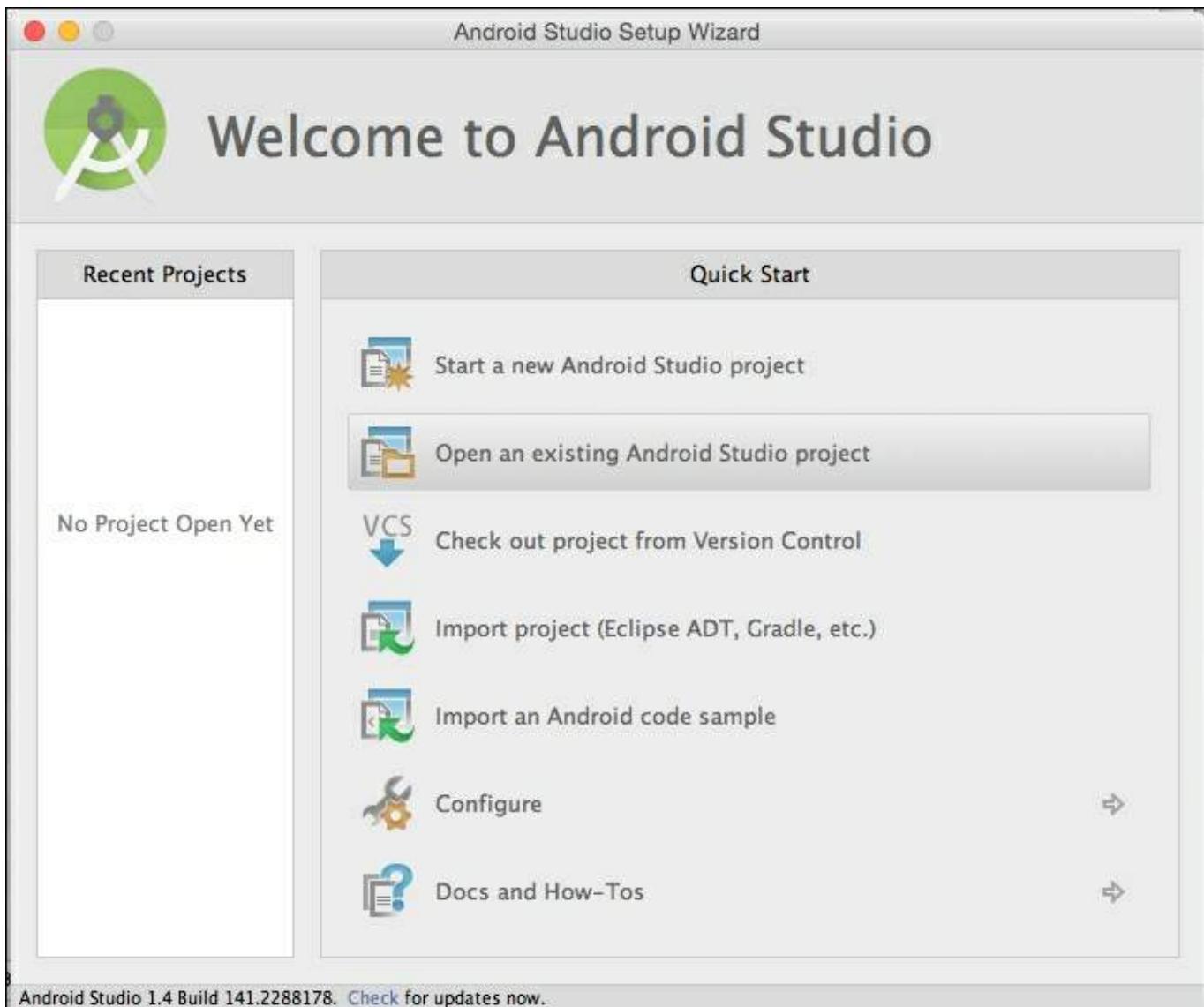
Grab the latest Android Studio for developers at <http://developer.android.com/sdk/installing/index.html?pkg=studio> and follow the installation instructions. The installation will take over 10 minutes, occupying approximately 0.5 GB of space:

Then:



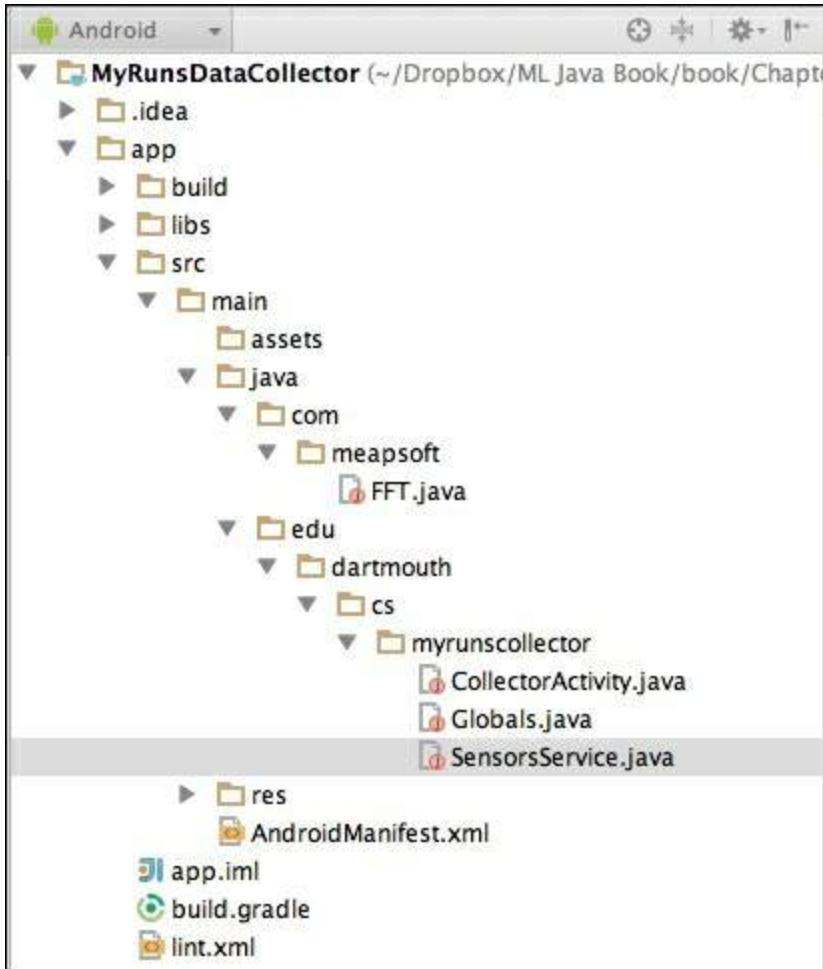
Loading the data collector

First, grab source code of `MyRunsDataCollector` from <http://www.cs.dartmouth.edu/~campbell/cs65/code/myrunsdatacollector.zip>. Once the Android Studio is installed, choose to open an existing Android Studio project as shown in the following image and select the `MyRunsDataCollector` folder. This will import the project to Android Studio:



After the project import is completed, you should be able to see the project files structure, as shown in the following image. As shown in the following, the collector consists of `CollectorActivity.java`, `Globals.java`, and `SensorsService.java`.

The project also shows `FFT.java` implementing low-level signal processing:



The main `myrunscollector` package contains the following classes:

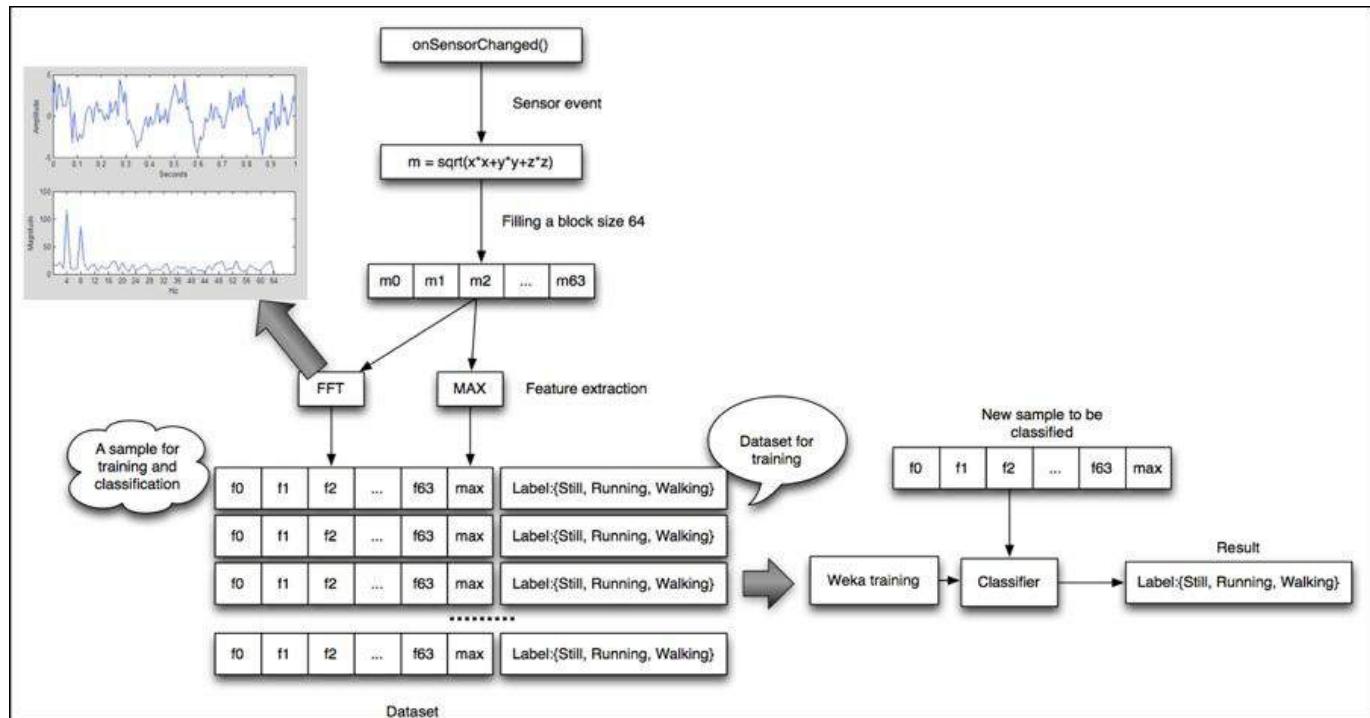
- `Globals.java`: This defines global constants such as activity labels and IDs, data filenames, and so on
- `CollectorActivity.java`: This implements user interface actions, that is, what happens when specific button is pressed
- `SensorsService.java`: This implements a service that collects data, calculates the feature vector as we will discuss in the following sections, and stores the data into a file on the phone

The next question that we will address is how to design features.

Feature extraction

Finding an appropriate representation of the person's activities is probably the most challenging part of activity recognition. The behavior needs to be represented with simple and general features so that the model using these features will also be general and work well on behaviors different from those in the learning set.

In fact, it is not difficult to design features specific to the captured observations in a training set; such features would work well on them. However, as the training set captures only a part of the whole range of human behavior, overly specific features would likely fail on general behavior:



Let's see how it is implemented in `MyRunsDataCollector`. When the application is started, a method called `onSensorChanged()` gets a triple of accelerometer sensor readings (x , y , and z) with a specific time stamp and calculates the magnitude from the sensor readings. The methods buffers up to 64 consecutive magnitudes marked before computing the FFT coefficients (Campbell, 2015):

"As shown in the upper left of the diagram, FFT transforms a time series of amplitude over time to magnitude (some representation of amplitude) across frequency; the example shows some oscillating system where the dominant frequency is between 4-8 cycles/second called Hertz (H) – imagine a ball attached to an elastic band that this stretched and oscillates for a short period"

of time, or your gait while walking, running -- one could look at these systems in the time and frequency domains. The x,y,z accelerometer readings and the magnitude are time domain variables. We transform these time domain data into the frequency domain because the can represent the distribution in a nice compact manner that the classifier will use to build a decision tree model. For example, the rate of the amplitude transposed to the frequency domain may look something like the figure bottom plot -- the top plot is time domain and the bottom plot a transformation of the time to the frequency domain.

The training phase also stores the maximum (MAX) magnitude of the (m0..m63) and the user supplied label (e.g., walking) using the collector. The individual features are computed as magnitudes (f0..f63), the MAX magnitude and the class label."

Now let's move on to the actual data collection.

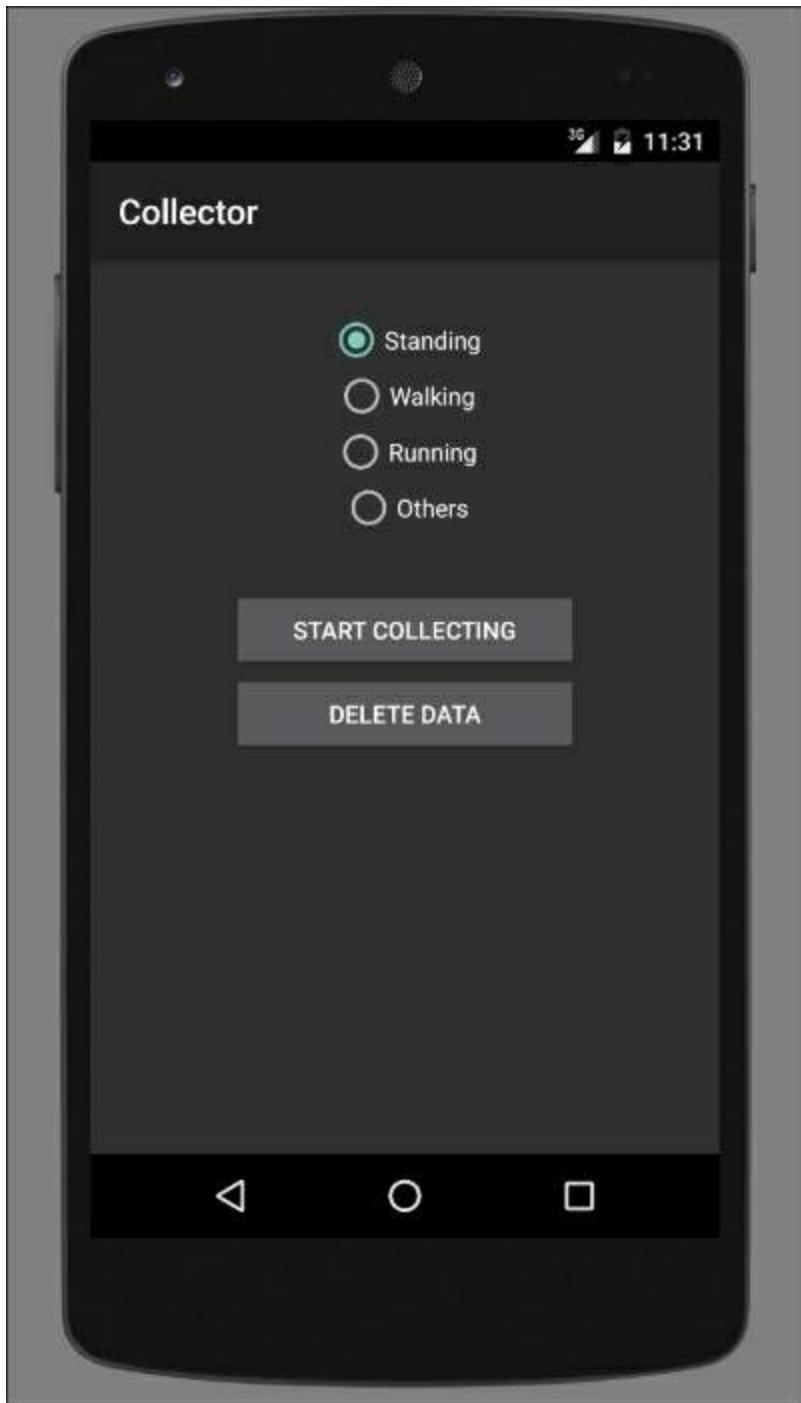
Collecting training data

We can now use the collector to collect training data for activity recognition. The collector supports three activities by default: standing, walking, and running, as shown in the application screenshot in the following figure.

You can select an activity, that is, target class value, and start recording the data by clicking the **START COLLECTING** button . Make sure that each activity is recorded for at least three minutes, for example, if the **Walking** activity is selected, press **START COLLECTING** and walk around for at least three minutes. At the end of the activity, press stop collecting. Repeat this for each of the activities.

You could also collect different scenarios involving these activities, for example, walking in the kitchen, walking outside, walking in a line, and so on. By doing so, you will have more data for each activity class and a better classifier. Makes sense, right? The more data, the less confused the classifier will be. If you only have a little data, overfitting will occur and the classifier will confuse classes—standing with walking, walking with running. However, the more data, the less they get confused. You might collect less than three minutes per class when you are debugging, but for your final polished product, the more data, the better it is. Multiple recording instances will simply be accumulated in the same file.

Note, the delete button removes the data that is stored in a file on the phone. If you want to start over again, hit delete before starting otherwise, the new collected data will be appended at the end of the file:



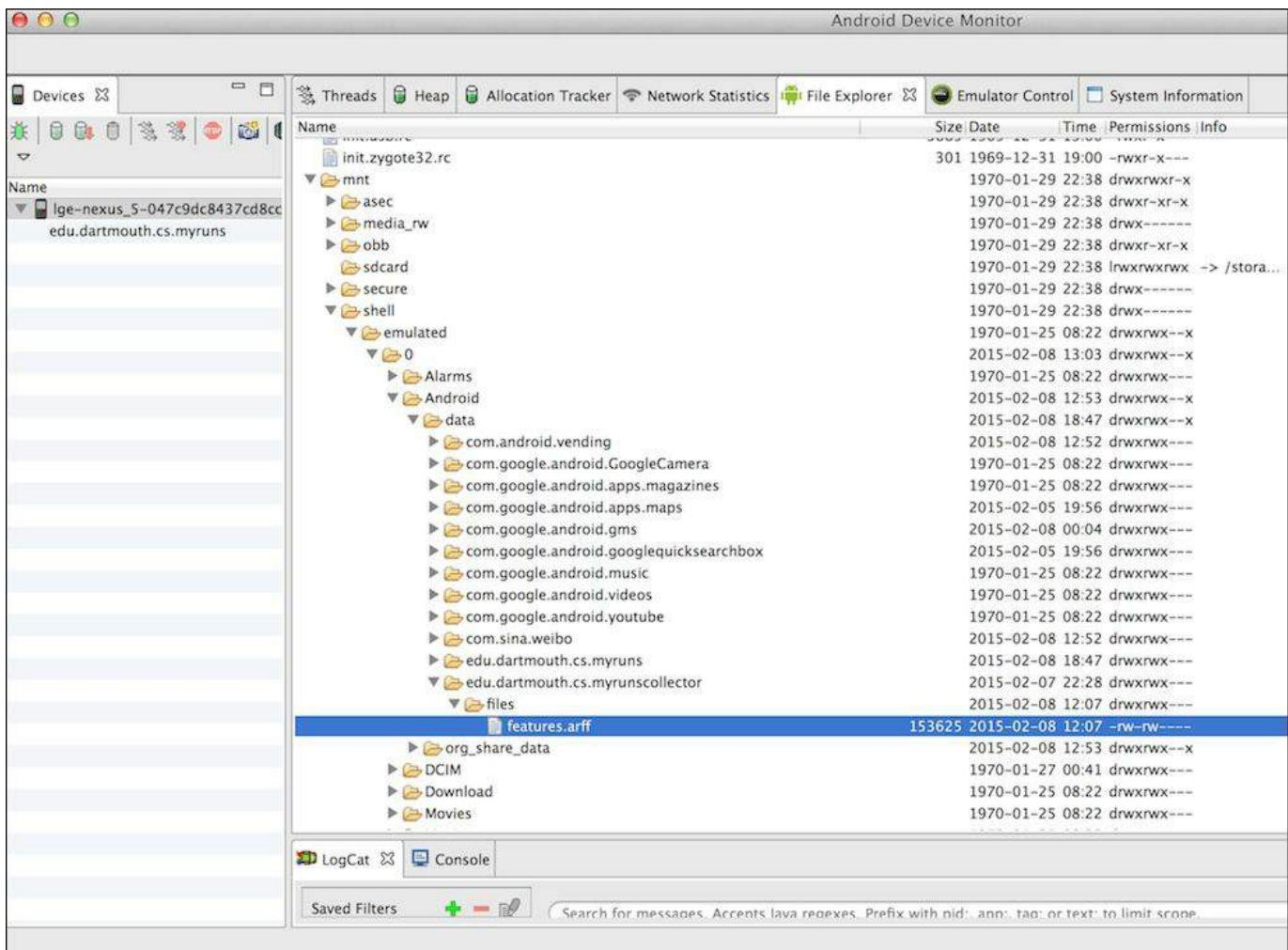
The collector implements the diagram as discussed in the previous sections: it collects accelerometer samples, computes the magnitudes, uses the `FFT.java` class to compute the coefficients, and produces the feature vectors. The data is then stored in a Weka formatted `features.arff` file. The number of feature vectors will vary as you will collect a small or large amount of data. The longer you collect the data, the

more feature vectors are accumulated.

Once you stop collecting the training data using the collector tool, we need to grab the data to carry on the workflow. We can use the file explorer in **Android Device Monitor** to upload the `features.arff` file from the phone and to store it on the computer. You can access your Android Device Monitor by clicking on the Android robot icon as shown in the following image:



By selecting your device on the left, your phone storage content will be shown on the right-hand side. Navigate through `mnt/shell/emulated/Android/data/edu.dartmouth.cs.myrunscollector/files/features.arff`:



To upload this file to your computer, you need to select the file (it is highlighted) and click **Upload**.

Now we are ready to build a classifier.

Building a classifier

Once sensor samples are represented as feature vectors having the class assigned, it is possible to apply standard techniques for supervised classification, including feature selection, feature discretization, model learning, k-fold cross validation, and so on. The chapter will not delve into the details of the machine learning algorithms. Any algorithm that supports numerical features can be applied, including SVMs, random forest, AdaBoost, decision trees, neural networks, multi-layer perceptrons, and others.

Therefore, let's start with a basic one, decision trees: load the dataset, build set class attribute, build a decision tree model, and output the model:

```
String databasePath = "/Users/bostjan/Dropbox/ML Java  
Book/book/datasets/chap9/features.arff";  
  
// Load the data in arff format  
Instances data = new Instances(new BufferedReader(new  
FileReader(databasePath)));  
  
// Set class the last attribute as class  
data.setClassIndex(data.numAttributes() - 1);  
  
// Build a basic decision tree model  
String[] options = new String[]{};  
J48 model = new J48();  
model.setOptions(options);  
model.buildClassifier(data);  
  
// Output decision tree  
System.out.println("Decision tree model:\n"+model);
```

The algorithm first outputs the model, as follows:

```
Decision tree model:  
J48 pruned tree  
-----  
  
max <= 10.353474  
|   fft_coef_0000 <= 38.193106: standing (46.0)  
|   fft_coef_0000 > 38.193106  
|   |   fft_coef_0012 <= 1.817792: walking (77.0/1.0)  
|   |   fft_coef_0012 > 1.817792  
|   |   |   max <= 4.573082: running (4.0/1.0)
```

```
|   |   | max > 4.573082: walking (24.0/2.0)
max > 10.353474: running (93.0)
```

Number of Leaves : 5

Size of the tree : 9

The tree is quite simplistic and seemingly accurate as majority class distributions in the terminal nodes are quite high. Let's run a basic classifier evaluation to validate the results:

```
// Check accuracy of model using 10-fold cross-validation
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(model, data, 10, new Random(1), new String[]
{});
System.out.println("Model performance:\n"+ eval.toSummaryString());
```

This outputs the following model performance:

Correctly Classified Instances	226	92.623	%
Incorrectly Classified Instances	18	7.377	%
Kappa statistic	0.8839		
Mean absolute error	0.0421		
Root mean squared error	0.1897		
Relative absolute error	13.1828	%	
Root relative squared error	47.519	%	
Coverage of cases (0.95 level)	93.0328	%	
Mean rel. region size (0.95 level)	27.8689	%	
Total Number of Instances	244		

The classification accuracy scores very high, 92.62%, which is an amazing result. One important reason why the result is so good lies in our evaluation design. What I mean here is the following: sequential instances are very similar to each other, if we split them randomly during a 10-fold cross validation, there is a high chance that we use almost identical instances for both training and testing; hence, straightforward k-fold cross validation produces an optimistic estimate of model performance.

A better approach is to use folds that correspond to different sets of measurements or even different people. For example, we can use the application to collect learning data of five people. Then, it makes sense to run k-person cross validation, where the model is trained on four people and tested on the fifth person. The procedure is repeated for each person and the results are averaged. This will give us a much more realistic estimate of the model performance.

Leaving evaluation comment aside, let's look at how to deal with classifier errors.

Reducing spurious transitions

At the end of the activity recognition pipeline, we want to make sure that the classifications are not too volatile, that is, we don't want activities to change every millisecond. A basic approach is to design a filter that ignores quick changes in the activity sequence.

We build a filter that remembers the last window activities and returns the most frequent one. If there are multiple activities with the same score, it returns the most recent one.

First, we create a new `SpuriousActivityRemoval` class that will hold a list of activities and the `window` parameter:

```
class SpuriousActivityRemoval {  
  
    List<Object> last;  
    int window;  
  
    public SpuriousActivityRemoval(int window) {  
        this.last = new ArrayList<Object>();  
        this.window = window;  
    }  
}
```

Next, we create the `Object filter(Object)` method that will take an activity and return a filtered activity. The method first checks whether we have enough observations. If not, it simply stores the observation and returns the same value, as shown in the following code:

```
public Object filter(Object obj) {  
    if(last.size() < window) {  
        last.add(obj);  
        return obj;  
    }  
}
```

If we already collected `window` observations, we simply return the most frequent observation, remove the oldest observation, and insert the new observation:

```
Object o = getMostFrequentElement(last);  
last.add(obj);  
last.remove(0);  
return o;  
}
```

What is missing here is a function that returns the most frequent element from a list of objects. We implement this with a hash map, as follows:

```
private Object getMostFrequentElement(List<Object> list) {  
  
    HashMap<String, Integer> objectCounts = new HashMap<String, Integer>();  
    Integer frequntCount = 0;  
    Object frequentObject = null;
```

Now, we iterate over all the elements in the list, insert each unique element into a hash map, or update its counter if it is already in the hash map. At the end of the loop, we store the most frequent element that we found so far, as follows:

```
for(Object obj : list){  
    String key = obj.toString();  
    Integer count = objectCounts.get(key);  
    if(count == null){  
        count = 0;  
    }  
    objectCounts.put(key, ++count);  
  
    if(count >= frequntCount){  
        frequntCount = count;  
        frequentObject = obj;  
    }  
}  
  
return frequentObject;  
}  
  
}
```

Let's run a simple example:

```
String[] activities = new String[]{"Walk", "Walk", "Walk", "Run",  
"Walk", "Run", "Run", "Sit", "Sit", "Sit"};  
SpuriousActivityRemoval dlpFilter = new SpuriousActivityRemoval(3);  
for(String str : activities){  
    System.out.println(str + " -> " + dlpFilter.filter(str));  
}
```

The example outputs the following activities:

```
Walk -> Walk  
Walk -> Walk  
Walk -> Walk
```

```
Run -> Walk  
Walk -> Walk  
Run -> Walk  
Run -> Run  
Sit -> Run  
Sit -> Run  
Sit -> Sit
```

The result is a continuous sequence of activities, that is, we do not have quick changes. This adds some delay, but unless this is absolutely critical for the application, it is acceptable.

Activity recognition may be enhanced by appending n previous activities as recognized by the classifier to the feature vector. The danger of appending previous activities is that the machine learning algorithm may learn that the current activity is always the same as the previous one, as this will often be the case. The problem may be solved by having two classifiers, A and B: the classifier B's attribute vector contains n previous activities as recognized by the classifier A. The classifier A's attribute vector does not contain any previous activities. This way, even if B gives a lot of weight to the previous activities, the previous activities as recognized by A will change as A is not burdened with B's inertia.

All that remains to do is to embed the classifier and filter into our mobile application.

Plugging the classifier into a mobile app

There are two ways to incorporate a classifier into a mobile application. The first one involves exporting a model in the Weka format, using the Weka library as a dependency in our mobile application, loading the model, and so on. The procedure is identical to the example we saw in [Chapter 3, Basic Algorithms – Classification, Regression, and Clustering](#). The second approach is more lightweight; we export the model as a source code, for example, we create a class implementing the decision tree classifier. Then we can simply copy and paste the source code into our mobile app, without even importing any Weka dependencies.

Fortunately, some Weka models can be easily exported to source code by the `toSource(String)` function:

```
// Output source code implementing the decision tree
System.out.println("Source code:\n" +
    model.toSource("ActivityRecognitionEngine"));
```

This outputs an `ActivityRecognitionEngine` class that corresponds to our model. Now, let's take a closer look at the outputted code:

```
class ActivityRecognitionEngine {

    public static double classify(Object[] i)
        throws Exception {

        double p = Double.NaN;
        p = ActivityRecognitionEngine.N17a7cec20(i);
        return p;
    }

    static double N17a7cec20(Object []i) {
        double p = Double.NaN;
        if (i[64] == null) {
            p = 1;
        } else if (((Double) i[64]).doubleValue() <= 10.353474) {
            p = ActivityRecognitionEngine.N65b3120a1(i);
        } else if (((Double) i[64]).doubleValue() > 10.353474) {
            p = 2;
        }
        return p;
    }

    ...
}
```

The outputted `ActivityRecognitionEngine` class implements the decision tree that

we discussed earlier. The machine-generated function names, such as `N17a7cec20 (Object [])`, correspond to decision tree nodes. The classifier can be called by the `classify (Object [])` method, where we should pass a feature vector obtained by the same procedure as we discussed in the previous sections. As usual, it returns a double, indicating a class label index.

Summary

In this chapter, we discussed how to implement an activity recognition model for mobile applications. We looked into the completed process, including data collection, feature extraction, model building, evaluation, and model deployment.

In the next chapter, we will move on to another Java library targeted at text analysis —Mallet.

Chapter 10. Text Mining with Mallet – Topic Modeling and Spam Detection

In this chapter, we will first discuss what text mining is, what kind of analysis is it able to offer, and why you might want to use it in your application. We will then discuss how to work with Mallet, a Java library for natural language processing, covering data import and text pre-processing. Afterwards, we will look into two text mining applications: topic modeling, where we will discuss how text mining can be used to identify topics found in the text documents without reading them individually; and spam detection, where we will discuss how to automatically classify text documents into categories.

This chapter will cover the following topics:

- Introducing text mining
- Installing and working with Mallet
- Topic modeling
- Spam detection

Introducing text mining

Text mining, or text analytics, refers to the process of automatically extracting high-quality information from text documents, most often written in natural language, where high-quality information is considered to be relevant, novel, and interesting.

While a typical text-analytics application is to scan a set of documents to generate a search index, text mining can be used in many other applications, including text categorization into specific domains; text clustering to automatically organize a set of documents; sentiment analysis to identify and extract subjective information in documents; concept/entity extraction that is capable of identifying people, places, organizations, and other entities from documents; document summarization to automatically provide the most important points in the original document; and learning relations between named entities.

The process based on statistical pattern mining usually involves the following steps:

1. Information retrieval and extraction.
2. Transforming unstructured text data into structured; for example, parsing, removing noisy words, lexical analysis, calculating word frequencies, deriving linguistic features, and so on.
3. Discovery of patterns from structured data and tagging/annotation.
4. Evaluation and interpretation of the results.

Later in this chapter, we will look at two application areas: topic modeling and text categorization. Let's examine what they bring to the table.

Topic modeling

Topic modeling is an unsupervised technique and might be useful if you need to analyze a large archive of text documents and wish to understand what the archive contains, without necessarily reading every single document by yourself. A text document can be a blog post, e-mail, tweet, document, book chapter, diary entry, and so on. Topic modeling looks for patterns in a corpus of text; more precisely, it identifies topics as lists of words that appear in a statistically meaningful way. The most well-known algorithm is **Latent Dirichlet Allocation** (Blei et al, 2003), which assumes that author composed a piece of text by selecting words from possible baskets of words, where each basket corresponds to a topic. Using this assumption, it becomes possible to mathematically decompose text into the most likely baskets from where the words first came. The algorithm then iterates over this process until it converges to the most likely distribution of words into baskets, which we call topics.

For example, if we use topic modeling on a series of news articles, the algorithm would return a list of topics and keywords that most likely comprise of these topics. Using the example of news articles, the list might look similar to the following:

- Winner, goal, football, score, first place
- Company, stocks, bank, credit, business
- Election, opponent, president, debate, upcoming

By looking at the keywords, we can recognize that the news articles were concerned with sports, business, upcoming election, and so on. Later in this chapter, we will learn how to implement topic modeling using the news article example.

Text classification

In text classification, or text categorization, the goal is to assign a text document according to its content to one or more classes or categories, which tend to be a more general subject area such as vehicles or pets. Such general classes are referred to as topics, and the classification task is then called text classification, text categorization, topic classification, or topic spotting. While documents can be categorized according to other attributes such as document type, author, printing year, and so on, the focus in this chapter will be on the document content only. Examples of text classification include the following components:

- Spam detection in e-mail messages, user comments, webpages, and so on
- Detection of sexually-explicit content
- Sentiment detection, which automatically classifies a product/service review as positive or negative
- E-mail sorting according to e-mail content
- Topic-specific search, where search engines restrict searches to a particular topic or genre, thus providing more accurate results

These examples show how important text classification is in information retrieval systems, hence most modern information retrieval systems use some kind of text classifier. The classification task that we will use as an example in this book is text classification for detecting e-mail spam.

We continue this chapter with an introduction to Mallet, a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text. We will then cover two text-analytics applications, namely, topics modeling and spam detection as text classification.

Installing Mallet

Mallet is available for download at UMass Amherst University website at <http://mallet.cs.umass.edu/download.php>. Navigate to the **Download** section as shown in the following image and select the latest stable release (2.0.8, at the time of writing this book):

The screenshot shows the Mallet website's download page. The header features the Mallet logo and the text "MACHINE LEARNING FOR LANGUAGE TOOLKIT". The right side of the header includes the UMass Amherst seal and the text "UMASS AMHERST". On the left, there's a sidebar with links like Home, Tutorial slides / video, Download, API, Quick Start, Sponsors, Mailing List, and About. Below that is a section for Importing Data, Classification, Sequence Tagging, Topic Modeling, Optimization, and Graphical Models. At the bottom of the sidebar, it says "MALLET is open source software" and "License: For research use, please remember to cite MALLET". The main content area starts with a "Current release" section mentioning "mallet-2.0.8RC3.tar.gz mallet-2.0.8RC3.zip". It notes that until 2.0.8 is an official release, 2.0.7 will remain available. It provides instructions for Windows installation (setting %MALLET_HOME% to point to the Mallet directory) and development releases (using GitHub). It also shows the command "git clone https://github.com/mimno/Mallet.git" and the "ant" command. It explains how to build a single ".jar" file using "ant jar". The "Older releases" section lists various tar.gz files from version 2.0.6 down to 0.4.

Current release: The following packaged release of MALLET 2.0 is available:

[mallet-2.0.8RC3.tar.gz](#) [mallet-2.0.8RC3.zip](#)

Until 2.0.8 is an official release the old 2.0.7 release will remain available.
2.0.8RC3 is much more stable than 2.0.7.

[mallet-2.0.7.tar.gz](#) [mallet-2.0.7.zip](#) ([notes](#))

Windows installation: After unzipping MALLET, set the environment variable %MALLET_HOME% to point to the MALLET directory. In all command line examples, substitute bin\mallet for bin\mallet.

Development release: To download the most current version of MALLET 2.0, use our public GitHub repository:

```
git clone https://github.com/mimno/Mallet.git
```

from the command prompt to get the Mallet package.

To build a Mallet 2.0 development release, you must have the [Apache ant](#) build tool installed. From the command prompt, first change to the mallet directory, and then type

```
ant
```

If ant finishes with "BUILD SUCCESSFUL", Mallet is now ready to use.

If you would like to deploy Mallet as part of a larger application, it is helpful to create a single ".jar" file that contains all of the compiled code. Once you have compiled the individual Mallet class files, use the command:

```
ant jar
```

This process will create a file "mallet.jar" in the "dist" directory within Mallet.

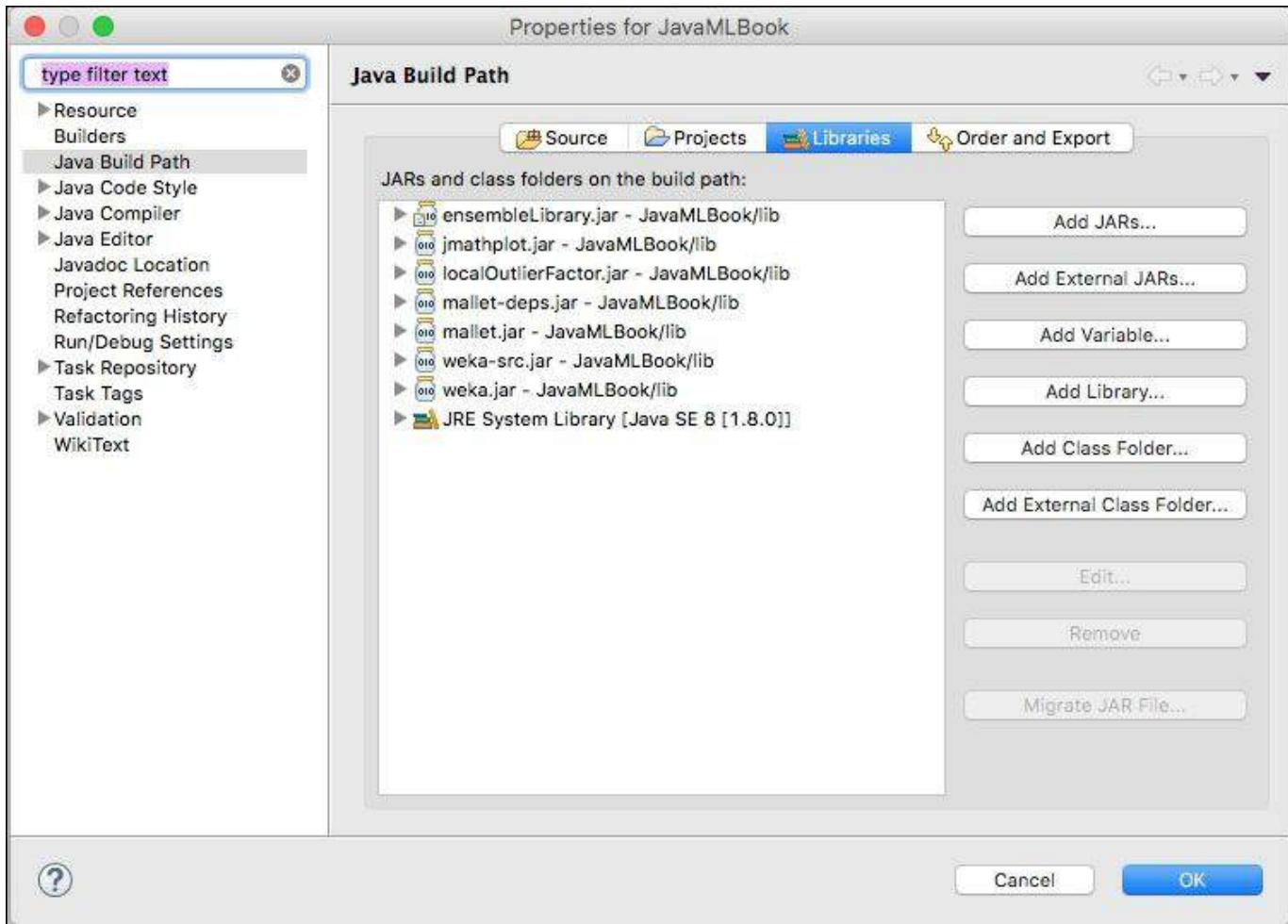
Older releases: MALLET version 0.4 is available for [download](#), but is not being actively maintained. This release includes classes in the package "edu.umass.cs.mallet.base", while MALLET 2.0 contains classes in the package "cc.mallet".

- [mallet-2.0.6.tar.gz](#)
- [mallet-2.0.5.tar.gz](#) ([notes](#))
- [mallet-2.0-RC4.tar.gz](#) ([notes](#))
- [mallet-2.0-RC3.tar.gz](#) ([notes](#))
- [mallet-2.0-RC2.tar.gz](#)
- [mallet-2.0-RC1.tar.gz](#)
- [mallet-0.4.tar.gz](#)

Download the ZIP file and extract the content. In the extracted directory, you should find a folder named `dist` with two JAR files: `mallet.jar` and `mallet-deps.jar`. The first one contains all the packaged Mallet classes, while the second one packs all the dependencies. Include both JARs in your project as referenced libraries, as shown in the following image:

Name		Size	Kind
► bin		--	Folder
build.xml		3 KB	XML
► class		--	Folder
▼ dist		--	Folder
mallet-deps.jar		2,6 MB	Java JAR file
mallet.jar		2,2 MB	Java JAR file
► lib		--	Folder
LICENSE		12 KB	TextEd...ument
Makefile		4 KB	TextEd...ument
pom.xml		3 KB	XML
README.md		2 KB	Markd...ument
► sample-data		--	Folder
► src		--	Folder
► stoplists		--	Folder
► test		--	Folder

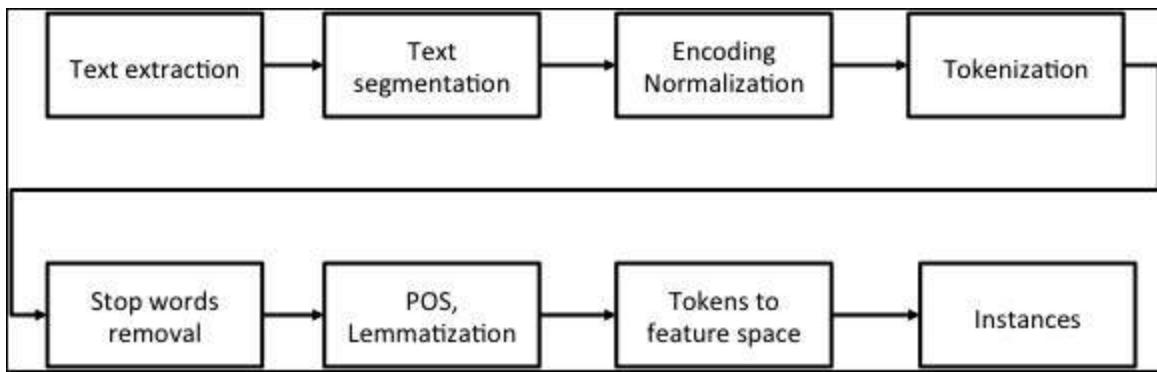
If you are using Eclipse, right click on **Project**, select **Properties**, and pick **Java Build Path**. Select the **Libraries** tab and click **Add External JARs**. Now, select the two JARs and confirm, as shown in the following screenshot:



Now we are ready to start using Mallet.

Working with text data

One of the main challenges in text mining is transforming unstructured written natural language into structured attribute-based instances. The process involves many steps as shown in the following image:



First, we extract some text from the Internet, existing documents, or databases. At the end of the first step, the text could still be presented in the XML format or some other proprietary format. The next step is to, therefore, extract the actual text only and segment it into parts of the document, for example, title, headline, abstract, body, and so on. The third step is involved with normalizing text encoding to ensure the characters are presented the same way, for example, documents encoded in formats such as ASCII, ISO 8859-1, and Windows-1250 are transformed into Unicode encoding. Next, tokenization splits the document into particular words, while the following step removes frequent words that usually have low predictive power, for example, the, a, I, we, and so on.

The **part-of-speech (POS)** tagging and lemmatization step could be included to transform each token (that is, word) to its basic form, which is known as lemma, by removing word endings and modifiers. For example, running becomes run, better becomes good, and so on. A simplified approach is stemming, which operates on a single word without any context of how the particular word is used, and therefore, cannot distinguish between words having different meaning, depending on the part of speech, for example, axes as plural of axe as well as axis.

The last step transforms tokens into a feature space. Most often feature space is a **bag-of-words (BoW)** presentation. In this presentation, a set of all words appearing

in the dataset is created, that is, a bag of words. Each document is then presented as a vector that counts how many times a particular word appears in the document.

Consider the following example with two sentences:

- Jacob likes table tennis. Emma likes table tennis too.
- Jacob also likes basketball.

The bag of words in this case consists of {Jacob, likes, table, tennis, Emma, too, also, basketball}, which has eight distinct words. The two sentences could be now presented as vectors using the indexes of the list, indicating how many times a word at a particular index appears in the document, as follows:

- [1, 2, 2, 2, 1, 0, 0, 0]
- [1, 1, 0, 0, 0, 0, 1, 1]

Such vectors finally become instances for further learning.

Note

Another very powerful presentation based on the BoW model is **word2vec**.

Word2vec was introduced in 2013 by a team of researchers led by Tomas Mikolov at Google. Word2vec is a neural network that learns distributed representations for words. An interesting property of this presentation is that words appear in clusters, such that some word relationships, such as analogies, can be reproduced using vector math. A famous example shows that king - man + woman returns queen.

Further details and implementation are available at the following link:

<https://code.google.com/archive/p/word2vec/>

Importing data

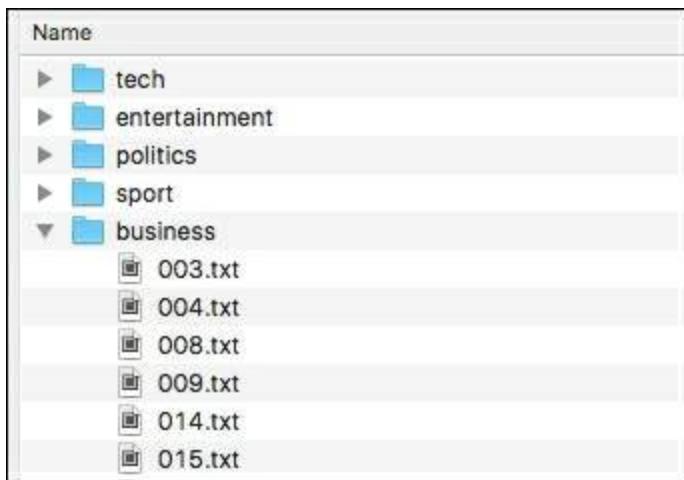
In this chapter, we will not look into how to scrap a set of documents from a website or extract them from database. Instead, we will assume that we already collected them as set of documents and store them in the `.txt` file format. Now let's look at two options how to load them. The first option addresses the situation where each document is stored in its own `.txt` file. The second option addresses the situation where all the documents are stored in a single file, one per line.

Importing from directory

Mallet supports reading from directory with the `cc.mallet.pipe.iterator.FileIterator` class. File iterator is constructed with the following three parameters:

- A list of `File[]` directories with text files
- File filter that specifies which files to select within a directory
- A pattern that is applied to a filename to produce a class label

Consider the data structured into folders as shown in the following image. We have documents organized in five topics by folders (`tech`, `entertainment`, `politics`, `sport`, `business`). Each folder contains documents on particular topics, as shown in the following image:



In this case, we initialize `iterator` as follows:

```

FileIterator iterator =
    new FileIterator(new File[] {new File("path-to-my-dataset")},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);

```

The first parameter specifies the path to our root folder, the second parameter limits the iterator to the `.txt` files only, while the last parameter asks the method to use the last directory name in the path as class label.

Importing from file

Another option to load the documents is through

`cc.mallet.pipe.iterator.CsvIterator.CsvIterator(Reader, Pattern, int, int, int)`, which assumes all the documents are in a single file and returns one instance per line extracted by a regular expression. The class is initialized by the following components:

- `Reader`: This is the object that specifies how to read from a file
- `Pattern`: This is a regular expression, extracting three groups: data, target label, and document name
- `int, int, int`: These are the indexes of data, target, and name groups as they appear in a regular expression

Consider a text document in the following format, specifying document name, category and content:

```

AP881218 local-news A 16-year-old student at a private Baptist...
AP880224 business The Bechtel Group Inc. offered in 1985 to...
AP881017 local-news A gunman took a 74-year-old woman hostage...
AP900117 entertainment Cupid has a new message for lovers this...
AP880405 politics The Reagan administration is weighing w...

```

To parse a line into three groups, we can use the following regular expression:

```
^(\\S*) [\\s,]* (\\S*) [\\s,]* (.*$)
```

There are three groups that appear in parenthesis, `()`, where the third group contains the data, the second group contains the target class, and the first group contains the document ID. The iterator is initialized as follows:

```

CsvIterator iterator = new CsvIterator (
fileReader,
Pattern.compile("^(\\S*) [\\s,]* (\\S*) [\\s,]* (.*$)");

```

```
3, 2, 1));
```

Here the regular expression extracts the three groups separated by an empty space and their order is 3, 2, 1.

Now let's move to data pre-processing pipeline.

Pre-processing text data

Once we initialized an iterator that will go through the data, we need to pass the data through a sequence of transformations as described at the beginning of this section. Mallet supports this process through a pipeline and a wide variety of steps that could be included in a pipeline, which are collected in the `cc.mallet.pipe` package. Some examples are as follows:

- `Input2CharSequence`: This is a pipe that can read from various kinds of text sources (either URI, File, or Reader) into `CharSequence`
- `CharSequenceRemoveHTML`: This pipe removes HTML from `CharSequence`
- `MakeAmpersandXMLFriendly`: This converts & to & in tokens of a token sequence
- `TokenSequenceLowercase`: This converts the text in each token in the token sequence in the data field to lower case
- `TokenSequence2FeatureSequence`: This converts the token sequence in the data field of each instance to a feature sequence
- `TokenSequenceNGrams`: This converts the token sequence in the data field to a token sequence of ngrams, that is, combination of two or more words

Note

The full list of processing steps is available in the following Mallet documentation:

<http://mallet.cs.umass.edu/api/index.html?cc/mallet/pipe/iterator/package-tree.html>

Now we are ready to build a class that will import our data.

First, let's build a pipeline, where each processing step is denoted as a pipeline in Mallet. Pipelines can be wired together in a serial fashion with a list of `ArrayList<Pipe>` objects:

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
```

Begin by reading data from a file object and converting all the characters into lower case:

```
pipeList.add(new Input2CharSequence("UTF-8"));
```

```
pipeList.add( new CharSequenceLowercase() );
```

Next, tokenize raw strings with a regular expression. The following pattern includes Unicode letters and numbers and the underscore character:

```
Pattern tokenPattern =  
    Pattern.compile("[\\p{L}\\p{N}_]+");  
  
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
```

Remove stop words, that is, frequent words with no predictive power, using a standard English stop list. Two additional parameters indicate whether stop word removal should be case-sensitive and mark deletions instead of just deleting the words. We'll set both of them to `false`:

```
pipeList.add(new TokenSequenceRemoveStopwords(false, false));
```

Instead of storing the actual words, we can convert them into integers, indicating a word index in the bag of words:

```
pipeList.add(new TokenSequence2FeatureSequence());
```

We'll do the same for the class label; instead of label string, we'll use an integer, indicating a position of the label in our bag of words:

```
pipeList.add(new Target2Label());
```

We could also print the features and the labels by invoking the `PrintInputAndTarget` pipe:

```
pipeList.add(new PrintInputAndTarget());
```

Finally, we store the list of pipelines in a `SerialPipes` class that will covert an instance through a sequence of pipes:

```
SerialPipes pipeline = new SerialPipes(pipeList);
```

Now let's take a look at how apply this in a text mining application!

Topic modeling for BBC news

As discussed earlier, the goal of topic modeling is to identify patterns in a text corpus that correspond to document topics. In this example, we will use a dataset originating from BBC news. This dataset is one of the standard benchmarks in machine learning research, and is available for non-commercial and research purposes.

The goal is to build a classifier that is able to assign a topic to an uncategorized document.

BBC dataset

Greene and Cunningham (2006) collected the BBC dataset to study a particular document-clustering challenge using support vector machines. The dataset consists of 2,225 documents from the BBC News website from 2004 to 2005, corresponding to the stories collected from five topical areas: business, entertainment, politics, sport, and tech. The dataset can be grabbed from the following website:

<http://mlg.ucd.ie/datasets/bbc.html>

Download the raw text files under the **Dataset: BBC** section. You will also notice that the website contains already processed dataset, but for this example, we want to process the dataset by ourselves. The ZIP contains five folders, one per topic. The actual documents are placed in the corresponding topic folder, as shown in the following screenshot:

The screenshot shows a website interface for 'Insight Resources'. The top navigation bar includes links for 'Insight Resources', 'Home', 'Datasets' (which is highlighted in teal), 'Software', 'Publications', and 'Insight Home'. Below the navigation, the main content area has a teal header 'BBC Datasets'. The text describes two news article datasets from BBC News, available for non-commercial research purposes. It cites a publication by D. Greene and P. Cunningham from ICML 2006, with links to PDF and BibTeX files. Under the 'Dataset: BBC' heading, it states that all rights are owned by the BBC and lists two bullet points: 'Consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005.' and 'Class Labels: 5 (business, entertainment, politics, sport, tech)'. There are two download links: '>> Download pre-processed dataset' and '>> Download raw text files'. A similar section for 'Dataset: BBCSport' is partially visible below it.

Now, let's build a topic classifier.

Modeling

Start by importing the dataset and processing the text:

```
import cc.mallet.types.*;
import cc.mallet.pipe.*;
import cc.mallet.pipe.iterator.*;
import cc.mallet.topics.*;

import java.util.*;
import java.util.regex.*;
import java.io.*;

public class TopicModeling {

    public static void main(String[] args) throws Exception {

String dataFolderPath = args[0];
String stopListFilePath = args[1];
```

Create a default pipeline as previously described:

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
pipeList.add(new Input2CharSequence("UTF-8"));
Pattern tokenPattern = Pattern.compile("[\\p{L}\\p{N}_]+");
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
pipeList.add(new TokenSequenceLowercase());
pipeList.add(new TokenSequenceRemoveStopwords(new
File(stopListFilePath), "utf-8", false, false, false));
pipeList.add(new TokenSequence2FeatureSequence());
pipeList.add(new Target2Label());
SerialPipes pipeline = new SerialPipes(pipeList);
```

Next, initialize folderIterator:

```
FileIterator folderIterator = new FileIterator(
    new File[] {new File(dataFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

Construct a new instance list with the pipeline that we want to use to process the text:

```
InstanceList instances = new InstanceList(pipeline);
```

Finally, process each instance provided by the iterator:

```
instances.addThruPipe(folderIterator);
```

Now let's create a model with five topics using the `cc.mallet.topics.ParallelTopicModel` class that implements a simple threaded **Latent Dirichlet Allocation (LDA)** model. LDA is a common method for topic modeling that uses Dirichlet distribution to estimate the probability that a selected topic generates a particular document. We will not dive deep into the details in this chapter; the reader is referred to the original paper by D. Blei et al. (2003). Note that there is another classification algorithm in machine learning with the same acronym that refers to **Linear Discriminant Analysis (LDA)**. Beside the common acronym, it has nothing in common with the LDA model.

The class is instantiated with parameters alpha and beta, which can be broadly interpreted, as follows:

- High alpha value means that each document is likely to contain a mixture of most of the topics, and not any single topic specifically. A low alpha value puts less of such constraints on documents, and this means that it is more likely that a document may contain mixture of just a few, or even only one, of the topics.
- A high beta value means that each topic is likely to contain a mixture of most of the words, and not any word specifically; while a low value means that a topic may contain a mixture of just a few of the words.

In our case, we initially keep both parameters low ($\alpha_t = 0.01$, $\beta_w = 0.01$) as we assume topics in our dataset are not mixed much and there are many words for each of the topics:

```
int numTopics = 5;
ParallelTopicModel model =
new ParallelTopicModel(numTopics, 0.01, 0.01);
```

Next, add instances to the model, and as we are using parallel implementation, specify the number of threads that will run in parallel, as follows:

```
model.addInstances(instances);
model.setNumThreads(4);
```

Run the model for a selected number of iterations. Each iteration is used for better estimation of internal LDA parameters. For testing, we can use a small number of iterations, for example, 50; while in real applications, use 1000 or 2000 iterations.

Finally, call the `void estimate()` method that will actually build an LDA model:

```
model.setNumIterations(1000);  
model.estimate();
```

The model outputs the following result:

```
0 0,06654 game england year time win world 6  
1 0,0863 year 1 company market growth economy firm  
2 0,05981 people technology mobile mr games users music  
3 0,05744 film year music show awards award won  
4 0,11395 mr government people labour election party blair  
  
[beta: 0,11328]  
<1000> LL/token: -8,63377  
  
Total time: 45 seconds
```

`LL/token` indicates the model's log-likelihood, divided by the total number of tokens, indicating how likely the data is given the model. Increasing values mean the model is improving.

The output also shows the top words describing each topic. The words correspond to initial topics really well:

- **Topic 0:** game, England, year, time, win, world, → sport
- **Topic 1:** year, 1, company, market, growth, economy, firm → finance
- **Topic 2:** people, technology, mobile, mr, games, users, music → tech
- **Topic 3:** film, year, music, show, awards, award, won → entertainment
- **Topic 4:** mr, government, people, labor, election, party, blair → politics

There are still some words that don't make much sense, for instance, mr, 1, and 6. We could include them in the stop word list. Also, some words appear twice, for example, award and awards. This happened because we didn't apply any stemmer or lemmatization pipe.

In the next section, we'll take a look to check whether the model is of any good.

Evaluating a model

As statistical topic modeling has unsupervised nature, it makes model selection difficult. For some applications, there may be some extrinsic tasks at hand, such as information retrieval or document classification, for which performance can be evaluated. However, in general, we want to estimate the model's ability to generalize topics regardless of the task.

Wallach et al. (2009) introduced an approach that measures the quality of a model by computing the log probability of held-out documents under the model. Likelihood of unseen documents can be used to compare models—higher likelihood implies a better model.

First, let's split the documents into training and testing set (that is, held-out documents), where we use 90% for training and 10% for testing:

```
// Split dataset
InstanceList[] instanceSplit= instances.split(new Randoms(), new
double[] {0.9, 0.1, 0.0});
```

Now, let's rebuild our model using only 90% of our documents:

```
// Use the first 90% for training
model.addInstances(instanceSplit[0]);
model.setNumThreads(4);
model.setNumIterations(50);
model.estimate();
```

Next, initialize an estimator that implements Wallach's log probability of held-out documents, MarginalProbEstimator:

```
// Get estimator
MarginalProbEstimator estimator = model.getProbEstimator();
```

Note

An intuitive description of LDA is summarized by Annalyn Ng in her blog:

<https://annalyzin.wordpress.com/2015/06/21/laymans-explanation-of-topic-modeling-with-lda-2/>

To get deeper insight into the LDA algorithm, its components, and its working, take a

look at the original paper LDA by David Blei et al. (2003) at <http://jmlr.csail.mit.edu/papers/v3/blei03a.html> or take a look at the summarized presentation by D. Santhanam of Brown University at http://www.cs.brown.edu/courses/csci2950-p/spring2010/lectures/2010-03-03_santhanam.pdf.

The class implements many estimators that require quite deep theoretical knowledge of how the LDA method works. We'll pick the left-to-right evaluator, which is appropriate for a wide range of applications, including text mining, speech recognition, and others. The left-to-right evaluator is implemented as the `double evaluateLeftToRight` method, accepting the following components:

- Instances `heldOutDocuments`: This test the instances
- int `numParticles`: This algorithm parameter indicates the number of left-to-right tokens, where default value is 10
- boolean `useResampling`: This states whether to resample topics in left-to-right evaluation; resampling is more accurate, but leads to quadratic scaling in the length of documents
- PrintStream `docProbabilityStream`: This is the file or `stdout` in which we write the inferred log probabilities per document

Let's run the estimator, as follows:

```
double loglike = estimator.evaluateLeftToRight(  
    instanceSplit[1], 10, false, null);)  
System.out.println("Total log likelihood: "+loglike);
```

In our particular case, the estimator outputs the following log likelihood, which makes sense when it is compared to other models that are either constructed with different parameters, pipelines, or data—the higher the log likelihood, the better the model is:

```
Total time: 3 seconds  
Topic Evaluator: 5 topics, 3 topic bits, 111 topic mask  
Total log likelihood: -360849.4240795393  
Total log likelihood
```

Now let's take a look at how to make use of this model.

Reusing a model

As we are usually not building models on the fly, it often makes sense to train a model once and use it repeatedly to classify new data.

Note that if you'd like to classify new documents, they need go through the same pipeline as other documents—the pipe needs to be the same for both training and classification. During training, the pipe's data alphabet is updated with each training instance. If you create a new pipe with the same steps, you don't produce the same pipeline as its data alphabet is empty. Therefore, to use the model on new data, save/load the pipe along with the model and use this pipe to add new instances.

Saving a model

Mallet supports a standard method for saving and restoring objects based on serialization. We simply create a new instance of `ObjectOutputStream` class and write the object into a file as follows:

```
String modelPath = "myTopicModel";

//Save model
ObjectOutputStream oos = new ObjectOutputStream(
new FileOutputStream (new File(modelPath+".model")));
oos.writeObject(model);
oos.close();

//Save pipeline
oos = new ObjectOutputStream(
new FileOutputStream (new File(modelPath+".pipeline")));
oos.writeObject(pipeline);
oos.close();
```

Restoring a model

Restoring a model saved through serialization is simply an inverse operation using the `ObjectInputStream` class:

```
String modelPath = "myTopicModel";

//Load model
ObjectInputStream ois = new ObjectInputStream(
new FileInputStream (new File(modelPath+".model")));
ParallelTopicModel model = (ParallelTopicModel) ois.readObject();
```

```
ois.close();

// Load pipeline
ois = new ObjectInputStream(
    new FileInputStream (new File(modelPath+".pipeline")));
SerialPipes pipeline = (SerialPipes) ois.readObject();
ois.close();
```

We discussed how to build an LDA model to automatically classify documents into topics. In the next example, we'll look into another text mining problem—text classification.

E-mail spam detection

Spam or electronic spam refers to unsolicited messages, typically carrying advertising content, infected attachments, links to phishing or malware sites, and so on. While the most widely recognized form of spam is e-mail spam, spam abuses appear in other media as well: website comments, instant messaging, Internet forums, blogs, online ads, and so on.

In this chapter, we will discuss how to build naive Bayesian spam filtering, using bag-of-words representation to identify spam e-mails. The naive Bayes spam filtering is one of the basic techniques that was implemented in the first commercial spam filters; for instance, **Mozilla Thunderbird** mail client uses native implementation of such filtering. While the example in this chapter will use e-mail spam, the underlying methodology can be applied to other type of text-based spam as well.

E-mail spam dataset

Androutsopoulos et al. (2000) collected one of the first e-mail spam datasets to benchmark spam-filtering algorithms. They studied how the naive Bayes classifier can be used to detect spam, if additional pipes such as stop list, stemmer, and lemmatization contribute to better performance. The dataset was reorganized by Andrew Ng in OpenClassroom's machine learning class, available for download at <http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex6/ex6.html>.

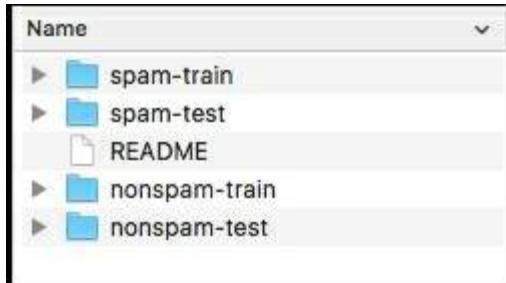
Select and download the second option, `ex6DataEmails.zip`, as shown in the following image:

The screenshot shows the OpenClassroom interface for the Machine Learning course by Andrew Ng. At the top, there is a navigation bar with the course title "Machine Learning" and the author's name "Andrew Ng". Below the title, there is a book icon labeled "Machine Learning". The main content area features a section titled "Exercise 6: Naive Bayes". A descriptive text explains that the exercise involves using Naive Bayes to classify email messages into spam and nonspam groups, using a preprocessed subset of the Ling-Spam Dataset. It mentions that the dataset is based on 960 real email messages from a linguistics mailing list. Two options are provided for completing the exercise: using Matlab/Octave-formatted features or generating features yourself. The "RESOURCES" sidebar on the right includes links to "Syllabus", "FAQ", and "Credits/Acknowledgments".

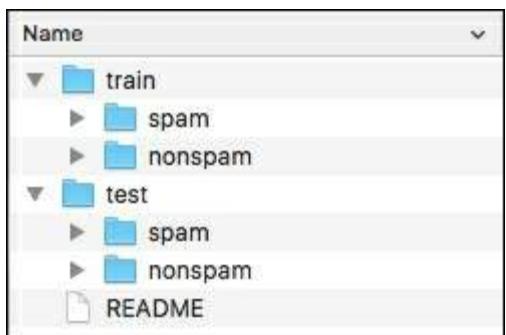
The ZIP contains four folders (Ng, 2015):

- The `nonspam-train` and `spam-train` folders contain the pre-processed e-mails that you will use for training. They have 350 e-mails each.
- The `nonspam-test` and `spam-test` folders constitute the test set, containing 130 spam and 130 nonspam e-mails. These are the documents that you will make predictions on. Notice that even though separate folders tell you the correct labeling, you should make your predictions on all the test documents without this knowledge. After you make your predictions, you can use the correct labeling to check whether your classifications were correct.

To leverage Mallet's folder iterator, let's reorganize the folder structure as follows. Create two folders, `train` and `test`, and put the `spam/nospam` folders under the corresponding folders. The initial folder structure is as shown in the following image:



The final folder structure will be as shown in the following image:



The next step is to transform e-mail messages to feature vectors.

Feature generation

Create a default pipeline as described previously:

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
pipeList.add(new Input2CharSequence("UTF-8"));
Pattern tokenPattern = Pattern.compile("[\\p{L}\\p{N}_]+");
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
pipeList.add(new TokenSequenceLowercase());
pipeList.add(new TokenSequenceRemoveStopwords(new
File(stopListFilePath), "utf-8", false, false, false));
pipeList.add(new TokenSequence2FeatureSequence());
pipeList.add(new FeatureSequence2FeatureVector());
pipeList.add(new Target2Label());
SerialPipes pipeline = new SerialPipes(pipeList);
```

Note that we added an additional `FeatureSequence2FeatureVector` pipe that transforms a feature sequence into a feature vector. When we have data in a feature vector, we can use any classification algorithm as we saw in the previous chapters. We'll continue our example in Mallet to demonstrate how to build a classification model.

Next, initialize a folder iterator to load our examples in the `train` folder comprising e-mail examples in the `spam` and `nonspam` subfolders, which will be used as example labels:

```
FileIterator folderIterator = new FileIterator(
    new File[] {new File(dataFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

Construct a new instance list with the pipeline that we want to use to process the text:

```
InstanceList instances = new InstanceList(pipeline);
```

Finally, process each instance provided by the iterator:

```
instances.addThruPipe(folderIterator);
```

We have now loaded the data and transformed it into feature vectors. Let's train our model on the training set and predict the `spam/nonspam` classification on the `test` set.

Training and testing

Mallet implements a set of classifiers in the `cc.mallet.classify` package, including decision trees, naive Bayes, AdaBoost, bagging, boosting, and many others. We'll start with a basic classifier, that is, a naive Bayes classifier. A classifier is initialized by the `ClassifierTrainer` class, which returns a classifier when we invoke its `train(Instances)` method:

```
ClassifierTrainer classifierTrainer = new NaiveBayesTrainer();
Classifier classifier = classifierTrainer.train(instances);
```

Now let's see how this classifier works and evaluate its performance on a separate dataset.

Model performance

To evaluate the classifier on a separate dataset, let's start by importing the e-mails located in our `test` folder:

```
InstanceList testInstances = new
InstanceList(classifier.getInstancePipe());
folderIterator = new FileIterator(
    new File[] {new File(testFolderPath)},
    new TxtFilter(),
    FileIterator.LAST_DIRECTORY);
```

We will pass the data through the same pipeline that we initialized during training:

```
testInstances.addThruPipe(folderIterator);
```

To evaluate classifier performance, we'll use the `cc.mallet.classify.Trial` class, which is initialized with a classifier and set of test instances:

```
Trial trial = new Trial(classifier, testInstances);
```

The evaluation is performed immediately at initialization. We can then simply take out the measures that we care about. In our example, we'd like to check the precision and recall on classifying spam e-mail messages, or F-measure, which returns a harmonic mean of both values, as follows:

```
System.out.println(
    "F1 for class 'spam': " + trial.getF1("spam"));
System.out.println()
```

```
"Precision:" + trial.getPrecision(1));
System.out.println(
    "Recall:" + trial.getRecall(1));
```

The evaluation object outputs the following results:

```
F1 for class 'spam': 0.9731800766283524
Precision: 0.9694656488549618
Recall: 0.9769230769230769
```

The results show that the model correctly discovers 97.69% of spam messages (recall), and when it marks an e-mail as spam, it is correct in 96.94% cases. In other words, it misses approximately 2 per 100 spam messages and marks 3 per 100 valid messages as spam. Not really perfect, but it is more than a good start!

Summary

In this chapter, we discussed how text mining is different from traditional attribute-based learning, requiring a lot of pre-processing steps in order to transform written natural language into feature vectors. Further, we discussed how to leverage Mallet, a Java-based library for natural language processing by applying it to two real life problems. First, we modeled topics in news corpus using the LDA model to build a model that is able to assign a topic to new document. We also discussed how to build a naive Bayesian spam-filtering classifier using the bag-of-words representation.

This chapter concludes the technical demonstrations of how to apply various libraries to solve machine learning tasks. As we were not able to cover more interesting applications and give further details at many points, the next chapter gives some further pointers on how to continue learning and dive deeper into particular topics.

Chapter 11. What is Next?

This chapter brings us to the end of our journey of reviewing machine learning Java libraries and discussing how to leverage them to solve real-life problems. However, this should not be the end of your journey by all means. This chapter will give you some practical advice on how to start deploying your models in the real world, what are the catches, and where to go to deepen your knowledge. It also gives you further pointers about where to find additional resources, materials, venues, and technologies to dive deeper into machine learning.

This chapter will cover the following topics:

- Important aspects of machine learning in real life
- Standards and markup languages
- Machine learning in the cloud
- Web resources and competitions

Machine learning in real life

Papers, conference presentations, and talks often don't discuss how the models were actually deployed and maintained in production environment. In this section, we'll look into some aspects that should be taken into consideration.

Noisy data

In practice, data typically contains errors and imperfections due to various reasons such as measurement errors, human mistakes, errors of expert judgment in classifying training examples, and so on. We refer to all of these as noise. Noise can also come from the treatment of missing values when an example with unknown attribute value is replaced by a set of weighted examples corresponding to the probability distribution of the missing value. The typical consequences of noise in learning data are low prediction accuracy of learned model in new data and complex models that are hard to interpret and to understand by the user.

Class unbalance

Class unbalance is a problem we come across in [Chapter 7, Fraud and Anomaly Detection](#), where the goal was to detect fraudulent insurance claims. The challenge is that a very large part of the dataset, usually more than 90%, describes normal activities and only a small fraction of the dataset contains fraudulent examples. In such a case, if the model always predicts normal, then it is correct 90% of the time. This problem is extremely common in practice and can be observed in various applications, including fraud detection, anomaly detection, medical diagnosis, oil spillage detection, facial recognition, and so on.

Now knowing what the class unbalance problem is and why is it a problem, let's take a look at how to deal with this problem. The first approach is to focus on measures other than classification accuracy, such as recall, precision, and f-measure. Such measures focus on how accurate a model is at predicting minority class (recall) and what is the share of false alarms (precision). The other approach is based on resampling, where the main idea is to reduce the number of overrepresented examples in such way that the new set contains a balanced ratio of both the classes.

Feature selection is hard

Feature selection is arguably the most challenging part of modeling that requires domain knowledge and good insights into the problem at hand. Nevertheless, properties of well-behaved features are as follows:

- **Reusability:** Features should be available for reuse in different models, applications, and teams
- **Transformability:** You should be able to transform a feature with an operation, for example, `log()`, `max()`, or combine multiple features together with a custom calculation
- **Reliability:** Features should be easy to monitor and appropriate unit tests should exist to minimize bugs/issues
- **Interpretability:** In order to perform any of the previous actions, you need to be able to understand the meaning of features and interpret their values

The better you are able to capture the features, the more accurate your results will be.

Model chaining

Some models might produce an output, which is used as the feature in another model. Moreover, we can use multiple models—ensembles—turning any model into a feature. This is a great way to get better results, but this can lead to problems too. Care must be taken that the output of your model is ready to accept dependencies. Also, try to avoid feedback loops, as they can create dependencies and bottlenecks in pipeline.

Importance of evaluation

Another important aspect is model evaluation. Unless you apply your models to actual new data and measure a business objective, you're not doing predictive analytics. Evaluation techniques, such as cross-validation and separated train/test set, simply split your test data, which can give only you an estimate of how your model will perform. Life often doesn't hand you a train dataset with all the cases defined, so there is a lot of creativity involved in defining these two sets in a real-world dataset.

At the end of the day, we want to improve a business objective, such as improve ad conversion rate, get more clicks on recommended items, and so on. To measure the improvement, execute A/B tests, measure differences in metrics across statistically identical populations that each experience a different algorithm. Decisions on the product are always data-driven.

Note

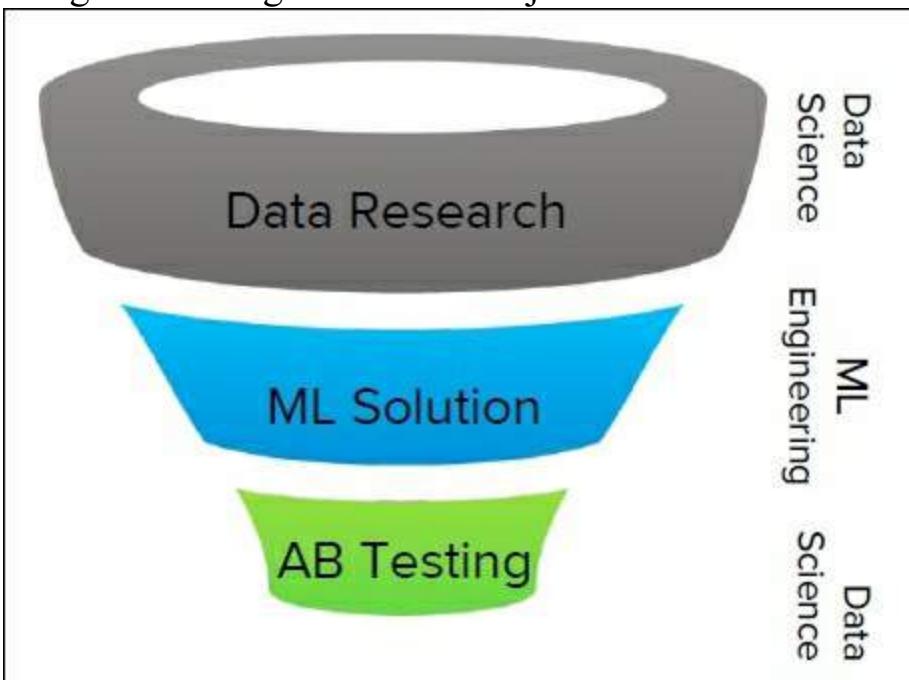
A/B testing is a method for a randomized experiment with two variants: A, which corresponds to the original version, controlling the experiment; and B, which corresponds to a variation. The method can be used to determine whether the variation outperforms the original version. It can be used to test everything from website changes to sales e-mails to search ads.

Udacity offers a free course, covering design and analysis of A/B tests at <https://www.udacity.com/course/ab-testing--ud257>.

Getting models into production

The path from building an accurate model in a lab to deploying it in a product involves collaboration of data science and engineering, as shown in the following three steps and diagram:

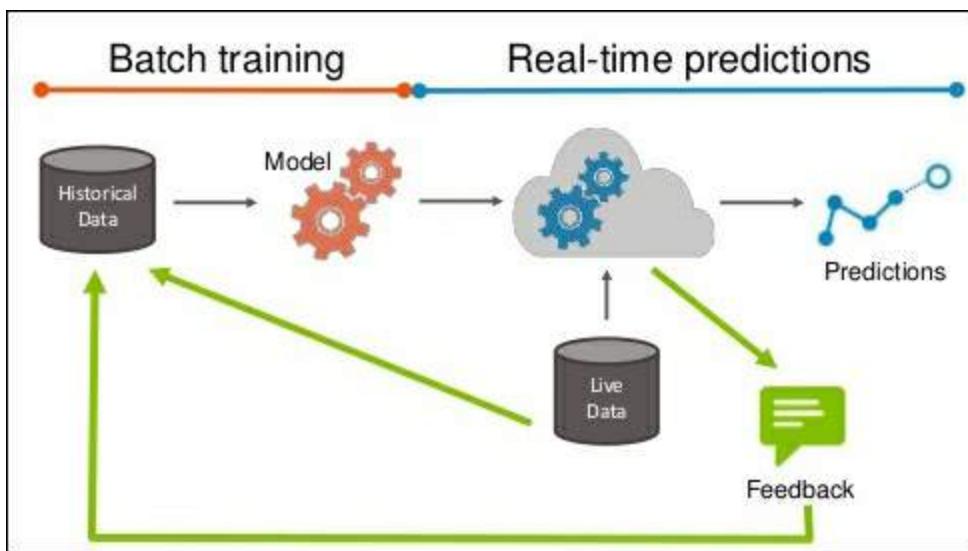
1. **Data research and hypothesis building** involves modeling the problem and executing initial evaluation.
2. **Solution building and implementation** is where your model finds its way into the product flow by rewriting it into more efficient, stable, and scalable code.
3. **Online evaluation** is the last stage where the model is evaluated with live data using A/B testing on business objectives.



Model maintenance

Another aspect that we need to address is how the model will be maintained. Is this a model that will not change over time? Is it modeling a dynamic phenomenon requiring the model to adjust its prediction over time?

The model is usually built in an of offline batch training and then used on live data to serve predictions as shown in the following figure. If we are able to receive feedback on model predictions; for instance, whether the stock went up as model predicted, whether the candidate responded to campaign, and so on, the feedback should be used to improve the initial model.



The feedback could be really useful to improve the initial model, but make sure to pay attention to the data you are sampling. For instance, if you have a model that predicts who will respond to a campaign, you will initially use a set of randomly contacted clients with specific responded/not responded distribution and feature properties. The model will focus only on a subset of clients that will most likely respond and your feedback will return you a subset of clients that responded. By including this data, the model is more accurate in a specific subgroup, but might completely miss some other group. We call this problem exploration versus exploitation. Some approaches to address this problem can be found in Osugi et al (2005) and Bondu et al (2010).

Standards and markup languages

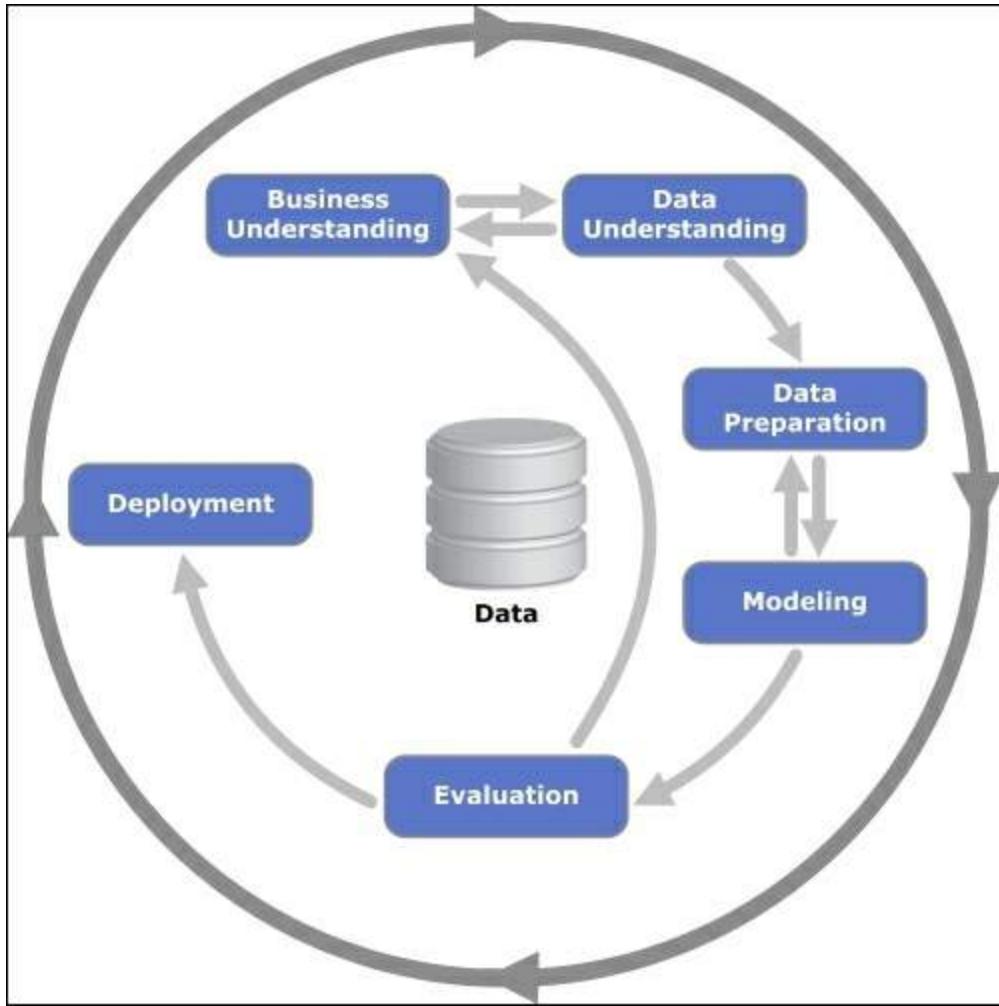
As predictive models become more pervasive, the need for sharing the models and completing the modeling process leads to formalization of development process and interchangeable formats. In this section, we'll review two de facto standards, one covering data science processes and the other specifying an interchangeable format for sharing models between applications.

CRISP-DM

Cross Industry Standard Process for Data Mining (CRISP-DM) describing a data mining process commonly used by data scientists in industry. CRISP-DM breaks the data mining science process into the following six major phases:

- **Business understanding**
- **Data understanding**
- **Data preparation**
- **Modeling**
- **Evaluation**
- **Deployment**

In the following diagram, the arrows indicate the process flow, which can move back and forth through the phases. Also, the process doesn't stop with model deployment. The outer arrow indicates the cyclic nature of data science. Lessons learned during the process can trigger new questions and repeat the process while improving previous results:



SEMMA methodology

Another methodology is **Sample, Explore, Modify, Model, and Assess (SEMMA)**. SEMMA describes the main modeling tasks in data science, while leaving aside business aspects such as data understanding and deployment. SEMMA was developed by SAS institute, which is one of the largest vendors of statistical software, aiming to help the users of their software to carry out core tasks of data mining.

Predictive Model Markup Language

Predictive Model Markup Language (PMML) is an XML-based interchange format that allows machine learning models to be easily shared between applications and systems. Supported models include logistic regression, neural networks, decision trees, naïve Bayes, regression models, and many others. A typical PMML file consists of the following sections:

- Header containing general information
- Data dictionary, describing data types
- Data transformations, specifying steps for normalization, discretization, aggregations, or custom functions
- Model definition, including parameters
- Mining schema listing attributes used by the model
- Targets allowing post-processing of the predicted results
- Output listing fields to be outputted and other post-processing steps

The generated PMML files can be imported to any PMML-consuming application, such as Zementis **Adaptive Decision and Predictive Analytics (ADAPA)** and **Universal PMML Plug-in (UPPI)** scoring engines; Weka, which has built-in support for regression, general regression, neural network, **TreeModel**, **RuleSetModel**, and **Support Vector Machine (SVM)** model; Spark, which can export k-means clustering, linear regression, ridge regression, lasso model, binary logistic model, and SVM; and cascading, which can transform PMML files into an application on Apache Hadoop.

The next generation of PMML is an emerging format called **Portable Format for Analytics (PFA)**, providing a common interface to deploy the complete workflows across environments.

Machine learning in the cloud

Setting up a complete machine learning stack that is able to scale with the increasing amount of data could be challenging. Recent wave of **Software as a Service (SaaS)** and **Infrastructure as a Service (IaaS)** paradigm was spilled over to machine learning domain as well. The trend today is to move the actual data preprocessing, modeling, and prediction to cloud environments and focus on modeling task only.

In this section, we'll review some of the promising services offering algorithms, predictive models already train in specific domain, and environments empowering collaborative workflows in data science teams.

Machine learning as a service

The first category is algorithms as a service, where you are provided with an API or even graphical user interface to connect pre-programmed components of data science pipeline together:

- **Google Prediction API** was one of the first companies that introduced prediction services through its web API. The service is integrated with **Google Cloud Storage** serving as data storage. The user can build a model and call an API to get predictions.
- **BigML** implements a user-friendly graphical interface, supports many storage providers (for instance, Amazon S3) and offers a wide variety of data processing tools, algorithms, and powerful visualizations.
- **Microsoft Azure Machine Learning** provides a large library of machine learning algorithms and data processing functions, as well as graphical user interface, to connect these components to an application. Additionally, it offers a fully-managed service that you can use to deploy your predictive models as ready-to-consume web services.
- **Amazon Machine Learning** entered the market quite late. It's main strength is seamless integration with other Amazon services, while the number of algorithms and user interface needs further improvements.
- **IBM Watson Analytics** focuses on providing models that are already hand-crafted to a particular domain such as speech recognition, machine translations, and anomaly detection. It targets a wide range of industries by solving specific use cases.
- **Prediction.IO** is a self-hosted open source platform, providing the full stack from data storage to modeling to serving the predictions. Prediciton.IO can talk to Apache Spark to leverage its learning algorithms. In addition, it is shipped with a wide variety of models targeting specific domains, for instance, recommender system, churn prediction, and others.

Predictive API is an emerging new field, so these are just some of the well-known examples; **KDnuggets** compiled a list of 50 machine learning APIs at <http://www.kdnuggets.com/2015/12/machine-learning-data-science-apis.html>.

Note

To learn more about it, you can visit PAPI, the International Conference on Predictive APIs and Apps at <http://www.papi.io> or take a look at a book by *Louis*

Dorard, Bootstrapping Machine Learning (L. Dorard, 2014).

Web resources and competitions

In this section, we'll review where to find additional resources for learning, discussing, presenting, or sharpening our data science skills.

Datasets

One of the most well-known repositories of machine learning datasets is hosted by the University of California, Irvine. The UCI repository contains over 300 datasets covering a wide variety of challenges, including poker, movies, wine quality, activity recognition, stocks, taxi service trajectories, advertisements, and many others. Each dataset is usually equipped with a research paper where the dataset was used, which can give you a hint on how to start and what is the prediction baseline.

The UCI machine learning repository can be accessed at <https://archive.ics.uci.edu>, as follows:

The screenshot shows the homepage of the UC Irvine Machine Learning Repository. At the top, there's a navigation bar with links for 'About', 'Citation Policy', 'Donate a Data Set', and 'Contact'. Below the navigation is a banner with the text 'Loading' and a link 'View ALL Data Sets'. The main content area features a large 'UCI Machine Learning Repository' logo with a stylized brain icon. Below the logo, it says 'Center for Machine Learning and Intelligent Systems'. A welcome message reads 'Welcome to the UC Irvine Machine Learning Repository!'. A note below it states: 'We currently maintain 335 data sets as a service to the machine learning community. You may [view all data sets](#) through our searchable interface. Our [old web site](#) is still available, for those who prefer the old format. For a general overview of the Repository, please visit our [About page](#). For information about citing data sets in publications, please read our [citation policy](#). If you wish to donate a data set, please consult our [donation policy](#). For any other questions, feel free to [contact the Repository librarians](#). We have also set up a [mirror site](#) for the Repository.'

Supported By: In Collaboration With:

Latest News:	Newest Data Sets:	Most Popular Data Sets (hits since 2007):
<p>2013-04-04: Welcome to the new Repository admins Kevin Bache and Moshe Lichman!</p> <p>2010-03-01: Note from donor regarding Netflix data</p> <p>2009-10-16: Two new data sets have been added.</p> <p>2009-09-14: Several data sets have been added.</p> <p>2008-07-23: Repository mirror has been set up.</p> <p>2008-03-24: New data sets have been added!</p> <p>2007-06-25: Two new data sets have been added: UJI Pen Characters, MAGIC Gamma Telescope</p>	<p>2015-10-26: Heterogeneity Activity Recognition</p> <p>2015-09-24: Educational Process Mining (EPM): A Learning Analytics Data Set</p> <p>2015-09-10: UJIIndoorLoc-Mag</p> <p>2015-08-04: Mice Protein Expression</p> <p>2015-07-29: Smartphone-Based Recognition of Human Activities and Postural Transitions</p> <p>2015-07-27: Cuff-Less Blood Pressure Estimation</p> <p> Taxi Service Trajectories</p>	<p>853437: Iris</p> <p>604861: Adult</p> <p>487962: Wine</p> <p>416656: Car Evaluation</p> <p>383128: Breast Cancer Wisconsin (Diagnostic)</p> <p>324668: Abalone</p> <p>291972: Wine Quality</p>

Another well-maintained collection by Xiaming Chen is hosted on GitHub:

<https://github.com/caesar0301/awesome-public-datasets>

The Awesome Public Datasets repository maintains links to more than 400 data sources from a variety of domains, ranging from agriculture, biology, economics, psychology, museums, and transportation. Datasets, specifically targeting machine learning, are collected under the image processing, machine learning, and data challenges sections.

Online courses

Learning how to become a data scientist has became much more accessible due to the availability of online courses. The following is a list of free resources to learn different skills online:

- Online courses for learning Java:
 - **Udemy: Learn Java Programming From Scratch** at
<https://www.udemy.com/learn-java-programming-from-scratch>
 - **Udemy: Java Tutorial for Complete Beginners** at
<https://www.udemy.com/java-tutorial>
 - **LearnJAvAOnline.org: Interactive Java tutorial** at
<http://www.learnjavaonline.org/>
- Online courses to learn more about machine learning:
 - **Coursera: Machine Learning (Stanford) by Andrew Ng**: This teaches you the math behind many the machine learning algorithms, explains how they work, and explores why they make sense at
<https://www.coursera.org/learn/machine-learning>.
 - **Statistics 110 (Harvard) by Joe Blitzstein**: This course lets you discover the probability of related terms that you will hear many times in your data science journey. Lectures are available on YouTube at
<http://projects.iq.harvard.edu/stat110/youtube>.
 - **Data Science CS109 (Harvard) by John A. Paulson**: This is a hands-on course where you'll learn about Python libraries for data science, as well as how to handle machine-learning algorithms at <http://cs109.github.io/2015/>.

Competitions

The best way to sharpen your knowledge is to work on real problems; and if you want to build a proven portfolio of your projects, machine learning competitions are a viable place to start:

- **Kaggle:** This is the number one competition platform, hosting a wide variety of challenges with large prizes, strong data science community, and lots of helpful resources. You can check it out at <https://www.kaggle.com/>.
- **CrowdANALYTIX:** This is a crowdsourced data analytics service that is focused on the life sciences and financial services industries at <https://www.crowdanalytix.com>.
- **DrivenData:** This hosts data science competitions for social good at <http://www.drivendata.org/>.

Websites and blogs

In addition to online courses and competitions, there are numerous websites and blogs publishing the latest developments in the data science community, their experience in attacking different problems, or just best practices. Some good starting points are as follows:

- **KDnuggets:** This is the de facto portal for data mining, analytics, big data, and data science, covering the latest news, stories, events, and other relevant issues at <http://www.kdnuggets.com/>.
- **Machine learning mastery:** This is an introductory-level blog with practical advice and pointers where to start. Check it out at <http://machinelearningmastery.com/>.
- **Data Science Central:** This consists of practical community articles on a variety of topics, algorithms, caches, and business cases at <http://www.datasciencecentral.com/>.
- **Data Mining Research** by Sandro Saitta at <http://www.dataminingblog.com/>.
- **Data Mining: Text Mining, Visualization and Social Media** by Matthew Hurst, covering interesting text and web mining topics, frequently with applications to Bing and Microsoft at http://datamining.typepad.com/data_mining/.
- **Geeking with Greg** by Greg Linden, inventor of Amazon recommendation engine and Internet entrepreneur. You can check it out at <http://glinden.blogspot.si/>.
- **DSGuide:** This is a collection of over 150 data science blogs at <http://dsguide.biz/reader/sources>.

Venues and conferences

The following are a few top-tier academic conferences with the latest algorithms:

- **Knowledge Discovery in Databases (KDD)**
- **Computer Vision and Pattern Recognition (CVPR)**
- **Annual Conference on Neural Information Processing Systems (NIPS)**
- **International Conference on Machine Learning (ICML)**
- **IEEE International Conference on Data Mining (ICDM)**
- **International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)**
- **International Joint Conference on Artificial Intelligence (IJCAI)**

Some business conferences are as follows:

- O'Reilly Strata Conference
- The Strata + Hadoop World Conferences
- Predictive Analytics World
- MLconf

You can also check local meetup groups.

Summary

In this chapter, we concluded the book by discussing some aspects of model deployment, we also looked into standards for data science process and interchangeable predictive model format PMML. We also reviewed online courses, competitions, web resources, and conferences that could help you in your journey towards mastering the art of machine learning.

I hope this book inspired you to dive deeper into data science and has motivated you to get your hands dirty, experiment with various libraries and get a grasp of how different problems could be attacked. Remember, all the source code and additional resources are available at the supplementary website <http://www.machine-learning-in-java.com>.

Appendix A. References

The following are the references for all the citations throughout the book:

- Adomavicius, G. and Tuzhilin, A.. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734-749. 2005.
- Bengio, Y.. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* 2(1), 1-127. 2009. Retrieved from <http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf>.
- Blei, D. M., Ng, A. Y., and Jordan, M. I.. Latent dirichlet allocation. *Journal of Machine Learning Research*. 3, 993–1022. 2003. Retrieved from: <http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf>.
- Bondu, A., Lemaire, V., Boulle, M.. Exploration vs. exploitation in active learning: A Bayesian approach. *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain. 2010.
- Breunig, M. M., Kriegel, H.-P., Ng, R. T., Sander, J.. LOF: Identifying Density-based Local Outliers (PDF). *Proceedings from the 2000 ACM SIGMOD International Conference on Management of Data*, 29(2), 93–104. 2000
- Campbell, A. T. (n.d.). Lecture 21 - Activity Recognition. Retrieved from <http://www.cs.dartmouth.edu/~campbell/cs65/lecture22/lecture22.html>.
- Chandra, N.S. Unraveling the Customer Mind. 2012. Retrieved from <http://www.cognizant.com/InsightsWhitepapers/Unraveling-the-Customer-Mind.pdf>.
- Dror, G., Boulle, M., Guyon, I., Lemaire, V., and Vogel, D.. *The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009) Volume 3, Challenges in Machine Learning*, Massachusetts, US. Microtome Publishing. 2009.
- Gelman, A. and Nolan, D.. *Teaching Statistics a bag of tricks*. Cambridge, MA. Oxford University Press. 2002.
- Goshtasby, A. A. *Image Registration Principles, Tools and Methods*. London, Springer. 2012.
- Greene, D. and Cunningham, P.. Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering. *Proceedings from the 23rd International Conference on Machine Learning*, Pittsburgh, PA. 2006. Retrieved from http://www.autonlab.org/icml_documents/camera-ready/048_Practical_Solutions.pdf.
- Gupta, A.. *Learning Apache Mahout Classification*, Birmingham, UK. Packt

Publishing. 2015.

- Gutierrez, N.. Demystifying Market Basket Analysis. 2006. Retrieved from <http://www.information-management.com/specialreports/20061031/1067598-1.html>.
- Hand, D., Manilla, H., and Smith, P.. *Principles of Data Mining*. USA. MIT Press. 2001. Retrieved from ftp://gamma.sbin.org/pub/doc/books/Principles_of_Data_Mining.pdf.
- Intel. What Happens in an Internet Minute?. 2013. Retrieved from <http://www.intel.co.uk/content/www/uk/en/communications/internet-minute-infographic.html>.
- Kaluža, B.. *Instant Weka How-To*. Birmingham. Packt Publishing. 2013.
- Karkera, K. R.. *Building Probabilistic Graphical Models with Python*. Birmingham, UK. Packt Publishing. 2014.
- KDD (n.d.). KDD Cup 2009: Customer relationship prediction. Retrieved from <http://www.kdd.org/kdd-cup/view/kdd-cup-2009>.
- Koller, D. and Friedman, N.. *Probabilistic Graphical Models Principles and Techniques*. Cambridge, Mass. MIT Press. 2012.
- Kurucz, M., Siklósi, D., Bíró, I., Csizsek, P., Fekete, Z., Iwatt, R., Kiss, T., and Szabó, A.. KDD Cup 2009 @ Budapest: feature partitioning and boosting 61. JMLR W&CP 7, 65–75. 2009.
- Laptev, N., Amizadeh, S., and Billawala, Y. (n.d.). A Benchmark Dataset for Time Series Anomaly Detection. Retrieved from <http://yahoolabs.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly>.
- LKurgan, L.A. and Musilek, P.. A survey of Knowledge Discovery and Data Mining process models. The Knowledge Engineering Review, 21(1), 1–24. 2006.
- Lo, H.-Y., Chang, K.-W., Chen, S.-T., Chiang, T.-H., Ferng, C.-S., Hsieh, C.-J., Ko, Y.-K., Kuo, T.-T., Lai, H.-C., Lin, K.-Y., Wang, C.-H., Yu, H.-F., Lin, C.-J., Lin, H.-T., and Lin, S.-de. An Ensemble of Three Classifiers for KDD Cup 2009: Expanded Linear Model, Heterogeneous Boosting, and Selective Naive Bayes, JMLR W&CP 7, 57–64. 2009.
- Magalhães, P. Incorrect information provided by your website. 2010. Retrevied from <http://www.best.eu.org/aboutBEST/helpdeskRequest.jsp?req=f5wpxc8&auth=Paulo>.
- Mariscal, G., Marban, O., and Fernandez, C.. A survey of data mining and knowledge discovery process models and methodologies. The Knowledge Engineering Review, 25(2), 137–166. 2010.

- Mew, K. (2015). *Android 5 Programming by Example*. Birmingham, UK. Packt Publishing.
- Miller, H., Clarke, S., Lane, S., Lonie, A., Lazaridis, D., Petrovski, S., and Jones, O.. Predicting customer behavior: The University of Melbourne's KDD Cup report, JMLR W&CP 7, 45–55. 2009.
- Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sind- hwani, V., Liu, Y., Melville, P., Wang, D., Xiao, J., Hu, J., Singh, M., Shang, W. X., and Zhu, Y.. Winning the KDD Cup Orange Challenge with Ensemble Selection. JMLR W&CP, 7, 23–34. 2009. Retrieved from <http://jmlr.org/proceedings/papers/v7/niculescu09/niculescu09.pdf>.
- Oracle (n.d.). Anomaly Detection. Retrieved from http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/anomalies.htm.
- Osugi, T., Deng, K., and Scott, S.. Balancing exploration and exploitation: a new algorithm for active machine learning. Fifth IEEE International Conference on Data Mining, Houston, Texas. 2005.
- Power, D. J. (ed.). DSS News. DSSResources.com, 3(23). 2002. Retreived from <http://www.dssresources.com/newsletters/66.php>.
- Quinlan, J. R. C4.5: *Programs for Machine Learning*. San Francisco, CA. Morgan Kaufmann Publishers. 1993.
- Rajak, A.. Association Rule Mining-Applications in Various Areas. 2008. Retrieved from https://www.researchgate.net/publication/238525379_Association_rule_mining_Applications_in_various_areas.
- Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B.. (eds.). Recommender Systems Handbook. New York, Springer. 2010.
- Rumsfeld, D. H. and Myers, G.. DoD News Briefing – Secretary Rumsfeld and Gen. Myers. 2002. Retrieved from <http://archive.defense.gov/transcripts/transcript.aspx?transcriptid=2636>.
- Stevens, S. S.. On the Theory of Scales of Measurement. Science, 103 (2684), 677–680. 1946.
- Sutton, R. S. and Barto, A. G.. *Reinforcement Learning An Introduction*. Cambridge, MA: MIT Press. 1998.
- Tiwary, C.. *Learning Apache Mahout*. Birmingham, UK. Packt Publishing. 2015.
- Tsai, J., Kaminka, G., Epstein, S., Zilka, A., Rika, I., Wang, X., Ogden, A., Brown, M., Fridman, N., Taylor, M., Bowring, E., Marsella, S., Tambe, M., and Sheel, A.. ESCAPES - Evacuation Simulation with Children, Authorities, Parents, Emotions, and Social comparison. Proceedings from 10th International

Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011) 2 (6), 457–464. 2011. Retrieved from http://www.aamas-conference.org/Proceedings/aamas2011/papers/D3_G57.pdf.

- Tsanas, A. and Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49, 560-567. 2012.
- Utts, J.. What Educated Citizens Should Know About Statistics and Probability. *The American Statistician*, 57 (2), 74-79. 2003.
- Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D.. Evaluation Methods for Topic Models. Proceedings from the 26th International conference on Machine Learning, Montreal, Canada. 2009. Retrieved from <http://mimno.infosci.cornell.edu/papers/wallach09evaluation.pdf>.
- Witten, I. H. and Frank, E.. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. USA. Morgan Kaufmann Publishers. 2000.
- Xie, J., Rojkova, V., Pal, S., and Coggeshall, S.. A Combination of Boosting and Bagging for KDD Cup 2009. *JMLR W&CP*, 7, 35–43. 2009.
- Zhang, H.. The Optimality of Naive Bayes. Proceedings from FLAIRS 2004 conference. 2004. Retrieved from <http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf>.
- Ziegler, C-N., McNee, S. M., Konstan, J. A., and Lausen, G.. Improving Recommendation Lists Through Topic Diversification. Proceedings from the 14th International World Wide Web Conference (WWW '05), Chiba, Japan. 2005. Retrieved from <http://www2.informatik.uni-freiburg.de/~cziegler/papers/WWW-05-CR.pdf>.

Part 3. Module 3

Mastering Java Machine Learning

Mastering and implementing advanced techniques in machine learning

Chapter 1. Machine Learning Review

Recent years have seen the revival of **artificial intelligence (AI)** and machine learning in particular, both in academic circles and the industry. In the last decade, AI has seen dramatic successes that eluded practitioners in the intervening years since the original promise of the field gave way to relative decline until its re-emergence in the last few years.

What made these successes possible, in large part, was the impetus provided by the need to process the prodigious amounts of ever-growing data, key algorithmic advances by dogged researchers in deep learning, and the inexorable increase in raw computational power driven by Moore's Law. Among the areas of AI leading the resurgence, machine learning has seen spectacular developments, and continues to find the widest applicability in an array of domains. The use of machine learning to help in complex decision making at the highest levels of business and, at the same time, its enormous success in improving the accuracy of what are now everyday applications, such as searches, speech recognition, and personal assistants on mobile phones, have made its effects commonplace in the family room and the board room alike. Articles breathlessly extolling the power of deep learning can be found today not only in the popular science and technology press but also in mainstream outlets such as *The New York Times* and *The Huffington Post*. Machine learning has indeed become ubiquitous in a relatively short time.

An *ordinary* user encounters machine learning in many ways in their day-to-day activities. Most e-mail providers, including Yahoo and Gmail, give the user automated sorting and categorization of e-mails into headings such as Spam, Junk, Promotions, and so on, which is made possible using text mining, a branch of machine learning. When shopping online for products on e-commerce websites, such as <https://www.amazon.com/>, or watching movies from content providers, such as Netflix, one is offered recommendations for other products and content by so-called recommender systems, another branch of machine learning, as an effective way to retain customers.

Forecasting the weather, estimating real estate prices, predicting voter turnout, and even election results—all use some form of machine learning to see into the future, as it were.

The ever-growing availability of data and the promise of systems that can enrich our

lives by learning from that data place a growing demand on the skills of the limited workforce of professionals in the field of data science. This demand is particularly acute for well-trained experts who know their way around the landscape of machine learning techniques in the more popular languages, such as Java, Python, R, and increasingly, Scala. Fortunately, thanks to the thousands of contributors in the open source community, each of these languages has a rich and rapidly growing set of libraries, frameworks, and tutorials that make state-of-the-art techniques accessible to anyone with an internet connection and a computer, for the most part. Java is an important vehicle for this spread of tools and technology, especially in large-scale machine learning projects, owing to its maturity and stability in enterprise-level deployments and the portable JVM platform, not to mention the legions of professional programmers who have adopted it over the years. Consequently, mastery of the skills so lacking in the workforce today will put any aspiring professional with a desire to enter the field at a distinct advantage in the marketplace.

Perhaps you already apply machine learning techniques in your professional work, or maybe you simply have a hobbyist's interest in the subject. If you're reading this, it's likely you can already bend Java to your will, no problem, but now you feel you're ready to dig deeper and learn how to use the best of breed open source ML Java frameworks in your next data science project. If that is indeed you, how fortuitous is it that the chapters in this book are designed to do all that and more!

Mastery of a subject, especially one that has such obvious applicability as machine learning, requires more than an understanding of its core concepts and familiarity with its mathematical underpinnings. Unlike an introductory treatment of the subject, a book that purports to help you master the subject must be heavily focused on practical aspects in addition to introducing more advanced topics that would have stretched the scope of the introductory material. To warm up before we embark on sharpening our skills, we will devote this chapter to a quick review of what we already know. For the ambitious novice with little or no prior exposure to the subject (who is nevertheless determined to get the fullest benefit from this book), here's our advice: make sure you do not skip the rest of this chapter; instead, use it as a springboard to explore unfamiliar concepts in more depth. Seek out external resources as necessary. Wikipedia them. Then jump right back in.

For the rest of this chapter, we will review the following:

- History and definitions
- What is not machine learning?

- Concepts and terminology
- Important branches of machine learning
- Different data types in machine learning
- Applications of machine learning
- Issues faced in machine learning
- The meta-process used in most machine learning projects
- Information on some well-known tools, APIs, and resources that we will employ in this book

Machine learning – history and definition

It is difficult to give an exact history, but the definition of machine learning we use today finds its usage as early as the 1860s. In Rene Descartes' *Discourse on the Method*, he refers to *Automata* and says:

For we can easily understand a machine's being constituted so that it can utter words, and even emit some responses to action on it of a corporeal kind, which brings about a change in its organs; for instance, if touched in a particular part it may ask what we wish to say to it; if in another part it may exclaim that it is being hurt, and so on.

Note

<http://www.earlymoderntexts.com/assets/pdfs/descartes1637.pdf>

<https://www.marxists.org/reference/archive/descartes/1635/discourse-method.htm>

Alan Turing, in his famous publication *Computing Machinery and Intelligence* gives basic insights into the goals of machine learning by asking the question "Can machines think?".

Note

<http://csmt.uchicago.edu/annotations/turing.htm>

<http://www.csee.umbc.edu/courses/471/papers/turing.pdf>

Arthur Samuel in 1959 wrote, "*Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.*".

Tom Mitchell in recent times gave a more exact definition of machine learning: "*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*"

Machine learning has a relationship with several areas:

- **Statistics:** It uses the elements of data sampling, estimation, hypothesis testing, learning theory, and statistical-based modeling, to name a few
- **Algorithms and computation:** It uses the basic concepts of search, traversal, parallelization, distributed computing, and so on from basic computer science
- **Database and knowledge discovery:** For its ability to store, retrieve, and access information in various formats
- **Pattern recognition:** For its ability to find interesting patterns from the data to explore, visualize, and predict
- **Artificial intelligence:** Though it is considered a branch of artificial intelligence, it also has relationships with other branches, such as heuristics, optimization, evolutionary computing, and so on

What is not machine learning?

It is important to recognize areas that share a connection with machine learning but cannot themselves be considered part of machine learning. Some disciplines may overlap to a smaller or larger extent, yet the principles underlying machine learning are quite distinct:

- **Business intelligence (BI) and reporting:** Reporting key performance indicators (KPI's), querying OLAP for slicing, dicing, and drilling into the data, dashboards, and so on that form the central components of BI are not machine learning.
- **Storage and ETL:** Data storage and ETL are key elements in any machine learning process, but, by themselves, they don't qualify as machine learning.
- **Information retrieval, search, and queries:** The ability to retrieve data or documents based on search criteria or indexes, which form the basis of information retrieval, are not really machine learning. Many forms of machine learning, such as semi-supervised learning, can rely on the searching of similar data for modeling, but that doesn't qualify searching as machine learning.
- **Knowledge representation and reasoning:** Representing knowledge for performing complex tasks, such as ontology, expert systems, and semantic webs, does not qualify as machine learning.

Machine learning – concepts and terminology

In this section, we will describe the different concepts and terms normally used in machine learning:

- **Data or dataset:** The basics of machine learning rely on understanding the data. The data or dataset normally refers to content available in structured or unstructured format for use in machine learning. Structured datasets have specific formats, and an unstructured dataset is normally in the form of some free-flowing text. Data can be available in various storage types or formats. In structured data, every element known as an instance or an example or row follows a predefined structure. Data can also be categorized by size: small or medium data have a few hundreds to thousands of instances, whereas *big* data refers to a large volume, mostly in millions or billions, that cannot be stored or accessed using common devices or fit in the memory of such devices.
- **Features, attributes, variables, or dimensions:** In structured datasets, as mentioned before, there are predefined elements with their own semantics and data type, which are known variously as features, attributes, metrics, indicators, variables, or dimensions.
- **Data types:** The features defined earlier need some form of typing in many machine learning algorithms or techniques. The most commonly used data types are as follows:
 - **Categorical or nominal:** This indicates well-defined categories or values present in the dataset. For example, eye color—black, blue, brown, green, grey; document content type—text, image, video.
 - **Continuous or numeric:** This indicates a numeric nature of the data field. For example, a person's weight measured by a bathroom scale, the temperature reading from a sensor, or the monthly balance in dollars on a credit card account.
 - **Ordinal:** This denotes data that can be ordered in some way. For example, garment size—small, medium, large; boxing weight classes: heavyweight, light heavyweight, middleweight, lightweight, and bantamweight.
- **Target or label:** A feature or set of features in the dataset, which is used for learning from training data and predicting in an unseen dataset, is known as a target or a label. The term "ground truth" is also used in some domains. A label can have any form as specified before, that is, categorical, continuous, or

ordinal.

- **Machine learning model:** Each machine learning algorithm, based on what it learned from the dataset, maintains the state of its learning for predicting or giving insights into future or unseen data. This is referred to as the machine learning model.
- **Sampling:** Data sampling is an essential step in machine learning. Sampling means choosing a subset of examples from a population with the intent of treating the behavior seen in the (smaller) sample as being representative of the behavior of the (larger) population. In order for the sample to be representative of the population, care must be taken in the way the sample is chosen.
Generally, a population consists of every object sharing the properties of interest in the problem domain, for example, all people eligible to vote in the general election, or all potential automobile owners in the next four years. Since it is usually prohibitive (or impossible) to collect data for all the objects in a population, a well-chosen subset is selected for the purpose of analysis. A crucial consideration in the sampling process is that the sample is unbiased with respect to the population. The following are types of probability-based sampling:
 - **Uniform random sampling:** This refers to sampling that is done over a uniformly distributed population, that is, each object has an equal probability of being chosen.
 - **Stratified random sampling:** This refers to the sampling method used when the data can be categorized into multiple classes. In such cases, in order to ensure all categories are represented in the sample, the population is divided into distinct strata based on these classifications, and each stratum is sampled in proportion to the fraction of its class in the overall population. Stratified sampling is common when the population density varies across categories, and it is important to compare these categories with the same statistical power. Political polling often involves stratified sampling when it is known that different demographic groups vote in significantly different ways. Disproportional representation of each group in a random sample can lead to large errors in the outcomes of the polls. When we control for demographics, we can avoid oversampling the majority over the other groups.
 - **Cluster sampling:** Sometimes there are natural groups among the population being studied, and each group is representative of the whole population. An example is data that spans many geographical regions. In cluster sampling, you take a random subset of the groups followed by a

random sample from within each of those groups to construct the full data sample. This kind of sampling can reduce the cost of data collection without compromising the fidelity of distribution in the population.

- **Systematic sampling:** Systematic or interval sampling is used when there is a certain ordering present in the sampling frame (a finite set of objects treated as the population and taken to be the source of data for sampling, for example, the corpus of Wikipedia articles, arranged lexicographically by title). If the sample is then selected by starting at a random object and skipping a constant k number of objects before selecting the next one, that is called systematic sampling. The value of k is calculated as the ratio of the population to the sample size.
- **Model evaluation metrics:** Evaluating models for performance is generally based on different evaluation metrics for different types of learning. In classification, it is generally based on accuracy, **receiver operating characteristics (ROC)** curves, training speed, memory requirements, false positive ratio, and so on, to name a few (see [Chapter 2, Practical Approach to Real-World Supervised Learning](#)). In clustering, the number of clusters found, cohesion, separation, and so on form the general metrics (see [Chapter 3, Unsupervised Machine Learning Techniques](#)). In stream-based learning, apart from the standard metrics mentioned earlier, adaptability, speed of learning, and robustness to sudden changes are some of the conventional metrics for evaluating the performance of the learner (see [Chapter 5, Real-Time Stream Machine Learning](#)).

To illustrate these concepts, a concrete example in the form of a commonly used sample weather dataset is given. The data gives a set of weather conditions and a label that indicates whether the subject decided to play a game of tennis on the day or not:

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,TRUE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
```

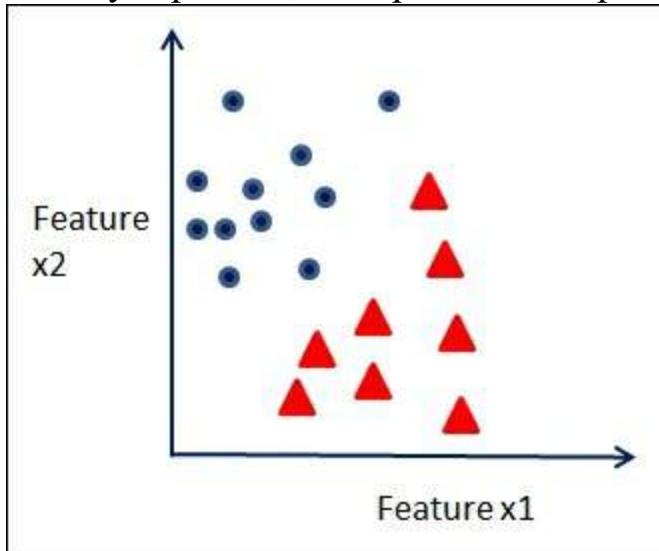
```
rainy,70,96,TRUE,no
rainy,68,80,TRUE,no
rainy,65,70,TRUE,no
overcast,64,65,TRUE,no
sunny,72,95,TRUE,no
sunny,69,70,TRUE,no
rainy,75,80,TRUE,no
sunny,75,70,TRUE,no
overcast,72,90,TRUE,no
overcast,81,75,TRUE,no
rainy,71,91,TRUE,no
```

The dataset is in the format of an **ARFF (attribute-relation file format)** file. It consists of a header giving the information about features or attributes with their data types and actual comma-separated data following the data tag. The dataset has five features, namely `outlook`, `temperature`, `humidity`, `windy`, and `play`. The features `outlook` and `windy` are categorical features, while `humidity` and `temperature` are continuous. The feature `play` is the target and is categorical.

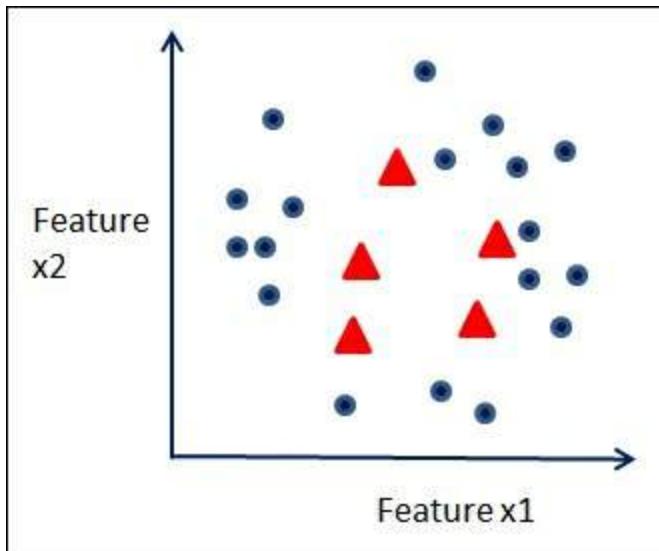
Machine learning – types and subtypes

We will now explore different subtypes or branches of machine learning. Though the following list is not comprehensive, it covers the most well-known types:

- **Supervised learning:** This is the most popular branch of machine learning, which is about learning from labeled data. If the data type of the label is categorical, it becomes a classification problem, and if numeric, it is known as a regression problem. For example, if the goal of using of the dataset is the detection of fraud, which has categorical values of either true or false, we are dealing with a classification problem. If, on the other hand, the target is to predict the best price to list the sale of a home, which is a numeric dollar value, the problem is one of regression. The following figure illustrates labeled data that warrants the use of classification techniques, such as logistic regression that is suitable for linearly separable data, that is, when there exists a line that can cleanly separate the two classes. For higher dimensional data that may be linearly separable, one speaks of a separating hyperplane:



Linearly separable data

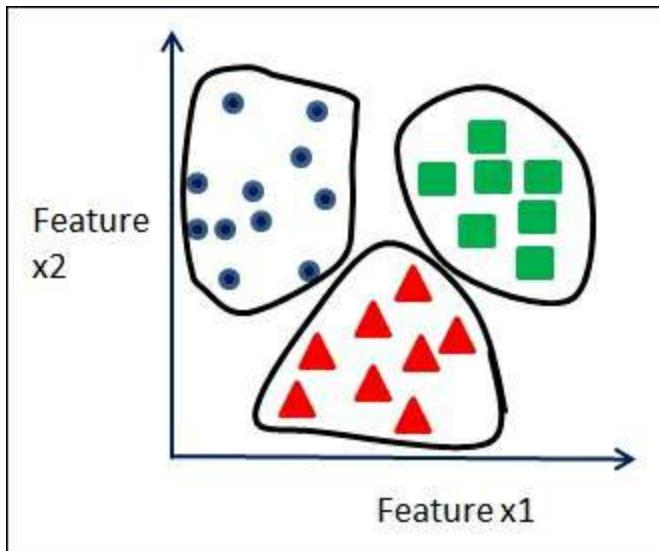


An example of a dataset that is not linearly separable.

This type of problem calls for classification techniques, such as support vector machines.

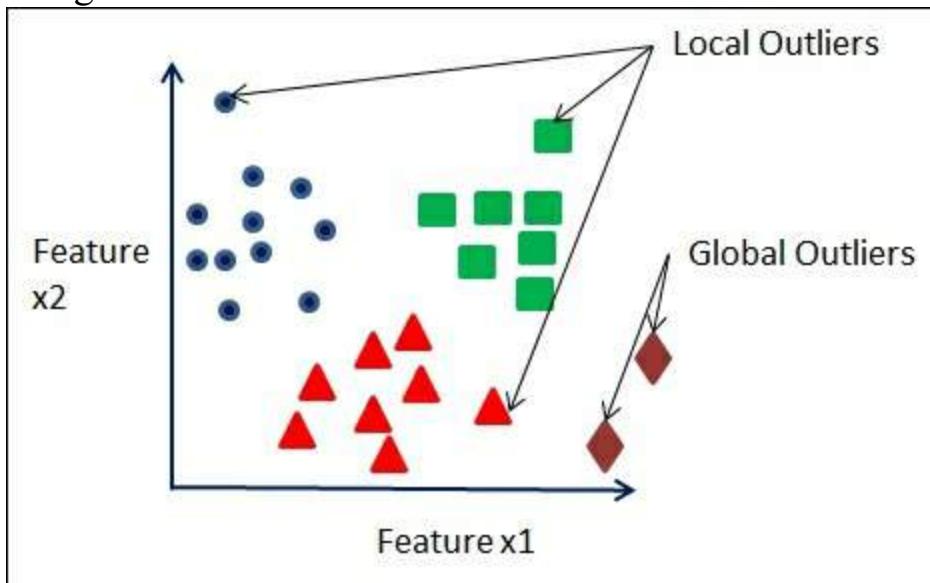
- **Unsupervised learning:** Understanding the data and exploring it for building machine learning models when the labels are not given is called unsupervised learning. Clustering, manifold learning, and outlier detection are techniques that are covered under this topic, which are dealt with in detail in [Chapter 3, Unsupervised Machine Learning Techniques](#). Examples of problems that require unsupervised learning are many. Grouping customers according to their purchasing behavior is one example. In the case of biological data, tissues samples can be clustered based on similar gene expression values using unsupervised learning techniques.

The following figure represents data with inherent structure that can be revealed as distinct clusters using an unsupervised learning technique, such as k-means:



Clusters in data

Different techniques are used to detect global outliers—examples that are anomalous with respect to the entire dataset, and local outliers—examples that are misfits in their neighborhood. In the following figure, the notion of local and global outliers is illustrated for a two-feature dataset:



Local and global outliers

- **Semi-supervised learning:** When the dataset has only some labeled data and a

large amount of data that is not labeled, learning from such a dataset is called **semi-supervised learning**. When dealing with financial data with the goal of detecting fraud, for example, there may be a large amount of unlabeled data and only a small number of known fraud and non-fraud transactions. In such cases, semi-supervised learning may be applied.

- **Graph mining:** Mining data represented as graph structures is known as **graph mining**. It is the basis of social network analysis and structure analysis in different bioinformatics, web mining, and community mining applications.
- **Probabilistic graph modeling and inferencing:** Learning and exploiting conditional dependence structures present between features expressed as a graph-based model comes under the branch of **probabilistic graph modeling**. Bayesian networks and Markov random fields are two classes of such models.
- **Time-series forecasting:** This refers to a form of learning where data has distinct temporal behavior and the relationship with time is modeled. A common example is in financial forecasting, where the performance of stocks in a certain sector may be the target of the predictive model.
- **Association analysis:** This is a form of learning where data is in the form of an item set or market basket, and association rules are modeled to explore and predict the relationships between the items. A common example in association analysis is to learn the relationships between the most common items bought by customers when they visit the grocery store.
- **Reinforcement learning:** This is a form of learning where machines learn to maximize performance based on feedback in the form of rewards or penalties received from the environment. A recent example that famously used reinforcement learning, among other techniques, was AlphaGo, the machine developed by Google that decisively beat the World Go Champion Lee Sedol in March 2016. Using a reward and penalty scheme, the model first trained on millions of board positions in the supervised learning stage, then played itself in the reinforcement learning stage to ultimately become good enough to triumph over the best human player.

Note

<http://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/>

<https://gogameguru.com/i/2016/03/deepmind-mastering-go.pdf>

- **Stream learning or incremental learning:** Learning in a supervised,

unsupervised, or semi-supervised manner from stream data in real time or pseudo-real time is called stream or incremental learning. Learning the behaviors of sensors from different types of industrial systems for categorizing into normal and abnormal is an application that needs real-time feeds and real-time detection.

Datasets used in machine learning

To learn from data, we must be able to understand and manage data in all forms. Data originates from many different sources, and consequently, datasets may differ widely in structure or have little or no structure at all. In this section, we present a high-level classification of datasets with commonly occurring examples.

Based on their structure, or the lack thereof, datasets may be classified as containing the following:

- **Structured data:** Datasets with structured data are more amenable to being used as input to most machine learning algorithms. The data is in the form of records or rows following a well-known format with features that are either columns in a table or fields delimited by separators or tokens. There is no explicit relationship between the records or instances. The dataset is available chiefly in flat files or relational databases. The records of financial transactions at a bank shown in the following figure are an example of structured data:

Account No	Card	Amount	Fraud	Mode	Zipcode	Time
A*122	xxxx2 345	50.23	N	Device	20147	06/22/2003:10:46 AM
A*121	xxxx1 245	12.43	F	Online	20123	06/22/2003:10:47 AM
A*122	xxxx2 345	1000.00	F	ATM	20901	06/22/2003:10:47 AM

Financial card transactional data with labels of fraud

- **Transaction or market data:** This is a special form of structured data where each entry corresponds to a collection of items. Examples of market datasets are the lists of grocery items purchased by different customers or movies viewed by customers, as shown in the following table:

CustomerId	Items
1121	Bread, Milk, Soda, Juice
1127	Diaper, Beer, Milk
1189	Cookies, Beer, Soda

Market dataset for items bought from grocery store

- **Unstructured data:** Unstructured data is normally not available in well-known formats, unlike structured data. Text data, image, and video data are different formats of unstructured data. Usually, a transformation of some form is needed to extract features from these forms of data into a structured dataset so that traditional machine learning algorithms can be applied.

'Go until jurong point, crazy..... Cine there got amore wat...',**ham**

'Ok lar... Joking wif u oni...',**ham**

'Free entry in 2 a wkly comp to win FA Cup final tkts 21st, CONTACT 8812283 over18s',**spam**

'U dun say so early hor... U c already then say...',**ham**'Nah f, he lives around here though',**ham**

Sample text data, with no discernible structure, hence unstructured. Separating spam from normal messages (ham) is a binary classification problem. Here true positives (spam) and true negatives (ham) are distinguished by their labels, the second token in each instance of data. SMS Spam Collection Dataset (UCI Machine Learning Repository), source: Tiago A. Almeida from the Federal University of Sao Carlos.

- **Sequential data:** Sequential data have an explicit notion of "order" to them. The order can be some relationship between features and a time variable in time series data, or it can be symbols repeating in some form in genomic datasets. Two examples of sequential data are weather data and genomic sequence data. The following figure shows the relationship between time and the sensor level for weather:

'Go until jurong point, crazy..... Cine there got amore wat...', **ham**
'Ok lar... Joking wif u oni...', **ham**
'Free entry in 2 a wkly comp to win FA Cup final tkts 21st, CONTACT 88122823 over 18s', **spam**
'U dun say so early hor... U c already then say...', **ham**
'Nah f, he lives around here though', **ham**

Time series from sensor data

Three genomic sequences are taken into consideration to show the repetition of the sequences CGGGT and TTGAAAGTGGTG in all three genomic sequences:

Gene 1>

CGGGTGCGGATTGTTGGAGGGGTTGAAAGTGGTGCCG

Gene 2>

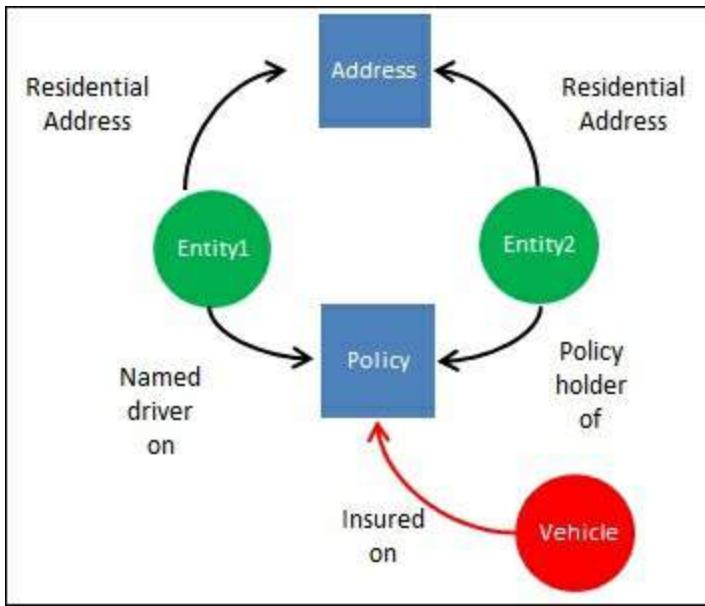
CGGGTTGGAGGAGGTTGGAGGGGTTGAAAGTGGTGCG

Gene 3>

TTGAAAGTGGTGCCGGGTTGGAGGAGGTTGAGGGGCG

Genomic sequences of DNA as a sequence of symbols.

- **Graph data:** Graph data is characterized by the presence of relationships between entities in the data to form a graph structure. Graph datasets may be in a structured record format or an unstructured format. Typically, the graph relationship has to be mined from the dataset. Claims in the insurance domain can be considered structured records containing relevant claim details with claimants related through addresses, phone numbers, and so on. This can be viewed in a graph structure. Using the World Wide Web as an example, we have web pages available as unstructured data containing links, and graphs of relationships between web pages that can be built using web links, producing some of the most extensively mined graph datasets today:



Insurance claim data, converted into a graph structure showing the relationship between vehicles, drivers, policies, and addresses

Machine learning applications

Given the rapidly growing use of machine learning in diverse areas of human endeavor, any attempt to list typical applications in the different industries where some form of machine learning is in use must necessarily be incomplete.

Nevertheless, in this section, we list a broad set of machine learning applications by domain and the type of learning employed:

Domain/Industry	Applications	Machine Learning Type
Financial	Credit risk scoring, fraud detection, and anti-money laundering	Supervised, unsupervised, graph models, time series, and stream learning
Web	Online campaigns, health monitoring, and ad targeting	Supervised, unsupervised, semi-supervised
Healthcare	Evidence-based medicine, epidemiological surveillance, drug events prediction, and claim fraud detection	Supervised, unsupervised, graph models, time series, and stream learning
Internet of things (IoT)	Cyber security, smart roads, and sensor health monitoring	Supervised, unsupervised, semi-supervised, and stream learning
Environment	Weather forecasting, pollution modeling, and water quality measurement	Time series, supervised, unsupervised, semi-supervised, and stream learning
Retail	Inventory, customer management and recommendations, layout, and forecasting	Time series, supervised, unsupervised, semi-supervised, and stream learning

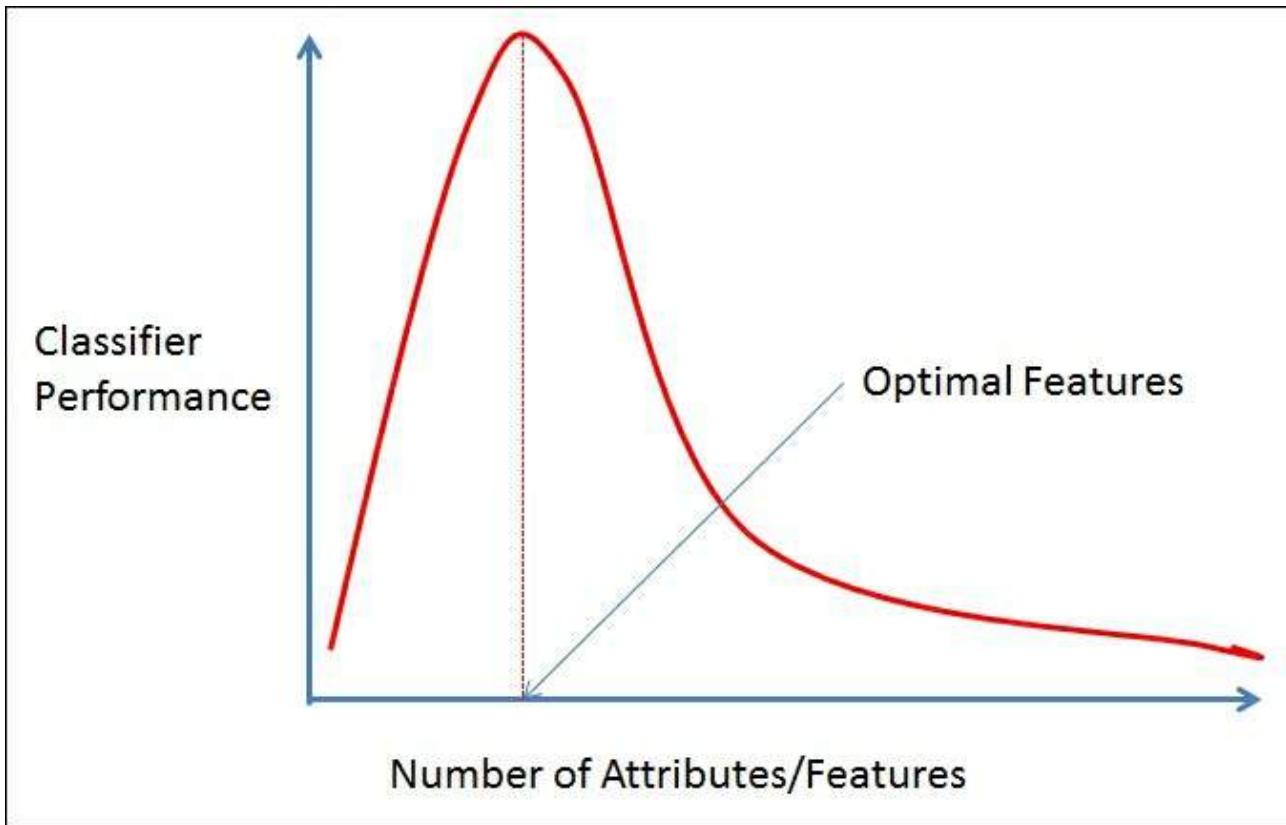
Applications of machine learning

Practical issues in machine learning

It is necessary to appreciate the nature of the constraints and potentially sub-optimal conditions one may face when dealing with problems requiring machine learning. An understanding of the nature of these issues, the impact of their presence, and the methods to deal with them will be addressed throughout the discussions in the coming chapters. Here, we present a brief introduction to the practical issues that confront us:

- **Data quality and noise:** Missing values, duplicate values, incorrect values due to human or instrument recording error, and incorrect formatting are some of the important issues to be considered while building machine learning models. Not addressing data quality can result in incorrect or incomplete models. In the next chapter, we will highlight some of these issues and some strategies to overcome them through data cleansing.
- **Imbalanced datasets:** In many real-world datasets, there is an imbalance among labels in the training data. This imbalance in a dataset affects the choice of learning, the process of selecting algorithms, model evaluation and verification. If the right techniques are not employed, the models can suffer large biases, and the learning is not effective. Detailed in the next few chapters are various techniques that use meta-learning processes, such as cost-sensitive learning, ensemble learning, outlier detection, and so on, which can be employed in these situations.
- **Data volume, velocity, and scalability:** Often, a large volume of data exists in raw form or as real-time streaming data at high speed. Learning from the entire data becomes infeasible either due to constraints inherent to the algorithms or hardware limitations, or combinations thereof. In order to reduce the size of the dataset to fit the resources available, data sampling must be done. Sampling can be done in many ways, and each form of sampling introduces a bias. Validating the models against sample bias must be performed by employing various techniques, such as stratified sampling, varying sample sizes, and increasing the size of experiments on different sets. Using big data machine learning can also overcome the volume and sampling biases.
- **Overfitting:** One of the core problems in predictive models is that the model is not generalized enough and is made to fit the given training data *too well*. This results in poor performance of the model when applied to unseen data. There are various techniques described in later chapters to overcome these issues.
- **Curse of dimensionality:** When dealing with high-dimensional data, that is,

datasets with a large number of features, scalability of machine learning algorithms becomes a serious concern. One of the issues with adding more features to the data is that it introduces sparsity, that is, there are now fewer data points on average per unit volume of feature space unless an increase in the number of features is accompanied by an exponential increase in the number of training examples. This can hamper performance in many methods, such as distance-based algorithms. Adding more features can also deteriorate the predictive power of learners, as illustrated in the following figure. In such cases, a more suitable algorithm is needed, or the dimensionality of the data must be reduced.



Curse of dimensionality illustrated in classification learning, where adding more features deteriorates classifier performance.

Machine learning – roles and process

Any effort to apply machine learning to a large-sized problem requires the collaborative effort of a number of roles, each abiding by a set of systematic processes designed for rigor, efficiency, and robustness. The following roles and processes ensure that the goals of the endeavor are clearly defined at the outset and the correct methodologies are employed in data analysis, data sampling, model selection, deployment, and performance evaluation—all as part of a comprehensive framework for conducting analytics consistently and with repeatability.

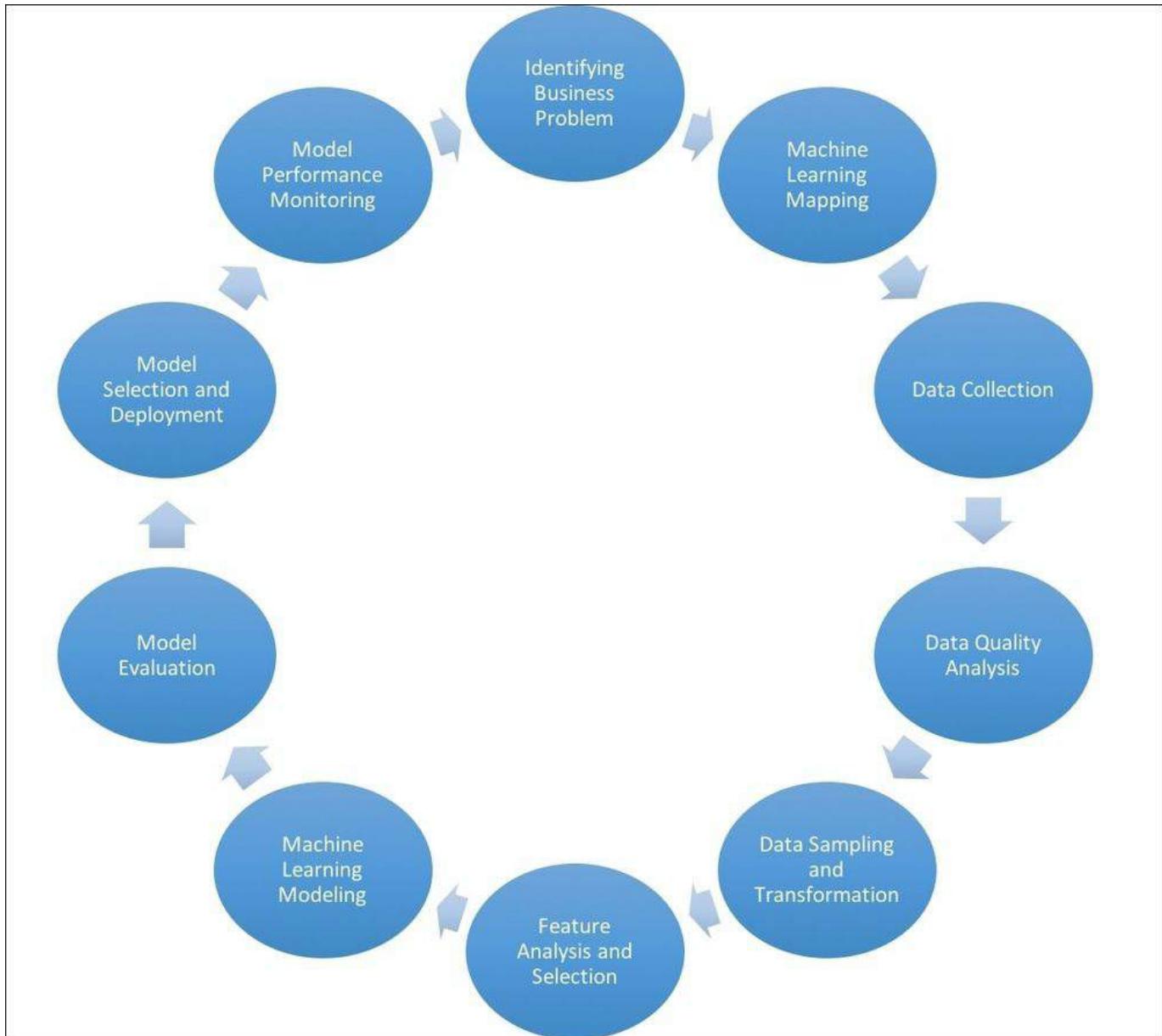
Roles

Participants play specific parts in each step. These responsibilities are captured in the following four roles:

- **Business domain expert:** A subject matter expert with knowledge of the problem domain
- **Data engineer:** Involved in the collecting, transformation, and cleaning of the data
- **Project manager:** Overseer of the smooth running of the process
- **Data scientist or machine learning expert:** Responsible for applying descriptive or predictive analytic techniques

Process

CRISP (Cross Industry Standard Process) is a well-known high-level process model for data mining that defines the analytics process. In this section, we have added some of our own extensions to the CRISP process that make it more comprehensive and better suited for analytics using machine learning. The entire iterative process is demonstrated in the following schematic figure. We will discuss each step of the process in detail in this section.



- **Identifying the business problem:** Understanding the objectives and the end

goals of the project or process is the first step. This is normally carried out by a business domain expert in conjunction with the project manager and machine learning expert. What are the end goals in terms of data availability, formats, specification, collection, ROI, business value, deliverables? All these questions are discussed in this phase of the process. Identifying the goals clearly, and *in quantifiable terms* where possible, such as dollar amount saved, finding a pre-defined number of anomalies or clusters, or predicting no more than a certain number of false positives, and so on, is an important objective of this phase.

- **Machine learning mapping:** The next step is mapping the business problem to one or more machine learning types discussed in the preceding section. This step is generally carried out by the machine learning expert. In it, we determine whether we should use just one form of learning (for example, supervised, unsupervised, semi-supervised) or if a hybrid of forms is more suitable for the project.
- **Data collection:** Obtaining the raw data in the agreed format and specification for processing follows next. This step is normally carried out by data engineers and may require handling some basic ETL steps.
- **Data quality analysis:** In this step, we perform analysis on the data for missing values, duplicates, and so on, conduct basic statistical analysis on the categorical and continuous types, and similar tasks to evaluate the quality of data. Data engineers and data scientists can perform the tasks together.
- **Data sampling and transformation:** Determining whether data needs to be divided into samples and performing data sampling of various sizes for training, validation, or testing—these are the tasks performed in this step. It consists of employing different sampling techniques, such as oversampling and random sampling of the training datasets for effective learning by the algorithms, especially when the data is highly imbalanced in the labels. The data scientist is involved in this task.
- **Feature analysis and selection:** This is an iterative process combined with modeling in many tasks to make sure the features are analyzed for either their discriminating values or their effectiveness. It can involve finding new features, transforming existing features, handling the data quality issues mentioned earlier, selecting a subset of features, and so on ahead of the modeling process. The data scientist is normally assigned this task.
- **Machine learning modeling:** This is an iterative process working on different algorithms based on data characteristics and learning types. It involves different steps, such as generating hypotheses, selecting algorithms, tuning parameters, and getting results from evaluation to find models that meet the criteria. The

data scientist carries out this task.

- **Model evaluation:** While this step is related to all the preceding steps to some degree, it is more closely linked to the business understanding phase and machine learning mapping phase. The evaluation criteria must map in some way to the business problem or the goal. Each problem/project has its own goal, whether that be improving true positives, reducing false positives, finding anomalous clusters or behaviors, or analyzing data for different clusters. Different techniques that implicitly or explicitly measure these targets are used based on learning techniques. Data scientists and business domain experts normally take part in this step.
- **Model selection and deployment:** Based on the evaluation criteria, one or more models—Independent or as an ensemble—are selected. The deployment of models normally needs to address several issues: runtime scalability measures, execution specifications of the environment, and audit information, to name a few. Audit information that captures the key parameters based on learning is an essential part of the process. It ensures that model performance can be tracked and compared to check for the deterioration and aging of the models. Saving key information, such as training data volumes, dates, data quality analysis, and so on, is independent of learning types. Supervised learning might involve saving the confusion matrix, true positive ratios, false positive ratios, area under the ROC curve, precision, recall, error rates, and so on. Unsupervised learning might involve clustering or outlier evaluation results, cluster statistics, and so on. This is the domain of the data scientist, as well as the project manager.
- **Model performance monitoring:** This task involves periodically tracking the model performance in terms of the criteria it was evaluated against, such as the true positive rate, false positive rate, performance speed, memory allocation, and so on. It is imperative to measure the deviations in these metrics with respect to the metrics between successive evaluations of the trained model's performance. The deviations and tolerance in the deviation will give insights into repeating the process or retuning the models as time progresses. The data scientist is responsible for this stage.

As may be observed from the preceding diagram, the entire process is an iterative one. After a model or set of models has been deployed, business and environmental factors may change in ways that affect the performance of the solution, requiring a re-evaluation of business goals and success criteria. This takes us back through the cycle again.

Machine learning – tools and datasets

A sure way to master the techniques necessary to successfully complete a project of any size or complexity in machine learning is to familiarize yourself with the available tools and frameworks by performing experiments with widely-used datasets, as demonstrated in the chapters to follow. A short survey of the most popular Java frameworks is presented in the following list. Later chapters will include experiments that you will do using the following tools:

- **RapidMiner:** A leading analytics platform, RapidMiner has multiple offerings, including Studio, a visual design framework for processes, Server, a product to facilitate a collaborative environment by enabling sharing of data sources, processes, and practices, and Radoop, a system with translations to enable deployment and execution on the Hadoop ecosystem. RapidMiner Cloud provides a cloud-based repository and on-demand computing power.
 - **License:** GPL (Community Edition) and Commercial (Enterprise Edition)
 - **Website:** <https://rapidminer.com/>
- **Weka:** This is a comprehensive open source Java toolset for data mining and building machine learning applications with its own collection of publicly available datasets.
 - **License:** GPL
 - **Website:** <http://www.cs.waikato.ac.nz/ml/weka/>
- **Knime:** KNIME (we are urged to pronounce it with a silent k, as "naime") Analytics Platform is written in Java and offers an integrated toolset, a rich set of algorithms, and a visual workflow to do analytics without the need for standard programming languages, such as Java, Python, and R. However, one can write scripts in Java and other languages to implement functionality not available natively in KNIME.
 - **License:** GNU GPL v3
 - **Website:** <https://www.knime.org/>
- **Mallet:** This is a Java library for NLP. It offers document classification, sequence tagging, topic modeling, and other text-based applications of machine learning, as well as an API for task pipelines.
 - **License:** Common Public License version 1.0 (CPL-1)
 - **Website:** <http://mallet.cs.umass.edu/>
- **Elki:** This is a research-oriented Java software primarily focused on data mining with unsupervised algorithms. It achieves high performance and scalability using data index structures that improve access performance of

- multi-dimensional data.
- **License:** AGPLv3
 - **Website:** <http://elki.dbs.ifi.lmu.de/>
- **JCLAL:** This is a Java Class Library for Active Learning, and is an open source framework for developing Active Learning methods, one of the areas that deal with learning predictive models from a mix of labeled and unlabeled data (semi-supervised learning is another).
 - **License:** GNU General Public License version 3.0 (GPLv3)
 - **Website:** <https://sourceforge.net/projects/jclal/>
 - **KEEL:** This is an open source software written in Java for designing experiments primarily suited to the implementation of evolutionary learning and soft computing based techniques for data mining problems.
 - **License:** GPLv3
 - **Website:** <http://www.keel.es/>
 - **DeepLearning4J:** This is a distributed deep learning library for Java and Scala. DeepLearning4J is integrated with Spark and Hadoop. Anomaly detection and recommender systems are use cases that lend themselves well to the models generated via deep learning techniques.
 - **License:** Apache License 2.0
 - **Website:** <http://deeplearning4j.org/>
 - **Spark-MLLib:** (Included in Apache Spark distribution) MLLib is the machine learning library included in Spark mainly written in Scala and Java. Since the introduction of Data Frames in Spark, the `spark.ml` package, which is written on top of Data Frames, is recommended over the original `spark.mllib` package. MLLib includes support for all stages of the analytics process, including statistical methods, classification and regression algorithms, clustering, dimensionality reduction, feature extraction, model evaluation, and PMML support, among others. Another aspect of MLLib is the support for the use of pipelines or workflows. MLLib is accessible from R, Scala, and Python, in addition to Java.
 - **License:** Apache License v2.0
 - **Website:** <http://spark.apache.org/mllib/>
 - **H2O:** H2O is a Java-based library with API support in R and Python, in addition to Java. H2O can also run on Spark as its own application called Sparkling Water. H2O Flow is a web-based interactive environment with executable cells and rich media in a single notebook-like document.
 - **License:** Apache License v2.0
 - **Website:** <http://www.h2o.ai/>

- **MOA/SAMOA**: Aimed at machine learning from data streams with a pluggable interface for stream processing platforms, SAMOA, at the time of writing, is an Apache Incubator project.
 - **License**: Apache License v2.0
 - **Website**: <https://samoa.incubator.apache.org/>
- **Neo4j**: Neo4j is an open source NoSQL graphical database implemented in Java and Scala. As we will see in later chapters, graph analytics has a variety of use cases, including matchmaking, routing, social networks, network management, and so on. Neo4j supports fully ACID transactions.
 - **License**: Community Edition—GPLv3 and Enterprise Edition—multiple options, including Commercial and Educational (<https://neo4j.com/licensing/>)
 - **Website**: <https://neo4j.com/>
- **GraphX**: This is included in the Apache Spark distribution. GraphX is the graph library accompanying Spark. The API has extensive support for viewing and manipulating graph structures, as well as some graph algorithms, such as PageRank, Connected Components, and Triangle Counting.
 - **License**: Apache License v2.0
 - **Website**: <http://spark.apache.org/graphx/>
- **OpenMarkov**: OpenMarkov is a tool for editing and evaluating **probabilistic graphical models (PGM)**. It includes a GUI for interactive learning.
 - **License**: EUPLv1.1 (https://joinup.ec.europa.eu/community/eupl/og_page/eupl)
 - **Website**: <http://www.openmarkov.org/>
- **Smile**: Smile is a machine learning platform for the JVM with an extensive library of algorithms. Its capabilities include NLP, manifold learning, association rules, genetic algorithms, and a versatile set of tools for visualization.
 - **License**: Apache License 2.0
 - **Website**: <http://haifengl.github.io/smile/>

Datasets

A number of publicly available datasets have aided research and learning in data science immensely. Several of those listed in the following section are well known and have been used by scores of researchers to benchmark their methods over the years. New datasets are constantly being made available to serve different communities of modelers and users. The majority are real-world datasets from different domains. The exercises in this volume will use several datasets from this list.

- **UC Irvine (UCI) database:** Maintained by the Center for Machine Learning and Intelligent Systems at UC Irvine, the UCI database is a catalog of some 350 datasets of varying sizes, from a dozen to more than forty million records and up to three million attributes, with a mix of multivariate text, time-series, and other data types. (<https://archive.ics.uci.edu/ml/index.html>)
- **Tunedit:** (<http://tunedit.org/>) This offers Tunedit Challenges and tools to conduct repeatable data mining experiments. It also offers a platform for hosting data competitions.
- **Mldata.org:** (<http://mldata.org/>) Supported by the PASCAL 2 organization that brings together researchers and students across Europe and the world, mldata.org is primarily a repository of user-contributed datasets that encourages data and solution sharing amongst groups of researchers to help with the goal of creating reproducible solutions.
- **KDD Challenge Datasets:** (<http://www.kdnuggets.com/datasets/index.html>) KDNuggets aggregates multiple dataset repositories across a wide variety of domains.
- **Kaggle:** Billed as the *Home of Data Science*, Kaggle is a leading platform for data science competitions and also a repository of datasets from past competitions and user-submitted datasets.

Summary

Machine learning has already demonstrated impressive successes despite being a relatively young field. With the ubiquity of Java resources, Java's platform independence, and the selection of ML frameworks in Java, superior skill in machine learning using Java is a highly desirable asset in the market today.

Machine learning has been around in some form—if only in the imagination of thinkers, in the beginning—for a long time. More recent developments, however, have had a radical impact in many spheres of our everyday lives. Machine learning has much in common with statistics, artificial intelligence, and several other related areas. Whereas some data management, business intelligence, and knowledge representation systems may also be related in the central role of data in each of them, they are not commonly associated with principles of learning from data as embodied in the field of machine learning.

Any discourse on machine learning would assume an understanding of what data is and what data types we are concerned with. Are they categorical, continuous, or ordinal? What are the data features? What is the target, and which ones are predictors? What kinds of sampling methods can be used—uniform random, stratified random, cluster, or systematic sampling? What is the model? We saw an example dataset for weather data that included categorical and continuous features in the ARFF format.

The types of machine learning include supervised learning, the most common when labeled data is available, unsupervised when it's not, and semi-supervised when we have a mix of both. The chapters that follow will go into detail on these, as well as graph mining, probabilistic graph modeling, deep learning, stream learning, and learning with Big Data.

Data comes in many forms: structured, unstructured, transactional, sequential, and graphs. We will use data of different structures in the exercises to follow later in this book.

The list of domains and the different kinds of machine learning applications keeps growing. This review presents the most active areas and applications.

Understanding and dealing effectively with practical issues, such as noisy data,

skewed datasets, overfitting, data volumes, and the curse of dimensionality, is the key to successful projects—it's what makes each project unique in its challenges.

Analytics with machine learning is a collaborative endeavor with multiple roles and well-defined processes. For consistent and reproducible results, adopting the enhanced CRISP methodology outlined here is critical—from understanding the business problem to data quality analysis, modeling and model evaluation, and finally to model performance monitoring.

Practitioners of data science are blessed with a rich and growing catalog of datasets available to the public and an increasing set of ML frameworks and tools in Java as well as other languages. In the following chapters, you will be introduced to several datasets, APIs, and frameworks, along with advanced concepts and techniques to equip you with all you will need to attain mastery in machine learning.

Ready? Onward then!

Chapter 2. Practical Approach to Real-World Supervised Learning

The ability to learn from observations accompanied by marked targets or labels, usually in order to make predictions about unseen data, is known as **supervised machine learning**. If the targets are categories, the problem is one of classification and if they are numeric values, it is called **regression**. In effect, what is being attempted is to infer the function that maps the data to the target. Supervised machine learning is used extensively in a wide variety of machine learning applications, whenever labeled data is available or the labels can be added manually.

The core assumption of supervised machine learning is that the patterns that are learned from the data used in training will manifest themselves in yet unseen data.

In this chapter, we will discuss the steps used to explore, analyze, and pre-process the data before proceeding to training models. We will then introduce different modeling techniques ranging from simple linear models to complex ensemble models. We will present different evaluation metrics and validation criteria that allow us to compare model performance. Some of the discussions are accompanied by brief mathematical explanations that should help express the concepts more precisely and whet the appetite of the more mathematically inclined readers. In this chapter, we will focus on classification as a method of supervised learning, but the principles apply to both classification and regression, the two broad applications of supervised learning.

Beginning with this chapter, we will introduce tools to help illustrate how the concepts presented in each chapter are used to solve machine learning problems. Nothing reinforces the understanding of newly learned material better than the opportunity to apply that material to a real-world problem directly. In the process, we often gain a clearer and more relatable understanding of the subject than what is possible with passive absorption of the theory alone. If the opportunity to learn new tools is part of the learning, so much the better! To meet this goal, we will introduce a classification dataset familiar to most data science practitioners and use it to solve a classification problem while highlighting the process and methodologies that guide the solution.

In this chapter, we will use RapidMiner and Weka for building the process by which

we learn from a single well-known dataset. The workflows and code are available on the website for readers to download, execute, and modify.

RapidMiner is a GUI-based Java framework that makes it very easy to conduct a data science project, end-to-end, from within the tool. It has a simple drag-and-drop interface to build process workflows to ingest and clean data, explore and transform features, perform training using a wide selection of machine learning algorithms, do validation and model evaluation, apply your best models to test data, and more. It is an excellent tool to learn how to make the various parts of the process work together and produce rapid results. Weka is another GUI-based framework and it has a Java API that we will use to illustrate more of the coding required for performing analysis.

The major topics that we will cover in this chapter are:

- Data quality analysis
- Descriptive data analysis
- Visualization analysis
- Data transformation and preprocessing
- Data sampling
- Feature relevance analysis and dimensionality reduction
- Model building
- Model assessment, evaluation, and comparison
- Detailed case study—Horse Colic Classification

Formal description and notation

We would like to introduce some notation and formal definitions for the terms used in supervised learning. We will follow this notation through the rest of the book when not specified and extend it as appropriate when new concepts are encountered. The notation will provide a precise and consistent language to describe the terms of art and enable a more rapid and efficient comprehension of the subject.

- **Instance:** Every observation is a data instance. Normally the variable X is used to represent the input space. Each data instance has many variables (also called features) and is referred to as \mathbf{x} (vector representation with bold) of dimension d where d denotes the number of variables or features or attributes in each instance. The features are represented as $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$, where each value is a scalar when it is numeric corresponding to the feature value.
- **Label:** The label (also called target) is the dependent variable of interest, generally denoted by y . In **classification**, values of the label are well-defined categories in the problem domain; they need not be numeric or things that can be ordered. In **regression**, the label is real-valued.
- **Binary classification**, where the target takes only two values, it is mathematically represented as:

$$y \in \{1, -1\}$$

- **Regression**, where the target can take any value in the real number domain, is represented as:

$$y \in \mathbb{R}$$

- **Dataset:** Generally, the dataset is denoted by D and consists of individual data instances and their labels. The instances are normally represented as set $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. The labels for each instance are represented as the set $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$. The entire labeled dataset is represented as paired elements in a set as given by $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ for real-valued features.

Data quality analysis

There are limitations to what can be learned from data that suffers from poor quality. Problems with quality can include, among other factors, noisy data, missing values, and errors in labeling. Therefore, the first step is to understand the data before us in order that we may determine how to address any data quality issues. Are the outliers merely noise or indicative of interesting anomalies in the population? Should missing data be handled the same way for all features? How should sparse features be treated? These and similar questions present themselves at the very outset.

If we're fortunate, we receive a cleansed, accurately labeled dataset accompanied by documentation describing the data elements, the data's pedigree, and what if any transformations were already done to the data. Such a dataset would be ready to be split into train, validation, and test samples, using methods described in the section on Data Sampling. However, if data is not cleansed and suitable to be partitioned for our purposes, we must first prepare the data in a principled way before sampling can begin. (The significance of partitioning the data is explained later in this chapter in a section dedicated to train, validation, and test sets).

In the following sections, we will discuss the data quality analysis and transformation steps that are needed before we can analyze the features.

Descriptive data analysis

The complete data sample (including train, validation, and test) should be analyzed and summarized for the following characteristics. In cases where the data is not already split into train, validate, and test, the task of data transformation needs to make sure that the samples have similar characteristics and statistics. This is of paramount importance to ensure that the trained model can generalize over unseen data, as we will learn in the section on data sampling.

Basic label analysis

The first step of analysis is understanding the distribution of labels in different sets as well as in the data as a whole. This helps to determine whether, for example, there is imbalance in the distribution of the target variable, and if so, whether it is consistent across all the samples. Thus, the very first step is usually to find out how many examples in the training and test sets belong to each class.

Basic feature analysis

The next step is to calculate the statistics for each feature, such as

- Number of unique values
- Number of missing values: May include counts grouped by different missing value surrogates (NA, null, ?, and so on).
- For categorical: This counts across feature categories, counts across feature categories by label category, most frequently occurring category (mode), mode by label category, and so on.
- For numeric: Minimum, maximum, median, standard deviation, variance, and so on.

Feature analysis gives basic insights that can be a useful indicator of missing values and noise that can affect the learning process or choice of the algorithms.

Visualization analysis

Visualization of the data is a broad topic and it is a continuously evolving area in the field of machine learning and data mining. We will only cover some of the important aspects of visualization that help us analyze the data in practice.

Univariate feature analysis

The goal here is to visualize one feature at a time, in relation to the label. The techniques used are as follows:

Categorical features

Stacked bar graphs are a simple way of showing the distribution of each feature category among the labels, when the problem is one of classification.

Continuous features

Histograms and box plots are two basic visualization techniques for continuous features.

Histograms have predefined bins whose widths are either fixed intervals or based on some calculation used to split the full range of values of the feature. The number of instances of data that falls within each bin is then counted and the height of the bin is adjusted based on this count. There are variations of histograms such as relative or frequency-based histograms, Pareto histograms, two-dimensional histograms, and so on; each is a slight variation of the concept and permits a different insight into the feature. For those interested in finding out more about these variants, the Wikipedia article on histograms is a great resource.

Box plots are a key visualization technique for numeric features as they show distributions in terms of percentiles and outliers.

Multivariate feature analysis

The idea of multivariate feature analysis is to visualize more than one feature to get insights into relationships between them. Some of the well-known plots are explained here.

- **Scatter plots:** An important technique for understanding the relationship between different features and between features and labels. Typically, two-

dimensional scatter plots are used in practice where numeric features form the dimensions. Alignment of data points on some imaginary axis shows correlation while scattering of the data points shows no correlation. It can also be useful to identify clusters in lower dimensional space. A bubble chart is a variation of a scatter plot where two features form the dimensional axes and the third is proportional to the size of the data point, with the plot giving the appearance of a field of "bubbles". Density charts help visualize even more features together by introducing data point color, background color, and so on, to give additional insights.

- **ScatterPlot Matrix:** ScatterPlot Matrix is an extension of scatter plots where pair-wise scatter plots for each feature (and label) is visualized. It gives a way to compare and perform multivariate analysis of high dimensional data in an effective way.
- **Parallel Plots:** In this visualization, each feature is linearly arranged on the x-axis and the ranges of values for each feature form the y axis. So each data element is represented as a line with values for each feature on the parallel axis. Class labels, if available, are used to color the lines. Parallel plots offer a great understanding of features that are effective in separating the data. Deviation charts are variations of parallel plots, where instead of showing actual data points, mean and standard deviations are plotted. Andrews plots are another variation of parallel plots where data is transformed using Fourier series and the function values corresponding to each is projected.

Data transformation and preprocessing

In this section, we will cover the broad topic of data transformation. The main idea of data transformation is to take the input data and transform it in careful ways so as to clean it, extract the most relevant information from it, and to turn it into a usable form for further analysis and learning. During these transformations, we must only use methods that are designed while keeping in mind not to add any bias or artifacts that would affect the integrity of the data.

Feature construction

In the case of some datasets, we need to create more features from features we are already given. Typically, some form of aggregation is done using common aggregators such as average, sum, minimum, or maximum to create additional features. In financial fraud detection, for example, Card Fraud datasets usually contain transactional behaviors of accounts over various time periods during which the accounts were active. Performing behavioral synthesis such as by capturing the "Sum of Amounts whenever a Debit transaction occurred, for each Account, over One Day" is an example of feature construction that adds a new dimension to the dataset, built from existing features. In general, designing new features that enhance the predictive power of the data requires domain knowledge and experience with data, making it as much an art as a science.

Handling missing values

In real-world datasets, often, many features have missing values. In some cases, they are missing due to errors in measurement, lapses in recording, or because they are not available due to various circumstances; for example, individuals may choose not to disclose age or occupation. Why care about missing values at all? One extreme and not uncommon way to deal with it is to ignore the records that have any missing features, in other words, retain only examples that are "whole". This approach may severely reduce the size of the dataset when missing features are widespread in the data. As we shall see later, if the system we are dealing with is complex, dataset size can afford us precious advantage. Besides, there is often predictive value that can be exploited even in the "un-whole" records, despite the missing values, as long as we use appropriate measures to deal with the problem. On the other hand, one may unwittingly be throwing out key information when the omission of the data itself is significant, as in the case of deliberate misrepresentation or obfuscation on a loan application by withholding information that could be used to conclusively establish bone fides.

Suffice it to say, that an important step in the learning process is to adopt some systematic way to handle missing values and understand the consequences of the decision in each case. There are some algorithms such as Naïve Bayes that are less sensitive to missing values, but in general, it is good practice to handle these missing values as a pre-processing step before any form of analysis is done on the data. Here are some of the ways to handle missing values.

- **Replacing by means and modes:** When we replace the missing value of a continuous value feature with the mean value of that feature, the new mean clearly remains the same. But if the mean is heavily influenced by outliers, a better approach may be to use the mean after dropping the outliers from the calculation, or use the median or mode, instead. Likewise, when a feature is sparsely represented in the dataset, the mean value may not be meaningful. In the case of features with categorical values, replacing the missing value with the one that occurs with the highest frequency in the sample makes for a reasonable choice.
- **Replacing by imputation:** When we impute a missing value, we are in effect constructing a classification or regression model of the feature and making a prediction based on the other features in the record in order to classify or estimate the missing value.

- **Nearest Neighbor imputation:** For missing values of a categorical feature, we consider the feature in question to be the target and train a KNN model with k taken to be the known number of distinct categories. This model is then used to predict the missing values. (A KNN model is non-parametric and assigns a value to the "incoming" data instance based on a function of its neighbors—the algorithm is described later in this chapter when we talk about non-linear models).
- **Regression-based imputation:** In the case of continuous value variables, we use linear models like Linear Regression to estimate the missing data—the principle is the same as for categorical values.
- **User-defined imputation:** In many cases, the most suitable value for imputing missing values must come from the problem domain. For instance, a pH value of 7.0 is neutral, higher is basic, and lower is acidic. It may make most sense to impute a neutral value for pH than either mean or median, and this insight is an instance of a user-defined imputation. Likewise, in the case of substitution with normal body temperature or resting heart rate—all are examples from medicine.

Outliers

Handling outliers requires a lot of care and analysis. Outliers can be noise or errors in the data, or they can be anomalous behavior of particular interest. The latter case is treated in depth in [Chapter 3, Unsupervised Machine Learning Techniques](#). Here we assume the former case, that the domain expert is satisfied that the values are indeed outliers in the first sense, that is, noise or erroneously acquired or recorded data that needs to be handled appropriately.

Following are different techniques in detecting outliers in the data

- **Interquartile Ranges (IQR):** Interquartile ranges are a measure of variability in the data or, equivalently, the statistical dispersion. Each numeric feature is sorted based on its value in the dataset and the ordered set is then divided into quartiles. The median value is generally used to measure central tendency. IQR is measured as the difference between upper and lower quartiles, $Q3 - Q1$. The outliers are generally considered to be data values above $Q3 + 1.5 * \text{IQR}$ and below $Q1 - 1.5 * \text{IQR}$.
- **Distance-based Methods:** The most basic form of distance-based methods uses **k-Nearest Neighbors (k-NN)** and distance metrics to score the data points. The usual parameter is the value k in k-NN and a distance metric such as Euclidean distance. The data points at the farthest distance are considered outliers. There are many variants of these that use local neighborhoods, probabilities, or other factors, which will all be covered in [Chapter 3, Unsupervised Machine Learning Techniques](#). Mixed datasets, which have both categorical and numeric features, can skew distance-based metrics.
- **Density-based methods:** Density-based methods calculate the proportion of data points within a given distance D , and if the proportion is less than the specified threshold p , it is considered an outlier. The parameter p and D are considered user-defined values; the challenge of selecting these values appropriately presents one of the main hurdles in using these methods in the preprocessing stage.
- **Mathematical transformation of feature:** With non-normal data, comparing the mean value is highly misleading, as in the case when outliers are present. Non-parametric statistics allow us to make meaningful observations about highly skewed data. Transformation of such values using the logarithm or square root function tends to normalize the data in many cases, or make them more amenable to statistical tests. These transformations alter the shape of the

distribution of the feature drastically—the more extreme an outlier, the greater the effect of the log transformation, for example.

- **Handling outliers using robust statistical algorithms in machine learning models:** Many classification algorithms which we discuss in the next section on modeling, implicitly or explicitly handle outliers. Bagging and Boosting variants, which work as meta-learning frameworks, are generally resilient to outliers or noisy data points and may not need a preprocessing step to handle them.
- **Normalization:** Many algorithms—distance-based methods are a case in point—are very sensitive to the scale of the features. Preprocessing the numeric features makes sure that all of them are in a well-behaved range. The most well-known techniques of normalization of features are given here:

- **Min-Max Normalization:** In this technique, given the range $[L, U]$, which is typically $[0, 1]$, each feature with value x is normalized in terms of the minimum and maximum values, x_{\max} and x_{\min} , respectively, using the formula:

$$x' = \frac{(x - x_{\min})}{(x_{\max} - x_{\min})} * (U - L) + L$$

- **Z-Score Normalization:** In this technique, also known as standardization, the feature values get auto-transformed so that the mean is 0 and standard deviation is 1. The technique to transform is as follows: for each feature f , the mean value $\mu(f)$ and standard deviation $\sigma(f)$ are computed and then the feature with value x is transformed as:

$$x' = \frac{(x - \mu(f))}{\sigma(f)}$$

Discretization

Many algorithms can only handle categorical values or nominal values to be effective, for example Bayesian Networks. In such cases, it becomes imperative to discretize the numeric features into categories using either supervised or unsupervised methods. Some of the techniques discussed are:

- **Discretization by binning:** This technique is also referred to as equal width discretization. The entire scale of data for each feature f , ranging from values x_{\max} and x_{\min} is divided into a predefined number, k , of equal intervals, each

$$w = \frac{(x_{\max} - x_{\min})}{k}$$

having the width w . The "cut points" or discretization intervals are:

$$\{x_{\min} + w, \quad x_{\min} + 2w, \quad \dots, \quad x_{\min} + (k-1)w\}$$

- **Discretization by frequency:** This technique is also referred to as equal frequency discretization. The feature is sorted and then the entire data is discretized into predefined k intervals, such that each interval contains the same proportion. Both the techniques, discretization by binning and discretization by frequency, suffer from loss of information due to the predefined value of k .
- **Discretization by entropy:** Given the labels, the entropy is calculated over the split points where the value changes in an iterative way, so that the bins of intervals are as pure or discriminating as possible. Refer to the *Feature evaluation techniques* section for entropy-based (information gain) theory and calculations.

Data sampling

The dataset one receives may often require judicious sampling in order to effectively learn from the data. The characteristics of the data as well as the goals of the modeling exercise determine whether sampling is needed, and if so, how to go about it. Before we begin to learn from this data it is crucially important to create train, validate, and test data samples, as explained in this section.

Is sampling needed?

When the dataset is large or noisy, or skewed towards one type, the question as to whether to sample or not to sample becomes important. The answer depends on various aspects such as the dataset itself, the objective and the evaluation criteria used for selecting the models, and potentially other practical considerations. In some situations, algorithms have scalability issues in memory and space, but work effectively on samples, as measured by model performance with respect to the regression or classification goals they are expected to achieve. For example, SVM scales as $O(n^2)$ and $O(n^3)$ in memory and training times, respectively. In other situations, the data is so imbalanced that many algorithms are not robust enough to handle the skew. In the literature, the step intended to re-balance the distribution of classes in the original data extract by creating new training samples is also called **resampling**.

Undersampling and oversampling

Datasets exhibiting a marked imbalance in the distribution of classes can be said to contain a distinct minority class. Often, this minority class is the set of instances that we are especially interested in precisely because its members occur in such rare cases. For example, in credit card fraud, less than 0.1% of the data belongs to fraud. This skewness is not conducive to learning; after all, when we seek to minimize the total error in classification, we give equal weight to all classes regardless of whether one class is underrepresented compared to another. In binary classification problems, we call the minority class the positive class and the majority class as the negative class, a convention that we will follow in the following discussion.

Undersampling of the majority class is a technique that is commonly used to address skewness in data. Taking credit-card fraud as an example, we can create different training samples from the original dataset such that each sample has all the fraud cases from the original dataset, whereas the non-fraud instances are distributed

across all the training samples in some fixed ratios. Thus, in a given training set created by this method, the majority class is now underrepresented compared to the original skewed dataset, effectively balancing out the distribution of classes. Training samples with labeled positive and labeled negative instances in ratios of, say, 1:20 to 1:50 can be created in this way, but care must be taken that the sample of negative instances used should have similar characteristics to the data statistics and distributions of the main datasets. The reason for using multiple training samples, and in different proportions of positive and negative instances, is so that any sampling bias that may be present becomes evident.

Alternatively, we may choose to oversample the minority class. As before, we create multiple samples wherein instances from the minority class have been selected by either sampling with replacement or without replacement from the original dataset. When sampling without replacement, there are no replicated instances across samples. With replacement, some instances may be found in more than one sample. After this initial seeding of the samples, we can produce more balanced distributions of classes by random sampling with replacement from within the minority class in each sample until we have the desired ratios of positive to negative instances. Oversampling can be prone to over-fitting as classification decision boundaries tend to become more specific due to replicated values. **SMOTE (Synthetic Minority Oversampling Technique)** is a technique that alleviates this problem by creating synthetic data points in the interstices of the feature space by interpolating between neighboring instances of the positive class (*References* [20]).

Stratified sampling

Creating samples so that data with similar characteristics is drawn in the same proportion as they appear in the population is known as stratified sampling. In multi-class classification, if there are N classes each in a certain proportion, then samples are created such that they represent each class in the same proportion as in the original dataset. Generally, it is good practice to create multiple samples to train and test the models to validate against biases of sampling.

Training, validation, and test set

The Holy Grail of creating good classification models is to train on a set of good quality, representative, (training data), tune the parameters and find effective models (validation data), and finally, estimate the model's performance by its behavior on unseen data (test data).

The central idea behind the logical grouping is to make sure models are validated or tested on data that has not been seen during training. Otherwise, a simple "rote learner" can outperform the algorithm. The generalization capability of the learning algorithm must be evaluated on a dataset which is different from the training dataset, but comes from the same population (*References [11]*). The balance between removing too much data from training to increase the budget of validation and testing can result in models which suffer from "underfitting", that is, not having enough examples to build patterns that can help in generalization. On the other hand, the extreme choice of allocating all the labeled data for training and not performing any validation or testing can lead to "overfitting", that is, models that fit the examples too faithfully and do not generalize well enough.

Typically, in most machine learning challenges and real world customer problems, one is given a training set and testing set upfront for evaluating the performance of the models. In these engagements, the only question is how to validate and find the most effective parameters given the training set. In some engagements, only the labeled dataset is given and you need to consider the training, validation, and testing sets to make sure your models do not overfit or underfit the data.

Three logical processes are needed for modeling and hence three logical datasets are needed, namely, training, validation, and testing. The purpose of the training dataset is to give labeled data to the learning algorithm to build the models. The purpose of the validation set is to see the effect of the parameters of the training model being evaluated by training on the validation set. Finally, the best parameters or models are retrained on the combination of the training and validation sets to find an optimum model that is then tested on the blind test set.

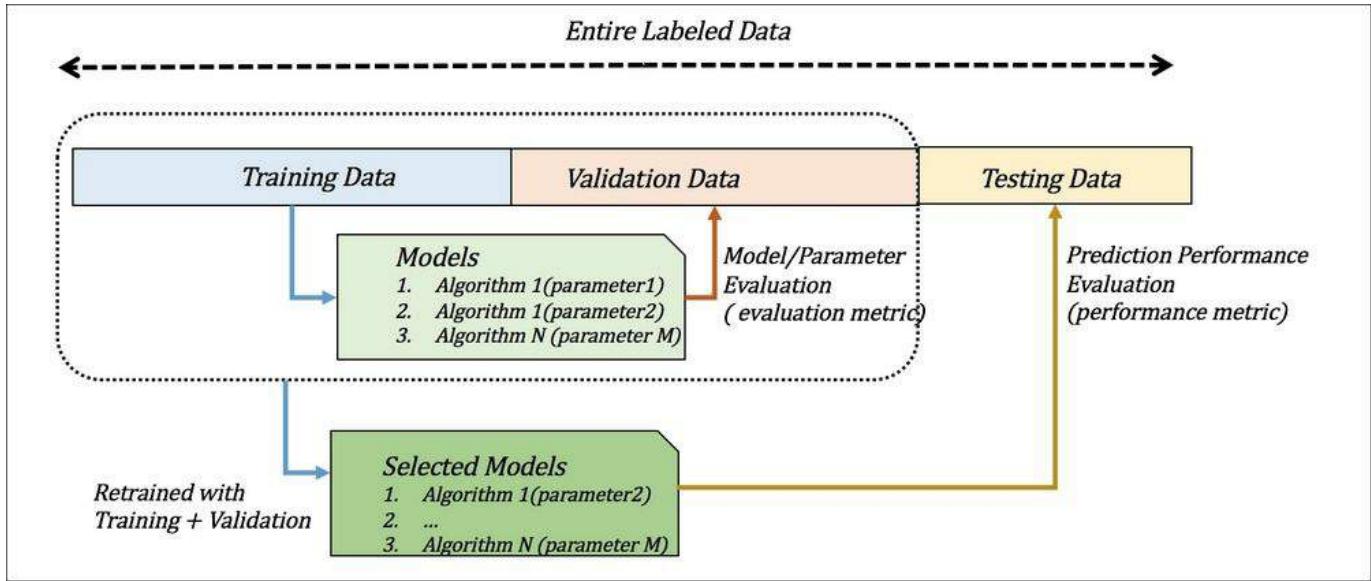


Figure 1: Training, Validation, and Test data and how to use them

Two things affect the learning or the generalization capability: the choice of the algorithm (and its parameters) and number of training data. This ability to generalize can be estimated by various metrics including the prediction errors. The overall estimate of unseen error or risk of the model is given by:

$$\mathbb{E}(Risk) = Noise + \mathbb{E}_x(Var(G, n)) + \mathbb{E}_x(Bias_x^2(G, n))$$

Here, *Noise* is the stochastic noise, *Var (G,n)* is called the variance error and is a measure of how susceptible our hypothesis or the algorithm (*G*) is, if given different datasets. $Bias_x^2(G, n)$ is called the bias error and represents how far away the best algorithm in the model (average learner over all possible datasets) is from the optimal one.

Learning curves as shown in *Figure 2* and *Figure 3*—where training and testing errors are plotted keeping either the algorithm with its parameters constant or the training data size constant—give an indication of underfitting or overfitting.

When the training data size is fixed, different algorithms or the same algorithms with

different parameter choices can exhibit different learning curves. The *Figure 2* shows two cases of algorithms on the same data size giving two different learning curves based on bias and variance.

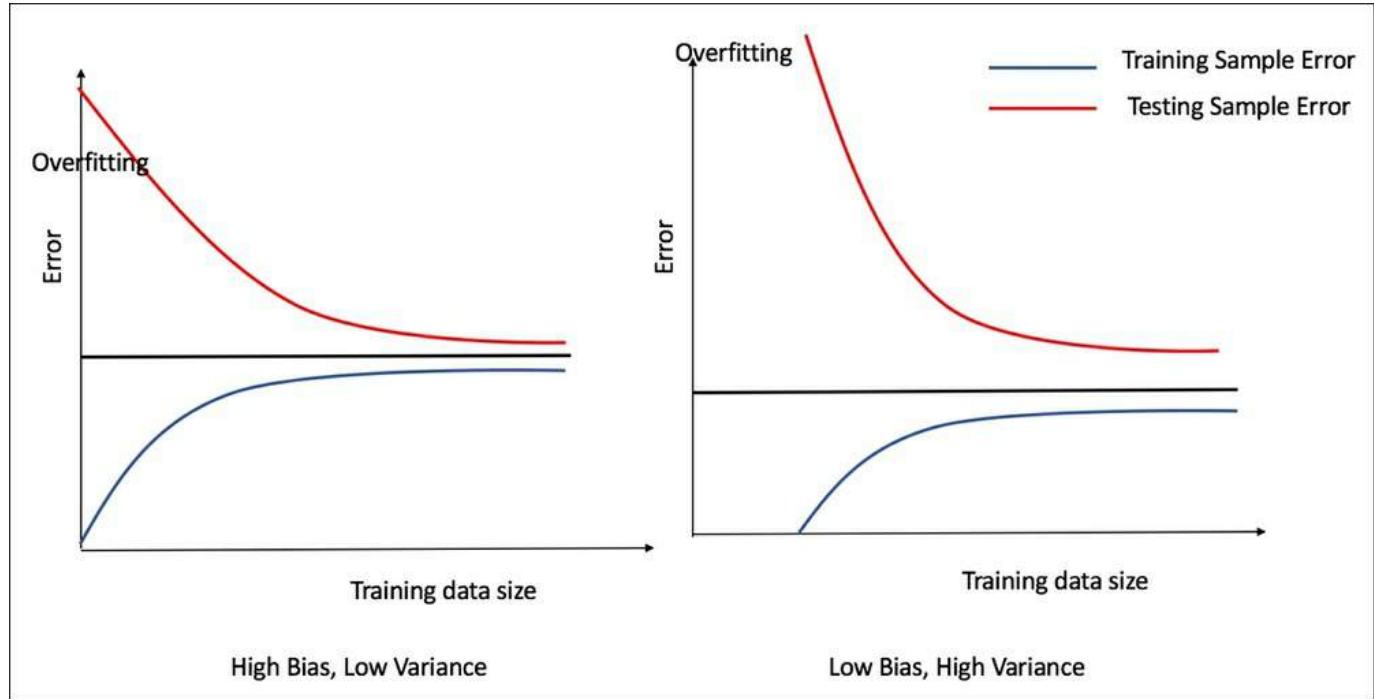


Figure 2: The training data relationship with error rate when the model complexity is fixed indicates different choices of models.

The algorithm or model choice also impacts model performance. A complex algorithm, with more parameters to tune, can result in overfitting, while a simple algorithm with less parameters might be underfitting. The classic figure to illustrate the model performance and complexity when the training data size is fixed is as follows:

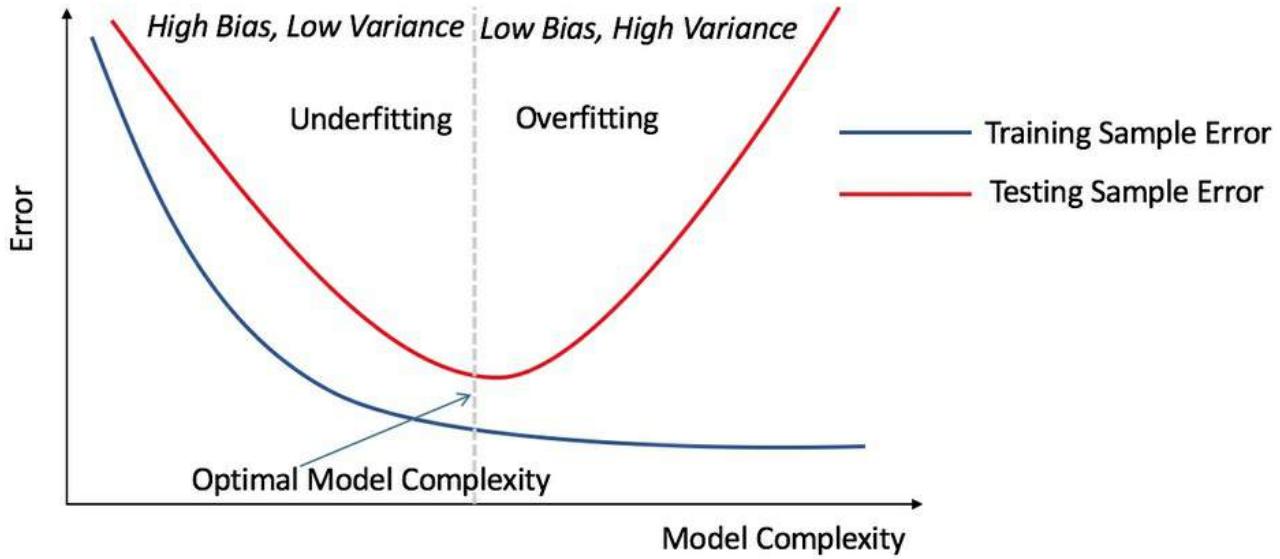


Figure 3: The Model Complexity relationship with Error rate, over the training and the testing data when training data size is fixed.

Validation allows for exploring the parameter space to find the model that generalizes best. Regularization (will be discussed in linear models) and validation are two mechanisms that should be used for preventing overfitting. Sometimes the "k-fold cross-validation" process is used for validation, which involves creating k samples of the data and using $(k - 1)$ to train on and the remaining one to test, repeated k times to give an average estimate. The following figure shows 5-fold cross-validation as an example:

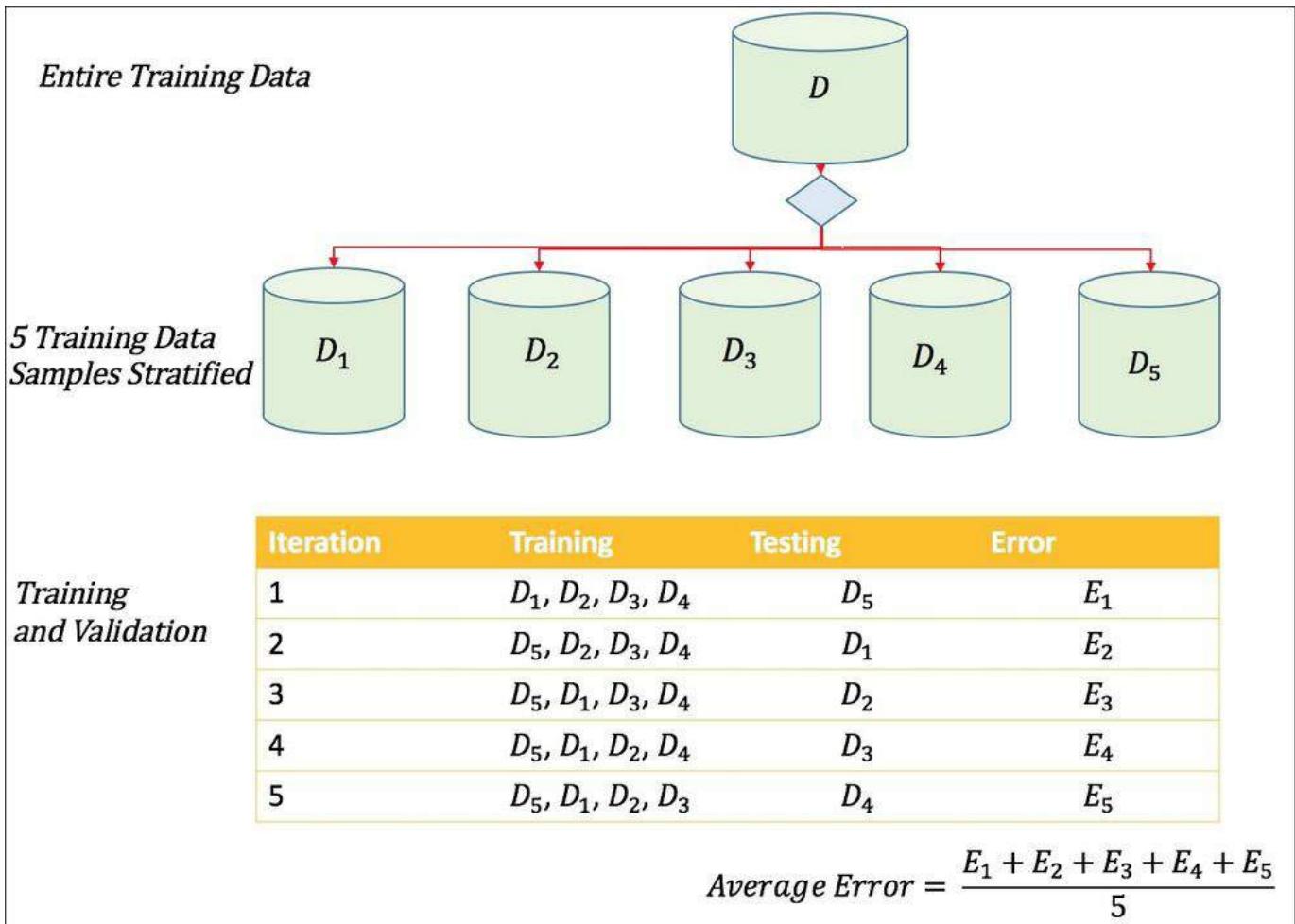


Figure 4: 5-fold cross-validation.

The following are some commonly used techniques to perform data sampling, validation, and learning:

- **Random split of training, validation, and testing:** 60, 20, 20. Train on 60%, use 20% for validation, and then combine the train and validation datasets to train a final model that is used to test on the remaining 20%. Split may be done randomly, based on time, based on region, and so on.
- **Training, cross-validation, and testing:** Split into Train and Test two to one, do validation using cross-validation on the train set, train on whole two-thirds and test on one-third. Split may be done randomly, based on time, based on region, and so on.
- **Training and cross-validation:** When the training set is small and only model selection can be done without much parameter tuning. Run cross-validation on

the whole dataset and chose the best models with learning on the entire dataset.

Feature relevance analysis and dimensionality reduction

The goal of feature relevance and selection is to find the features that are discriminating with respect to the target variable and help reduce the dimensions of the data [1,2,3]. This improves the model performance mainly by ameliorating the effects of the curse of dimensionality and by removing noise due to irrelevant features. By carefully evaluating models on the validation set with and without features removed, we can see the impact of feature relevance. Since the exhaustive search for k features involves $2^k - 1$ sets (consider all combinations of k features where each feature is either retained or removed, disregarding the degenerate case where none is present) the corresponding number of models that have to be evaluated can become prohibitive, so some form of heuristic search techniques are needed. The most common of these techniques are described next.

Feature search techniques

Some of the very common search techniques employed to find feature sets are:

- **Forward or hill climbing:** In this search, one feature is added at a time until the evaluation module outputs no further change in performance.
- **Backward search:** Starting from the whole set, one feature at a time is removed until no performance improvement occurs. Some applications interleave both forward and backward techniques to search for features.
- **Evolutionary search:** Various evolutionary techniques such as genetic algorithms can be used as a search mechanism and the evaluation metrics from either filter- or wrapper-based methods can be used as fitness criterion to guide the process.

Feature evaluation techniques

At a high level, there are three basic methods to evaluate features.

Filter approach

This approach refers to the use of techniques without using machine learning algorithms for evaluation. The basic idea of the filter approach is to use a search technique to select a feature (or subset of features) and measure its importance using some statistical measure until a stopping criterion is reached.

Univariate feature selection

This search is as simple as ranking each feature based on the statistical measure employed.

Information theoretic approach

All information theoretic approaches use the concept of entropy mechanism at their core. The idea is that if the feature is randomly present in the dataset, there is maximum entropy, or, equivalently, the ability to compress or encode is low, and the feature may be irrelevant. On the other hand, if the distribution of the feature value is such some range of values are more prevalent in one class relative to the others, then the entropy is minimized and the feature is discriminating. Casting the problem in terms of entropy in this way requires some form of discretization to convert the numeric features into categories in order to compute the probabilities.

Consider a binary classification problem with training data D_X . If X_i is the i^{th} feature with v distinct categorical values such that $D_{Xi} = \{D_1, D_2, \dots, D_v\}$, then information or the entropy in feature X_i is:

$$\text{Info}(D_{X_i}) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{Info}(D_j)$$

Here, $\text{Info}(D_j)$ is the entropy of the partition and is calculated as:

$$Info(D) = -p_+(D) \log_2 p_+(D) - p_-(D) \log_2 p_-(D)$$

Here, $p_+(D)$ is the probability that the data in set D is in the positive class and $p_-(D)$ is the probability that it is in the negative class, in that sample. Information gain for the feature is calculated in terms of the overall information and information of the feature as

$$InfoGain(X_i) = Info(D) - Info(D_{X_i})$$

For numeric features, the values are sorted in ascending order and split points between neighboring values are considered as distinct values.

The greater the decrease in entropy, the higher the relevance of the feature. Information gain has problems when the feature has a large number of values; that is when Gain Ratio comes in handy. Gain Ratio corrects the information gain over large splits by introducing Split Information. Split Information for feature X_i and $GainRatio$ is given by:

$$SplitInfo(D_{X_i}) = \sum_{j=1}^v \frac{|D_j|}{|D|} \log_2 \left(\frac{|D_j|}{|D|} \right)$$

$$GainRatio(X_i) = \frac{InfoGain(X_i)}{SplitInfo(D_{X_i})}$$

There are other impurity measures such as Gini Impurity Index (as described in the section on the *Decision Tree* algorithm) and Uncertainty-based measures to compute

feature relevance.

Statistical approach

Chi-Squared feature selection is one of the most common feature selection methods that has statistical hypothesis testing as its base. The null hypothesis is that the feature and the class variable are independent of each other. The numeric features are discretized so that all features have categorical values. The contingency table is calculated as follows:

Feature Values	Class=P	Class=N	Sum over classes $n_iP + n_iN$
X_1	$(n_{1P} \mu_{1P})$	$(n_{1N} \mu_{1N})$	n_1
....
X_m	$(n_{mP} \mu_{mP})$	$(n_{mN} \mu_{mN})$	n_m
	n^*P	n^*P	n

Contingency Table 1: Showing feature values and class distribution for binary class.

In the preceding table, n_{ij} is a count of the number of features with value—after discretization—equal to x_i and class value of j .

The value summations are:

$$n_{*p} = \sum_1^m n_{ij}$$

$$n_{i*} = n_{iP} + n_{iN}$$

$$n = n_{*P} + n_{*N}$$

$$\mu_{ij} = \frac{(n_{*j} * n_{i*})}{n}$$

Here n is number of data instances, $j = P, N$ is the class value and $i = 1, 2, \dots, m$ indexes the different discretized values of the feature and the table has $m - 1$ degrees of freedom.

The Chi-Square Statistic is given by:

$$\chi^2 = \sum_{i=1}^m \left(\left(\frac{(n_{iP} - \mu_{iP})^2}{\mu_{iP}} \right) + \left(\frac{(n_{iN} - \mu_{iN})^2}{\mu_{iN}} \right) \right)$$

The Chi-Square value is compared to confidence level thresholds for testing significance. For example, for $i = 2$, the Chi-Squared value at threshold of 5% is 3.84; if our value is smaller than the table value of 3.83, then we know that the feature is interesting and the null hypothesis is rejected.

Multivariate feature selection

Most multivariate methods of feature selection have two goals:

- Reduce the redundancy between the feature and other selected features
- Maximize the relevance or correlation of the feature with the class label

The task of finding such subsets of features cannot be exhaustive as the process can have a large search space. Heuristic search methods such as backward search,

forward search, hill-climbing, and genetic algorithms are typically used to find a subset of features. Two very well-known evaluation techniques for meeting the preceding goals are presented next.

Minimal redundancy maximal relevance (mRMR)

In this technique, numeric features are often discretized—as done in univariate pre-processing—to get distinct categories of values.

For each subset S , the redundancy between two features X_i and X_j can be measured as:

$$W_I(S) = \frac{1}{|S|^2} \sum_{X_i, X_j \in S} MI(X_i, X_j)$$

Here, $MI(X_i, X_j)$ = measure of mutual information between two features X_i and X_j . Relevance between feature X_i and class C can be measured as:

$$V_I(S) = \frac{1}{|S|} \sum_{X_i \in S} MI(X_i, C)$$

Also, the two goals can be combined to find the best feature subset using:

$$S^2 = \underset{S \subseteq U}{\operatorname{argmax}} (V_I(S) - W_I(S))$$

Correlation-based feature selection (CFS)

The basic idea is similar to the previous example; the overall merit of subset S is measured as:

$$Merit(S) = \frac{k \bar{r}_{cf}}{\sqrt{k + (k-1) \bar{r}_{ff}}}$$

Here, k is the total number of features, \bar{r}_{cf} is the average feature class correlation and \bar{r}_{ff} is the average feature-feature inter correlation. The numerator gives the relevance factor while the denominator gives the redundancy factor and hence the goal of the search is to maximize the overall ratio or the $Merit(S)$.

There are other techniques such as Fast-Correlation-based feature selection that is based on the same principles, but with variations in computing the metrics. Readers can experiment with this and other techniques in Weka.

The advantage of the Filter approach is that its methods are independent of learning algorithms and hence one is freed from choosing the algorithms and parameters. They are also faster than wrapper-based approaches.

Wrapper approach

The search technique remains the same as discussed in the feature search approach; only the evaluation method changes. In the wrapper approach, a machine learning algorithm is used to evaluate the subset of features that are found to be discriminating based on various metrics. The machine learning algorithm used as the wrapper approach may be the same or different from the one used for modeling.

Most commonly, cross-validation is used in the learning algorithm. Performance metrics such as area under curve or F-score, obtained as an average on cross-validation, guide the search process. Since the cost of training and evaluating models is very high, we choose algorithms that have fast training speed, such as Linear Regression, linear SVM, or ones that are Decision Tree-based.

Some wrapper approaches have been very successful using specific algorithms such as Random Forest to measure feature relevance.

Embedded approach

This approach does not require feature search techniques. Instead of performing feature selection as preprocessing, it is done in the machine learning algorithm itself. Rule Induction, Decision Trees, Random Forest, and so on, perform feature selection as part of the training algorithm. Some algorithms such as regression or SVM-based methods, known as **shrinking methods**, can add a regularization term in the model to overcome the impact of noisy features in the dataset. Ridge and lasso-based regularization are well-known techniques available in regressions to provide feature selection implicitly.

There are other techniques using unsupervised algorithms that will be discussed in [Chapter 3, Unsupervised Machine Learning Techniques](#), that can be used effectively in a supervised setting too, for example, **Principal Component Analysis (PCA)**.

Model building

In real-world problems, there are many constraints on learning and many ways to assess model performance on unseen data. Each modeling algorithm has its strengths and weaknesses when applied to a given problem or to a class of problems in a particular domain. This is articulated in the famous **No Free Lunch Theorem (NFLT)**, which says—for the case of supervised learning—that averaged over all distributions of data, every classification algorithm performs about as well as any other, including one that always picks the same class! Application of NFLT to supervised learning and search and optimization can be found at <http://www.no-free-lunch.org/>.

In this section, we will discuss the most commonly used practical algorithms, giving the necessary details to answer questions such as what are the algorithm's inputs and outputs? How does it work? What are the advantages and limitations to consider while choosing the algorithm? For each model, we will include sample code and outputs obtained from testing the model on the chosen dataset. This should provide the reader with insights into the process. Some algorithms such as neural networks and deep learning, Bayesian networks, stream-based earning, and so on, will be covered separately in their own chapters.

Linear models

Linear models work well when the data is linearly separable. This should always be the first thing to establish.

Linear Regression

Linear Regression can be used for both classification and estimation problems. It is one of the most widely used methods in practice. It consists of finding the best-fitting hyperplane through the data points.

Algorithm input and output

Features must be numeric. Categorical features are transformed using various pre-processing techniques, as when a categorical value becomes a feature with 1 and 0 values. Linear Regression models output a categorical class in classification or numeric values in regression. Many implementations also give confidence values.

How does it work?

The model tries to learn a "hyperplane" in the input space that minimizes the error between the data points of each class (*References [4]*).

A hyperplane in d-dimensional inputs that linear model learns is given by:

$$G(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j$$

The two regions (binary classification) the model divides the input space into are $w_0 + \sum_{j=1}^d w_j x_j > 0$ and $w_0 + \sum_{j=1}^d w_j x_j < 0$. Associating a value of 1 to the coordinate of feature 0, that is, $x_0=1$, the vector representation of hypothesis space or the model is:

$$G(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$$

The weight matrix can be derived using various methods such as ordinary least squares or iterative methods using matrix notation as follows:

$$\hat{\mathbf{W}} = \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Here \mathbf{X} is the input matrix and \mathbf{y} is the label. If the matrix $\mathbf{X}^T \mathbf{X}$ in the least squares problem is not of full rank or if encountering various numerical stability issues, the solution is modified as:

$$\widehat{\mathbf{W}} = \mathbf{X}(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_n)^{-1} \mathbf{X}^T \mathbf{y}$$

Here, $\lambda \in \mathbb{R}$ is added to the diagonal of an identity matrix \mathbf{I}_n of size $(n + 1, n + 1)$ with the rest of the values being set to 0. This solution is called **ridge regression** and parameter λ theoretically controls the trade-off between the square loss and low norm of the solution. The constant λ is also known as regularization constant and helps in preventing "overfitting".

Advantages and limitations

- It is an appropriate method to try and get insights when there are less than 100 features and a few thousand data points.
- Interpretable to some level as the weights give insights on the impact of each feature.
- Assumes linear relationship, additive and uncorrelated features, hence it doesn't model complex non-linear real-world data. Some implementations of Linear Regression allow removing collinear features to overcome this issue.
- Very susceptible to outliers in the data, if there are huge outliers, they have to be treated prior to performing Linear Regression.
- Heteroskedasticity, that is, unequal training point variances, can affect the simple least square regression models. Techniques such as weighted least squares are employed to overcome this situation.

Naïve Bayes

Based on the Bayes rule, the Naïve Bayes classifier assumes the features of the data are independent of each other (*References* [9]). It is especially suited for large datasets and frequently performs better than other, more elaborate techniques, despite its naïve assumption of feature independence.

Algorithm input and output

The Naïve Bayes model can take features that are both categorical and continuous. Generally, the performance of Naïve Bayes models improves if the continuous features are discretized in the right format. Naïve Bayes outputs the class and the probability score for all class values, making it a good classifier for scoring models.

How does it work?

It is a probability-based modeling algorithm. The basic idea is using Bayes' rule and measuring the probabilities of different terms, as given here. Measuring probabilities can be done either using pre-processing such as discretization, assuming a certain distribution, or, given enough data, mapping the distribution for numeric features.

Bayes' rule is applied to get the posterior probability as predictions and k represents k^{th} class.:

$$P(y = y^k | \mathbf{x}) = \frac{P(y = y^k) * \prod_{j=1}^d P(x_j | y = y^k)}{P(\mathbf{x})}$$

$$P(y = y^k | \mathbf{x}) \propto P(y = y^k) * \prod_{j=1}^d P(x_j | y = y^k)$$

Advantages and limitations

- It is robust against isolated noisy data points because such points are averaged when estimating the probabilities of input data.
- Probabilistic scores as confidence values from Bayes classification can be used as scoring models.

- Can handle missing values very well as they are not used in estimating probabilities.
- Also, it is robust against irrelevant attributes. If the features are not useful the probability distribution for the classes will be uniform and will cancel itself out.
- Very good in training speed and memory, it can be parallelized as each computation of probability in the equation is independent of the other.
- Correlated features can be a big issue when using Naïve Bayes because the conditional independence assumption is no longer valid.
- Normal distribution of errors is an assumption in most optimization algorithms.

Logistic Regression

If we employ Linear Regression model using, say, the least squares regression method, the outputs have to be converted to classes, say 0 and 1. Many Linear Regression algorithms output class and confidence as probability. As a rule of thumb, if we see that the probabilities of Linear Regression are mostly beyond the ranges of 0.2 to 0.8, then logistic regression algorithm may be a better choice.

Algorithm input and output

Similar to Linear Regression, all features must be numeric. Categorical features have to be transformed to numeric. Like in Naïve Bayes, this algorithm outputs class and probability for each class and can be used as a scoring model.

How does it work?

Logistic regression models the posterior probabilities of classes using linear functions in the input features.

The logistic regression model for a binary classification is given as:

$$\log \frac{P(y=0|\mathbf{x})}{P(y=1|\mathbf{x})} = w_0 + \sum_{j=1}^d w_j x_j$$

The model is a log-odds or logit transformation of linear models (*References [6]*). The weight vector is generally computed using various optimization methods such as

iterative reweighted least squares (IRLS) or the **Broyden–Fletcher–Goldfarb–Shanno (BFGS)** method, or variants of these methods.

Advantages and limitations

- Overcomes the issue of heteroskedasticity and some non-linearity between inputs and outputs.
- No need of normal distribution assumptions in the error estimates.
- It is interpretable, but less so than Linear Regression models as some understanding of statistics is required. It gives information such as odds ratio, p values, and so on, which are useful in understanding the effects of features on the classes as well as doing implicit feature relevance based on significance of p values.
- L1 or L2 regularization has to be employed in practice to overcome overfitting in the logistic regression models.
- Many optimization algorithms are available for improving speed of training and robustness.

Non-linear models

Next, we will discuss some of the well-known, practical, and most commonly used non-linear models.

Decision Trees

Decision Trees are also known as **Classification and Regression Trees (CART)** (*References [5]*). Their representation is a binary tree constructed by evaluating an inequality in terms of a single attribute at each internal node, with each leaf-node corresponding to the output value or class resulting from the decisions in the path leading to it. When a new input is provided, the output is predicted by traversing the tree starting at the root.

Algorithm inputs and outputs

Features can be both categorical and numeric. It generates class as an output and most implementations give a score or probability using frequency-based estimation. Decision Trees probabilities are not smooth functions like Naïve Bayes and Logistic Regression, though there are extensions that are.

How does it work?

Generally, a single tree is created, starting with single features at the root with decisions split into branches based on the values of the features while at the leaf there is either a class or more features. There are many choices to be made, such as how many trees, how to choose features at the root level or at subsequent leaf level, and how to split the feature values when not categorical. This has resulted in many different algorithms or modifications to the basic Decision Tree. Many techniques to split the feature values are similar to what was discussed in the section on discretization. Generally, some form of pruning is applied to reduce the size of the tree, which helps in addressing overfitting.

Gini index is another popular technique used to split the features. Gini index of data in set S of all the data points is
$$G(S) = 1 - \sum_{j=1}^k p_j^2$$
 where $p_1, p_2 \dots p_k$ are probability distribution for each class.

If p is the fraction or probability of data in set S of all the data points belonging to say class positive, then $1 - p$ is the fraction for the other class or the error rate in

binary classification. If the dataset S is split in r ways S_1, S_2, \dots, S_r then the error rate of each set can be quantified as $|S_i|$. Gini index for an r way split is as follows:

$$GiniSplit(S \Rightarrow S_1, S_2, \dots, S_r) = \sum_{i=1}^r \frac{|S_i|}{|S|} G(S_i)$$

The split with the lowest Gini index is used for selection. The CART algorithm, a popular Decision Tree algorithm, uses Gini index for split criteria.

The entropy of the set of data points S can similarly be computed as:

$$E(S) = \sum_{j=1}^k p_j \log_2 p_j$$

Similarly, entropy-based split is computed as:

$$EntropySplit(S \Rightarrow S_1, S_2, \dots, S_r) = \sum_{i=1}^r \frac{|S_i|}{|S|} E(S_i)$$

The lower the value of the entropy split, the better the feature, and this is used in ID3 and C4.5 Decision Tree algorithms (*References [12]*).

The stopping criteria and pruning criteria are related. The idea behind stopping the growth of the tree early or pruning is to reduce the "overfitting" and it works similar to regularization in linear and logistic models. Normally, the training set is divided into tree growing sets and pruning sets so that pruning uses different data to overcome any biases from the growing set. **Minimum Description Length (MDL)**,

which penalizes the complexity of the tree based on number of nodes is a popular methodology used in many Decision Tree algorithms.

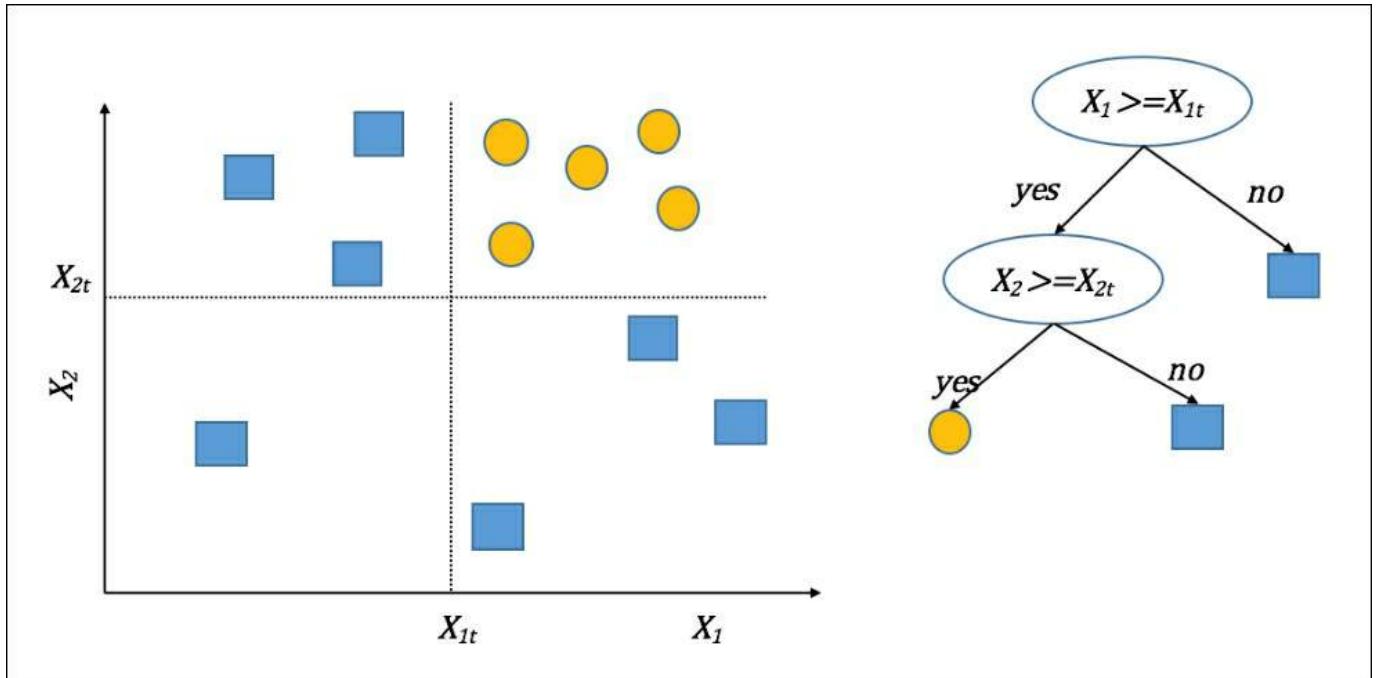


Figure 5: Shows a two-dimensional binary classification problem and a Decision Tree induced using splits at thresholds X_{1t} and X_{2t} , respectively

Advantages and limitations

- The main advantages of Decision Trees are they are quite easily interpretable. They can be understood in layman's terms and are especially suited for business domain experts to easily understand the exact model.
- If there are a large number of features, then building Decision Tree can take lots of training time as the complexity of the algorithm increases.
- Decision Trees have an inherent problem with overfitting. Many tree algorithms have pruning options to reduce the effect. Using pruning and validation techniques can alleviate the problem to a large extent.
- Decision Trees work well when there is correlation between the features.
- Decision Trees build axis-parallel boundaries across classes, the bias of which can introduce errors, especially in a complex, smooth, non-linear boundary.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors falls under the branch of non-parametric and lazy algorithms.

K-Nearest neighbors doesn't make any assumptions on the underlying data and doesn't build and generalize models from training data (*References* [10]).

Algorithm inputs and outputs

Though KNN's can work with categorical and numeric features, the distance computation, which is the core of finding the neighbors, works better with numeric features. Normalization of numeric features to be in the same ranges is one of the mandatory steps required. KNN's outputs are generally the classes based on the neighbors' distance calculation.

How does it work?

KNN uses the entire training data to make predictions on unseen test data. When unseen test data is presented KNN finds the K "nearest neighbors" using some distance computation and based on the neighbors and the metric of deciding the category it classifies the new point. If we consider two vectors represented by \mathbf{x}_1 and \mathbf{x}_2 corresponding to two data points the distance is calculated as:

- Euclidean Distance:

$$d_{EUC}(\mathbf{x}_1, \mathbf{x}_2) = \left[(\mathbf{x}_1 - \mathbf{x}_2)(\mathbf{x}_1 - \mathbf{x}_2) \right]^{1/2}$$

- Cosine Distance or similarity:

$$S_{\cos}(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}$$

The metric used to classify an unseen may simply be the majority class among the K neighbors.

The training time is small as all it has to do is build data structures to hold the data in such a way that the computation of the nearest neighbor is minimized when unseen data is presented. The algorithm relies on choices of how the data is stored from training data points for efficiency of searching the neighbors, which distance

computation is used to find the nearest neighbor, and which metrics are used to categorize based on classes of all neighbors. Choosing the value of "K" in KNN by using validation techniques is critical.

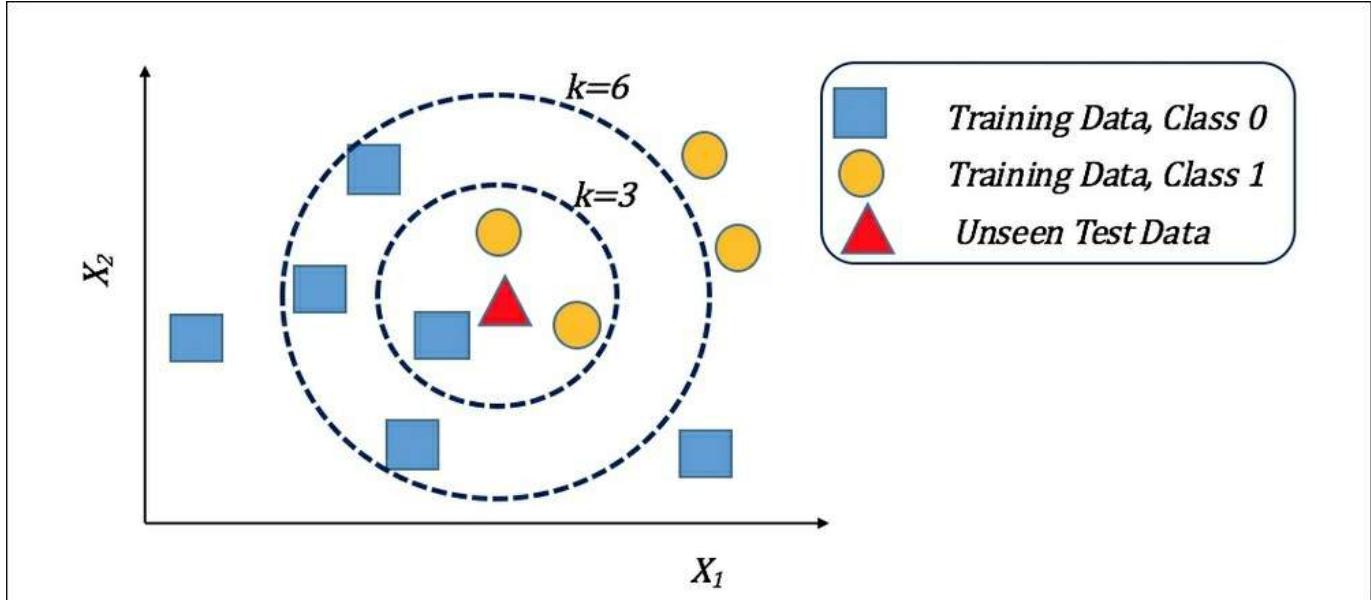


Figure 6: K-Nearest Neighbor illustrated using two-dimensional data with different choices of k.

Advantages and limitations

- No assumption on underlying data distribution and minimal training time makes it a very attractive method for learning.
- KNN uses local information for computing the distances and in certain domains can yield highly adaptive behaviors.
- It is robust to noisy training data when K is effectively chosen.
- Holding the entire training data for classification can be problematic depending on the number of data points and hardware constraints
- Number of features and the curse of dimensionality affects this algorithm more hence some form of dimensionality reduction or feature selection has to be done prior to modeling in KNN.

Support vector machines (SVM)

SVMs, in simple terms, can be viewed as linear classifiers that maximize the margin between the separating hyperplane and the data by solving a constrained optimization problem. SVMs can even deal with data that is not linearly separable by

invoking transformation to a higher dimensional space using kernels described later.

Algorithm inputs and outputs

SVM is effective with numeric features only, though most implementations can handle categorical features with transformation to numeric or binary. Normalization is often a choice as it helps the optimization part of the training. Outputs of SVM are class predictions. There are implementations that give probability estimates as confidences, but this requires considerable training time as they use k-fold cross-validation to build the estimates.

How does it work?

In its linear form, SVM works similar to Linear Regression classifier, where a linear decision boundary is drawn between the two classes. The difference between the two is that with SVM, the boundary is drawn in such a way that the "margin" or the distance between the points near the boundary is maximized. The points on the boundaries are known as "support vectors" (*References [13 and 8]*).

Thus, SVM tries to find the weight vector in linear models similar to Linear Regression model as given by the following:

$$G(\mathbf{x}) = b + \sum_{j=1}^d w_j x_j$$

The weight w_0 is represented by b here. SVM for a binary class $y \in \{1, -1\}$ tries to find a hyperplane:

$$b + \sum_{j=1}^d w_j x_j = 0$$

The hyperplane tries to separate the data points such that all points with the class lie on the side of the hyperplane as:

$$b + \sum_{j=1}^d w_j x_j \geq 0 \quad \forall i : y = 1$$

$$b + \sum_{j=1}^d w_j x_j \leq 0 \quad \forall i : y = -1$$

The models are subjected to maximize the margin using constraint-based optimization with a penalty function denoted by C for overcoming the errors denoted by ξ_i :

$$\min\left(\frac{1}{2}\|\mathbf{w}\|^2\right) + C \sum_i \xi_i$$

Such that $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall \mathbf{x}_i$ and $\xi_i \geq 0$.

They are also known as large margin classifiers for the preceding reason. The kernel-based SVM transforms the input data into a hypothetical feature space where SV machinery works in a linear way and the boundaries are drawn in the feature spaces.

A kernel function on the transformed representation is given by:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Here Φ is a transformation on the input space. It can be seen that the entire optimization and solution of SVM remains the same with the only exception that the dot-product $\mathbf{x}_i \cdot \mathbf{x}_j$ is replaced by the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, which is a function involving the two vectors in a different space without actually transforming to that space. This is known as the **kernel trick**.

The most well-known kernels that are normally used are:

- **Gaussian Radial Basis Kernel:**

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$$

- **Polynomial Kernel:**

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$$

- **Sigmoid Kernel:**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i \cdot \mathbf{x}_j - d)$$

SVM's performance is very sensitive to some of the parameters of optimization and the kernel parameters and the core SV parameter such as the cost function C . Search techniques such as grid search or evolutionary search combined with validation techniques such as cross-validation are generally used to find the best parameter values.

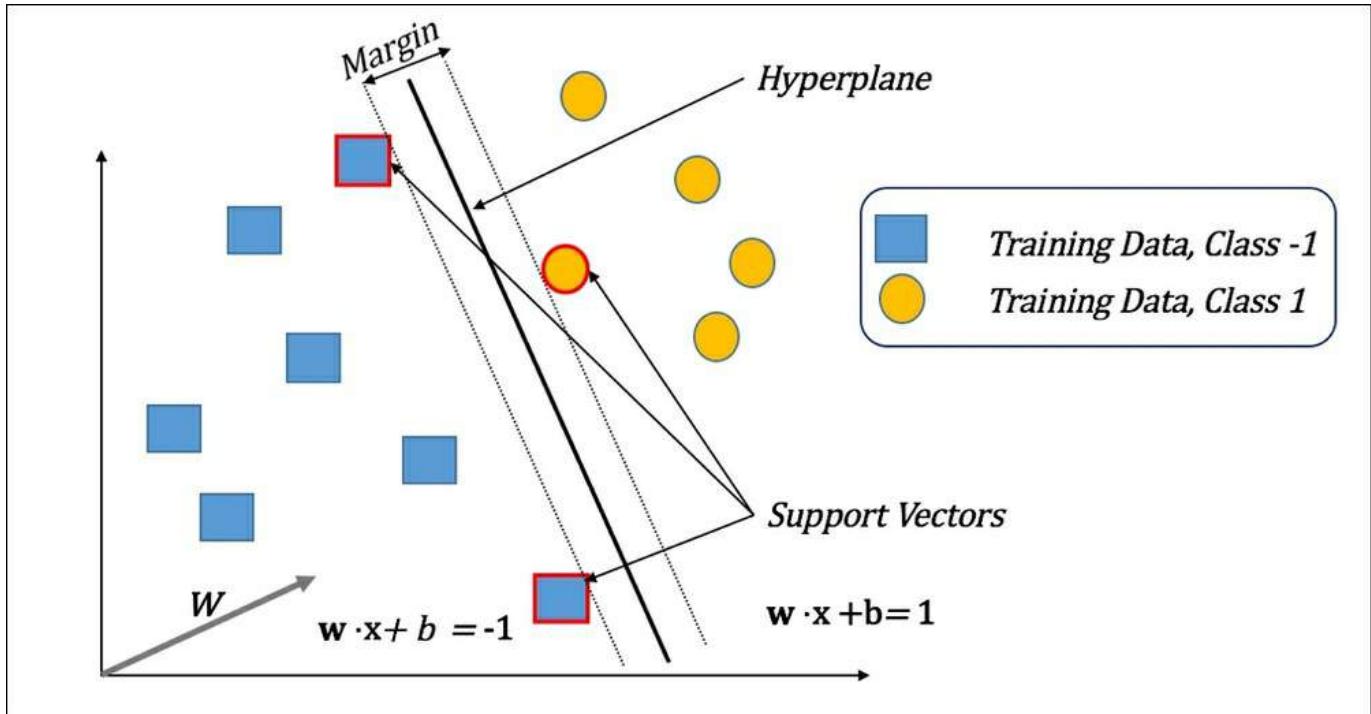


Figure 7: SVM Linear Hyperplane learned from training data that creates a maximum margin separation between two classes.

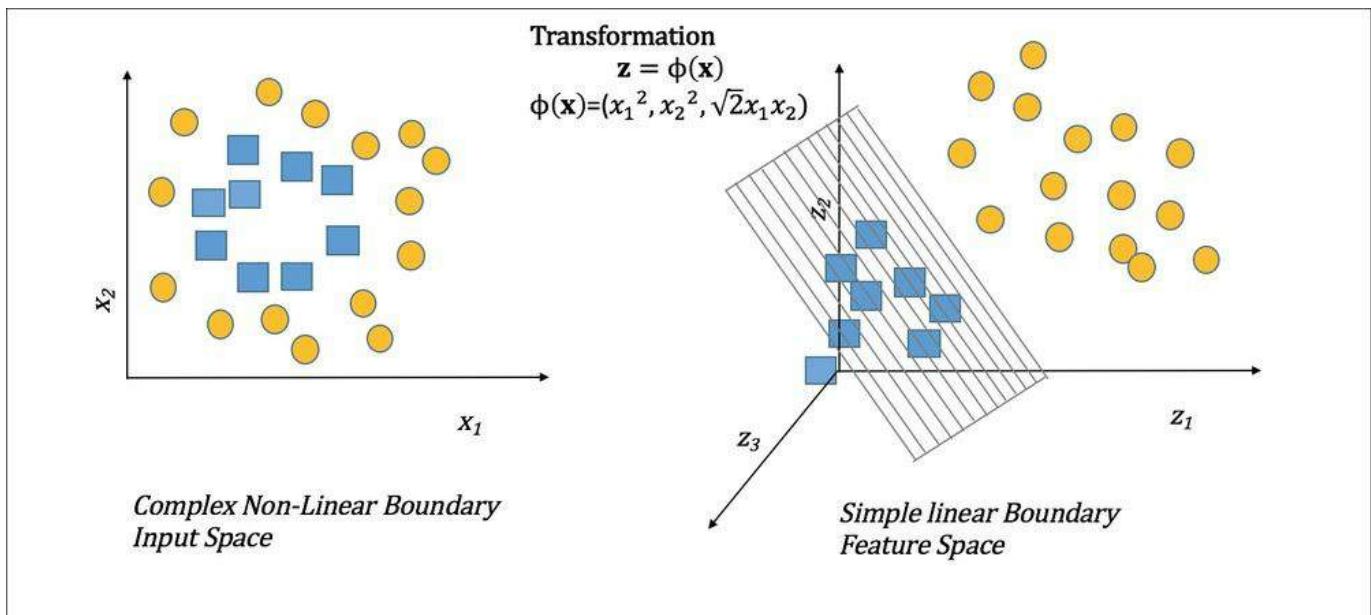


Figure 8: Kernel transformation illustrating how two-dimensional input space can be transformed using a polynomial transformation into a three-dimensional feature space where data is linearly separable.

Advantages and limitations

- SVMs are among the best in generalization, low overfitting, and have a good theoretical foundation for complex non-linear data if the parameters are chosen judiciously.
- SVMs work well even with a large number of features and less training data.
- SVMs are less sensitive to noisy training data.
- The biggest disadvantage of SVMs is that they are not interpretable.
- Another big issue with SVM is its training time and memory requirements. They are $O(n^2)$ and $O(n^3)$ and can result in major scalability issues when the data is large or there are hardware constraints. There are some modifications that help in reducing both.
- SVM generally works well for binary classification problems, but for multiclass classification problems, though there are techniques such as one versus many and one versus all, it is not as robust as some other classifiers such as Decision Trees.

Ensemble learning and meta learners

Combining multiple algorithms or models to classify instead of relying on just one is known as ensemble learning. It helps to combine various models as each model can be considered—at a high level—as an expert in detecting specific patterns in the whole dataset. Each base learner can be made to learn on slightly different datasets too. Finally, the results from all models are combined to perform prediction. Based on how similar the algorithms used in combination are, how the training dataset is presented to each algorithm, and how the algorithms combine the results to finally classify the unseen dataset, there are many branches of ensemble learning:

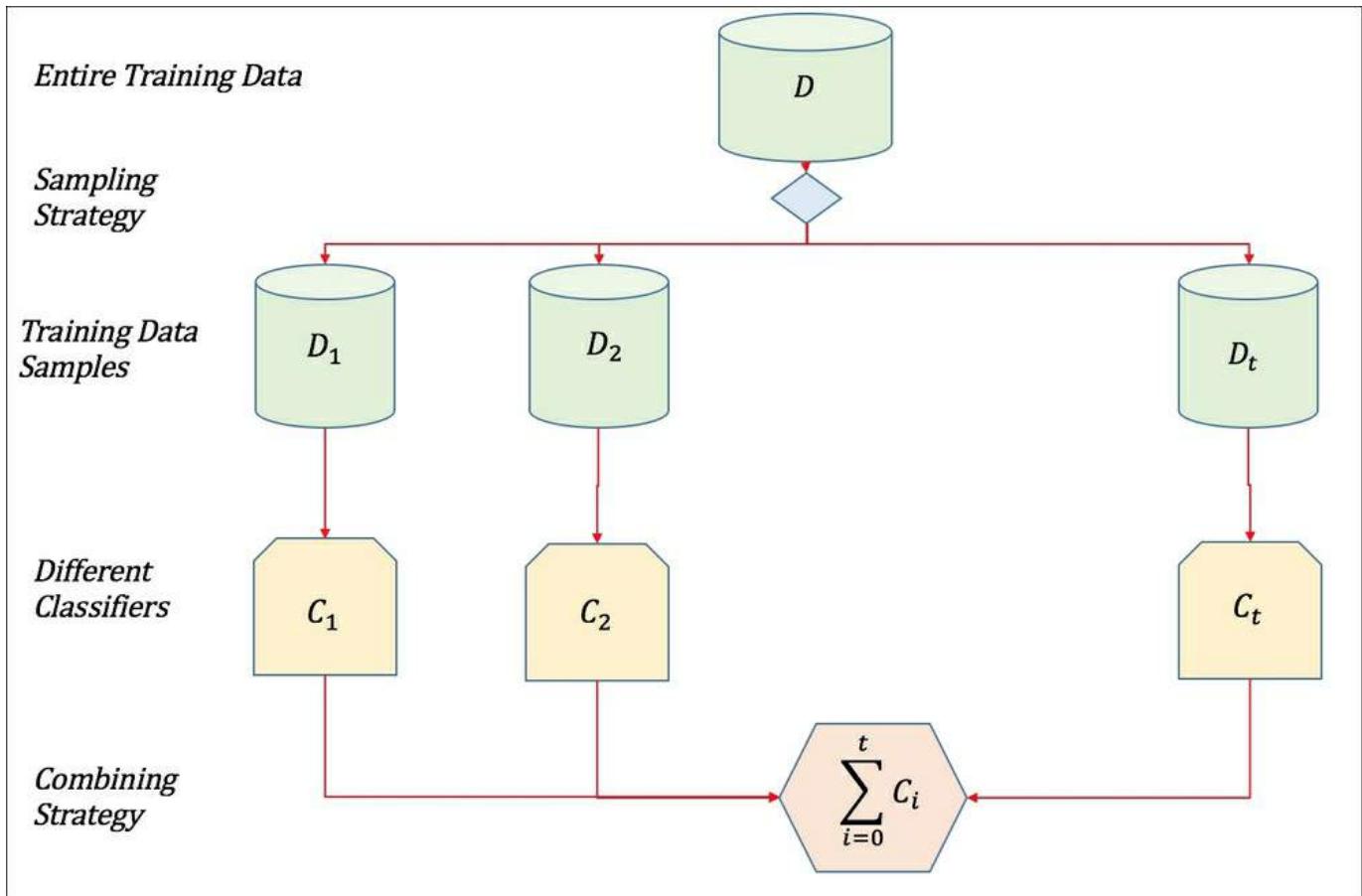


Figure 9: Illustration of ensemble learning strategies

Some common types of ensemble learning are:

- Different learning algorithms
- Same learning algorithms, but with different parameter choices

- Different learning algorithms on different feature sets
- Different learning algorithms with different training data

Bootstrap aggregating or bagging

It is one of the most commonly used ensemble methods for dividing the data in different samples and building classifiers on each sample.

Algorithm inputs and outputs

The input is constrained by the choice of the base learner used—if using Decision Trees there are basically no restrictions. The method outputs class membership along with the probability distribution for classes.

How does it work?

The core idea of bagging is to apply the bootstrapping estimation to different learners that have high variance, such as Decision Trees. Bootstrapping is any statistical measure that depends on random sampling with replacement. The entire data is split into different samples using bootstrapping and for each sample, a model is built using the base learner. Finally, while predicting, the average prediction is arrived at using a majority vote—this is one technique to combine over all the learners.

Random Forest

Random Forest is an improvement over basic bagged Decision Trees. Even with bagging, the basic Decision Tree has a choice of all the features at every split point in creating a tree. Because of this, even with different samples, many trees can form highly correlated submodels, which causes the performance of bagging to deteriorate. By giving random features to different models in addition to a random dataset, the correlation between the submodels reduces and Random Forest shows much better performance compared to basic bagged trees. Each tree in Random Forest grows its structure on the random features, thereby minimizing the bias; combining many such trees on decision reduces the variance (*References [15]*). Random Forest is also used to measure feature relevance by averaging the impurity decrease in the trees and ranking them across all the features to give the relative importance of each.

Advantages and limitations

- Better generalization than the single base learner. Overcomes the issue of

overfitting of base learners.

- Interpretability of bagging is very low as it works as meta learner combining even the interpretable learners.
- Like most other ensemble learners, Bagging is resilient to noise and outliers.
- Random Forest generally does not tend to overfit given the training data is iid.

Boosting

Boosting is another popular form of ensemble learning, which is based on using a weak learner and iteratively learning the points that are "misclassified" or difficult to learn. Thus, the idea is to "boost" the difficult to learn instances and making the base learners learn the decision boundaries more effectively. There are various flavors of boosting such as AdaBoost, LogitBoost, ConfidenceBoost, Gradient Boosting, and so on. We present a very basic form of AdaBoost here (*References* [14]).

Algorithm inputs and outputs

The input is constrained by the choice of the base learner used—if using Decision Trees there are basically no restrictions. Outputs class membership along with probability distribution for classes.

How does it work?

The basic idea behind boosting is iterative reweighting of input samples to create new distribution of the data for learning a model from a simple base learner in every iteration.

Initially, all the instances are uniformly weighted with weights $D_0 = \frac{1}{n}$ and at every iteration t , the population is resampled or reweighted as

$$D_{t+1} = \frac{D_t(i)}{z_t} \left(\exp(-\alpha_t h_t y_i(x_i)) \right)$$

where $\alpha_t = \frac{1}{2} \ln \frac{(1 - errD_t(h_t))}{errD_t(h_t)}$ and Z_t is the normalization constant.

The final model works as a linear combination of all the models learned in the iteration:

$$G(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T a_t h_t(\mathbf{x}) \right)$$

The reweighting or resampling of the data in each iteration is based on "errors"; the data points that result in errors are sampled more or have larger weights.

Advantages and limitations

- Better generalization than the base learner and overcomes the issue of overfitting very effectively.
- Some boosting algorithms such as AdaBoost can be susceptible to uniform noise. There are variants of boosting such as "GentleBoost" and "BrownBoost" that decrease the effect of outliers.
- Boosting has a theoretical bounds and guarantee on the error estimation making it a statistically robust algorithm.

Model assessment, evaluation, and comparisons

The key ideas discussed here are:

- How to assess or estimate the performance of the classifier on unseen datasets that it will be predicting on future unseen datasets.
- What are the metrics that we should use to assess the performance of the model?
- How do we compare algorithms if we have to choose between them?

Model assessment

In order to train the model(s), tune the model parameters, select the models, and finally estimate the predictive behavior of models on unseen data, we need many datasets. We cannot train the model on one set of data and estimate its behavior on the same set of data, as it will have a clear optimistic bias and estimations will be unlikely to match the behavior in the unseen data. So at a minimum, there is a need to partition data available into training sets and testing sets. Also, we need to tune the parameters of the model and test the effect of the tuning on a separate dataset before we perform testing on the test set. The same argument of optimistic bias and wrong estimation applies if we use the same dataset for training, parameter tuning, and testing. Thus there is a theoretical and practical need to have three datasets, that is, training, validation, and testing.

The models are trained on the training set, the effect of different parameters on the training set are validated on the validation set, and the finalized model with the selected parameters is run on the test set to gauge the performance of the model on future unseen data. When the dataset is not large enough, or is large but the imbalance between classes is wide, that is, one class is present only in a small fraction of the total population, we cannot create too many samples. Recall that one of the steps described in our methodology is to create different data samples and datasets. If the total training data is large and has a good proportion of data and class ratios, then creating these three sets using random stratified partitioning is the most common option employed. In certain datasets that show seasonality and time-dependent behaviors, creating datasets based on time bounds is a common practice. In many cases, when the dataset is not large enough, only two physical partitions, that is, training and testing may be created. The training dataset ranges roughly from 66% to 80% while the rest is used for testing. The validation set is then created from the training dataset using the k-fold cross-validation technique. The training dataset is split k times, each time producing $k-1/k$ random training $1/k$ testing data samples, and the average metrics of performance needed is generated. This way the limited training data is partitioned k times and average performance across different split of training/testing is used for gauging the effect of the parameters. Using 10-fold cross-validation is the most common practice employed in cross-validation.

Model evaluation metrics

The next important decision when tuning parameters or selecting models is to base your decision on certain performance metrics. In classification learning, there are different metrics available on which you can base your decision, depending on the business requirement. For example, in certain domains, not missing a single true positive is the most important concern, while in other domains where humans are involved in adjudicating results of models, having too many false positives is the greater concern. In certain cases, having overall good accuracy is considered more vital. In highly imbalanced datasets such as fraud or cyber attacks, there are just a handful of instances of one class and millions of the other classes. In such cases accuracy gives a wrong indication of model performance and some other metrics such as precision, true positive ratio, or area under the curve are used as metrics.

We will now discuss the most commonly employed metrics in classification algorithms evaluation (*References [16, 17, and 19]*).

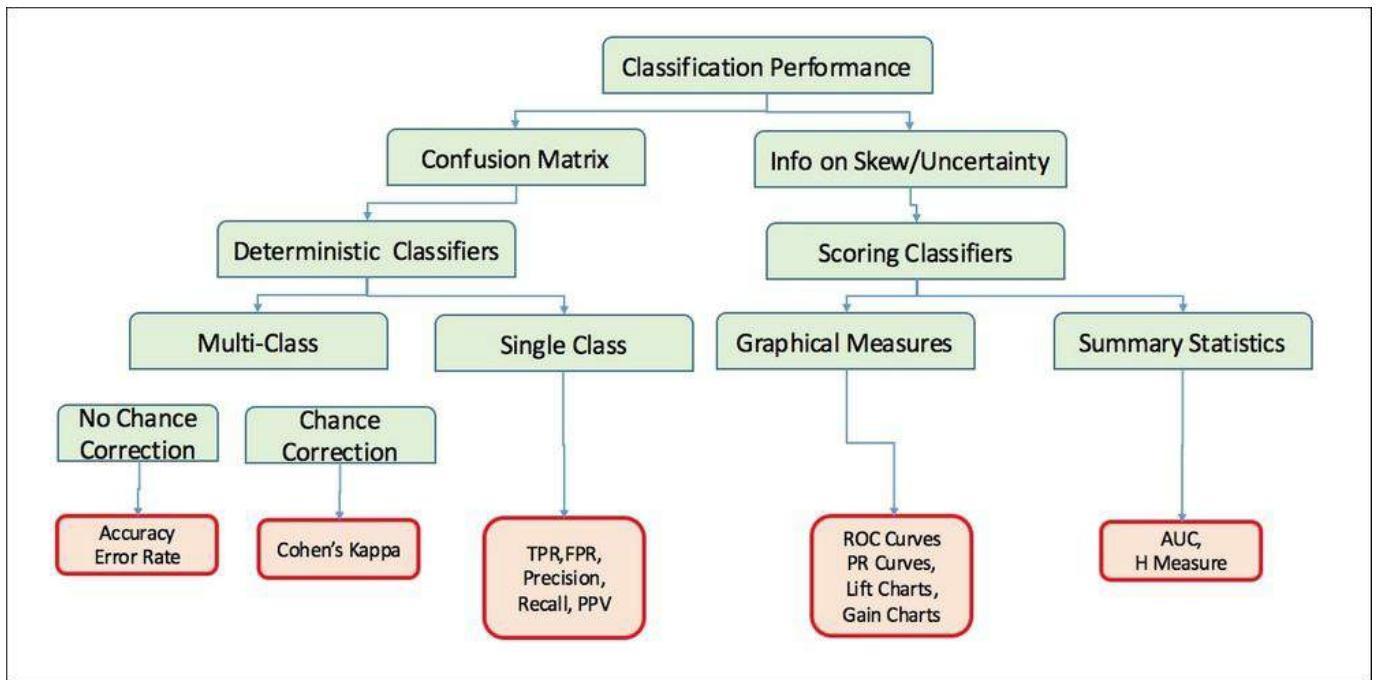


Figure 10: Model evaluation metrics for classification models

Confusion matrix and related metrics

		<i>Predicted</i>			
		<i>Class Positive</i>	<i>Class Negative</i>		
<i>Actual</i>	<i>Class Positive</i>	<i>True Positive</i>	<i>False Negative</i> <i>Type II Error</i>	<i>TPR, Sensitivity, Recall</i> $\frac{\sum \text{True Positive}}{\sum \text{Actual Class Positive}}$	<i>FNR</i> $\frac{\sum \text{False Negative}}{\sum \text{Actual Class Positive}}$
	<i>Class Negative</i>	<i>False Positive</i> <i>Type I Error</i>	<i>True Negative</i>	<i>FPR, Fall out</i> $\frac{\sum \text{False Positive}}{\sum \text{Actual Class Negative}}$	<i>TNR, Specificity</i> $\frac{\sum \text{True Negative}}{\sum \text{Actual Class Negative}}$
	<i>PPV, Precision</i> $\frac{\sum \text{True Positive}}{\sum \text{Predicted Class Positive}}$	<i>False Omission Rate</i> $\frac{\sum \text{False Negative}}{\sum \text{Predicted Class Negative}}$	<i>ACCURACY</i>		
	<i>False Discovery Rate</i> $\frac{\sum \text{False Positive}}{\sum \text{Predicted Class Positive}}$	<i>NPV</i> $\frac{\sum \text{True Negative}}{\sum \text{Predicted Class Negative}}$	$\frac{\sum \text{True Positive} + \sum \text{True Negative}}{\sum \text{Total Data}}$		

Figure 11: Confusion Matrix

The confusion matrix is central to the definition of a number of model performance metrics. The proliferation of metrics and synonymous terms is a result of the utility of different quantities derived from the elements of the matrix in various disciplines, each emphasizing a different aspect of the model's behavior.

The four elements of the matrix are raw counts of the number of False Positives, False Negatives, True Positives, and True Negatives. Often more interesting are the different ratios of these quantities, the True Positive Rate (or Sensitivity, or Recall), and the False Positive Rate (FPR, or 1—Specificity, or Fallout). Accuracy reflects the percentage of correct predictions, whether Class 1 or Class 0. For skewed datasets, accuracy is not particularly useful, as even a constant prediction can appear to perform well.

ROC and PRC curves

The previously mentioned metrics such as accuracy, precision, recall, sensitivity, and specificity are aggregates, that is, they describe the behavior of the entire dataset. In

many complex problems it is often valuable to see the trade-off between metrics such as TPs and say FPs.

Many classifiers, mostly probability-based classifiers, give confidence or probability of the prediction, in addition to giving classification. The process to obtain the ROC or PRC curves is to run the unseen validation or test set on the learned models, and then obtain the prediction and the probability of prediction. Sort the predictions based on the confidences in decreasing order. For every probability or confidence calculate two metrics, the fraction of FP (FP rate) and the fraction of TP (TP rate).

Plotting the TP rate on the y axis and FP rate on the x axis gives the ROC curves. ROC curves of random classifiers lie close to the diagonal while the ROC curves of good classifiers tend towards the upper left of the plot. The **area under the curve (AUC)** is the area measured under the ROC curve by using the trapezoidal area from 0 to 1 of ROC curves. While running cross-validation for instance there can be many ROC curves. There are two ways to get "average" ROC curves: first, using vertical averaging, that is, TPR average is plotted at different FP rate or second, using horizontal averaging, that is, FPR average is plotted at different TP rate. The classifiers that have area under curves greater than 0.8, as a rule-of-thumb are considered good for prediction for unseen data.

Precision Recall curves or PRC curves are similar to ROC curves, but instead of TPR versus FPR, metrics Precision and Recall are plotted on the y and x axis, respectively. When the data is highly imbalanced, that is, ROC curves don't really show the impact while PRC curves are more reliable in judging performance.

Gain charts and lift curves

Lift and Gain charts are more biased towards sensitivity or true positives. The whole purpose of these two charts is to show how instead of random selection, the models prediction and confidence can detect better quality or true positives in the sample of unseen data.

This is usually very appealing for detection engines that are used in detecting fraud in financial crime or threats in cyber security. The gain charts and lift curves give exact estimates of real true positives that will be detected at different quartiles or intervals of total data. This will give insight to the business decision makers on how many investigators would be needed or how many hours would be spent towards detecting fraudulent actions or cyber attacks and thus can give real ROI of the

models.

The process for generating gain charts or lift curves has a similar process of running unseen validation or test data through the models and getting the predictions along with the confidences or probabilities. It involves ranking the probabilities in decreasing order and keeping count of TPs per quartile of the dataset. Finally, the histogram of counts per quartile give the lift curve, while the cumulative count of TPs added over quartile gives the gains chart. In many tools such as RapidMiner, instead of coarse intervals such as quartiles, fixed larger intervals using binning is employed for obtaining the counts and cumulative counts.

Model comparisons

When it comes to choosing between algorithms, or the right parameters for a given algorithm, we make the comparison either on different datasets, or, as in the case of cross-validation, on different splits of the same dataset. Measures of statistical testing are employed in decisions involved in these comparisons. The basic idea of using hypothesis testing from classical statistics is to compare the two metrics from the algorithms. The null hypothesis is that there is no difference between the algorithms based on the measured metrics and so the test is done to validate or reject the null hypothesis based on the measured metrics (*References [16]*). The main question answered by statistical tests is- are the results or metrics obtained by the algorithm its real characteristics, or is it by chance?

In this section, we will discuss the most common methods for comparing classification algorithms used in practical scenarios.

Comparing two algorithms

The general process is to train the algorithms on the same training set and run the models on either multiple validation sets, different test sets, or cross-validation, gauge the metrics of interest discussed previously, such as error rate or area under curve, and then get the statistics of the metrics for each of the algorithms to decide which worked better. Each method has its advantages and disadvantages.

McNemar's Test

This is a non-parametric test and thus it makes no assumptions on data and distribution. McNemar's test builds a contingency table of a performance metric such as "misclassification or errors" with:

Count of misclassification by both algorithms (c_{00})

- Count of misclassification by algorithm $G1$, but correctly classified by algorithm $G2(c_{01})$
- Count of misclassification by algorithm $G2$, but correctly classified by algorithm $G1 (c_{10})$
- Count of correctly classified by both $G1$ and $G2(c_{11})$

$$\chi^2 = \frac{\left((c_{01} - c_{10}) - 1 \right)^2}{c_{01} + c_{10}}$$

If χ^2 exceeds $\chi^2_{1,1-\alpha}$ statistic then we can reject the null hypothesis that the two performance metrics on algorithms $G1$ and $G2$ were equal under the confidence value of $1 - \alpha$.

Paired-t test

This is a parametric test and an assumption of normally distributed computed metrics becomes valid. Normally it is coupled with cross-validation processes and results of metrics such as area under curve or precision or error rate is computed for each and then the mean and standard deviations are measured. Apart from normal distribution assumption, the additional assumption that two metrics come from a population of equal variance can be a big disadvantage for this method.

$$\bar{d} = \overline{pm(G1)} - \overline{pm(G2)}$$

\bar{d} is difference of means in performance metrics of two algorithms $G1$ and $G2$.

$$\sigma_d = \sqrt{\frac{\sum_{i=1}^n (d_i - \bar{d})^2}{n-1}}$$

Here, d_i is the difference between the performance metrics of two algorithms $G1$ and $G2$ in the trial and there are n trials.

The t -statistic is computed using the mean differences and the standard errors from

the standard deviation as follows and is compared to the table for the right alpha value to check for significance:

$$t = \frac{\bar{d}}{\sigma_d / \sqrt{n}}$$

Wilcoxon signed-rank test

The most popular non-parametric method of testing two metrics over datasets is to use the Wilcoxon signed-rank test. The algorithms are trained on the same training data and metrics such as error rate or area under accuracy are calculated over different validation or test sets. Let d_i be the difference between the performance metrics of two classifiers in the i^{th} trial for N datasets. Differences are then ranked according to their absolute values, and mean ranks associated for ties. Let R^+ be the sum of ranks where the second algorithm outperformed the first and R^- be the sum of ranks where the first outperformed the second:

$$R^+ = \sum_{d_i > 0} rank(d_i) + \frac{1}{2} \sum_{d_i = 0} rank(d_i)$$

$$R^- = \sum_{d_i < 0} rank(d_i) + \frac{1}{2} \sum_{d_i = 0} rank(d_i)$$

The statistic $T_{\text{wilcoxon}} = \min(R^+, R^-)$ is then compared to threshold value at an alpha, $T_{\text{wilcoxon}} \leq V_\alpha$ to reject the hypothesis.

Comparing multiple algorithms

We will now discuss the two most common techniques used when there are more than two algorithms involved and we need to perform comparison across many algorithms for evaluation metrics.

ANOVA test

These are parametric tests that assume normal distribution of the samples, that is, metrics we are calculating for evaluations. ANOVA test follows the same process as others, that is, train the models/algorithms on similar training sets and run it on different validation or test sets. The main quantities computed in ANOVA are the metric means for each algorithm performance and then compute the overall metric means across all algorithms.

Let p_{ij} be the performance metric for $i = 1, 2, \dots, k$ and $j = 1, 2, \dots, l$ for k trials and l classifiers. The mean performance of classifier j on all trials and overall mean performance is:

$$m_j = \frac{\sum_{i=1}^k p_{ij}}{k}$$

$$m = \frac{\sum_{j=1}^l m_j}{l}$$

Two types of variation are evaluated. The first is within-group variation, that is, total deviation of each algorithm from the overall metric mean, and the second is between-group variation, that is, deviation of each algorithm metric mean. Within-group variation and between-group variation are used to compute the respective within- and between- sum of squares as:

$$SS_b = k \sum_j (m_j - m)^2, SS_w = \sum_j \sum_i (X_{ij} - m_j)^2$$

Using the two sum of squares and a computation such as F-statistic, which is the ratio of the two, the significance test can be done at alpha values to accept or reject the null hypothesis:

$$f = \frac{\frac{SS_b}{(l-1)}}{\frac{SS_w}{k(l-1)}}$$

ANOVA tests have the same limitations as paired-t tests on the lines of assumptions of normal distribution of metrics and assuming the variances being equal.

Friedman's test

Friedman's test is a non-parametric test for multiple algorithm comparisons and it has no assumption on the data distribution or variances of metrics that ANOVA does. It uses ranks instead of the performance metrics directly for its computation. On each dataset or trials, the algorithms are sorted and the best one is ranked 1 and so on for all classifiers. The average rank of an algorithm over n datasets is computed, say R_j . The Friedman's statistic over l classifiers is computed as follows and compared to alpha values to accept or reject the null hypothesis:

$$\chi^2 = \left[\frac{12}{n \times l \times (l+1)} \times \sum_{j=1}^l R_j^2 \right] - 3 \times n \times (l+1)$$

Case Study – Horse Colic Classification

To illustrate the different steps and methodologies described in [Chapter 1](#), *Machine Learning Review*, from data analysis to model evaluation, a representative dataset that has real-world characteristics is essential.

We have chosen "Horse Colic Dataset" from the UCI Repository available at the following link: <https://archive.ics.uci.edu/ml/datasets/Horse+Colic>

The dataset has 23 features and has a good mix of categorical and continuous features. It has a large number of features and instances with missing values, hence understanding how to replace these missing values and using it in modeling is made more practical in this treatment. The large number of missing data (30%) is in fact a notable feature of this dataset. The data consists of attributes that are continuous, as well as nominal in type. Also, the presence of self-predictors makes working with this dataset instructive from a practical standpoint.

The goal of the exercise is to apply the techniques of supervised learning that we have assimilated so far. We will do this using a real dataset and by working with two open source toolkits—WEKA and RapidMiner. With the help of these tools, we will construct the pipeline that will allow us to start with the ingestion of the data file through data cleansing, the learning process, and model evaluation.

Weka is a Java framework for machine learning—we will see how to use this framework to solve a classification problem from beginning to end in a few lines of code. In addition to a Java API, Weka also has a GUI.

RapidMiner is a graphical environment with drag and drop capability and a large suite of algorithms and visualization tools that makes it extremely easy to quickly run experiments with data and different modeling techniques.

Business problem

The business problem is to determine given values for the well-known variables of the dataset—if the lesion of the horse was surgical. We will use the test set as the unseen data that must be classified.

Machine learning mapping

Based on the data and labels, this is a binary classification problem. The data is already split into training and testing data. This makes the evaluation technique simpler as all methodologies from feature selection to models can be evaluated on the same test data.

The dataset contains 300 training and 68 test examples. There are 28 attributes and the target corresponds to whether or not a lesion is surgical.

Data analysis

After looking at the distribution of the label categories over the training and test samples, we combine the 300 training samples and the 68 test samples prior to feature analyzes.

Label analysis

The ratio of the No Class to Yes Class is $109/191 = 0.57$ in the Training set and 0.66 in the Test set:

Training dataset		
Surgical Lesion?	1 (Yes)	2 (No)
Number of examples	191	109
Testing dataset		
Surgical Lesion?	1 (Yes)	2 (No)
Number of examples	41	27

Table 2: Label analysis

Features analysis

The following is a screenshot of top features with characteristics of types, missing values, basic statistics of minimum, maximum, modes, and standard deviations sorted by missing values. Observations are as follows:

- There are no categorical or continuous features with non-missing values; the least is the feature "pulse" with 74 out of 368 missing, that is, 20% values missing, which is higher than general noise threshold!
- Most numeric features have missing values, for example, "nasogastric reflux PH" has 247 out of 368 values missing, that is, 67% values are missing!
- Many categorical features have missing values, for example, "abdominocentesis appearance" have 165 out of 368 missing, that is, 45% values are missing!

- Missing values have to be handled in some way to overcome the noise created by such large numbers!

✓ 'nasogastric reflux PH'	Numeric	247	Min 1	Max 7.500	Average 4.708
✓ 'abdomcentesis total protein'	Numeric	198	Min 0.100	Max 10.100	Average 3.020
✓ 'abdominocentesis appearance'	Nominal	165	Least 1 (41)	Most 2 (48)	Values 2 (48), 3 (46), ...[1 more]
✓ abdomen	Nominal	118	Least 3 (13)	Most 5 (79)	Values 5 (79), 4 (43), ...[3 more]
✓ 'nasogastric reflux'	Nominal	106	Least 2 (35)	Most 1 (120)	Values 1 (120), 3 (39), ...[1 more]
✓ 'nasogastric tube'	Nominal	104	Least 3 (23)	Most 2 (102)	Values 2 (102), 1 (71), ...[1 more]
✓ 'rectal examination'	Nominal	102	Least 2 (13)	Most 4 (79)	Values 4 (79), 1 (57), ...[2 more]
✓ 'peripheral pulse'	Nominal	69	Least 2 (5)	Most 1 (115)	Values 1 (115), 3 (103), ...[2 more]
✓ 'rectal temperature'	Numeric	60	Min 35.400	Max 40.800	Average 38.168
✓ 'respiratory rate'	Numeric	58	Min 8	Max 96	Average 30.417
✓ 'temperature of extremities'	Nominal	56	Least 4 (27)	Most 3 (109)	Values 3 (109), 1 (78), ...[2 more]
✓ 'abdominal distension'	Nominal	56	Least 4 (38)	Most 1 (76)	Values 1 (76), 2 (65), ...[2 more]
✓ pain	Nominal	55	Least 1 (38)	Most 3 (67)	Values 3 (67), 2 (59), ...[3 more]
✓ 'mucous membranes'	Nominal	47	Least 6 (20)	Most 1 (79)	Values 1 (79), 3 (58), ...[4 more]
✓ peristalsis	Nominal	44	Least 2 (16)	Most 3 (128)	Values 3 (128), 4 (73), ...[2 more]
✓ 'total protein'	Numeric	33	Min 3.300	Max 89	Average 24.457
✓ 'capillary refill time'	Nominal	32	Least 3 (2)	Most 1 (188)	Values 1 (188), 2 (78), ...[1 more]
✓ 'packed cell volume'	Numeric	29	Min 23	Max 75	Average 46.295
✓ pulse	Numeric	24	Min 30	Max 184	Average 71.913

Figure 12: Basic statistics of features from datasets.

Supervised learning experiments

In this section, we will cover supervised learning experiments using two different tools—highlighting coding and analysis in one tool and the GUI framework in the other. This gives the developers the opportunity to explore whichever route they are most comfortable with.

Weka experiments

In this section, we have given the entire code and will walk through the process from loading data, transforming the data, selecting features, building sample models, evaluating them on test data, and even comparing the algorithms for statistical significance.

Sample end-to-end process in Java

In each algorithm, the same training/testing data is used and evaluation is performed for all the metrics as follows. The training and testing file is loaded in memory as follows:

```
DataSource source = new DataSource(trainingFile);
Instances data = source.getDataSet();
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);
```

The generic code, using WEKA, is shown here, where each classifier is wrapped by a filtered classifier for replacing missing values:

```
//replacing the nominal and numeric with modes and means
Filter missingValuesFilter= new ReplaceMissingValues();
//create a filtered classifier to use filter and classifier
FilteredClassifier filteredClassifier = new FilteredClassifier();
filteredClassifier.setFilter(f);
// create a bayesian classifier
NaiveBayes naiveBayes = new NaiveBayes();
// use supervised discretization
naiveBayes.setUseSupervisedDiscretization(true);
//set the base classifier e.g naïvebayes, linear //regression etc.
fc.setClassifier(filteredClassifier)
```

When the classifier needs to perform Feature Selection, in Weka, AttributeSelectedClassifier further wraps the FilteredClassifier as shown in the following listing:

```

AttributeSelectedClassifier attributeSelectionClassifier = new
AttributeSelectedClassifier();
//wrap the classifier
attributeSelectionClassifier.setClassifier(filteredClassifier);
//univariate information gain based feature evaluation
    InfoGainAttributeEval evaluator = new InfoGainAttributeEval();
//rank the features
Ranker ranker = new Ranker();
//set the threshold to be 0, less than that is rejected
ranker.setThreshold(0.0);
attributeSelectionClassifier.setEvaluator(evaluator);
attributeSelectionClassifier.setSearch(ranker);
//build on training data
attributeSelectionClassifier.buildClassifier(trainingData);
// evaluate classifier giving same training data
Evaluation eval = new Evaluation(trainingData);
//evaluate the model on test data
eval.evaluateModel(attributeSelectionClassifier, testingData);

```

The sample output of evaluation is given here:

==== Summary ====

Correctly Classified Instances	53	77.9412 %
Incorrectly Classified Instances	15	22.0588 %
Kappa statistic	0.5115	
Mean absolute error	0.3422	
Root mean squared error	0.413	
Relative absolute error	72.4875 %	
Root relative squared error	84.2167 %	
Total Number of Instances	68	

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC	
Area Class									
	0.927	0.444	0.760	0.927	0.835	0.535	0.823	0.875	1
	0.556	0.073	0.833	0.556	0.667	0.535	0.823	0.714	2
Weighted Avg.	0.779	0.297	0.789	0.779	0.768	0.535	0.823	0.812	

==== Confusion Matrix ====

```

a b <-- classified as
38 3 | a = 1
12 15 | b = 2

```

Weka experimenter and model selection

As explained in the *Model evaluation metrics* section, to select models, we need to validate which one will work well on unseen datasets. Cross-validation must be done on the training set and the performance metric of choice needs to be analyzed using standard statistical testing metrics. Here we show an example using the same training data, 10-fold cross validation, performing 30 experiments on two models, and comparison of results using paired-t tests.

One uses Naïve Bayes with preprocessing that includes replacing missing values and performing feature selection by removing any features with a score below 0.0.

Another uses the same preprocessing and AdaBoostM1 with Naïve Bayes.

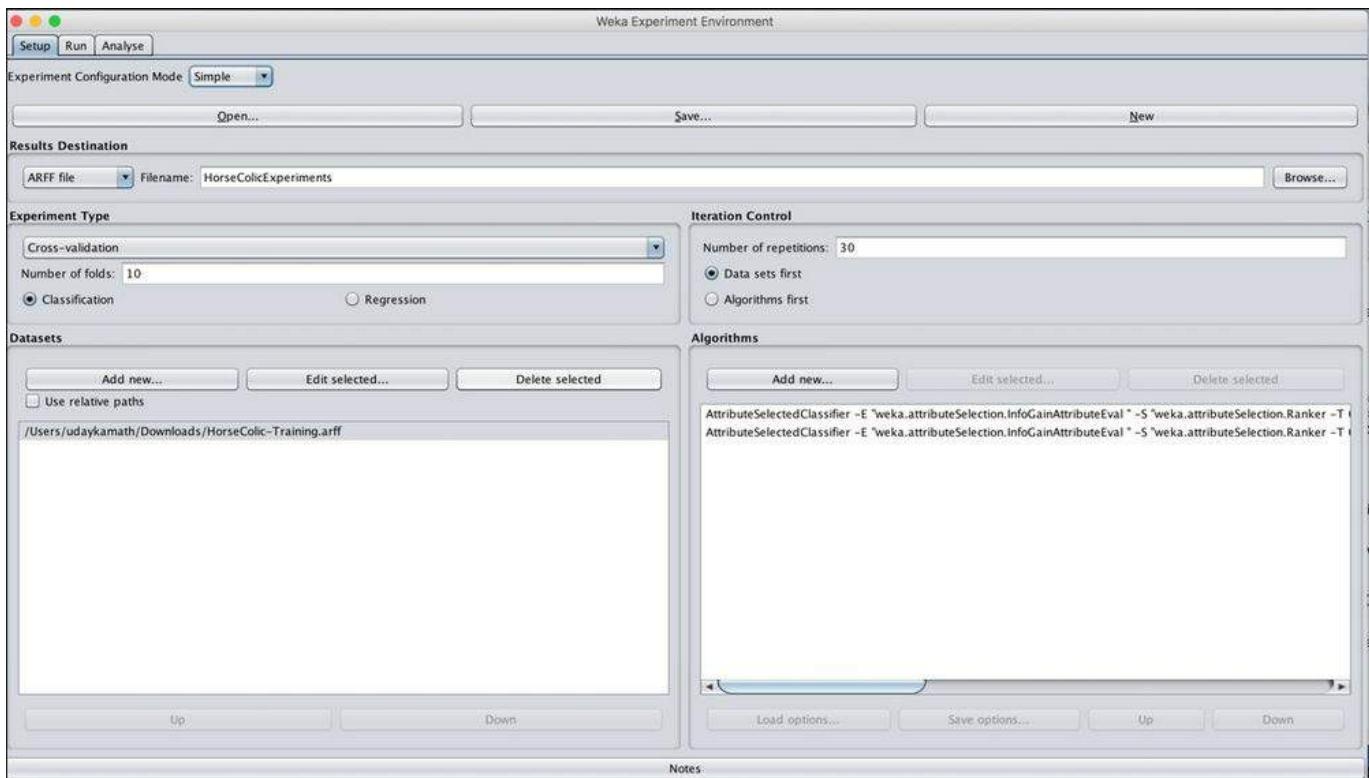


Figure 13: WEKA experimenter showing the process of using cross-validation runs with 30 repetitions with two algorithms.

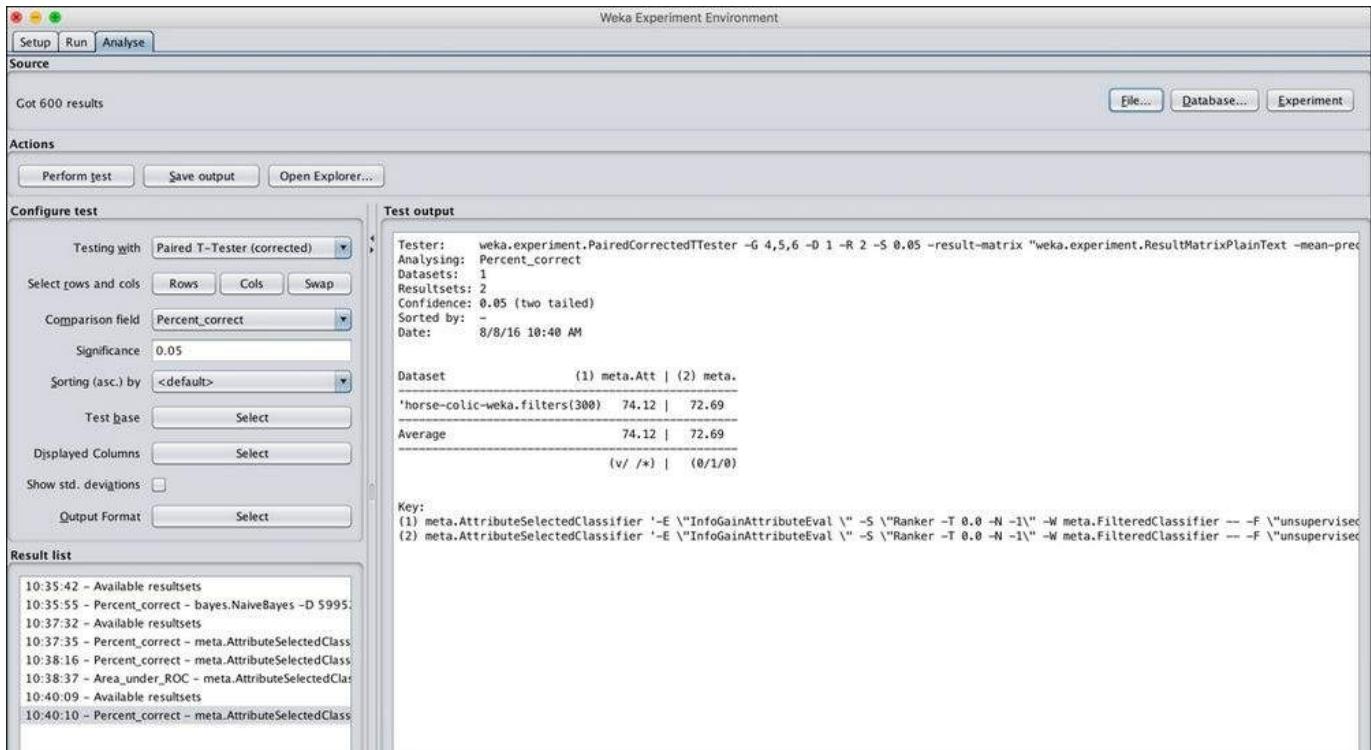


Figure 14: WEKA Experimenter results showing two algorithms compared on metric of percent correct or accuracy using paired-t test.

RapidMiner experiments

Let's now run some experiments using the Horse-colic dataset in RapidMiner. We will again follow the methodology presented in the first part of the chapter.

Note

This section is not intended as a tutorial on the RapidMiner tool. The experimenter is expected to read the excellent documentation and user guide to familiarize themselves with the use of the tool. There is a tutorial dedicated to every operator in the software—we recommend you make use of these tutorials whenever you want to learn how a particular operator is to be used.

Once we have imported the test and training data files using the data access tools, we will want to visually explore the dataset to familiarize ourselves with the lay of the land. Of particular importance is to recognize whether each of the 28 attributes are continuous (numeric, integer, or real in RapidMiner) or categorical (nominal,

binomial, or polynominal in RapidMiner).

Visualization analysis

From the **Results** panel of the tool, we perform univariate, bivariate, and multivariate analyses of the data. The Statistics tool gives a short summary for each feature—min, max, mean, and standard deviation for continuous types and least, most, and frequency by category for nominal types.

Interesting characteristics of the data begin to show themselves as we get into bivariate analysis. In the Quartile Color Matrix, the color represents the two possible target values. As seen in the box plots, we immediately notice some attributes discriminate between the two target values more clearly than others. Let's examine a few:

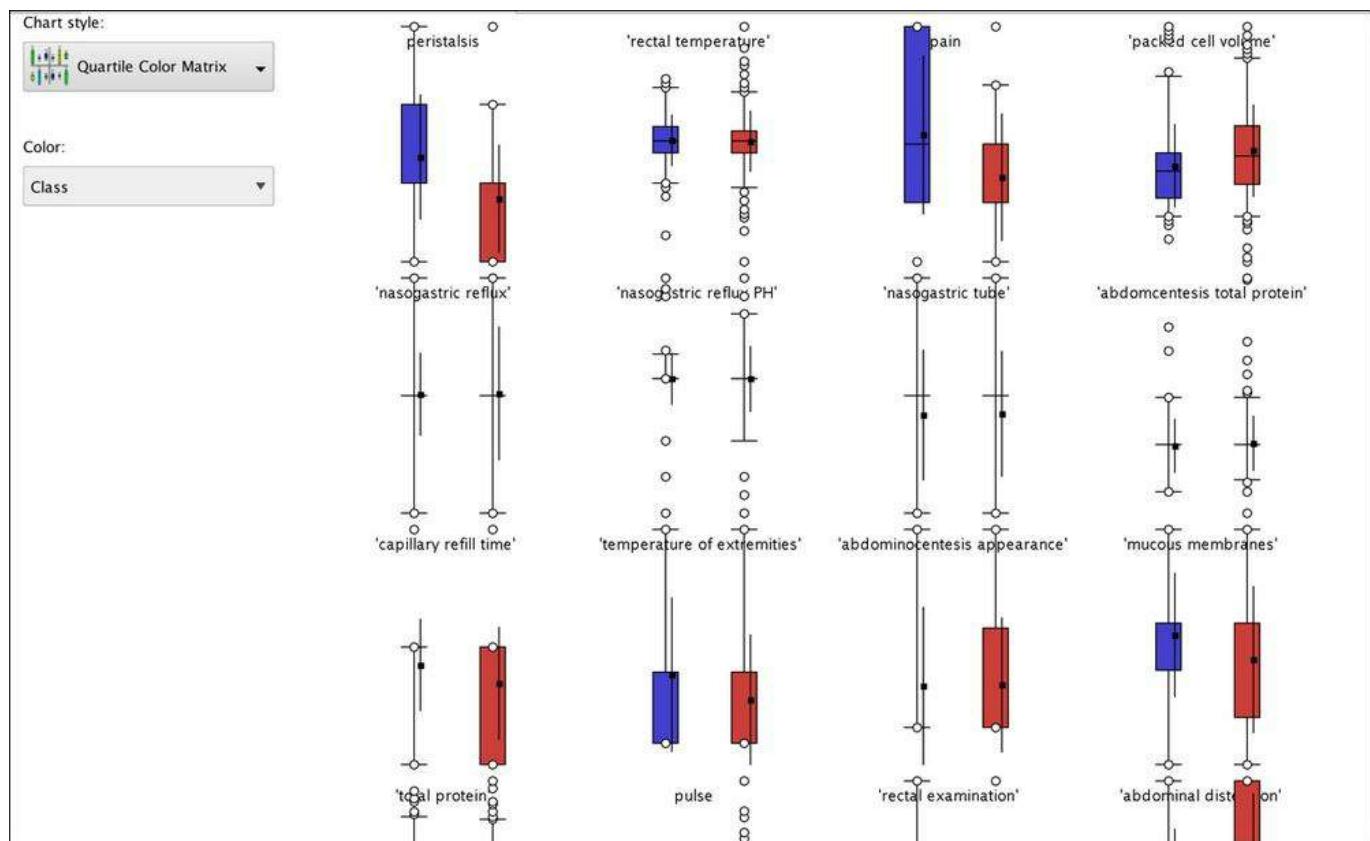


Figure 15: Quartile Color Matrix

Peristalsis: This feature shows a marked difference in distribution when separated by target value. There is almost no overlap in the inter-quartile regions between the two.

This points to the discriminating power of this feature with respect to the target.

The plot for Rectal Temperature, on the other hand, shows no perceptible difference in the distributions. This suggests that this feature has low correlation with the target. A similar inference may be drawn from the feature Pulse. We expect these features to rank fairly low when we evaluate the features for their discriminating power with respect to the target.

Lastly, the plot for Pain has a very different characteristic. It is also discriminating of the target, but in a very different way than Peristalsis. In the case of Pain, the variance in data for Class 2 is much larger than Class 1. Abdominal Distension also has markedly dissimilar variance across the classes, except with the larger variance in Class 2 compared to Class 1.

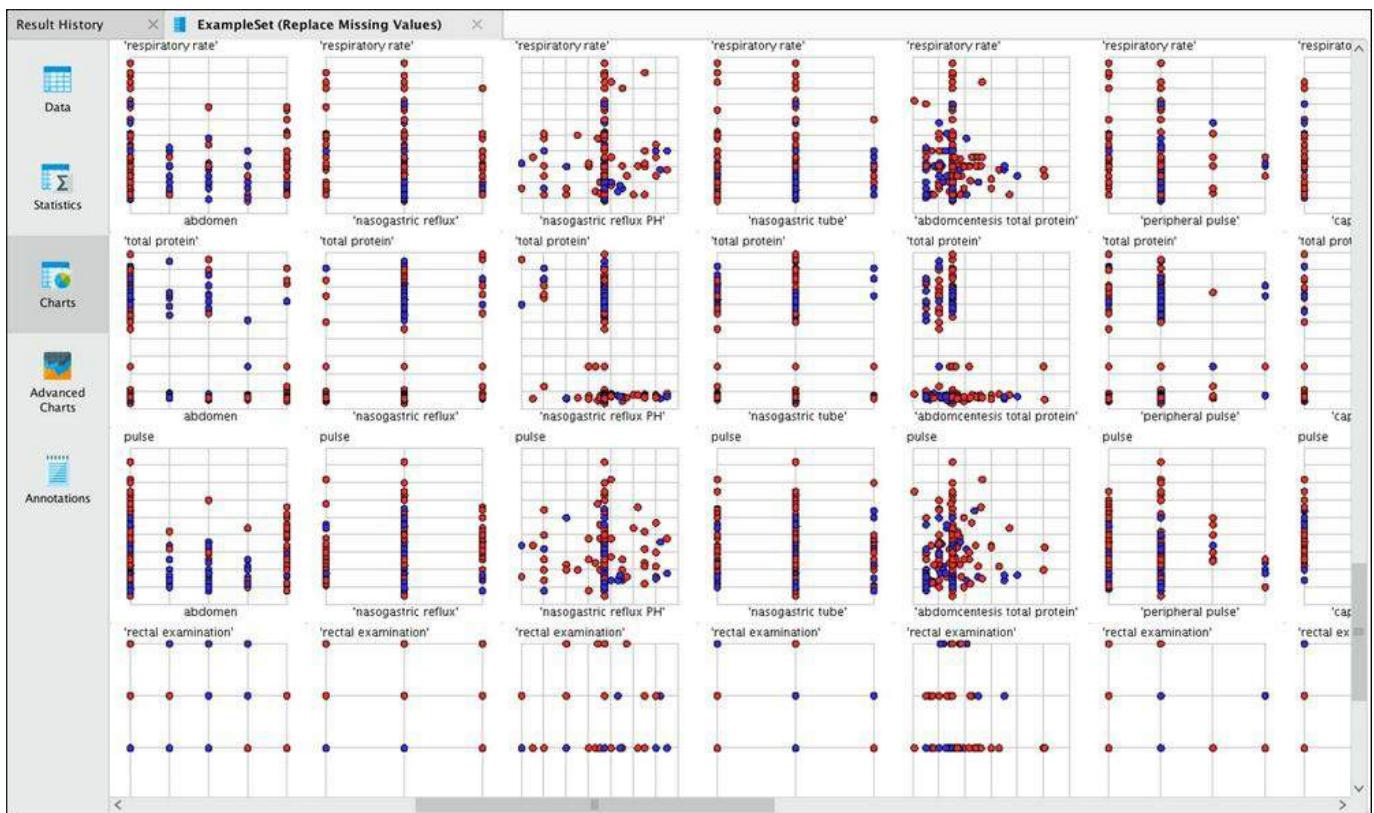


Figure 16: Scatter plot matrix

An important part of exploring the data is understanding how different attributes correlate with each other and with the target. Here we consider pairs of features and see if the occurrence of values *in combination* tells us something about the target. In

these plots, the color of the data points is the target.

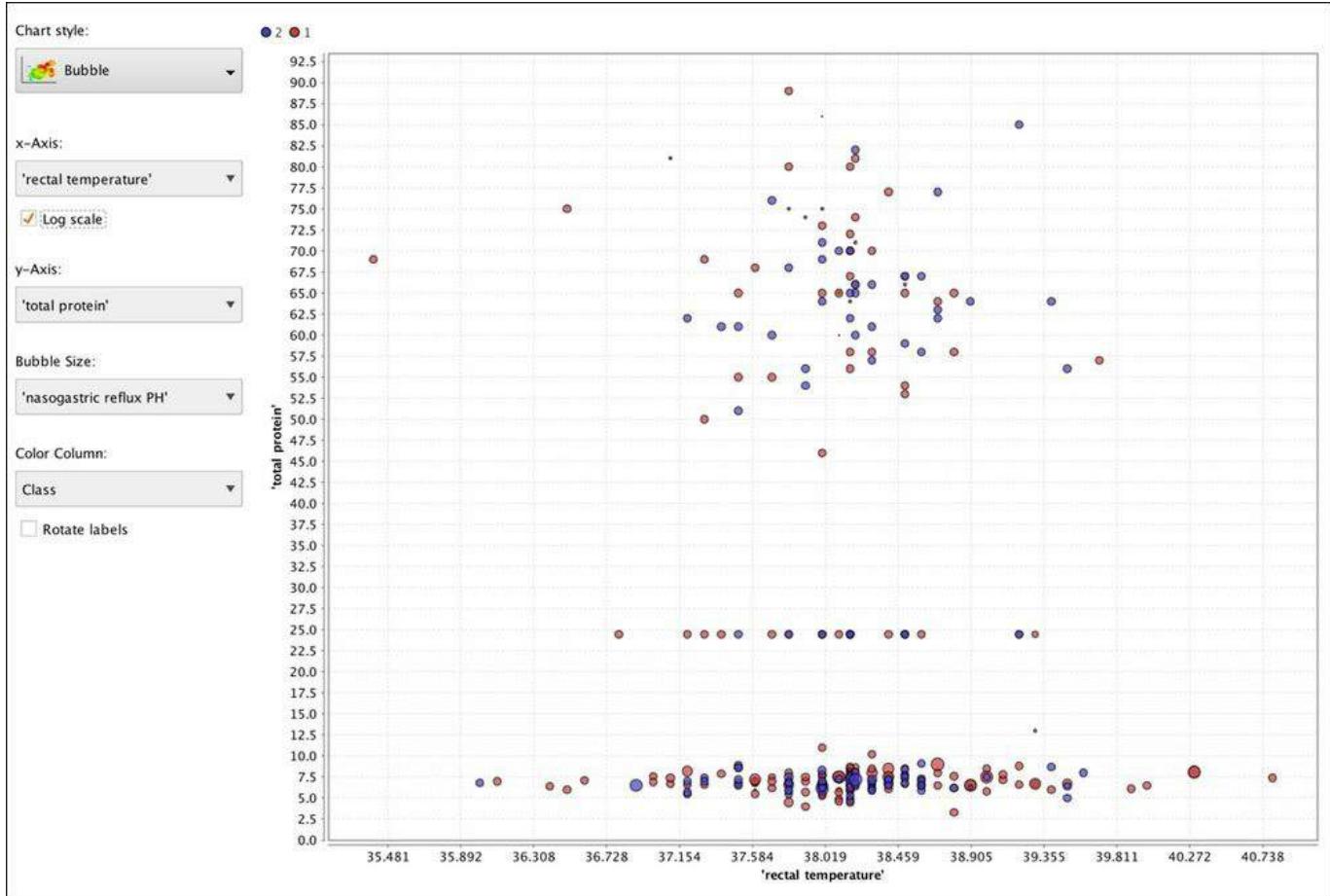


Figure 17: Bubble chart

In the bubble chart we can visualize four features at once by using the graphing tools to specify the x and y axes as well as a third dimension expressed as the size of bubble representing the feature. The target class is denoted by the color.

At the low end of total protein, we see higher pH values in the mid-range of rectal temperature values. In this cluster, high pH values appear to show a stronger correlation to lesions that were surgical. Another cluster with wider variance in total protein is also found for values of total protein greater than 50. The variance in pH is also low in this cluster.

Feature selection

Having gained some insight into the data, we are ready to use some of the techniques

presented in the theory that evaluate feature relevance.

Here we use two techniques: one that calculates the weights for features based on Chi-squared statistics with respect to the target attribute and the other based on the Gini Impurity Index. The results are shown in the table. Note that as we inferred while doing analysis of the features via visualization, both Pulse and Rectal Temperature prove to have low relevance as shown by both techniques.

Chi-squared		Gini index	
Attribute	Weight	Attribute	Weight
Pain	54.20626	Pain	0.083594
Abdomen	53.93882	Abdomen	0.083182
Peristalsis	38.73474	Peristalsis	0.059735
AbdominalDistension	35.11441	AbdominalDistension	0.054152
PeripheralPulse	23.65301	PeripheralPulse	0.036476
AbdominocentesisAppearance	20.00392	AbdominocentesisAppearance	0.030849
TemperatureOfExtremeties	17.07852	TemperatureOfExtremeties	0.026338
MucousMembranes	15.0938	MucousMembranes	0.023277
NasogastricReflux	14.95926	NasogastricReflux	0.023069
PackedCellVolume	13.5733	PackedCellVolume	0.020932
RectalExamination-Feces	11.88078	RectalExamination-Feces	0.018322
CapillaryRefillTime	8.078319	CapillaryRefillTime	0.012458
RespiratoryRate	7.616813	RespiratoryRate	0.011746
TotalProtein	5.616841	TotalProtein	0.008662
NasogastricRefluxPH	2.047565	NasogastricRefluxPH	0.003158

Pulse	1.931511	Pulse	0.002979
Age	0.579216	Age	8.93E-04
NasogastricTube	0.237519		
AbdomcentecisTotalProtein	0.181868		
RectalTemperature	0.139387		

Table 3: Relevant features determined by two different techniques, Chi-squared and Gini index.

Model process flow

In RapidMiner you can define a pipeline of computations using operators with inputs and outputs that can be chained together. The following process represents the flow used to perform the entire set of operations starting with loading the training and test data, handling missing values, weighting features by relevance, filtering out low scoring features, training an ensemble model that uses Bagging with Random Forest as the algorithm, and finally applying the learned model to the test data and outputting the performance metrics. Note that all the preprocessing steps that are applied to the training dataset must also be applied, in the same order, to the test set by means of the Group Models operator:

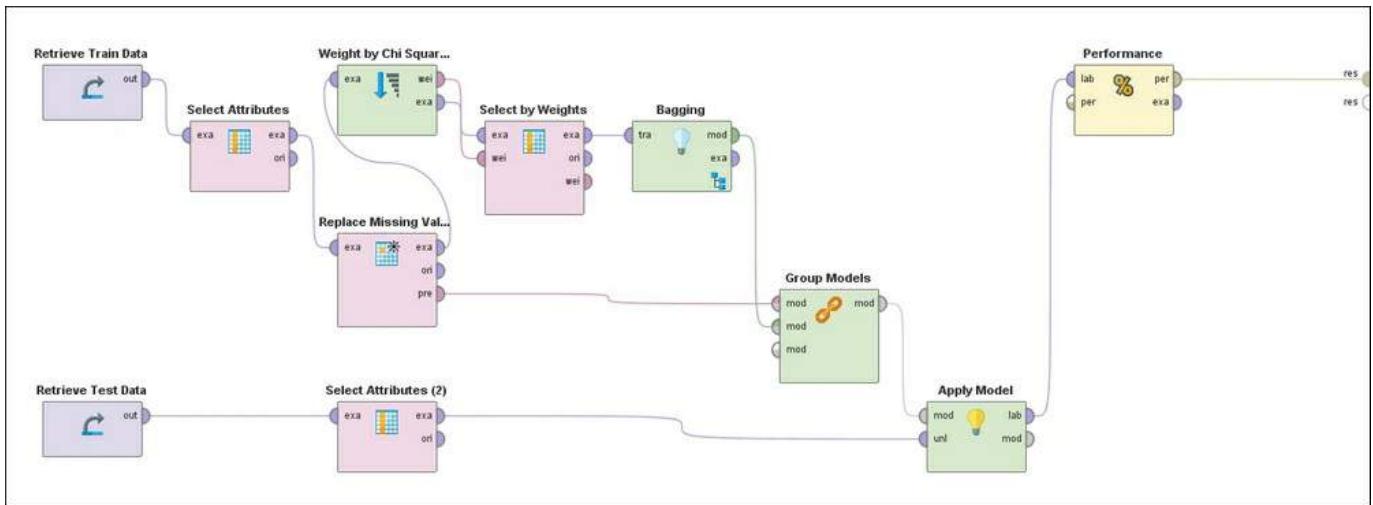


Figure 18: RapidMiner process diagram

Following the top of the process, the training set is ingested in the left-most operator, followed by the exclusion of non-predictors (Hospital Number, CP data) and self-predictors (Lesion 1). This is followed by the operator that replaces missing values with the mean and mode for continuous and categorical attributes, respectively. Next, the Feature Weights operator evaluates weights for each feature based on the Chi-squared statistic, which is followed by a filter that ignores low-weighted features. This pre-processed dataset is then used to train a model using Bagging with a Random Forest classifier.

The preprocessing steps used on the training data are grouped together in the appropriate order via the Group Models operator and applied to the test data in the penultimate step. Finally, the predictions of the target variable on the test examples accompanied by the confusion matrix and other performance metrics are made evaluated and presented in the last step.

Model evaluation metrics

We are now ready to compare the results from the various models. If you have followed along you may find that your results vary from what's presented here—that may be due to the stochastic nature of some learning algorithms, or differences in the values of some hyper-parameters used in the models.

We have considered three different training datasets:

- Original training data with missing values
- Training data transformed with missing values handled
- Training data transformed with missing values handled and with feature selection (Chi-Square) applied to select features that are highly discriminatory.

We have considered three different sets of algorithms on each of the datasets:

- Linear algorithms (Naïve Bayes and Logistic Regression)
- Non-linear algorithms (Decision Tree and KNN)
- Ensemble algorithms (Bagging, Ada Boost, and Random Forest).

Evaluation on Confusion Metrics

Models	TPR	FPR	Precision	Specificity	Accuracy	AUC
Naïve Bayes	68.29%	14.81%	87.50%	85.19%	75.00%	0.836

Logistic Regression	78.05%	14.81%	88.89%	85.19%	80.88%	0.856
Decision Tree	68.29%	33.33%	75.68%	66.67%	67.65%	0.696
k-NN	90.24%	85.19%	61.67%	14.81%	60.29%	0.556
Bagging (GBT)	90.24%	74.07%	64.91%	25.93%	64.71%	0.737
Ada Boost (Naïve Bayes)	63.41%	48.15%	66.67%	51.85%	58.82%	0.613

Table 4: Results on unseen (Test) data for models trained on Horse-colic data with missing values

Models	TPR	FPR	Precision	Specificity	Accuracy	AUC
Naïve Bayes	68.29%	66.67%	60.87%	33.33%	54.41%	0.559
Logistic Regression	78.05%	62.96%	65.31%	37.04%	61.76%	0.689
Decision Tree	97.56%	96.30%	60.61%	3.70%	60.29%	0.812
k-NN	75.61%	48.15%	70.45%	51.85%	66.18%	0.648
Bagging (Random Forest)	97.56%	74.07%	66.67%	25.93%	69.12%	0.892
Bagging (GBT)	82.93%	18.52%	87.18%	81.48%	82.35%	0.870
Ada Boost (Naïve Bayes)	68.29%	7.41%	93.33%	92.59%	77.94%	0.895

Table 5: Results on unseen (Test) data for models trained on Horse-colic data with missing values replaced

Models	TPR	FPR	Precision	Specificity	Accuracy	AUC
Naïve Bayes	75.61%	77.78%	59.62%	29.63%	54.41%	0.551
Logistic Regression	82.93%	62.96%	66.67%	37.04%	64.71%	0.692
Decision Tree	95.12%	92.59%	60.94%	7.41%	60.29%	0.824

k-NN	75.61%	48.15%	70.45%	51.85%	66.18%	0.669
Bagging (Random Forest)	92.68%	33.33%	80.85%	66.67%	82.35%	0.915
Bagging (GBT)	78.05%	22.22%	84.21%	77.78%	77.94%	0.872
Ada Boost (Naïve Bayes)	68.29%	18.52%	84.85%	81.48%	73.53%	0.848

Table 6: Results on unseen (Test) data for models trained on Horse-colic data using features selected by Chi-squared statistic technique

ROC Curves, Lift Curves, and Gain Charts

The performance plots enable us to visually assess the models used in two of the three experiments—without any replacement of missing data, and with using features from Chi-squared weighting after replacing missing data—and to compare them against each other. Pairs of plots display the performance curves of each Linear (Logistic Regression), Non-linear (Decision Tree), and Ensemble (Bagging, using Gradient Boosted Tree) technique we learned about earlier in the chapter, drawn from results of the two experiments.

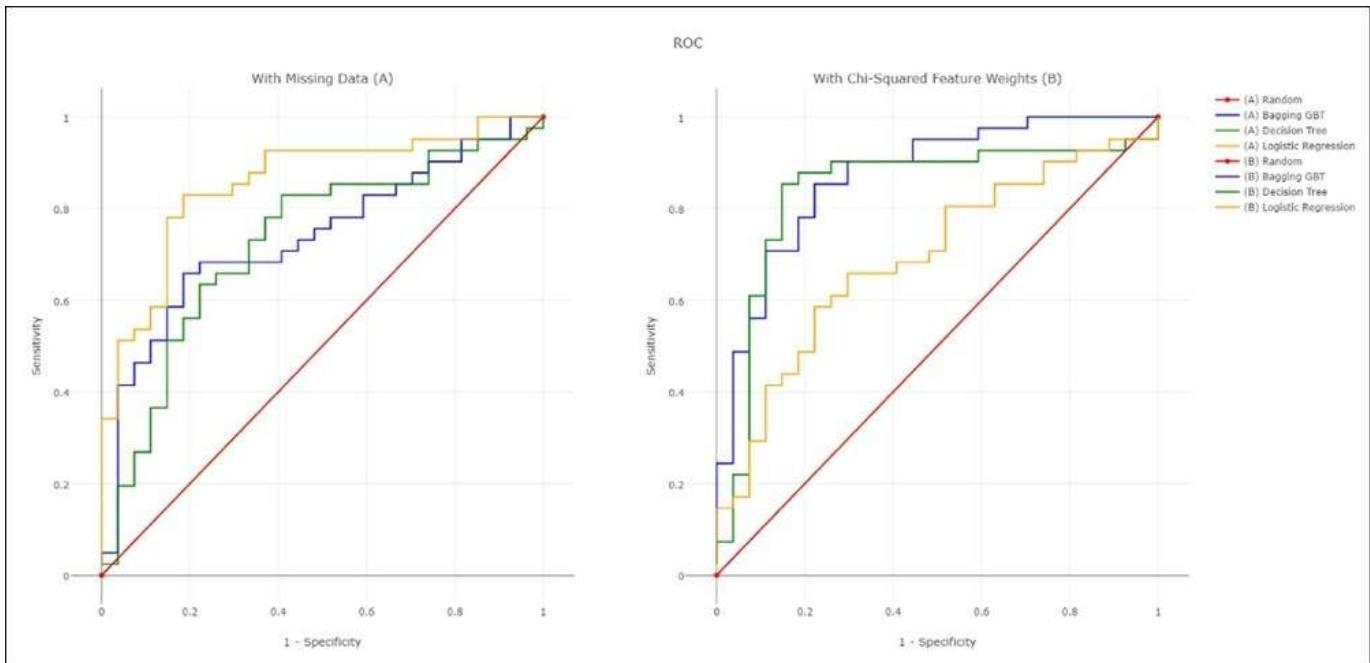


Figure 19: ROC Performance curves for experiment using Missing Data

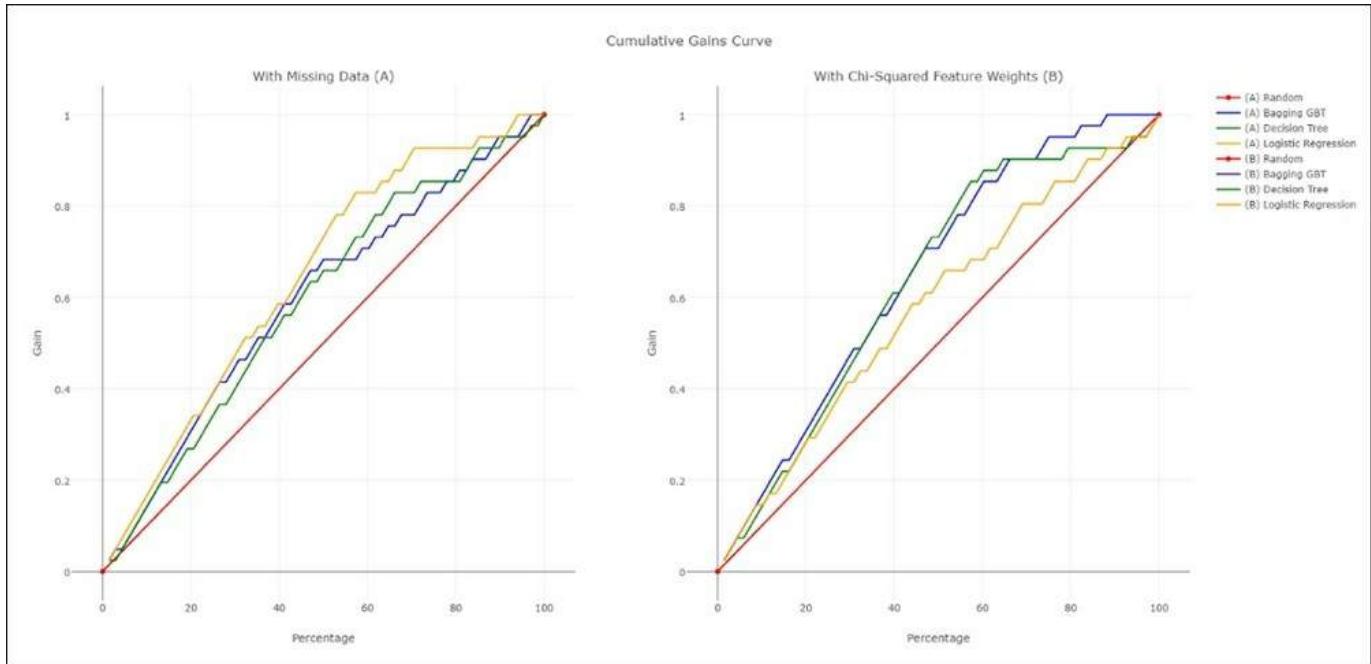


Figure 20: Cumulative Gains performance curves for experiment using Missing Data

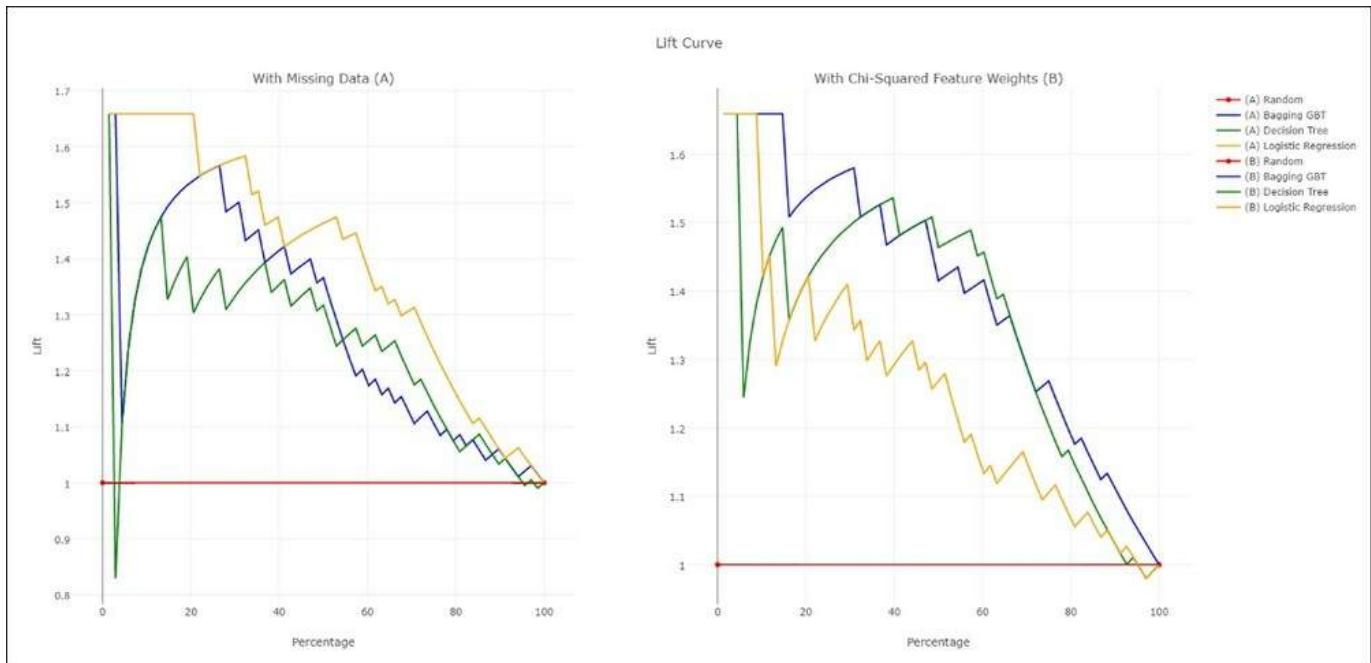


Figure 21: Lift performance curves for experiment using Missing Data

Results, observations, and analysis

The impact of handling missing values is significant. Of the seven classifiers, with the exception of Naïve Bayes and Logistic Regression, all show remarkable improvement when missing values are handled as indicated by various metrics, including AUC, precision, accuracy, and specificity. This tells us that handling missing values that can be "noisy" is an important aspect of data transformation. Naive Bayes has its own internal way of managing missing values and the results from our experiments show that it does a better job of null-handling than our external transformations. But in general, the idea of transforming missing values seems beneficial when you consider all of the classifiers.

As discussed in the section on modeling, some of the algorithms require the right handling of missing values and feature selection to get optimum performance. From the results, we can see that the performance of Decision Trees, for example, improved incrementally from 0.696 with missing data, 0.812 with managed missing data, and for the best performance of 0.824 with missing data handled together with feature selection. Six out of seven classifiers improve the performance in AUC (and in others metrics) when both the steps are performed; comparing *Table 5* and *Table 6* for AUC gives us these quick insights. This demonstrates the importance of doing preprocessing such as missing value handling along with feature selection before performing modeling.

A major conclusion from the results is that the problem is highly non-linear and therefore most non-linear classifiers from the simplest Decision Trees to ensemble Random Forest perform very well. The best performance comes from the meta-learning algorithm Random Forest, with missing values properly handled and the most relevant features used in training. The best linear model performance measured by AUC was 0.856 for Logistic Regression with data as-is (that is, with missing values), whereas Random Forest achieved AUC performance of 0.915 with proper handling of missing data accompanied by feature selection. Generally, as evident from *Table 3*, the non-linear classifiers or meta-learners performed better than linear classifiers by most performance measures.

Handling missing values, which can be thought as "noise", in the appropriate manner improves the performance of AdaBoost by a significant amount. The AUC improves from 0.613 to 0.895 and FPR reduces from 48.15 to 7.41%. This indeed conforms to the expected theoretical behavior for this technique.

Meta-learning techniques, which use concepts of boosting and bagging, are relatively more effective when dealing with real-world data, when compared to other common techniques. This seems to be justified by the results since AdaBoost with Naïve Bayes as base learner trained on data that has undergone proper handling of noise outperforms Naive Bayes in most of the metrics, as shown in *Table 5* and *Table 6*. Random Forest and GBTs also show the best performance along with AdaBoost as compared to base classifiers in *Table 6*, again confirming that the right process and ensemble learning can produce the most optimum results in real-world noisy datasets.

Note

All data, models, and results for both WEKA and RapidMiner process files from this chapter are available at: <https://github.com/mjmlbook/mastering-java-machine-learning/tree/master/Chapter2>.

Summary

Supervised learning is the predominant technique used in machine learning applications. The methodology consists of a series of steps beginning with data exploration, data transformation, and data sampling, through feature reduction, model building, and ultimately, model assessment and comparison. Each step of the process involves some decision making which must answer key questions: How should we impute missing values? What data sampling strategy should we use? What is the most appropriate algorithm given the amount of noise in the dataset and the prescribed goal of interpretability? This chapter demonstrated the application of these processes and techniques to a real-world problem—the classification problem using the UCI Horse Colic dataset.

Whether the problem is one of classification, when the target is a categorical value, or Regression, when it is a real-valued continuous variable, the methodology used for supervised learning is similar. In this chapter, we have used classification for illustration.

The first step is data quality analysis, which includes descriptive statistics of the features, visualization analysis using univariate, and multivariate feature analysis. With the help of various plot types, we can uncover different trends in the data and examine how certain features may or may not correlate with the label values and with each other. Data analysis is followed by data pre-processing, where the techniques include ways to address noise, as in the case of missing data, and outliers, as well as preparing the data for modeling techniques through normalization and discretization.

Following pre-processing, we must suitably split the data into train, validation, and test samples. Different sampling strategies may be used depending on the characteristics of the data and the problem at hand, for example, when the data is skewed or when we have a multi-class classification problem. Depending on data size, cross-validation is a common alternative to creating a separate validation set.

The next step is the culling of irrelevant features. In the filter approach, techniques that use univariate analysis are either entropy-based (Information Gain, Gains Ratio) or based on statistical hypothesis testing (Chi-Squared). With the main multivariate methods, the aim is reduction of redundant features when considered together, or using the ones that correlate most closely with the target label. In the wrapper

approach, we use machine learning algorithms to tell us about the more discriminating features. Finally, some learning techniques have feature selection embedded in the algorithm in the form of a regularization term, typically using ridge or lasso techniques. These represent the embedded approach.

Modeling techniques are broadly classified into linear, non-linear, and ensemble methods. Among linear algorithms, the type of features can determine the algorithms to use—Linear Regression (numeric features only), Naïve Bayes (numeric or categorical), and logistic regression (numeric features only, or categorical transformed to numeric) are the work-horses. The outlined advantages and disadvantages of each method must be understood when choosing between them or interpreting the results of learning using these models.

Decision Tree, k-NN, and SVM are non-linear techniques, each with their own strengths and limitations. For example, interpretability is the main advantage of Decision Tree. k-NN is robust in the face of noisy data, but it does poorly with high-dimensional data. SVM suffers from poor interpretability, but shines even when the dataset is limited, and the number of features is large.

With a number of different models collaborating, ensemble methods can leverage the best of all. Bagging and boosting both are techniques that generalize better in the ensemble compared to the base learner they use and are popular in many applications.

Finally, what are the strategies and methods that can be used in evaluating model performance and comparing models to each other? The role of validation sets or cross-validation is essential to the ability to generalize over unseen data.

Performance evaluation metrics derived from the confusion matrix are used universally to evaluate classifiers; some are used more commonly in certain domains and disciplines than others. ROC, Gain, and Lift curves are great visual representations of the range of model performance as the classification threshold is varied. When comparing models in pairs, several metrics based on statistical hypothesis testing are used. Wilcoxon and McNemar's are two non-parametric tests; Paired-t test is an example of a parametric method. Likewise, when comparing multiple algorithms, a common non-parametric test that does not make assumptions about the data distribution is Friedman's test. ANOVA, which are parametric tests, assume normal distribution of the metrics and equal variances.

The final sections of the chapter present the process undertaken using the

RapidMiner tool to develop and evaluate models generated to classify test data from the UCI Horse-colic dataset. Three experiments are designed to compare and contrast the performance of models under different data pre-processing conditions, namely, without handling missing data, with replacement of missing data using standard techniques, and finally, with feature selection following null replacement. In each experiment we choose multiple linear, non-linear, and ensemble methods. As part of the overall process, we illustrate how the modeling environment is used. We can draw revealing conclusions from the results, which give us insights into the data as well as demonstrating the relative strengths and weakness of the various classes of techniques in different situations. We conclude that the data is highly non-linear and that ensemble learning demonstrates clear advantages over other techniques.

References

1. D. Bell and H. Wang (2000). *A Formalism for Relevance and its Application in Feature Subset Selection*. *Machine Learning*, 41(2):175–195.
2. J. Doak (1992). *An Evaluation of Feature Selection Methods and their Application to Computer Security*. Technical Report CSE-92-18, Davis, CA: University of California, Department of Computer Science.
3. M. Ben-Bassat (1982). *Use of Distance Measures, Information Measures and Error Bounds in Feature Evaluation*. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics*, volume 2, pages 773–791, North Holland.
4. Littlestone N, Warmuth M (1994) *The weighted majority algorithm*. *Information Computing* 108(2):212–261
5. Breiman L., Friedman J.H., Olshen R.A., Stone C.J. (1984) *Classification and Regression Trees*, Wadsforth International Group.
6. B. Ripley(1996), *Pattern recognition and neural networks*. Cambridge University Press, Cambridge.
7. Breiman, L., (1996). *Bagging Predictors*, *Machine Learning*, 24 123-140.
8. Burges, C. (1998). *A tutorial on support vector machines for pattern recognition*. *Data Mining and Knowledge Discovery*. 2(2):1-47.
9. Bouckaert, R. (2004), *Naive Bayes Classifiers That Perform Well with Continuous Variables*, *Lecture Notes in Computer Science*, Volume 3339, Pages 1089 – 1094.
10. Aha D (1997). *Lazy learning*, Kluwer Academic Publishers, Dordrecht
11. Nadeau, C. and Bengio, Y. (2003), *Inference for the generalization error*. In *Machine Learning* 52:239– 281.
12. Quinlan, J.R. (1993). *C4.5: Programs for machine learning*, Morgan Kaufmann, San Francisco.
13. Vapnik, V. (1995), *The Nature of Statistical Learning Theory*. Springer Verlag.
14. Schapire RE, Singer Y, Singhal A (1998). *Boosting and Rocchio applied to text filtering*. In SIGIR '98: Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval, pp 215–223
15. Breiman L.(2001). *Random Forests*. *Machine Learning*, 45 (1), pp 5-32.
16. Nathalie Japkowicz and Mohak Shah (2011). *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press.
17. Hanley, J. & McNeil, B. (1982). *The meaning and use of the area under a receiver operating characteristic (ROC) curve*. *Radiology* 143, 29–36.
18. Tjen-Sien, L., Wei-Yin, L., Yu-Shan, S. (2000). *A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New*

- Classification Algorithms.* Machine Learning 40: 203–228.
- 19. A. W. Moore and M. S. Lee (1994). *Efficient Algorithms for Minimizing Cross Validation Error*. In Proc. of the 11th Int. Conf. on Machine Learning, pages 190–198, New Brunswick, NJ. Morgan Kaufmann.
 - 20. Nitesh V. Chawla et. al. (2002). *Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research. 16:321-357.

Chapter 3. Unsupervised Machine Learning Techniques

In the last chapter, we focused on supervised learning, that is, learning from a training dataset that was labeled. In the real world, obtaining data with labels is often difficult. In many domains, it is virtually impossible to label data either due to the cost of labeling or difficulty in labeling due to the sheer volume or velocity at which data is generated. In those situations, unsupervised learning, in its various forms, offers the right approaches to explore, visualize, and perform descriptive and predictive modeling. In many applications, unsupervised learning is often coupled with supervised learning as a first step to isolate interesting data elements for labeling.

In this chapter, we will focus on various methodologies, techniques, and algorithms that are practical and well-suited for unsupervised learning. We begin by noting the issues that are common between supervised and unsupervised learning when it comes to handling data and transformations. We will then briefly introduce the particular challenges faced in unsupervised learning owing to the lack of "ground truth" and the nature of learning under those conditions.

We will then discuss the techniques of feature analysis and dimensionality reduction applied to unlabeled datasets. This is followed by an introduction to the broad spectrum of clustering methods and discussions on the various algorithms in practical use, just as we did with supervised learning in [Chapter 2, Practical Approach to Real-World Supervised Learning](#), showing how each algorithm works, when to use it, and its advantages and limitations. We will conclude the section on clustering by presenting the different cluster evaluation techniques.

Following the treatment of clustering, we will approach the subject of outlier detection. We will contrast various techniques and algorithms that illustrate what makes some objects outliers—also called anomalies—within a given dataset.

The chapter will conclude with clustering and outlier detection experiments, conducted with a real-world dataset and an analysis of the results obtained. In this case study, we will be using ELKI and SMILE Java libraries for the machine learning tasks and will present code and results from the experiments. We hope that this will provide the reader with a sense of the power and ease of use of these tools.

Issues in common with supervised learning

Many of the issues that we discussed related to supervised learning are also common with unsupervised learning. Some of them are listed here:

- **Types of features handled by the algorithm:** Most clustering and outlier algorithms need numeric representation to work effectively. Transforming categorical or ordinal data has to be done carefully
- **Curse of dimensionality:** Having a large number of features results in sparse spaces and affects the performance of clustering algorithms. Some option must be chosen to suitably reduce dimensionality—either feature selection where only a subset of the most relevant features are retained, or feature extraction, which transforms the feature space into a new set of principal variables of a lower dimensional space
- **Scalability in memory and training time:** Many unsupervised learning algorithms cannot scale up to more than a few thousands of instances either due to memory or training time constraints
- **Outliers and noise in data:** Many algorithms are affected by noise in the features, the presence of anomalous data, or missing values. They need to be transformed and handled appropriately

Issues specific to unsupervised learning

The following are some issues that pertain to unsupervised learning techniques:

- **Parameter setting:** Deciding on number of features, usefulness of features, number of clusters, shapes of clusters, and so on, pose enormous challenges to certain unsupervised methods
- **Evaluation methods:** Since unsupervised learning methods are ill-posed due to lack of ground-truth, evaluation of algorithms becomes very subjective.
- **Hard or soft labeling:** Many unsupervised learning problems require giving labels to the data in an exclusive or probabilistic manner. This poses a problem for many algorithms
- **Interpretability of results and models:** Unlike supervised learning, the lack of ground truth and the nature of some algorithms make interpreting the results from both model and labeling even more difficult

Feature analysis and dimensionality reduction

Among the first tools to master are the different feature analysis and dimensionality reduction techniques. As in supervised learning, the need for reducing dimensionality arises from numerous reasons similar to those discussed earlier for feature selection and reduction.

A smaller number of discriminating dimensions makes visualization of data and clusters much easier. In many applications, unsupervised dimensionality reduction techniques are used for compression, which can then be used for transmission or storage of data. This is particularly useful when the larger data has an overhead. Moreover, applying dimensionality reduction techniques can improve the scalability in terms of memory and computation speeds of many algorithms.

Notation

We will use similar notation to what was used in the chapter on supervised learning. The examples are in d dimensions and are represented as vector:

$$\mathbf{x} = (x_1, x_2, \dots, x_d)^T$$

The entire dataset containing n examples can be represented as an observation matrix:

$$\mathbf{X} = \left\{ x_{ij} : 1 \leq i \leq d, 1 \leq j \leq n \right\}$$

The idea of dimensionality reduction is to find $k \leq d$ features either by transformation of the input features, projecting or combining them such that the lower dimension k captures or preserves interesting properties of the original dataset.

Linear methods

Linear dimensionality methods are some of the oldest statistical techniques to reduce features or transform the data into lower dimensions, preserving interesting discriminating properties.

Mathematically, with linear methods we are performing a transformation, such that a new data element is created using a linear transformation of the original data element:

$$s_i = w_{i,1}x_1 + w_{i,2}x_2 + \cdots w_{id}x_d \text{ for } i=1,2,\dots,k$$

$$\mathbf{s} = \mathbf{W}\mathbf{x}$$

Here, $\mathbf{W}_{k \times d}$ is the linear transformation matrix. The variables \mathbf{s} are also referred to as latent or hidden variables.

In this topic, we will discuss the two most practical and often-used methodologies. We will list some variants of these techniques so that the reader can use the tools to experiment with them. The main assumption here—which often forms the limitation—is the linear relationships between the transformations.

Principal component analysis (PCA)

PCA is a widely-used technique for dimensionality reduction(*References [1]*). The original coordinate system is rotated to a new coordinate system that exploits the directions of maximum variance in the data, resulting in uncorrelated variables in a lower-dimensional subspace that were correlated in the original feature space. PCA is sensitive to the scaling of the features.

Inputs and outputs

PCA is generally effective on numeric datasets. Many tools provide the categorical-to-continuous transformations for the nominal features, but this affects the performance. The number of principal components, or k , is also an input provided by the user.

How does it work?

PCA, in its most basic form, tries to find projections of data onto new axes, which are known as **principal components**. Principal components are projections that capture maximum variance directions from the original space. In simple words, PCA finds the first principal component through rotation of the original axes of the data in the direction of maximum variance. The technique finds the next principal component by again determining the next best axis, orthogonal to the first axis, by seeking the second highest variance and so on until most variances are captured. Generally, most tools give either a choice of number of principal components or the option to keep finding components until some percentage, for example, 99%, of variance in the original dataset is captured.

Mathematically, the objective of finding maximum variance can be written as

$$= \max_{\|w\|=1} \|W^T X\|^2$$

= largest eigenvalue of $X^T X$ (covariance matrix)

$\lambda v = Cv$ is the eigendecomposition

This is equivalent to:

$$X_{d \times n} = W_{d \times d} \times S_{d \times n}$$

Here, W is the principal components and S is the new transformation of the input data. Generally, eigenvalue decomposition or singular value decomposition is used in the computation part.

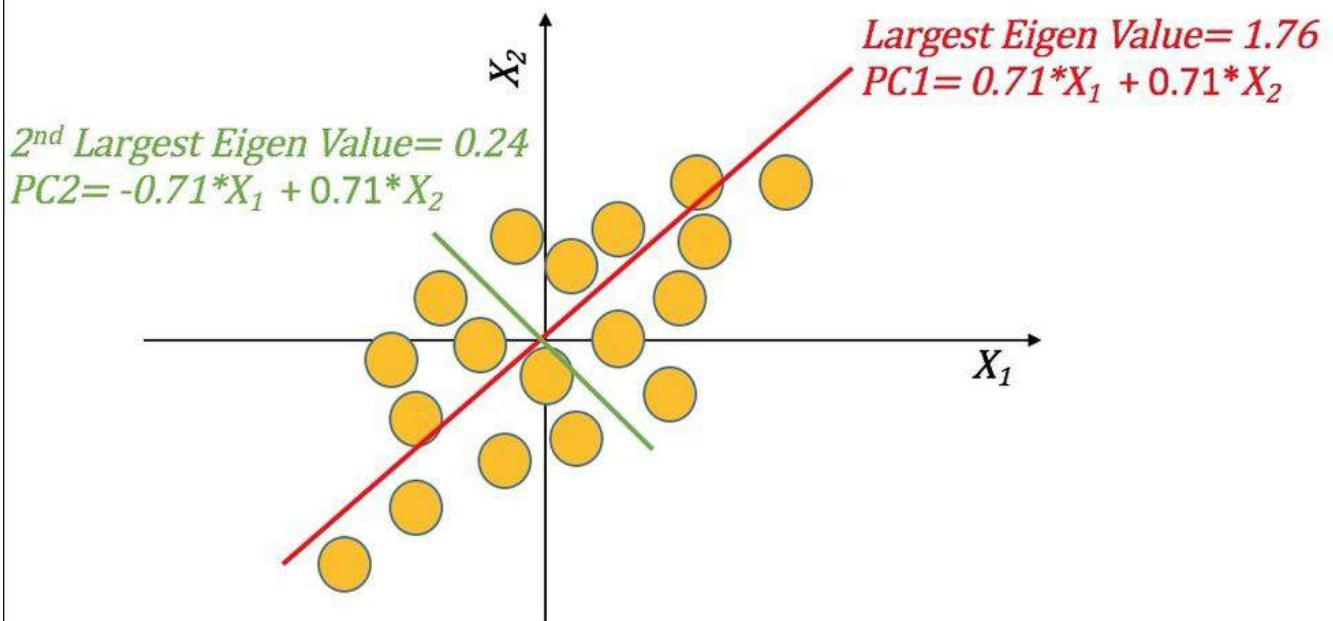


Figure 1: Principal Component Analysis

Advantages and limitations

- One of the advantages of PCA is that it is optimal in that it minimizes the reconstruction error of the data.
- PCA assumes normal distribution.
- The computation of variance-covariance matrix can become intensive for large datasets with high-dimensions. Alternatively, **Singular Value Decomposition (SVD)** can be used as it works iteratively and there is no need for an explicit covariance matrix.
- PCA has issues when there is noise in the data.
- PCA fails when the data lies in the complex manifold, a topic that we will discuss in the non-linear dimensionality reduction section.
- PCA assumes a correlation between the features and in the absence of those correlations, it is unable to do any transformations; instead, it simply ranks them.
- By transforming the original feature space into a new set of variables, PCA causes a loss in interpretability of the data.

- There are many other variants of PCA that are popular and overcome some of the biases and assumptions of PCA.

Independent Component Analysis (ICA) assumes that there are mixtures of non-Gaussians from the source and, using the generative technique, tries to find the decompositions of original data in the smaller mixtures or components (*References [2]*). The key difference between PCA and ICA is that PCA creates components that are uncorrelated, while ICA creates components that are independent.

Mathematically, it assumes $X \in \mathbb{R}^{d \times n}$ as a mixture of independent sources $\in \mathbb{R}^{k \times n}$, such that each data element $y = [y^1, y^2, \dots, y^k]^T$ and independence is implied by $p(y) \approx \prod_{j=1}^k p(y^j)$:

Probabilistic Principal Component Analysis (PPCA) is based on finding the components using mixture models and maximum likelihood formulations using **Expectation Maximization (EM)** (*References [3]*). It overcomes the issues of missing data and outlier impacts that PCA faces.

Random projections (RP)

When data is separable by a large margin—even if it is high-dimensional data—one can randomly project the data down to a low-dimensional space without impacting separability and achieve good generalization with a relatively small amount of data. Random Projections use this technique and the details are described here (*References [4]*).

Inputs and outputs

Random projections work with both numeric and categorical features, but categorical features are transformed into binary. Outputs are lower dimensional representations of the input data elements. The number of dimensions to project, k , is part of user-defined input.

How does it work?

This technique uses random projection matrices to project the input data into a lower dimensional space. The original data $\mathbf{X} \in \mathbb{R}^d$ is transformed to the lower dimension space $\mathbf{S} \in \mathbb{R}^k$ where $k \ll p$ using:

$$\mathbf{S} = \mathbf{R}\mathbf{X}$$

Here columns in the $k \times d$ matrix \mathbf{R} are i.i.d zero mean normal variables and are scaled to unit length. There are variants of how the random matrix \mathbf{R} is constructed using probabilistic sampling. Computational complexity of RP is $O(knd)$, which scales much better than PCA. In many practical datasets, it has been shown that RP gives results comparable to PCA and can scale to large dimensions and datasets.

Advantages and limitations

- It scales to very large values of dataset size and dimensionalities. In text and image learning problems, with large dimensions, this technique has been successfully used as the preprocessing technique.
- Sometimes a large information loss can occur while using RP.

Multidimensional Scaling (MDS)

There are many forms of MDS—classical, metric, and non-metric. The main idea of MDS is to preserve the pairwise similarity/distance values. It generally involves transforming the high dimensional data into two or three dimensions (*References [5]*).

Inputs and outputs

MDS can work with both numeric and categorical data based on the user-selected distance function. The number of dimensions to transform to, k , is a user-defined input.

How does it work?

Given n data elements, an $n \times n$ affinity or distance matrix is computed. There are choices of using distances such as Euclidean, Mahalanobis, or similarity concepts such as cosine similarity, Jaccard coefficients, and so on. MDS in its very basic form tries to find a mapping of the distance matrix in a lower dimensional space where the Euclidean distance between the transformed points is similar to the affinity matrix.

Mathematically:

$$\min_Y \sum_{i=1}^n \sum_{j=1}^n (d_{ij}^X - d_{ij}^Y)^2$$

Here $d_{ij}^X = \|x_i - x_j\|$ input space and $d_{ij}^Y = \|y_i - y_j\|$ mapped space.

If the input affinity space is transformed using kernels then the MDS becomes a non-linear method for dimensionality reduction. Classical MDS is equivalent to PCA when the distances between the points in input space is Euclidean distance.

Advantages and limitations

- The key disadvantage is the subjective choice of the lower dimension needed to interpret the high dimensional data, normally restricted to two or three for humans. Some data may not map effectively in this lower dimensional space.
- The advantage is you can perform linear and non-linear mapping to the lowest dimensions using the framework.

Nonlinear methods

In general, nonlinear dimensionality reduction involves either performing nonlinear transformations to the computations in linear methods such as KPCA or finding nonlinear relationships in the lower dimension as in manifold learning. In some domains and datasets, the structure of the data in lower dimensions is nonlinear—and that is where techniques such as KPCA are effective—while in some domains the data does not unfold in lower dimensions and you need manifold learning.

Kernel Principal Component Analysis (KPCA)

Kernel PCA uses the Kernel trick described in [Chapter 2, Practical Approach to Real-World Supervised Learning](#), with the PCA algorithm for transforming the data in a high-dimensional space to find effective mapping (*References [6]*).

Inputs and outputs

Similar to PCA with addition of choice of kernel and kernel parameters. For example, if **Radial Basis Function (RBF)** or Gaussian Kernel is chosen, then the kernel, along with the gamma parameter, becomes user-selected values.

How does it work?

In the same way as **Support Vector Machines (SVM)** was discussed in the previous chapter, KPCA transforms the input space to high dimensional feature space using the "kernel trick". The entire PCA machinery of finding maximum variance is then carried out in the transformed space.

As in PCA:

= largest eigenvalue of $X^T X$ (covariance matrix)

$$C = \frac{1}{n} \sum_{i=1}^n x_i x_i^T$$

Instead of linear covariance matrix, a nonlinear transformation is applied to the input space using kernel methods by constructing the $N \times N$ matrix, in place of doing the actual transformations using $\phi(x)$.

$$k(x,y) = (\phi(x), \phi(y)) = \phi(x)^T \phi(y)$$

Since the kernel transformation doesn't actually transform the features into explicit feature space, the principal components found can be interpreted as projections of data onto the components. In the following figure, a binary nonlinear dataset, generated using the scikit-learn example on circles (*References [27]*), demonstrates the linear separation after KPCA using the RBF kernel and returning to almost similar input space by the inverse transform:

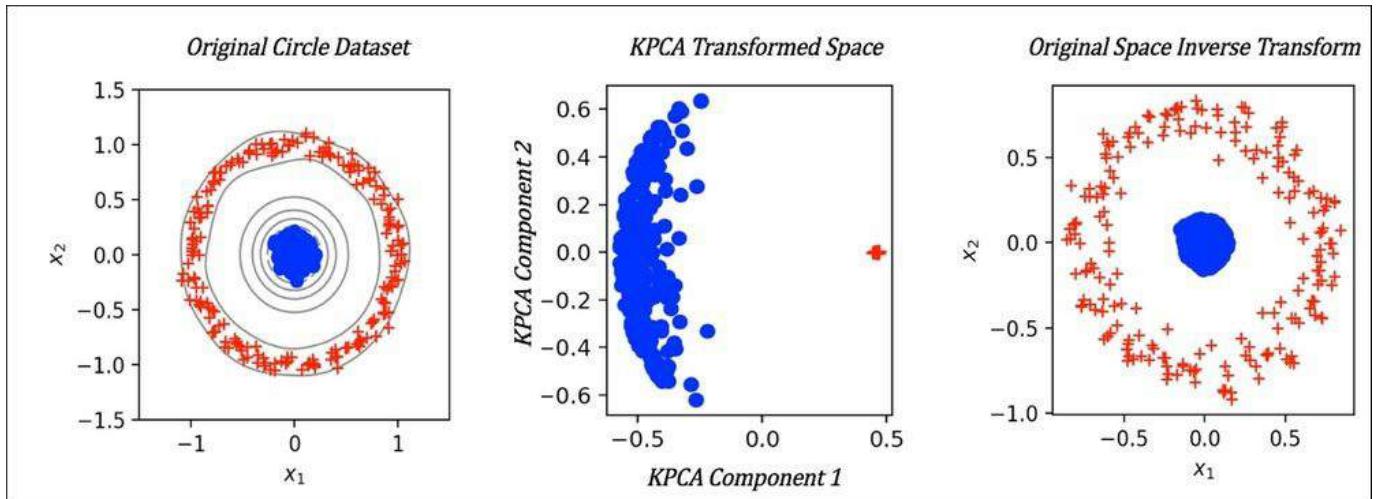


Figure 2: KPCA on Circle Dataset and Inverse Transform.

Advantages and limitations

- KPCA overcomes the nonlinear mapping presented by PCA.
- KPCA has similar issues with outlier, noisy, and missing values to standard PCA. There are robust methods and variations to overcome this.
- KPCA has scalability issues in space due to an increase in the kernel matrix, which can become a bottleneck in large datasets with high dimensions. SVD can be used in these situations, as an alternative.

Manifold learning

When high dimensional data is embedded in lower dimensions that are nonlinear, but

have complex structure, manifold learning is very effective.

Inputs and outputs

Manifold learning algorithms require two user-provided parameters: k , representing the number of neighbors for the initial search, and n , the number of manifold coordinates.

How does it work?

As seen in the following figure, the three-dimensional S-Curve, plotted using the scikit-learn utility (*References [27]*), is represented in 2D PCA and in 2D manifold using LLE. It is interesting to observe how the blue, green, and red dots are mixed up in the PCA representation while the manifold learning representation using LLE cleanly separates the colors. It can also be observed that the rank ordering of Euclidean distances is not maintained in the manifold representation:

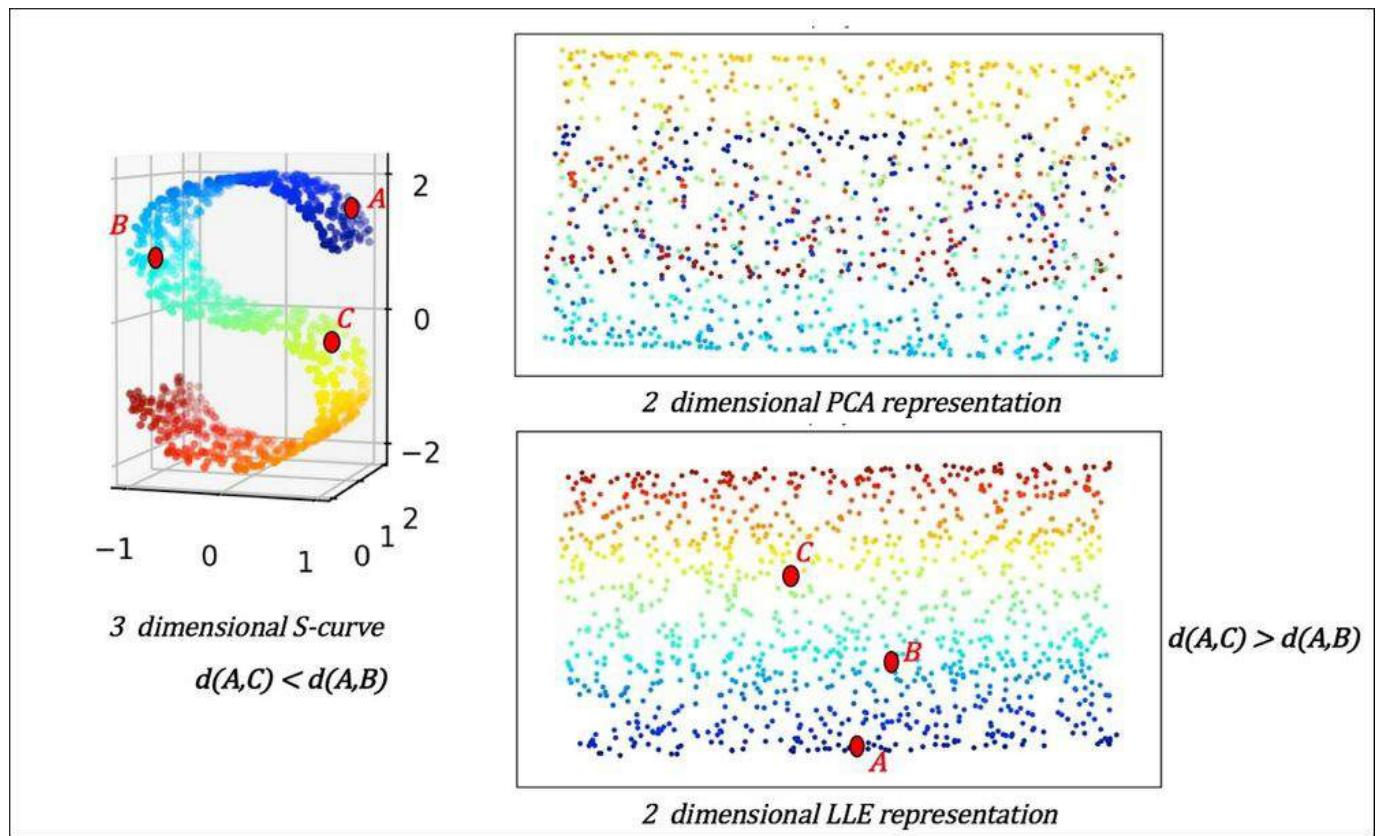


Figure 3: Data representation after PCA and manifold learning

To preserve the structure, the geodesic distance is preserved instead of the Euclidean

distance. The general approach is to build a graph structure such as an adjacency matrix, and then compute geodesic distance using different assumptions. In the Isomap Algorithm, the global pairwise distances are preserved (*References* [7]). In the **Local Linear Embedding (LLE)** Algorithm, the mapping is done to take care of local neighborhood, that is, nearby points map to nearby points in the transformation (*References* [9]). Laplacian Eigenmaps is similar to LLE, except it tries to maintain the "locality" instead of "local linearity" in LLE by using graph Laplacian (*References* [8]).

Advantages and limitations

- Isomap is non-parametric; it preserves the global structure, and has no local optimum, but is hampered by speed.
- LLE and Laplacian Eigenmaps are non-parametric, have no local optima, are fast, but don't preserve global structure.

Clustering

Clustering algorithms can be categorized in different ways based on the techniques, the outputs, the process, and other considerations. In this topic, we will present some of the most widely used clustering algorithms.

Clustering algorithms

There is a rich set of clustering techniques in use today for a wide variety of applications. This section presents some of them, explaining how they work, what kind of data they can be used with, and what their advantages and drawbacks are. These include algorithms that are prototype-based, density-based, probabilistic partition-based, hierarchy-based, graph-theory-based, and those based on neural networks.

k-Means

k-means is a centroid- or prototype-based iterative algorithm that employs partitioning and relocation methods (*References [10]*). k-means finds clusters of spherical shape depending on the distance metric used, as in the case of Euclidean distance.

Inputs and outputs

k-means can handle mostly numeric features. Many tools provide categorical to numeric transformations, but having a large number of categoricals in the computation can lead to non-optimal clusters. User-defined k , the number of clusters to be found, and the distance metric to use for computing closeness are two basic inputs. k-means generates clusters, association of data to each cluster, and centroids of clusters as the output.

How does it work?

The most common variant known as Lloyd's algorithm initializes k centroids for the given dataset by picking data elements randomly from the set. It assigns each data element to the centroid it is closest to, using some distance metric such as Euclidean distance. It then computes the mean of the data points for each cluster to form the new centroid and the process is repeated until either the maximum number of iterations is reached or there is no change in the centroids.

Mathematically, each step of the clustering can be seen as an optimization step where the equation to optimize is given by:

$$\arg \min_c \sum_{i=1}^k \sum_{x \in c_i} (x - \mu_i) = \arg \min_c \sum_{i=1}^k \sum_{x \in c_i} \|x - \mu_i\|_2^2$$

Here, c_i is all points belong to cluster i . The problem of minimizing is classified as NP-hard and hence k-Means has a tendency to get stuck in local optimum.

Advantages and limitations

- The choice of the number of clusters, k , is difficult to pick, but normally search techniques such as varying k for different values and measuring metrics such as sum of square errors can be used to find a good threshold. For smaller datasets, hierarchical k-Means can be tried.
- k-means can converge faster than most algorithms for smaller values of k and can find effective global clusters.
- k-means convergence can be affected by initialization of the centroids and hence there are many variants to perform random restarts with different seeds and so on.
- k-means can perform badly when there are outliers and noisy data points. Using robust techniques such as medians instead of means, k-Medoids, overcomes this to a certain extent.
- k-means does not find effective clusters when they are of arbitrary shapes or have different densities.

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based partitioning algorithm. It separates dense region in the space from sparse regions (*References* [14]).

Inputs and outputs

Only numeric features are used in DBSCAN. The user-defined parameters are $MinPts$ and the neighborhood factor given by ϵ .

How does it work?

The algorithm first finds the ϵ -neighborhood of every point p , given by

$N_\epsilon : \{q | d(p, q) \leq \epsilon\}$. A *high density* region is identified as a region where the number of points in a ϵ -neighborhood is greater than or equal to the given *MinPts*; the point such a ϵ -neighborhood is defined around is called a *core points*. Points within the ϵ -neighborhood of a *core point* are said to be *directly reachable*. All *core points* that can in effect be reached by hopping from one directly reachable core point to another point *directly reachable* from the second point, and so on, are considered to be in the same cluster. Further, any point that has fewer than *MinPts* in its ϵ -neighborhood, but is directly reachable from a core point, belongs to the same cluster as the core point. These points at the edge of a cluster are called *border points*. A *noise point* is any point that is neither a core point nor a border point.

Advantages and limitations

- The DBSCAN algorithm does not require the number of clusters to be specified and can find it automatically from the data.
- DBSCAN can find clusters of various shapes and sizes.
- DBSCAN has in-built robustness to noise and can find outliers from the datasets.
- DBSCAN is not completely deterministic in its identification of the points and its categorization into border or core depends on the order of data processed.
- Distance metrics selected such as Euclidean distance can often affect performance due to the curse of dimensionality.
- When there are clusters with large variations in the densities, the static choice of $\{\text{MinPts}, \epsilon\}$ can pose a big limitation.

Mean shift

Mean shift is a very effective clustering algorithm in many image, video, and motion detection based datasets (*References* [11]).

Inputs and outputs

Only numeric features are accepted as data input in the mean shift algorithm. The choice of kernel and the bandwidth of the kernel are user-driven choices that affect the performance. Mean shift generates modes of data points and clusters data around the modes.

How does it work?

Mean shift is based on the statistical concept of **kernel density estimation (KDE)**, which is a probabilistic method to estimate the underlying data distribution from the

sample.

A kernel density estimate for kernel $K(\mathbf{x})$ of given bandwidth h is given by:

$$f(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

For n points with dimensionality d . The mean shift algorithm works by moving each data point in the direction of local increasing density. To estimate this direction, gradient is applied to the KDE and the gradient takes the form of:

$$\nabla f(\mathbf{x}) = \frac{2}{nh^{d+2}} \left(\sum_{i=1}^n g\left(\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right) \right) m(\mathbf{x})$$

$$m(\mathbf{x}) = \left[\frac{\sum_{i=1}^n \mathbf{x}_i g\left(\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right)}{\sum_{i=1}^n g\left(\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right)} \right]$$

Here $\mathbf{g}(\mathbf{x}) = -K'(\mathbf{x})$ is the derivative of the kernel. The vector, $m(\mathbf{x})$, is called the mean shift vector and it is used to move the points in the direction

$$\mathbf{x}^{(t+1)} = \mathbf{x}^t + m(\mathbf{x})$$

Also, it is guaranteed to converge when the gradient of the density function is zero. Points that end up in a similar location are marked as clusters belonging to the same

region.

Advantages and limitations

- Mean shift is non-parametric and makes no underlying assumption on the data distribution.
- It can find non-complex clusters of varying shapes and sizes.
- There is no need to explicitly give the number of clusters; the choice of the bandwidth parameter, which is used in estimation, implicitly controls the clusters.
- Mean shift has no local optima for a given bandwidth parameter and hence it is deterministic.
- Mean shift is robust to outliers and noisy points because of KDE.
- The mean shift algorithm is computationally slow and does not scale well with large datasets.
- Bandwidth selection should be done judiciously; otherwise it can result in merged modes, or the appearance of extra, shallow modes.

Expectation maximization (EM) or Gaussian mixture modeling (GMM)

GMM or EM is a probabilistic partition-based method that partitions data into k clusters using probability distribution-based techniques (*References [13]*).

Input and output

Only numeric features are allowed in EM/GMM. The model parameter is the number of mixture components, given by k .

How does it work?

GMM is a generative method that assumes that there are k Gaussian components, each Gaussian component has a mean μ_i and covariance Σ_i . The following expression represents the probability of the dataset given the k Gaussian components:

$$= p(data | \mu_1, \mu_2, \dots, \mu_k)$$

$$= \prod_{i=1}^n p(\mathbf{x}_i | \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k)$$

$$= \prod_{i=1}^n \sum_{j=1}^k p(\mathbf{x}_i | \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k) P(w_j)$$

The two-step task of finding the means $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k\}$ for each of the k Gaussian components such that the data points assigned to each maximizes the probability of that component is done using the **Expectation Maximization (EM)** process.

The iterative process can be defined into an E-step, that computes the *expected* cluster for all data points for the cluster, in an iteration i :

$$P(w_i | x_k, \lambda_t) = \frac{p(x_k | w_i, \lambda_t) P(w_i | \lambda_t)}{p(x_k | \lambda_t)}$$

The M-step maximizes to compute μ_{t+1} given the data points belonging to the cluster:

$$\mu_{t+1} = \frac{\sum_k P(w_i | x_k, \lambda_t) x_k}{\sum_k P(w_i | x_k, \lambda_t)}$$

The EM process can result in GMM convergence into local optimum.

Advantages and limitations

- Works very well with any features; for categorical data, discrete probability is calculated, while for numeric a continuous probability function is estimated.
- It has computational scalability problems. It can result in local optimum.
- The value of k Gaussians has to be given *a priori*, similar to k-Means.

Hierarchical clustering

Hierarchical clustering is a connectivity-based method of clustering that is widely used to analyze and explore the data more than it is used as a clustering technique (*References* [12]). The idea is to iteratively build binary trees either from top or bottom, such that similar points are grouped together. Each level of the tree provides interesting summarization of the data.

Input and output

Hierarchical clustering generally works on similarity-based transformations and so both categorical and continuous data are accepted. Hierarchical clustering only needs the similarity or distance metric to compute similarity and does not need the number of clusters like in k-means or GMM.

How does it work?

There are many variants of hierarchical clustering, but we will discuss agglomerative clustering. Agglomerative clustering works by first putting all the data elements in their own groups. It then iteratively merges the groups based on the similarity metric used until there is a single group. Each level of the tree or groupings provides unique segmentation of the data and it is up to the analyst to choose the right level that fits the problem domain. Agglomerative clustering is normally visualized using a dendrogram plot, which shows merging of data points at similarity. The popular choices of similarity methods used are:

- **Single linkage:** Similarity is the minimum distance between the groups of points:

$$d_{SL}(A, B) = \min_{i \in A, j \in B} d_{ij}$$

- **Complete linkage:** Similarity is the maximum distance between the groups of points:

$$d_{CL}(A, B) = \max_{i \in A, j \in B} d_{ij}$$

- **Average linkage:** Average similarity between the groups of points:

$$d_{AL}(A, B) = \frac{1}{N_A N_B} \sum_{i \in A} \sum_{j \in B} d_{ij}$$

Advantages and limitations

- Hierarchical clustering imposes a hierarchical structure on the data even when there may not be such a structure present.
- The choice of similarity metrics can result in a vastly different set of merges and dendrogram plots, so it has a large dependency on user input.
- Hierarchical clustering suffers from scalability with increased data points.
Based on the distance metrics used, it can be sensitive to noise and outliers.

Self-organizing maps (SOM)

SOM is a neural network based method that can be viewed as dimensionality reduction, manifold learning, or clustering technique (*References [17]*). Neurobiological studies show that our brains map different functions to different areas, known as topographic maps, which form the basis of this technique.

Inputs and outputs

Only numeric features are used in SOM. Model parameters consists of distance function, (generally Euclidean distance is used) and the lattice parameters in terms of width and height or number of cells in the lattice.

How does it work?

SOM, also known as Kohonen networks, can be thought of as a two-layer neural network where each output layer is a two-dimensional lattice, arranged in rows and columns and each neuron is fully connected to the input layer.

Like neural networks, the weights are initially generated using random values. The

process has three distinct training phases:

- **Competitive phase:** Neurons in this phase compete based on the discriminant values, generally based on distance between neuron weight and input vector; such that the minimal distance between the two decides which neuron the input gets assigned to. Using Euclidean distance, the distance between an input x_i and neuron in the lattice position (j, i) is given by w_{ji} :

$$d_j(\mathbf{x}) = \sum_{i=1}^d (x_i - w_{ji})^2$$

- **Cooperation phase:** In this phase, the winning neurons find the best spatial location in the topological neighborhood. The topological neighborhood for the winning neuron $I(x)$ for a given neuron (j, i) , at a distance S_{ij} , neighborhood of size σ , is defined by:

$$T_{j,I(x)} = \exp\left(\frac{-S_{ij}^2}{2\sigma(t)^2}\right)$$

The neighborhood size is defined in the way that it decreases with time using some well-known decay functions such as an exponential, function defined as follows:

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0}\right)$$

- **Adaptive phase:** In this phase, the weights of the winning neuron and its neighborhood neurons are updated. The update to weights is generally done using:

$$\Delta w_{ji} = \eta(t) \cdot T_{j,I(x)}(t) \cdot (x_i - w_{ji})$$

Here, the learning rate $n(t)$ is again defined as exponential decay like the neighborhood size.

SOM Visualization using Unified Distance Matrix (U-Matrix) creates a single metric of average distance between the weights of the neuron and its neighbors, which then can be visualized in different color intensities. This helps to identify *similar* neurons in the neighborhood.

Advantages and limitations

- The biggest advantage of SOM is that it is easy to understand and clustering of the data in two dimensions with U-matrix visualization enables understanding the patterns very effectively.
- Choice of similarity/distance function makes vast difference in clusters and must be carefully chosen by the user.
- SOM's computational complexity makes it impossible to use on datasets greater than few thousands in size.

Spectral clustering

Spectral clustering is a partition-based clustering technique using graph theory as its basis (*References* [15]). It converts the dataset into a connected graph and does graph partitioning to find the clusters. This is a popular method in image processing, motion detection, and some unstructured data-based domains.

Inputs and outputs

Only numeric features are used in spectral clustering. Model parameters such as the choice of kernel, the kernel parameters, the number of eigenvalues to select, and partitioning algorithms such as k-Means must be correctly defined for optimum performance.

How does it work?

The following steps describe how the technique is used in practice:

1. Given the data points, an affinity (or adjacency) matrix is computed using a smooth kernel function such as the Gaussian kernel:

$$A_{ij} \approx \exp\left(-\alpha \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

For the points that are closer, $A_{ij} \rightarrow 0$ and for points further away, $A_{ij} \rightarrow 1$

2. The next step is to compute the graph Laplacian matrix using various methods of normalizations. All Laplacian matrix methods use the diagonal degree matrix D , which measures degree at each node in the graph:

$$D_{ij} = \sum_j^n a_{ij}$$

A simple Laplacian matrix is $L = D$ (degree matrix) – A (affinity matrix).

3. Compute the first k eigenvalues from the eigenvalue problem or the generalized eigenvalue problem.
4. Use a partition algorithm such as k-Means to further separate clusters in the k-

dimensional subspace.

Advantages and limitations

- Spectral clustering works very well when the cluster shape or size is irregular and non-convex. Spectral clustering has too many parameter choices and tuning to get good results is quite an involved task.
- Spectral clustering has been shown, theoretically, to be more stable in the presence of noisy data. Spectral clustering has good performance when the clusters are not well separated.

Affinity propagation

Affinity propagation can be viewed as an extension of K-medoids method for its similarity with picking exemplars from the data (*References* [16]). Affinity propagation uses graphs with distance or the similarity matrix and picks all examples in the training data as exemplars. Iterative message passing as *affinities* between data points automatically detects clusters, the exemplars, and even the number of clusters.

Inputs and outputs

Typically, other than maximum number of iterations, which is common to most algorithms, no input parameters are required.

How does it work?

Two kinds of messages are exchanged between the data points that we will explain first:

- Responsibility $r(i,k)$: This is a message from the data point to the candidate exemplar. This gives a metric of how well the exemplar is suited for that data point compared to other exemplars. The rules for updating the responsibility are

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} (a(i,k') + s(i,k'))$$

as follows:

where $s(i, k)$ = similarity between two data points i and k .

$a(i, k)$ = availability of exemplar k for i .

- Availability $a(i,k)$: This is a message from the candidate exemplar to a data point. This gives a metric indicating how good of a support the exemplar can be to the data point, considering other data points in the calculations. This can be viewed as soft cluster assignment. The rule for updating the availability is as follows:

$$a(i,k) \leftarrow \min \left\{ 0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k)) \right\}$$

$$a(i,k) \leftarrow \sum_{i' \notin k} \max(0, r(i',k))$$

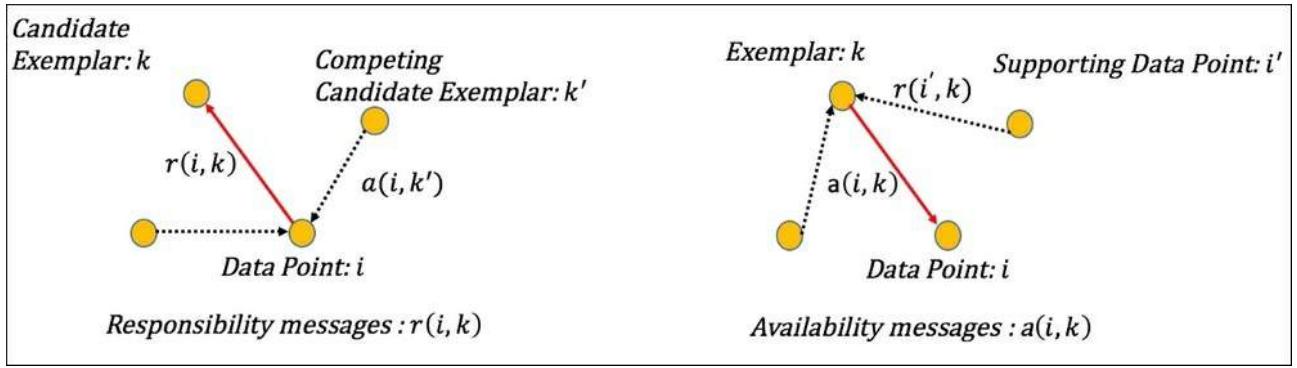


Figure 4: Message types used in Affinity Propagation

The algorithm can be summed up as follows:

$$1. \text{ Initialize } r(i, j) = 0, a(i, j) = 0 \forall i, j$$

2. For all increments i to n :

$$r(i, k) \leftarrow s(i, k) - \max_{k \neq k} (a(i, k') + s(i, k'))$$

$$a(i, k) \leftarrow \min \left\{ 0, r(k, k) + \sum_{i' \notin \{i, k\}} \max(0, r(i', k)) \right\}$$

$$(i, k) \leftarrow \sum_{i' \notin k} \max(0, r(i', k))$$

3. End.

4. For all x_i such that $(r(i, i) + a(i, i) > 0)$

1. x_i is exemplar.

2. All non-exemplars x_j are assigned to the closest exemplar using the similarity measure $s(i, j)$.

5. End.

Advantages and limitations

- Affinity propagation is a deterministic algorithm. Both k-means or K-medoids are sensitive to the selection of initial points, which is overcome by considering every point as an exemplar.
- The number of clusters doesn't have to be specified and is automatically determined through the process.
- It works in non-metric spaces and doesn't require distances/similarity to even have constraining properties such as triangle inequality or symmetry. This makes the algorithm usable on a wide variety of datasets with categorical and text data and so on:
- The algorithm can be parallelized easily due to its update methods and it has fast training time.

Clustering validation and evaluation

Clustering validation and evaluation is one of the most important mechanisms to determine the usefulness of the algorithms (*References [18]*). These topics can be broadly classified into two categories:

- **Internal evaluation measures:** In this the measures uses some form of clustering quality from the data themselves, without any access to the ground truth.
- **External evaluation measures:** In this the measures use some external information such as known ground truth or class labels.

Internal evaluation measures

Internal evaluation uses only the clusters and data information to gather metrics about how good the clustering results are. The applications may have some influence over the choice of the measures. Some algorithms are biased towards particular evaluation metrics. So care must be taken in choosing the right metrics, algorithms, and parameters based on these considerations. Internal evaluation measures are based on different qualities, as mentioned here:

- **Compactness:** Variance in the clusters measured using different strategies is used to give compactness values; the lower the variance, the more compact the cluster.
- **Separation:** How well are the clusters separated from each other?

Notation

Here's a compact explanation of the notation used in what follows: dataset with all data elements = D , number of data elements = n , dimensions or features of each data element= d , center of entire data $D = c$, number of clusters = NC , i^{th} cluster = C_i , number of data in the i^{th} cluster = n_i , center of i^{th} cluster = c_i , variance in the i^{th} cluster = $\sigma(C_i)$, distance between two points x and y = $d(x,y)$.

R-Squared

The goal is to measure the degree of difference between clusters using the ratio of the sum of squares between clusters to the total sum of squares on the whole data. The formula is given as follows:

$$\frac{\sum_{x \in D} \left(\|x - c\|^2 - \sum_i \sum_{x \in C_i} \|x - c_i\|^2 \right)}{\sum_{x \in D} \|x - c\|^2}$$

Dunn's Indices

The goal is to identify dense and well-separated clusters. The measure is given by maximal values obtained from the following formula:

$$\min_i \left\{ \min_j \left(\frac{\min_{x \in C_i, y \in C_j} d(x, y)}{\max_k \left\{ \max_{x, y \in C_k} d(x, y) \right\}} \right) \right\}$$

Davies-Bouldin index

The goal is to identify clusters with low intra-cluster distances and high inter-cluster distances:

$$\frac{1}{NC} \cdot \sum_i \max_{j, j \neq i} \left\{ \frac{\left(\frac{1}{n_i} \sum_{x \in c_i} d(x, c_i) + \frac{1}{n_j} \sum_{x \in c_j} d(x, c_j) \right)}{d(c_i, c_j)} \right\}$$

Silhouette's index

The goal is to measure the pairwise difference of between-cluster and within-cluster distances. It is also used to find optimal cluster number by maximizing the index. The formula is given by:

$$\frac{1}{NC} \cdot \sum_i \left\{ \frac{1}{n_i} \sum_{x \in C_i} \frac{b(x) - a(x)}{\max[b(x) - a(x)]} \right\}$$

Here $a(x) = \frac{1}{n_i - 1} \sum_{y \in C_i, y \neq x} d(x, y)$ and $b(x) = \min_{j, j \neq i} \left[\frac{1}{n_j} \sum_{y \in C_j} d(x, y) \right]$.

External evaluation measures

The external evaluation measures of clustering have similarity to classification metrics using elements from the confusion matrix or using information theoretic metrics from the data and labels. Some of the most commonly used measures are as follows.

Rand index

Rand index measures the correct decisions made by the clustering algorithm using the following formula:

$$= \frac{TP + TN}{TP + FP + FN + TN}$$

F-Measure

F-Measure combines the precision and recall measures applied to clustering as given in the following formula:

$$Precision(i, j) = \frac{n_{ij}}{n_j} \quad Recall(i, j) = \frac{n_{ij}}{n_i}$$

$$F\text{-}Measure = \frac{2 \cdot Precision(i, j) \cdot Recall(i, j)}{Precision(i, j) + Recall(i, j)}$$

Here, n_{ij} is the number of data elements of class i in the cluster j , n_j is the number of data in the cluster j and n_i is the number of data in the class i . The higher the F-Measure, the better the clustering quality.

Normalized mutual information index

NMI is one of the many entropy-based measures applied to clustering. The entropy associated with a clustering C is a measure of the uncertainty about a cluster picking a data element randomly.

$$\mathcal{H}(C) = -\sum_{i=1}^k P(i) \cdot \log_2 P(i) \quad \text{where } P(i) = \frac{|c_i|}{n} \text{ is the probability of the element getting picked in cluster } C_i.$$

Mutual information between two clusters is given by:

$$I(C, C') = \sum_{i=1}^k \sum_{j=1}^i P(i, j) \cdot \log_2 \frac{P(i, j)}{P(i) \cdot P(j)}$$

$$P(i, j) = \frac{|c_i \cap c_j|}{n}, \text{ which is the probability of the element being picked by both}$$

clusters C and C' .

Normalized mutual information (NMI) has many forms; one is given by:

$$NMI = \frac{I(C, C')}{\sqrt{\mathcal{H}(C)\mathcal{H}(C')}}$$

Outlier or anomaly detection

Grubbs, in 1969, offers the definition, "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs".

Hawkins, in 1980, defined outliers or anomaly as "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism".

Barnett and Lewis, 1994, defined it as "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data".

Outlier algorithms

Outlier detection techniques are classified based on different approaches to what it means to be an outlier. Each approach defines outliers in terms of some property that sets apart some objects from others in the dataset:

- **Statistical-based:** This is improbable according to a chosen distribution
- **Distance-based:** This is isolated from neighbors according to chosen distance measure and fraction of neighbors within threshold distance
- **Density-based:** This is more isolated from its neighbors than they are in turn from their neighbors
- **Clustering-based:** This is in isolated clusters relative to other clusters or is not a member of any cluster
- **High-dimension-based:** This is an outlier by usual techniques after data is projected to lower dimensions, or by choosing an appropriate metric for high dimensions

Statistical-based

Statistical-based techniques that use parametric methods for outlier detection assume some knowledge of the distribution of the data (*References [19]*). From the observations, the model parameters are estimated. Data points that have probabilities lower than a threshold value in the model are considered outliers. When the distribution is not known or none is suitable to assume, non-parametric methods are used.

Inputs and outputs

Statistical methods for outlier detection work with real-valued datasets. The choice of distance metric may be a user-selected input in the case of parametric methods assuming multivariate distributions. In the case of non-parametric methods using frequency-based histograms, a user-defined threshold frequency is used. Selection of kernel method and bandwidth are also user-determined in Kernel Density Estimation techniques. The output from statistical-based methods is a score indicating outlierness.

How does it work?

Most of the statistical-based outlier detections either assume a distribution or fit a distribution to the data to detect probabilistically the least likely data generated from

the distribution. These methods have two distinct steps:

1. **Training step:** Here, an estimate of the model to fit the data is performed
2. **Testing step:** On each instance a goodness of fit is performed based on the model and the particular instance, yielding a score and the outlierness

Parametric-based methods assume a distribution model such as multivariate Gaussians and the training normally involves estimating the means and variance using techniques such as **Maximum Likelihood Estimates (MLE)**. The testing typically includes techniques such as mean-variance or box-plot tests, accompanied by assumptions such as "if outside three standard deviations, then outlier".

A normal multivariate distribution can be estimated as:

$$N(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{\frac{-1}{2}(\mathbf{x}-\boldsymbol{\mu})\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

with the mean $\boldsymbol{\mu}$ and covariance Σ .

The Mahalanobis distance can be the estimate of the data point from the distribution given by the equation $(\mathbf{x}-\boldsymbol{\mu})\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})$. Some variants such as **Minimum Covariant Determinant (MCD)** are also used when Mahalanobis distance is affected by outliers.

A non-parametric method involves techniques such as constructing histograms for every feature using frequency or width-based methods. When the ratio of the data in a bin to that of the average over the histogram is below a user defined threshold, such a bin is termed sparse. A lower probability of feature results in a higher outlier score. The total outlier score can be computed as:

$$\text{OutlierScore} = \sum_{f \in F} w_f \frac{(1 - p_f)}{|F|}$$

Here, w_f is the weight given to feature f , p_f is the probability of the value of the feature in the test data point, and F is the sum of weights of the feature set. Kernel Density Estimations are also used in non-parametric methods using user-defined kernels and bandwidth.

Advantages and limitations

- When the model fits or distribution of the data is known, these methods are very efficient as you don't have to store entire data, just the key statistics for doing tests.
- Assumptions of distribution, however, can pose a big issue in parametric methods. Most non-parametric methods using kernel density estimates don't scale well with large datasets.

Distance-based methods

Distance-based algorithms work under the general assumption that normal data has other data points closer to it while anomalous data is well isolated from its neighbors (*References [20]*).

Inputs and outputs

Distance-based techniques require natively numeric or categorical features to be transformed to numeric values. Inputs to distance-based methods are the distance metric used, the distance threshold ϵ , and π , the threshold fraction, which together determine if a point is an outlier. For KNN methods, the choice k is an input.

How does it work?

There are many variants of distance-based outliers and we will discuss how each of them works at a high level:

- DB (ϵ, π) Algorithms: Given a radius of ϵ and threshold of π , a data point is considered as an outlier if π percentage of points have distance to the point less than ϵ . There are further variants using nested loop structures, grid-based structures, and index-based structures on how the computation is done.
- KNN-based methods are also very common where the outlier score is computed either by taking the KNN distance to the point or the average distance to point from $\{1NN, 2NN, 3NN \dots KNN\}$.

Advantages and limitations

- The main advantage of distance-based algorithms is that they are non-parametric and make no assumptions on distributions and how to fit models.
- The distance calculations are straightforward and computed in parallel, helping the algorithms to scale on large datasets.
- The major issues with distance-based methods is the curse of dimensionality discussed in the first chapter; for large dimensional data, sparsity can lead to noisy outliers.

Density-based methods

Density-based methods extend the distance-based methods by not only measuring the local density of the given point, but also the local densities of its neighborhood points. Thus, the relative factor added gives it the edge in finding more complex outliers that are local or global in nature, but at the added cost of computation.

Inputs and outputs

Density-based algorithm must be supplied the minimum number of points $MinPts$ in a neighborhood of input radius ϵ centered on an object that determines it is a core object in a cluster.

How does it work?

We will first discuss the **Loca Outlier Factor (LOF)** method and then discuss some variants of LOF [21].

Given the $MinPts$ as the parameter, LOF of a data point is:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in MinPts(p)} \frac{lrdf_{MinPts}(o)}{lrdf_{MinPts}(p)}}{|N_{MinPts}(p)|}$$

Here $|N_{MinPts}(p)|$ is the number of data points in the neighborhood of point p , and $lrdf_{MinPts}$ is the local reachability density of the point and is defined as:

$$lrd_{MinPts}(p) = \frac{1}{\sum_{o \in MinPts(p)} reachDist_{MinPts}(o, p) / |N_{MinPts}(p)|}$$

Here $reachDist_{MinPts}(o, p)$ is the reachability of the point and is defined as:

$$reachDist_{MinPts}(o, p) = \max(MinPtsDistance(o, p), distance(o, p))$$

One of the disadvantages of LOF is that it may miss outliers whose neighborhood density is close to that of its neighborhood. **Connectivity-based outliers (COF)** using set-based nearest path and set-based nearest trail originating from the data point are used to improve on LOF. COF treats the low-density region differently to the isolated region and overcomes the disadvantage of LOF:

$$COF(p)_k = \frac{|N(p)_k| \cdot averageChainDist_{N(p)_k}(p)}{\sum_{o \in N(p)_k} averageChainDist_{N(p)_k}(p)}$$

Another disadvantage of LOF is that when clusters are in varying densities and not separated, LOF will generate counter-intuitive scores. One way to overcome this is to use the **influence space (IS)** of the points using KNNs and its reverse KNNs or RNNs. RNNs have the given point as one of their K nearest neighbors. Outlierness of the point is known as Influenced Outliers or INFLO and is given by:

$$INFLO(p)_k = \frac{den_{avg} IS_k(p)}{den(p)}$$

Here, $den(p)$ is the local density of p :

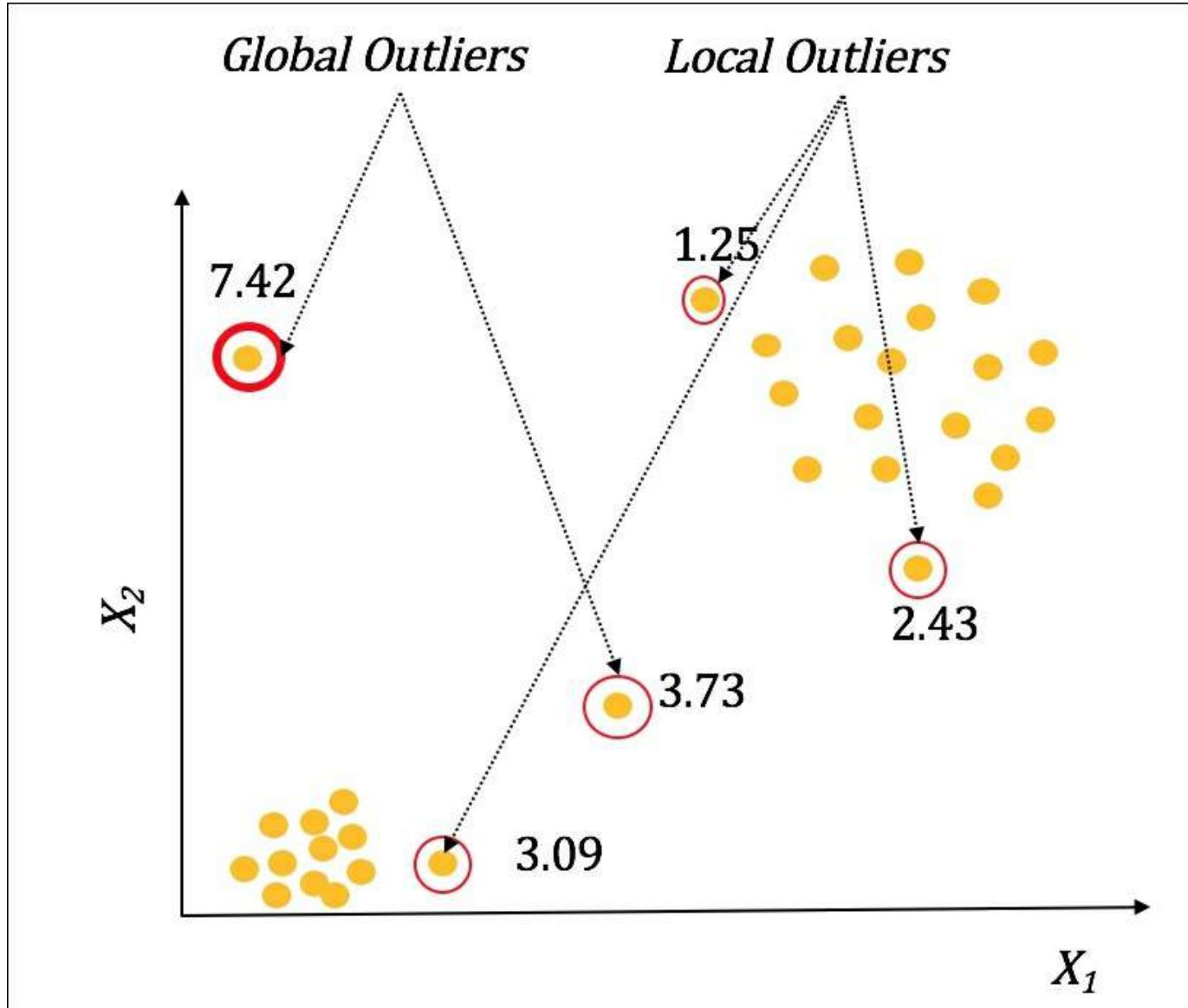


Figure 5: Density-based outlier detection methods are particularly suited for finding local as well as global outliers

Advantages and limitations

- It has been shown that density-based methods are more effective than distance-based methods.
- Density-based outlier detection has high computational cost and, often, poor interpretability.

Clustering-based methods

Some believe that clustering techniques, where the goal is to find groups of data points located together, are in some sense antithetical to the problem of anomaly or outlier detection. However, as an advanced unsupervised learning technique, clustering analysis offers several methods to find interesting groups of clusters that are either located far off from other clusters or do not lie in any clusters at all.

Inputs and outputs

As seen before, clustering techniques work well with real-valued data, although some categorical values translated to numeric values are tolerated. In the case of k-Means and k-Medoids, input values include the number of clusters k and the distance metric. Variants may require a threshold score to identify outlier groups. For Gaussian Mixture Models using EM, the number of mixture components must be supplied by the user. When using CBLOF, two user-defined parameters are expected: the size of small clusters and the size of large clusters. Depending on the algorithm used, individual or groups of objects are output as outliers.

How does it work?

As we discussed in the section on clustering, there are various types of clustering methods and we will give a few examples of how clustering algorithms have been extended for outlier detection.

k-Means or k-Medoids and their variants generally cluster data elements together and are affected by outliers or noise. Instead of preprocessing these data points by removal or transformation, such points that weaken the "tightness" of the clusters are considered outliers. Typically, outliers are revealed by a two-step process of first running clustering algorithms and then evaluating some form of outlier score that measures distance from point to centroid. Also, many variants treat clusters of size smaller than a threshold as an outlier group.

Gaussian mixture modeling (GMM) using Expectation maximization (EM) is

another well-known clustering-based outlier detection technique, where a data point that has low probability of belonging to a cluster becomes an outlier and the outlier score becomes the inverse of the EM probabilistic output score.

Cluster-based Local Outlier Factor (CBLOF) uses a two-stage process to find outliers. First, a clustering algorithm performs partitioning of data into clusters of various sizes. Using two user-defined parameters, size of large clusters, and size of small clusters, two sets of clusters are formed:

$$LC = \text{Set of Large Clusters} \{C_1, C_2, \dots, C_i\}$$

$$SC = \text{Set of Small Clusters} \{C_{i+1}, C_{i+2}, \dots, C_n\}$$

$$CBLOF(p) = \begin{cases} |C_i| \cdot \min(p, d(p, C_j)) & \text{if } p \in C_i, C_i \in SC, C_j \in LC \\ |C_i| \cdot d(p, C_i) & \text{if } p \in C_i, C_i \in LC \end{cases}$$

Advantages and limitations

- Given that clustering-based techniques are well-understood, results are more interpretable and there are more tools available for these techniques.
- Many clustering algorithms only detect clusters, and are less effective in unsupervised techniques compared to outlier algorithms that give scores or ranks or otherwise identify outliers.

High-dimensional-based methods

One of the key issues with distance-, density-, or even clustering-based methods, is the curse of dimensionality. As dimensions increase, the contrast between distances diminishes and the concept of neighborhood becomes less meaningful. The normal points in this case look like outliers and false positives increase by large volume. We will discuss some of the latest approaches taken in addressing this problem.

Inputs and outputs

Algorithms that project data to lower-dimensional subspaces can handle missing data well. In these techniques, such as SOD, ϕ , the number of ranges in each dimension becomes an input (*References* [25]). When using an evolutionary algorithm, the

number of cells with the lowest sparsity coefficients is another input parameter to the algorithm.

How does it work?

The broad idea to solve the high dimensional outlier issue is to:

- Either have a robust distance metric coupled with all of the previous techniques, so that outliers can be identified in full dimensions
- Or project data on to smaller subspaces and find outliers in the smaller subspaces

The **Angle-based Outlier Degree (ABOD)** method uses the basic assumption that if a data point in high dimension is an outlier, all the vectors originating from it towards data points nearest to it will be in more or less the same direction.

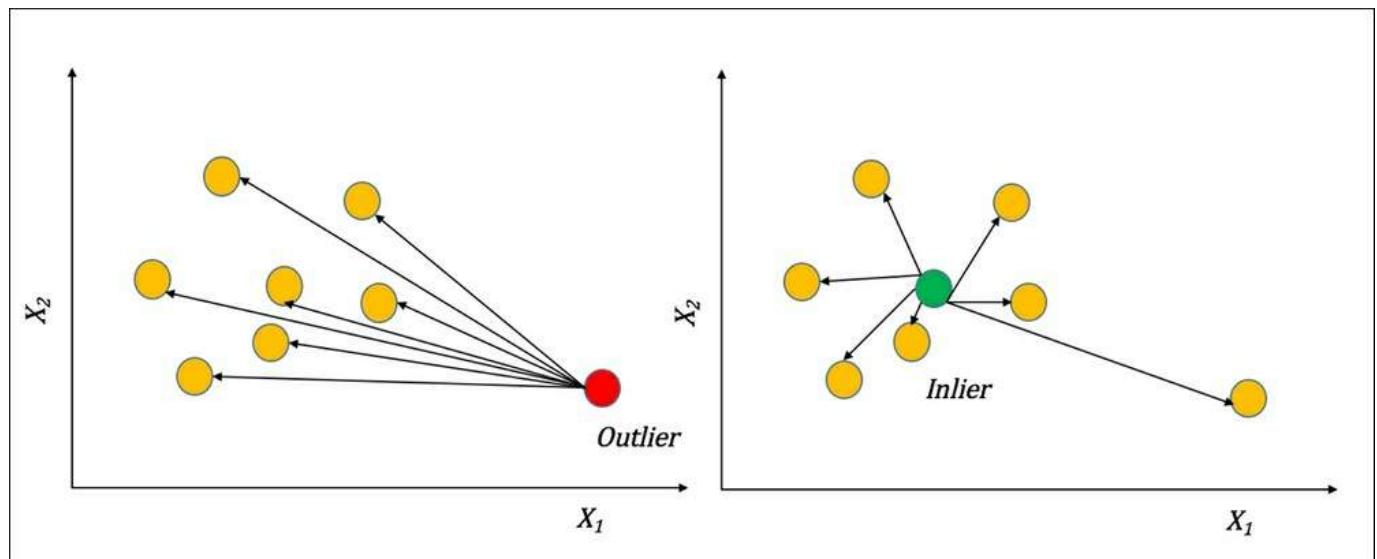


Figure 6: The ABOD method of distinguishing outliers from inliers

Given a point p , and any two points x and y , the angle between the two points and p is given by:

$$(\overrightarrow{px}, \overrightarrow{py})$$

Measure of variance used as the ABOD score is given by:

$$ABOD(p) = VAR_{x,y} \left[\frac{(\vec{px}, \vec{py})}{\|\vec{px}\|^2 \|\vec{py}\|^2} \right]$$

The smaller the ABOD value, the smaller the measure of variance in the angle spectrum, and the larger the chance of the point being the outlier.

Another method that has been very useful in high dimensional data is using the **Subspace Outlier Detection (SOD)** approach (*References [23]*). The idea is to partition the high dimensional space such that there are an equal number of ranges, say ϕ , in each of the d dimensions. Then the Sparsity Coefficient for a cell C formed by picking a range in each of the d dimensions is measured as follows:

$$\text{Sparsity}(C) = \frac{N(C) - n * \left(\frac{1}{\phi}\right)^d}{\sqrt{n * \left(\frac{1}{\phi}\right)^d * \left(1 - n * \left(\frac{1}{\phi}\right)^d\right)}}$$

Here, n is the total number of data points and $N(C)$ is the number of data points in cell C . Generally, the data points lying in cells with negative sparsity coefficient are considered outliers.

Advantages and limitations

- The ABOD method is $O(n^3)$ with the number of data points and becomes impractical with larger datasets.
- The sparsity coefficient method in subspaces requires efficient search in lower dimension and the problem becomes NP-Hard and some form of evolutionary or heuristic based search is employed.
- The sparsity coefficient methods being NP-Hard can result in local optima.

One-class SVM

In many domains there is a particular class or category of interest and the "rest" do not matter. Finding a boundary around this class of interest is the basic idea behind one-class SVM (*References [26]*). The basic assumption is that all the points of the positive class (class of interest) cluster together while the other class elements are spread around and we can find a tight hyper-sphere around the clustered instances. SVM, which has great theoretical foundations and applications in binary classifications is reformulated to solve one-class SVM. The following figure illustrates how a nonlinear boundary is simplified by using one-class SVM with slack so as to not overfit complex functions:

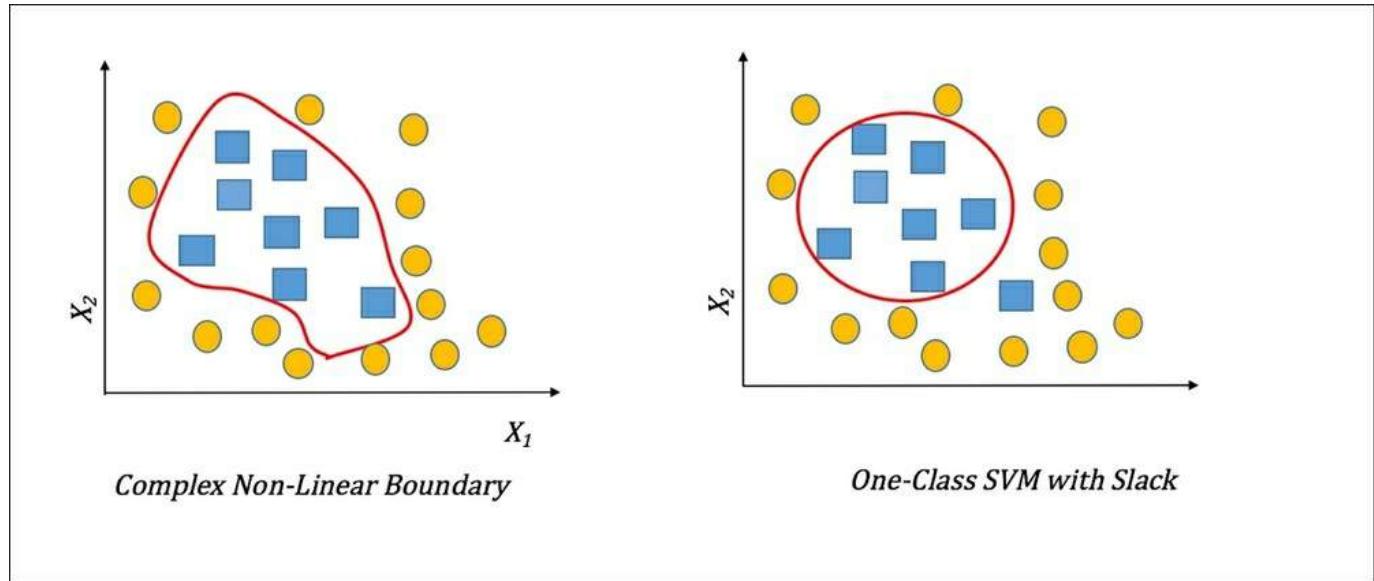


Figure 7: One-Class SVM for nonlinear boundaries

Inputs and outputs

Data inputs are generally numeric features. Many SVMs can take nominal features and apply binary transformations to them. Also needed are: marking the class of interest, SVM hyper-parameters such as kernel choice, kernel parameters and cost parameter, among others. Output is a SVM model that can predict whether instances belong to the class of interest or not. This is different from scoring models, which we have seen previously.

How does it work?

The input is training instances $\{x_1, x_2 \dots x_n\}$ with certain instances marked to be in class +1 and rest in -1.

The input to SVM also needs a kernel that does the transformation ϕ from input space to features space as $x \rightarrow \mathcal{H}$ using:

$$k(x, x') = (\phi(x), \phi(x'))_{\mathcal{H}}$$

Create a hyper-sphere that bounds the classes using SVM reformulated equation as:

$$\min_{R, \xi, c} R^2 + \frac{1}{vn} \sum_i \xi_i$$

Such that $\|\phi(x_i) - c\|^2 \leq R^2 + \xi_i$, $\xi_i \geq 0 \forall i \in [n]$

R is the radius of the hyper-sphere with center c and $v \in (0,1]$ represents an upper bound on the fraction of the data that are outliers.

As in normal SVM, we perform optimization using quadratic programming is done to obtain the solution as the decision boundary.

Advantages and limitations

- The key advantage to using one-class SVM—as is true of binary SVM—is the many theoretical guarantees in error and generalization bounds.
- High-dimensional data can be easily mapped in one-class SVM.
- Non-linear SVM with kernels can even find non-spherical shapes to bound the clusters of data.
- The training cost in space and memory increases as the size of the data increases.
- Parameter tuning, especially the kernel parameters and the cost parameter tuning with unlabeled data is a big challenge.

Outlier evaluation techniques

Measuring outliers in terms of labels, ranks, and scores is an area of active research. When the labels or the ground truth is known, the idea of evaluation becomes much easier as the outlier class is known and standard metrics can be employed. But when the ground truth is not known, the evaluation and validation methods are very subjective and there is no well-defined, rigorous statistical process.

Supervised evaluation

In cases where the ground truth is known, the evaluation of outlier algorithms is basically the task of finding the best thresholds for outlier scores (scoring-based outliers).

The balance between reducing the false positives and improving true positives is the key concept and Precision-Recall curves (described in [Chapter 2, Practical Approach to Real-World Supervised Learning](#)) are used to find the best optimum threshold. Confidence score, predictions, and actual labels are used in supervised learning to plot PRCurves, and instead of confidence scores, outlier scores are ranked and used here. ROC curves and area under curves are also used in many applications to evaluate thresholds. Comparing two or more algorithms and selection of the best can also be done using area under curve metrics when the ground truth is known.

Unsupervised evaluation

In most real-world cases, knowing the ground truth is very difficult, at least during the modeling task. Hawkins describes the evaluation method in this case at a very high level as "a sample containing outliers would show up such characteristics as large gaps between 'outlying' and 'inlying' observations and the deviation between outliers and the group of inliers, as measured on some suitably standardized scale".

The general technique used in evaluating outliers when the ground truth is not known is:

- **Histogram of outlier scores:** A visualization-based method, where outlier scores are grouped into predefined bins and users can select thresholds based on outlier counts, scores, and thresholds.
- **Score normalization and distance functions:** In this technique, some form of

normalization is done to make sure all outlier algorithms that produce scores have the same ranges. Some form of distance or similarity or correlation based method is used to find commonality of outliers across different algorithms. The general intuition here is: the more the algorithms that weigh the data point as outlier, the higher the probability of that point actually being an outlier.

Real-world case study

Here we present a case study that illustrates how to apply clustering and outlier techniques described in this chapter in the real world, using open-source Java frameworks and a well-known image dataset.

Tools and software

We will now introduce two new tools that were used in the experiments for this chapter: SMILE and Elki. SMILE features a Java API that was used to illustrate feature reduction using PCA, Random Projection, and IsoMap. Subsequently, the graphical interface of Elki was used to perform unsupervised learning—specifically, clustering and outlier detection. Elki comes with a rich set of algorithms for cluster analysis and outlier detection including a large number of model evaluators to choose from.

Note

Find out more about SMILE at: <http://haifengl.github.io/smile/> and to learn more about Elki, visit <http://elki.dbs.ifi.lmu.de/>.

Business problem

Character-recognition is a problem that occurs in many business areas, for example, the translation of medical reports and hospital charts, postal code recognition in the postal service, check deposit service in retail banking, and others. Human handwriting can vary widely among individuals. Here, we are looking exclusively at handwritten digits, 0 to 9. The problem is made interesting due to the verisimilitude within certain sets of digits, such as 1/2/7 and 6/9/0. In our experiments in this chapter we use clustering and outlier analysis using several different algorithms to illustrate the relative strengths and weaknesses of the methods. Given the widespread use of these techniques in data mining applications, our main focus is to gain insights into the data and the algorithms and evaluation measures; we do not apply the models for prediction on test data.

Machine learning mapping

As suggested by the title of the chapter, our experiments aim to demonstrate Unsupervised Learning by ignoring the labels identifying the digits in the dataset. Having learned from the dataset, clustering and outlier analyses can yield invaluable information for describing patterns in the data, and are often used to explore these patterns and inter-relationships in the data, and not just to predict the class of unseen data. In the experiments described here, we are concerned with description and exploration rather than prediction. Labels are used when available by external evaluation measures, as they are in these experiments as well.

Data collection

This is already done for us. For details on how the data was collected, see: The MNIST database: <http://yann.lecun.com/exdb/mnist/>.

Data quality analysis

Each feature in a data point is the greyscale value of one of 784 pixels. Consequently, the type of all features is numeric; there are no categorical types except for the class attribute, which is a numeral in the range 0-9. Moreover, there are no missing data elements in the dataset. Here is a table with some basic statistics for a few pixels. The images are pre-centred in the 28 x 28 box so in most examples, the data along the borders of the box are zeros:

Feature	Average	Std Dev	Min	Max
pixel300	94.25883	109.117	0	255
pixel301	72.778	103.0266	0	255
pixel302	49.06167	90.68359	0	255
pixel303	28.0685	70.38963	0	255
pixel304	12.84683	49.01016	0	255
pixel305	4.0885	27.21033	0	255
pixel306	1.147	14.44462	0	254
pixel307	0.201667	6.225763	0	254
pixel308	0	0	0	0
pixel309	0.009167	0.710047	0	55
pixel310	0.102667	4.060198	0	237

Table 1: Summary of features from the original dataset before pre-processing

The **Mixed National Institute of Standards and Technology (MNIST)** dataset is a widely used dataset for evaluating unsupervised learning methods. The MNIST dataset is mainly chosen because the clusters in high dimensional data are not well separated.

The original MNIST dataset had black and white images from NIST. They were normalized to fit in a 20 x 20 pixel box while maintaining the aspect ratio. The images were centered in a 28 x 28 image by computing the center of mass and translating it to position it at the center of the 28 x 28 dimension grid.

Each pixel is in a range from 0 to 255 based on the intensity. The 784 pixel values are flattened out and become a high dimensional feature set for each image. The following figure depicts a sample digit 3 from the data, with mapping to the grid where each pixel has an integer value from 0 to 255.

The experiments described in this section are intended to show the application of unsupervised learning techniques to a well-known dataset. As was done in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning* with supervised learning techniques, multiple experiments were carried out using several clustering and outlier methods. Results from experiments with and without feature reduction are presented for each of the selected methods followed by an analysis of the results.

Data sampling and transformation

Since our focus is on exploring the dataset using various unsupervised techniques and not on the predictive aspect, we are not concerned with train, validation, and test samples here. Instead, we use the entire dataset to train the models to perform clustering analysis.

In the case of outlier detection, we create a reduced sample of only two classes of data, namely, 1 and 7. The choice of a dataset with two similarly shaped digits was made in order to set up a problem space in which the discriminating power of the various anomaly detection techniques would stand out in greater relief.

Feature analysis and dimensionality reduction

We demonstrate different feature analysis and dimensionality reduction methods—PCA, Random Projection, and IsoMap—using the Java API of the SMILE machine learning toolkit.

Figure 8: Showing digit 3 with pixel values distributed in a 28 by 28 matrix ranging from 0 to 254.

The code for loading the dataset and reading the values is given here along with inline comments:

```
//parser to parse the tab delimited file
```

```

DelimitedTextParser parser = new
DelimitedTextParser();parser.setDelimiter("[\t]+");
//parse the file from the location
parser.parse("mnistData", new File(fileLocation);
//the header data file has column names to map
parser.setColumnNames(true);
//the class attribute or the response variable index
AttributeDataSet dataset = parser.setResponseIndex(new
NominalAttribute("class"), 784);

//convert the data into two-dimensional array for using various
techniques
double[][] data = dataset.toArray(new double[dataset.size()][]);

```

PCA

The following snippet illustrates dimensionality reduction achieved using the API for PCA support:

```

//perform PCA with double data and using covariance //matrix
PCA pca = new PCA(data, true);
//set the projection dimension as two (for plotting here)
pca.setProjection(2);
//get the new projected data in the dimension
double[][] y = pca.project(data);

```

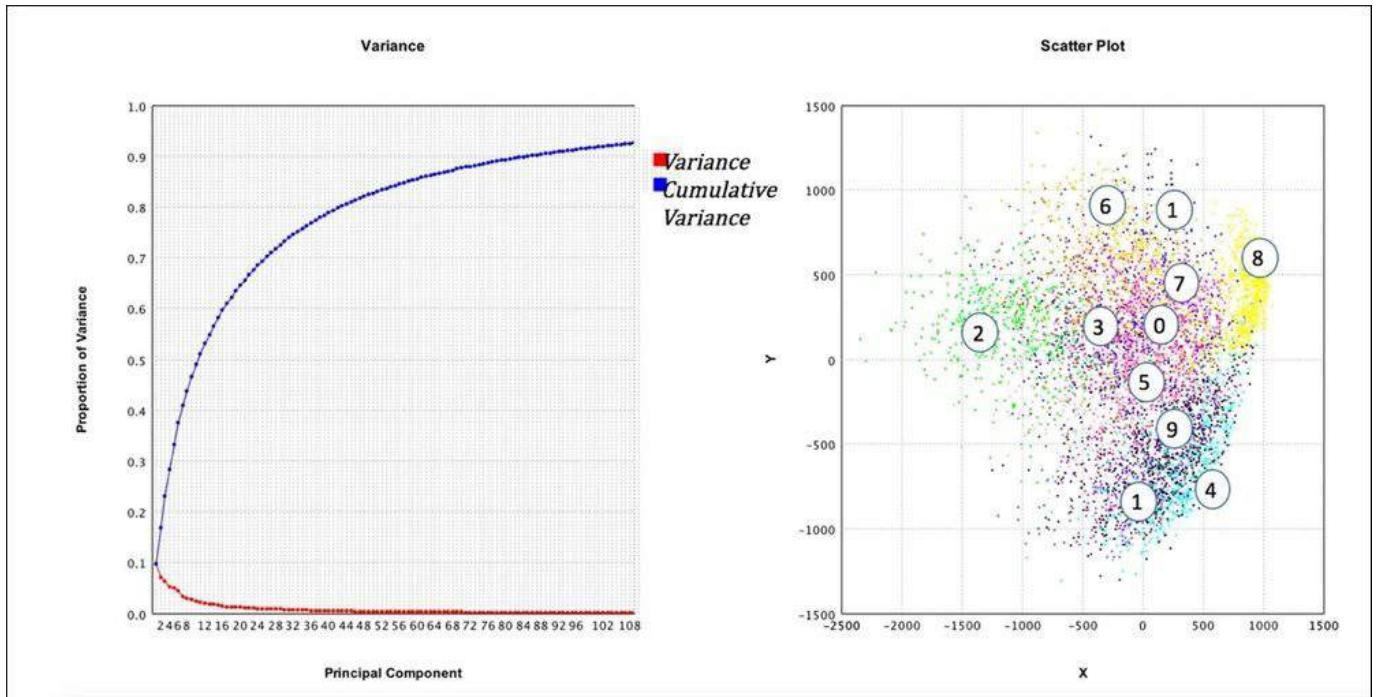


Figure 9: PCA on MNIST – On the left, we see that over 90 percent of variance in

data is accounted for by fewer than half the original number of features; on the right, a representation of the data using the first two principal components.

Table 2: Summary of set of 11 random features after PCA

The PCA computation reduces the number of features to 274. In the following table you can see basic statistics for a randomly selected set of features. Feature data has been normalized as part of the PCA:

Features	Average	Std Dev	Min	Max
1	0	2.982922	-35.0821	19.73339
2	0	2.415088	-32.6218	31.63361
3	0	2.165878	-21.4073	16.50271
4	0	1.78834	-27.537	31.52653
5	0	1.652688	-21.4661	22.62837
6	0	1.231167	-15.157	10.19708
7	0	0.861705	-6.04737	7.220233
8	0	0.631403	-6.80167	3.633182
9	0	0.606252	-5.46206	4.118598
10	0	0.578355	-4.21456	3.621186
11	0	0.528816	-3.48564	3.896156

Table 2: Summary of set of 11 random features after PCA

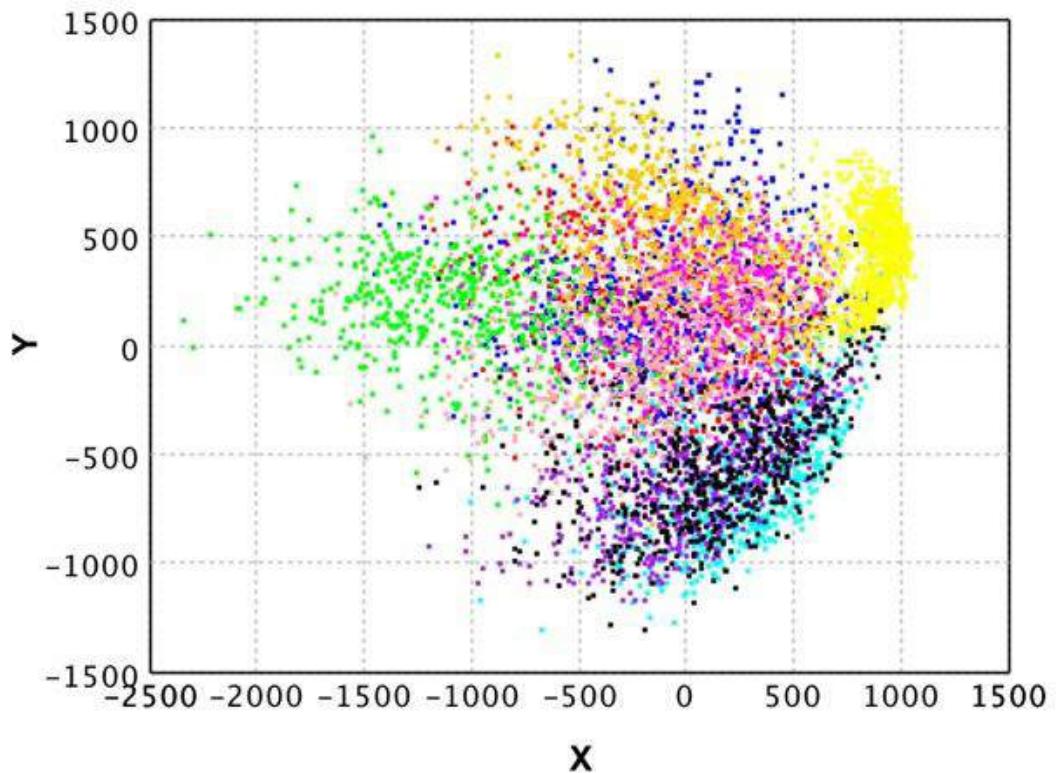
Random projections

Here, we illustrate the straightforward usage of the API for performing data

transformation using random projection:

```
//random projection done on the data with projection in //2 dimension  
RandomProjection rp = new RandomProjection(data.length, 2, false);  
//get the transformed data for plotting  
double[][] projectedData = rp.project(data);
```

PCA



Random Projection

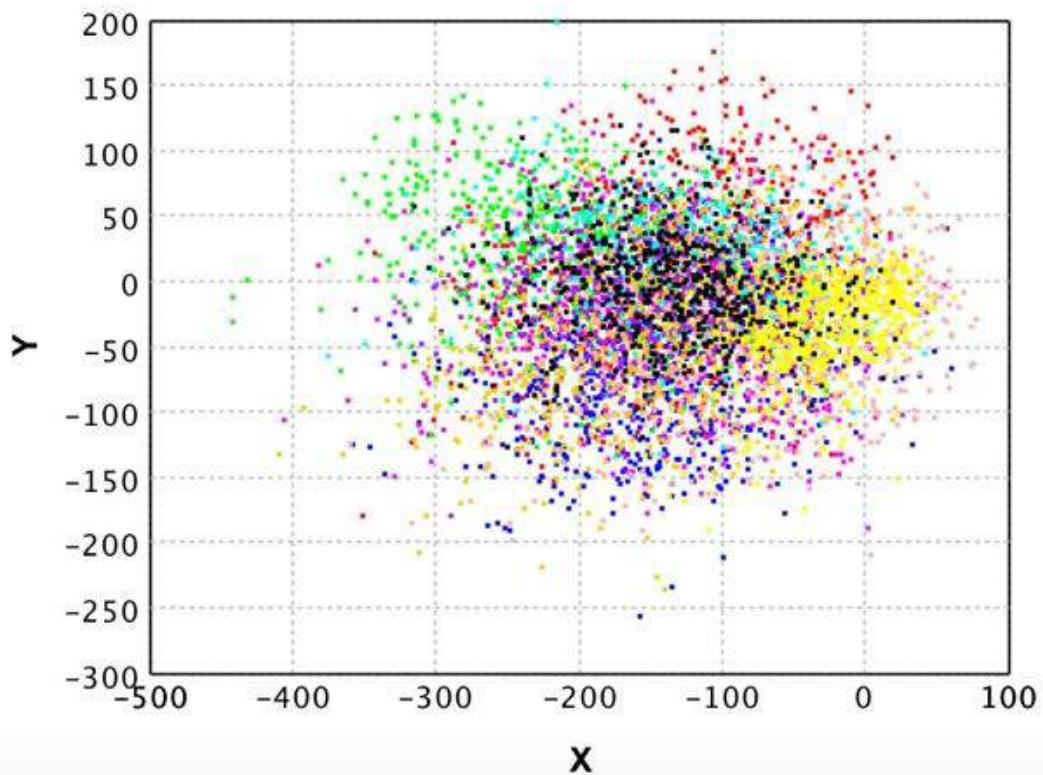


Figure 10: PCA and Random projection - representations in two dimensions using Smile API

ISOMAP

This code snippet illustrates use of the API for Isomap transformation:

```
//perform isomap transformation of data, here in 2 //dimensions with  
k=10  
IsoMap isomap = new IsoMap(data, 2, 10);  
//get the transformed data back  
double[][] y = isomap.getCoordinates();
```

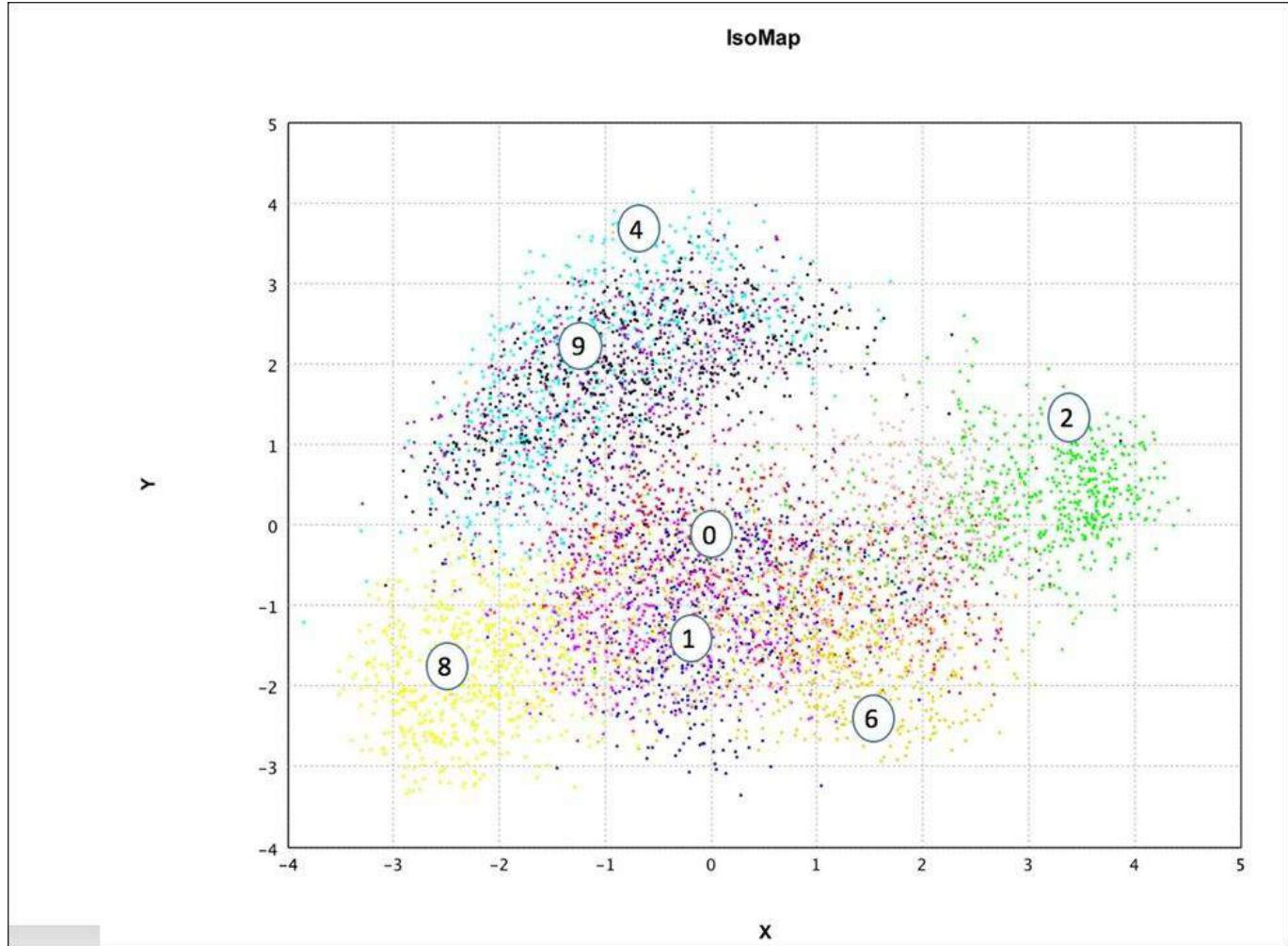


Figure 11: IsoMap – representation in two dimensions with k = 10 using Smile API

Observations on feature analysis and dimensionality reduction

We can make the following observations from the results shown in the plots:

- The PCA variance and number of dimensions plot clearly shows that around 100 linearly combined features have a similar representation or variance in the data ($> 95\%$) as that of the 784 original features. This is the key first step in any unsupervised feature reduction analysis.
- Even PCA with two dimensions and not 100 as described previously shows some really good insights in the scatterplot visualization. Clearly, digits 2, 8, and 4 are very well separated from each other and that makes sense as they are written quite distinctly from each other. Digits such as {1,7}, {3,0,5}, and {1,9} in the low dimensional space are either overlapping or tightly clustered. This shows that with just two features it is not possible to discriminate effectively. It also shows that there is overlap in the characteristics or features amongst these classes.
- The next plot comparing PCA with Random Projections, both done in lower dimension of 2, shows that there is much in common between the outputs. Both have similar separation for distinct classes as described in PCA previously. It is interesting to note that PCA does much better in separating digits {8,9,4}, for example, than Random Projections.
- Isomap, the next plot, shows good discrimination, similar to PCA. Subjectively, it seems to be separating the data better than Random Projections. Visually, for instance, {3,0,5} is better separated out in Isomap than PCA.

Clustering models, results, and evaluation

Two sets of experiments were conducted using the MNIST-6000 dataset. The dataset consists of 6,000 examples, each of which represents a hand-written digit as greyscale values of a 28 x 28 square of pixels.

First, we run some clustering techniques to identify the 10 clusters of digits. For the experiments in this part of the case study, we use the software Elki.

In the first set of experiments, there is no feature-reduction involved. All 28x28 pixels are used. Clustering techniques including k-Means, EM (Diagonal Gaussian Model Factory), DBSCAN, Hierarchical (HDBSCAN Hierarchy Extraction), as well as Affinity Propagation were used. In each case, we use metrics from two internal evaluators: Davies Bouldin and Silhouette, and several external evaluators: Precision, Recall, F1 measure, and Rand Index.

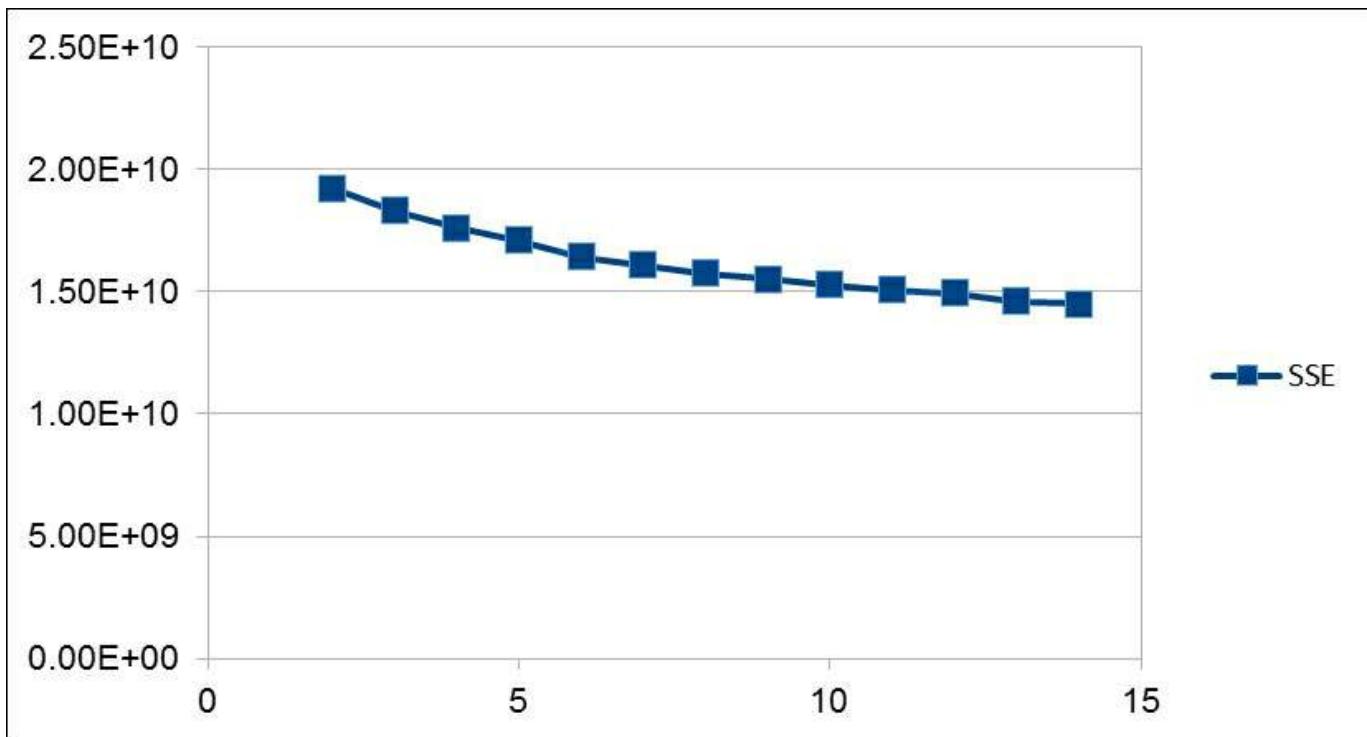


Figure 12: K-Means – using Sum of Squared Errors (SSE) to find optimal k, the number of clusters. An elbow in the curve, which is typically used to pick the optimal k value, is not particularly detectable in the plot.

In the case of k-Means, we did several runs using a range of k values. The plot shows that the Sum of Squared Errors (SSE) metric decreases with k.

The table shows results for $k=10$ and ranks for each are in parentheses:

Algorithm	Silhouette	Davies-Bouldin Index	Precision	Recall	F1	Rand
K-Means Lloyd	+0.09 0.0737 (1)	2.8489 (3)	0.4463 (3)	0.47843 (3)	0.4618 (1)	0.8881 (3)
EM (Diagonal Gaussian Model Factory)	NaN	0 (1)	0.1002 (6)	1 (1)	0.1822 (4)	0.1003 (5)
DBSCAN	0 (4)	0 (1)	0.1003 (5)	1 (1)	0.1823 (3)	0.1003 (5)
Hierarchical (HDBSCAN Hierarchy Extraction)	+0.05 0.0435 (3)	2.7294	0.1632 (4)	0.9151 (2)	0.2770 (2)	0.5211 (4)
Hierarchical (Simplified Hierarchy Extraction)	NaN	0 (1)	1 (1)	0.0017 (5)	0.0033 (6)	0.8999 (2)
Affinity Propagation	+0.07 0.04690 (2)	1.7872 (2)	0.8279 (2)	0.0281 (4)	0.0543 (5)	0.9019 (1)

Table 3. Evaluation of clustering algorithms for MNIST data

In the second clustering experiment, the dataset was first pre-processed using PCA, and the resulting data with 273 features per example was used with the same algorithms as in the first experiment. The results are shown in the table:

Algorithm	Silhouette	Davies-Bouldin Index	Precision	Recall	F1	Rand
K-Means Lloyd	+0.14 0.0119	3.1830	0.3456	0.4418	0.3878 (1)	0.8601
EM (Diagonal Gaussian Model Factory)	+0.16 -0.0402	3.5429	0.1808	0.3670	0.2422	0.7697
DBSCAN	+0.13 -0.0351	1.3236	0.1078	0.9395 (1)	0.1934	0.2143
Hierarchical (HDBSCAN Hierarchy Extraction)	+0.05 0.7920					

Extraction)	(1)	0.0968	0.1003	0.9996	0.1823	0.1005
Affinity Propagation	+0.09 0.0575	1.6296	0.6130 (1)	0.0311	0.0592	0.9009 (1)
Subspace (DOC)	+0.00 0.0	0 (1)	0.1003	1	0.1823	0.1003

Table 4. Evaluation of clustering algorithms for MNIST data after PCA

Observations and clustering analysis

As shown in tables 2.1 and 2.2, different algorithms discussed in the sections on clustering are compared using different evaluation measures.

Generally, comparing different internal and external measures based on technical, domain and business requirements is very important. When labels or outcomes are available in the dataset, using external measures becomes an easier choice. When labeled data is not available, the norm is to use internal measures with some ranking for each and looking at comparative ranking across all measures. The important and often interesting observations are made at this stage:

- Evaluating the performance of k-Means with varying k , (shown in the figure) using a measure such as Sum of Squared Errors, is the basic step to see "optimality" of number of clusters. The figure clearly shows that as k increases the score improves as cluster separation improves.
- When we analyze Table 2.1 where all 784 features were used and all evaluation measures for the different algorithms are shown, some key things stand out:
 - k-Means and Affinity Propagation both show a large overlap in the Silhouette index in terms of standard deviation and average, respectively (k-Means +0.09 0.0737; Affinity Propagation +0.07 0.04690). Hence it is difficult to analyze them on this metric.
 - In the measures such as DB Index (minimal is good), Rand Index (closer to 1 is good), we see that Affinity Propagation and Hierarchical Clustering show very good results.
 - In the measures where the labels are taken into account, Hierarchical Clustering, DBSCAN, and EM has either high Precision or high Recall and consequently, the F1 measure is low. k-Means gives the highest F1 measure when precision and recall are taken into consideration.
- In Table 2.2 where the dataset with 273 features—reduced using PCA with 95%

variance retained—is run through the same algorithms and evaluated by the same measures, we make the following interesting observations:

By reducing the features there is a negative impact on every measure for certain algorithms; for example, all the measures of k-Means degrade. An algorithm such as Affinity Propagation has a very low impact and in some cases even a positive impact when using reduced features. When compared to the results where all the features were used, AP shows similar Rand Index and F1, better Recall, DB Index and Silhouette measures, and small changes in Precision, demonstrating clear robustness.

Hierarchical Clustering shows similar results as before in terms of better DB index and Rand Index, and scores close to AP in Rand Index.

Outlier models, results, and evaluation

For the outlier detection techniques, we used a subset of the original dataset containing all examples of digit 1 and an under-sampled subset of digit 7 examples. The idea is that the similarity in shape of the two digits would cause the digit 7 examples to be found to be outliers.

The models used were selected from Angular, Distance-based, clustering, LOF, and One-Class SVM.

The outlier metrics used in the evaluation were ROC AUC, Average Precision, R-Precision, and Maximum F1 measure.

The following table shows the results obtained, with ranks in parentheses:

Algorithm	ROC AUC	Avg. Precision	R-Precision	Maximum F1
Angular (ABOD)	0.9515 (3)	0.1908 (4)	0.24 (4)	0.3298 (4)
Distance-based (KNN Outlier)	0.9863 (1)	0.4312 (3)	0.4533 (3)	0.4545 (3)
Distance Based (Local Isolation Coefficient)	0.9863 (1)	0.4312 (3)	0.4533 (3)	0.4545 (3)
Clustering (EM Outlier)	0.5 (5)	0.97823827 (1)	0.989 (1)	0.9945 (1)
LOF	0.4577 (6)	0.0499 (6)	0.08 (6)	0.0934 (6)
LOF (ALOKI)	0.5 (5)	0.0110 (7)	0.0110 (7)	0.0218 (7)
LOF (COF)	0.4577 (6)	0.0499 (6)	0.08 (6)	0.0934 (6)
One-Class SVM (RBF)	0.9820 (2)	0.5637 (2)	0.5333 (2)	0.5697 (2)
One-Class SVM (Linear)	0.8298 (4)	0.1137 (5)	0.16 (5)	0.1770 (5)

Table 5 Evaluation measures of Outlier analysis algorithms

Observations and analysis

In the same way as we evaluated different clustering methods, we used several observations to compare a number of outlier algorithms. Once again, the right methodology is to judge an algorithm based on ranking across all the metrics and then getting a sense of how it does across the board as compared to other algorithms. The outlier metrics used here are all standard external measures used to compare outlier algorithms:

- It is interesting to see with the right parameters, that is, $k=2$, EM can find the right distribution and find outliers more efficiently than most. It ranks very high and is first among the important metrics that include Maximum F1, R-Precision, and Avg. Precision.
- 1-Class SVM with non-linear RBF Kernel does consistently well across most measures, that is, ranks second best in ROC area, R-Precision and Avg. Precision, and Maximum F1. The difference between Linear SVM, which ranks about fifth in most rankings and 1-Class SVM, which ranks second shows that the problem is indeed nonlinear in nature. Generally, when the dimensions are high (784), and outliers are nonlinear and rare, 1-Class SVM with kernels do really well.
- Local outlier-based techniques (LOF and its variants) are consistently ranked lower in almost all the measures. This gives the insight that the outlier problem may not be local, but rather global. Distance-based algorithms (KNN and Local Isolation) perform the best in ROC area under the curve and better than local outlier-based, even though using distance-based metrics gives the insight that the problem is indeed global and suited for distance-based measures.

Table 1: Summary of features from the original dataset before pre-processing

Summary

Both supervised and unsupervised learning methods share common concerns with respect to noisy data, high dimensionality, and demands on memory and time as the size of data grows. Other issues peculiar to unsupervised learning, due to the lack of ground truth, are questions relating to subjectivity in the evaluation of models and their interpretability, effect of cluster boundaries, and so on.

Feature reduction is an important preprocessing step that mitigates the scalability problem, in addition to presenting other advantages. Linear methods such as PCA, Random Projection, and MDS, each have specific benefits and limitations, and we must be aware of the assumptions inherent in each. Nonlinear feature reduction methods include KPCA and Manifold learning.

Among clustering algorithms, k-Means is a centroid-based technique initialized by selecting the number of clusters and it is sensitive to the initial choice of centroids. DBSCAN is one of the density-based algorithms that does not need initializing with number of clusters and is robust against noise and outliers. Among the probabilistic-based techniques are Mean Shift, which is deterministic and robust to noise, and EM/GMM, which performs well with all types of features. Both Mean Shift and EM/GMM tend to have scalability problems.

Hierarchical clustering is a powerful method involving building binary trees that iteratively groups data points until a similarity threshold is reached. Tolerance to noise depends on the similarity metric used. SOM is a two-layer neural network, allowing visualization of clusters in a 2-D grid. Spectral clustering treats the dataset as a connected graph and identifies clusters by graph partitioning. Affinity propagation, another graph-based technique, uses message passing between data points as affinities to detect clusters.

The validity and usefulness of clustering algorithms is demonstrated using various validation and evaluation measures. Internal measures have no access to ground truth; when labels are available, external measures can be used. Examples of internal measures are Silhouette index and Davies-Bouldin index. Rand index and F-measure are external evaluation measures.

Outlier and anomaly detection is an important area of unsupervised learning. Techniques are categorized as Statistical-based, Distance-based, Density-based,

Clustering-based, High-dimensional-based, and One Class SVM. Outlier evaluation techniques include supervised evaluation, where ground truth is known, and unsupervised evaluation, when ground truth is not known.

Experiments using the SMILE Java API and Elki toolkit illustrate the use of the various clustering and outlier detection techniques on the MNIST6000 handwritten digits dataset. Results from different evaluation techniques are presented and compared.

References

1. K. Pearson (1901). *On lines and planes of closest fit to systems of points in space*. Philosophical Magazine, 2:559–572.
2. A. D. Back (1997). "A first application of independent component analysis to extracting structure from stock returns," Neural Systems, vol. 8, no. 4, pp. 473–484.
3. Tipping ME, Bishop CM (1999). *Probabilistic principal component analysis*. Journal of the Royal Statistical Society, Series B, 61(3):611–622.
10.1111/1467-9868.00196
4. Sanjoy Dasgupta (2000). *Experiments with random projection*. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence (UAI'00), Craig Boutilier and Moisés Goldszmidt (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 143-151.
5. T. Cox and M. Cox (2001). *Multidimensional Scaling*. Chapman Hall, Boca Raton, 2nd edition.
6. Bernhard Schoelkopf, Alexander J. Smola, and Klaus-Robert Mueller (1999). *Kernel principal component analysis*. In Advances in kernel methods, MIT Press, Cambridge, MA, USA 327-352.
7. Tenenbaum, J.B.; De Silva, V.; & Langford, J.C (2000). *A global geometric framework for nonlinear dimensionality reduction*. Science. Vol. 290, Issue 5500, pp. 2319-2323
8. M. Belkin and P. Niyogi (2003). *Laplacian eigenmaps for dimensionality reduction and data representation*. Neural Computation, 15(6):1373–1396.
9. S. Roweis and L. Saul (2000). *Nonlinear dimensionality reduction by locally linear embedding*. Science, 290:2323–2326.
10. Hartigan, J. and Wong, M (1979). *Algorithm AS136: A k-means clustering algorithm*. Applied Statistics, 28, 100-108.
11. Dorin Comaniciu and Peter Meer (2002). *Mean Shift: A robust approach toward feature space analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 603-619.
12. Hierarchical Clustering Jain, A. and Dubes, R (1988). *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, NJ.
13. McLachlan, G. and Basford, K (1988). *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker, New York, NY
14. Ester, M., Kriegel, H-P., Sander, J. and Xu, X (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise*. In

- Proceedings of the 2nd ACM SIGKDD, 226-231, Portland, Oregon.
- 15. Y. Ng, M. I. Jordan, and Y. Weiss (2001). *On spectral clustering: Analysis and an algorithm*, in Advances in Neural Information Processing Systems. MIT Press, pp. 849–856.
 - 16. Delbert Dueck and Brendan J. Frey (2007). *Non-metric affinity propagation for unsupervised image categorization*. In IEEE Int. Conf. Computer Vision (ICCV), pages 1–8.
 - 17. Teuvo Kohonen (2001). *Self-Organizing Map*. Springer, Berlin, Heidelberg. 1995.Third, Extended Edition.
 - 18. M. Halkidi, Y. Batistakis, and M. Vazirgiannis (2001). *On clustering validation techniques*, J. Intell. Inf. Syst., vol. 17, pp. 107–145.
 - 19. M. Markou, S. Singh (2003). *Novelty detection: a review – part 1: statistical approaches*, Signal Process. 83 (12) 2481–2497
 - 20. Byers, S. D. AND Raftery, A. E (1998). *Nearest neighbor clutter removal for estimating features in spatial point processes*. J. Amer. Statis. Assoc. 93, 577–584.
 - 21. Breunig, M. M., Kriegel, H.-P., Ng, R. T., AND Sander, J (1999). *Optics-of: Identifying local outliers*. In Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery. Springer-Verlag, 262–270.
 - 22. Brito, M. R., Chavez, E. L., Quiroz, A. J., AND yukich, J. E (1997). *Connectivity of the mutual k-nearest neighbor graph in clustering and outlier detection*. Statis. Prob. Lett. 35, 1, 33–42.
 - 23. Aggarwal C and Yu P S (2000). *Outlier detection for high dimensional data*. In Proc ACM SIGMOD International Conference on Management of Data (SIGMOD), Dallas, TX.
 - 24. Ghoting, A., Parthasarathy, S., and Otey, M (2006). *Fast mining of distance-based outliers in high dimensional spaces* In Proceedings SIAM Int Conf on Data Mining (SDM) Bethesda ML dimensional spaces. In Proc. SIAM Int. Conf. on Data Mining (SDM), Bethesda, ML.
 - 25. Kriegel, H.-P., Schubert, M., and Zimek, A (2008). *Angle-based outlier detection*, In Proceedings ACM SIGKDD Int. Conf on Knowledge Discovery and Data Mining (SIGKDD) Las Vegas NV Conf. on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV.
 - 26. Schoelkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., AND Williamson, R. C (2001). *Estimating the support of a high-dimensional distribution*. Neural Comput. 13, 7, 1443–1471.
 - 27. F Pedregosa, et al. *Scikit-learn: Machine learning in Python*. Journal of

Machine Learning Research, 2825-2830.

Chapter 4. Semi-Supervised and Active Learning

In [Chapter 2, Practical Approach to Real-World Supervised Learning](#) and [Chapter 3, Unsupervised Machine Learning Techniques](#), we discussed two major groups of machine learning techniques which apply to opposite situations when it comes to the availability of labeled data—one where all target values are known and the other where none are. In contrast, the techniques in this chapter address the situation when we must analyze and learn from data that is a mix of a small portion with labels and a large number of unlabeled instances.

In speech and image recognition, a vast quantity of data is available, and in various forms. However, the cost of labeling or classifying all that data is costly and therefore, in practice, the proportion of speech or images that are classified to those that are not classified is very small. Similarly, in web text or document classification, there are an enormous number of documents on the World Wide Web but classifying them based on either topics or contexts requires domain experts—this makes the process complex and expensive. In this chapter, we will discuss two broad topics that cover the area of "learning from unlabeled data", namely **Semi-Supervised Learning (SSL)** and Active Learning. We will introduce each of the topics and discuss the taxonomy and algorithms associated with each as we did in previous chapters. Since the book emphasizes the practical approach, we will discuss tools and libraries available for each type of learning. We will then consider real-world case studies and demonstrate the techniques that are useful when applying the tools in practical situations.

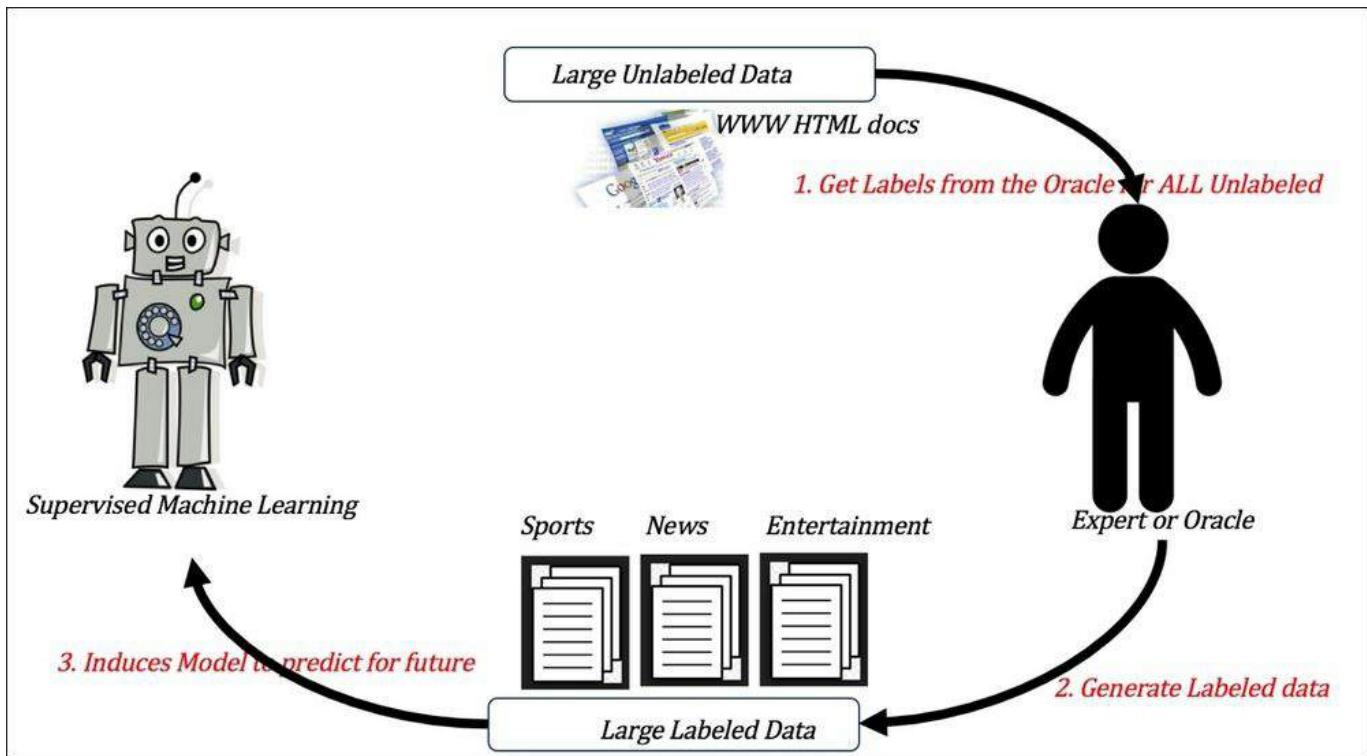
Here is the list of topics that are covered in this chapter:

- Semi-Supervised Learning:
 - Representation, notation, and assumptions
 - Semi-Supervised Learning techniques:
 - Self-training SSL
 - Co-training SSL
 - Cluster and label SSL
 - Transductive graph label propagation
 - Transductive SVM
 - Case study in Semi-Supervised Learning

- Active Learning:
 - Representation and notation
 - Active Learning scenarios
 - Active Learning approaches:
 - Uncertainty sampling
 - Least confident sampling
 - Smallest margin sampling
 - Label entropy sampling
 - Version space sampling:
 - Query by disagreement
 - Query by committee
 - Data distribution sampling:
 - Expected model change
 - Expected error reduction
 - Variance reduction
 - Density weighted methods
 - Case study in Active Learning

Semi-supervised learning

The idea behind semi-supervised learning is to learn from labeled and unlabeled data to improve the predictive power of the models. The notion is explained with a simple illustration, *Figure 1*, which shows that when a large amount of unlabeled data is available, for example, HTML documents on the web, the expert can classify a few of them into known categories such as sports, news, entertainment, and so on. This small set of labeled data together with the large unlabeled dataset can then be used by semi-supervised learning techniques to learn models. Thus, using the knowledge of both labeled and unlabeled data, the model can classify unseen documents in the future. In contrast, supervised learning uses labeled data only:



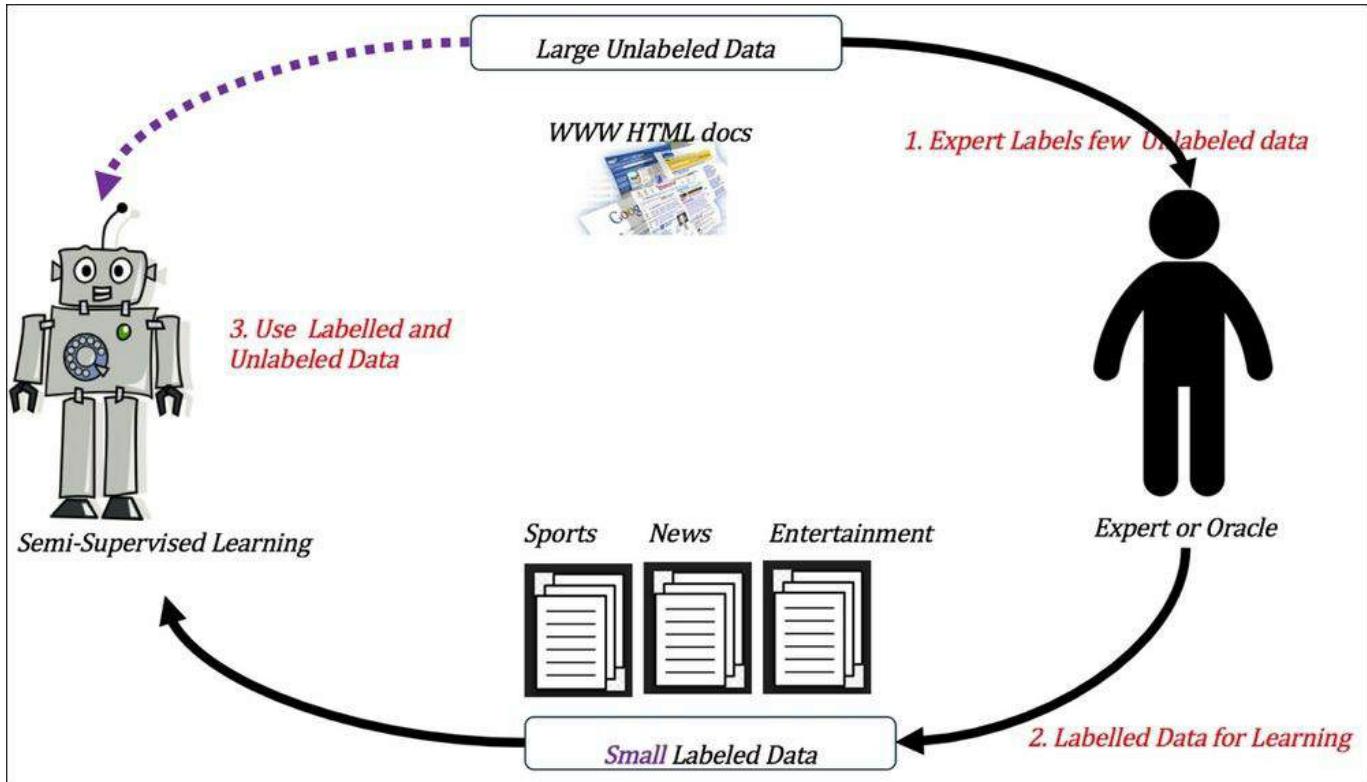


Figure 1. Semi-Supervised Learning process (bottom) contrasted with Supervised Learning (top) using classification of web documents as an example. The main difference is the amount of labeled data available for learning, highlighted by the qualifier "small" in the semi-supervised case.

Representation, notation, and assumptions

As before, we will introduce the notation we use in this chapter. The dataset D consists of individual data instances represented as \mathbf{x} , which is also represented as a set $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the set of data instances without labels. The labels associated with these data instances are $\{y_1, y_2, \dots, y_n\}$. The entire labeled dataset can be represented as paired elements in a set, as given by $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$. In semi-supervised learning, we divide the dataset D further into two sets U and L for unlabeled and labeled data respectively.

The labeled data $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ consists of all labeled data with known outcomes $\{y_1, y_2, \dots, y_l\}$. The unlabeled data $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ is the dataset where the outcomes are not known. $|U| > |L|$.

Inductive semi-supervised learning consists of a set of techniques, which, given the training set D with labeled data $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ and unlabeled data $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$, learns a model represented as $f : \mathbf{x} \rightarrow y$ so that the model f can be a good predictor on unseen data beyond the training unlabeled data U . It "induces" a model that can be used just like supervised learning algorithms to predict on unseen instances.

Transductive semi-supervised learning consists of a set of techniques, which, given the training set D , learns a model $f : \mathbf{x}_{l+1}^n \rightarrow y_{l+1}^n$ that makes predictions on unlabeled data alone. It is not required to perform on unseen future instances and hence is a simpler form of SSL than inductive based learning.

Some of the assumptions made in the semi-supervised learning algorithms that should hold true for these types of learning to be successful are noted in the following list. For SSL to work, one or more of these assumptions must be true:

- **Semi-supervised smoothness:** In simple terms, if two points are "close" in terms of density or distance, then their labels agree. Conversely, if two points are separated and in different density regions, then their labels need not agree.
- **Cluster togetherness:** If the data instances of classes tend to form a cluster, then the unlabeled data can aid the clustering algorithm to find better clusters.
- **Manifold togetherness:** In many real-world datasets, the high-dimensional data

lies in a low-dimensional manifold, enabling learning algorithms to overcome the curse of dimensionality. If this is true in the given dataset, the unlabeled data also maps to the manifold and can improve the learning.

Semi-supervised learning techniques

In this section, we will describe different SSL techniques, and some accompanying algorithms. We will use the same structure as in previous chapters and describe each method in three subsections: *Inputs and outputs*, *How does it work?*, and *Advantages and limitations*.

Self-training SSL

Self-training is the simplest form of SSL, where we perform a simple iterative process of imputing the data from the unlabeled set by applying the model learned from the labeled set (*References [1]*):

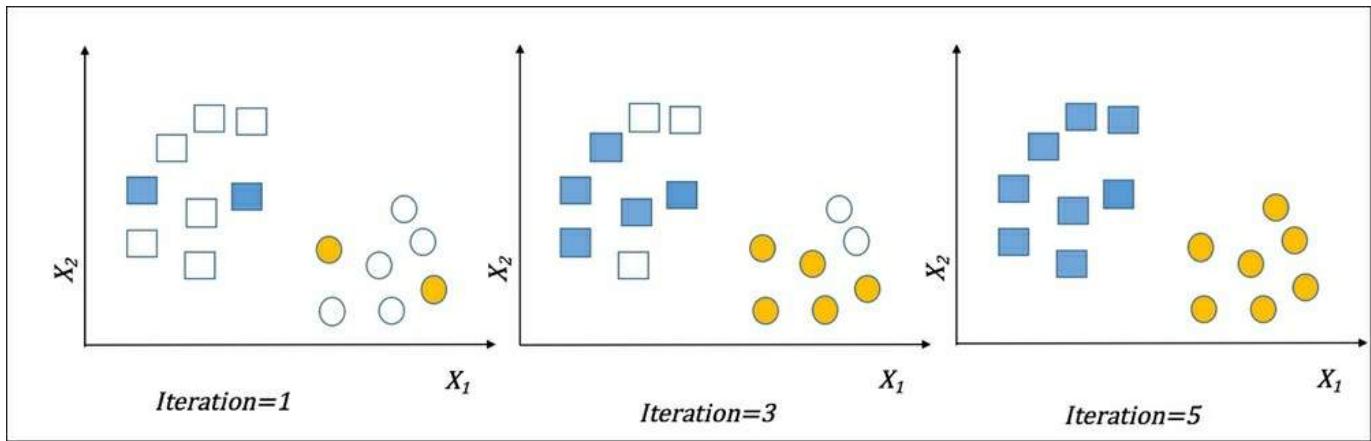


Figure 2. Self-training SSL in binary classification with some labeled data shown with blue rectangles and yellow circles. After various iterations, the unlabeled data gets mapped to the respective classes.

Inputs and outputs

The input is training data with a small amount of labeled and a large amount of unlabeled data. A base classifier, either linear or non-linear, such as Naïve Bayes, KNN, Decision Tree, or other, is provided along with the hyper-parameters needed for each of the algorithms. The constraints on data types will be similar to the base learner. Stopping conditions such as *maximum iterations reached* or *unlabeled data exhausted* are choices that must be made as well. Often, we use base learners which give probabilities or ranks to the outputs. As output, this technique generates models that can be used for performing predictions on unseen datasets other than the

unlabeled data provided.

How does it work?

The entire algorithm can be summarized as follows:

1. While stopping criteria not reached:
 1. Train the classifier model $f : \mathbf{x} \rightarrow y$ with labeled data L
 2. Apply the classifier model f on unlabeled data U
 3. Choose k most confident predictions from U as set L_u
 4. Augment the labeled data with the k data points $L = L \cup L_u$
2. Repeat all the steps under 2.

In the abstract, self-training can be seen as an expectation maximization process applied to a semi-supervised setting. The process of training the classifier model is finding the parameter θ using MLE or MAP. Computing the labels using the learned

model is similar to the *EXPECTATION* step where $\mathbb{E}[y_i]$ is estimating the label from U given the parameter θ . The iterative next step of learning the model with augmented labels is akin to the *MAXIMIZATION* step where the new parameter is tuned to θ' .

Advantages and limitations

The advantages and limitations are as follows:

- Simple, works with most supervised learning techniques.
- Outliers and noise can cause mistakes in predictions to be reinforced and the technique to degrade.

Co-training SSL or multi-view SSL

Co-training based SSL involves learning from two different "views" of the same data. It is a special case of multi-view SS (References [2]). Each view can be considered as a feature set of the point capturing some domain knowledge and is orthogonal to the other view. For example, a web documents dataset can be considered to have two views: one view is features representing the text and the other view is features representing hyperlinks to other documents. The assumption is that there is enough data for each view and learning from each view improves the overall labeling process. In datasets where such partitions of features are not

possible, splitting features randomly into disjoint sets forms the views.

Inputs and outputs

Input is training data with a few labeled and a large number of unlabeled data. In addition to providing the data points, there are feature sets corresponding to each view and the assumption is that these feature sets are not overlapping and solve different classification problems. A base classifier, linear or non-linear, such as Naïve Bayes, KNN, Decision Tree, or any other, is selected along with the hyper-parameters needed for each of the algorithms. As output, this method generates models that can be used for performing predictions on unseen datasets other than the unlabeled data provided.

How does it work?

We will demonstrate the algorithm using two views of the data:

1. Initialize the data as $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ labeled and $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ unlabeled. Each data point has two views $\mathbf{x} = [\mathbf{x}^1, \mathbf{x}^2]$ and $L = [L^1, L^2]$.
2. While stopping criteria not reached:
 1. Train the classifier models $f^1 : \mathbf{x} \rightarrow y$ and $f^2 : \mathbf{x} \rightarrow y$ with labeled data L^1 and L^2 respectively.
 2. Apply the classifier models f^1 and f^2 on unlabeled data U using their own features.
 3. Choose k the most confident predictions from U , applying f^1 and f^2 as set L_u^1 and L_u^2 respectively.
 4. Augment the labeled data with the k data points $L^1 = L^1 \cup L_u^1$ and $L^2 = L^2 \cup L_u^2$
3. Repeat all the steps under 2.

Advantages and limitations

The advantages and limitations are:

- When the features have different aspects or a mix of different domains, co-training becomes more beneficial than simple self-training
- The necessary and sufficient condition of having orthogonal views and ability to learn from them poses challenges for the generality of the technique

Cluster and label SSL

This technique, like self-training, is quite generic and applicable to domains and datasets where the clustering supposition mentioned in the assumptions section holds true (References [3]).

Inputs and outputs

Input is training data with a few labeled and a large number of unlabeled instances. A clustering algorithm and its parameters along with a classification algorithm with its parameters constitute additional inputs. The technique generates a classification model that can help predict the classes of unseen data.

How does it work?

The abstract algorithm can be given as:

1. Initialize data as $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ labeled and $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ unlabeled.
2. Cluster the entire data, both labeled and unlabeled using the clustering algorithm.
3. For each cluster let S be the set of labeled instances drawn from set L .
 1. Learn a supervised model from S , $f_S = L_S$.
 2. Apply the model f_S and classify unlabeled instances for each cluster using the preceding model.
4. Since all the unlabeled instances $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ get assigned a label by the preceding process, a supervised classification model is run on the entire set.

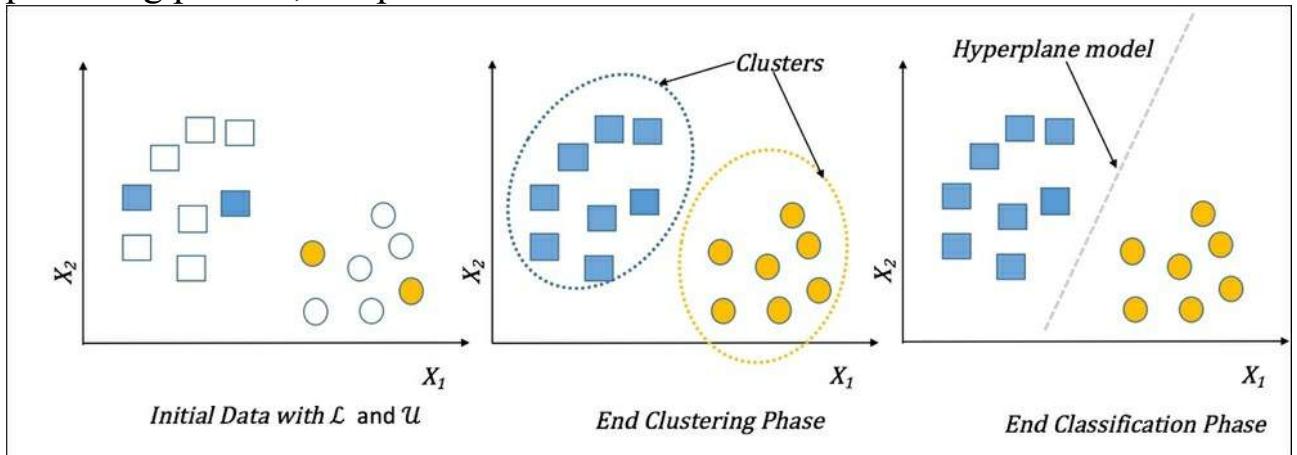


Figure 3. Clustering and label SSL – clustering followed by classification

Advantages and limitations

The advantages and limitations are:

- Works very well when the cluster assumption holds true and the choice of clustering algorithm and parameters are correct
- Large number of parameters and choices make this an unwieldy technique in many real-world problems

Transductive graph label propagation

The key idea behind graph-based methods is to represent every instance in the dataset, labeled and unlabeled, as a node and compute the edges as some form of "similarity" between them. Known labels are used to propagate labels in the unlabeled data using the basic concepts of label smoothness as discussed in the assumptions section, that is, similar data points will lie "close" to each other graphically (*References [4]*).

Figure 4 shows how the similarity indicated by the thickness of the arrow from the first data point to the last varies when the handwritten digit pattern changes. Knowing the first label, the label propagation can effectively label the next three digits due to the similarity in features while the last digit, though labeled the same, has a lower similarity as compared to the first three.

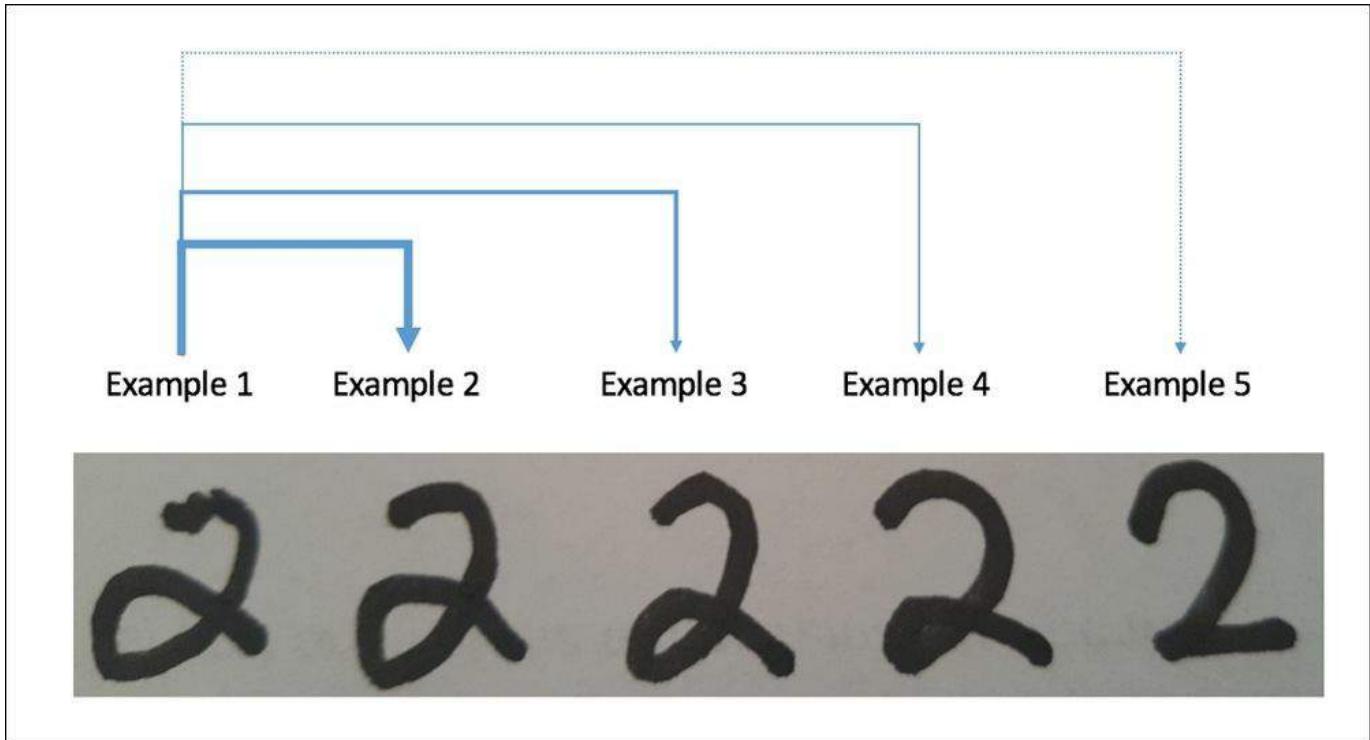


Figure 4. Transductive graph label propagation – classification of hand-written digits. Leftmost and rightmost images are labeled, others are unlabeled. Arrow thickness is a visual measure of similarity to labeled digit "2" on the left.

Inputs and outputs

Input is training data with a few labeled and a large number of unlabeled data. The graph weighting or similarity computing method, such as k-nearest weighting, Gaussian decaying distance, or ϵ -radius method is chosen. Output is the labeled set for the entire data; it generally doesn't build inductive models like the algorithms seen previously.

How does it work?

The general label propagation method follows:

1. Build a graph $g = (V, E)$ where:
 - Vertices $V = \{1, 2 \dots n\}$ correspond to data belonging to both labeled set L and unlabeled set U .
 - Edges E are weight matrices \mathbf{W} , such that $\mathbf{W}_{i,j}$ represents similarity in some form between two data points $\mathbf{x}_i, \mathbf{x}_j$.

2. Compute the diagonal degree matrix \mathbf{D} by $\mathbf{D}_{i,i} \leftarrow \sum_j \mathbf{W}_{i,j}$.
3. Assume the labeled set is binary and has $y_i \in \{1, -1\}$. Initialize the labels of all unlabeled data to be 0. $\hat{\mathbf{y}}^{(0)} = \{y_1, y_2, \dots, y_l, 0, 0, \dots, 0\}$
4. Iterate at $t = 0$:
 1. $\hat{\mathbf{y}}^{(t+1)} \leftarrow \mathbf{D}^{-1} \mathbf{W} \hat{\mathbf{Y}}^{(t)}$
 2. $\hat{\mathbf{Y}}^{(t+1)} \leftarrow Y_l$ (reset the labels of labeled instances back to the original)
 3. Go back to step 4, until convergence $\hat{\mathbf{Y}}^{(\infty)}$
5. Label the unlabeled points $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ using the convergence labels $\hat{\mathbf{Y}}^{(\infty)}$.

There are many variations based on similarity, optimization selected in iterations, and so on.

Advantages and limitations

The advantages and limitations are:

- The graph-based semi-supervised learning methods are costly in terms of computations—generally $O(n^3)$ where n is the number of instances. Though speeding and caching techniques help, the computational cost over large data makes it infeasible in many real-world data situations.
- The transductive nature makes it difficult for practical purposes where models need to be induced for unseen data. There are extensions such as Harmonic Mixtures, and so on, which address these concerns.

Transductive SVM (TSVM)

Transductive SVM is one of the oldest and most popular transductive semi-supervised learning methods, introduced by Vapnik (*References [5]*). The key principle is that unlabeled data along with labeled data can help find the decision boundary using concepts of large margins. The underlying principle is that the decision boundaries normally don't lie in high density regions!

Inputs and outputs

Input is training data with few labeled and a large number of unlabeled data. Input has to be numeric features for TSVM computations. The choice of kernels, kernel

parameters, and cost factors, which are all SVM-based parameters, are also input variables. The output is labels for the unlabeled dataset.

How does it work?

Generally, SVM works as an optimization problem in the labeled hard boundary

$$\min(f(\mathbf{w}, b)) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w}$$

SVM formulated in terms of weight vector \mathbf{w} and the bias b

$$\text{subject to } \forall_{i=1}^n : y_i |\mathbf{w}_i \mathbf{x}_i + b| \geq 1$$

1. Initialize the data as $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ labeled and $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ unlabeled.
2. In TSVM, the equation is modified as follows:

$$\min(f(y_{l+1}, y_{l+2} \dots y_n, \mathbf{w}, b)) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w}$$

This is subject to the following condition:

$$\forall_{i=1}^l : y_{l_i} |\mathbf{w}_i \mathbf{x}_{l_i} + b| \geq 1$$

$$\forall_{j=1}^u : y_{u_j} |\mathbf{w}_i \mathbf{x}_{u_j} + b| \geq 1$$

$$\forall_{j=1}^u : y_{u_j} \in (+1, -1)$$

This is exactly like inductive SVM but using only labeled data. When we constrain the unlabeled data to conform to the side of the hyperplane of labeled data in order to

maximize the margin, it results in unlabeled data being labeled with maximum margin separation! By adding the penalty factor to the constraints or replacing the dot product in the input space with kernels as in inductive SVM, complex non-linear noisy datasets can be labeled from unlabeled data.

Figure 5 illustrates the concept of TSVM in comparison with inductive SVM run on labeled data only and why TSVM can find better decision boundaries using the unlabeled datasets. The unlabeled datasets on either side of hyperplane are closer to their respective classes, thus helping find better margin separators.

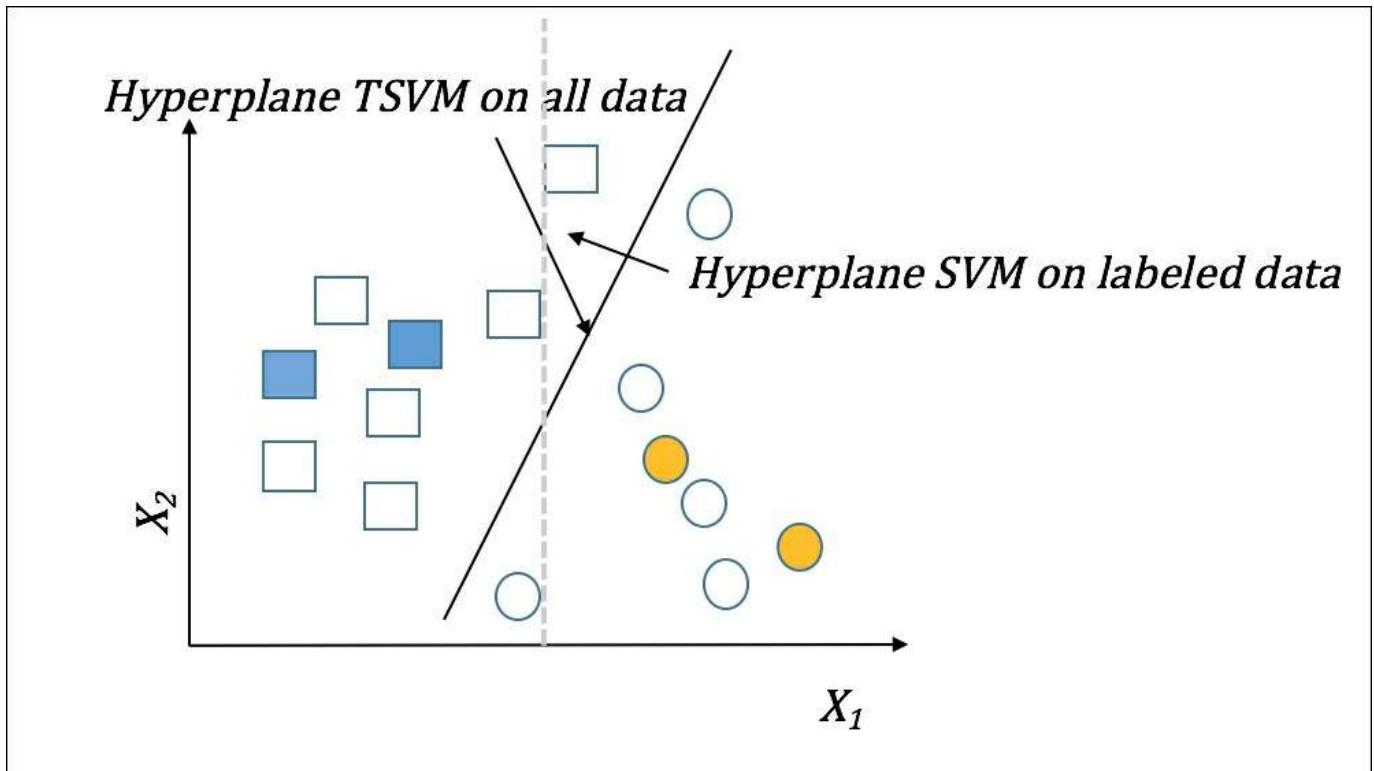


Figure 5. Transductive SVM

Advantages and limitations

The advantages and limitations:

- TSVMs can work very well in linear or non-linear datasets given noiseless labeled data.
- TSVMs have the same issues in finding hyper-parameters and tuning them to get the best results as inductive SVMs.

Case study in semi-supervised learning

For this case study, we use another well-studied dataset from the UCI Repository, the Wisconsin Breast Cancer dataset. In the first part of the experiment, we demonstrate how to apply the Transductive SVM technique of semi-supervised learning using the open-source library called `JKernelMachines`. We choose the SVMLight algorithm and a Gaussian kernel for this technique.

In the second part, we use KEEL, a GUI-based framework and compare results from several evolutionary learning based algorithms using the UCI Breast Cancer dataset. The tools, methodology, and evaluation measures are described in the following subsections.

Tools and software

The two open source Java tools used in the semi-supervised learning case study are `JKernelMachines`, a Transductive SVM, and KEEL, a GUI-based tool that uses evolutionary algorithms for learning.

Note

JKernelMachines (Transductive SVM)

`JKernelMachines` is a pure Java library that provides an efficient framework for using and rapidly developing specialized kernels. Kernels are similarity functions used in SVMs. `JKernelMachines` provides kernel implementations defined on structured data in addition to standard kernels on vector data such as Linear and Gaussian. In particular, it offers a combination of kernels, kernels defined over lists, and kernels with various caching strategies. The library also contains SVM optimization algorithm implementations including LaSVM and One-Class SVM using SMO. The creators of the library report that the results of `JKernelMachines` on some common UCI repository datasets are comparable to or better than the Weka library.

The example of loading data and running Transductive SVM using `JKernelMachines` is given here:

```
try {
//load the labeled training data
List<TrainingSample<double[]>> labeledTraining =
```

```

ArffImporter.importFromFile("resources/breast-labeled.arff");
//load the unlabeled data
List<TrainingSample<double[]>> unlabeledData
=ArffImporter.importFromFile("resources/breast-unlabeled.arff");
//create a kernel with Gaussian and gamma set to 1.0
DoubleGaussL2 k = new DoubleGaussL2(1.0);
//create transductive SVM with SVM light
S3VMLight<double[]> svm = new S3VMLight<double[]>(k);
//send the training labeled and unlabeled data
svm.train(labeledTraining, unlabeledData);
} catch (IOException e) {
    e.printStackTrace();
}

```

In the second approach, we use KEEL with the same dataset.

Note

KEEL

KEEL (Knowledge Extraction based on Evolutionary Learning) is a non-commercial (GPLv3) Java tool with GUI which enables users to analyze the behavior of evolutionary learning for a variety of data mining problems, including regression, classification, and unsupervised learning. It relieves users from the burden of programming sophisticated evolutionary algorithms and allows them to focus on new learning models created using the toolkit. KEEL is intended to meet the needs of researchers as well as students.

KEEL contains algorithms for data preprocessing and post-processing as well as statistical libraries, and a Knowledge Extraction Algorithms Library which incorporates multiple evolutionary learning algorithms with classical learning techniques.

The GUI wizard included in the tool offers different functional components for each stage of the pipeline, including:

- Data management: Import, export of data, data transformation, visualization, and so on
- Experiment design: Selection of classifier, estimator, unsupervised techniques, validation method, and so on
- SSL experiments: Transductive and inductive classification (see image of off-line method for SSL experiment design in this section)

- Statistical analysis: This provides tests for pair-wise and multiple comparisons, parametric, and non-parametric procedures.

For more info, visit <http://sci2s.ugr.es/keel/> and <http://sci2s.ugr.es/keel/pdf/keel/articulo/Alcaletal-SoftComputing-Keel1.0.pdf>.

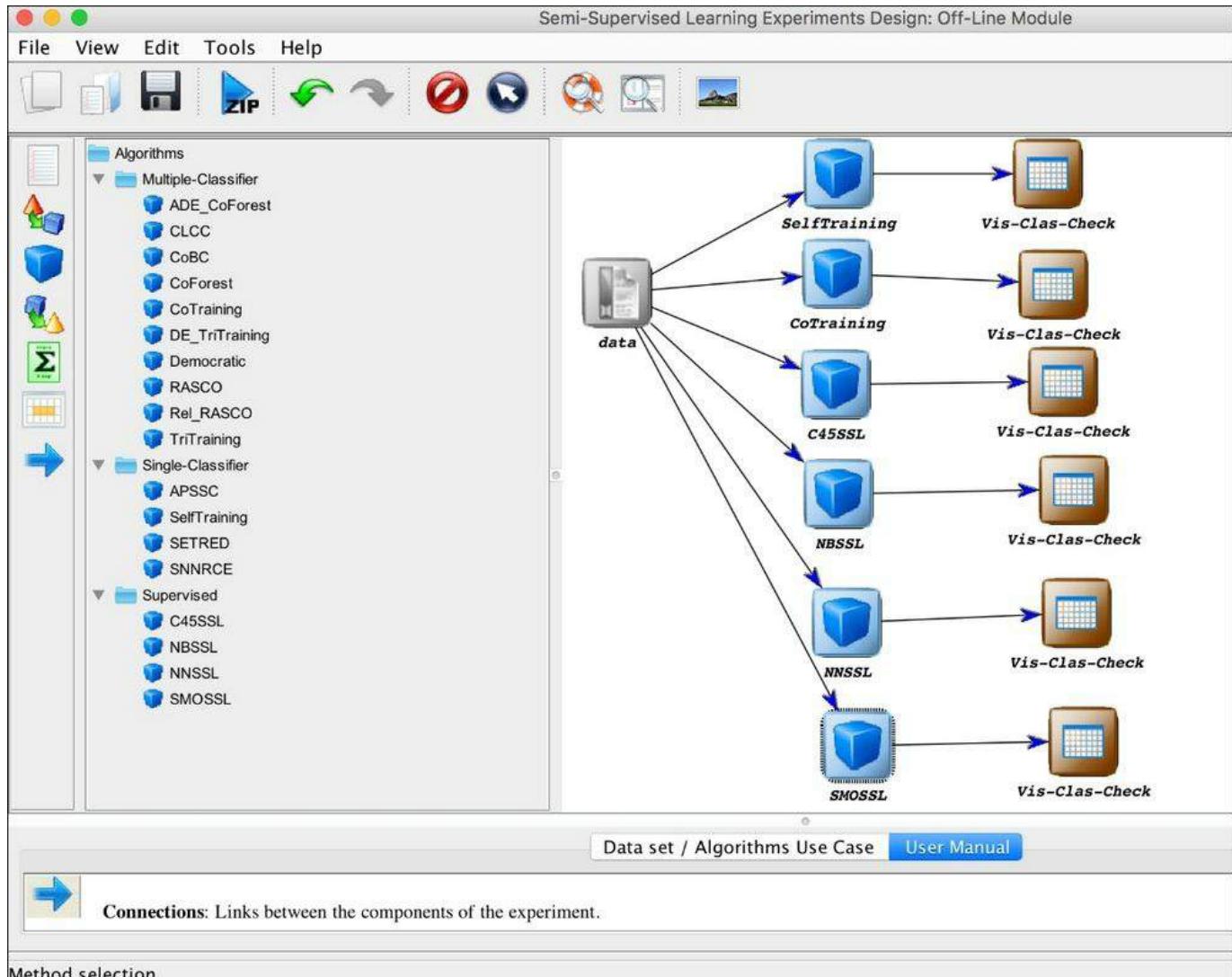


Figure 6: KEEL – wizard-based graphical interface

Business problem

Breast cancer is the top cancer in women worldwide and is increasing particularly in developing countries where the majority of cases are diagnosed in late stages. Examination of tumor mass using a non-surgical procedure is an inexpensive and

preventative measure for early detection of the disease.

In this case-study, a marked dataset from such a procedure is used and the goal is to classify the breast cancer data into Malignant and Benign using multiple SSL techniques.

Machine learning mapping

To illustrate the techniques learned in this chapter so far, we will use SSL to classify. Whereas the dataset contains labels for all the examples, in order to treat this as a problem where we can apply SSL, we will consider a fraction of the data to be unlabeled. In fact, we run multiple experiments using different fractions of unlabeled data for comparison. The different base learners used are classification algorithms familiar to us from previous chapters.

Data collection

This dataset was collected by the University of Wisconsin Hospitals, Madison. The dataset is available in Weka ARFF format. The data is not partitioned into training, validation and test.

Data quality analysis

The examples in the data contain no unique identifier. There are 16 examples for which the Bare Nuclei attribute has missing values. The target Class is the only categorical attribute and has two values. All other attributes are continuous and in the range [1, 10].

Data sampling and transformation

In the experiments, we present results for 10-fold cross-validation. For comparison, four runs were performed each using a different fraction of labeled data—10%, 20%, 30%, and 40%.

A numeric sample code number was added to each example as a unique identifier. The categorical values Malignant and Benign, for the class attribute, were replaced by the numeric values 4 and 2 respectively.

Datasets and analysis

The Breast Cancer Dataset Wisconsin (Original) is available from the UCI Machine

Learning Repository at:

[https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Original\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)).

This database was originally obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The dataset was created by Dr. Wolberg for the diagnosis and prognosis of breast tumors. The data is based exclusively on measurements involving the **Fine Needle Aspirate (FNA)** test. In this test, fluid from a breast mass is extracted using a small-gauge needle and then visually examined under a microscope.

A total of 699 instances with nine numeric attributes and a binary class (malignant/benign) constitute the dataset. The percentage of missing values is 0.2%. There are 65.5% malignant and 34.5% benign cases in the dataset. The feature names and range of valid values are listed in the following table:

Num.	Feature Name	Domain
1	Sample code number	id number
2	Clump Thickness	1 - 10
3	Uniformity of Cell Size	1 - 10
4	Uniformity of Cell Shape	1 - 10
5	Marginal Adhesion	1 - 10
6	Single Epithelial Cell Size	1 - 10
7	Bare Nuclei	1 - 10
8	Bland Chromatin	1 - 10
9	Normal Nucleoli	1 - 10
10	Mitoses	1 - 10
11	Class	2 for benign, 4 for malignant

Feature analysis results

Summary statistics by feature appear in Table 1.

	Clump Thickness	Cell Size Uniformity	Cell Shape Uniformity	Marginal Adhesion	Single Epi Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
mean	4.418	3.134	3.207	2.807	3.216	3.545	3.438	2.867	1.589
std	2.816	3.051	2.972	2.855	2.214	3.644	2.438	3.054	1.715
min	1	1	1	1	1	1	1	1	1
25%	2	1	1	1	2	2	1	1	1
50%	4	1	1	1	2	3	1	1	1
75%	6	5	5	4	4	5	4	4	1
max	10	10	10	10	10	10	10	10	10

Table 1. Features summary

Experiments and results

Two SSL algorithms were selected for the experiments—self-training and co-training. In addition, four classification methods were chosen as base learners—Naïve Bayes, C4.5, K-NN, and SMO. Further, each experiment was run using four different partitions of labeled and unlabeled data (10%, 20%, 30%, and 40% labeled).

The hyper-parameters for the algorithms and base classifiers are given in Table 2. You can see the accuracy across the different runs corresponding to four partitions of labeled and unlabeled data for the two SSL algorithms.

Finally, we give the performance results for each experiment for the case of 40% labeled. Performance metrics provided are Accuracy and the Kappa statistic with standard deviations.

Method	Parameters
Self-	MAX_ITER = 40

Training	
Co-Training	MAX_ITER = 40, Initial Unlabeled Pool=75
KNN	K = 3, Euclidean distance
C4.5	pruned tree, confidence = 0.25, 2 examples per leaf
NB	No parameters specified
SMO	C = 1.0, Tolerance Parameter = 0.001, Epsilon= 1.0E-12, Kernel Type = Polynomial, Polynomial degree = 1, Fit logistic models = true

Table 2. Base classifier hyper-parameters for self-training and co-training

SSL Algorithm	10%	20%	30%	40%
Self-Training C 4.5	0.9	0.93	0.94	0.947
Co-Training SMO	0.959	0.949	0.962	0.959

Table 3. Model accuracy for samples with varying fraction of labeled examples

Algorithm	Accuracy (no unlabeled)
C4.5 10-fold CV	0.947
SMO 10 fold CV	0.967

		10 fold CV Wisconsin 40% Labeled Data		
Self-Training (kNN)	Accuracy	0.9623 (1)	Kappa	0.9170 (2)
	Std Dev	0.0329	Std Dev	0.0714
Self-Training (C45)	Accuracy	0.9606 (3)	Kappa	0.9144
	Std Dev	0.0241	Std Dev	0.0511

Self-Training (NB)	Accuracy	0.9547	Kappa	0.9036
	Std Dev	0.0252	Std Dev	0.0533
Self-Training (SMO)	Accuracy	0.9547	Kappa	0.9035
	Std Dev	0.0208	Std Dev	0.0435
Co-Training (NN)	Accuracy	0.9492	Kappa	0.8869
	Std Dev	0.0403	Std Dev	0.0893
Co-Training (C45)	Accuracy	0.9417	Kappa	0.8733
	Std Dev	0.0230	Std Dev	0.0480
Co-Training (NB)	Accuracy	0.9622 (2)	Kappa	0.9193 (1)
	Std Dev	0.0290	Std Dev	0.0614
Co-Training (SMO)	Accuracy	0.9592	Kappa	0.9128 (3)
	Std Dev	0.0274	Std Dev	0.0580

Table 4. Model performance comparison using 40% labeled examples. The top ranking performers in each category are shown in parentheses.

Analysis of semi-supervised learning

With 40% of labeled data, semi-supervised self-training with C4.5 achieves the same result as 100% of labeled data with just C4.5. This shows the strength of semi-supervised learning when the data is sparsely labeled.

SMO with polynomial kernel, with 30-40% data comes close to the 100% data but not as good as C4.5.

Self-training and co-training with four classifiers on 40% labeled training data shows

- KNN as base classifier and self-training has the highest accuracy (0.9623) which indicates the non-linear boundary of the data. Co-training with Naïve Bayes comes very close.

- Self-training with classifiers such as linear Naïve Bayes, non-linear C4.5 and highly non-linear KNN shows steady improvements in accuracy: 0.9547, 0.9606, 0.9623, which again shows that using self-training but choosing the right underlying classifier for the problem is very important.
- Co-training with Naive Bayes has highest Kappa statistic (0.9193) and almost similar accuracy as KNN with self-training. The independence relationship between features—hence breaking the feature sets into orthogonal feature sets and using them for classifiers—improves the learning.

Active learning

Although active learning has many similarities with semi-supervised learning, it has its own distinctive approach to modeling with datasets containing labeled and unlabeled data. It has roots in the basic human psychology that asking more questions often tends to solve problems.

The main idea behind active learning is that if the learner gets to pick the instances to learn from rather than being handed labeled data, it can learn more effectively with less data (*Reference [6]*). With very small amount of labeled data, it can carefully pick instances from unlabeled data to get label information and use that to iteratively improve learning. This basic approach of querying for unlabeled data to get labels from a so-called oracle—an expert in the domain—distinguishes active learning from semi-supervised or passive learning. The following figure illustrates the difference and the iterative process involved:

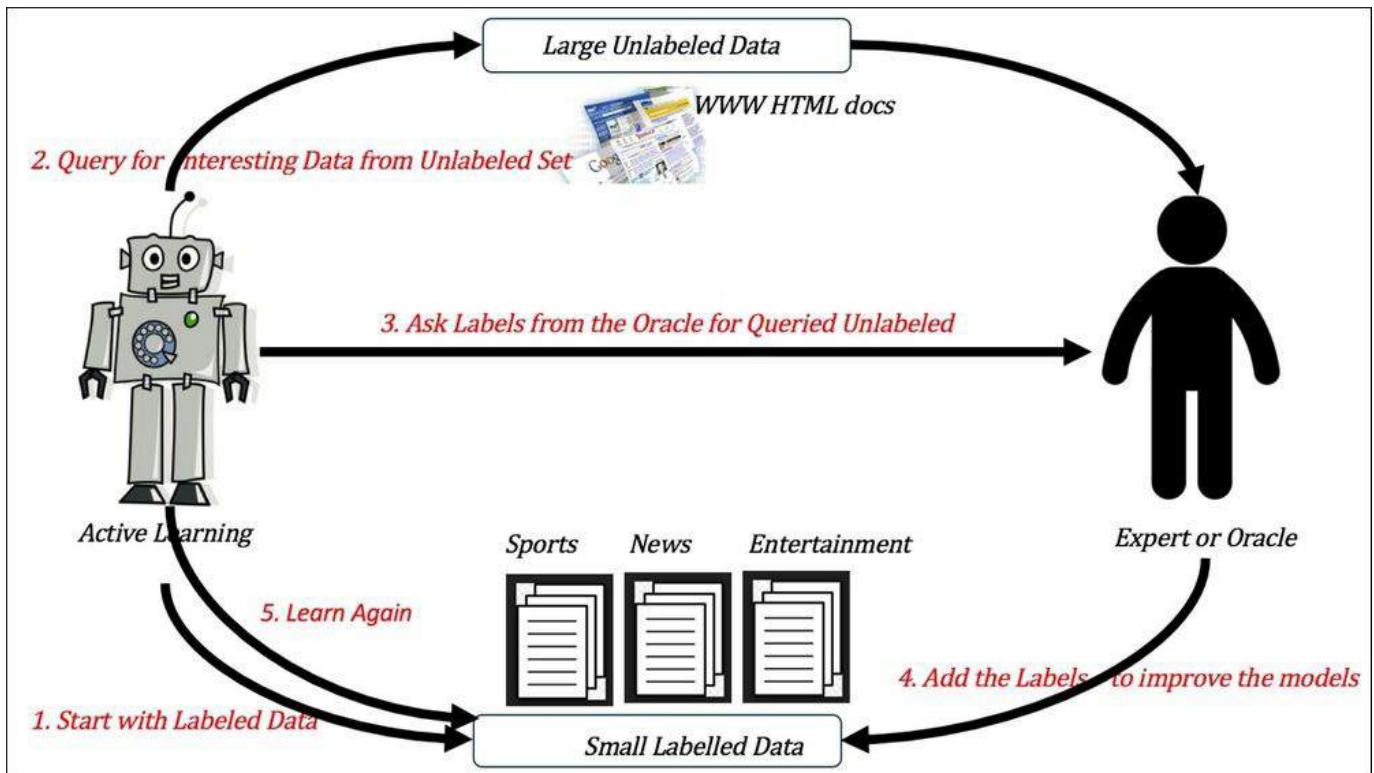


Figure 7. Active Machine Learning process contrasted with Supervised and Semi-Supervised Learning process.

Representation and notation

The dataset D , which represents all the data instances and their labels, is given by $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ are the individual instances of data and $\{y_1, y_2, \dots, y_n\}$ is the set of associated labels. D is composed of two sets U , labeled data and L , unlabeled data. \mathbf{x} is the set $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ of data instances without labels.

The dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ consists of all labeled data with known outcomes $\{y_1, y_2, \dots, y_l\}$ whereas $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ is the dataset where the outcomes are not known. As before, $|U| \gg |L|$.

Active learning scenarios

Active learning scenarios can be broadly classified as:

- **Stream-based active learning:** In this approach, an instance or example is picked only from the unlabeled dataset and a decision is made whether to ignore the data or pass it on to the oracle to get its label (*Referee*[10,11]).
- **Pool-based active learning:** In this approach, the instances are queried from the unlabeled dataset and then ranked on the basis of informativeness and a set from these is sent to the Oracle to get the labels (*Referees* [12]).
- **Query synthesis:** In this method, the learner has only information about input space (features) and synthesizes queries from the unlabeled set for the membership. This is not used in practical applications, as most often it doesn't consider the data generating distribution and hence often the queries are arbitrary or meaningless.

Active learning approaches

Regardless of the scenario involved, every active learning approach includes selecting a query strategy or sampling method which establishes the mechanism for picking the queries in each iteration. Each method reveals a distinct way of seeking out unlabeled examples with the best information content for improving the learning process. In the following subsections, we describe the major query strategy frameworks, how they work, their merits and limitations, and the different strategies within each framework.

Uncertainty sampling

The key idea behind this form of sampling is to select instances from the unlabeled pool that the current model is most uncertain about. The learner can then avoid the instances the model is more certain or confident in classifying (*Reerences [8]*).

Probabilistic based models (Naïve Bayes, Logistic Regression, and so on) are the most natural choices for such methods as they give confidence measures on the given model, say θ for the data \mathbf{x} , for a class y_i $i \in \text{classes}$, and the probability

$P_\theta(\hat{y}|x)$ as the posterior probability.

How does it work?

The general process for all uncertainty-based algorithms is outlined as follows:

1. Initialize the data as labeled, $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ and unlabeled, $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$.
2. While there is still unlabeled data:
 1. Train the classifier model $f: \mathbf{x} \rightarrow y$ with the labeled data L .
 2. Apply the classifier model f on the unlabeled data U to assess informativeness J using one of the sampling mechanisms (see next section)
 3. Choose k most informative data from U as set L_u to get labels from the oracle.
 4. Augment the labeled data with the k new labeled data points obtained in the previous step: $L = L \cup L_u$.
3. Repeat all the steps under 2.

Some of the most common query synthesis algorithms to sample the informative instances from the data are given next.

Least confident sampling

In this technique, the data instances are sorted based on their confidence in reverse, and the instances most likely to be queried or selected are the ones the model is least confident about. The idea behind this is the least confident ones are the ones near the margin or separating hyperplane and getting their labels will be the best way to learn the boundaries effectively.

This can be formulated as $Q\{\mathbf{x}_u\} = \operatorname{argmin} P_\theta(\hat{y}|\mathbf{x})$.

The disadvantage of this method is that it effectively considers information of the best; information on the rest of the posterior distribution is not used.

Smallest margin sampling

This is margin-based sampling, where the instances with smaller margins have more ambiguity than instances with larger margins.

This can be formulated as $Q\{\mathbf{x}_u\} = \operatorname{argmin} [P_\theta(\hat{y}_1|\mathbf{x}) - P_\theta(\hat{y}_2|\mathbf{x})]$ where \hat{y}_1 and \hat{y}_2 are two labels for the instance \mathbf{x} .

Label entropy sampling

Entropy, which is the measure of average information content in the data and is the impurity measure, can be used to sample the instances. This can be formulated as:

$$Q\{\mathbf{x}_u\} = \operatorname{argmax}_{\mathbf{x}} - \sum_{y \in \text{classes}} P_\theta(y|\mathbf{x}) \log(P_\theta(y|\mathbf{x}))$$

Advantages and limitations

The advantages and limitations are:

- Label entropy sampling is the simplest approach and can work with any probabilistic classifiers—this is the biggest advantage
- Presence of outliers or wrong feedback can go unnoticed and models can

degrade

Version space sampling

Hypothesis H is the set of all the particular models that generalize or explain the training data; for example, all possible sets of weights that separate two linearly separable classes. Version spaces V are subsets of hypothesis H , which are consistent with the training data as defined by Tom Mitchell (*References [15]*) such that $\mathcal{V} \subseteq \mathcal{H}$.

The idea behind this sampling is to query instances from the unlabeled dataset that reduce the size of the version space or minimize $|V|$.

Query by disagreement (QBD)

QBD is one of the earliest algorithms which works on maintaining a version space V —when two hypotheses disagree on the label of new incoming data, that instance is selected for getting labels from the oracle or expert.

How does it work?

The entire algorithm can be summarized as follows:

1. Initialize $\mathcal{V} \subseteq \mathcal{H}$ as the set of all legal hypotheses.
2. Initialize the data as $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ labeled and $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ unlabeled.
3. While data \mathbf{x}' is in U :
 1. If $h_1(\mathbf{x}') \neq h_2(\mathbf{x}')$ for any $h_2 \in V$:
 1. Query the label of \mathbf{x}' and get y' .
 2. $V = \{h : h(\mathbf{x}') = y'\text{ for all points.}\}$
 2. Otherwise:
 1. Ignore \mathbf{x}' .

Query by Committee (QBC)

Query by Committee overcomes the limitation of Query by Disagreement related to maintaining all possible version spaces by creating a committee of classifiers and using their votes as the mechanism to capture the disagreement (*References [7]*).

How does it work?

For this algorithm:

1. Initialize the data as $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ labeled and $\mathcal{U} = \{\mathbf{x}_i, y_i\}_{i=l+1}^n$ unlabeled.
2. Train the committee of models $C = \{\theta^1, \theta^2, \dots, \theta^c\}$ on the labeled data w (see the following).
3. For all data \mathbf{x}' in the \mathcal{U} :
 1. Vote for predictions on \mathbf{x}' as $\{y_\theta^1, y_\theta^2, \dots, y_\theta^c\}$.
 2. Rank the instances based on maximum disagreement (see the following).
 3. Choose k most informative data from \mathcal{U} as set L_u to get labels from the oracle.
 4. Augment the labeled data with the k new labeled data points $L = L \cup L_u$.
 5. Retrain the models $\{\theta_1, \theta_2, \dots, \theta_c\}$ with new L .

With the two tasks of training the committee of learners and choosing the disagreement methods, each has various choices.

Training different models can be either done using different samples from L or they can be trained using ensemble methods such as boosting and bagging.

Vote entropy is one of the methods chosen as the disagreement metric to rank. The mathematical way of representing it is:

$$Q\{\mathbf{x}_u\} = \operatorname{argmax}_{\mathbf{x}} \sum_i \frac{V(y_i)}{|C|} \log \frac{V(y_i)}{|C|}$$

Here, $V(y_i)$ is number of votes given to the label y_i from all possible labels and $|C|$ is size of the committee.

Kullback-Leibler (KL) divergence is an information theoretic measure of divergence between two probability distributions. The disagreement is quantified as the average divergence of each committee's prediction from that of the consensus in the committee C :

$$Q\left\{\mathbf{x}_u\right\} = argmax_{\mathbf{x}} \frac{1}{|C|}\Sigma_{\theta \in C}KL\Big(P_{\theta}\left(y|\mathbf{x}\right) || P_c\left(y|\mathbf{x}\right)\Big)$$

Advantages and limitations

The advantages and limitations are the following:

- Simplicity and the fact that it can work with any supervised algorithm gives it a great advantage.
- There are theoretical guarantees of minimizing errors and generalizing in some conditions.
- Query by Disagreement suffers from maintaining a large number of valid hypotheses.
- These methods still suffer from the issue of wrong feedback going unnoticed and models potentially degrading.

Data distribution sampling

The previous methods selected the best instances from the unlabeled set based either on the uncertainty posed by the samples on the models or by reducing the hypothesis space size. Neither of these methods worked on what is best for the model itself. The idea behind data distribution sampling is that adding samples that help reduce the errors to the model serves to improve predictions on unseen instances using expected values (*References [13 and 14]*).

How does it work?

There are different ways to find what is the best sample for the given model and here we will describe each one in some detail.

Expected model change

The idea behind this is to select examples from the unlabeled set that will bring maximum change in the model:

$$Q\{\mathbf{x}_u\} = \operatorname{argmin}_{\mathbf{x}} \sum_y P_{\theta}(y|\mathbf{x}) \sum_{x' \in u} H_{\theta+}(y|x')$$

Here, $P_{\theta}(y|\mathbf{x})$ = expectation over labels of \mathbf{x} , $\sum_{x' \in u} H_{\theta+}(y|x')$ is the sum over unlabeled instances of the entropy of including \mathbf{x}' after retraining with \mathbf{x} .

Expected error reduction

Here, the approach is to select examples from the unlabeled set that reduce the model's generalized error the most. The generalized error is measured using the unlabeled set with expected labels:

$$Q\{\mathbf{x}_u\} = \operatorname{argmin}_{\mathbf{x}} \sum_y P_{\theta}(y|\mathbf{x}) \sum_{x' \in u} 1 - p_{\theta+}(y|x')$$

Here, $P\theta(y|\mathbf{x}) = \text{expectation over labels of } \mathbf{x}, \sum_{x' \in u} H_\theta + (y|x')$ is the sum over unlabeled instances of the entropy of including \mathbf{x}' after retraining with \mathbf{x} .

Variance reduction

The general equation of estimation on an out-of-sample error in terms of noise-bias-variance is given by the following:

$$E((G(\mathbf{x}) - y))^2 | \mathbf{x}) = \text{noise} + \text{bias} + \text{variance}$$

Here, $G(\mathbf{x})$ is the model's prediction given the label y . In variance reduction, we select examples from the unlabeled set that most reduce the variance in the model:

$$Q\{\mathbf{x}_u\} = \operatorname{argmin}_{\mathbf{x}} \sum_{x' \in u} VAR_\theta + (y'| \mathbf{x}')$$

Here, $\theta +$ represents the model after it has been retrained with the new point \mathbf{x}' and its label y' .

Density weighted methods

In this approach, we select examples from the unlabeled set that have average similarity to the labeled set.

This can be represented as follows:

$$Q\{\mathbf{x}_u\} = \operatorname{argmax}_{\mathbf{x}} \sum_{x \in u} sim(\mathbf{x} | \mathbf{x}') \times H_\theta(y | \mathbf{x})$$

Here, $sim(\mathbf{x}, \mathbf{x}')$ is the density term or the similarity term where $H_\theta(y | \mathbf{x})$ is the base utility measure.

Advantages and limitations

The advantages and limitations are as follows:

- The biggest advantage is that they work directly on the model as an optimization objective rather than implicit or indirect methods described before
- These methods can work on pool- or stream-based scenarios
- These methods have some theoretical guarantees on the bounds and generalizations
- The biggest disadvantage of these methods is computational cost and difficulty in implementation

Case study in active learning

This case study uses another well-known publicly available dataset to demonstrate active learning techniques using open source Java libraries. As before, we begin with defining the business problem, what tools and frameworks are used, how the principles of machine learning are realized in the solution, and what the data analysis steps reveal. Next, we describe the experiments that were conducted, evaluate the performance of the various models, and provide an analysis of the results.

Tools and software

For the experiments in Active Learning, JCLAL was the tool used. JCLAL is a Java framework for Active Learning, supporting single-label and multi-label learning.

Note

JCLAL is open source and is distributed under the GNU general public license:
<https://sourceforge.net/p/jclal/git/ci/master/tree/>.

Business problem

The abalone dataset, which is used in these experiments, contains data on various physical and anatomical characteristics of abalone—commonly known as sea snails. The goal is to predict the number of rings in the shell, which is indicative of the age of the specimen.

Machine learning mapping

As we have seen, active learning is characterized by starting with a small set of labeled data accompanied by techniques of querying the unlabeled data such that we incrementally add instances to the labeled set. This is performed over multiple iterations, a batch at a time. The number of iterations and batch size are hyper-parameters for these techniques. The querying strategy and the choice of supervised learning method used to train on the growing number of labeled instances are additional inputs.

Data Collection

As before, we will use an existing dataset available from the UCI repository (<https://archive.ics.uci.edu/ml/datasets/Abalone>). The original owners of the database are the Department of Primary Industry and Fisheries in Tasmania, Australia.

The data types and descriptions of the attributes accompany the data and are reproduced in *Table 5*. The class attribute, Rings, has 29 distinct classes:

Name	Data type	Measurement units	Description
Sex	nominal	M, F, and I (infant)	sex of specimen
Length	continuous	mm	longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer	count	+1.5 gives the age in years

Table 5. Abalone dataset features

Data sampling and transformation

For this experiment, we treated a randomly selected 4,155 records as unlabeled and kept the remaining 17 as labeled. There is no transformation of the data.

Feature analysis and dimensionality reduction

With only eight features, there is no need for dimensionality reduction. The dataset comes with some statistics on the features, reproduced in *Table 6*:

	Length	Diameter	Height	Whole	Shucked	Viscera	Shell	Rings
Min	0.075	0.055	0	0.002	0.001	0.001	0.002	1
Max	0.815	0.65	1.13	2.826	1.488	0.76	1.005	29
Mean	0.524	0.408	0.14	0.829	0.359	0.181	0.239	9.934
SD	0.12	0.099	0.042	0.49	0.222	0.11	0.139	3.224
Correl	0.557	0.575	0.557	0.54	0.421	0.504	0.628	1

Table 6. Summary statistics by feature

Models, results, and evaluation

We conducted two sets of experiments. The first used pool-based scenarios and the second, stream-based. In each set, we used entropy sampling, least confident sampling, margin sampling, and vote entropy sampling. The classifiers used were Naïve Bayes, Logistic Regression, and J48 (implementation of C4.5). For every experiment, 100 iterations were run, with batch sizes of 1 and 10. In *Table 7*, we present a subset of these results, specifically, pool-based and stream-based scenarios for each sampling method using Naïve Bayes, Simple Logistic, and C4.5 classifiers with a batch size of 10.

Note

The full set of results can be seen at <https://github.com/mjmlbook/mastering-java-machine-learning/tree/master/Chapter4>.

The JCLAL library requires an XML configuration file to specify which scenario to use, the query strategy selected, batch size, max iterations, and base classifier. The following is an example configuration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<experiment>
    <process evaluation-method-
type="net.sf.jclal.evaluation.method.RealScenario">
        <file-labeled>datasets/abalone-labeled.arff</file-labeled>
        <file-unlabeled>datasets/abalone-unlabeled.arff</file-
unlabeled>
        <algorithm
type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
            <stop-criterion
type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
                <max-iteration>10</max-iteration>
            </stop-criterion>
            <stop-criterion
type="net.sf.jclal.activelearning.stopcriteria.UnlabeledSetEmpty"/>
                <listener
type="net.sf.jclal.listener.RealScenarioListener">
                    <informative-instances>reports/real-scenario-
informative-data.txt</informative-instances>
                </listener>
                <scenario
type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
                    <batch-mode
type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
```

```

        <batch-size>1</batch-size>
    </batch-mode>
    <oracle
type="net.sf.jclal.activelearning.oracle.ConsoleHumanOracle"/>
        <query-strategy
type="net.sf.jclal.activelearning.singlelabel.querystrategy.EntropySamplingQueryStrategy">
            <wrapper-classifier
type="net.sf.jclal.classifier.WekaClassifier">
                <classifier
type="weka.classifiers.bayes.NaiveBayes"/>
                    </wrapper-classifier>
                </query-strategy>
            </scenario>
        </algorithm>
    </process>
</experiment>

```

The tools itself is invoked via the following:

```
java -jar jclal-<version>.jar -cfg <config-file>
```

Pool-based scenarios

In the following three tables, we compare results using pool-based scenarios when using Naïve Bayes, Simple Logistic, and C4.5 classifiers.

Naïve Bayes:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall
PoolBased-EntropySampling-NaiveBayes-b10	0.6021	0.1032	0.0556(1)	0.1805	0.1304
PoolBased-KLDivergence-NaiveBayes-b10	0.6639(1)	0.1441(1)	0.0563	0.1765	0.1504
PoolBased-LeastConfidentSampling-NaiveBayes-b10	0.6406	0.1300	0.0827	0.1835(1)	0.1810(1)
PoolBased-VoteEntropy-NaiveBayes-b10	0.6639(1)	0.1441(1)	0.0563	0.1765	0.1504

Table 7. Performance of pool-based scenario using Naïve Bayes classifier

Logistic Regression:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall
PoolBased-EntropySampling-SimpleLogistic-b10	0.6831	0.1571	0.1157	0.1651	0.2185(1)
PoolBased-KLDivergence-SimpleLogistic-b10	0.7175(1)	0.1616	0.1049	0.2117(1)	0.2065
PoolBased-LeastConfidentSampling-SimpleLogistic-b10	0.6629	0.1392	0.1181(1)	0.1751	0.1961
PoolBased-VoteEntropy-SimpleLogistic-b10	0.6959	0.1634(1)	0.0895	0.2307	0.1880

Table 8. Performance of pool-based scenario using Logistic Regression classifier

C4.5:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall
PoolBased-EntropySampling-J48-b10	0.6730(1)	0.3286(1)	0.0737	0.3432(1)	0.32780(1)
PoolBased-KLDivergence-J48-b10	0.6686	0.2979	0.0705(1)	0.3153	0.2955
PoolBased-LeastConfidentSampling-J48-b10	0.6591	0.3094	0.0843	0.3124	0.3227
PoolBased-VoteEntropy-J48-b10	0.6686	0.2979	0.0706	0.3153	0.2955

Table 9. Performance of pool-based scenario using C4.5 classifier

Stream-based scenarios

In the following three tables, we have results for experiments on stream-based scenarios using Naïve Bayes, Logistic Regression, and C4.5 classifiers with four different sampling methods.

Naïve Bayes:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall

StreamBased-EntropySampling-NaiveBayes-b10	0.6673(1)	0.1432(1)	0.0563	0.1842(1)	0.1480
StreamBased-LeastConfidentSampling-NaiveBayes-b10	0.5585	0.0923	0.1415	0.1610	0.1807(1)
StreamBased-MarginSampling-NaiveBayes-b10	0.6736(1)	0.1282	0.0548(1)	0.1806	0.1475
StreamBased-VoteEntropyQuery-NaiveBayes-b10	0.5585	0.0923	0.1415	0.1610	0.1807(1)

Table 10. Performance of stream-based scenario using Naïve Bayes classifier

Logistic Regression:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall
StreamBased-EntropySampling-SimpleLogistic-b10	0.7343(1)	0.1994(1)	0.0871	0.2154	0.2185(1)
StreamBased-LeastConfidentSampling-SimpleLogistic-b10	0.7068	0.1750	0.0906	0.2324(1)	0.2019
StreamBased-MarginSampling-SimpleLogistic-b10	0.7311	0.1994(1)	0.0861	0.2177	0.214
StreamBased-VoteEntropy-SimpleLogistic-b10	0.5506	0.0963	0.0667(1)	0.1093	0.1117

Table 11. Performance of stream-based scenario using Logistic Regression classifier

C4.5:

Experiment	Area Under ROC	F Measure	False Positive Rate	Precision	Recall
StreamBased-EntropySampling-J48-b10	0.6648	0.3053	0.0756	0.3189(1)	0.3032
StreamBased-LeastConfidentSampling-J48-b10	0.6748(1)	0.3064(1)	0.0832	0.3128	0.3189(1)
StreamBased-MarginSampling-J48-b10	0.6660	0.2998	0.0728(1)	0.3163	0.2967
StreamBased-VoteEntropy-J48-b10	0.4966	0.0627	0.0742	0.1096	0.0758

Table 12. Performance of stream-based scenario using C4.5 classifier

Analysis of active learning results

It is quite interesting to see that pool-based, Query By Committee—an ensemble method—using KL-Divergence sampling does really well across most classifiers. As discussed in the section, these methods have been proven to have a theoretical guarantee on reducing the errors by keeping a large hypothesis space, and this experimental result supports that empirically.

Pool-based, entropy-based sampling using C4.5 as a classifier has the highest Precision, Recall, FPR and F-Measure. Also with stream-based, entropy sampling with C4.5, the metrics are similarly high. With different sampling techniques and C4.5 using pool-based as in KL-Divergence, LeastConfident or vote entropy, the metrics are significantly higher. Thus, this can be attributed more strongly to the underlying classifier C4.5 in finding non-linear patterns.

The Logistic Regression algorithm performs very well in both stream-based and pool-based when considering AUC. This may be completely due to the fact that LR has a good probabilistic approach in confidence mapping, which is an important factor for giving good AUC scores.

Summary

After a tour of supervised and unsupervised machine learning techniques and their application to real-world datasets in the previous chapters, this chapter introduces the concepts, techniques, and tools of **Semi-Supervised Learning (SSL)** and **Active Learning (AL)**.

In SSL, we are given a few labeled examples and many unlabeled ones—the goal is either to simply train on the labeled ones in order to classify the unlabeled ones (transductive SSL), or use the unlabeled and labeled examples to train models to correctly classify new, unseen data (inductive SSL). All techniques in SSL are based on one or more of the assumptions related to semi-supervised smoothness, cluster togetherness, and manifold togetherness.

Different SSL techniques are applicable to different situations. The simple self-training SSL is straightforward and works with most supervised learning algorithms; when the data is from more than just one domain, the co-training SSL is a suitable method. When the cluster togetherness assumption holds, the cluster and label SSL technique can be used; a "closeness" measure is exploited by transductive graph label propagation, which can be computationally expensive. Transductive SVM performs well with linear or non-linear data and we see an example of training a TSVM with a Gaussian kernel on the UCI Breast Cancer dataset using the `JKernelMachines` library. We present experiments comparing SSL models using the graphical Java tool KEEL in the concluding part of the SSL portion of the chapter.

We introduced active learning (AL) in the second half of the chapter. In this type of learning, various strategies are used to query the unlabeled portion of the dataset in order to present the expert with examples that will prove most effective in learning from the entire dataset. As the expert, or oracle, provides the labels to the selected instances, the learner steadily improves its ability to generalize. The techniques of AL are characterized by the choice of classifier, or committee of classifiers, and importantly, on the querying strategy chosen. These strategies include uncertainty sampling, where the instances with the least confidence are queries, version sampling, where a subset of the hypotheses that explain the training data are selected, and data distribution sampling, which involves improving the model by selections that would decrease the generalization error. We presented a case study using the UCI abalone dataset to demonstrate active learning in practice. The tool used here is the JCLAL Java framework for active learning.

References

1. Yarowsky, D (1995). *Unsupervised word sense disambiguation rivaling supervised methods*. Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (pp. 189–196)
2. Blum, A., and Mitchell, T (1998). *Combining labeled and unlabeled data with co-training*. COLT: Proceedings of the Workshop on Computational Learning Theory.
3. Demiriz, A., Bennett, K., and Embrechts, M (1999). *Semi-supervised clustering using genetic algorithms*. Proceedings of Artificial Neural Networks in Engineering.
4. Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux (2006). *Label Propagation and Quadratic Criterion*. In Semi-Supervised Learning, pp. 193-216
5. T. Joachims (1998). *Transductive Inference for Text Classification using Support Vector Machines*, ICML.
6. B. Settles (2008). *Curious Machines: Active Learning with Structured Instances*. PhD thesis, University of Wisconsin–Madison.
7. D. Angluin (1988). *Queries and concept learning*. Machine Learning, 2:319–342.
8. D. Lewis and W. Gale (1994). *A sequential algorithm for training text classifiers*. In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval, pages 3–12. ACM/Springer.
9. H.S. Seung, M. Opper, and H. Sompolinsky (1992). *Query by committee*. In Proceedings of the ACM Workshop on Computational Learning Theory, pages 287–294.
10. D. Cohn, L. Atlas, R. Ladner, M. El-Sharkawi, R. Marks II, M. Aggoune, and D. Park (1992). *Training connectionist networks with queries and selective sampling*. In Advances in Neural Information Processing Systems (NIPS). Morgan Kaufmann.
11. D. Cohn, L. Atlas, and R. Ladner (1994). *Improving generalization with active learning*. Machine Learning, 15(2):201–221.
12. D. Lewis and J. Catlett (1994). *Heterogeneous uncertainty sampling for supervised learning*. In Proceedings of the International Conference on Machine Learning (ICML), pages 148–156. Morgan Kaufmann.
13. S. Dasgupta, A. Kalai, and C. Monteleoni (2005). *Analysis of perceptron-based active learning*. In Proceedings of the Conference on Learning Theory (COLT), pages 249–263. Springer.
14. S. Dasgupta, D. Hsu, and C. Monteleoni (2008). *A general agnostic active*

- learning algorithm.* In Advances in Neural Information Processing Systems (NIPS), volume 20, pages 353–360. MIT Press.
15. T. Mitchell (1982). *Generalization as search*. Artificial Intelligence, 18:203–226.

Chapter 5. Real-Time Stream Machine Learning

In [Chapter 2, Practical Approach to Real-World Supervised Learning](#), [Chapter 3, Unsupervised Machine Learning Techniques](#), and [Chapter 4, Semi-Supervised and Active Learning](#), we discussed various techniques of classification, clustering, outlier detection, semi-supervised, and active learning. The form of learning done from existing or historic data is traditionally known as batch learning.

All of these algorithms or techniques assume three things, namely:

- Finite training data is available to build different models.
- The learned model will be static; that is, patterns won't change.
- The data distribution also will remain the same.

In many real-world data scenarios, there is either no training data available a priori or the data is dynamic in nature; that is, changes continuously with respect to time. Many real-world applications may also have data which has a transient nature to it and comes in high velocity or volume such as IoT sensor information, network monitoring, and Twitter feeds. The requirement here is to learn from the instance immediately and then update the learning.

The nature of dynamic data and potentially changing distribution renders existing batch-based algorithms and techniques generally unsuitable for such tasks. This gave rise to adaptable or updatable or incremental learning algorithms in machine learning. These techniques can be applied to continuously learn from the data streams. In many cases, the disadvantage of learning from Big Data due to size and the need to fit the entire data into memory can also be overcome by converting the Big Data learning problem into an incremental learning problem and inspecting one example at a time.

In this chapter, we will discuss the assumptions and discuss different techniques in supervised and unsupervised learning that facilitate real-time or stream machine learning. We will use the open source library **Massive Online Analysis (MOA)** for performing a real-world case study.

The major sections of this chapter are:

- Assumptions and mathematical notation.
- Basic stream processing and computational techniques. A discussion of stream computations, sliding windows including the ADWIN algorithm, and sampling.
- Concept drift and drift detection: Introduces learning evolving systems and data management, detection methods, and implicit and explicit adaptation.
- Incremental supervised learning: A discussion of learning from labeled stream data, modeling techniques including linear, non-linear, and ensemble algorithms. This is followed by validation, evaluation, and model comparison methods.
- Incremental unsupervised learning: Clustering techniques similar to those discussed in [Chapter 3](#), *Unsupervised Machine Learning Techniques*, including validation and evaluation techniques.
- Unsupervised learning using outlier detection: Partition-based and distance-based, and the validation and evaluation techniques used.
- Case study for stream-based learning: Introduces the MOA framework, presents the business problem, feature analysis, mapping to machine learning blueprint; describes the experiments, and concludes with the presentation and analysis of the results.

Assumptions and mathematical notations

There are some key assumptions made by many stream machine learning techniques and we will state them explicitly here:

- The number of features in the data is fixed.
- Data has small to medium dimensions, or number of features, typically in the hundreds.
- The number of examples or training data can be infinite or very large, typically in the millions or billions.
- The number of class labels in supervised learning or clusters are small and finite, typically less than 10.
- Normally, there is an upper bound on memory; that is, we cannot fit all the data in memory, so learning from data must take this into account, especially lazy learners such as K-Nearest-Neighbors.
- Normally, there is an upper bound on the time taken to process the event or the data, typically a few milliseconds.
- The patterns or the distributions in the data can be evolving over time.
- Learning algorithms must converge to a solution in finite time.

Let $D_t = \{\mathbf{x}_i, y_i : y = f(x)\}$ be the given data available at time $t \in \{1, 2, \dots, i\}$.

An incremental learning algorithm produces sequences of models/hypotheses $\{\dots, G_{j-1}, G_j, G_{j+1}\dots\}$ for the sequence of data $\{\dots, D_{j-1}, D_j, D_{j+1}\dots\}$ and model/hypothesis G_i depends only on the previous hypothesis G_{i-1} and current data D_i .

Basic stream processing and computational techniques

We will now describe some basic computations that can be performed on the stream of data. If we must run summary operations such as aggregations or histograms with limits on memory and speed, we can be sure that some kind of trade-off will be needed. Two well-known types of approximations in these situations are:

- ϵ Approximation: The computation is close to the exact value within the fraction ϵ of error.
- (ϵ, δ) Approximation: The computation is close to the exact value within $1 \pm \epsilon$ with probability within $1 - \delta$.

Stream computations

We will illustrate some basic computations and aggregations to highlight the difference between batch and stream-based calculations when we must compute basic operations with constraints on memory and yet consider the entire data:

- **Frequency count or point queries:** The generic technique of Count-Min Sketch has been successfully applied to perform various summarizations on the data streams. The primary technique is creating a window of size $w \times d$. Then, given a desired probability (δ) and admissible error (ϵ), the size of data in

$$d = \left\lceil \log\left(\frac{1}{\delta}\right) \right\rceil$$

memory can be created using $w = 2/\epsilon$ and associated with each row is a hash function: $h(\cdot)$. This uniformly transforms a value x to a value in the interval $[1, 2 \dots w]$. This method of lookup and updates can be used for performing point queries of values or dot products or frequency counts.

- **Distinct count:** The generic technique of Hash-Sketch can be used to perform "distinct values" queries or counts. Given the domain of incoming stream values $x \in [0, 1, 2, \dots, N-1]$, the hash function $h(x)$ maps the values uniformly across $[0, 1, \dots, 2L-1]$, where $L = O(\log N)$.
- **Mean:** Computing the mean without the need for storing all the values is very useful and is normally employed using a recursive method where only the number of observations (n) and sum of values seen so far ($\sum x_n$) is needed:

$$\bar{x}_n = \frac{x_n + (n-1) \times \bar{x}_{n-1}}{n}$$

- **Standard deviation:** Like the mean, standard deviation can be computed using the memoryless option with only the number of observations (n), sum of values seen so far ($\sum x_n$), and sum of squares of the values ($\sum x_n^2$):

$$\sigma_n = \sqrt{\frac{\left(\sum x_n^2 - \frac{(\sum x_n)^2}{n} \right)}{n-1}}$$

- **Correlation coefficient:** Given a stream of two different values, many algorithms need to compute the correlation coefficient between the two which can be done by maintaining the running sum of each stream ($\sum x_n$ and $\sum y_n$), the sum of squared values ($\sum x_n^2$ and $\sum y_n^2$), and the cross-product ($\sum x_n \times y_n$). The correlation is given by:

$$corr(x, y) = \frac{\sum x_n \times y_n - \frac{\sum x_n \times y_n}{n}}{\sqrt{\sum x_n^2 - \frac{(\sum x_n)^2}{n}} \sqrt{\sum y_n^2 - \frac{(\sum y_n)^2}{n}}}$$

Sliding windows

Often, you don't need the entire data for computing statistics or summarizations but only the "recent past". In such cases, sliding window techniques are used to calculate summary statistics by keeping the window size either fixed or adaptable and moving it over the recent past.

ADaptable sliding WINdow (ADWIN) is one well-known technique used to detect change as well as estimating values needed in the computation. The idea behind ADWIN is to keep a variable-length window of last seen values with the characteristic that the window has a maximum length statistically consistent with the fact that there has been no change in the average value within the window. In other words, the older values are dropped if and only if a new incoming value would change the average. This has the two-fold advantage of recording change and maintaining the dynamic value, such as aggregate, over the recent streams. The determination of the subjective notion "large enough" for dropping items can be determined using the well-known Hoeffding bound as:

$$\varepsilon_{cut} = \sqrt{\left(\frac{1}{2m} \ln \frac{4|W|}{\delta} \right)}$$

Here $m = \frac{2}{\left(\frac{1}{|W_0|} + \frac{1}{|W_1|} \right)}$ is the harmonic mean between two windows W_0 and W_1 of size $|W_0|$ and $|W_1|$ respectively, with W_1 containing the more recent elements.

Further, let $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ be the respective calculated averages.

The algorithm can be generalized as:

1. ADWIN (x : *inputstream*, δ : *confidence*)
2. init (W) //Initialize Window W
3. while (x) {

- W \leftarrow W \cup { x_t } //add new instance x_t to the head of Window W
- 4. repeat W \leftarrow W - x_{old} //drop elements from tail of the window
- 5. until $|\hat{\mu}W_0 - \hat{\mu}W_1| < \varepsilon_{cut}$ holds for every split of W
- 6. output $\hat{\mu}V$
- 7. }

ADWIN has also shown that it provides theoretical bounds on false positives and false negatives, which makes it a very promising technique to use.

Sampling

In many stream-based algorithms, there is a need to reduce the data or select a subset of data for analysis. The normal methodology of sampling on the whole data must be augmented for stream-based data.

The key concerns in sampling that must be addressed are how unbiased the samples are and how representative they are of the population from which streams are being generated. In a non-streaming environment, this depends completely on the sample size and the sampling method. Uniform random sampling ([Chapter 2, Practical Approach to Real-World Supervised Learning](#)) is one of the most well-known techniques employed to reduce the data in the batch data world. The reservoir sampling technique is considered to be a very effective way of reducing the data given the memory constraints.

The basic idea of reservoir sampling is to keep a reservoir or sample of fixed size, say k , and every element that enters the stream has a probability k/n of replacing an older element in the reservoir. The detailed algorithm is shown here:

```
ReservoirSampling(x:inputstream, k:sizeOfReservoir)
//add first k elements to reservoir
for(i = 0; i < k; i++)
    addToReservoir(x)
while (x) {
    for(i = 0; i < k; i++)
        //flip a coin to get random integer
        r = randomInteger[1..n]
        if(r ≤ k) {
            //move it inside the reservoir
            addToReservoir(x)
            //delete an instance randomly from reservoir
            position = randomInteger[1..k]
            removeInstance(position)
        }
}
```

There are extensions to these such as the Min-wise Sampling and Load Shedding that overcome some issues associated with the base method.

Concept drift and drift detection

As discussed in the introduction of the chapter, the dynamic nature of infinite streams stands in direct opposition to the basic principles of stationary learning; that is, that the distribution of the data or patterns remain constant. Although there can be changes that are *swift* or *abrupt*, the discussion here is around slow, gradual changes. These slow, gradual changes are fairly hard to detect and separating the changes from the noise becomes tougher still:

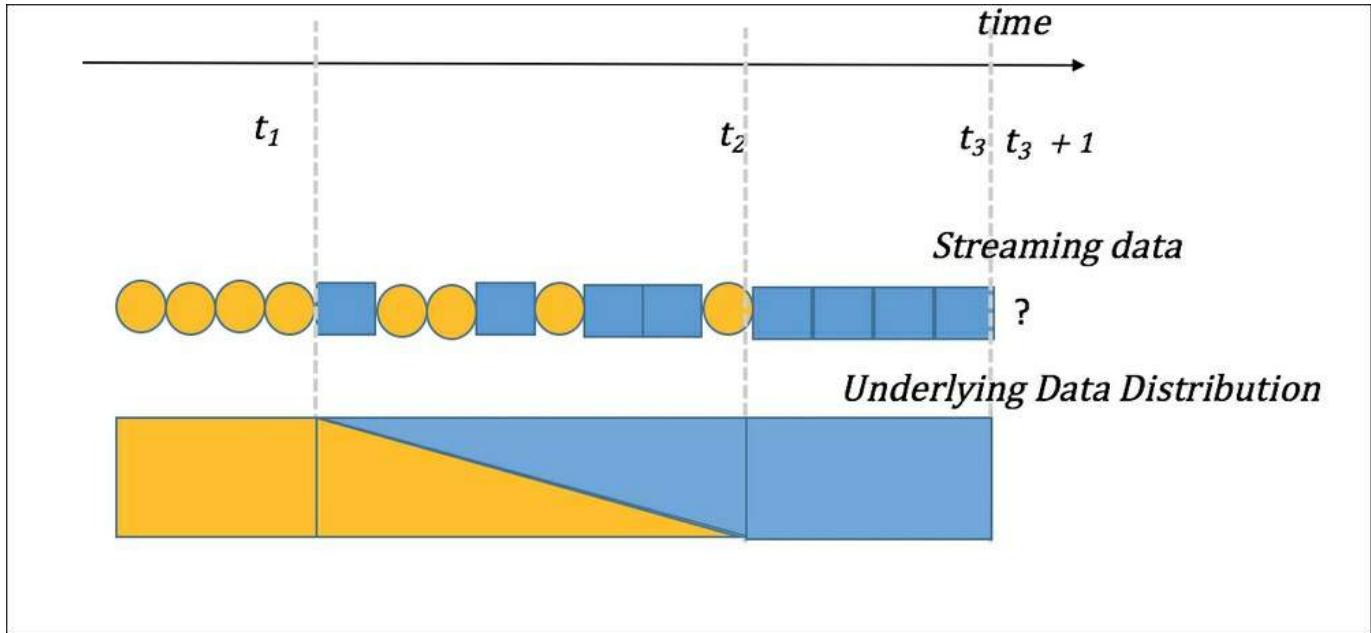


Figure 1 Concept drift illustrated by the gradual change in color from yellow to blue in the bottom panel. Sampled data reflects underlying change in data distribution, which must be detected and a new model learned.

There have been several techniques described in various studies in the last two decades that can be categorized as shown in the following figure:

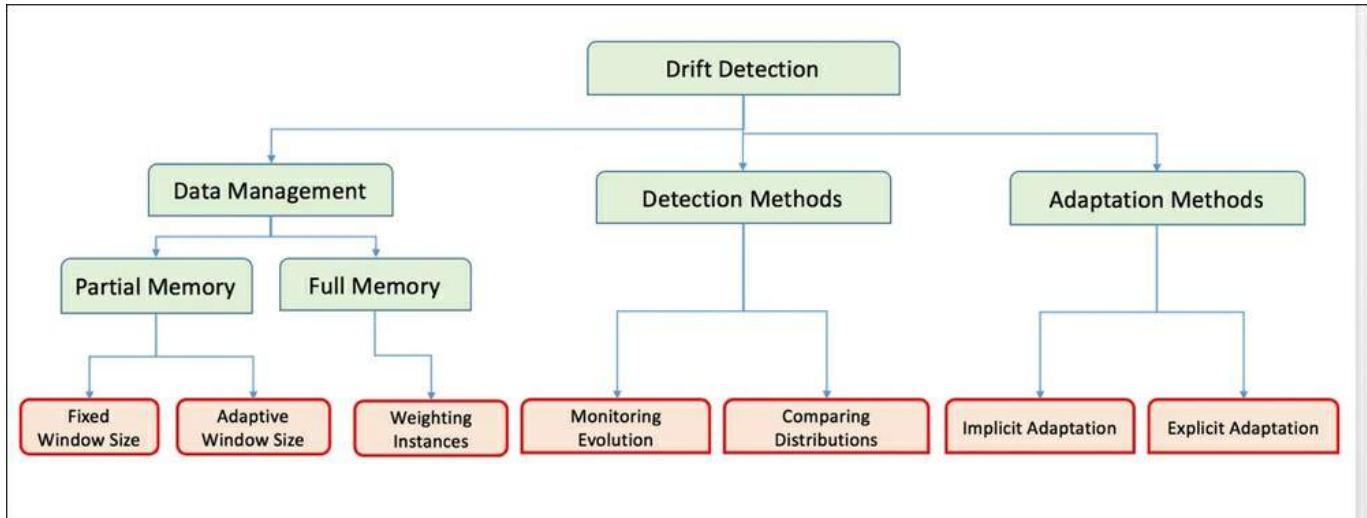


Figure 2 Categories of drift detection techniques

Data management

The main idea is to manage a model in memory that is consistent with the dynamic nature of the data.

Partial memory

These techniques use the most recently used data in a memory buffer to learn or derive summary information. The key question as discussed previously is: what is the right window size to be effective in detecting the change and learning effectively? In Fixed Window Size based techniques, we use the idea of a queue where a new instance with a recent timestamp comes in and the one with the oldest is evicted. The window thus contains all the recent enough examples and the size is generally chosen based on physical availability of memory and size of data elements in the queue. In Adaptive Window Size, the queue is used in conjunction with a detection algorithm. When the detection algorithm indicates signs of drifts based on performance evaluation, the window size can be reduced to effectively remove old examples which no longer help the model.

Full memory

The idea is to store sufficient statistics over all the examples or data seen. One way to do this is to put weights on the data and weights decay over time. Exponential weighting using the rate factor given by λ can be very effective:

$$w_\lambda(x) = \exp(-\lambda * i)$$

Detection methods

Given the probability $P(X)$ that the given data is observed, the probability of patterns/class $P(C)$, and the probability of data given class $P(X|C)$ —which is the model—the detection method can be divided into two categories, at a high level:

- Monitoring evolution or performance of the model, classifier, or $P(C|X)$
- Monitoring distributions from the environment or observing $P(X)$, $P(C)$, and $P(X|C)$

Monitoring model evolution

Although this method is based on the assumption that all the learning of models is stationary and data is coming from **independent, identical distributions (i.i.d.)**, which doesn't hold true in many applications, it has nevertheless been shown to be effective. Some of the well-known techniques are described next.

Widmer and Kubat

This is one of the earliest methods which observed the error rates or misclassification rates and the changes to the model such as tree structures due to new branches, for instance. Using these and known thresholds, the learning window size is increased or decreased.

Drift Detection Method or DDM

This method assumes that the parameter being observed, such as the classifier labeling things correctly or incorrectly, is a binary random variable which follows a binomial distribution. It assumes probability of misclassification at probability p_i

with standard deviation of $s_i = \sqrt{p_i(1-p_i)/i}$ where the values are computed at the i^{th} point in the sequence. The method then uses two levels:

- Warning level: When $p_i + s_i \geq p_{\min} + 2 * s_{\min}$
- Detection level: When $p_i + s_i \geq p_{\min} + 3 * s_{\min}$

All the examples between the "warning" and "detection" levels are used to train a new classifier that will replace the "non-performing" classifier when the "detection" level is reached.

Early Drift Detection Method or EDDM

EDDM uses the same technique as DDM but with a slight modification. It uses classification rate (that is, recall) rather than error rate (1 – accuracy) and uses the distance between the number of right predictions and two wrong predictions to change the levels.

EDDM computes the mean distance between the two errors p_i' and standard deviation between the two s_i' . The levels are then:

- Warning level: $(p_i' + 2 * s_i') / (p_{\max}' + 2 * s_{\max}') < \alpha$
- Detection level: $(p_i' + 2 * s_i') / (p_{\max}' + 2 * s_{\max}') < \beta$

The parameters α and β are normally tuned by the user to something around 90% and 95% respectively.

Monitoring distribution changes

When there are no models or classifiers to detect changes, we apply techniques that use some form of statistical tests for monitoring distribution changes. These tests are

used to identify the distribution changes. Owing to assumptions, whether parametric or non-parametric, and different biases, it is difficult to say concretely which works best. Here we provide some of the well-known statistical tests.

Welch's t test

This is an adaptation of the Student t test with two samples. The test is adapted to take two windows of size N_1 and N_2 with means \bar{X}_1 and \bar{X}_2 and variances s_1^2 and s_2^2 to compute the p value and use that to reject or accept the null hypothesis:

$$pvalue = \left(\bar{X}_1 - \bar{X}_2 \right) / \left(\sqrt{s_1^2/N + s_2^2/N} \right)$$

Kolmogorov-Smirnov's test

This statistical test is normally used to compare distances between two distributions and validate if they are below certain thresholds or not. This can be adapted for change detection by using windows of two different sample sizes, N_1 and N_2 , with different cumulative distribution functions, $F_1(x)$ and $F_2(x)$, KS distance:

$$ksDistance = \max_x |F_1(x) - F_2(x)|$$

The null hypothesis, which assumes the two distributions are similar, is rejected with confidence of α if and only if $ksDistance * \sqrt{N_1 N_2 / (N_1 + N_2)} > K_\alpha$, which is obtained by a lookup in the Kolmogorov-Smirnov's table.

CUSUM and Page-Hinckley test

The **cumulative sum (CUSUM)** is designed to indicate when the mean of the input is significantly different from zero:

$$g_0 = 0, g_t = \max(0, g_{t-1}) + \epsilon_t - v$$

We raise change detection when $g_t > h$, where (h, v) are user-defined parameters. Note that the CUSUM test is memoryless and is one-sided or asymmetric, detecting only the increase.

The Page Hinckley test is similar to CUSUM with a small variation as shown here:

$$g_0 = 0, g_t = g_{t-1} + \epsilon_t - v$$

For increasing and decreasing the values, we use $G_t = \min(g_t, G_{t-1})$ or $G_t = \max(g_t, G_{t-1})$, and $G_t - g_t > h$ for change detection.

Adaptation methods

Explicit and implicit adaptation are the two well-known techniques for adapting to environment changes when a change is detected.

Explicit adaptation

In explicit adaptation, an additional technique from among the following is used:

- Retrain the model from scratch with new data so the previous model or data does not impact the new model
- Update the model with the changes or new data such that the transition is smooth—assumes changes are gradual and not drastic
- Create a sequence or ensemble of models that are learned over time—when a collaborative approach is better than any single model

Implicit adaptation

In implicit adaptation, we generally use ensemble algorithms/models to adapt to the concept change. This can mean using different combinations ranging from a single classifier, to predicting in the ensemble, to using ADWIN for adaptive window-based with the classifier—all fall within the choices for implicit adaptation.

Incremental supervised learning

This section introduces several techniques used to learn from stream data when the true label for each instance is available. In particular, we present linear, non-linear, and ensemble-based algorithms adapted to incremental learning, as well as methods required in the evaluation and validation of these models, keeping in mind that learning is constrained by limits on memory and CPU time.

Modeling techniques

The modeling techniques are divided into linear algorithms, non-linear algorithms, and ensemble methods.

Linear algorithms

The linear methods described here require little to no adaptation to handle stream data.

Online linear models with loss functions

Different loss functions such as hinge, logistic, and squared error can be used in this algorithm.

Inputs and outputs

Only numeric features are used in these methods. The choice of loss function l and learning rate λ at which to apply the weight updates are taken as input parameters. The output is typically updatable models that give predictions accompanied by confidence values.

How does it work?

The basic algorithm assumes linear weight combinations similar to linear/logistic regression explained in [Chapter 2, Practical Approach to Real-World Supervised Learning](#). The stream or online learning algorithm can be summed up as:

1. for($t=1,2,\dots,T$) do
 1. $\mathbf{x}_t = \text{receive}();$ // receive the data
 2. $\hat{y}_t = \text{sgn}(f(\mathbf{x}_t, \mathbf{w}_t));$ //predict the label
 3. $y_t = \text{obtainTrueLabel}();$ // get the true label
 4. $loss = l(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t));$ // calculate the loss
 5. if($l(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t)) > 0$) then
 6. $\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda * \Delta(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t));$ //update the weights
 7. end
2. end

Different loss functions can be plugged in based on types of problems; some of the well-known types are shown here:

- Classification:

- Hinge loss: $l(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t)) = \max(0, 1 - yf(\mathbf{x}_t, \mathbf{w}_t))$

$$\ell(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t)) = \frac{1}{\ln 2} \ln \left(1 + e^{-yf(\mathbf{x}_t, \mathbf{w}_t)} \right)$$

- Logistic loss:

- Regression:

- Squared loss: $\ell(\mathbf{w}_t, (\mathbf{x}_t, \mathbf{w}_t)) = (1 - yf(\mathbf{x}_t, \mathbf{w}_t))^2$

Stochastic Gradient Descent (SGD) can be thought of as changing the weights to minimize the squared loss as in the preceding loss functions but going in the direction of the gradient with each example. The update of weights can be described as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda * \Delta J * \mathbf{x}_t$$

$$\Delta J = -(y_t - \hat{y}_t) \hat{y}_t (1 - \hat{y}_t)$$

Advantages and limitations

Online linear models have similar advantages and disadvantages as the linear models described in [Chapter 2, Practical Approach to Real-World Supervised Learning](#):

- Interpretable to some level as the weights of each features give insights on the impact of each feature
- Assumes linear relationship, additive and uncorrelated features, and hence doesn't model complex non-linear real-world data
- Very susceptible to outliers in the data
- Very fast and normally one of the first algorithms to try or baseline

Online Naïve Bayes

Bayes theorem is applied to get predictions as the posterior probability, given for an

m dimensional input:

$$P(y | \mathbf{x}) = \frac{P(y) * \prod_{j=1}^m P(X_j | Y)}{P(\mathbf{x})}$$

Inputs and outputs

Online Naïve Bayes can accept both categorical and continuous inputs. The categorical features are easier, as the algorithm must maintain counts for each class while computing the $P(X_j|Y)$ probability for each feature given the class. For continuous features, we must either assume a distribution, such as Gaussian, or to compute online Kernel Density estimates in an incremental way or discretize the numeric features incrementally. The outputs are updatable models and can predict the class accompanied by confidence value. Being probabilistic models, they have better confidence scores distributed between 0 and 1.

How does it work?

1. for($t = 1, 2, \dots, T$) do
 1. $\mathbf{x}_t = receive();$ // receive the data
 2. $incrementCounters(\mathbf{x}_t);$ //update the $P(X_j|Y)$
 3. $\langle \hat{y}_t, confidence \rangle = applyBayesRule(\mathbf{x}_t)$ //posterior probability
2. end

Advantages and limitations

- This is the fastest algorithm and has a low memory footprint as well as computation cost. It is very popular among online or fast learners.
- Assumes distribution or some biases on the numeric features that can impact the predictive quality.

Non-linear algorithms

One of the most popular non-linear stream learning classifiers in use is the Hoeffding Tree. In the following subsections, the notion of the Hoeffding bound is introduced, followed by the algorithm itself.

Hoeffding trees or very fast decision trees (VFDT)

The key idea behind **Hoeffding Trees (HT)** is the concept of the Hoeffding bound. Given a real-valued random variable \mathbf{x} whose range of values has size \mathbf{R} , suppose we have \mathbf{n} independent observations of \mathbf{x} and compute the mean as $\bar{\mathbf{x}}$.

The Hoeffding bound states that, with a probability of $1 - \delta$, the actual mean of the

$$\varepsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$$

variable \mathbf{x} is at least $\bar{\mathbf{x}} - \varepsilon$ where

The Hoeffding bound is independent of the probability distribution generating the samples and gives a good approximation with just \mathbf{n} examples.

The idea of the Hoeffding bound is used in the leaf expansion. If x_1 is the most informative feature and x_2 ranks second, then split using a user-defined split function $G(\cdot)$ in a way such that:

$$G(x_1) - G(x_2) > \varepsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$$

Inputs and outputs

Both categorical and continuous data can be part of the data input. Continuous features are discretized in many implementations. The desired probability parameter $1 - \delta$ and the split function common to Decision Trees $G(\cdot)$ becomes part of the input. The output is the interpretable Decision Tree model and can predict/learn with class and confidence values.

How does it work?

`HoeffdingTree(x:inputstream,G(.):splitFunction,δ:probabilityBound)`

1. Let HT be a tree with single leaf(root)
2. `InitCounts(n_{ijk} , root)`
3. `for(t=1,2,...T) do //all data from stream`
 1. `$\mathbf{x}_t = receive();$ //receive the data`

2. $y_t = obtainTrueLabel();$ //get the true label
3. HTGrow((\mathbf{x}_t, y_t), \mathbf{HT} , δ)
4. end

HTGrow((\mathbf{x}_t, y_t), \mathbf{HT} , $G(\cdot)$, δ)

1. $l = sort((\mathbf{x}_t, y_t), \mathbf{HT});$ //sort the data to leaf l using HT
2. $updateCounts(n_{ijk}, l);$ // update the counts at leaf l
3. $if(examplesSoFarNotOfSameClass());$ // check if there are multiple classes
4. $computeForEachFeature(G(\cdot))$

$$if(G(x_1) - G(x_2) > \varepsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}; // difference between two features$$

$$splitLeaf(x_1)$$

$$for each branch$$

$$createNewLeaf$$
 1. initializeCounts();

Hoeffding Trees have interesting properties, such as:

- They are a robust low variance model
- They exhibit lower overfitting
- Theoretical guarantees with high probability on the error rate exist due to Hoeffding bounds

There are variations to Hoeffding Trees that can adapt concept drift, known as Concept-Adapting VFDT. They use the sliding window concept on the stream. Each node in the decision tree keeps sufficient statistics; based on the Hoeffding test, an alternate subtree is grown and swapped in when the accuracy is better.

Advantages and limitations

The advantages and limitations are as follows:

- The basic HT has issues with attributes being close to the chosen ϵ and breaks the ties. Deciding the number of attributes at any node is again an issue. Some of it is resolved in VFDT.

- Memory constraints on expansion of the trees as well as time spent on an instance becomes an issue as the tree changes.
- VFDT has issues with changes in patterns and CVFDT tries to overcome these as discussed previously. It is one of the most elegant, fast, interpretable algorithms for real-time and big data.

Ensemble algorithms

The idea behind ensemble learning is similar to batch supervised learning where multiple algorithms are trained and combined in some form to predict unseen data. The same benefits accrue even in the online setting from different approaches to ensembles; for example, using multiple algorithms of different types, using models of similar type but with different parameters or sampled data, all so that different search spaces or patterns are found and the total error is reduced.

Weighted majority algorithm

The **weighted majority algorithm (WMA)** trains a set of base classifiers and combines their votes, weighted in some way, and makes predictions based on the majority.

Inputs and outputs

The constraint on types of inputs (categorical only, continuous only, or mixed) depends on the chosen base classifiers. The interpretability of the model depends on the base model(s) selected but it is difficult to interpret the outputs of a combination of models. The weights for each model get updated by a factor (β) per example/instance when the prediction is incorrect. The combination of weights and models can give some idea of interpretability.

How does it work?

WeightedMajorityAlgorithm(x : inputstream, hm : m learner models)

1. *initializeWeights(w_i)*
2. *for*($t=1, 2, \dots T$) *do*
 1. $x_t = receive();$
 2. *foreach* model $h_k \in h$
 3. $y_i \leftarrow h_k(x_t);$
3. *if* $\sum_{i:y_i=1} w_i \geq \sum_{i:y_i=0} w_i$ *then* $\hat{y} \leftarrow 1$

```

4. else  $\hat{y} \leftarrow 0$ 
5. if  $y$  is known then
   1. for  $i = 1$  to  $m$  do
   2. if  $y_i \neq y$  then
   3.  $w_i \leftarrow w_i * \beta$ 

   end if

end for

6. end

```

Advantages and limitations

The advantages and limitations are as follows:

- WMA has simple implementation and theoretic bounds on ensemble errors
- The difficulty is to choose the right base algorithm as the model and the number of models in the pool

Online Bagging algorithm

As we saw in the chapter on supervised learning, the bagging algorithm, which creates different samples from the training sets and uses multiple algorithms to learn and predict, reduces the variance and is very effective in learning.

Inputs and outputs

The constraint on the types of inputs (categorical only, continuous only, or mixed) depends on the chosen base classifiers. The base classifier algorithm with parameter choices corresponding to the algorithm are also the inputs. The output is the learned model that can predict the class/confidence based on the classifier chosen.

How does it work?

The basic batch bagging algorithm requires the entire data to be available to create different samples and provide these samples to different classifiers. Oza's Online Bagging algorithm changes this constraint and makes it possible to learn from unbounded data streams. Based on sampling, each training instance in the original algorithm gets replicated many times and each base model is trained with k copies of the original instances where:

$$P(k) = \exp(-1)/k!$$

This is equivalent to taking one training example and choosing for each classifier $k \sim \text{Poisson}(1)$ and updating the base classifier k times. Thus, the dependency on the number of examples is removed and the algorithm can run on an infinite stream:

OnlineBagging(x : inputstream, h_m : m learner models)

1. initialize base models h_m for all $m \in \{1, 2, \dots, M\}$
2. for($t=1, 2, \dots, T$) do
 1. $x_t = \text{receive}();$
 2. foreach model $m = \{1, 2, \dots, M\}$

$$w = \text{Poisson}(1)$$

$$\text{updateModel}(h_m, w, x_t)$$
3. end
3. return
4. $\mathbf{h}_{final} = \arg\max_{y \in Y} \sum_{t=1}^T \mathbf{I}(\mathbf{h}(\mathbf{x})_t = y)$

Advantages and limitations

The advantages and limitations are as follows:

- It has been empirically shown to be one of the most successful online or stream algorithms.
- The weight must be given to the data instance without looking at the other instances; this reduces the choices of different weighting schemes which are available in batch and are good in model performance.

The performance is entirely determined by the choice of the M learners—the type of learner used for the problem domain. We can only decide on this choice by adopting different validation techniques described in the section on model validation techniques.

Online Boosting algorithm

The supervised boosting algorithm takes many *weak learners* whose accuracy is slightly greater than random and combines them to produce a strong learner by

iteratively sampling the misclassified examples. The concept is identical in Oza's Online Boosting algorithm with modification done for a continuous data stream.

Inputs and outputs

The constraint on types of inputs (categorical only, continuous only, or mixed) depends on the chosen base classifiers. The base classifier algorithms and their respective parameters are inputs. The output is the learned model that can predict the class/confidence based on the classifier chosen.

How does it work?

The modification of batch boosting to online boosting is done as follows:

1. Keep two sets of weights for M base models, λ^C is a vector of dimension M which carries the sum of weights of correctly classified instances, and λ^W is a vector of dimension M , which carries the sum of weights of incorrectly classified instances.
2. The weights are initialized to 1.
3. Given a new instance (\mathbf{x}_t, y_t) , the algorithm goes through the iterations of updating the base models.
4. For each base model, the following steps are repeated:
 1. For the first iteration, $k = Poisson(\lambda)$ is set and the learning classifier updates the algorithm (denoted here by h_1) k times using (\mathbf{x}_t, y_t) :
 2. If h_1 incorrectly classifies the instance, the λ^{W1} is incremented, ϵ_1 , the weighted fraction, incorrectly classified by h_1 , is computed and the weight of the example is multiplied by $1/2 \epsilon_1$.

Advantages and limitations

The advantages and limitations are as follows:

- Again, the performance is determined by the choice of the multiple learners, their types and the particular domain of the problem. The different methods described in the section on model validation techniques help us in choosing the learners.
- Oza's Online Boosting has been shown theoretically and empirically not to be "lossless"; that is, the model is different compared to its batch version. Thus, it suffers from performance issues and different extensions have been studied in recent years to improve performance.

Validation, evaluation, and comparisons in online setting

In contrast to the modes of machine learning we saw in the previous chapters, stream learning presents unique challenges to performing the core steps of validation and evaluation. The fact that we are no longer dealing with batch data means the standard techniques for validation evaluation and model comparison must be adapted for incremental learning.

Model validation techniques

In the off-line or the batch setting, we discussed various methods of tuning the parameters of the algorithm or testing the generalization capability of the algorithms as a counter-measure against overfitting. Some of the techniques in the batch labeled data, such as cross-validation, are not directly applicable in the online or stream settings. The most common techniques used in online or stream settings are given next.

Prequential evaluation

The prequential evaluation method is a method where instances are provided to the algorithm and the output prediction of the algorithm is then measured in comparison with the actual label using a loss function. Thus, the algorithm is always tested on the unseen data and needs no "holdout" data to estimate the generalization. The prequential error is computed based on the sum of the accumulated loss function between actual values and predicted values, given by:

$$P = \sum_{i=1}^N l(y_i, \hat{y}_i)$$

Three variations of basic prequential evaluation are done for better estimation on changing data, which are:

- Using Landmark Window (basic)
- Using Sliding Window
- Using forgetting mechanism

The last two methods are extensions of previously described techniques where you put weights or fading factors on the predictions that reduce over time.

Holdout evaluation

This is the extension of the holdout mechanism or "independent test set" methodology of the batch learning. Here the total labeled set or stream data is separated into training and testing sets, either based on some fixed intervals or the number of examples/instances the algorithm has seen. Imagine a continuous stream

of data and we place well-known intervals at $t = t_{start+trainingIntervalEnd}$ and

$t = t_{trainingIntervalEnd+testingIntervalEnd}$ to compare the evaluation metrics, as discussed in next section, for performance.

Controlled permutations

The issue with the aforementioned mechanisms is that they provide "average" behavior over time and can mask some basic issues such as the algorithm doing well at the start and very poorly at the end due to drift, for example. The advantage of the preceding methods is that they can be applied to real incoming streams to get estimates. One way to overcome the disadvantage is to create different random sets of the data where the order is shuffled a bit while maintaining the proximity in time and the evaluation is done over a number of these random sets.

Evaluation criteria

Most of the evaluation criteria are the same as described in the chapter on supervised learning and should be chosen based on the business problem, the mapping of the business problem to the machine learning techniques, and on the benefits derived. In this section, the most commonly used online supervised learning evaluation criteria are summarized for the reader:

- **Accuracy:** A measure of getting the true positives and true negatives correctly classified by the learning algorithm:

$$Accuracy = \frac{\sum True\ Positive + \sum True\ Negative}{\sum Total\ Data}$$

- **Balanced accuracy:** When the classes are imbalanced, balanced accuracy is often used as a measure. Balanced accuracy is an arithmetic mean of specificity and sensitivity. It can be also thought of as accuracy when positive and negative instances are drawn from the same probability in a binary classification problem.
- **Area under the ROC curve (AUC):** Area under the ROC curve gives a good measure of generalization of the algorithm. Closer to 1.0 means the algorithm has good generalization capability while close to 0.5 means it is closer to a random guess.
- **Kappa statistic (K):** The Kappa statistic is used to measure the observed accuracy with the expected accuracy of random guessing in the classification. In online learning, the Kappa statistic is used by computing the prequential accuracy (p_o) and the random classifier accuracy (p_c) and is given by:

$$K = \frac{(p_o - p_c)}{(1 - p_c)}$$

- **Kappa Plus statistic:** The Kappa Plus statistic is a modification to the Kappa statistic obtained by replacing the random classifier by the persistent classifier. The persistent classifier is a classifier which predicts the next instance based on the label or outcome of the previous instance.

When considering "drift" or change in the concept as discussed earlier, in addition to these standard measures, some well-known measures given are used to give a quantitative measure:

- **Probability of true change detection:** Usually, measured with synthetic data or data where the changes are known. It gives the ability of the learning algorithm to detect the change.
- **Probability of false alarm:** Instead of using the False Positive Rate in the off-line setting, the online setting uses the inverse of *time to detection or the average run length* which is computed using the expected time between false positive detections.
- **Delay of detection:** This is measured as the time required, terms of instances, to identify the drift.

Comparing algorithms and metrics

When comparing two classifiers or learners in online settings, the usual mechanism is the method of taking a performance metric, such as the error rate, and using a statistical test adapted to online learning. Two widely used methods are described next:

- **McNemar test:** McNemar's test is a non-parametric statistical test normally employed to compare two classifiers' evaluation metrics, such as "error rate", by storing simple statistics about the two classifiers. By computing statistic a , the number of correctly classified points by one algorithm that are incorrectly classified by the other, and statistic b , which is the inverse, we obtain the McNemar's Test as:

$$\chi^2$$

The test follows a χ^2 distribution and the p-value can be used to check for statistical significance.

- **Nemenyi test:** When there are multiple algorithms and multiple datasets, we use the Nemenyi test for statistical significance, which is based on average ranks across all. Two algorithms are considered to be performing differently in a statistically significant way if the ranks differ by a critical difference given by:

$$\text{Critical Difference} = q_\alpha \sqrt{k(k+1)/6N}$$

Here, K=number of algorithms, N=number of datasets.

The critical difference values are assumed to follow a Student-T distribution.

Incremental unsupervised learning using clustering

The concept behind clustering in a data stream remains the same as in batch or offline modes; that is, finding interesting clusters or patterns which group together in the data while keeping the limits on finite memory and time required to process as constraints. Doing single-pass modifications to existing algorithms or keeping a small memory buffer to do mini-batch versions of existing algorithms, constitute the basic changes done in all the algorithms to make them suitable for stream or real-time unsupervised learning.

Modeling techniques

The clustering modeling techniques for online learning are divided into partition-based, hierarchical-based, density-based, and grid-based, similar to the case of batch-based clustering.

Partition based

The concept of partition-based algorithms is similar to batch-based clustering where k clusters are formed to optimize certain objective functions such as minimizing the inter-cluster distance, maximizing the intra-cluster distance, and so on.

Online k-Means

k-Means is the most popular clustering algorithm, which partitions the data into user-specified k clusters, mostly to minimize the squared error or distance between centroids and cluster assigned points. We will illustrate a very basic online adaptation of k-Means, of which several variants exist.

Inputs and outputs

Mainly, numeric features are considered as inputs; a few tools take categorical features and convert them into some form of numeric representation. The algorithm itself takes the parameters' number of clusters k and number of max iterations n as inputs.

How does it work?

1. The input data stream is considered to be infinite but of constant block size.
2. A memory buffer of the block size is kept reserved to store the data or a compressed representation of the data.
3. Initially, the first stream of data of block size is used to find the k centroids of the clusters, the centroid information is stored and the buffer is cleared.
4. For the next data after it reaches the block size:
 1. For either max number of iterations or until there is no change in the centroids:
 2. Execute k-Means with buffer data and the present centroids.
 3. Minimize the squared sum error between centroids and data assigned to the cluster.
 4. After the iterations, the buffer is cleared and new centroids are obtained.
5. Repeat step 4 until the data is no longer available.

Advantages and limitations

The advantages and limitations are as follows:

- Similar to batch-based, the shape of the detected cluster depends on the distance measure and is not appropriate in problem domains with irregular shapes.
- The choice of parameter **k**, as in batch-based, can limit the performance in datasets with many distinct patterns or clusters.
- Outliers and missing data can pose lots of irregularities in clustering behavior of online k-Means.
- If the selected buffer size or the block size of the stream on which iterative k-Means runs is small, it will not find the right clusters. If the chosen block size is large, it can result in slowdown or missed changes in the data. Extensions such as **Very Fast k-Means Algorithm (VFKM)**, which uses the Hoeffding bound to determine the buffer size, overcome this limitation to a large extent.

Hierarchical based and micro clustering

Hierarchical methods are normally based on **Clustering Features (CF)** and **Clustering Trees (CT)**. We will describe the basics and elements of hierarchical clustering and the BIRCH algorithm, the extension of which the CluStream algorithm is based on.

The Clustering Feature is a way to compute and preserve a summarization statistic about the cluster in a compressed way rather than holding on to the whole data belonging to the cluster. In a **d** dimensional dataset, with **N** points in the cluster, two aggregates in the form of total sum **LS** for each dimensions and total squared sum of data **SS** again for each dimension, are computed and the vector representing this triplet form the Clustering Feature:

$$CF_j = \langle N, LS_j, SS_j \rangle$$

These statistics are useful in summarizing the entire cluster information. The centroid of the cluster can be easily computed using:

$$centroid_j = LS_j/N$$

The radius of the cluster can be estimated using:

$$radius_j = \sqrt{\left(\frac{SS_j}{N} - \left(\frac{LS_j}{N} \right)^2 \right)}$$

The diameter of the cluster can be estimated using:

$$diameter_j = \sqrt{\frac{2N * SS_j - 2LS_j^2}{N(N-1)}}$$

CF vectors have great incremental and additive properties which becomes useful in stream or incremental updates.

For an incremental update, when we must update the CF vector, the following holds true:

$$LS_j \leftarrow LS_j + x^j$$

$$SS_j \leftarrow SS_j + (x^j)^2$$

$$N \leftarrow N + 1$$

When two CFs have to be merged, the following holds true:

$$N_3 \leftarrow N_1 + N_2$$

$$LS_3^j \leftarrow LS_1^j + LS_2^j$$

$$SS_3^j \leftarrow SS_1^j + SS_2^j$$

The **Clustering Feature Tree (CF Tree)** represents an hierarchical tree structure. The construction of the CF tree requires two user defined parameters:

- Branching factor **b** which is the maximum number of sub-clusters or non-leaf nodes any node can have
- Maximum diameter (or radius) **T**, the number of examples that can be absorbed by the leaf node for a CF parent node

CF Tree operations such as insertion are done by recursively traversing the CF Tree and using the CF vector for finding the closest node based on distance metrics. If a leaf node has already absorbed the maximum elements given by parameter *T*, the node is split. At the end of the operation, the CF vector is appropriately updated for its statistic:

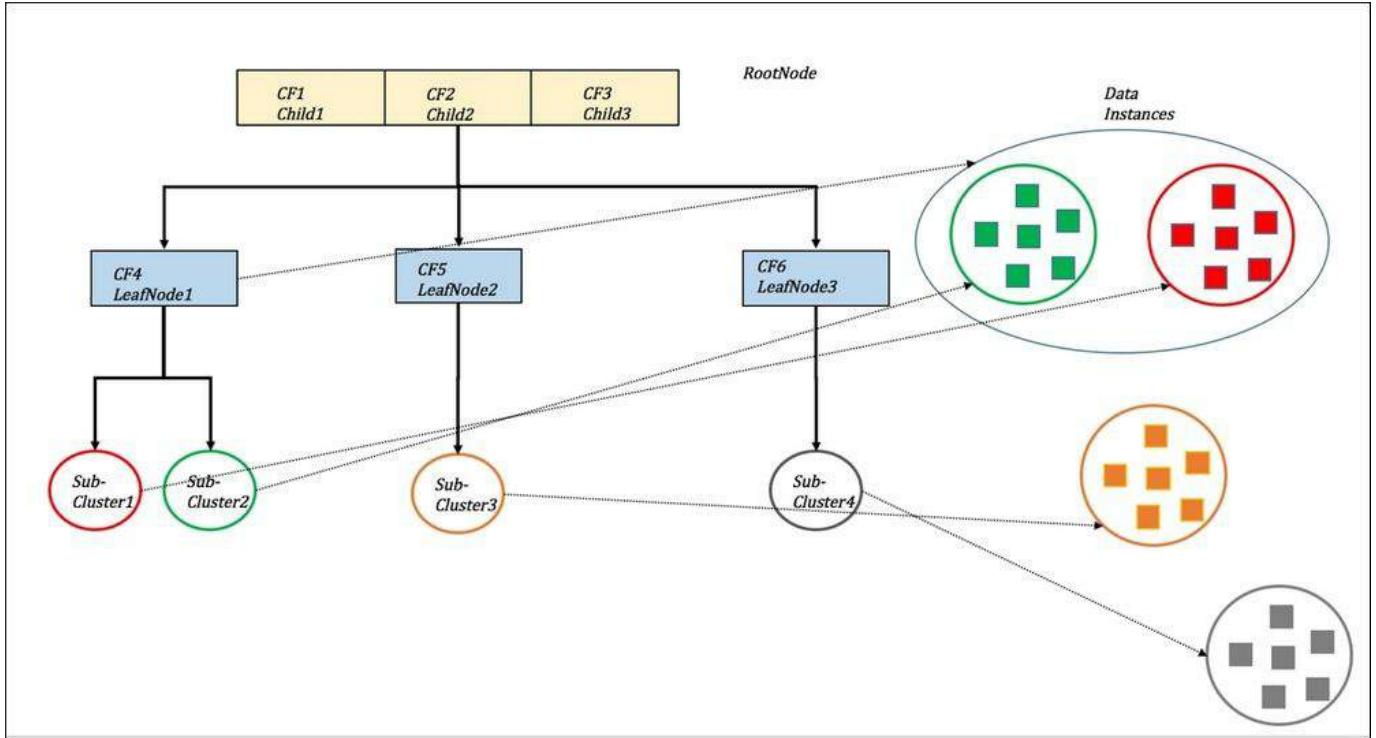


Figure 3 An example Clustering Feature Tree illustrating hierarchical structure.

We will discuss **BIRCH (Balanced Iterative Reducing and Clustering Hierarchies)** following this concept.

Inputs and outputs

BIRCH only accepts numeric features. CF and CF tree parameters, such as branching factor b and maximum diameter (or radius) T for leaf are user-defined inputs.

How does it work?

BIRCH, designed for very large databases, was meant to be a *two-pass* algorithm; that is, scan the entire data once and re-scan it again, thus being an $O(N)$ algorithm. It can be modified easily enough for online as a single pass algorithm preserving the same properties:

1. In the first phase or scan, it goes over the data and creates an in-memory CF Tree structure by sequentially visiting the points and carrying out CF Tree operations as discussed previously.
2. In the second phase, an optional phase, we remove outliers and merge sub-clusters.

3. Phase three is to overcome the issue of order of data in phase one. We use agglomerative hierarchical clustering to refactor the CF Tree.
4. Phase four is the last phase which is an optional phase to compute statistics such as centroids, assign data to closest centroids, and so on, for more effectiveness.

Advantages and limitations

The advantages and limitations are as follows:

- It is one of the most popular algorithms that scales linearly on a large database or stream of data.
- It has compact memory representation in the form of the CF and CF Tree for statistics and operations on incoming data.
- It handles outliers better than most algorithms.
- One of the major limitations is that it has been shown not to perform well when the shape of the clusters is not spherical.
- The concepts of CF vector and clustering in BIRCH were extended for efficient stream mining requirements by Aggarwal *et al* and named *micro-cluster* and *CluStream*.

Inputs and outputs

CluStream only accepts numeric features. Among the user-defined parameters are the number of micro-clusters in memory (q) and the threshold (δ) in time after which they can be deleted. Additionally, included in the input are time-sensitive parameters for storing the micro-clusters information, given by α and l .

How does it work?

1. The micro-cluster extends the CF vector and keeps two additional measures. They are the sum of the timestamps and sum of the squares of timestamps:

$$\text{microCluster}_j = \langle N, LS_j, SS_j, ST, SST \rangle$$
2. The algorithm stores q micro-clusters in memory and each micro-cluster has a *maximum boundary* that can be computed based on means and standard deviations between centroid and cluster instance distances. The measures are multiplied by a factor which decreases exponentially with time.
3. For each new instance, we select the closest micro-cluster based on Euclidean distance and decide whether it should be absorbed:
 1. If the distance between the new instance and the centroid of the closest micro-cluster falls within the maximum boundary, it is absorbed and the

- micro-cluster statistics are updated.
2. If none of the micro-clusters can absorb, a new micro-cluster is created with the instance and based on the timestamp and threshold (δ), the oldest micro-cluster is deleted.
 3. Assuming normal distribution of timestamps, if the relevance time—the time of arrival of instance found by CluStream—is below the user-specified threshold, it is considered an outlier and removed. Otherwise, the two closest micro-clusters are merged.
 4. The micro-cluster information is stored in secondary storage from time to time by using a pyramidal time window concept. Each micro-cluster has time intervals decrease exponentially using αl to create snapshots. These help in efficient search in both time and space.

Advantages and limitations

The advantages and limitations are as follows:

- CluStream has been shown to be very effective in finding clusters in real time
- The CluStream algorithm, through effective storage using a pyramidal timestamp, has efficient time and space usage. CluStream, like BIRCH, can find only spherical shaped clusters

Density based

Similar to batch clustering, density-based techniques overcome the "shape" issue faced by distance-based algorithms. Here we will present a well-known density-based algorithm, DenStream, which is based on the concepts of CF and CF Trees discussed previously.

Inputs and outputs

The extent of the neighborhood of a core micro-cluster is the user-defined radius ϵ . A second input value is the minimum total weight μ of the micro-cluster which is the sum over the weighted function of the arrival time of each instance in the object, where the weight decays with a time constant proportional to another user-defined parameter, λ . Finally, an input factor $\beta \in (0,1)$ is used to distinguish potential core micro-clusters from outlier micro-clusters.

How does it work?

1. Based on the micro-cluster concepts of CluStream, DenStream holds two data

structures: *p-micro-cluster* for potential clusters and *o-micro-clusters* for outlier detection.

2. Each *p-micro-cluster* structure has:

1. A weight associated with it which decreases exponentially with the timestamps it has been updated with. If there are j objects in the micro-

$$w = \sum_{j \in p\text{-microCluster}} f(T - t^j) \quad \text{where } f(t) = 2^{-\lambda t}$$

2. **Weighted linear sum (WLS)** and **weighted linear sum of squares (WSS)** are stored in micro-clusters similar to linear sum and sum of squares:

$$WLS = \sum_{j \in p\text{-microCluster}} f(T - t^j) x^j$$

$$WSS = w = \sum_{j \in p\text{-microCluster}} f(T - t^j) (x^j)^2$$

3. The mean, radius, and diameter of the clusters are then computed using the weighted measures defined previously, exactly like in CF. For example, the radius can be given as:

$$\text{weightedRadius}_j = \sqrt{\left(\frac{WSS_j}{w} - \left(\frac{WLS_j}{w} \right)^2 \right)}$$

3. Each *o-micro-cluster* has the same structure as *p-micro-cluster* and timestamps associated with it.
4. When a new instance arrives:
 1. A closest *p-micro-cluster* is found and the instance is inserted if the new radius is within the user-defined boundary ϵ . If inserted, the *p-micro-*

cluster statistics are updated accordingly.

2. Otherwise, an *o-micro-cluster* is found and the instance is inserted if the new radius is again within the boundary. The boundary is defined by $\beta \times \mu$, the product of the user-defined parameters, and if the radius grows beyond this value, the *o-micro-cluster* is moved to the *p-micro-cluster*.
3. If the instance cannot be absorbed by an *o-micro-cluster*, then a new micro-cluster is added to the *o-micro-clusters*.
5. At the time interval t based on weights, the *o-micro-cluster* can become the *p-micro-cluster* or vice versa. The time interval is defined in terms of λ , β , and μ as:

$$T = \frac{1}{\lambda} \log \frac{\beta\mu}{\beta\mu - 1}$$

Advantages and limitations

The advantages and limitations are as follows:

- Based on the parameters, DenStream can find effective clusters and outliers for real-time data.
- It has the advantage of finding clusters and outliers of any shape or size.
- The house keeping job of updating the *o-micro-cluster* and *p-micro-cluster* can be computationally expensive if not selected properly.

Grid based

This technique is based on discretizing the multi-dimensional continuous space into a multi-dimensional discretized version with grids. The mapping of the incoming instance to grid online and maintaining the grid offline results in an efficient and effective way of finding clusters in real-time.

Here we present D-Stream, which is a grid-based online stream clustering algorithm.

Inputs and outputs

As in density-based algorithms, the idea of decaying weight of instances is used in D-Stream. Additionally, as described next, cells in the grid formed from the input

space may be deemed sparse, dense, or sporadic, distinctions that are central to the computational and space efficiency of the algorithm. The inputs to the grid-based algorithm, then, are:

- λ : The decay factor
- $0 < C_l < 1$ and $C_m > 1$: Parameters that control the boundary between dense and sparse cells in the grid
- $\beta > 0$: A constant that controls one of the conditions when a sparse cell is to be considered sporadic.

How does it work?

1. Each instance arriving at time t has a density coefficient that decreases exponentially over time:

$$D(x^j, t) = \lambda^{t-t^j}$$

2. Density of the grid cell g at any given time t is given by $D(g, t)$ and is the sum of the adjusted density of all instances given by $E(g, t)$ that are mapped to grid cell g :

$$D(g, t) = \sum_{x \in E(g, t)} D(x, t)$$

3. Each cell in the grid captures the statistics as a characterization vector given by:
 - $CV(g) = \langle t_g, t_m, D, label, status \rangle$ where:
 - t_g = last time grid cell was updated
 - t_m = last time grid cell was removed due to sparseness
 - D = density of the grid cell when last updated
 - $label$ = class label of the grid cell
 - $status$ = {NORMAL or SPORADIC}
4. When the new instance arrives, it gets mapped to a cell g and the characteristic vector is updated. If g is not available, it is created and the list of grids is updated.
5. Grid cells with empty instances are removed. Also, cells that have not been updated over a long time can become sparse, and conversely, when many

- instances are mapped, they become dense.
6. At a regular time interval known as a gap, the grid cells are inspected for status and the cells with fewer instances than a number—determined by a density threshold function—are treated as outliers and removed.

Advantages and limitations

The advantages and limitations are as follows:

- D-Streams have theoretically and empirically been shown to find sporadic and normal clusters with very high efficiency in space and time.
- It can find clusters of any shape or size effectively.

Validation and evaluation techniques

Many of the static clustering evaluation measures discussed in [Chapter 3](#), *Unsupervised Machine Learning Techniques*, have an assumption of static and non-evolving patterns. Some of these internal and external measures are used even in streaming based cluster detection. Our goal in this section is to first highlight problems inherent to cluster evaluation in stream learning, then describe different internal and external measures that address these, and finally, present some existing measures—both internal and external—that are still valid.

Key issues in stream cluster evaluation

It is important to understand some of the important issues that are specific to streaming and clustering, as the measures need to address them:

- **Aging:** The property of points being not relevant to the clustering measure after a given time.
- **Missed points:** The property of a point not only being missed as belonging to the cluster but the amount by which it was missed being in the cluster.
- **Misplaced points:** Changes in clusters caused by evolving new clusters. Merging existing or deleting clusters results in ever-misplaced points with time. The impact of these changes with respect to time must be taken into account.
- **Cluster noise:** Choosing data that should not belong to the cluster or forming clusters around noise and its impact over time must be taken into account.

Evaluation measures

Evaluation measures for clustering in the context of streaming data must provide a useful index of the quality of clustering, taking into consideration the effect of

evolving and noisy data streams, overlapping and merging clusters, and so on. Here we present some external measures used in stream clustering. Many internal measures encountered in [Chapter 3](#), *Unsupervised Machine Learning Techniques*, such as the Silhouette coefficient, Dunn's Index, and R-Squared, are also used and are not repeated here.

Cluster Mapping Measures (CMM)

The idea behind CMM is to quantify the connectivity of the points to clusters given the ground truth. It works in three phases:

Mapping phase: In this phase, clusters assigned by the stream learning algorithm are mapped to the ground truth clusters. Based on these, various statistics of distance and point connectivity are measured using the concepts of k-Nearest Neighbors.

The average distance of point p to its closest k neighbors in a cluster C_i is given by:

$$KnnhDistance(p, C_i) = \frac{1}{k} \sum_{o \in knnh(p, C_i)} dist(o, p)$$

The average distance for a cluster C_i is given by:

$$knnhDistance(C_i) = \frac{1}{|C_i|} \sum_{p \in C_i} KnnhDistance(p, C_i)$$

The point connectivity of a point p in a cluster C_i is given by:

$$conn(p, C_i) = \begin{cases} 1 & \text{if } KnnhDistance(p, C_i) < knnhDistance(C_i) \\ 0 & \text{if } C_i = 0 \\ \frac{knnhDistance(C_i)}{KnnhDistance(p, C_i)} & \text{otherwise} \end{cases}$$

Class frequencies are counted for each cluster and the mapping of the cluster to the ground truth is performed by calculating the histograms and similarity in the clustering.

Specifically, a cluster C_i is mapped to the ground truth class, and Cl_j is mapped to a ground truth cluster Cl_j^o , which covers the majority of class frequencies of C_i . The surplus is defined as the number of instances from class Cl_i not covered by the ground truth cluster Cl_j^o and total surplus for instances in classes $Cl_1, Cl_2, Cl_3 \dots$ Cl_1 in cluster C_i compared to Cl_j^o is given by:

$$\Delta(C_i, Cl_j^o) = \sum_{a=1}^l \max \left\{ 0, \rho(C_i)_a - \rho(Cl_j^o)_a \right\}$$

Cluster C_i is mapped using:

$$map(C_i) = \begin{cases} \arg \min \left\{ \Delta(C_i, Cl_j^o) \right\} \text{ if } \forall Cl_j \Delta(C_i, Cl_j^o) > 0 \\ 0 \text{ if } C_i = C_0 \\ \arg \max \left\{ |C_i \cap Cl_j^o| \right\} \end{cases}$$

Penalty phase: The penalty for every instance that is mapped incorrectly is calculated in this step using computations of fault objects; that is, objects which are not noise and yet incorrectly placed, using:

$$pen(o, C_i) = con(o, Cl(o)) * (1 - con(o, map(C_i)))$$

The overall penalty of point o with respect to all clusters found is given by:

$$pen(o, C) = \max \{ pen(o, C_i) \}$$

CMM calculation: Using all the penalties weighted over the lifespan is given by:

$$CMM(C, Cl) = 1 - \frac{\sum_{o \in F} w(o) * pen(o, C)}{\sum_{o \in F} w(o) * conn(o, Cl(o))}$$

Here, C is found clusters, Cl is ground truth clusters, F is the fault objects and $w(o)$ is the weight of instance.

V-Measure

Validity or V-Measure is an external measure which is computed based on two properties that are of interest in stream clustering, namely, **Homogeneity** and **Completeness**. If there are n classes as set $C = \{c_1, c_2, \dots, c_n\}$ and k clusters $K = \{k_1, k_2, \dots, k_m\}$, contingency tables are created such that $A = \{a_{ij}\}$ corresponds to the count of instances in class c_i and cluster k_j .

Homogeneity: Homogeneity is defined as a property of a cluster that reflects the extent to which all the data in the cluster belongs to the same class.

Conditional entropy and class entropy:

$$H(C|K) = - \sum_{k=1}^{|K|} \sum_{c=1}^{|C|} \frac{a_{ck}}{N} \log \frac{a_{ck}}{\sum_{c=1}^{|C|} a_{ck}}$$

$$H(C) = - \sum_{c=1}^{|C|} \frac{\sum_{k=1}^{|K|} a_{ck}}{n} \log \frac{\sum_{k=1}^{|K|} a_{ck}}{n}$$

Homogeneity is defined as:

$$h = \begin{cases} 1 & \text{if } H(C|K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{otherwise} \end{cases}$$

A higher value of homogeneity is more desirable.

Completeness: Completeness is defined as the mirror property of Homogeneity, that is, having all instances of a single class belong to the same cluster.

Similar to Homogeneity, conditional entropies and cluster entropy are defined as:

$$H(K|C) = - \sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{a_{ck}}{N} \log \frac{a_{ck}}{\sum_{k=1}^{|K|} a_{ck}}$$

$$H(K) = - \sum_{k=1}^{|K|} \frac{\sum_{c=1}^{|C|} a_{ck}}{n} \log \frac{\sum_{c=1}^{|C|} a_{ck}}{n}$$

Completeness is defined as:

$$c = \begin{cases} 1 & \text{if } H(K|C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{otherwise} \end{cases}$$

V-Measure is defined as the harmonic mean of homogeneity and completeness using a weight factor β :

$$v_\beta = \frac{(1+\beta)h*c}{\beta*h + c}$$

A higher value of completeness or V-measure is better.

Other external measures

Some external measures which are quite popular in comparing the clustering algorithms or measuring the effectiveness of clustering when the classes are known are given next:

Purity and Entropy: They are similar to homogeneity and completeness defined previously.

Purity is defined as:

$$purity = \sum_{r=1}^k \frac{1}{n} \max_i n_r^i$$

Entropy is defined as:

$$entropy = \sum_{r=1}^k \frac{n_r}{n} \left(-\frac{1}{\log q} \sum_{i=1}^q \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r} \right)$$

Here, q = number of classes, k = number of clusters, n_r = size of cluster r and n_r^i = number of data in cluster r belonging to class i .

Precision, Recall, and F-Measure: Information retrieval measures modified for clustering algorithms are as follows:

Given, N = data points, C = set of classes, K = set of Clusters and

n_{ij} = number of data points in class i belonging to cluster j .

Precision is defined as:

$$P(c_i, k_j) = \frac{n_{ij}}{|k_j|}$$

Recall is defined as:

$$R(c_i, k_j) = \frac{n_{ij}}{|c_i|}$$

F-measures is defined as:

$$F - Measures(c_i, k_j) = \frac{2 * P(c_i, k_j) * R(c_i, k_j)}{P(c_i, k_j) + R(c_i, k_j)}$$

Unsupervised learning using outlier detection

The subject of finding outliers or anomalies in the data streams is one of the emerging fields in machine learning. This area has not been explored by researchers as much as classification and clustering-based problems have. However, there have been some very interesting ideas extending the concepts of clustering to find outliers from data streams. We will provide some of the research that has been proved to be very effective in stream outlier detection.

Partition-based clustering for outlier detection

The central idea here is to use an online partition-based clustering algorithm and based on either cluster size ranking or inter-cluster distance ranking, label the clusters as outliers.

Here we present one such algorithm proposed by Koupaei *et al.*, using incremental k-Means.

Inputs and outputs

Only numeric features are used, as in most k-Means algorithms. The number of clusters k and the number of windows of outliers n , on which offline clustering happens, are input parameters. The output is constant outliers (local and global) and an updatable model that detects these.

How does it work?

1. This algorithm works by having the k-Means algorithm in two modes, an offline mode and an online mode, both working in parallel.
2. For the online mode:
 1. Apply k-Means on the given window w and find clusters and partitions of the data with clusters.
 2. Rank the clusters based on the cluster distances and cluster size. The clusters which are farthest apart and small in size are considered outliers.
 3. Store the outliers in memory for the window as a set $O_w = \{x_1, x_2, \dots, x_n\}$ and regard them as local outliers.
 4. The window is cleared and the process is repeated.
3. For the offline mode:
 1. Get outliers from n , previous windows, and create a set:
$$S = \{O_w^1, O_w^2, \dots, O_w^n\}$$
 2. Cluster this window with set S using k-Means and find clusters which are farthest away and small in size.
 3. These clusters are global outliers.
 4. The window is cleared and the process is repeated.

Advantages and limitations

The advantages and limitations are as follows:

- It is very sensitive to the two parameters k and n and can generate lots of noise.
- Only spherical clusters/outliers are found and outliers with different shapes get missed.

Distance-based clustering for outlier detection

Distance-based outlier detection is the most studied, researched, and implemented method in the area of stream learning. There are many variants of the distance-based methods, based on sliding windows, the number of nearest neighbors, radius and thresholds, and other measures for considering outliers in the data. We will try to give a sampling of the most important algorithms in this section.

Inputs and outputs

Most algorithms take the following parameters as inputs:

- Window size w , corresponding to the fixed size on which the algorithm looks for outlier patterns
- Sliding size s , corresponds to the number of new instances that will be added to the window, and old ones removed
- The count threshold k of instances when using nearest neighbor computation
- The distance threshold R used to define the outlier threshold in distances

Outliers as labels or scores (based on neighbors and distance) are outputs.

How does it work?

We present different variants of distance-based stream outlier algorithms, giving insights into what they do differently or uniquely. The unique elements in each algorithm define what happens when the slide expires, how a new slide is processed, and how outliers are reported.

Exact Storm

Exact Storm stores the data in the current window w in a well-known index structure, so that the range query search or query to find neighbors within the distance R for a given point is done efficiently. It also stores k preceding and succeeding neighbors of all data points:

- **Expired Slide:** Instances in expired slides are removed from the index structure that affects range queries but are preserved in the preceding list of neighbors.
- **New Slide:** For each data point in the new slide, range query R is executed, results are used to update the preceding and succeeding list for the instance, and the instance is stored in the index structure.
- **Outlier Reporting:** In any window, after the processing of expired and new

slide elements is complete, any instance with at least k elements from the succeeding list and non-expired preceding list is reported as an outlier.

Abstract-C

Abstract-C keeps the index structure similar to Exact Storm but instead of preceding and succeeding lists for every object it just maintains a list of counts of neighbors for the windows the instance is participating in:

- **Expired Slide:** Instances in expired slides are removed from the index structure that affects range queries and the first element from the list of counts is removed corresponding to the last window.
- **New Slide:** For each data point in the new slide, range query R is executed and results are used to update the list count. For existing instances, the count gets updated with new neighbors and instances are added to the index structure.
- **Outlier Reporting:** In any window, after the processing of expired and new slide elements is complete, all instances with a neighbors count less than k in the current window are considered outliers.

Direct Update of Events (DUE)

DUE keeps the index structure for efficient range queries exactly like the other algorithms but has a different assumption, that when an expired slide occurs, not every instance is affected in the same way. It maintains two priority queues: the unsafe inlier queue and the outlier list. The unsafe inlier queue has sorted instances based on the increasing order of smallest expiration time of their preceding neighbors. The outlier list has all the outliers in the current window:

- **Expired Slide:** Instances in expired slides are removed from the index structure that affects range queries and the unsafe inlier queue is updated for expired neighbors. Those unsafe inliers which become outliers are removed from the priority queue and moved to the outlier list.
- **New Slide:** For each data point in the new slide, range query R is executed, results are used to update the succeeding neighbors of the point, and only the most recent preceding points are updated for the instance. Based on the updates, the point is added to the unsafe inlier priority queue or removed from the queue and added to the outlier list.
- **Outlier Reporting:** In any window, after the processing of expired and new slide elements is complete, all instances in the outlier list are reported as outliers.

Micro Clustering based Algorithm (MCOD)

Micro-clustering based outlier detection overcomes the computational issues of performing range queries for every data point. The micro-cluster data structure is used instead of range queries in these algorithms. A micro-cluster is centered around an instance and has a radius of R . All the points belonging to the micro-clusters become inliers. The points that are outside can be outliers or inliers and stored in a separate list. It also has a data structure similar to DUE to keep a priority queue of unsafe inliers:

- **Expired Slide:** Instances in expired slides are removed from both micro-clusters and the data structure with outliers and inliers. The unsafe inlier queue is updated for expired neighbors as in the DUE algorithm. Micro-clusters are also updated for non-expired data points.
- **New Slide:** For each data point in the new slide, the instance either becomes a center of a micro-cluster, or part of a micro-cluster or added to the event queue and the data structure of the outliers. If the point is within the distance R , it gets assigned to an existing micro-cluster; otherwise, if there are k points within R , it becomes the center of the new micro cluster; if not, it goes into the two structures of the event queue and possible outliers.
- **Outlier Reporting:** In any window, after the processing of expired and new slide elements is complete, any instance in the outlier structure with less than k neighboring instances is reported as an outlier.

Approx Storm

Approx Storm, as the name suggests, is an approximation of Exact Storm. The two approximations are:

- Reducing the number of data points in the window by adding a factor ρ and changing the window to ρW .
- Storing the number instead of the data structure of preceding neighbors by using the fraction of the number of neighbors which are safe inliers in the preceding list to the number in the current window.

The processing of expired and new slides and how outliers are determined based on these steps follows:

1. **Expired Slide:** Same as Exact Storm—instances in expired slides are removed from the index structure that affects range queries but preserved in the preceding list of neighbors.
2. **New Slide:** For each data point in the new slide, range query R is executed, results are used to compute the fraction discussed previously, and the index

structure is updated. The number of safe inliers are constrained to ρW by removing random inliers if the size exceeds that value. The assumption is that most of the points in safe inliers are safe.

3. **Outlier Reporting:** In any window, after the processing of expired and new slide elements has been completed, when an approximation (see *References* [17]) of the number of neighbors of an instance based on the fraction, window size, and preceding list is a value less than k , it is considered as an outlier.

Advantages and limitations

The advantages and limitations are as follows:

- Exact Storm is demanding in storage and CPU for storing lists and retrieving neighbors. Also, it introduces delays; even though they are implemented in efficient data structures, range queries can be slow.
- Abstract-C has a small advantage over Exact Storm, as no time is spent on finding active neighbors for each instance in the window. The storage and time spent is still very much dependent on the window and slide chosen.
- DUE has some advantage over Exact Storm and Abstract-C as it can efficiently re-evaluate the "inlierness" of points (that is, whether unsafe inliers remain inliers or become outliers) but sorting the structure impacts both CPU and memory.
- MCOD has distinct advantages in memory and CPU owing to the use of the micro-cluster structure and removing the pair-wise distance computation. Storing the neighborhood information in micro-clusters helps memory too.
- Approx Storm has an advantage of time over the others as it doesn't process the expired data points over the previous window.

Validation and evaluation techniques

Validation and evaluation of stream-based outliers is still an open research area. In many research comparisons, we see various metrics being used, such as:

- Time to evaluate in terms of CPU times per object
- Number of outliers detected in the streams
- Number of outliers that correlate to existing labels, TP/Precision/Recall/Area under PRC curve, and so on

By varying parameters such as window-size, neighbors within radius, and so on, we determine the sensitivity to the performance metrics mentioned previously and determine the robustness.

Case study in stream learning

The case study in this chapter consists of several experiments that illustrate different methods of stream-based machine learning. A well-studied dataset was chosen as the stream data source and supervised tree based methods such as Naïve Bayes, Hoeffding Tree, as well as ensemble methods, were used. Among unsupervised methods, clustering algorithms used include k-Means, DBSCAN, CluStream, and CluTree. Outlier detection techniques include MCOD and SimpleCOD, among others. We also show results from classification experiments that demonstrate handling concept drift. The ADWIN algorithm for calculating statistics in a sliding window, as described earlier in this chapter, is employed in several algorithms used in the classification experiments.

Tools and software

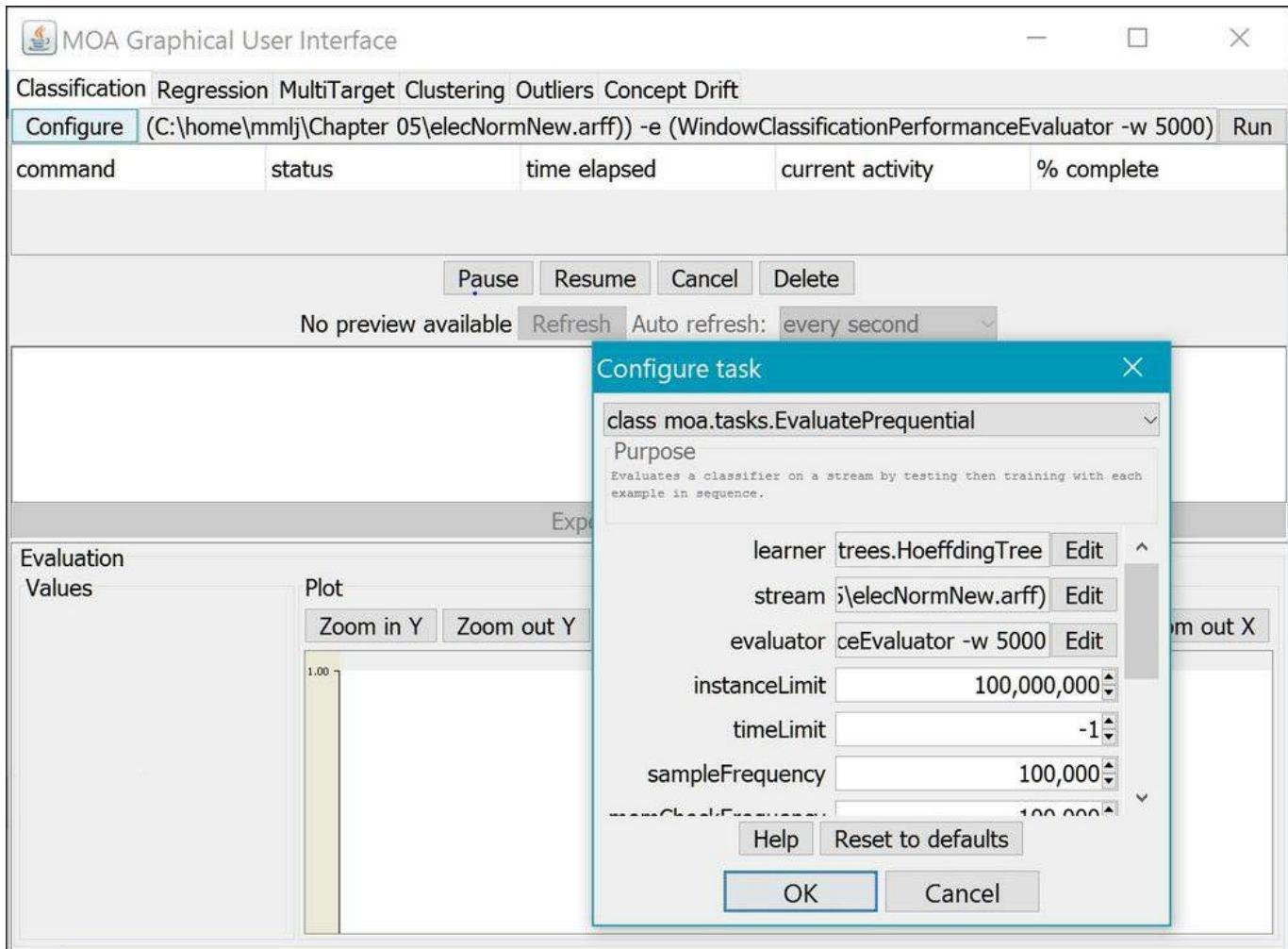
One of the most popular and arguably the most comprehensive Java-based frameworks for data stream mining is the open source **Massive Online Analysis (MOA)** software created by the University of Waikato. The framework is a collection of stream classification, clustering, and outlier detection algorithms and has support for change detection and concept drift. It also includes data generators and several evaluation tools. The framework can be extended with new stream data generators, algorithms, and evaluators. In this case study, we employ several stream data learning methods using a file-based data stream.

Note

Product homepage: <http://moa.cms.waikato.ac.nz/>

GitHub: <https://github.com/Waikato/moa>

As shown in the series of screenshots from the MOA tool shown in *Figure 4* and *Figure 5*, the top-level menu lets you choose the type of learning to be done. For the classification experiments, for example, configuration of the tools consists of selecting the task to run (selected to be prequential evaluation here), and then configuring which learner and evaluator we want to use, and finally, the source of the data stream. A window width parameter shown in the **Configure Task** dialog can affect the accuracy of the model chosen, as we will see in the experiment results. Other than choosing different values for the window width, all base learner parameters were left as default values. Once the task is configured it is run by clicking the **Run** button:



Configure task



class moa.tasks.EvaluatePrequential



Purpose

Evaluates a classifier on a stream by testing then training with each example in sequence.

sampleFrequency

100,000



memCheckFrequency

100,000



dumpFile

Browse

outputPredictionFile

Browse

width

1,000



alpha

0.01



taskResultFile

Browse



Help

Reset to defaults

OK

Cancel

Figure 4. MOA graphical interface for configuring prequential evaluation for classification which includes setting the window width

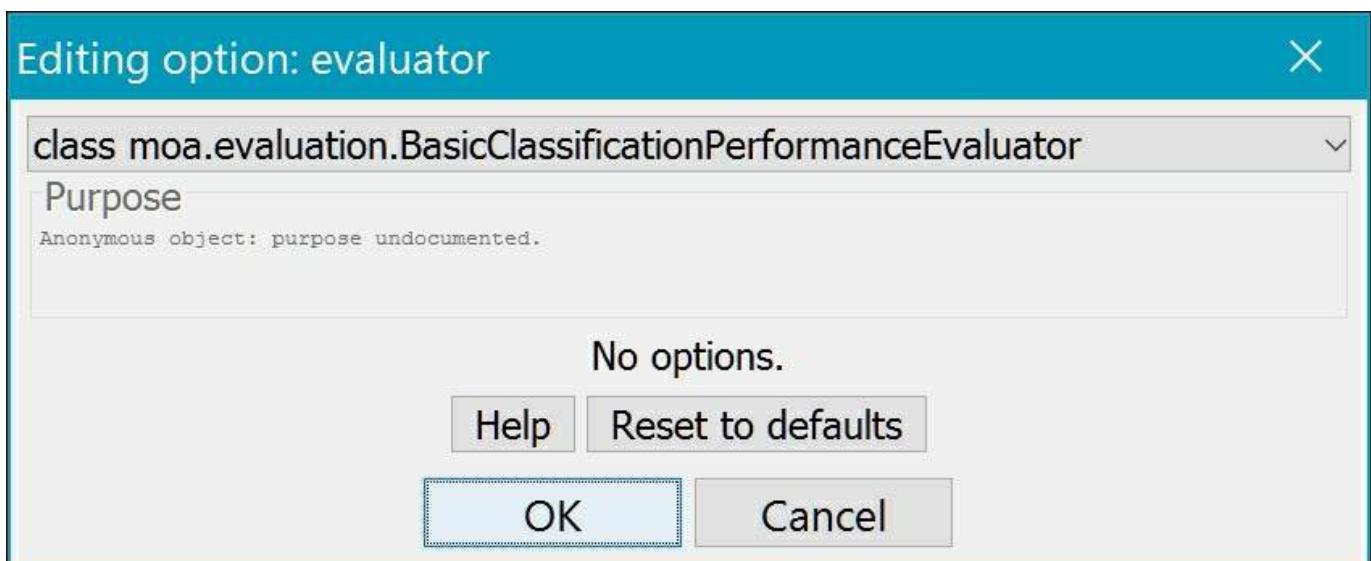
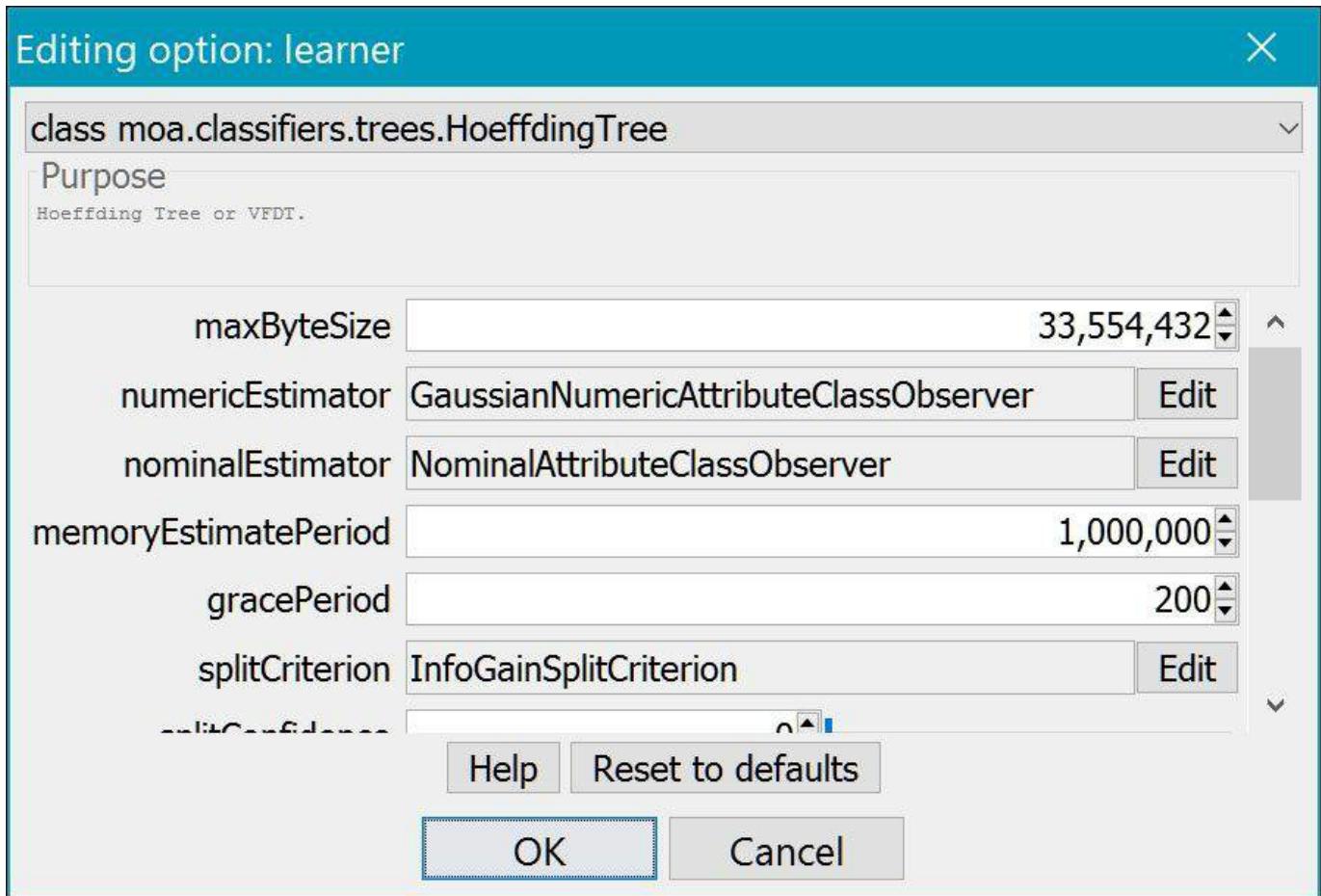


Figure 5. MOA graphical interface for prequential classification task. Within the Configure task, you must choose a learner, locate the data stream (details not

shown), and select an evaluator

After the task has completed running, model evaluation results can be exported to a CSV file.

Business problem

The problem for this case study is to continuously learn from a stream of electricity market data and predict the direction of movement of the market price. We compare the accuracy and average cost of different classification methods including concept drift as well as the performance of clustering and outlier detection.

Machine learning mapping

The dataset used in this case study can be used to illustrate classical batch-based supervised and unsupervised learning techniques. However, here we treat it as a stream-based data source to show how we can employ the techniques described in this chapter to perform classification, clustering, and outlier detection tasks using the MOA framework. Within this context, we demonstrate how incremental learning can be achieved under assumptions of a stationary as well as an evolving data stream exhibiting concept drift.

Data collection

The dataset is known as the Electricity or ELEC dataset, which was collected by the New South Wales Electricity Market. The prices in this market are variable, and are adjusted every 5 minutes based on supply and demand. This dataset consists of 45,312 such data points obtained every half-hour between May 1996 and December 1998. The target is an indication of the movement of the price, whether up or down, relative to the 24-hour moving average.

Note

The data file is a publicly available file in the ARFF format at

[http://downloads.sourceforge.net/project/mao-datasetstream/Datasets/Classification/elecNormNew.arff.zip?
r=http%3A%2F%2Fmoa.cms.waikato.ac.nz%2Fdatasets%2F&ts=1483128450&use_](http://downloads.sourceforge.net/project/mao-datasetstream/Datasets/Classification/elecNormNew.arff.zip?r=http%3A%2F%2Fmoa.cms.waikato.ac.nz%2Fdatasets%2F&ts=1483128450&use_)

Data sampling and transformation

In the experiments conducted here, no data sampling is done; each example in the dataset is processed individually and no example is excluded. All numeric data elements have been normalized to a value between 0 and 1.

Feature analysis and dimensionality reduction

The ELEC dataset has 45,312 records with nine features, including the target class. The features class and day are nominal (categorical), all others are numeric (continuous). The features are listed in *Table 1* and *Table 2* and give descriptive statistics for the ELEC dataset:

Name	Data Type	Description
class	nominal	UP, DOWN—direction of price movement relative to 24-hour moving average
date	continuous	Date price recorded
day	nominal	Day of the week (1-7)
period	continuous	
nswprice	continuous	Electricity price in NSW
nswdemand	continuous	Electricity demand in NSW
vicprice	continuous	Electricity price in Victoria
vicdemand	continuous	Electricity demand in Victoria
transfer	integer	

Table 1. ELEC dataset features

	count	mean	std	25%	50%	75%
date	45312	0.49908	0.340308	0.031934	0.456329	0.880547

period	45312	0.5	0.294756	0.25	0.5	0.75
nswprice	45312	0.057868	0.039991	0.035127	0.048652	0.074336
nswdemand	45312	0.425418	0.163323	0.309134	0.443693	0.536001
vicprice	45312	0.003467	0.010213	0.002277	0.003467	0.003467
vicdemand	45312	0.422915	0.120965	0.372346	0.422915	0.469252
transfer	45312	0.500526	0.153373	0.414912	0.414912	0.605702

Table 2. Descriptive statistics of ELEC dataset features

The feature reduction step is omitted here as it is in most stream-based learning.

Models, results, and evaluation

The experiments are divided into classification, concept drift, clustering, and outlier detection. Details of the learning process for each set of experiments and the results of the experiments are given here.

Supervised learning experiments

For this set of experiments, a choice of linear, non-linear, and ensemble learners were chosen in order to illustrate the behavior of a variety of classifiers. **Stochastic Gradient Descent (SGD)**, which uses a linear SVM, and Naïve Bayes are the linear classifiers, while Lazy k-NN is the non-linear classifier. For ensemble learning, we use two meta-learners, **Leveraging Bagging (LB)** and OxaBag, with different linear and non-linear base learners such as SGD, Naïve Bayes, and Hoeffding Trees. The algorithm used in OxaBag is described in the section on ensemble algorithms. In LB, the weight factor used for resampling is variable (the default value of 6 is used here) whereas the weight in OxaBag is fixed at 1.

Prequential evaluation is chosen for all the classification methods, so each example is first tested against the prediction with the existing model, and then used for training the model. This requires the selection of a window width, and the performance of the various models for different values of the window width are listed in *Table 3*. Widths of 100, 500, 1,000, and 5,000 elements were used:

Algorithm	Window width	Evaluation time (CPU seconds)	Model cost (RAM-Hours)	Classifications correct (percent)	Kappa Statistic (percent)
SGD	100	0.5781	3.76E-10	67	0
SGD	500	0.5781	3.76E-10	55.6	0
SGD	1000	0.5469	3.55E-10	53.3	0
SGD	5000	0.5469	3.55E-10	53.78	0
NaiveBayes	100	0.7656	8.78E-10	86	65.7030
NaiveBayes	500	0.6094	8.00E-10	82.2	62.6778
NaiveBayes	1000	0.6719	7.77E-10	75.3	48.8583

NaiveBayes	5000	0.6406	7.35E-10	77.84	54.1966
kNN	100	34.6406	4.66E-06	74	36.3057
kNN	500	34.5469	4.65E-06	79.8	59.1424
kNN	1000	35.8750	4.83E-06	82.5	64.8049
kNN	5000	35.0312	4.71E-06	80.32	60.4594
LB-kNN	100	637.8125	2.88E-04	74	36.3057
LB-kNN	500	638.9687	2.89E-04	79.8	59.1424
LB-kNN	1000	655.8125	2.96E-04	82.4	64.5802
LB-kNN	5000	667.6094	3.02E-04	80.66	61.0965
LB-HoeffdingTree	100	13.6875	2.98E-06	91	79.1667
LB-HoeffdingTree	500	13.5781	2.96E-06	93	85.8925
LB-HoeffdingTree	1000	12.5625	2.74E-06	92.1	84.1665
LB-HoeffdingTree	5000	12.7656	2.78E-06	90.74	81.3184

Table 3. Classifier performance for different window sizes

For the algorithms in *Table 4*, the performance was the same for each value of the window width used:

Algorithm	Evaluation time (CPU seconds)	Model cost (RAM-Hours)	Classifications correct (percent)	Kappa Statistic (percent)
HoeffdingTree	1.1562	3.85E-08	79.1953	57.2266
HoeffdingAdaptiveTree	2.0469	2.84E-09	83.3863	65.5569

OzaBag-NaiveBayes	2.01562	1.57E-08	73.4794	42.7636
OzaBagAdwin-HoeffdingTree	5.7812	2.26E-07	84.3485	67.5221
LB-SGD	2	1.67E-08	57.6977	3.0887
LB-NaiveBayes	3.5937	3.99E-08	78.8753	55.7639

Table 4. Performance of classifiers (same for all window widths used)

Concept drift experiments

In this experiment, we continue using EvaluatePrequential when configuring the classification task. This time we select the DriftDetectionMethodClassifier as the learner and DDM as the drift detection method. This demonstrates adapting to an evolving data stream. Base learners used and the results obtained are shown in *Table 5*:

Algorithm	Evaluation time (CPU seconds)	Model cost (RAM-Hours)	Classifications correct (percent)	Kappa Statistic (percent)	Change detected
SGD	0.307368829	1.61E-09	53.3	0	132
Naïve-Bayes	0.298290727	1.58E-09	86.6	73.03986	143
Lazy-kNN	10.34161893	1.74E-06	87.4	74.8498	12
HoeffdingTree	0.472981754	5.49E-09	86.2	72.19816	169
HoeffdingAdaptiveTree	0.598665043	7.19E-09	84	67.80878	155
LB-SGD	0.912737325	2.33E-08	53.3	0	132
LB-NaiveBayes	1.990137758	3.61E-08	85.7	71.24056	205
OzaBag-NaiveBayes	1.342189725	2.29E-08	77.4	54.017	211
LB-kNN	173.3624715	1.14E-04	87.5	75.03296	4
LB-HoeffdingTree	5.660440101	1.61E-06	91.3	82.56317	59

OzaBag-HoeffdingTree	4.306455545	3.48E-07	85.4	70.60209	125
----------------------	-------------	----------	------	----------	-----

Table 5. Performance of classifiers with concept drift detection

Clustering experiments

Almost all of the clustering algorithms implemented in the MOA tool were used in this experiment. Both extrinsic and intrinsic evaluation results were collected and are tabulated in *Table 6*. CMM, homogeneity, and completeness were defined earlier in this chapter. We have encountered Purity and Silhouette coefficients before, from the discussion in [Chapter 3, Unsupervised Machine Learning Techniques](#). SSQ is the sum of squared distances of instances from their respective cluster centers; the lower the value of SSQ, the better. The use of micro-clustering is indicated by $m = 1$ in the table. How often the macro-clusters are calculated is determined by the selected time horizon h , in instances:

Algorithm	CMM	Homogeneity	Completeness	Purity	SSQ	Silhouette Coefficient
Clustream With k-Means ($h = 5000$; $k = 2$; $m = 1$)	0.7168	-1.0000	0.1737	0.9504	9.1975	0.5687
Clustream With k-Means ($h = 1000$; $k = 5$)	0.5391	-1.0000	0.8377	0.7238	283.6543	0.8264
Clustream ($h = 1000$; $m = 1$)	0.6241	-1.0000	0.4363	0.9932	7.2734	0.4936
Denstream With DBSCAN ($h = 1000$)	0.4455	-1.0000	0.7586	0.9167	428.7604	0.6682
ClusTree ($h = 5000$; $m = 1$)	0.7984	0.4874	-0.4815	0.9489	11.7789	0.6879
ClusTree ($h = 1000$; $m = 1$)	0.7090	-1.0000	0.3979	0.9072	13.4190	0.5385
AbstractC	1.0000	1.0000	-8.1354	1.0000	0.0000	0.0000
MCOD ($w = 1000$)	1.0000	1.0000	-8.1354	1.0000	0.0000	0.0000

Outlier detection experiments

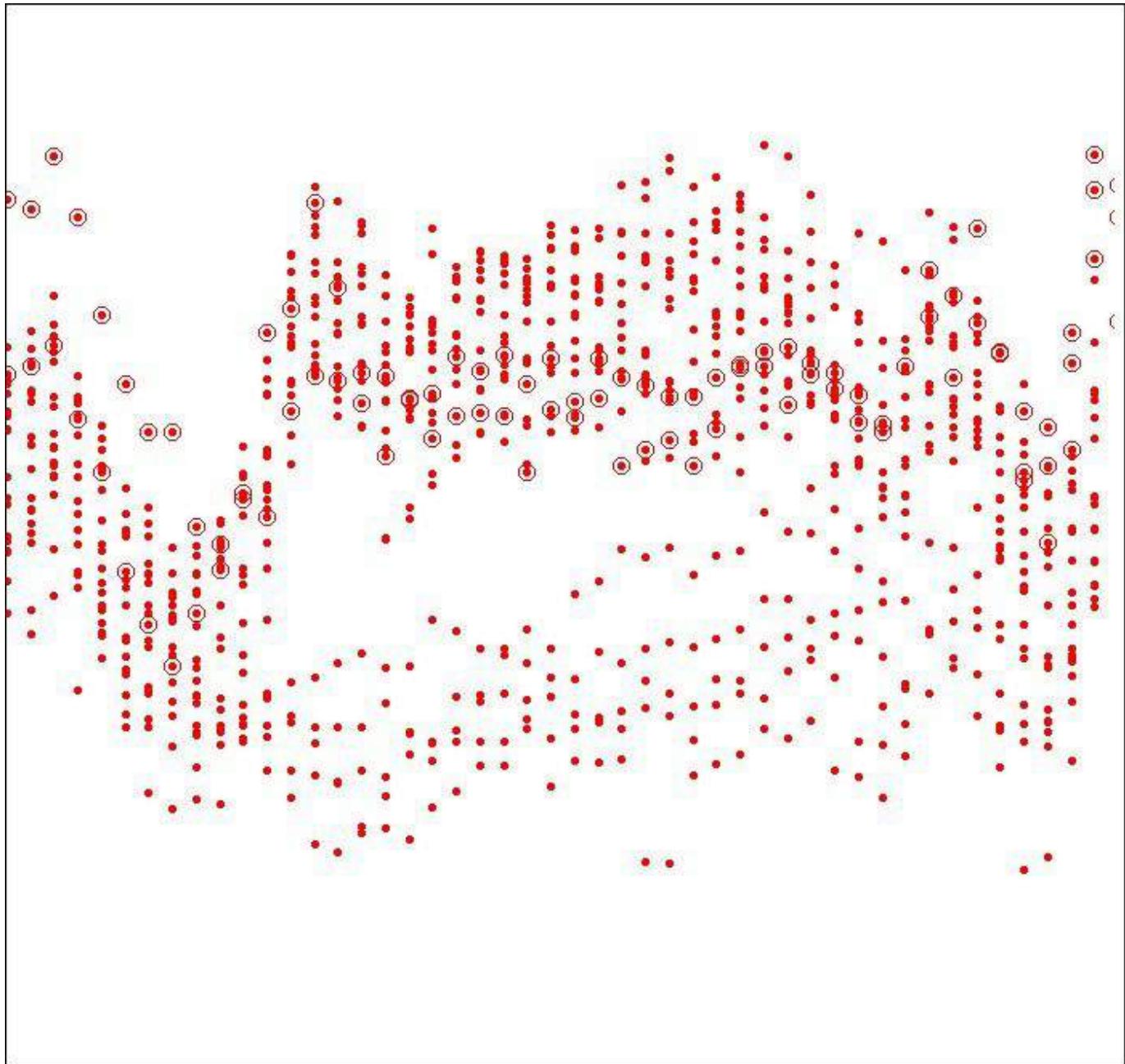
In the final set of experiments, five outlier detection algorithms were used to process

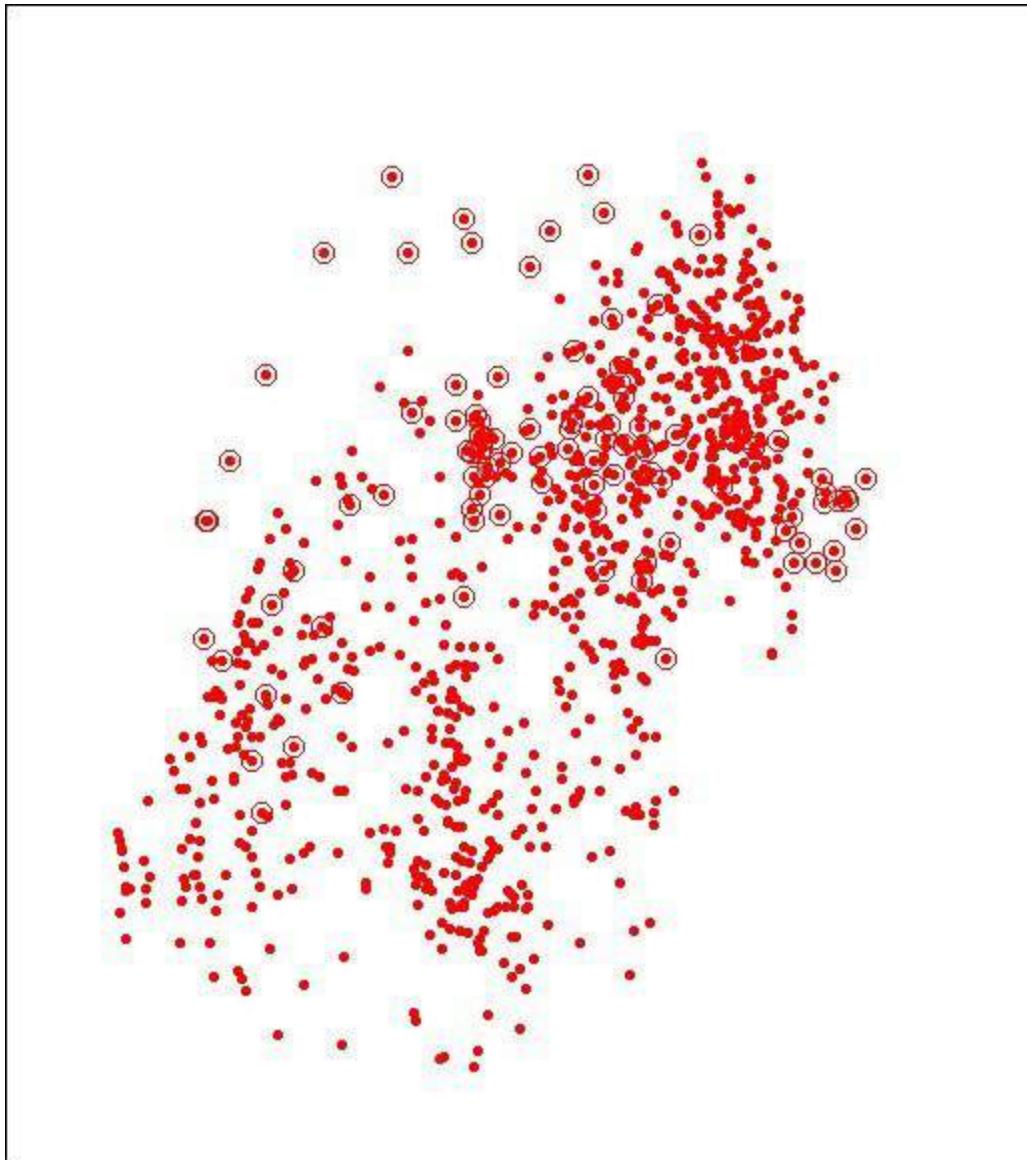
the ELEC dataset. Results are given in *Table 7*:

Algorithm	Nodes always inlier	Nodes always outlier	Nodes both inlier and outlier
MCOD	42449 (93.7%)	302 (0.7%)	2561 (5.7%)
ApproxSTORM	41080 (90.7%)	358 (0.8%)	3874 (8.5%)
SimpleCOD	42449 (93.7%)	302 (0.7%)	2561 (5.7%)
AbstractC	42449 (93.7%)	302 (0.7%)	2561 (5.7%)
ExactSTORM	42449 (93.7%)	302 (0.7%)	2561 (5.7%)

Table 7. Evaluation of outlier detection

The following plots (*Figure 6*) show results for three pairs of features after running algorithm Abstract-C on the entire dataset. In each of the plots, it is easy to see the outliers identified by the circles surrounding the data points. Although it is difficult to visualize the outliers spatially in multiple dimensions simultaneously, the set bivariate plots give some indication of the result of outlier detection methods applied in a stream-based setting:





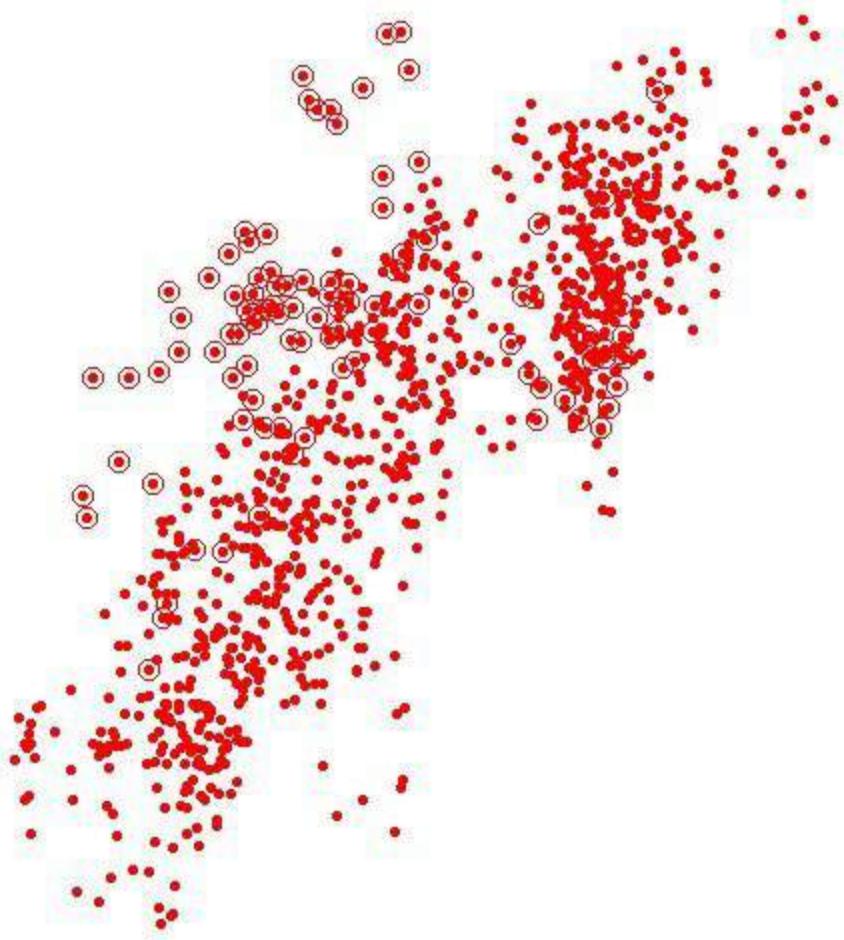


Figure 6. Outlier detection using Abstract-C, for three pairs of features, after processing all 45,300 instances

The image in *Figure 7* shows a screenshot from MOA when two algorithms, `Angiulli.ExactSTORM` and `Angiulli.ApproxSTORM`, were run simultaneously; a bivariate scatter-plot for each algorithm is shown side-by-side, accompanied by a comparison of the per-object processing time:

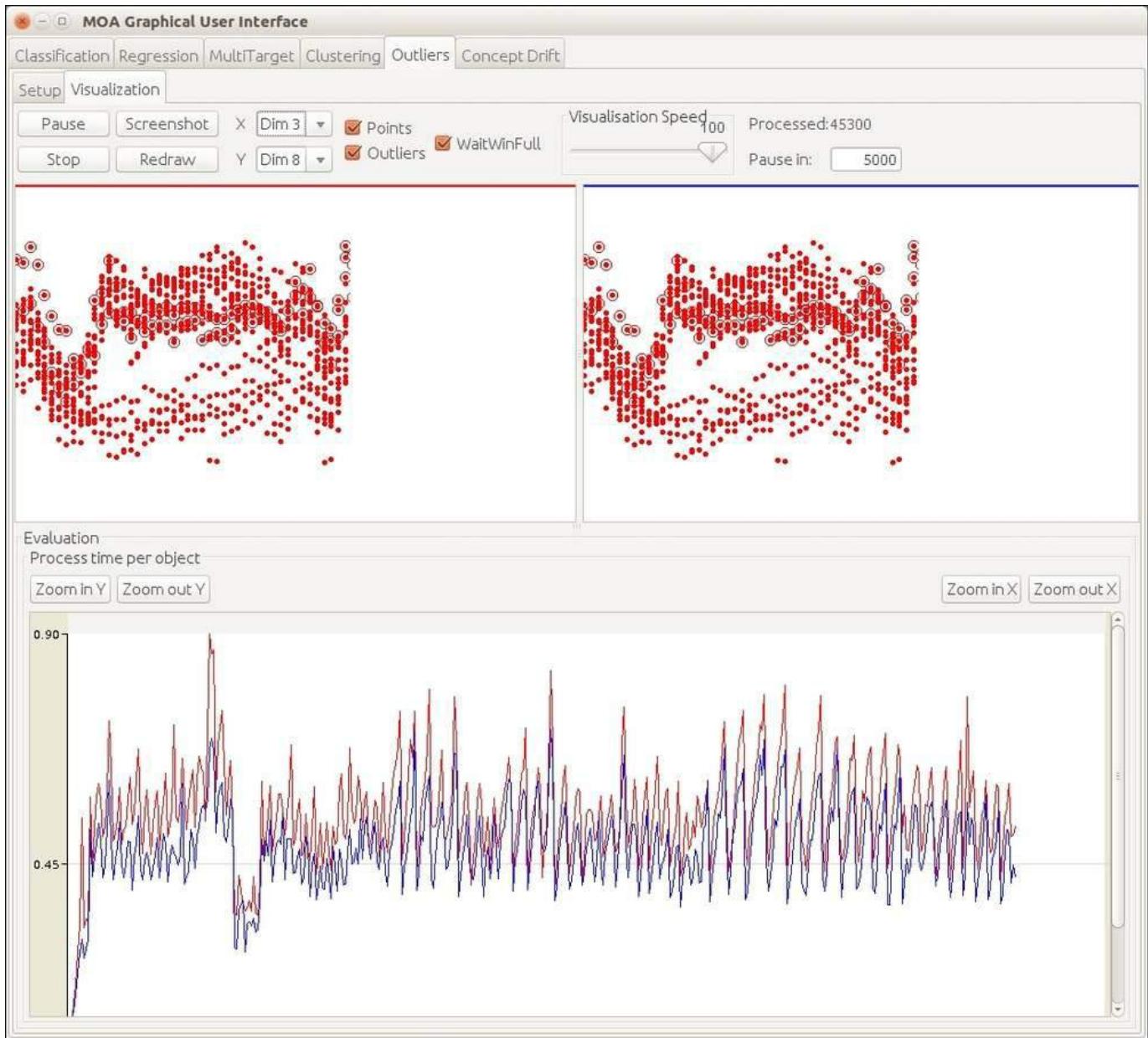


Figure 7. Visualization of outlier detection in MOA

Analysis of stream learning results

Based on the evaluation of the learned models from the classification, clustering, and outlier detection experiments, analysis reveals several interesting observations.

Classification experiments:

- Performance improves from linear to non-linear algorithms in quite a significant way as shown in Table 3. Linear SGD has the best performance using an accuracy metric of 67%, while KNN and Hoeffding Tree show 82.4 to 93%. This clearly shows that the problem is non-linear and using a non-linear algorithm will give better performance.
- K-NNs give good performance but come at the cost of evaluation time, as shown in Table 3. Evaluation time as well as memory are significantly higher—about two orders of magnitude—than the linear methods. When the model has to perform in tighter evaluation cycles, extreme caution in choosing algorithms such as KNNs must be observed.
- Hoeffding Trees gives the best classification rate and Kappa statistic. The evaluation time is also not as high as KNNs but is still in the order of seconds, which may or may not be acceptable in many real-time stream-based applications.
- The evaluation time of Naive Bayes is the lowest—though not much different from SGD—and with the right choice of window width can give performance second only to Hoeffding Trees. For example, at width 100, we have a classification rate of 86 with Naïve Bayes, which is next best to 93 of Hoeffding Trees but compared to over 13 seconds, Naïve Bayes takes only 0.76 seconds, as shown in *Table 3*.
- Keeping the window width constant, there is a clear pattern of improvement from linear (SGD, Naïve Bayes) to non-linear (Hoeffding Trees) to ensemble based (OzaBag, Adwin, Hoeffding Tree) shown in Table 4. This clearly shows that in theory, the choice of ensembles can help reduce the errors but comes at the cost of foregoing interpretability in the models.
- *Table 5*, when compared to *Table 3* and *Table 4*, shows why having drift protection and learning with automated drift detection increases robustness. Ensemble-based learning of OzaBag-NaiveBayes, OzaBag-HoeffdingTrees, and OzaBag-HoeffdingAdaptiveTree all show improvements over the non- drift protected runs as an example.

Clustering experiments:

- From the first two models in *Table 6*, we see that k-Means, with a horizon of 5,000 instances and k of 2, exhibits higher purity, higher CMM, and lower SSQ compared to the model with a smaller horizon and k of 5. In the complete set of results (available on this book's website, see link below), one can see that the effect of the larger horizon is the predominant factor responsible for the differences.
- In the clustering models using micro-clustering, the SSQ is typically significantly smaller than when no micro-clustering is used. This is understandable, as there are far more clusters and fewer instances per cluster, and SSQ is measured with respect to the cluster center.
- DBSCAN was found to be insensitive to micro-clustering, and size of the horizon. Compared to all other models, it ranks high on both intrinsic (Silhouette coefficient) as well as extrinsic measures (Completeness, Purity).
- The two ClusTree models have among the best CMM and Purity scores, with low SSQ due to micro-clusters.
- The final two outlier-based clustering algorithms have perfect CMM and Purity scores. The metrics are not significantly affected by the window size (although this impacts the evaluation time), or the value of k , the neighbor count threshold.

Outlier detection experiments:

- All the techniques in this set of experiments performed equally well, with the exception of ApproxSTORM, which is to be expected considering the reduced window used in this method compared to the exact version.
- The ratio of instances, always inlier versus those always outlier, is close to 140 for the majority of the models. Whether this implies adequate discriminatory power for a given dataset depends on the goals of the real-time learning problem.

Note

All MOA configuration files and results from the experiments are available at:
<https://github.com/mjmlbook/mastering-java-machine-learning/Chapter5>.

Summary

The assumptions in stream-based learning are different from batch-based learning, chief among them being upper bounds on operating memory and computation times. Running statistics using sliding windows or sampling must be computed in order to scale to a potentially infinite stream of data. We make the distinction between learning from stationary data, where it is assumed the generating data distribution is constant, and dynamic or evolving data, where concept drift must be accounted for. This is accomplished by techniques involving the monitoring of model performance changes or the monitoring of data distribution changes. Explicit and implicit adaptation methods are ways to adjust to the concept change.

Several supervised and unsupervised learning methods have been adapted for incremental online learning. Supervised methods include linear, non-linear, and ensemble techniques, The HoeffdingTree is introduced which is particularly interesting due largely in part to its guarantees on upper bounds on error rates. Model validation techniques such as prequential evaluation are adaptations unique to incremental learning. For stationary supervised learning, evaluation measures are similar to those used in batch-based learning. Other measures are used in the case of evolving data streams.

Clustering algorithms operating under fixed memory and time constraints typically use small memory buffers with standard techniques in a single pass. Issues specific to streaming must be considered during evaluations of clustering, such as aging, noise, and missed or misplaced points. Outlier detection in data streams is a relatively new and growing field. Extending ideas in clustering to anomaly detection has proved very effective.

The experiments in the case study in this chapter use the Java framework MOA, illustrating various stream-based learning techniques for supervised, clustering, and outlier detection.

In the next chapter, we embark on a tour of the probabilistic graph modelling techniques that are useful in representing, eliciting knowledge, and learning in various domains.

References

1. G. Cormode and S. Muthukrishnan (2010). *An improved data stream summary: The Count-Min sketch and its applications*. Journal of Algorithms, 55(1):58–75, 2005.
2. João Gama (2010). *Knowledge Discovery from Data Streams*, Chapman and Hall / CRC Data Mining and Knowledge Discovery Series, CRC Press 2010, ISBN 978-1-4398-2611-9, pp. I-XIX, 1-237.
3. B. Babcock, M. Datar, R. Motwani (2002). *Sampling from a moving window over streaming data*, in Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pp.633–634, 2002.
4. Bifet, A. and Gavalda, R. (2007). *Learning from time-changing data with adaptive windowing*. In Proceedings of SIAM int. conf. on Data Mining. SDM. 443–448.
5. Vitter, J. (1985). *Random sampling with a reservoir*. *ACM Trans. Math. Softw.* 11, 1, 37–57.
6. Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). *Learning with drift detection*. In Proceedings of the 17th Brazilian symp. on Artif. Intell. SBIA. 286–295.
7. Gama, J., Sebastiao, R., and Rodrigues, P. 2013. *On evaluating stream learning algorithms*. Machine Learning 90, 3, 317–346.
8. Domingos, P. and Hulten, G. (2000). *Mining high-speed data streams*. In Proceedings. of the 6th ACM SIGKDD int. conference on Knowledge discovery and data mining. KDD. 71–80.
9. Oza, N. (2001). *Online ensemble learning*. Ph.D. thesis, University of California Berkeley.
10. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A. (2014). *A Survey on Concept Drift Adaptation*. *ACM Computing Surveys* 46(4), Article No. 44.
11. Farnstrom, F., Lewis, J., and Elkan, C. (2000). *Scalability for clustering algorithms revisited*. SIGKDD Exploration, 51–57.
12. Zhang, T., Ramakrishnan, R., and Livny, M. (1996). *BIRCH: An Efficient Data Clustering Method for Very Large Databases*. In Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM Press, 103–114.
13. Aggarwal, C. (2003). *A Framework for Diagnosing Changes in Evolving Data Streams*. In ACM SIGMOD Conference. 575–586.
14. Chen, Y. and Tu, L. (2007). *Density-based clustering for real-time stream data*.

- In KDD '07: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, 133–142.
- 15. Kremer, H., Kranen, P., Jansen, T., Seidl, T., Bifet, A., Holmes, G., and Pfahringer, B. (2011). *An effective evaluation measure for clustering on evolving data streams*. In proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '11. ACM, New York, NY, USA, 868–876.
 - 16. Mahdiraji, A. R. (2009). *Clustering data stream: A survey of algorithms*. International Journal of Knowledge-Based and Intelligent Engineering Systems, 39–44.
 - 17. F. Angiulli and F. Fassetti (2007). *Detecting distance-based outliers in streams of data*. In proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM '07, pages 811–820, New York, NY, USA, 2007. ACM.
 - 18. D. Yang, E. A. Rundensteiner, and M. O. Ward (2009). *Neighbor-based pattern detection for windows over streaming data*. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '09, pages 529–540, New York, NY, USA, 2009. ACM.
 - 19. M. Kontaki, A. Gounaris, A. Papadopoulos, K. Tsichlas, and Y. Manolopoulos (2011). *Continuous monitoring of distance-based outliers over data streams*. In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pages 135–146, April 2011.

Chapter 6. Probabilistic Graph Modeling

Probabilistic graph models (PGMs), also known as graph models, capture the relationship between different variables and represent the probability distributions. PGMs capture joint probability distributions and can be used to answer different queries and make inferences that allow us to make predictions on unseen data. PGMs have the great advantage of capturing domain knowledge of experts and the causal relationship between variables to model systems. PGMs represent the structure and they can capture knowledge in a representational framework that makes it easier to share and understand the domain and models. PGMs capture the uncertainty or the probabilistic nature very well and are thus very useful in applications that need scoring or uncertainty-based approaches. PGMs are used in a wide variety of applications that use machine learning such as applications to domains of language processing, text mining and information extraction, computer vision, disease diagnosis, and DNA structure predictions.

Judea Pearl is the pioneer in the area of PGMs and was the first to introduce the topic of Bayesian Networks (*References* [2] and [7]). Although covering all there is to know about PGMs is beyond the scope of this chapter, our goal is to cover the most important aspects of PGMs—Bayes network and directed PGMs—in some detail. We will divide the subject into the areas of representation, inference, and learning and will discuss specific algorithms and sub-topics in each of these areas. We will cover Markov Networks and undirected PGMs, summarizing some differences and similarities with PGMs, and addressing related areas such as inference and learning. Finally, we will discuss specialized networks such as **tree augmented networks (TAN)**, Markov chains and **hidden Markov models (HMM)**. For an in-depth treatment of the subject, see *Probabilistic Graphical Models*, by Koller and Friedman (*References* [1]).

Probability revisited

Many basic concepts of probability are detailed in [Appendix B, *Probability*](#). Some of the key ideas in probability theory form the building blocks of probabilistic graph models. A good grasp of the relevant theory can help a great deal in understanding PGMs and how they are used to make inferences from data.

Concepts in probability

In this section, we will discuss important concepts related to probability theory that will be used in the discussion later in this chapter.

Conditional probability

The essence of conditional probability, given two related events α and β , is to capture how we assign a value for one of the events when the other is known to have occurred. The conditional probability, or the conditional distribution, is represented by $P(\alpha | \beta)$, that is, the probability of event α happening given that the event β has occurred (equivalently, given that β is true) and is formally defined as:

$$P(\alpha | \beta) = \frac{P(\alpha \cap \beta)}{P(\beta)}$$

The $P(\alpha \cap \beta)$ captures the events where both α and β occur.

Chain rule and Bayes' theorem

The conditional probability definition gives rise to the chain rule of conditional probabilities that says that when there are multiple events $\alpha_1, \alpha_2, \dots, \alpha_n$ then:

$$P(\alpha_1 \cap \alpha_2 \cap \dots \cap \alpha_n) = P(\alpha_1)P(\alpha_2 | \alpha_1)P(\alpha_3 | \alpha_1 \cap \alpha_2) \dots P(\alpha_n | \alpha_1 \cap \alpha_2 \cap \dots \cap \alpha_{n-1})$$

The probability of several events can be expressed as the probability of the first times the probability of the second given the first, and so on. Thus, the probability of α_n depends on everything α_1 to α_n and is independent of the order of the events.

Bayes rule also follows from the conditional probability rule and can be given formally as:

$$P(\alpha|\beta) = \frac{P(\beta|\alpha)P(\alpha)}{P(\alpha\beta)}$$

Random variables, joint, and marginal distributions

It is natural to map the event spaces and outcomes by considering them as attributes and values. Random variables are defined as attributes with different known specific values. For example, if *Grade* is an attribute associated with *Student*, and has values $\{A, B, C\}$, then $P(Grade = A)$ represents a random variable with an outcome.

Random variables are generally denoted by capital letters, such as X , Y , and Z and values taken by them are denoted by $Val(X) = x$. In this chapter, we will primarily discuss values that are categorical in nature, that is, that take a fixed number of discrete values. In the real world, the variables can have continuous representations too. The distribution of a variable with categories $\{x^1, x^2 \dots x^n\}$ can be represented as:

$$\sum_{i=0}^n P(X = x^i) = 1$$

Such a distribution over many categories is called a **multinomial distribution**. In the special case when there are only two categories, the distribution is said to be the **Bernoulli distribution**.

Given a random variable, a probability distribution over all the events described by that variable is known as the marginal distribution. For example, if Grade is the random variable, the marginal distribution can be defined as $(Grade = A) = 0.25$, $P(Grade = b) = 0.37$ and $P(Grade = C) = 0.38$.

In many real-world models, there are more than one random variables and the distribution that considers all of these random variable is called the **joint distribution**. For example, if *Intelligence* of student is considered as another

variable and denoted by $P(\text{Intelligence})$ or $P(I)$ and has binary outcomes $\{\text{low}, \text{high}\}$, then the distribution considering *Intelligence* and *Grade* represented as $P(\text{Intelligence}, \text{Grade})$ or $P(I, G)$, is the joint distribution.

Marginal distribution of one random variable can be computed from the joint distribution by summing up the values over all of the other variables. The marginal distribution over grade can be obtained by summing over all the rows as shown in *Table 1* and that over the intelligence can be obtained by summation over the columns.

		<i>Intelligence or I</i>		$P(G) = \sum_I G, I$
		low	high	
<i>Grade or G</i>	A	0.07	0.18	=0.25
	B	0.28	0.09	=0.37
	C	0.35	0.03	=0.38
		=0.7	=0.3	

$P(I) = \sum_G (I, G)$

Table 1. Marginal distributions over I and G

Marginal independence and conditional independence

Marginal Independence is defined as follows. Consider two random variables X and Y ; then $P(X|Y) = P(X)$ means random variable X is independent of Y . It is formally represented as $P = (X \perp Y)$ (P satisfies X is independent of Y).

This means the joint distribution can be given as:

$$P(X, Y) = P(X)P(Y)$$

If the difficulty level of the exam (D) and the intelligence of the student (I) determine the grade (G), we know that the difficulty level of the exam is independent of the intelligence of the student and $(D \perp I)$ also implies $P(D, I) = P(D)P(I)$.

When two random variables are independent given a third variable, the independence is called conditional independence. Given a set of three random variables X , Y , and

Z , we can say that $P(X \perp Y|Z)$; that is, variable X is independent of Y given Z . The necessary condition for conditional independence is

$$P(X \perp Y|Z) = P(X|Z)P(Y|Z)$$

Factors

Factors are the basic building blocks for defining the probability distributions in high-dimensional (large number of variables) spaces. They give basic operations that help in manipulating the probability distributions.

A "factor" is defined as a function that takes as input the random variables known as "scope" and gives a real-value output.

Formally, a factor is represented as $\phi(X_1, X_2, \dots, X_k)$ where $\phi : Val(X_1, X_2, \dots, X_k) \rightarrow \mathbb{R}$ and scope is (X_1, X_2, \dots, X_k) .

Factor types

Different types of factors are as follows:

- **Joint distribution:** For every combination of variables, you get a real-valued output.
- **Unnormalized measure:** When, in a joint distribution, one of the variables is constant, the output is also real-valued, but it is unnormalized as it doesn't sum to one. However, it is still a factor.
- **Conditional probability distribution:** A probability distribution of the form $P(G|I)$ is also a factor.

Various operations are performed on factors, such as:

- **Factor product:** If two factors $\phi_1(X_1, X_2)$ and $\phi_2(X_2, X_3)$ are multiplied, it gives rise to $\phi_3(X_1, X_2, X_3)$. In effect, it is taking tables corresponding to ϕ_1 and multiplying it with ϕ_2
- **Factor marginalization:** This is the same as marginalization, where $\phi_1(X_1, X_2, X_3)$ can be marginalized over a variable, say X_2 , to give $\phi_2(X_1, X_3)$.
- **Factor reduction:** This is only taking the values of other variables when one of the variables is constant.

Distribution queries

Given the probability over random variables, many queries can be performed to answer certain questions. Some common types of queries are explained in the subsequent sections.

Probabilistic queries

This is one of the most common types of query and it has two parts:

- **Evidence:** A subset of variables with a well-known outcome or category. For example, a random variable $E = e$.
- **Query:** A random variable from the rest of the variables. For example, a random variable X .

$$P(X|E = e)$$

Examples of probabilistic queries are posterior marginal estimations such as $P(I = \text{high}|L = \text{bad}, S = \text{low}) = ?$ and evidential probability such as $P(L = \text{bad}, S = \text{low}) = ?$.

MAP queries and marginal MAP queries

MAP queries are used to find the probability assignment to the subset of variables that are most likely and hence are also called **most probable explanation (MPE)**. The difference between these and probabilistic queries is that, instead of getting the probability, we get the most likely values for all the variables.

Formally, if we have variables $W = X - E$, where we have $E = e$ as the evidence and are interested in finding the most likely assignment to the variables in W ,

$$MAP(\mathbf{W}|\mathbf{e}) = \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w}, \mathbf{e})$$

A much more general form of marginal query is when we have a subset of variables, say given by Y that forms our query and with evidence of $E = e$, we are interested in finding most likely assignments to the variables in Y . Using the MAP definition, we get:

$$MAP(\mathbf{Y}|\mathbf{e}) = \operatorname{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{e})$$

Let's say, $Z = X - Y - E$, then the marginal MAP query is:

$$MAP(Y|e) = \operatorname{argmax}_y \sum_z P(Y, Z | e)$$

Graph concepts

Next, we will briefly revisit the concepts from graph theory and some of the definitions that we will use in this chapter.

Graph structure and properties

A graph is defined as a data structure containing nodes and edges connecting these nodes. In the context of this chapter, the random variables are represented as nodes, and edges show connections between the random variables.

Formally, if $X = \{X_1, X_2, \dots, X_k\}$ where X_1, X_2, \dots, X_k are random variables representing the nodes, then there can either be a directed edge belonging to the set e , for example, between the nodes given by $X_i \rightarrow X_j$ or an **undirected edge** $X_i \rightarrow X_j$, and the graph is defined as a data structure $k = (x, e)$. A graph is said to be a **directed graph** when every edge in the set e between nodes from set X is directed and similarly an **undirected graph** is one where every edge between the nodes is undirected as shown in *Figure 1*. Also, if there is a graph that has both directed and undirected edges, the notation of $X_i \leftrightarrow X_j$ represents an edge that may be directed or undirected.

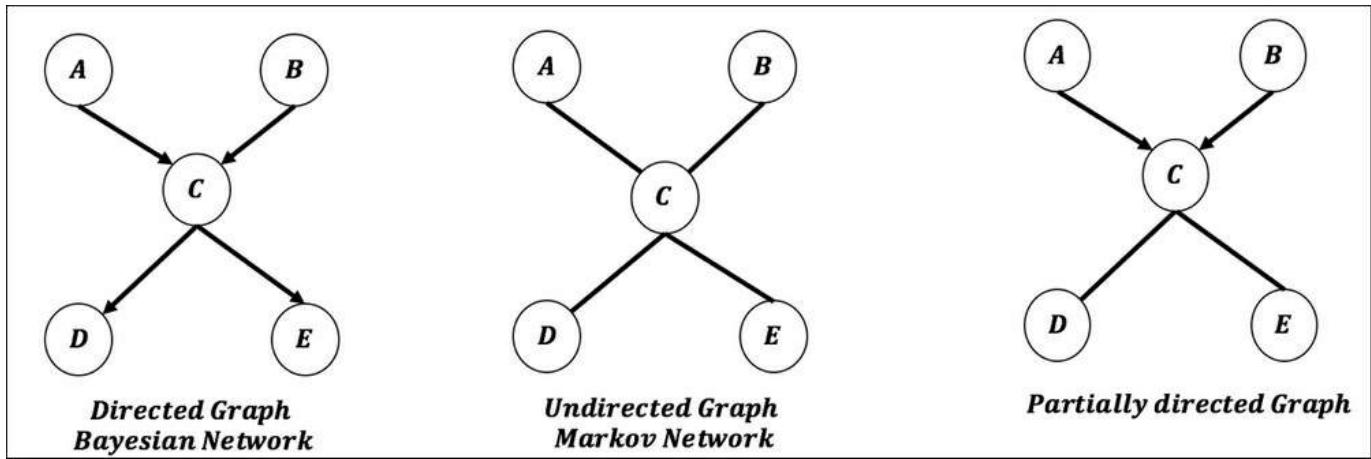


Figure 1. Directed, undirected, and partially-directed graphs

If a directed edge $X_i \rightarrow X_j \in e$ exists in the graph, the node X_i is called the *parent* node and the node X_j is called the *child* node.

In the case of an undirected graph, if there is an edge $X_i - X_j$, the nodes X_i and X_j are said to be neighbors.

The set of parents of node X in a directed graph is called the boundary of the node X and similarly, adjacent nodes in an undirected graph form each other's boundary. The degree of the node X is the number of edges it participates in. The indegree of the node X is the number of edges in the directed graph that have a relationship to node X such that the edge is between node Y and node X and $X \rightarrow Y$. The degree of the graph is the maximal degree of the node in that graph.

Subgraphs and cliques

A subgraph is part of the graph that represents some of the nodes from the entire set. A **clique** is a subset of vertices in an undirected graph such that every two distinct vertices are adjacent.

Path, trail, and cycles

If there are variables X_1, X_2, \dots, X_k in the graph $K = (X, E)$, it forms a path if, for every $i = 1, 2 \dots k - 1$, we have either $X_i \rightarrow X_{i+1}$ or $X_i - X_j$; that is, there is either a directed edge or an undirected edge between the variables—recall this can be depicted as $X_i ? X_j$. A directed path has at least one directed edge: $X_i \rightarrow X_{i+1}$.

If there are variables X_1, X_2, \dots, X_k in the graph $K = (X, E)$ it forms a *trail* if for every $i = 1, 2 \dots k - 1$, we have either $X_i \Leftarrow X_{i+1}$.

A graph is called a **connected** graph, if for every X_i, \dots, X_j there is a trail between X_i and X_j .

In a graph $K = (X, e)$, if there is a directed path between nodes X and Y , X is called the ancestor of Y and Y is called the *descendant* of X .

If a graph K has a directed path X_1, X_2, \dots, X_k where $X_1 ? X_k$, the path is called a **cycle**. Conversely, a graph with no cycles is called an **acyclic** graph.

Bayesian networks

Generally, all Probabilistic Graphical Models have three basic elements that form the important sections:

- **Representation:** This answers the question of what does the model mean or represent. The idea is how to represent and store the probability distribution of $P(X_1, X_2, \dots, X_n)$.
- **Inference:** This answers the question: given the model, how do we perform queries and get answers. This gives us the ability to infer the values of the unknown from the known evidence given the structure of the models.
Motivating the main discussion points are various forms of inferences involving trade-offs between computational and correctness concerns.
- **Learning:** This answers the question of what model is right given the data.
Learning is divided into two main parts:
 - Learning the parameters given the structure and data
 - Learning the structure with parameters given the data

We will use the well-known student network as an example of a Bayesian network in our discussions to illustrate the concepts and theory. The student network has five random variables capturing the relationship between various attributes defined as follows:

- Difficulty of the exam (D)
- Intelligence of the student (I)
- Grade the student gets (G)
- SAT score of the student (S)
- Recommendation Letter the student gets based on grade (L).

Each of these attributes has binary categorical values, for example, the variable *Difficulty* (D) has two categories (d_0, d_1) corresponding to low and high, respectively. *Grades* (G) has three categorical values corresponding to the grades (A, B, C). The arrows as indicated in the section on graphs indicate the dependencies encoded from the domain knowledge—for example, *Grade* can be determined given that we know the *Difficulty* of the exam and *Intelligence* of the student while the *Recommendation Letter* is completely determined if we know just the *Grade* (Figure 2). It can be further observed that no explicit edge between the variables indicates that they are independent of each other—for example, the *Difficulty* of the exam and

Intelligence of the student are independent variables.

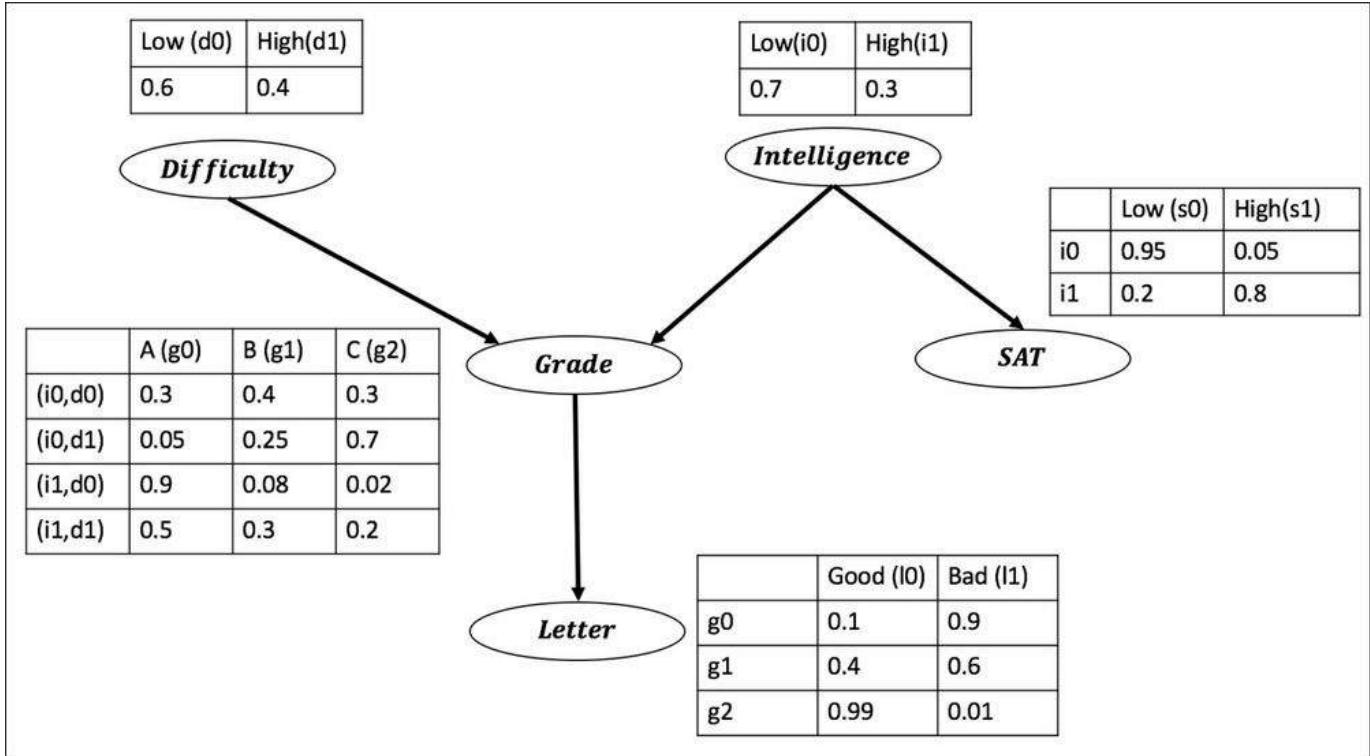


Figure 2. The "Student" network

Representation

A graph compactly represents the complex relationships between random variables, allowing fast algorithms to make queries where a full enumeration would be prohibitive. In the concepts defined here, we show how directed acyclic graph structures and conditional independence make problems involving large numbers of variables tractable.

Definition

A Bayesian network is defined as a model of a system with:

- A number of random variables $\{X_1, X_2, \dots, X_k\}$
- A **Directed Acyclic Graph (DAG)** with nodes representing random variables.
- A local **conditional probability distribution (CPD)** for each node with dependence to its parent nodes $P(X_i | \text{parent}(X_i))$.
- A joint probability distribution obtained using the chain rule of distribution is a factor given as:

$$P(X_1, X_2, \dots, X_k) = \prod_{i=1}^k P(X_i | \text{parent}(X_i))$$

- For the student network defined, the joint distribution capturing all nodes can be represented as:

$$P(D, I, G, S, L) = P(D)P(I)P(G|D, I)P(S|I)P(L|G)$$

Reasoning patterns

The Bayesian networks help in answering various queries given some data and facts, and these reasoning patterns are discussed here.

Causal or predictive reasoning

If evidence is given as, for example, "low intelligence", then what would be the chances of getting a "good letter" as shown in *Figure 3*, in the top right quadrant? This is addressed by causal reasoning. As shown in the first quadrant, causal reasoning flows from the top down.

Evidential or diagnostic reasoning

If evidence such as a "bad letter" is given, what would be the chances that the student got a "good grade"? This question, as shown in *Figure 3* in the top left quadrant, is addressed by evidential reasoning. As shown in the second quadrant, evidential reasoning flows from the bottom up.

Intercausal reasoning

Obtaining interesting patterns from finding a "related cause" is the objective of intercausal reasoning. If evidence of "grade C" and "high intelligence" is given, then what would be the chance of course difficulty being "high"? This type of reasoning is also called "explaining away" as one cause explains the reason for another cause and this is illustrated in the third quadrant, in the bottom-left of *Figure 3*.

Combined reasoning

If a student takes an "easy" course and has a "bad letter", what would be the chances of him getting a "grade C"? This is explained by queries with combined reasoning patterns. Note that it has mixed information and does not flow in a single fixed direction as in the case of other reasoning patterns and is shown in the bottom-right of the figure, in quadrant 4:

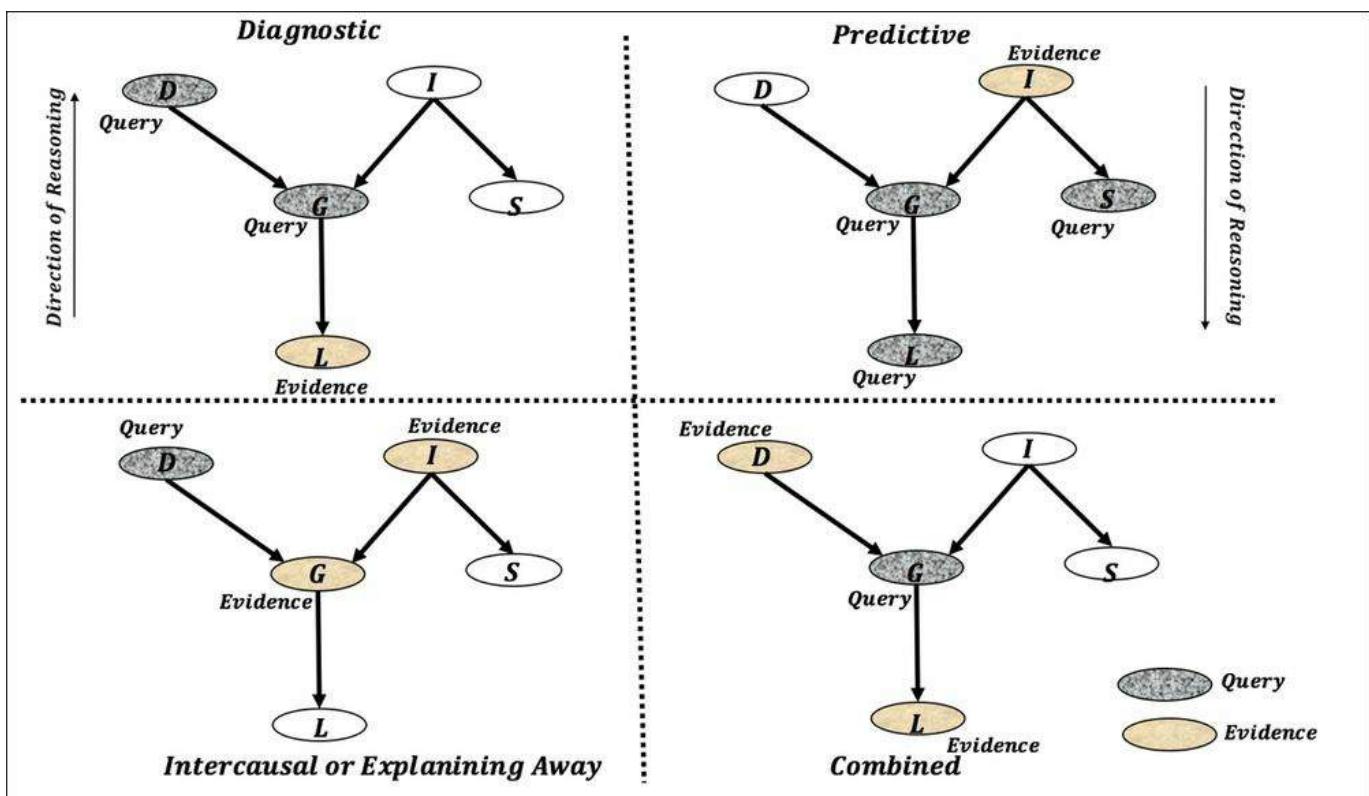


Figure 3. Reasoning patterns

Independencies, flow of influence, D-Separation, I-Map

The conditional independencies between the nodes can be exploited to reduce the computations when performing queries. In this section, we will discuss some of the important concepts that are associated with independencies.

Flow of influence

Influence is the effect of how the condition or outcome of one variable changes the value or the belief associated with another variable. We have seen this from the reasoning patterns that influence flows from variables in direct relationships (parent/child), causal/evidential (parent and child with intermediates) and in combined structures.

The only case where the influence doesn't flow is when there is a "v-structure". That is, given edges between three variables $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ there is a v-structure and no influence flows between X_{i-1} and X_{i+1} . For example, no influence flows between the Difficulty of the course and the Intelligence of the student.

D-Separation

Random variables X and Y are said to be d-separated in the graph \mathbf{G} , given there is no active trail between \mathbf{X} and \mathbf{Y} in \mathbf{G} given \mathbf{Z} . It is formally denoted by:

$$dsep_{\mathbf{G}}(X, Y | Z)$$

The point of d-separation is that it maps perfectly to the conditional independence between the points. This gives to an interesting property that in a Bayesian network any variable is independent of its non-descendants given the parents of the node.

In the Student network example, the node/variable Letter is d-separated from Difficulty, Intelligence, and SAT given the grade.

I-Map

From the d-separation, in graph \mathbf{G} , we can collect all the independencies from the d-

separations and these independencies are formally represented as:

$$I(\mathbf{G}) = \left\{ (X \perp Y | Z) : dsep_G(X, Y | Z) \right\}$$

If P satisfies $I(\mathbf{G})$ then we say the \mathbf{G} is an independency-map or I-Map of P .

The main point of I-Map is it can be formally proven that a factorization relationship to the independency holds. The converse can also be proved.

In simple terms, one can read in the Bayesian network graph G , all the independencies that hold in the distribution P regardless of any parameters!

Consider the student network—its whole distribution can be shown as:

$$P(D, I, G, S, L) = P(D)P(I|D)P(G|D, I)P(S|D, I, G)P(L|D, I, G, S)$$

Now, consider the independence from I-Maps:

- Variables I and D are non-descendants and not conditional on parents so $P(I|D) = P(I)$
- Variable S is independent of its non-descendants D and G , given its parent I .
 $P(S|D, I, G) = P(S|I)$
- Variable L is independent of its non-descendants D , I , and S , given its parent G .
 $P(L|D, I, G, S) = P(L|G)$

$$(D, I, G, S, L) = P(D)P(I)P(G|D, I)P(S|I)P(L|G)$$

Thus, we have shown that I-Map helps in factorization given just the graph network!

Inference

The biggest advantage of probabilistic graph models is their ability to answer probability queries in the form of conditional or MAP or marginal MAP, given some evidence.

Formally, the probability of evidence $\mathbf{E} = \mathbf{e}$ is given by:

$$P(\mathbf{E} = \mathbf{e}) = \sum_{\mathbf{X}/\mathbf{E}} \prod_{i=1}^n P(X_i | \text{parent}(X_i)|_{\mathbf{E}=\mathbf{e}})$$

But the problem has been shown to be NP-Hard (*Reference [3]*) or specifically, #P-complete. This means that it is intractable when there are a large number of trees or variables. Even for a tree-width (number of variables in the largest clique) of 25, the problem seems to be intractable—most real-world models have tree-widths larger than this.

So if the exact inference discussed before is intractable, can some approximations be used so that within some bounds of the error, we can make the problem tractable? It has been shown that even an approximate algorithm to compute inferences with an error $\epsilon < 0.5$, so that we find a number p such that $|P(\mathbf{E} = \mathbf{e}) - p| < \epsilon$, is also NP-Hard.

But the good news is that this is among the "worst case" results that show exponential time complexity. In the "general case" there can be heuristics applied to reduce the computation time both for exact and approximate algorithms.

Some of the well-known techniques for performing exact and approximate inferencing are depicted in *Figure 4*, which covers most probabilistic graph models in addition to Bayesian networks.

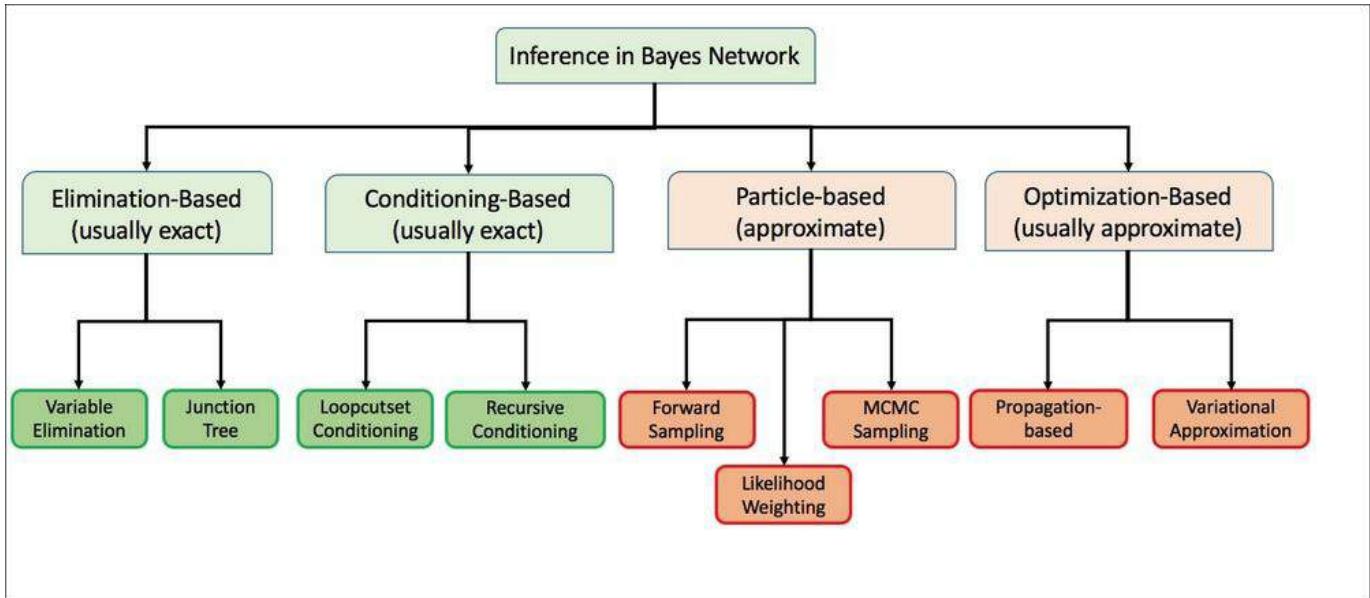


Figure 4. Exact and approximate inference techniques

It is beyond the scope of this chapter to discuss each of these in detail. We will explain a few of the algorithms in some detail accompanied by references to give the reader a better understanding.

Elimination-based inference

Here we will describe two techniques, the variable elimination algorithm and the clique-tree or junction-tree algorithm.

Variable elimination algorithm

The basics of the **Variable elimination (VE)** algorithm lie in the distributive property as shown:

$$(ab+ac+ad) = a(b+c+d)$$

In other words, five arithmetic operations of three multiplications and two additions can be reduced to four arithmetic operations of one multiplication and three additions by taking a common factor a out.

Let us understand the reduction of the computations by taking a simple example in the student network. If we have to compute a probability query such as the difficulty

of the exam given the letter was good, that is, $P(D|L=good)=?$.

Using Bayes theorem:

$$P(D|L=good) = \frac{P(D, L=good)}{P(L=good)}$$

To compute $P(D|L=good)=?$ we can use the chain rule and joint probability:

$$P(D, L=good) = \sum_{G,I,S} P(S|I)P(I)P(G|I,D)P(L=good|G)$$

If we rearrange the terms on the right-hand side:

$$P(D, L=good) = \sum_{G,I} P(I)P(G|I,D)P(L=good|G)\sum_S P(S|I)$$

If we now replace $\sum_S P(S|I) = \phi_I(S)$ since the factor is independent of the variable I that S is conditioned on, we get:

$$P(D, L=good) = \sum_{G,I} P(I)P(G|I,D)P(L=good|G)\phi_I(S)$$

Thus, if we proceed carefully, eliminating one variable at a time, we have effectively converted $O(2^n)$ factors to $O(nk^2)$ factors where n is the number of variables and k is the number of observed values for each.

Thus, the main idea of the VE algorithm is to impose an order on the variables such

that the query variable comes last. A list of factors is maintained over the ordered list of variables and summation is performed. Generally, we use dynamic programming in the implementation of the VE algorithm (*References* [4]).

Input and output

Inputs:

- List of Condition Probability Distribution/Table **F**
- List of query variables **Q**
- List of observed variables **E** and the observed value **e**

Output:

- $P(\mathbf{Q}|\mathbf{E} = e)$

How does it work?

The algorithm calls the `eliminate` function in a loop, as shown here:

VariableElimination:

1. While \mathcal{Z} , the set of all random variables in the Bayesian network is not empty
 1. Remove the first variable **Z** from \mathcal{Z}
 2. $\text{eliminate}(F, \mathbf{Z})$
2. end loop.
3. Set \mathcal{Z} product of all factors in F
4. Instantiate observed variables in \mathcal{Z} to their observed values.
5. return $\phi_i(\mathbf{Q}) / \sum_{\mathbf{Q}} \phi(\mathbf{Q})$ (renormalization)

eliminate (F, \mathbf{Z})

1. Remove from the F all functions, for example, X_1, X_2, \dots, X_k that involve **Z**.
2. Compute new function $g = \prod_{i=0}^k X_i$
3. Compute new function $\phi = \sum_z g$
4. Add new function ϕ to F
5. Return F

Consider the same example of the student network with $P(D, L = \text{good})$ as the goal.

1. Pick a variable ordering list: S, I, L, G , and D
2. Initialize the active factor list and introduce the evidence:

List: $P(S|I)P(I)P(D)P(G|I,D)P(L|G)d(L = good)$

3. Eliminate the variable SAT or S off the list

$$\phi_1(I) = \sum_S P(S|I)$$

List: $P(I)P(D)P(G|I,D)P(L|G)d(L = good) ?1 (I)$

4. Eliminate the variable Intelligence or I

$$\phi_2(G, D) = \sum_I P(I)P(G|I, D)\phi_1(I)$$

List: $P(D)P(L|G)d(L = good) ?2 (G, D)$

5. Eliminate the variable Letter or L

$$\phi_3(G) = \sum_L P(L|G)\delta(L = good)$$

List: $P(D) ?_3 (G) ?_2 (G, D)$

6. Eliminate the variable Grade or G

$$\phi_4(D) = \sum_G \phi_3(G)\phi_2(G, D)$$

List: $P(D) ?_4 (D)$

Thus with two values, $P(D=high) ?_4 (D=high)$ and $P(D=low) ?_4 (D=low)$, we get the answer.

Advantages and limitations

The advantages and limitations are as follows:

- The main advantage of the VE algorithm is its simplicity and generality that can be applied to many networks.
- The computational reduction advantage of VE seems to go away when there are many connections in the network.
- The choice of optimal ordering of variables is very important for the computational benefit.

Clique tree or junction tree algorithm

Junction tree or Clique Trees are more efficient forms of variable elimination-based techniques.

Input and output

Inputs:

- List of Condition Probability Distribution/Table **F**
- List of query variables **Q**
- List of observed variables **E** and the observed value **e**

Output:

- $P(\mathbf{Q}|\mathbf{E} = e)$

How does it work?

The steps involved are as follows:

1. **Moralization:** This is a process of converting a directed graph into an undirected graph with the following two steps:
 1. Replace directed edges with undirected edges between the nodes.
 2. If there are two nodes or vertices that are not connected but have a common child, add an edge connecting them. (Note the edge between V_4 and V_5 and V_2 and V_3 in *Figure 5*):

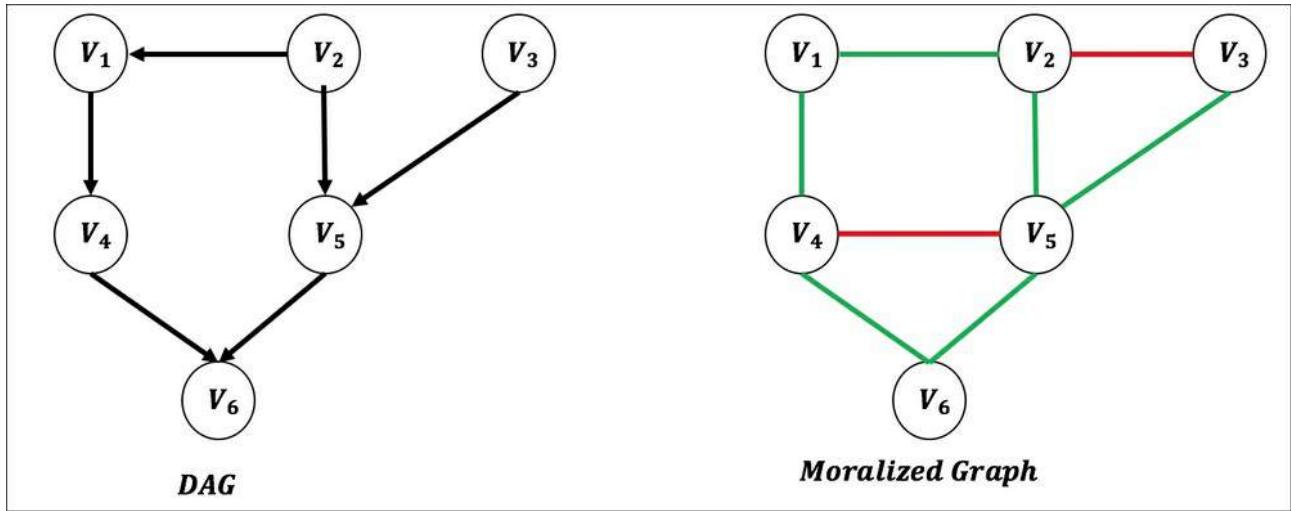


Figure 5. Graph moralization of DAG showing in green how the directional edges are changed and red edges showing new additions.

2. **Triangulation:** For understanding triangulation, chords must be formed. The chord of a cycle is a pair of vertices V_i and V_j of non-consecutive vertices that have an edge between them. A graph is called a **chordal or triangulated graph** if every cycle of length ≥ 4 has chords. Note the edge between V_1 and V_5 in *Figure 6* forming chords to make the moralized graph a chordal/triangulated graph:

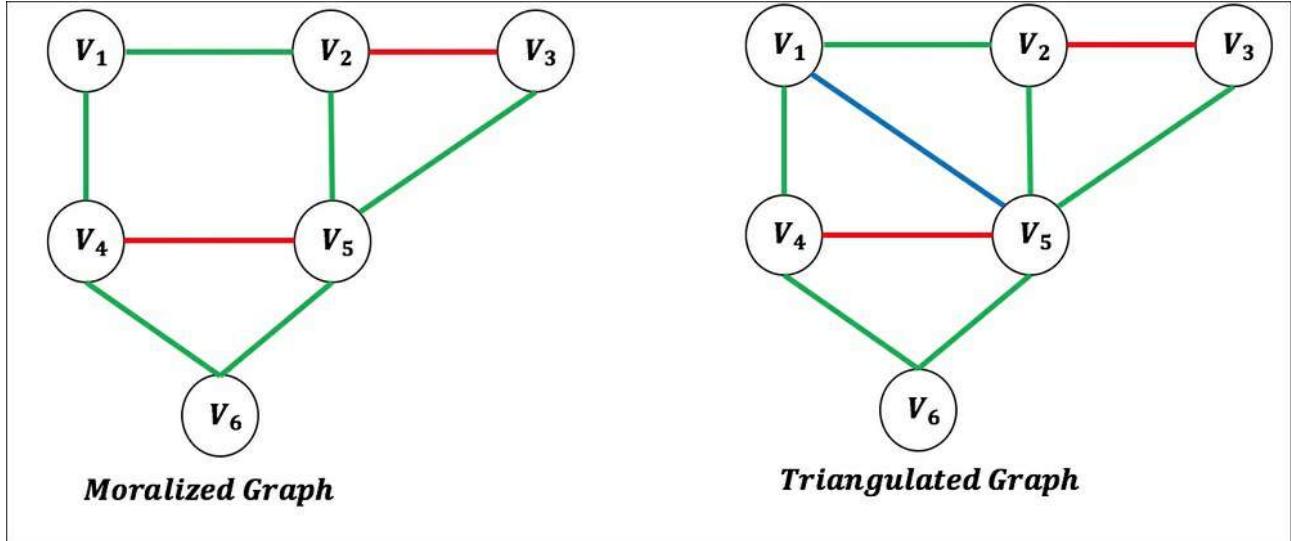


Figure 6. Graph triangulation with blue edge addition to convert a moralized graph to a chordal graph.

3. **Junction Tree:** From the chordal graphs a junction tree is formed using the following steps:

1. Find all the cliques in the graph and make them nodes with the cluster of all vertices. A clique is a subgraph where an edge exists between each pair of nodes. If two nodes have one or more common vertices create an edge consisting of the intersecting vertices as a separator or sepset. For example, the cycle with edges V_1, V_4, V_5 and V_6, V_4, V_5 that have a common edge between V_4, V_5 can be reduced to a clique as shown with the common edge as separator.

If the preceding graph contains a cycle, all separators in the cycle contain the same variable. Remove the cycle in the graph by creating a minimum spanning tree, while including maximum separators. The entire transformation process is shown in Figure 7:

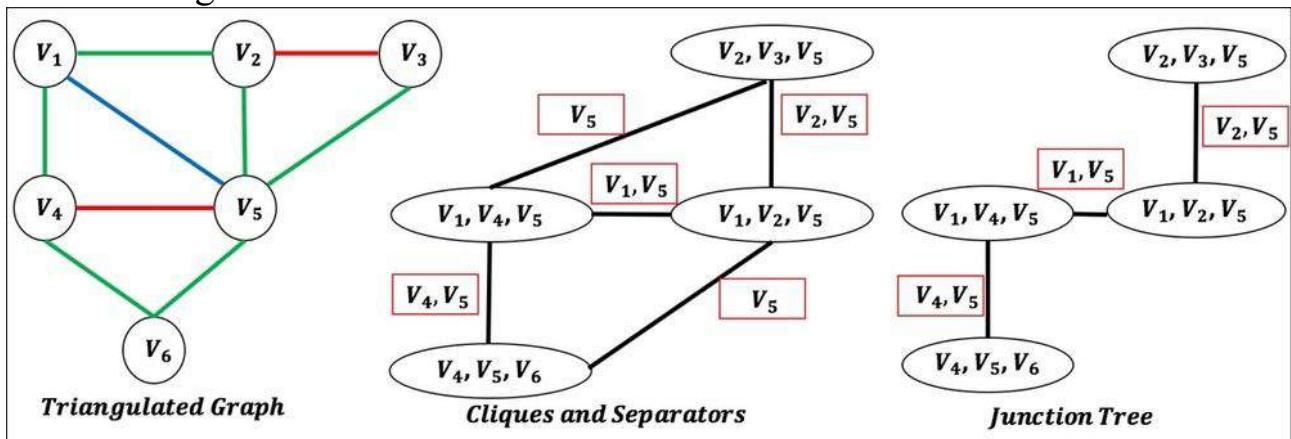


Figure 7. Formation of a Junction Tree

4. **Run the Message Passing algorithm on Junction Tree:** Junction tree can be used to compute the joint distribution using factorization of cliques and separators as

$$P(V) = \frac{\prod_C \varphi_C(V_C)}{\prod_S \varphi_S(V_S)}$$

5. **Compute the parameters of Junction Tree:** The junction tree parameters can be obtained per node using the parent nodes in the original Bayesian network and are called clique potentials, as shown here:

1. $\varphi_1(V_2, V_3, V_5) = P(V_5 | V_2, V_3)P(V_3)$ (Note in the original Bayesian network edge V_5 is dependent on V_2, V_3 , whereas V_3 is independent)
2. $\varphi_2(V_1, V_2, V_5) = P(V_1 | V_2)$
3. $\varphi_3(V_1, V_4, V_5) = P(V_4 | V_1)$
4. $\varphi_3(V_4, V_5, V_6) = P(V_6 | V_4, V_5)$

6. **Message Passing between nodes/cliques in Junction Tree:** A node in the junction tree, represented by clique C_i , multiplies all incoming messages from its neighbors with its own clique potential, resulting in a factor φ_i whose scope is the clique. It then sums out all the variables except the ones on sepset or separators $S_{i,j}$ between C_i and C_j and then sends the resulting factor as a message to C_j .

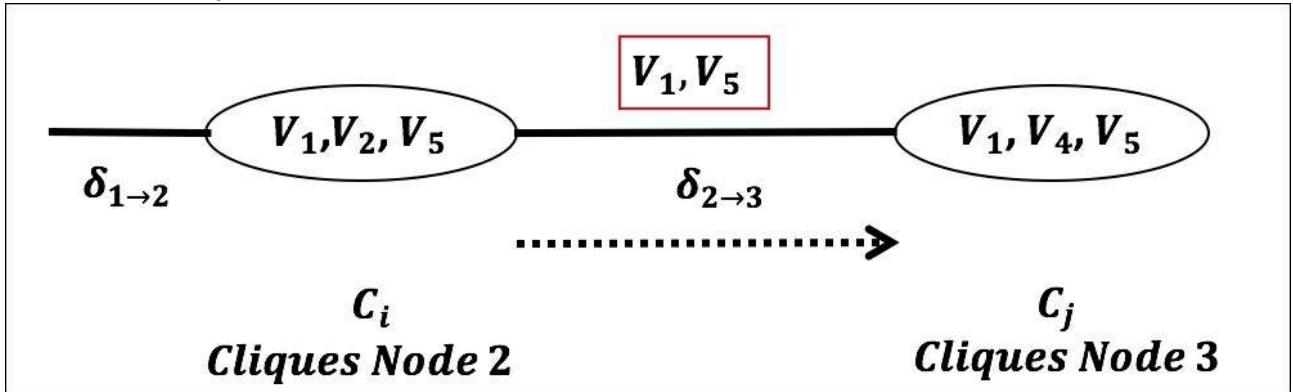


Figure 8. Message passing between nodes/cliques in Junction Tree

$$\delta_{i \rightarrow j} = \sum_{C_i - S_{i,j}} \varphi_i \prod_{k \in (Nb_i - \{j\})} \delta_{k \rightarrow i}$$

Thus, when the message passing reaches the tree root, the joint probability

distribution is completed.

Advantages and limitations

The advantages and limitaions are as follows:

- The algorithm has a theoretical upper bound on the computations that are related to the tree width in the junction tree.
- Multiplication of each potential in the cliques can result in numeric overflow and underflow.

Propagation-based techniques

Here we discuss belief propagation, a commonly used message passing algorithm for doing inference by introducing factor graphs and the messages that can flow in these graphs.

Belief propagation

Belief propagation is one of the most practical inference techniques that has applicability across most probabilistic graph models including directed, undirected, chain-based, and temporal graphs. To understand the belief propagation algorithm, we need to first define factor graphs.

Factor graph

We know from basic probability theory that the entire joint distribution can be represented as a factor over a subset of variables as

$$p(\mathbf{X}) = \prod_S f_s(\mathbf{X}_s)$$

In DAG or Bayesian networks $f_s(\mathbf{X}_s)$ is a conditional distribution. Thus, there is a great advantage in expressing the joint distribution over factors over the subset of variables.

Factor graph is a representation of the network where the variables and the factors involving the variables are both made into explicit nodes (*References [11]*). In a simplified student network from the previous section, the factor graph is shown in *Figure 9*.

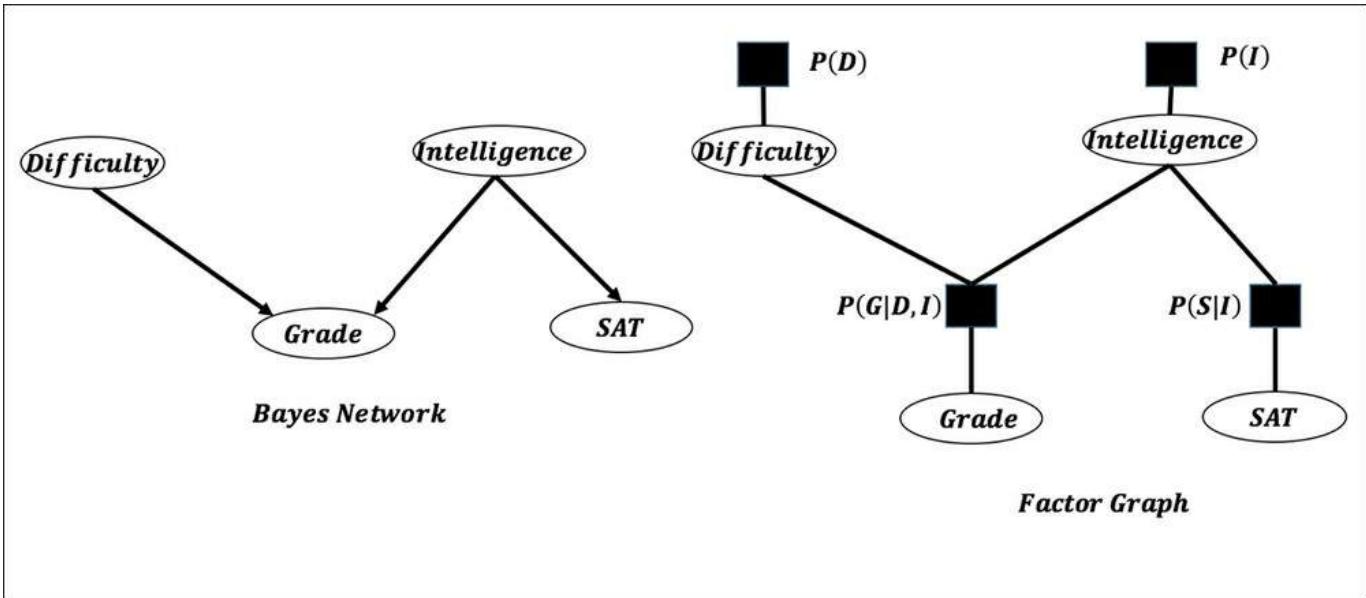


Figure 9. Factor graph of simplified "Student" network

A factor graph is a bipartite graph, that is, it has two types of nodes, variables and factors.

The edges flow between two opposite types, that is, from variables to factors and vice versa.

Converting the Bayesian network to a factor graph is a straightforward procedure as shown previously where you start adding variable nodes and conditional probability distributions as factor nodes. The relationship between the Bayesian network and factor graphs is one-to-many, that is, the same Bayesian network can be represented in many factor graphs and is not unique.

Messaging in factor graph

There are two distinct messages that flow in these factor graphs that form the bulk of all computations through communication.

- **Message from factor nodes to variable nodes:** The message that is sent from a factor node to the variable node can be mathematically represented as follows:

$$P(\mathbf{X}) = \prod_{S \in Nb(x)} \sum_{\mathbf{X}_s} \mathbf{F}_s(x, \mathbf{X}_s)$$

\mathbf{X}_s = set of variables connected in subtree to x via factor f_s

$\mathbf{F}_s(x, \mathbf{X}_s)$ = product of all factors in group associated with f_s

$S \in Nb(x)$ = set of nodes (factors) that are neighbors of x

$P(\mathbf{X}) = \prod_{S \in Nb(x)} \mu_{f_s \rightarrow x}(x)$ where $\mu_{f_s \rightarrow x}(x) = \sum_{\mathbf{X}_s} \mathbf{F}_s(x, \mathbf{X}_s)$ Thus, $\mu_{f_s \rightarrow x}(x)$ is the message from factor node f_s to x and the product of all such messages from neighbors of x to x gives the combined probability to x :

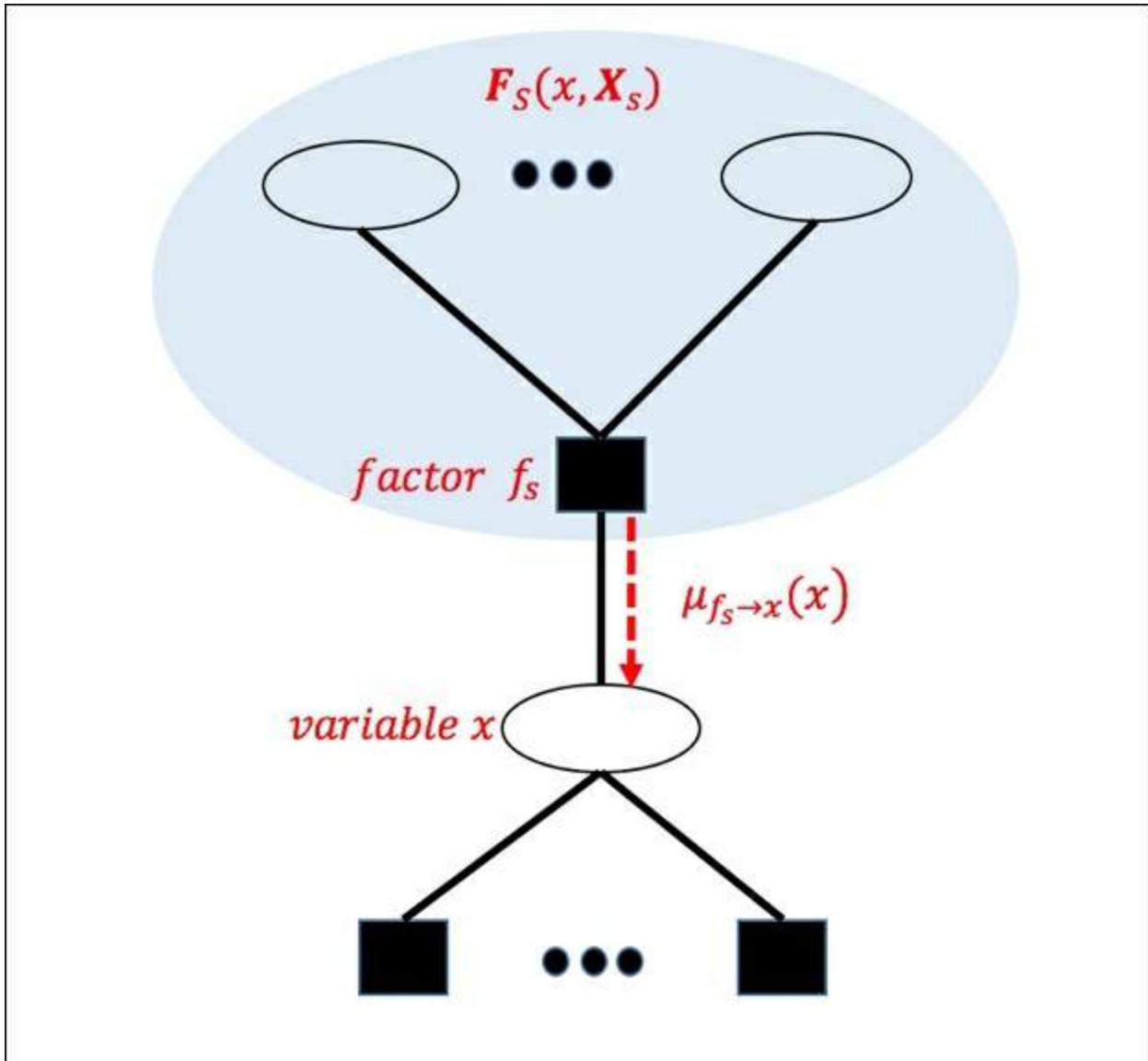


Figure 10. Message-passing from factor node to variable node

- **Message from variable nodes to factor nodes:** Similar to the previous example, messages from variable to factor can be shown to be

$$\mu_{x_m \rightarrow f_s}(x_m) = \prod_{l \in Nb(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)$$

$$Nb(x_m) \setminus f_s = \text{all neighbors of } x_m \text{ except } f_s$$

Thus, all the factors coming to the node x_m are multiplied except for the factor it is sending to.

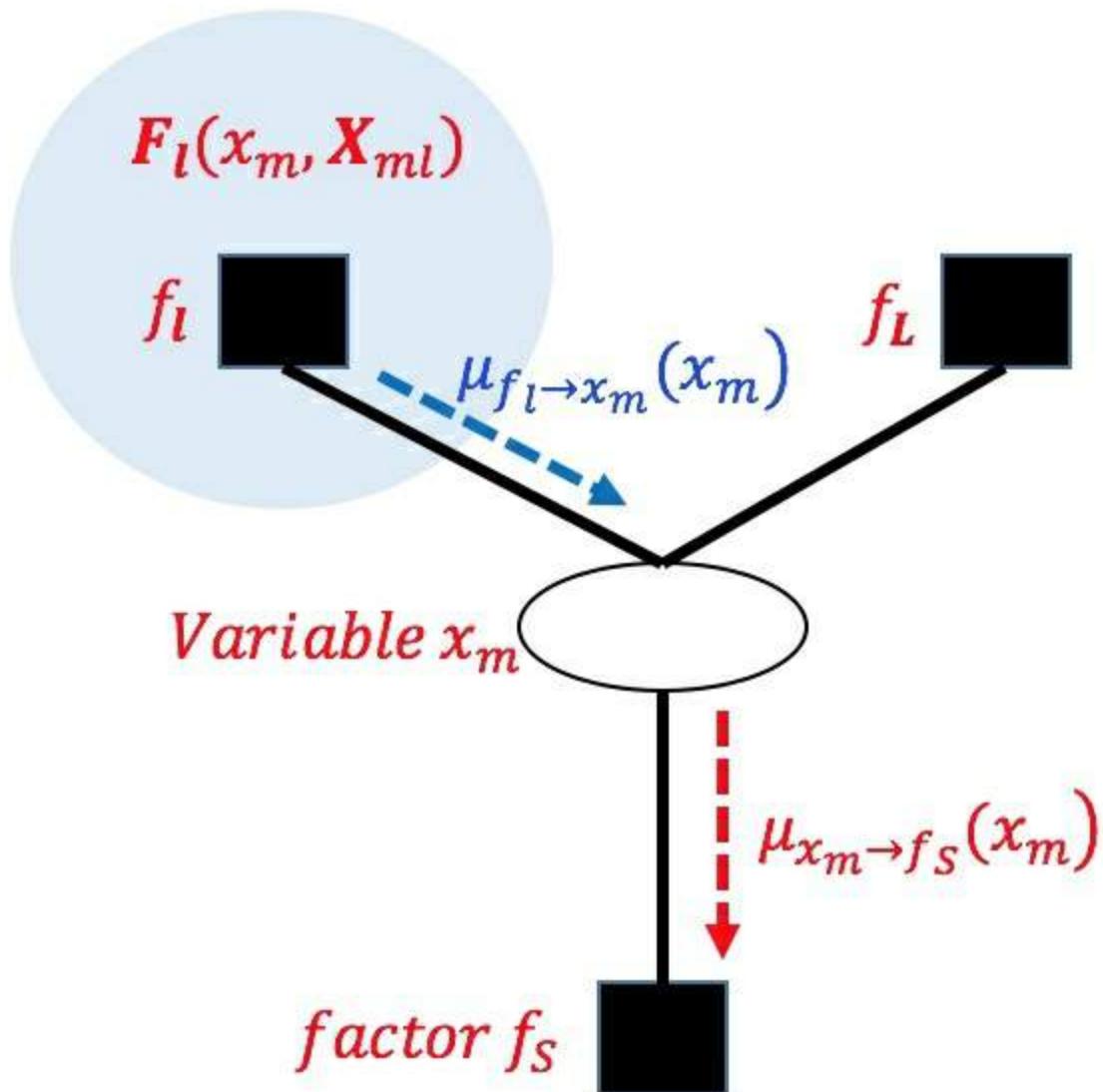


Figure 11. Message-passing from variable node to factor node

Input and output

Inputs:

- List of Condition Probability Distribution/Table (CPD/CPT) F
- List of query variables \mathbf{Q}
- List of observed variables \mathbf{E} and the observed value e

Output:

- $P(\mathbf{Q}|\mathbf{E} = e)$

How does it work?

1. Create a factor graph from the Bayesian network as discussed previously.
2. View the node \mathbf{Q} as the root of the graph.

3. Initialize all the leaf nodes, that is: $\mu_{x \rightarrow f} = 1$ and $\mu_{f \rightarrow x} = f(x)$
4. Apply message passing from a leaf to the next node in a recursive manner.
5. Move to the next node, until root is reached.
6. Marginal at the root node gives the result.

Advantages and limitations

The advantages and limitaions are as follows:

- This algorithm as discussed is very generic and can be used for most graph models. This algorithm gives exact inference in directed trees when there are no cycles.
- This can be easily implemented in parallel and helps in scalability. Based on connectivity, the memory requirement can be very high.

Sampling-based techniques

We will discuss a simple approach using particles and sampling to illustrate the process of generating the distribution $P(X)$ from the random variables. The idea is to repeatedly sample from the Bayesian network and use the samples with counts to approximate the inferences.

Forward sampling with rejection

The key idea is to generate i.i.d. samples iterating over the variables using a topological order. In case of some evidence, for example, $P(X|E = e)$ that contradicts the sample generated, the easiest way is to reject the sample and proceed.

Input and output

Inputs:

- List of Condition Probability Distribution/Table F
- List of query variables \mathbf{Q}
- List of observed variables \mathbf{E} and the observed value \mathbf{e}

Output:

- $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$

How does it work?

1. For $j = 1$ to m //number of samples
 1. Create a topological order of variables, say $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$.
 2. For $i = 1$ to n
 1. $\mathbf{u}_i \sim \mathbf{X}(\text{parent}(\mathbf{X}_i))$ //assign $\text{parent}(\mathbf{X}_i)$ to variables
 2. $\text{sample}(\mathbf{x}_i, P(\mathbf{X}_i | \mathbf{u}_i))$ //sample \mathbf{X}_i given parent assignments
 3. if($\mathbf{x}_i \neq \mathbf{e}_i$, $P(\mathbf{X}_i | \mathbf{E} = \mathbf{e})$) reject and go to 1.1.2. //reject sample if it doesn't agree with the evidence.
 3. Return $(\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n)$ as sample.
2. Compute $P(\mathbf{Q} | \mathbf{E} = \mathbf{e})$ using counts from the samples.

An example of one sample generated for the student network can be by sampling Difficulty and getting Low, next, sampling Intelligence and getting High, next, sampling grade using the CPD table for Difficulty=low and Intelligence=High and getting Grade=A, sampling SAT using CPD for Intelligence=High and getting SAT=good and finally, using Grade=A to sample from Letter and getting Letter=Good. Thus, we get first sample (Difficulty=low, Intelligence=High, Grade=A, SAT=good, Letter=Good)

Advantages and limitations

The advantages and limitations are as follows:

- This technique is fairly simple to implement and execute. It requires a large

number of samples to be approximate within the bounds.

- When evidence set is large, the rejection process becomes costly.

Learning

The idea behind learning is to generate either a structure or find parameters or both, given the data and the domain experts.

The goals of learning are as follows:

- To facilitate inference in Bayesian networks. The pre-requisite of inferencing is that the structure and parameters are known, which are the output of learning.
- To facilitate prediction using Bayesian networks. Given observed variables \mathbf{X} , predict the target variables \mathbf{Y} .
- To facilitate knowledge discovery using Bayesian networks. This means understanding causality, relationships, and other features from the data.

Learning, in general, can be characterized by *Figure 12*. The assumption is that there is a known probability distribution P^* that may or may not have been generated from a Bayesian network G^* . The observed data samples are assumed to be generated or sampled from that known probability distribution P^* . The domain expert may or may not be present to include the knowledge or prior beliefs about the structure. Bayesian networks are one of the few techniques where domain experts' inputs in terms of relationships in variables or prior probabilities can be used directly, in contrast to other machine learning algorithms. At the end of the process of knowledge elicitation and learning from data, we get as an output a Bayesian network with defined structure and parameters (CPTs).

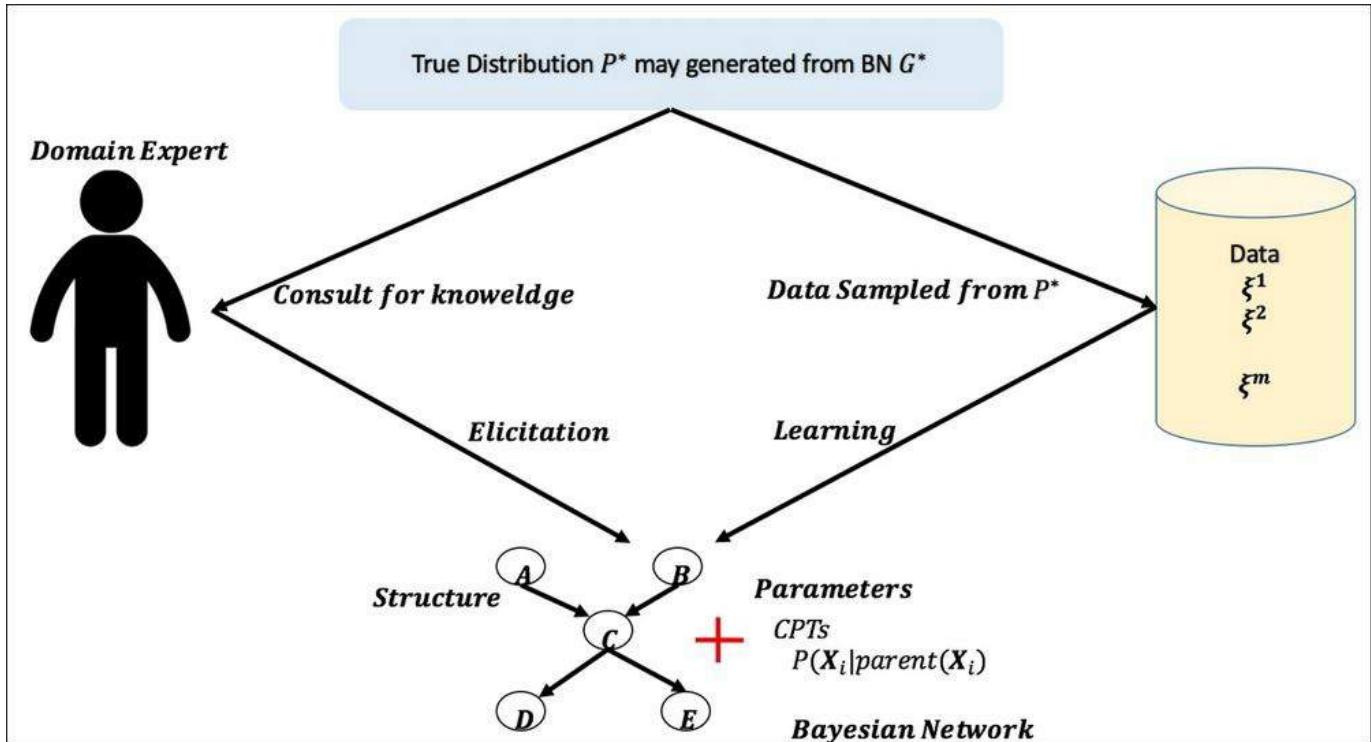


Figure 12. Elements of learning with Bayesian networks

Based on data quality (missing data or complete data) and knowledge of structure from the expert (unknown and known), the following are four classes that Learning in Bayesian networks fall into, as shown in *Table 2*:

		Structure
Data	Known Structure (Learn Parameters)	
	Unknown Structure (Learn Structure and Parameters)	
Complete Data	Parameter Estimation (Maximum Likelihood, Bayesian Estimation)	Optimization (Search and Scoring Techniques)
Incomplete Data	Non-Linear Parametric Optimization (Expectation Maximization, Gradient Descent)	Structure and Parameter Optimization (Structural EM, Mixture Models)

Table 2. Classes of Bayesian network learning

Learning parameters

In this section, we will discuss two broadly used methodologies to estimate parameters given the structure. We will discuss only with the complete data and readers can refer to the discussion in (*References [8]*) for incomplete data parameter estimation.

Maximum likelihood estimation for Bayesian networks

Maximum likelihood estimation (MLE) is a very generic method and it can be defined as: given a data set D , choose parameters $\hat{\theta}$ that satisfy:

- $L(\hat{\theta} : \mathcal{D}) = \max_{\theta \in \Theta} L(\theta : \mathcal{D})$
- $\theta = \text{set of parameters}$
- $\Theta = \text{parameter space of all parameters}$

Maximum likelihood is the technique of choosing parameters of the Bayesian network given the training data. For a detailed discussion, see (*References [6]*).

Given the known Bayesian network structure of graph G and training data $\mathcal{D} = \xi[1], \xi[2], \dots, \xi[M]$, we want to learn the parameters or CPDs—or CPTs to be precise. This can be formulated as:

$$L(\theta : \mathcal{D}) = \prod_m P_G(\xi[m] : \theta)$$

Now each example or instance ξ can be written in terms of variables. If there are i variables represented by x_i and the parents of each is given by $parent_{X_i}$ then:

$$L(\theta : \mathcal{D}) = \prod_m \prod_i P(x_i[m] | parent_{X_i}[m] : \theta)$$

Interchanging the variables and instances:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_i \left[\prod_m P(x_i[m] | parent_{X_i}[m] : \boldsymbol{\theta}) \right]$$

The term is:

$$L_i(\boldsymbol{\theta} : \mathcal{D}) = \prod_m P(x_i[m] | parent_{X_i}[m] : \boldsymbol{\theta})$$

This is the conditional likelihood of a particular variable x_i given its parents $parent_{X_i}$. Thus, parameters for these conditional likelihoods are a subset of parameters given by $\boldsymbol{\theta}_{X_i|parent_{X_i}}$. Thus:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_m L_i(\boldsymbol{\theta}_{X_i|parent_{X_i}} : \mathcal{D})$$

Here, $L_i(\boldsymbol{\theta}_{X_i|parent_{X_i}} : \mathcal{D}) = \prod_m P(x_i[m] | parent_{X_i}[m] : \boldsymbol{\theta}_{X_i|parent_{X_i}})$ is called the local likelihood function. This becomes very important as the total likelihood decomposes into independent terms of local likelihood and is known as the global decomposition property of the likelihood function. The idea is that these local likelihood functions can be further decomposed for a tabular CPD by simply using the count of different outcomes from the training data.

Let N_{ijk} be the number of times we observe variable or node i in the state k , given the

parent node configuration j :

$$N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$$

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}}$$

For example, we can have a simple entry corresponding to $X_i = a$ and $\text{parent}_{X_i} = b$ by estimating the likelihood function from the training data as:

$$L_i(\boldsymbol{\theta}_{X_i=a|\text{parent}_{X_i}=b} : \mathcal{D}) = \frac{\text{Count}(X_i = a, \text{parent}_{X_i} = b)}{\text{Count}(\text{parent}_{X_i} = b)}$$

Consider two cases, as an example. In the first, $X_i = a, \text{parent}_{X_i} = b$ is satisfied by 10 instances with $\text{parent}_{X_i} = b = 100$. In the second, $X_i = a, \text{parent}_{X_i} = b$ is satisfied by 100 when $\text{parent}_{X_i} = b = 1000$. Notice both probabilities come to the same value, whereas the second has 10 times more data and is the "more likely" estimate! Similarly, familiarity with domain or prior knowledge, or lack of it due to uncertainty, is not captured by MLE. Thus, when the number of samples are limited or when the domain experts are aware of the priors, then this method suffers from serious issues.

Bayesian parameter estimation for Bayesian network

This technique overcomes the issue of MLE by encoding prior knowledge about the

parameter θ with a probability distribution. Thus, we can encode our beliefs or prior knowledge about the parameter space as a probability distribution and then the joint distribution of variables and parameters are used in estimation.

Let us consider single variable parameter learning where we have instances $x[1], x[2] \dots x[M]$ and they all have parameter \hat{x} .

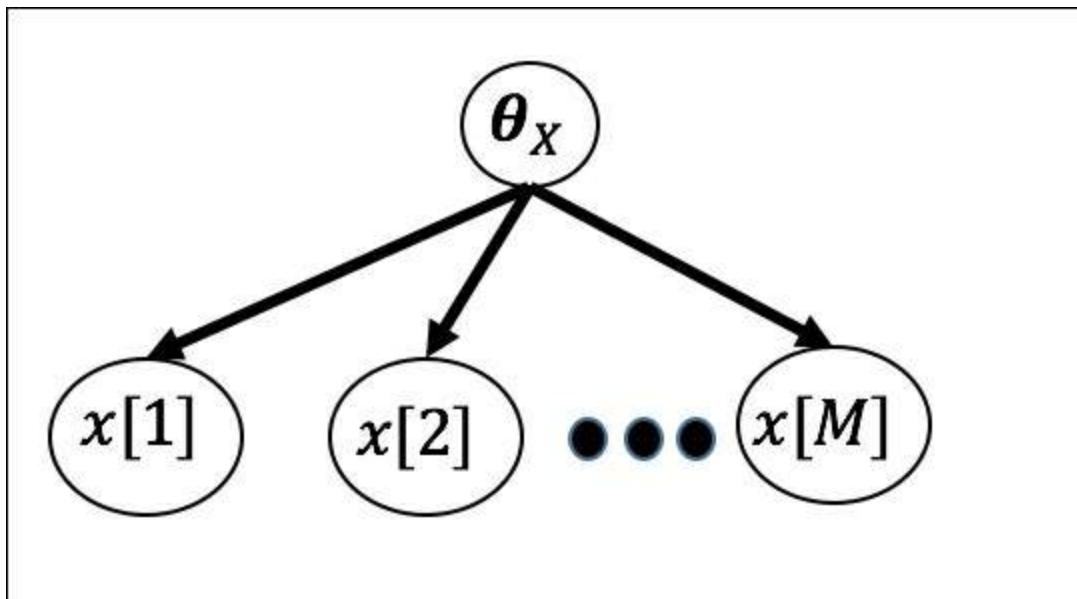


Figure 13. Single variable parameter learning

$$P(x[1], x[2], \dots, x[m], \theta) = P(x[1], x[2], \dots, x[m] | \theta) P(\theta)$$

$$P(x[1], x[2], \dots, x[m], \theta) = P(\theta) \prod_{m=1}^M P(x[m] | \theta)$$

Thus, the network is a joint probability model over parameters and data. The advantage is we can use it for the posterior distribution:

$$P(\theta | x[1], x[2], \dots, x[m]) = \frac{P(x[1], x[2], \dots, x[m] | \theta) P(\theta)}{P(x[1], x[2], \dots, x[m])}$$

$$P(x[1], x[2], \dots, x[m] | \theta) = Likelihood$$

, $P(?) = prior$,

$P(x[1], x[2], \dots, x[m]) = normalizing\ constant$ Thus, the difference between the maximum likelihood and Bayesian estimation is the use of the priors.

Generalizing it to a Bayesian network G given the dataset D :

$$P(\boldsymbol{\theta} | \mathcal{D}) = \frac{P(\mathcal{D} | \boldsymbol{\theta}) P(\boldsymbol{\theta})}{P(\mathcal{D})}$$

$$P(\mathcal{D} | \boldsymbol{\theta}) = \prod_i L_i(\boldsymbol{\theta}_{X_i | parent_{X_i}} : \mathcal{D})$$

If we assume global independence of parameters

$$P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_{X_i | parent_{X_i}})$$

Thus, we get

$$P(\boldsymbol{\theta} | \mathcal{D}) = \frac{1}{\mathcal{D}} \prod_i L_i(\boldsymbol{\theta}_{X_i | \text{parent}_{X_i}} : \mathcal{D}) P(\boldsymbol{\theta}_{X_i | \text{parent}_{X_i}})$$

Again, as before, subset $\mathbf{?}_{X_i} | \text{parent}_{X_i}$ of $\mathbf{?}$ is local and thus the entire posterior can be computed in local terms!

Prior and posterior using the Dirichlet distribution

Often, in practice, a continuous probability distribution known as Dirichlet distribution—which is a Beta distribution—is used to represent priors over the parameters.

$$(\theta_{ij1}, \theta_{ij2}, \dots, \theta_{ijr_i}) \sim \text{Dirichlet}(\alpha_{ij1}, \alpha_{ij2}, \dots, \alpha_{ijr_i})$$

Probability Density Function:

$$f(\theta_{ij1}, \theta_{ij2}, \dots, \theta_{ijr_i}) = \frac{1}{B(\alpha_{ij})} \prod_{k=1}^{r_i} \theta_{ijk}^{\alpha_{ijk}-1}$$

Here, $\theta_{ijk} \geq 0$, $\sum_i \theta_{ijk} = 1$. The alpha terms are known as hyperparameters and $a_{ijri} > 0$. The $\alpha_{ij} = \sum_k \alpha_{ijk}$ is the pseudo count, also known as equivalent sample size and it gives us a measure of the prior.

The Beta function, $B(a_{ij})$ is normally expressed in terms of gamma function as follows

$$B(\alpha_{ij}) = \frac{\prod_{k=1}^{r_i} \Gamma(\alpha_{ijk} r_i)}{\Gamma(\alpha_{ij})}$$

The advantage of using Dirichlet distribution is it is conjugate in nature, that is, irrespective of the likelihood, the posterior is also a Dirichlet if the prior is Dirichlet!

It can be shown that the posterior distribution for the parameters θ_{ijk} is a Dirichlet with updated hyperparameters and has a closed form solution!

$$a_{ijk} = a_{ijk} + N_{ijk}$$

If we use maximum a posteriori estimate and posterior means they can be shown to be:

$$\tilde{\theta}_{ijk} = \frac{\alpha_{ijk} + N_{ijk} - 1}{\alpha_{ijk} + N_{ijk} - r_i}$$

$$\bar{\theta}_{ijk} = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ijk} + N_{ijk}}$$

Learning structures

Learning Bayesian network without any domain knowledge or understanding of

structures includes learning the structure and the parameters. We will first discuss some measures that are used for evaluating the network structures and then discuss a few well-known algorithms for building optimal structures.

Measures to evaluate structures

The measures used to evaluate a Bayes network structure, given the dataset, can be broadly divided into the following categories and details of many are available here (*References* [14]).

- **Deviance-Threshold Measure:** The two common techniques to measure deviance between two variables used in the network and structure are Pearson's chi-squared statistic and the Kullback-Leibler distance.

Given the dataset D of M samples, consider two variables X_i and X_j , the Pearson's chi-squared statistic measuring divergence is

$$d_{\chi^2}(D) = \sum_{X_i, X_j} \frac{\left(M[X_i, X_j] - M \cdot \hat{P}(X_i) \cdot \hat{P}(X_j) \right)^2}{M \cdot \hat{P}(X_i) \cdot \hat{P}(X_j)}$$

$M[X_i, X_j]$ = joint count of X_i, X_j

$\hat{P}(X_i)$ = expected count of X_i

$\hat{P}(X_j)$ = expected count of X_j

$d_{\chi^2}(D)$ is 0; when the variables are independent and larger values indicate there

is dependency between the variables.

Kullback-Leibler divergence is:

$$d_I(\mathcal{D}) = \frac{1}{M} \sum_{X_i, X_j} M[X_i, X_j] \log \frac{M[X_i, X_j]}{M[X_i]M[X_j]}$$

$d_I(D)$ is again 0, it shows independence and the larger values indicates dependency. Using various statistical hypothesis tests, a threshold can be used to determine the significance.

- **Structure Score Measure:** There are various measures to give scores to a structure in a Bayes network. We will discuss the most commonly used measures here. A log-likelihood score discussed in parameter learning can be used as a score for the structure:

$$score_L(G : \theta : D) = l(\hat{\theta}_G : \mathcal{D}) = \sum_{\mathcal{D}} \sum_{i=1}^m \log \hat{P}(X_i | parent(X_i))$$

- **Bayesian information score (BIC)** is also quite a popular scoring technique as it avoids overfitting by taking into consideration the penalty for complex structures, as shown in the following equation

$$score_{BIC}(G : \theta : \mathcal{D}) = l(\hat{\theta}_G : \mathcal{D}) - \frac{\log M}{2} Dim(G)$$

$$Dim(G) = \text{number of independent parameters in } G$$

The penalty function is logarithmic in M , so, as it increases, the penalty is less severe

for complex structures.

The Akaike information score (AIC), similar to BIC, has similar penalty based scoring and is:

$$score_{AIC}(G : \theta : \mathcal{D}) = l(\hat{\theta}_G : \mathcal{D}) - c \cdot \text{Dim}(G)$$

Bayesian scores discussed in parameter learning are also employed as scoring measures.

Methods for learning structures

We will discuss a few algorithms that are used for learning structures in this section; details can be found here (*References* [15]).

Constraint-based techniques

Constraint-based algorithms use independence tests of various variables, trying to find different structural dependencies that we discussed in previous sections such as the d-separation, v-structure, and so on, by following the step-by-step process discussed here.

Inputs and outputs

The input is the dataset D with all the variables $\{X, Y, \dots\}$ known for every instance $\{1, 2, \dots, m\}$, and no missing values. The output is a Bayesian network graph G with all edges, directions known in \mathbf{E} and the CPT table.

How does it work?

1. Create an empty set of undirected edge \mathbf{E} .
2. Test for conditional independence between two variables independent of directions to have an edge.
 1. If for all subset $\mathbf{S} = U - \{X, Y\}$, if X is independent of Y , then add it to the set of undirected edge \mathbf{E}' .
3. Once all potential undirected edges are identified, directionality of the edge is inferred from the set \mathbf{E}' .
 1. Considering a triplet $\{X, Y, Z\}$, if there is an edge $X - Z$ and $Y - Z$, but no edge between $X - Y$ using all variables in the set, and further, if X is not

independent of Y given all the edges $\mathbf{S} = U - \{X, Y, Z\}$, this implies the direction of $X \rightarrow Y$ and $Y \rightarrow Z$.

2. Add the edges $X \rightarrow Y$ and $Y \rightarrow Z$ to set \mathbf{E} .
3. Update the CPT table using local calculations.
4. Return the Bayes network G , edges \mathbf{E} , and the CPT tables.

Advantages and limitations

- Lack of robustness is one of the biggest drawbacks of this method. A small error in data can cause a big impact on the structure due to the assumptions of independence that will creep into the individual independence tests.
- Scalability and computation time is a major concern as every subset of variables are tested and is approximately 2^n . As the number of variables increase to the 100s, this method fails due to computation time.

Search and score-based techniques

The search and score method can be seen as a heuristic optimization method where iteratively, structure is changed through small perturbations, and measures such as BIC or MLE are used to give score to the structures to find the optimal score and structure. Hill climbing, depth-first search, genetic algorithms, and so on, have all been used to search and score.

Inputs and outputs

Input is dataset D with all the variables $\{X, Y, \dots\}$ known for every instance $\{1, 2, \dots, m\}$ and no missing values. The output is a Bayesian network graph G with all edges and directions known in \mathbf{E} .

How does it work?

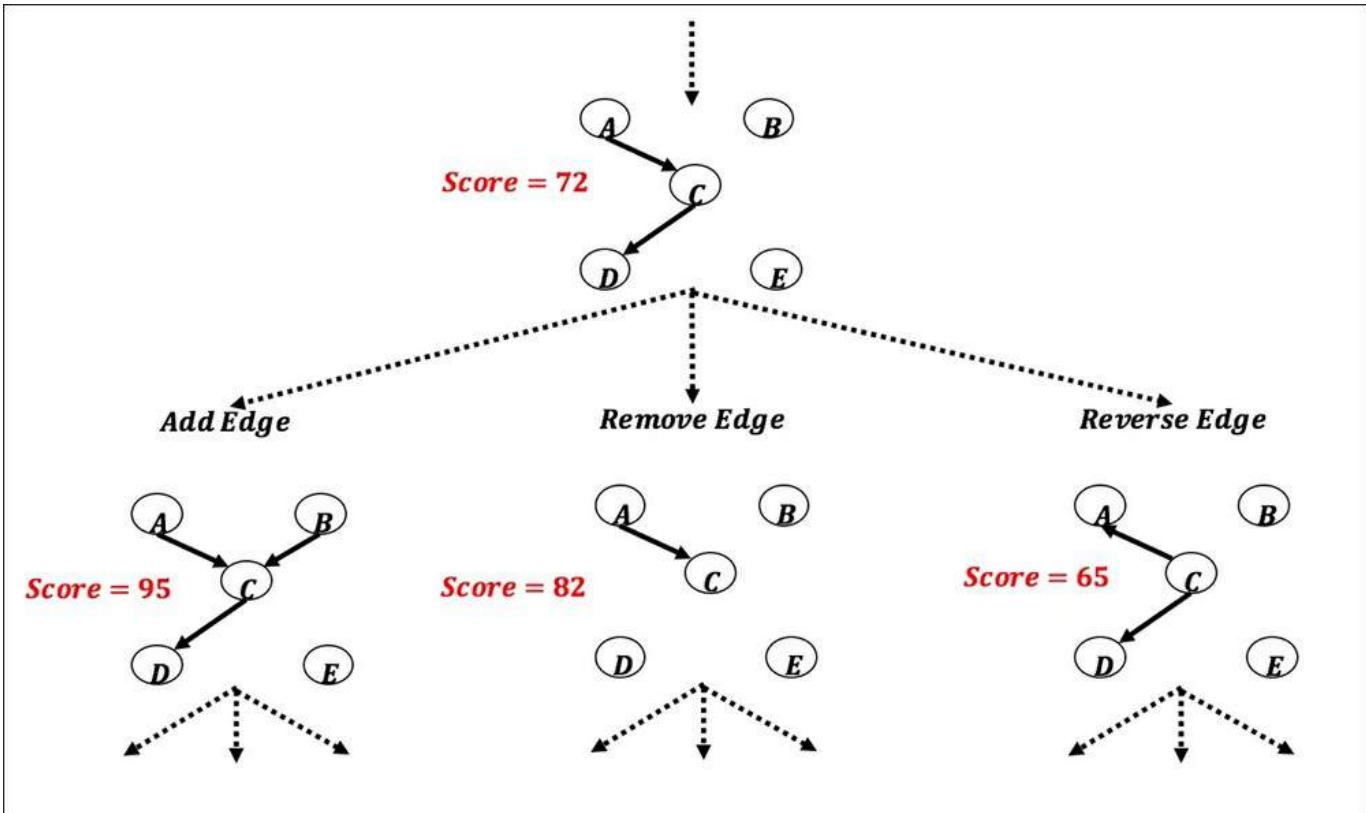


Figure 14. Search and Score

1. Initialize the Graph G , either based on domain knowledge or empty or full. Initialize the Edge set E based on the graph and initialize the CPT tables T based on the graph G , E , and the data D . Normally terminating conditions are also mentioned such as *maxIterations*:
2. $maxScore = -8$, $score = computeScore(G, E, T)$
3. Do
 1. $maxScore = score$
 2. For each variable pair (X, Y)
 1. For each

$$E' \in \{E \cup \{X \rightarrow Y\}, E - \{X \rightarrow Y\}, E - \{X \rightarrow Y\} \cup \{Y \rightarrow X\}\}$$
 2. New Graph G' based on parents and variables with edge changes.
 3. Compute new CPTs T' ? $computeCPT(G', E', D)$.
 4. $currentScore = computeScore(G', E', T')$
 5. If $currentScore > score$:
 1. $score = currentScore$
 2. $G' = G$, $E' = E$

4. Repeat 3 while ($score > maxScore$) or $numIterations < maxIterations$

Advantages and limitations

- Getting stuck in a local optimum, which is the drawback of most of these heuristic search methods, is one of the biggest disadvantages.
- Convergence or theoretical guarantees are not available in heuristic search, so searching for termination is very much by guess work.

Markov networks and conditional random fields

So far, we have covered directed acyclic graphs in the area of probabilistic graph models, including every aspect of representation, inference, and learning. When the graphs are undirected, they are known as **Markov networks (MN)** or **Markov random field (MRF)**. We will discuss some aspects of Markov networks in this section covering areas of representation, inference, and learning, as before. Markov networks or MRF are very popular in various areas of computer vision such as segmentation, de-noising, stereo, recognition, and so on. For further reading, see (*References [10]*).

Representation

Even though a Markov network, like Bayesian networks, has undirected edges, it still has local interactions and distributions. We will first discuss the concept of parameterization, which is a way to capture these interactions, and then the independencies in MN.

Parameterization

The affinities between the variables in MN are captured through three alternative parameterization techniques discussed in the following sections.

Gibbs parameterization

The probability distribution function is said to be in Gibb's distribution or parameterized by Gibb's distribution if

$$P_\phi(X_1, X_2, \dots, X_k) = \frac{1}{Z} (\phi_1(\mathbf{D}_1), \phi_2(\mathbf{D}_2), \dots, \phi_m(\mathbf{D}_m))$$

Z is called the partitioning function defined as:

$$Z = \sum_{X_1, X_2, \dots, X_k} \phi_1(\mathbf{D}_1), \phi_2(\mathbf{D}_2), \dots, \phi_m(\mathbf{D}_m)$$

Note that interaction between variables are captured by factors

$\phi_1(\mathbf{D}_1), \phi_2(\mathbf{D}_2), \dots, \phi_m(\mathbf{D}_m)$ and are not the marginal probabilities, but contribute to the joint probability. The factors that parameterize a Markov network are called clique potentials. By choosing factors over maximal cliques in the graph, the number of parameters are reduced substantially.

Factor graphs

Graph structure of Markov network does not reveal properties such as whether the factors involve maximal cliques or their subsets when using Gibbs parameterization. Factor graphs discussed in the section of inferencing in Bayesian networks have a step to recognize maximal cliques and thus can capture these parameterizations. Please refer to the section on factor graphs in BN.

Log-linear models

Another form of parameterization is to use the energy model representation from statistical physics.

The potential is represented as a set of features and a potential table is generally represented by features with weights associated with them.

If D is a set of variables, $\phi(D)$ is a factor then:

$$\epsilon(D) = -\ln \phi(D)$$

Thus, as the energy increases, the probability decreases and vice versa. The logarithmic cell frequencies captured in $\phi(D)$ are known as log-linear in statistical physics. The joint probability can be represented as:

$$P(X_1, X_2, \dots, X_k : \theta) = \frac{1}{Z(\theta)} \exp \left[\sum_{i=1}^m \theta_i f_i(D_i) \right]$$

$\mathcal{F} = \{(f_i(D_i))\}^m$ is the feature function defined over the variables in D_i .

Independencies

Like Bayesian networks, Markov networks also encode a set of independence assumptions governing the flow of influence in undirected graphs.

Global

A set of nodes \mathbf{Z} separates sets of nodes \mathbf{X} and \mathbf{Y} , if there is no active path between any node in $X \setminus \mathbf{Z}$ and $Y \setminus \mathbf{Z}$ given \mathbf{Z} . Independence in graph G is:

$$\mathcal{J}(G) = \left\{ (X \perp Y | \mathbf{Z}) : sep_G((X; Y | \mathbf{Z}) \right.$$

Pairwise Markov

Two nodes, X and Y , are independent given all other nodes if there is no direct edge between them. This property is of local independence and is weakest of all:

$$\mathcal{J}_p(G) = \left\{ (X \perp Y | x - \{X, Y\}) : X - Y \notin x \right\}$$

Markov blanket

A node is independent of all other nodes in the graph, given its Markov blanket, which is an important concept in Markov networks:

$$\mathcal{J}_l(G) = \left\{ X \perp x - \{X\} - \mathbf{U} | \mathbf{U} : X \notin x \right\}$$

Here $\mathbf{U} = \text{markov blanket of } X$.

Figure 15 shows a Markov blanket for variable X as its parents, children, and children's parents:

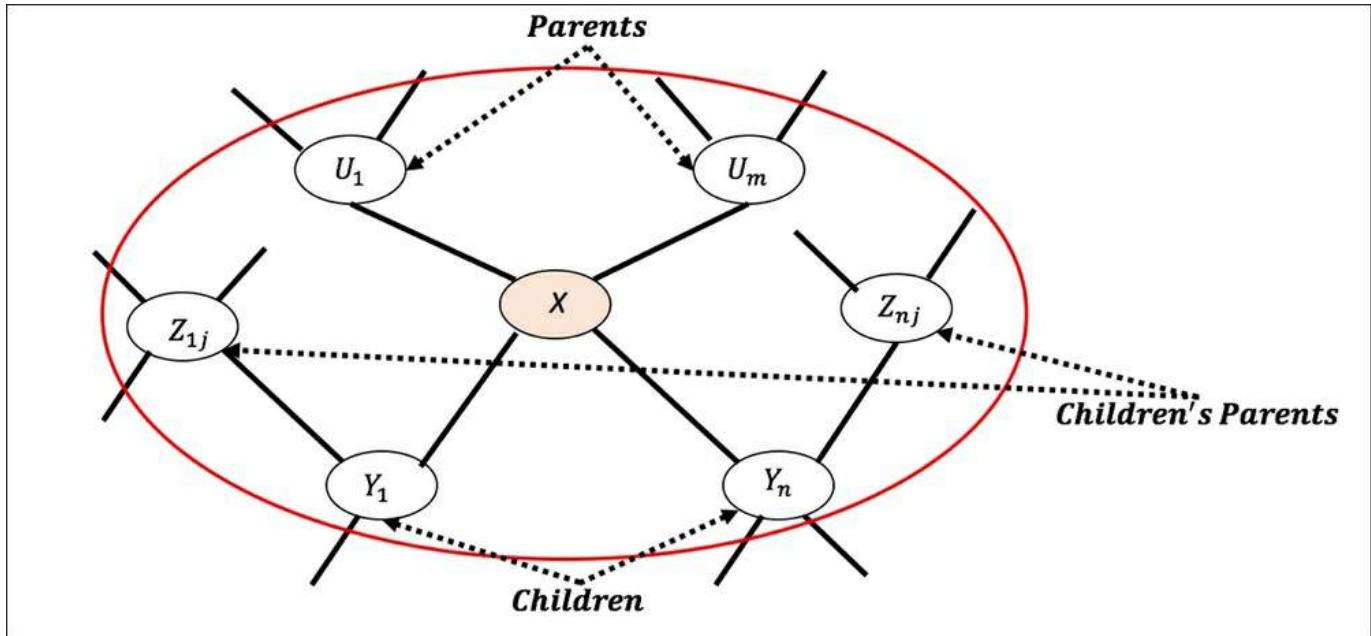


Figure 15. Markov blanket for Node X - its Parents, Children, and Children's Parents.

Inference

Inference in MNs is similarly #P-complete problem and hence similar approximations or heuristics get applied. Most exact and approximate inferencing techniques, such as variable elimination method, junction tree method, belief propagation method, and so on, which were discussed in Bayes network, are directly applicable to Markov networks. The marginals and conditionals remain similar and computed over the potential functions over the cliques as

$$P(X) = \frac{1}{Z} \prod_c \phi_c(X)_c$$

$$Z = \sum_X \prod_C \phi_c(X)_c$$

Markov blankets simplify some of the computations.

Learning

Learning the parameters in Markov networks is complex and computationally expensive due to the entanglement of all the parameters in the partitioning function. The advantageous step of decomposing the computations into local distributions cannot be done because of the partitioning function needing the factor coupling of all the variables in the network.

Maximum likelihood estimation in MN does not have a closed-form solution and hence incremental techniques such as gradient descent are used for optimizing over the entire parameter space. The optimization function can be shown to be a concave function, thus ensuring a global optimum, but each step of the iterations in gradient descent requires inferencing over the entire network, making it computationally expensive and sometimes intractable.

Bayesian parameter estimation requires integration over the entire space of parameters, which again has no closed-form solution and is even harder. Thus, most often, approximate learning methods such as **Markov Chain Monte Carlo (MCMC)** are used for MNs.

Structure learning in the MNs is similar or even harder than parameter learning and has been shown to be NP-hard. In the constraint-based approach, for a given dataset, conditional independence between the variables is tested. In MNs, each pair of variables is tested for conditional independence using mutual information between the pair. Then, based on a threshold, an edge is either considered to be existing between the pair or not. One disadvantage of this is it requires extremely large numbers of samples to refute any noise present in the data. Complexity of the network due to occurrence of pairwise edges is another limitation.

In search and score-based learning, the goal is similar to BNs, where search is done for structures and scoring—based on various techniques—is computed to help and adjust the search. In the case of MNs, we use features described in the log-linear models rather than the potentials. The weighting of the features is considered during optimization and scoring.

Conditional random fields

Conditional random fields (CRFs) are a specialized form of Markov network where the hidden and observables are mostly modeled for labeled sequence prediction problems (*References [16]*). Sequence prediction problems manifest in many text mining areas such as next word/letter predictions, **Part of speech (POS)** tagging, and so on, and in bioinformatics domain for DNA or protein sequence predictions.

The idea behind CRFs is the conditional distribution of sequence is modeled as feature functions and the labeled data is used to learn using optimization the empirical distribution, as shown in the following figure.

The conditional distribution is expressed as follows where $Z(\mathbf{x})$ is the normalizing constant. Maximum likelihood is used for parameter estimation for λ and is generally a convex function in log-linear obtained through iterative optimization methods such as gradient descent.

$$P(\mathbf{y} | \mathbf{x}, \lambda) = \frac{1}{Z(\mathbf{x})} \exp \sum_{i=1}^n \sum_j \lambda_j f(y_{i-1}, y_i, \mathbf{x}, i)$$

Feature Functions
 $f(y_{i-1}, y_i, \mathbf{x}, i)$

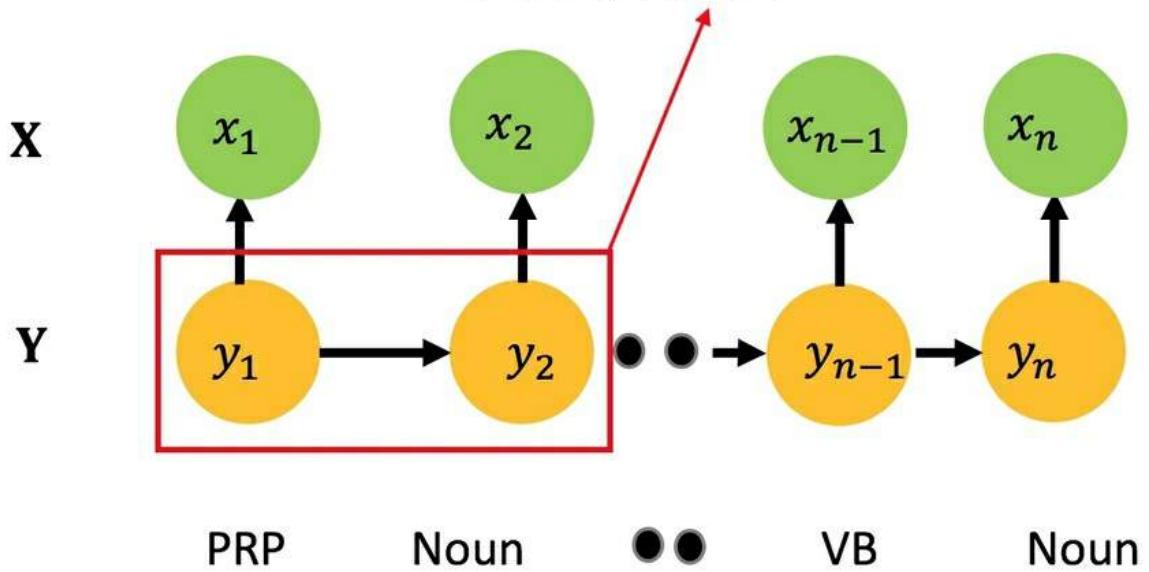


Figure 16: Conditional random fields mapped to the area of sequence prediction in the POS tagging domain.

Specialized networks

In this section, we will cover some basic specialized probabilistic graph models that are very useful in different machine learning applications.

Tree augmented network

In [Chapter 2, Practical Approach to Real-World Supervised Learning](#), we discussed the Naïve Bayes network, which makes the simplified assumption that all variables are independent of each other and only have dependency on the target or the class variable. This is the simplest Bayesian network derived or assumed from the dataset. As we saw in the previous sections, learning complex structures and parameters in Bayesian networks can be difficult or sometimes intractable. The **tree augmented network** or TAN ([References \[9\]](#)) can be considered somewhere in the middle, introducing constraints on how the trees are connected. TAN puts a constraint on features or variable relationships. A feature can have only one other feature as parent in addition to the target variable, as illustrated in the following figure:

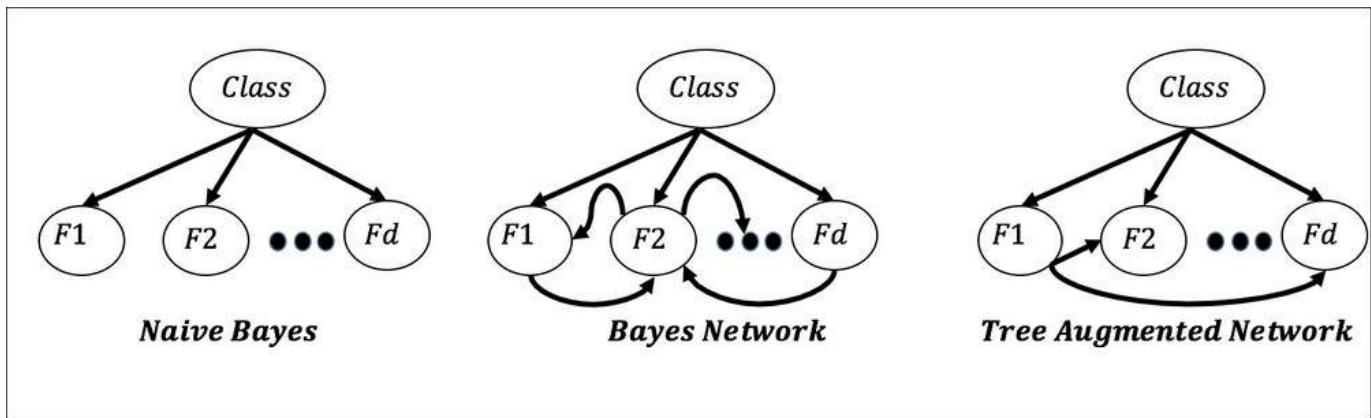


Figure 17. Tree augmented network showing comparison with Naïve Bayes and Bayes network and the constraint of one parent per node.

Input and output

Inputs are the training dataset D with all the features as variables $\{X, Y..\}$. The features have discrete outcomes, if they don't need to be discretized as a pre-processing step.

Outputs are TAN as Bayesian network with CPTs.

How does it work?

1. Compute mutual information between every pair of variables from the training dataset.

2. Build an undirected graph with each node being the variable and edge being the mutual information between them.
3. Create a maximum weighted spanning tree.
4. Transform the spanning tree to a directed graph by selecting the outcome or the target variable as the root and having all the edges flowing in the outwards direction.
5. If there is no directed edge between the class variable and other features, add it.
6. Compute the CPTs based on the DAG or TAN constructed previously.

Advantages and limitations

- It is more accurate than Naïve Bayes in many practical models. It is less complex and faster to build and compute than complete Bayes networks.

Markov chains

Markov Chains are specialized probabilistic graph models, with directed graphs containing loops. Markov chains can be seen as extensions of automata where the weights are probabilities of transition. Markov chains are useful to model temporal or sequence of changes that are directly observable. See (*References [12]*) for further study.

Figure 17 represents a Markov chain (first order) and the general definition can be given as a stochastic process consisting of

Nodes as states, $Q = \{q_1, q_2, \dots, q_L\}$ and $|Q| = N$

Edges representing transition probabilities between the states or nodes. It is generally represented as a matrix $A = a_{kl}$, which is a $N \times N$ matrix where N is the number of nodes or states. The value of a_{kl} captures the transition probability to node q_l given the state q_k . The rows of matrix add to 1 and the values of $0 \leq a_{kl} \leq 1$.

Initial probabilities of being in the state, $p = \{p_1, p_2, \dots, p_N\}$.

Thus, it can be written as a triple $M = (Q, A, p)$ and the probability of being in any state only depends on the last state (first order):

$$P(q_i = k_i | q_1 = k_1, q_2 = k_2, \dots, q_{i-1} = k_{i-1}) = P(q_i = k_i | q_{i-1} = k_{i-1})$$

The joint probability:

$$P(q_1, q_2, \dots, q_L) = \pi_{q_1} \prod_{i=2}^L a_{q_i} a_{q_{i-1}}$$

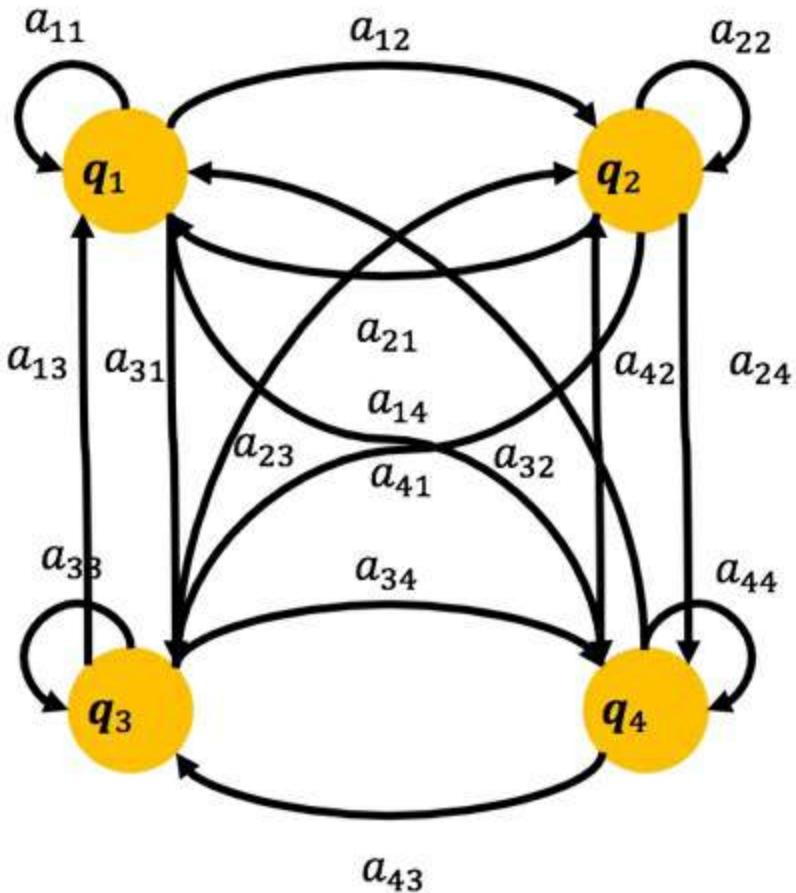


Figure 18. First-order Markov chain

Hidden Markov models

In many real-world situations, the events we are interested in are not directly observable. For example, the words in sentences are observable, but the part-of-speech that generated the sentence is not. **Hidden Markov models (HMM)** help us in modeling such states where there are observable events and hidden states (*References [13]*). HMM are widely used in various modeling applications for speech recognition, language modeling, time series analysis, and bioinformatics applications such as DNA/protein sequence predictions, to name a few.

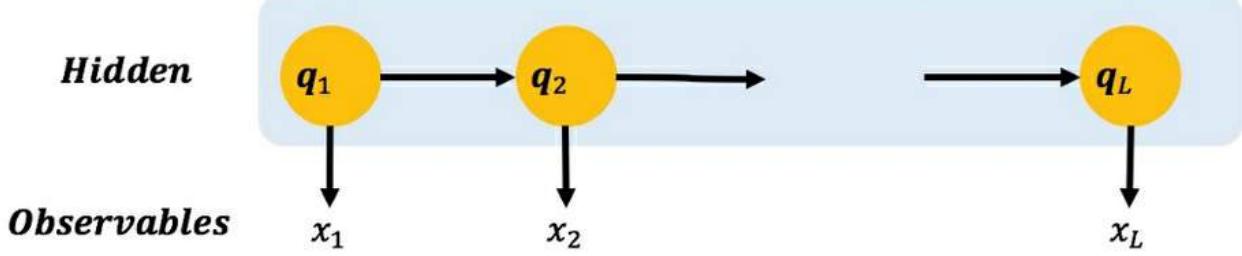


Figure 19. Hidden Markov model showing hidden variables and the observables.

Hidden Markov models can be defined again as a triple $\mathcal{H} = (\Sigma, Q, \Theta)$, where:

- is a set of finite states or symbols that are observed. $\Sigma = \{x_1, x_2, \dots, x_L\}$
- Q is a set of finite states that are not observed $|Q| = N$.
- Θ are the parameters.

The state transition matrix, given as $\mathbf{A} = (a_{kl})_{k, l \in Q}$ captures the probability of transition from state q_k to q_l .

Emission probabilities capturing relationships between the hidden and observed state, given as $e_k(b), k \in Q$ and $b \in \Sigma$. $e_k(b) = P(x_i = b | q_i = k)$.

Initial state distribution $p = \{p_1, p_2, \dots, p_N\}$.

Thus, a path in HMM consisting of a sequence of hidden states $Q = \{q_1, q_2, \dots, q_L\}$ is a first order Markov chain $M = (Q, \mathbf{A}, p)$. This path in HMM emits a sequence of symbols, x_1, x_2, x_L , referred to as the observations. Thus, knowing both the observations and hidden states the joint probability is:

$$P(x, q) = \pi_{q_1} e_{q_1}(x_1) \prod_{i=2}^L a_{q_i} a_{q_{i-1}} e_{q_i}(x_i)$$

In real-world situations, we only know the observations x and do not know the hidden states q . HMM helps us to answer the following questions:

- What is the most likely path that could have generated the observation x ?
- What is the probability of x ?
- What is the probability of being in state $q_i = k$ given the observation ?

Most probable path in HMM

Let us assume the observations $x = x_1, x_2, x_L$ and we want to find the path q^* that generated the observations. This can be given as:

$$q^* = \operatorname{argmax} P(x, q)$$

The path q^* need not be unique, but for computation and explanation the assumption of the unique path is often made. In a naïve way, we can compute all possible paths of length L of q and chose the one(s) with the highest probability giving exponential computing terms or speed. More efficient is using Viterbi's algorithm using the concept of dynamic programming and recursion. It works on the simple principle of breaking the equation into simpler terms as:

$$V_k(i) = \text{highest probability of path } q_1, q_2, \dots, q_i \text{ emitting observables } x_1, x_2, \dots, x_i$$

$$P(x, q^*) = \max \{V_k(L) | k \in Q\}$$

$$V_k(i) = e_k(x_i) \max V_l(i-1) a_{lk}$$

Here, $e_k(x_i)$ = probability of stake k emitting x_i and

$V_l(i-1) a_{lk}$ = probability of best path that ends in l and transitions to k . Given the

initial condition $e_k(x_1)\pi_1$ and using dynamic programming with keeping pointer to the path, we can efficiently compute the answer.

Posterior decoding in HMM

The probability of being in a state $q_i = k$ given the observation x can be written using Bayes theorem as:

$$P(q_i = k | x) = \frac{P(q_i = k, x)}{P(x)}$$

The numerator can be rewritten as:

$$P(q_i = k | x) = P(q_i = k, x_1, x_2, \dots, x_i, x_{i+1}, x_{i+2}, \dots, x_L)$$

$$P(q_i = k, x) = P(x_1, x_2, \dots, x_i, q_i = k) P(x_{i+1}, x_{i+2}, \dots, x_L | x_1, x_2, \dots, x_i, q_i = k)$$

$$P(q_i = k, x) = F_k(i) B_k(i)$$

$$P(q_i = k | x) = \frac{F_k(i)B_k(i)}{P(x)}$$

Where $F_k(i) = P(x_1, x_2, \dots, x_i, q_i = k)$ is called a Forward variable and $B_k(i) = P(x_{i+1}, x_{i+2}, \dots, x_L | x_1, x_2, \dots, x_i, q_i = k)$ is called a Backward variable.

The computation of the forward variable is similar to Viterbi's algorithm using dynamic programming and recursion where summation is done instead:

$$F_k(i) = P(x_1, x_2, \dots, x_i, q_i = k)$$

$$F_k(i) = e_k(x_i) \sum_{l \in Q} P(x_1, x_2, \dots, x_{i-1}, q_{i-1} = l) a_{lk}$$

$$F_k(i) = e_k(x_i) \sum_{l \in Q} F_l(i-1) a_{lk}$$

The probability of observing x can be, then

$$P(x) = \sum_{k \in Q} F_k(L)$$

The forward variable is the joint probability and the backward variable is a conditional probability:

$$B_k(i) = P(x_{i+1}, x_{i+2}, \dots, x_L | x_1, x_2, \dots, x_i, q_i = k)$$

$$B_k(i) = \sum_l a_{kl} e_l(x_{i+1}) P(x_{i+2}, \dots, x_L | q_{i+1} = l)$$

$$B_k(i) = \sum_l a_{kl} e_l(x_{i+1}) B_l(i+1)$$

It is called a backward variable as the dynamic programming table is filled starting with the L^{th} column to the first in a backward manner. The backward probabilities can also be used to compute the probability of observing x as:

$$P(x) = \sum_{k \in Q} B_k(1) e_k(x_1) \pi_k$$

Tools and usage

In this section, we will introduce two tools in Java that are very popular for probabilistic graph modeling.

OpenMarkov

OpenMarkov is a Java-based tool for PGMs and here is the description from www.openmarkov.org:

Note

OpenMarkov is a software tool for probabilistic graphical models (PGMs) developed by the Research Centre for Intelligent Decision-Support Systems of the UNED in Madrid, Spain.

It has been designed for: editing and evaluating several types of PGMs, such as Bayesian networks, influence diagrams, factored Markov models, and so on, learning Bayesian networks from data interactively, and cost-effectiveness analysis.

OpenMarkov is very good in performing interactive and automated learning from the data. It has capabilities to preprocess the data (discretization using frequency and value) and perform structure and parameter learning using a few search algorithms such as search-based Hill Climbing and score-based PC. OpenMarkov stores the models in a format known as pgmx. To apply the models in most traditional packages there may be a need to convert the pgmx models to XMLBIF format. Various open source tools provide these conversions.

Here we have some screenshots illustrating the usage of OpenMarkov to learn the structure and parameters from the data.

In *Figure 20*, we see the screen for interactive learning where you select the data file and algorithm to use:

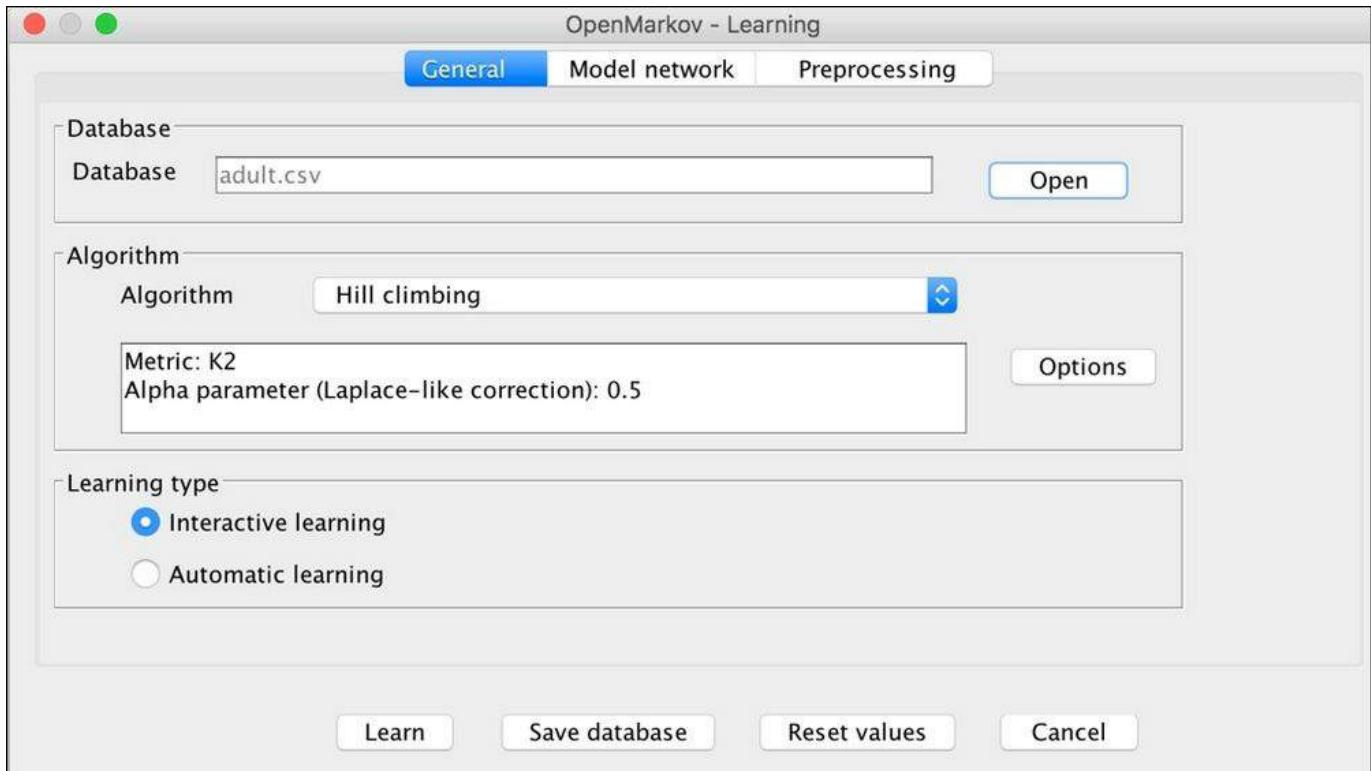


Figure 20. OpenMarkov GUI – Interactive learning, algorithm selection

The next step is the **Preprocessing** tab (Figure 21) where we can select how discretization is done:

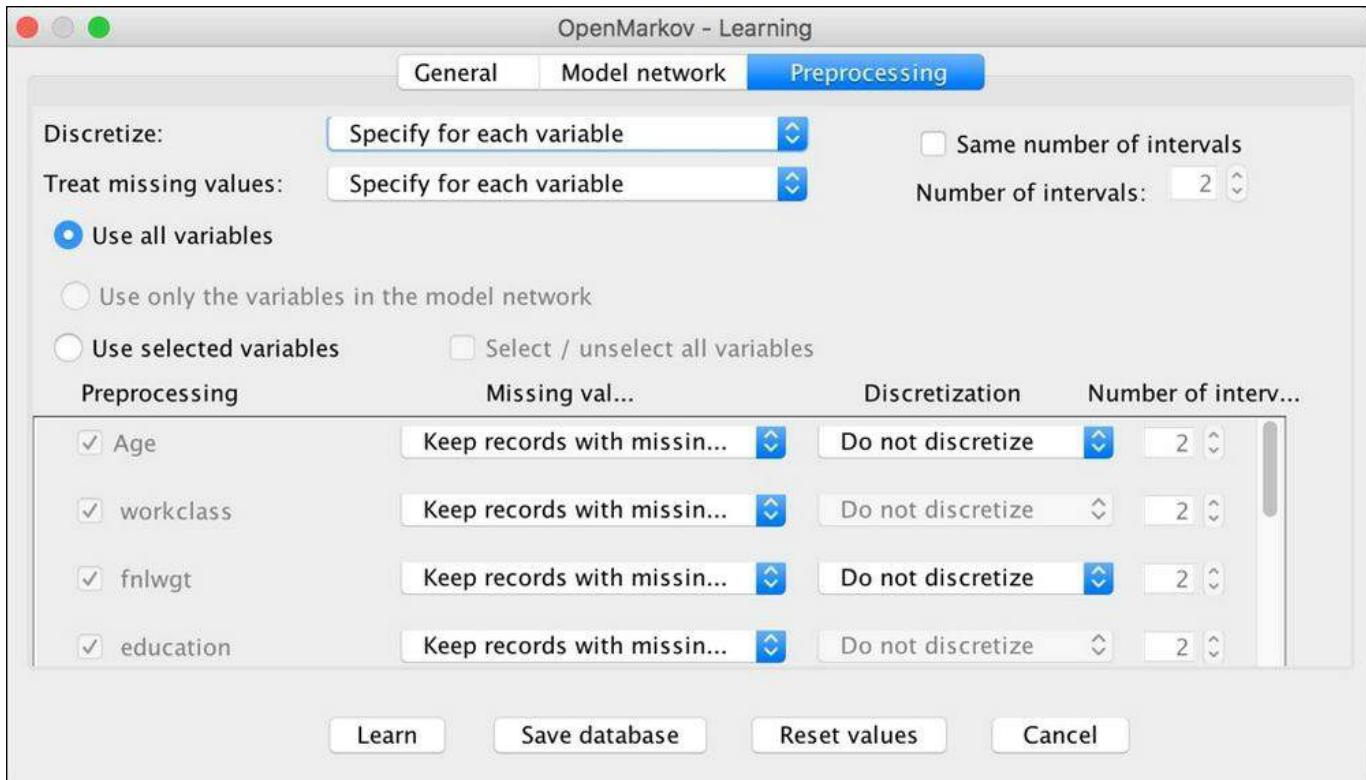


Figure 21. OpenMarkov GUI – Preprocessing screen

Finally, in *Figure 22*, we see the display of the learned Bayes network structure:

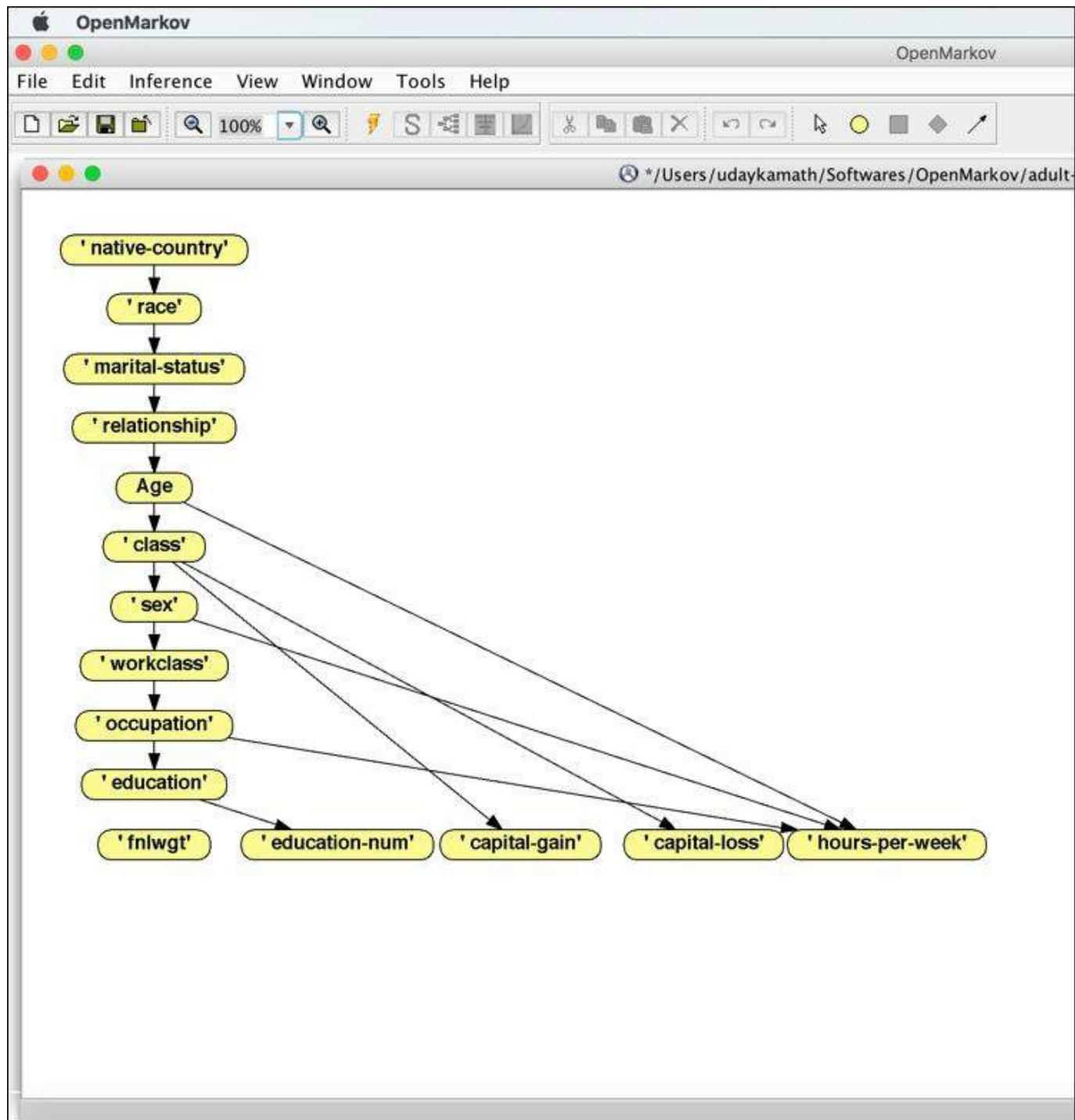


Figure 22. OpenMarkov GUI – Structure output

Weka Bayesian Network GUI

Weka's Bayes Network editor for interactive and automated learning has a large number of options for Bayes network representation, inference and learning as compared to OpenMarkov. The advantage in using Weka is the availability of a number of well-integrated preprocessing and transformation filters, algorithms, evaluation, and experimental metrics.

In *Figure 23*, we see the Bayes Network Editor where the search algorithm is selected and various options can be configured:

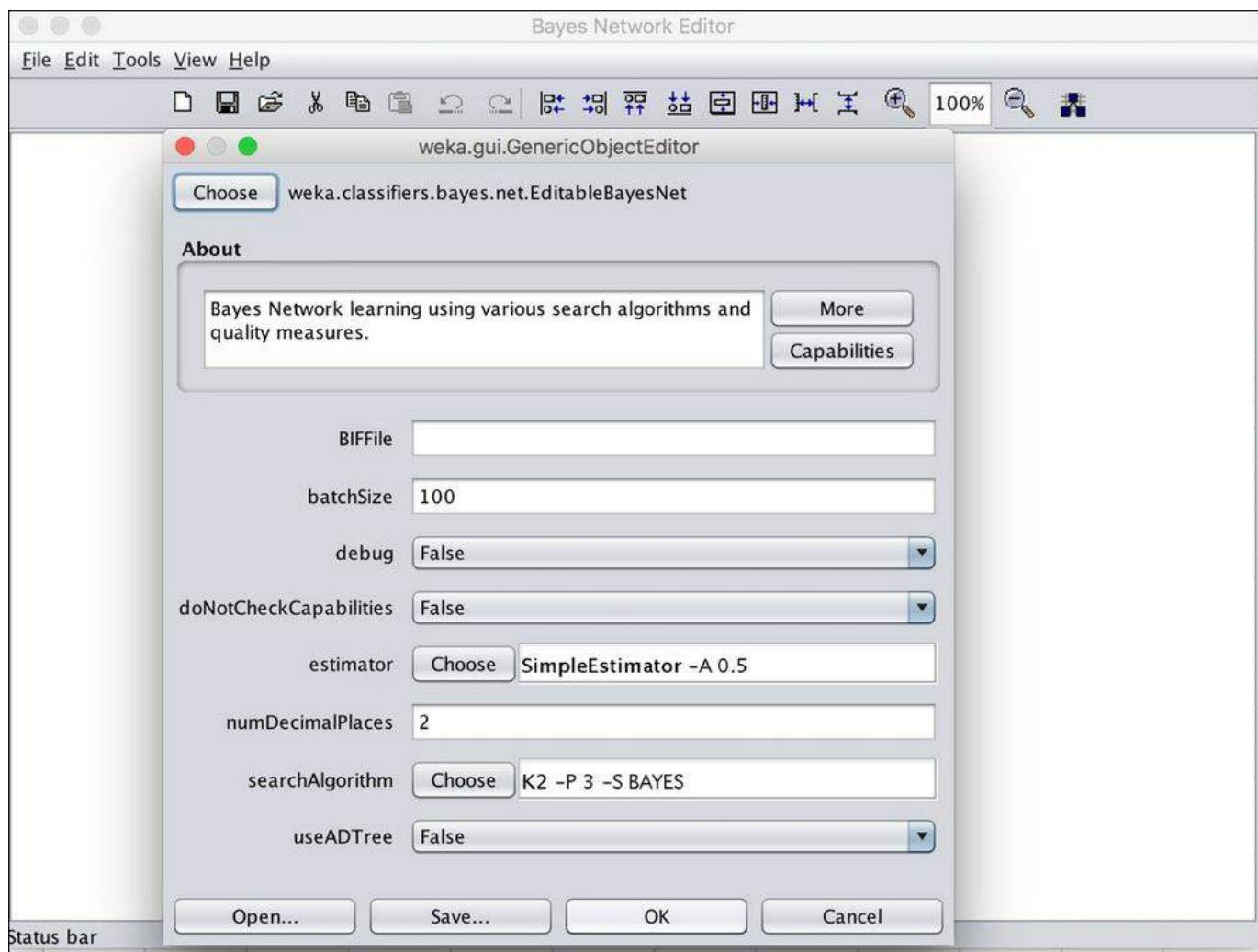


Figure 23. WEKA Bayes Network – configuring search algorithm

The learned structure and parameters of the BayesNet are shown in the output screen

in Figure 24:

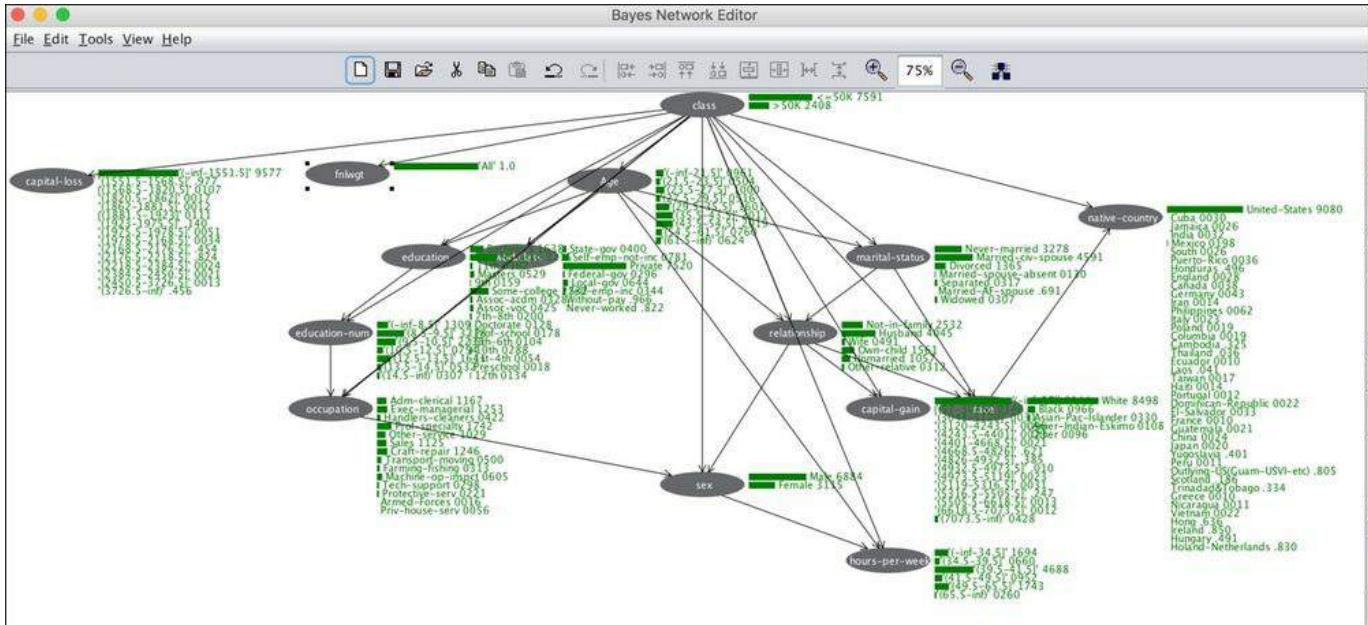


Figure 24. WEKA Bayes Network – Learned parameter and structure

Case study

In this section, we will perform a case study with real-world machine learning datasets to illustrate some of the concepts from Bayesian networks.

We will use the UCI Adult dataset, also known as the *Census Income* dataset (<http://archive.ics.uci.edu/ml/datasets/Census+Income>). This dataset was extracted from the United States Census Bureau's 1994 census data. The donors of the data is Ronny Kohavi and Barry Becker, who were with Silicon Graphics at the time. The dataset consists of 48,842 instances with 14 attributes, with a mix of categorical and continuous types. The target class is binary.

Business problem

The problem consists of predicting the income of members of a population based on census data, specifically, whether their income is greater than \$50,000.

Machine learning mapping

This is a problem of classification and this time around we will be training Bayesian graph networks to develop predictive models. We will be using linear, non-linear, and ensemble algorithms, as we have done in experiments in previous chapters.

Data sampling and transformation

In the original dataset, there are 3,620 examples with missing values and six duplicate or conflicting instances. Here we include only examples with no missing values. This set, without unknowns, is divided into 30,162 training instances and 15,060 test instances.

Feature analysis

The features and their descriptions are given in *Table 3*:

Feature	Type information
age	continuous.
workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt	continuous.
education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num	continuous.
marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex	Female, Male.
capital-gain	continuous.
capital-loss	continuous.
hours-per-week	continuous.
native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Table 3. UCI Adult dataset – features

The dataset is split by label as 24.78% (>50K) to 75.22% (<= 50K). Summary statistics of key features are given in *Figure 25*:

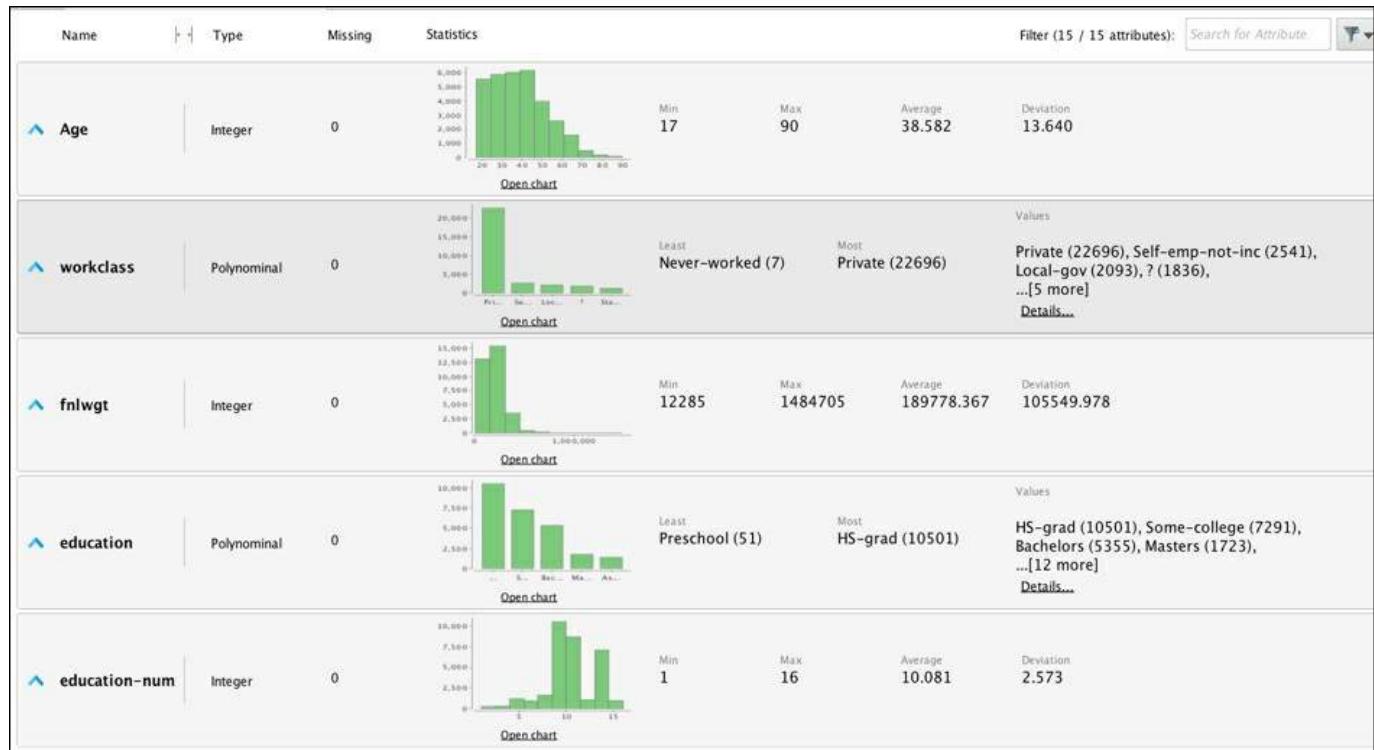


Figure 25. Feature summary statistics

Models, results, and evaluation

We will perform detailed analysis on the Adult dataset using different flavors of Bayes network structures and with regular linear, non-linear, and ensemble algorithms. Weka also has an option to visualize the graph model on the trained dataset using the menu item, as shown in *Figure 26*. This is very useful when the domain expert wants to understand the assumptions and the structure of the graph model. If the domain expert wants to change or alter the network, it can be done easily and saved using the Bayes Network editor.

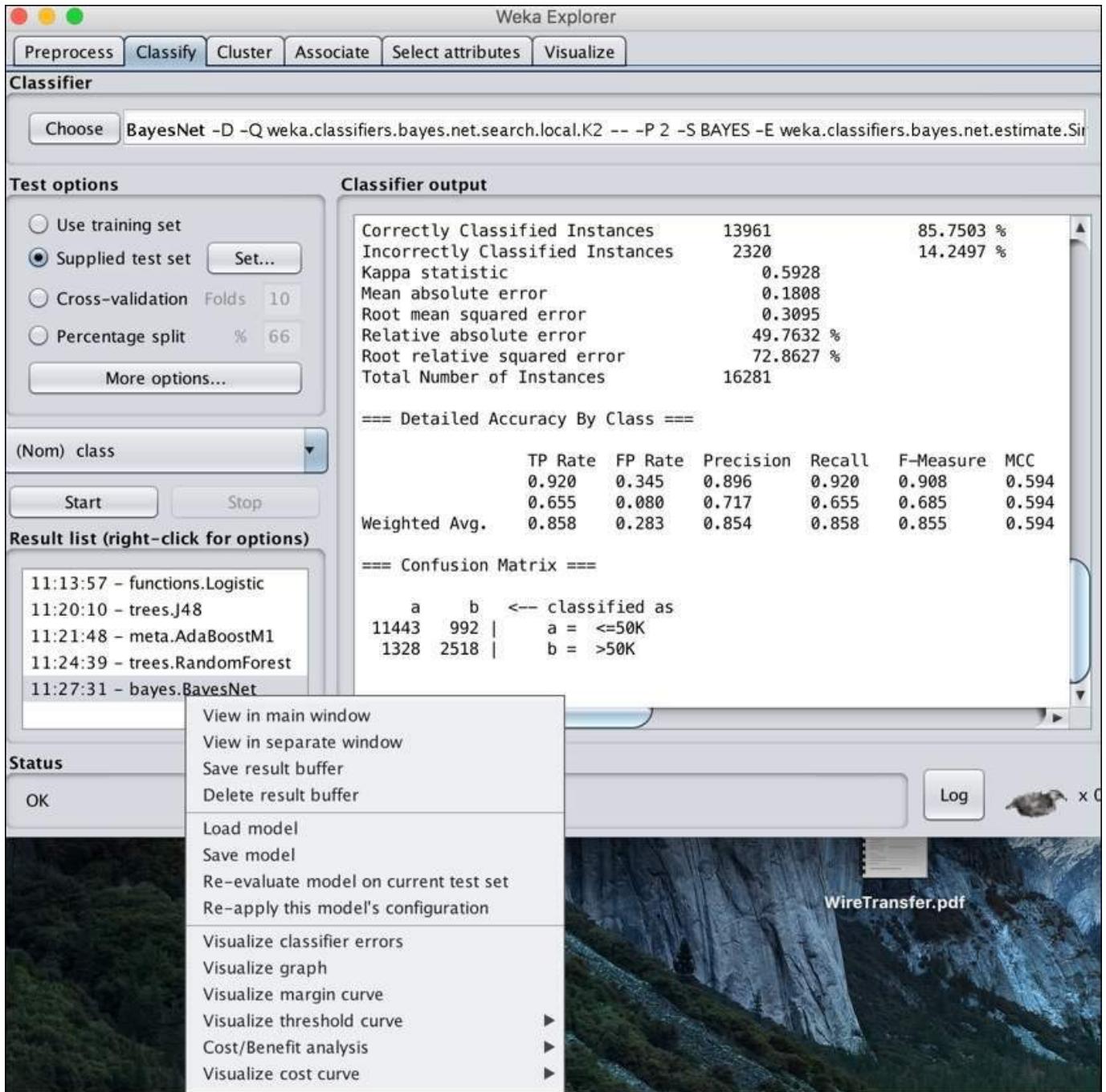


Figure 26. Weka Explorer – visualization menu

Figure 27 shows the visualization of the trained Bayes Network model's graph structure:

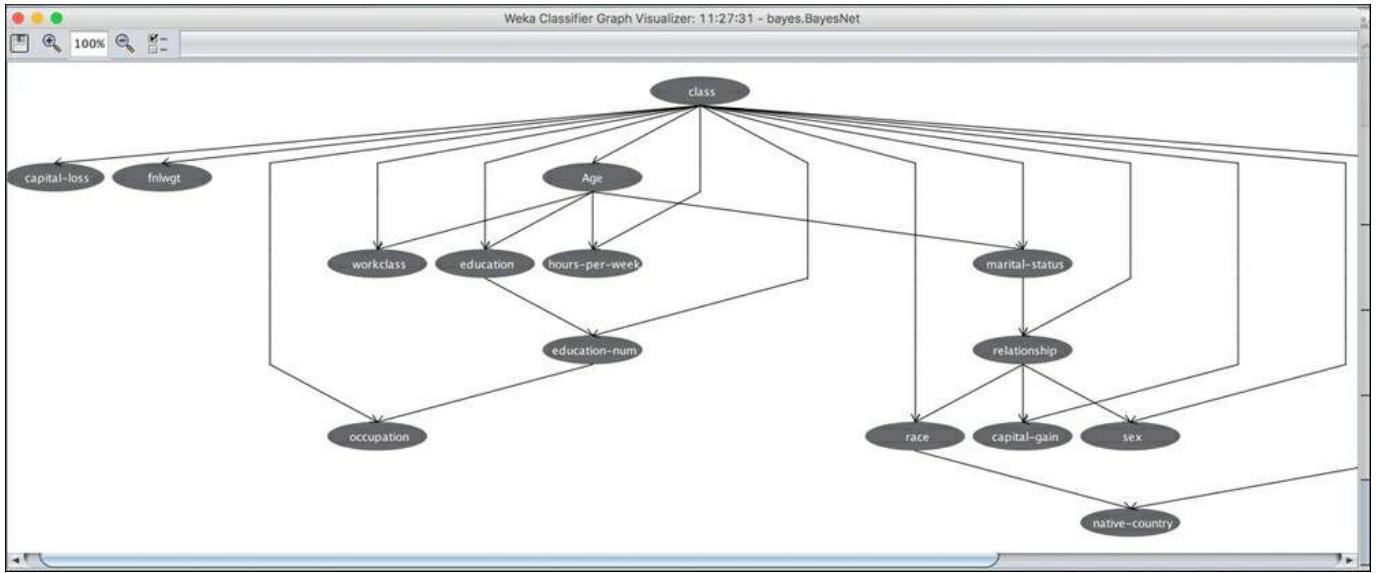


Figure 27: Visualization of learned structure of the Bayesian network.

The algorithms used for experiments are:

- Bayesian network Classifiers
- Naïve Bayes with default Kernel estimation on continuous data
- Naïve Bayes with supervised discretization on continuous data
- Tree augmented network (TAN) with search-score structure parameter learning using the K2 algorithm and a choice of three parents per node
- Bayesian network with search and score
- Searching using Hill Climbing and K2
- Scoring using Simple Estimation
- Choice of parents changed from two to three to illustrate the effect on metrics
- Non-Bayesian algorithms
- Logistic Regression (default parameters)
- KNN (IBK with 10 Neighbors)
- Decision Tree (J48, default parameters)
- AdaBoostM1 (DecisionStump and default parameters)
- Random Forest (default parameters)

Table 4 presents the evaluation metrics for all the learners used in the experiments, including Bayesian network classifiers as well as the non-Bayesian algorithms:

Algorithms	TP	FP	Precision	Recall	F-	MCC	ROC	PRC
------------	----	----	-----------	--------	----	-----	-----	-----

	Rate	Rate			Measure		Area	Area
Naïve Bayes (Kernel Estimator)	0.831	0.391	0.821	0.831	0.822	0.494	0.891	0.906
Naïve Bayes (Discretized)	0.843	0.191	0.861	0.843	0.848	0.6	0.917	0.93
TAN (K2, 3 Parents, Simple Estimator)	0.859	0.273	0.856	0.859	0.857	0.6	0.916	0.931
BayesNet (K2, 3 Parents, Simple Estimator)	0.863	0.283	0.858	0.863	0.86	0.605	0.934	0.919
BayesNet (K2, 2 Parents, Simple Estimator)	0.858	0.283	0.854	0.858	0.855	0.594	0.917	0.932
BayesNet (Hill Climbing, 3 Parents, Simple Estimator)	0.862	0.293	0.857	0.862	0.859	0.602	0.918	0.933
Logistic Regression	0.851	0.332	0.844	0.851	0.845	0.561	0.903	0.917
KNN (10)	0.834	0.375	0.824	0.834	0.826	0.506	0.867	0.874
Decision Tree (J48)	0.858	0.300	0.853	0.858	0.855	0.590	0.890	0.904
AdaBoostM1	0.841	0.415	0.833	0.841	0.826	0.513	0.872	0.873
Random Forest	0.848	0.333	0.841	0.848	0.843	0.555	0.896	0.913

Table 4. Classifier performance metrics

Analysis of results

Naïve Bayes with supervised discretization shows relatively better performance than kernel estimation. This gives a useful hint that discretization, which is needed in most Bayes networks, will play an important role.

The results in the table show continuous improvement when Bayes network complexity is increased. For example, Naïve Bayes with discretization assumes independence from all features and shows a TP rate of 84.3, the TAN algorithm where there can be one more parent shows a TP rate of 85.9, and BN with three parents shows the best TP rate of 86.2. This clearly indicates that a complex BN with some nodes having no more than three parents can capture the domain knowledge and encode it well to predict on unseen test data.

Bayes network where structure is learned using search and score (with K2 search with three parents and scoring using Bayes score) and estimation is done using simple estimation, performs the best in almost all the metrics of the evaluation, as shown in the highlighted values.

There is a very small difference between Bayes Networks—where structure is learned using search and score of Hill Climbing—and K2, showing that even local search algorithms can find an optimum.

Bayes network with a three-parent structure beats most linear, non-linear, and ensemble methods such as AdaBoostM1 and Random Forest on almost all the metrics on unseen test data. This shows the strength of BNs in not only learning the structure and parameters on small datasets with large number of missing values as well as predicting well on unseen data, but in beating other sophisticated algorithms too.

Summary

PGMs capture domain knowledge as relationships between variables and represent joint probabilities. They are used in a range of applications.

Probability maps an event to a real value between 0 and 1 and can be interpreted as a measure of the frequency of occurrence (frequentist view) or as a degree of belief in that occurrence (Bayesian view). Concepts of random variables, conditional probabilities, Bayes' theorem, chain rule, marginal and conditional independence and factors form the foundations to understanding PGMs. MAP and Marginal Map queries are ways to ask questions about the variables and relationships in the graph.

The structure of graphs and their properties such as paths, trails, cycles, sub-graphs, and cliques are vital to the understanding of Bayesian networks. Representation, Inference, and Learning form the core elements of networks that help us capture, extract, and make predictions using these methods. From the representation of graphs, we can reason about the flow of influence and detect independencies that help reduce the computational load when querying the model. Junction trees, variable elimination, and belief propagation methods likewise make inference from queries more tractable by reductive steps. Learning from Bayesian networks involves generating the structure and model parameters from the data. We discussed several methods of learning parameters and structure.

Markov networks (MN), which have undirected edges, also contain interactions that can be captured using parameterization techniques such as Gibbs parameterization, Factor Graphs, and Log-Linear Models. Independencies in MN govern flows of influence, as in Bayesian networks. Inference techniques are also similar. Learning of parameters and structure in MN is hard, and approximate methods are used. Specialized networks such as **Tree augmented networks (TAN)** make assumptions of independence amongst nodes and are very useful in some applications. Markov Chains and hidden Markov models are other specialty networks that also find application in a range of fields.

Open Markov and Weka Bayesian Network GUI are introduced as Java-based tools for PGMs. The case study in this chapter used Bayesian Networks to learn from the UCI Adult census dataset and its performance was compared to other (non-PGM) classifiers.

References

1. Daphne Koller and Nir Friedman (2009). *Probabilistic Graphical Models*. MIT Press. ISBN 0-262-01319-3.
2. T. Verma and J. Pearl (1988), In proceedings for fourth workshop on Uncertainty in Artificial Intelligence, Montana, Pages 352-359. Causal Networks- Semantics and expressiveness.
3. Dagum, P., and Luby, M. (1993). *Approximating probabilistic inference in Bayesian belief networks is NP hard*. Artificial Intelligence 60(1):141–153.
4. U. Bertele and F. Brioschi, *Nonserial Dynamic Programming*, Academic Press. New York, 1972.
5. Shenoy, P. P. and G. Shafer (1990). *Axioms for probability and belief-function propagation*, in Uncertainty in Artificial Intelligence, 4, 169-198, North-Holland, Amsterdam
6. Bayarri, M.J. and DeGroot, M.H. (1989). *Information in Selection Models*. Probability and Bayesian Statistics, (R. Viertl, ed.), Plenum Press, New York.
7. Spiegelhalter and Lauritzen (1990). *Sequential updating of conditional probabilities on directed graphical structures*. Networks 20. Pages 579-605.
8. David Heckerman, Dan Geiger, David M Chickering (1995). In journal of Machine Learning. *Learning Bayesian networks: The combination of knowledge and statistical data*.
9. Friedman, N., Geiger, D., & Goldszmidt, M. (1997). *Bayesian network classifiers*. Machine Learning, 29, 131– 163.
10. Isham, V. (1981). *An introduction to spatial point processes and Markov random fields*. International Statistical Review, 49(1):21–43
11. Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger, *Factor graphs and sum-product algorithm*, IEEE Trans. Info. Theory, vol. 47, pp. 498–519, Feb. 2001.
12. Kemeny, J. G. and Snell, J. L. *Finite Markov Chains*. New York: Springer-Verlag, 1976.
13. Baum, L. E.; Petrie, T. (1966). *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*. The Annals of Mathematical Statistics. 37 (6): 1554–1563.
14. Gelman, A., Hwang, J. and Vehtari, A. (2004). *Understanding predictive information criteria for Bayesian models*. Statistics and Computing Journal 24: 997. doi:10.1007/s11222-013-9416-2
15. Dimitris. Margaritis (2003). *Learning Bayesian Network Model Structure From Data*. Ph.D Thesis Carnegie Mellon University.

16. John Lafferty, Andrew McCallum, Fernando C.N. Pereira (2001). *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*, International Conference on Machine Learning 2001 (ICML 2001), pages 282-289.

Chapter 7. Deep Learning

In [Chapter 2, Practical Approach to Real-World Supervised Learning](#), we discussed different supervised classification techniques that are general and can be used in a wide range of applications. In the area of supervised non-linear techniques, especially in computer-vision, deep learning and its variants are having a remarkable impact. We find that deep learning and associated methodologies can be applied to image-recognition, image and object annotation, movie descriptions, and even areas such as text classification, language modeling, translations, and so on. (*References [1, 2, 3, 4, and 5]*)

To set the stage for deep learning, we will start with describing what neurons are and how they can be arranged to build multi-layer neural networks, present the core elements of these networks, and explain how they work. We will then discuss the issues and problems associated with neural networks that gave rise to advances and structural changes in deep learning. We will learn about some building blocks of deep learning such as Restricted Boltzmann Machines and Autoencoders. We will then explore deep learning through different variations in supervised and unsupervised learning. Next, we will take a tour of Convolutional Neural Networks (CNN) and by means of a use case, illustrate how they work by deconstructing an application of CNNs in the area of computer-vision. We will introduce Recurrent Neural Networks (RNN) and its variants and how they are used in the text/sequence mining fields. We will finally present a case study using real-life data of MNIST images and use it to compare/contrast different techniques. We will use DeepLearning4J as our Java toolkit for performing these experiments.

Multi-layer feed-forward neural network

Historically, artificial neural networks have been largely identified by multi-layer feed-forward perceptrons, and so we will begin with a discussion of the primitive elements of the structure of such networks, how to train them, the problem of overfitting, and techniques to address it.

Inputs, neurons, activation function, and mathematical notation

A single neuron or perceptron is the same as the unit described in the Linear Regression topic in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning*. In this chapter, the data instance vector will be represented by x and has d dimensions, and each dimension can be represented as $x_1, x_2 \dots x_d$. The weights associated with each dimension are represented as a weight vector w that has d dimensions, and each dimension can be represented as $w_1, w_2 \dots w_d$. Each neuron has an extra input b , known as the bias, associated with it.

Neuron pre-activation performs the linear transformation of inputs given by:

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^T \mathbf{x}$$

The activation function is given by $g(\cdot)$, which transforms the neuron input $a(x)$ as follows:

$$h(x) = g(a(\mathbf{x})) = g\left(b + \sum_i w_i x_i\right)$$

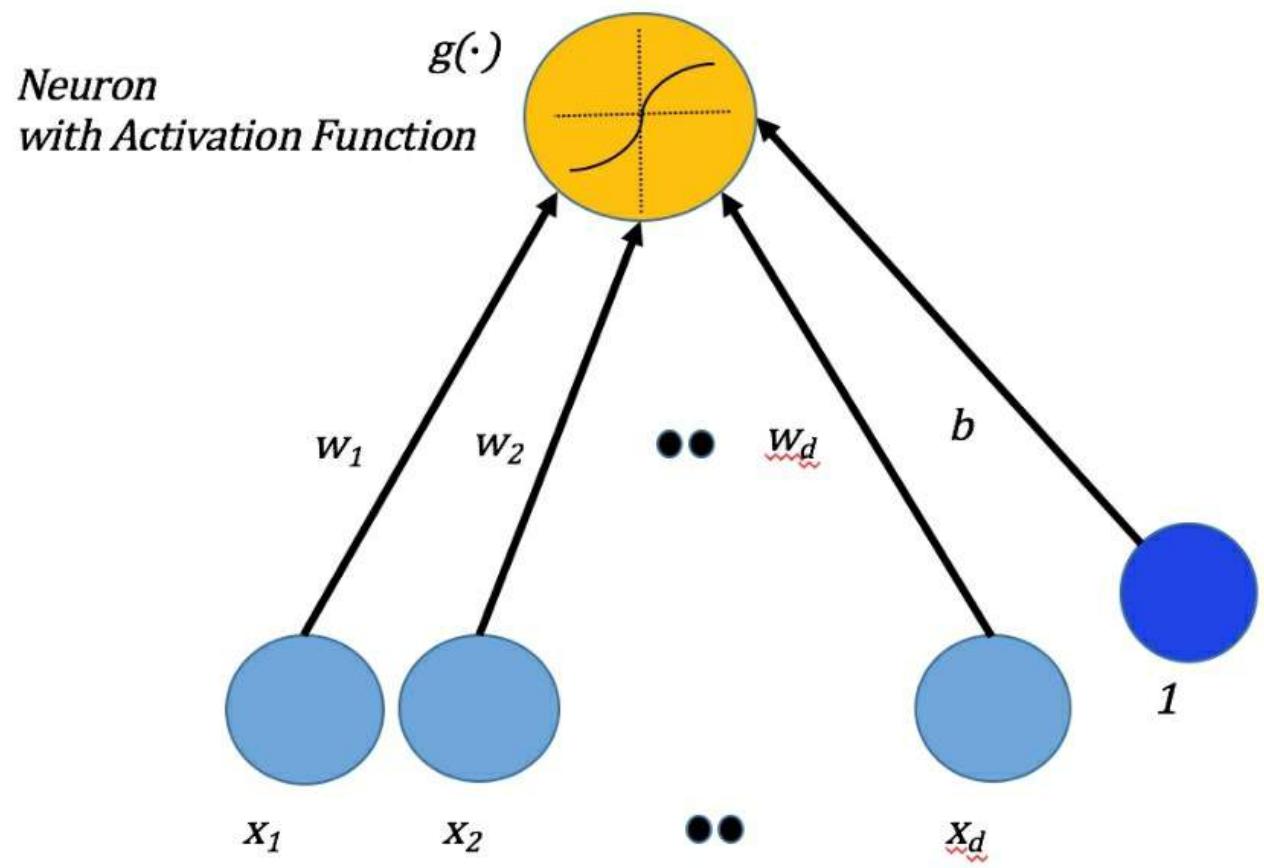


Figure 1. Perceptron with inputs, weights, and bias feeding to generate outputs.

Multi-layered neural network

Multi-layered neural networks are the first step to understanding deep learning networks as the fundamental concepts and primitives of multi-layered nets form the basis of all deep neural nets.

Structure and mathematical notations

We introduce the generic structure of neural networks in this section. Most neural nets are variants of the structure outlined here. We also present the relevant notation that we will use in the rest of the chapter.

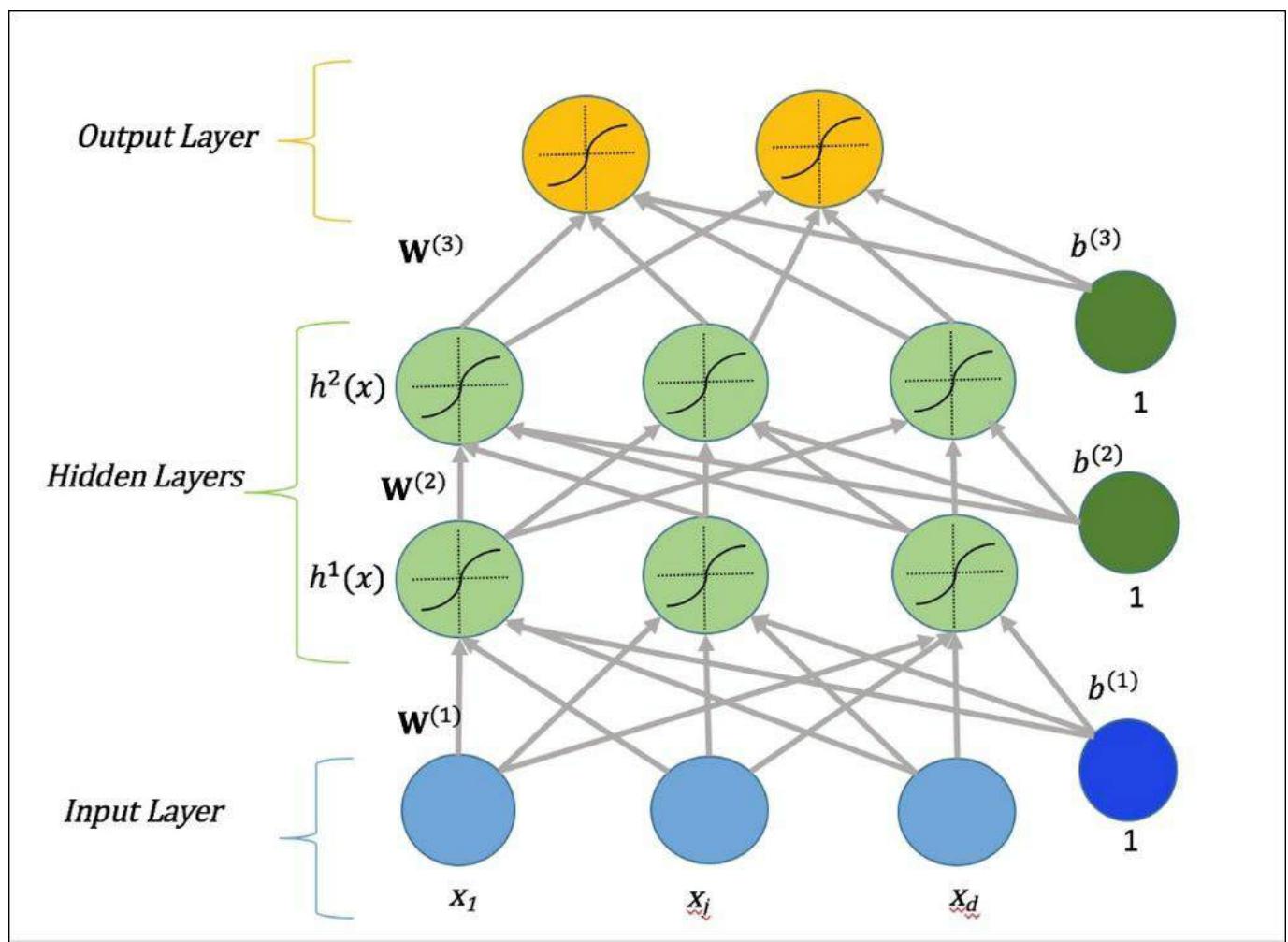


Figure 2. Multilayer neural network showing an input layer, two hidden layers, and an output layer.

The most common supervised learning algorithms pertaining to neural networks use multi-layered perceptrons. The Input Layer consists of several neurons, each connected independently to the input, with its own set of weights and bias. In addition to the Input Layer, there are one or more layers of neurons known as Hidden Layers. The input layer neurons are connected to every neuron in the first hidden layer, that layer is similarly connected to the next hidden layer, and so on, resulting in a fully connected network. The layer of neurons connected to the last hidden layer is called the Output Layer.

Each hidden layer is represented by $h^k(x)$ where k is the layer. The pre-activation for layer $0 < k < l$ is given by:

$$\mathbf{a}^k(x) = \mathbf{b}^k(x) + \mathbf{W}^k \mathbf{h}^{k-1}(x)$$

The hidden layer activation for $1 < k \leq L$:

$$\mathbf{h}^k(x) = \mathbf{g}(\mathbf{a}^k(x))$$

The final output layer activation is:

$$\mathbf{h}^{l+1}(x) = \mathbf{o}(\mathbf{a}^{l+1}(x))$$

The output is generally one class per neuron and it is tuned in such a way that only one neuron activates and all others have 0 as the output. A softmax function with $\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a})$ is used for giving the result.

Activation functions in NN

Some of the most well-known activation functions that are used in neural networks are given in the following sections and they are used because the derivatives needed in learning can be expressed in terms of the function itself.

Sigmoid function

Sigmoid activation functions are given by the following equation:

$$g(a) = \text{sigma}(a) = \frac{1}{1 + \exp(-a)}$$

It can be seen as a bounded, strictly increasing and positive transformation function that squashes the values between 0 and 1.

Hyperbolic tangent ("tanh") function

The Tanh function is given by the following equation:

$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

It can be seen as bounded, strictly increasing, but as a positive or negative transformation function that squashes the values between -1 and 1.

Training neural network

In this section, we will discuss the key elements of training neural networks from input training sets, in much the same fashion as we did in [Chapter 2, Practical Approach to Real-World Supervised Learning](#). The dataset is denoted by D and consists of individual data instances. The instances are normally represented as the set $\{x^1, x^2 \dots x^n\}$. The labels for each instance are represented as the set $\{y^1, y^2 \dots y^n\}$. The entire labeled dataset with numeric or real-valued features is represented as

paired elements in a set as given by $\mathcal{D} = \{(x^1, y^1), (x^2, y^2), \dots, (x^n, y^n)\}$.

Empirical risk minimization

Empirical risk minimization is a general machine learning concept that is used in many classifications or supervised learning. The main idea behind this technique is to convert a training or learning problem into an optimization problem (*References [13]*).

Given the parameters for a neural network as $\theta = (\{\mathbf{W}^1, \mathbf{W}^2, \dots, \mathbf{W}^{l+1}\}, \{\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^{L+1}\})$ the training problem can be seen as finding the best parameters (θ) such that

$$\arg \min_{\theta} \frac{1}{n} \sum_{t=1}^n l(f(x^t; \theta), y^t) + \lambda \Omega(\theta)$$

Where

$\sum_{t=1}^n l(f(x^t; \theta), y^t)$ = Average Loss using Loss function and $\Omega(\theta)$ is penalty function or Regularizer
 Stochastic gradient descent (SGD) discussed in [Chapter 2, Practical Approach to Real-World Supervised Learning](#) and [Chapter 5, Real-time Stream Machine Learning](#), is commonly used as the optimization procedure. The SGD applied to training neural networks is:

1. initialize $\theta = (\{\mathbf{W}^1, \mathbf{W}^2, \dots, \mathbf{W}^{l+1}\}, \{\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^{L+1}\})$
2. for $i=1$ to N epochs

1. for each training sample (x^t, y^t)
 1. $\Delta = -\nabla_{\theta} (l(f(x^t; \theta), y^t)) - \lambda \nabla_{\theta} \Omega(\theta)$ //
 2. find the gradient of function
 3. $\theta = \theta + a \Delta$ //move in direction

The learning rate used here (a) will impact the algorithm convergence by reducing the oscillation near the optimum; choosing the right value of a is often a hyper parameter search that needs the validation techniques described in [Chapter 2, Practical Approach to Real-World Supervised Learning](#).

Thus, to learn the parameters of a neural network, we need to choose a way to do

parameter initialization, select a loss function $\left(l(f(\mathbf{x}'; \boldsymbol{\theta}), y')\right)$, compute the

parameter gradients $\nabla_{\boldsymbol{\theta}}\left(l(\mathbf{f}(\mathbf{x}'; \boldsymbol{\theta}), y')\right)$, propagate the losses back, select the regularization/penalty function $O(\cdot)$, and compute the gradient of regularization $\nabla_{\boldsymbol{\theta}}\Omega(\boldsymbol{\theta})$. In the next few sections, we will describe this step by step.

Parameter initialization

The parameters of neural networks are the weights and biases of each layer from the input layer, through hidden layers, to the output layer. There has been much research in this area as the optimization depends on the start or initialization. Biases are generally set to value 0. The weight initialization depends on the activation functions as some, such as tanh, value 0, cannot be used. Generally, the way to initialize the weights of each layer is by random initialization using a symmetric function with a user-defined boundary.

Loss function

The loss function's main role is to maximize how well the predicted output label matches the class of the input data vector.

Thus, maximization $f(x)_c = p(y=c|\mathbf{x})$ is equivalent to minimizing the negative of the log-likelihood or cross-entropy:

$$l(\mathbf{f}(\mathbf{x}), y) = -\log(f(\mathbf{x})_y)$$

Gradients

We will describe gradients at the output layer and the hidden layer without going into the derivation as it is beyond the scope of this book. Interested readers can see the derivation in the text by Rumelhart, Hinton and Williams (*References [6]*).

Gradient at the output layer

Gradient at the output layer can be calculated as:

$$= \nabla_{\mathbf{f}(\mathbf{x})} - \log(f(\mathbf{x})_y)$$

$$= \frac{-1}{f(\mathbf{x})_y} \begin{bmatrix} 1_{(y=0)} \\ \dots \\ 1_{(y=c-1)} \end{bmatrix}$$

$$= \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y}$$

Where $e(y)$ is called the "one hot vector" where only one value in the vector is 1 corresponding to the right class y and the rest are 0.

The gradient at the output layer pre-activation can be calculated similarly:

$$= \nabla_{a^{l+1}(\mathbf{x})} - \log(f(\mathbf{x})_y)$$

$$= -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

Gradient at the Hidden Layer

A hidden layer gradient is computed using the chain rule of partial differentiation.

Gradient at the hidden layer $= \nabla_{\mathbf{h}^k(\mathbf{x})} - \log(f(\mathbf{x})_y)$

$$= \mathbf{W}^{k+1} \left(\nabla_{a^{k+1}(x)} - \log(f(\mathbf{x})_y) \right)$$

Gradient at the hidden layer pre-activation can be shown as:

$$= \left(\nabla_{\mathbf{h}^k(\mathbf{x})} - \log(f(\mathbf{x})_y) \right)^T \nabla_{\mathbf{a}^k(\mathbf{x})} \mathbf{h}^k(\mathbf{x})$$

$$= \left(\nabla_{\mathbf{h}^k(\mathbf{x})} - \log(f(\mathbf{x})_y) \right) \odot [.., g'(a^k \mathbf{x}_j) ...]$$

Since the hidden layer pre-activation needs partial derivatives of the activation functions as shown previously ($g'(a^k \mathbf{x}_j)$), some of the well-known activation functions described previously have partial derivatives in terms of the equation itself, which makes computation very easy.

For example, the partial derivative of the sigmoid function is $g'(a) = g(a)(1 - g(a))$ and, for the tanh function, it is $1 - g(a)^2$.

Parameter gradient

The loss gradient of parameters must be computed using gradients of weights and biases. Gradient of weights can be shown as:

$$= \nabla_{\mathbf{w}^k} - \log(f(\mathbf{x})_y)$$

$$= \left(\nabla_{a^k(x)} - \log(f(\mathbf{x})_y) \right) \mathbf{h}^{k-1}(\mathbf{x})^T$$

Gradient of biases can be shown as:

$$= \nabla_{\mathbf{b}^k} - \log(f(\mathbf{x})_y)$$

$$= \nabla_{z^k(x)} - \log(f(\mathbf{x}))_y$$

Feed forward and backpropagation

The aim of neural network training is to adjust the weights and biases at each layer so that, based on the feedback from the output layer and the loss function that estimates the difference between the predicted output and the actual output, that difference is minimized.

The neural network algorithm based on initial weights and biases can be seen as forwarding the computations layer by layer as shown in the acyclic flow graph with one hidden layer to demonstrate the flow:

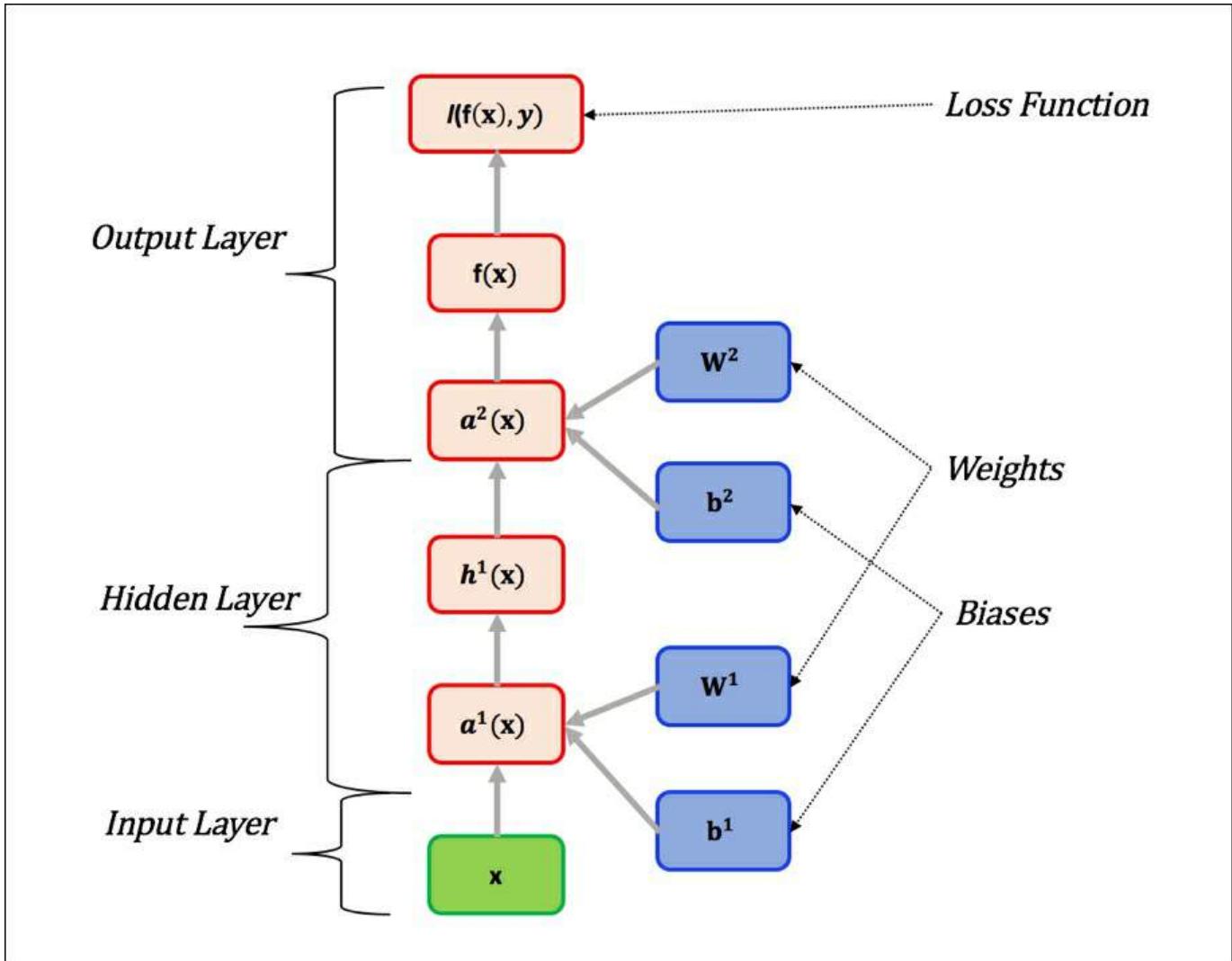


Figure 3: Neural network flow as a graph in feed forward.

From the input vector and pre-initialized values of weights and biases, each subsequent element is computed: the pre-activation, hidden layer output, final layer pre-activation, final layer output, and loss function with respect to the actual label. In backward propagation, the flow is exactly reversed, from the loss at the output down to the weights and biases of the first layer, as shown in the following figure:

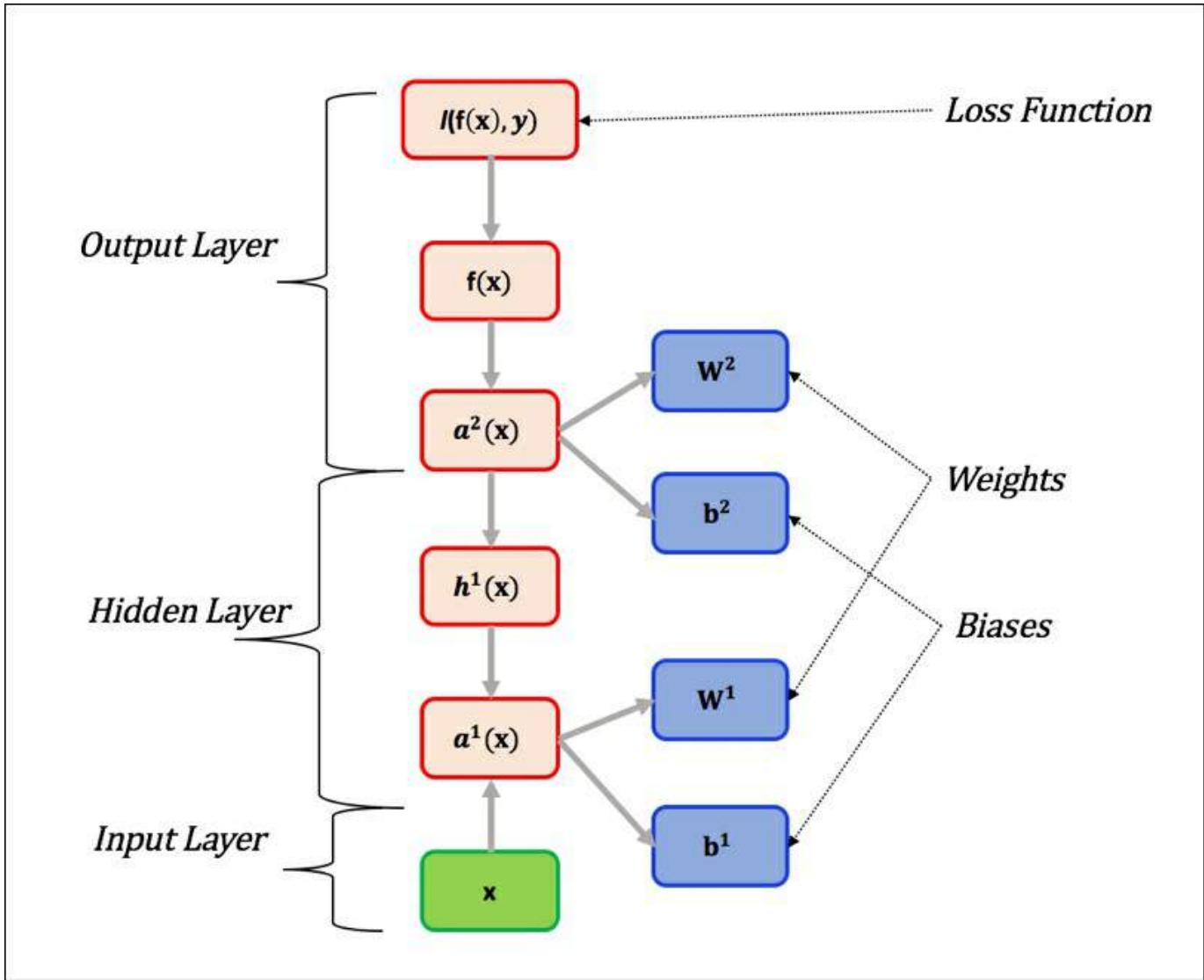


Figure 4: Neural network flow as a graph in back propagation.

How does it work?

The backpropagation algorithm (*References [6 and 7]*) in its entirety can be summarized as follows:

Compute the output gradient before activation:

$$= \nabla_{a^{l+1}(x)} - \log(f(\mathbf{x})_y) = -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

For hidden layers $k=l+1$ to l :

Compute the gradient of hidden layer parameters:

$$\nabla_{W^k} - \log(f(\mathbf{x})_y) = (\nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y)) \mathbf{h}^{k-1}(\mathbf{x})^T$$

$$\nabla_{b^k} - \log(f(\mathbf{x})_y) = \nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y)$$

Compute the gradient of the hidden layer below the current:

$$\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) = \mathbf{W}^k (\nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y))$$

Compute the gradient of the layer before activation:

$$\begin{aligned} & \left(\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) \right)^T \nabla_{a^{k-1}(\mathbf{x})} \mathbf{h}^k(\mathbf{x}) \\ &= \left(\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) \right) \odot [\dots, g'(a^{k-1} \mathbf{x}_j), \dots] \end{aligned}$$

Regularization

In the empirical risk minimization objective defined previously, regularization is used to address the over-fitting problem in machine learning as introduced in

[Chapter 2](#), *Practical Approach to Real-World Supervised Learning*. The well-known regularization functions are given as follows.

L2 regularization

This is applied only to the weights and not to the biases and is given for layers connecting (i,j) components as:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j (W_{i,j}^k)^2$$

$$= \sum_k \|\mathbf{W}^k\|_F^2$$

Also, the gradient of the regularizer can be computed as $\nabla_{\theta} \Omega(\theta) = 2\mathbf{W}^k$. They are often interpreted as the "Gaussian Prior" over the weight distribution.

L1 regularization

This is again applied only to the weights and not to the biases and is given for layers connecting (i,j) components as:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^k|$$

And the gradient of this regularizer can be computed as $\nabla_{\theta} \Omega(\theta) = sign(\mathbf{W}^k)$. It is often interpreted as the "Laplacian Prior" over the weight distribution.

Limitations of neural networks

In this section, we will discuss in detail the issues faced by neural networks, which will become the stepping stone for building deep learning networks.

Vanishing gradients, local optimum, and slow training

One of the major issues with neural networks is the problem of "vanishing gradient" (*References [8]*). We will try to give a simple explanation of the issue rather than exploring the mathematical derivations in depth. We will choose the sigmoid activation function and a two-layer neural network, as shown in the following figure, to demonstrate the issue:

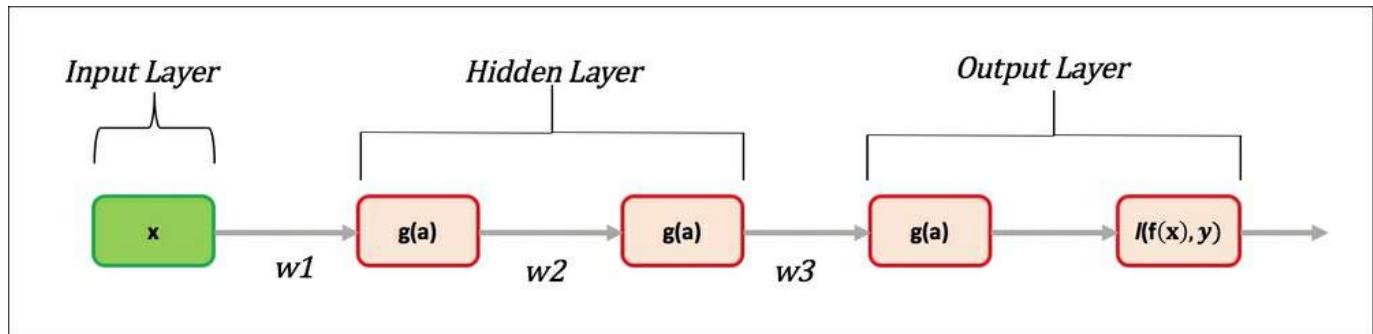


Figure 5: Vanishing Gradient issue.

As we saw in the activation function description, the sigmoid function squashes the output between the range 0 and 1. The derivative of the sigmoid function $g'(a) = g(a)(1 - g(a))$ has a range between 0 and 0.25. The goal of learning is to minimize the

$$\frac{\delta l(f(x), y)}{\delta x} \rightarrow 0$$

output loss, that is, $\frac{\delta l(f(x), y)}{\delta x}$. In general, the output error does not go to 0, so maximum iterations; a user-specified parameter determines the quality of learning and backpropagation of the errors.

Simplifying to illustrate the effect of output error on the input weight layer:

$$\frac{\delta l(f(x), y)}{\delta w1} = \frac{\delta l(f(x), y)}{\delta \text{output}} * \frac{\delta \text{output}}{\delta \text{hidden2}} * \frac{\delta \text{hidden2}}{\delta \text{hidden1}} * \frac{\delta \text{hidden1}}{\delta w1}$$

Each of the transformations, for instance, from output to hidden, involves multiplication of two terms, both less than 1:

$$\frac{\delta \text{output}}{\delta \text{hidden2}} = w3 * g'(a)$$

Thus, the value becomes so small when it reaches the input layer that the propagation of the gradient has almost vanished. This is known as the vanishing gradient problem.

A paradoxical situation arises when you need to add more layers to make features more interesting in the hidden layers. But adding more layers also increases the errors. As you add more layers, the input layers become "slow to train," which causes the output layers to be more inaccurate as they are dependent on the input layers; further, and for the same number of iterations, the errors increase with the increase in the number of layers.

With a fixed number of maximum iterations, more layers and slow propagation of errors can lead to a "local optimum."

Another issue with basic neural networks is the number of parameters. Finding effective size and weights for each hidden layer and bias becomes more challenging with the increase in the number of layers. If we increase the number of layers, the parameters increase in polynomials. Fitting the parameters for the data requires a large number of data samples. This can result in the problem discussed before, that is, overfitting.

In the next few sections, we will start learning about the building blocks of deep learning that help overcome these issues.

Deep learning

Deep learning includes architectures and techniques for supervised and unsupervised learning with the capacity to internalize the abstract structure of high-dimensional data using networks composed of building blocks to create discriminative or generative models. These techniques have proved enormously successful in recent years and any reader interested in mastering them must become familiar with the basic building blocks of deep learning first and understand the various types of networks in use by practitioners. Hands-on experience building and tuning deep neural networks is invaluable if you intend to get a deeper understanding of the subject. Deep learning, in various domains such as image classification and text learning, incorporates feature generation in its structures thus making the task of mining the features redundant in many applications. The following sections provide a guide to the concepts, building blocks, techniques for composing architectures, and training deep networks.

Building blocks for deep learning

In the following sections, we introduce the most important components used in deep learning, including Restricted Boltzmann machines, Autoencoders, and Denoising Autoencoders, how they work, and their advantages and limitations.

Rectified linear activation function

The Reclin function is given by the equation:

$$g(a) = \text{reclin}(a) = \max(0, a)$$

It can be seen as having a lower bound of 0 and no upper bound, strictly increasing, and a positive transformation function that just does linear transformation of positives.

It is easier to see that the rectified linear unit or ReLu has a derivative of 1 or identity for values greater than 0. This acts as a significant benefit as the derivatives are not squashed and do not have diminishing values when chained. One of the issues with ReLu is that the value is 0 for negative inputs and the corresponding neurons act as "dead", especially when a large negative value is learned for the bias term. ReLu cannot recover from this as the input and derivative are both 0. This is generally solved by having a "leaky ReLu". These functions have a small value for negative

$$g(a) = \text{reclin}(a) = \max(\epsilon a, a)$$

inputs and are given by where $\epsilon = 0.01$, typically.

Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBM) is an unsupervised learning neural network (*References [11]*). The idea of RBM is to extract "more meaningful features" from labeled or unlabeled data. It is also meant to "learn" from the large quantity of unlabeled data available in many domains when getting access to labeled data is costly or difficult.

Definition and mathematical notation

In its basic form RBM assumes inputs to be binary values 0 or 1 in each dimension. RBMs are undirected graphical models having two layers, a visible layer represented

as x and a hidden layer h , and connections W .

RBM defines a distribution over the visible layer that involves the latent variables from the hidden layer. First an energy function is defined to capture the relationship between the visible and the hidden layers in vector form as:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^T \mathbf{W} \mathbf{x} - \mathbf{c}^T \mathbf{x} - \mathbf{b}^T \mathbf{h}$$

In scalar form the energy function can be defined as:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j$$

The probability of the distribution is given by $p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h})) / Z$ where Z is called the "partitioning function", which is an enumeration over all the values of x and h , which are binary, resulting in exponential terms and thus making it intractable!

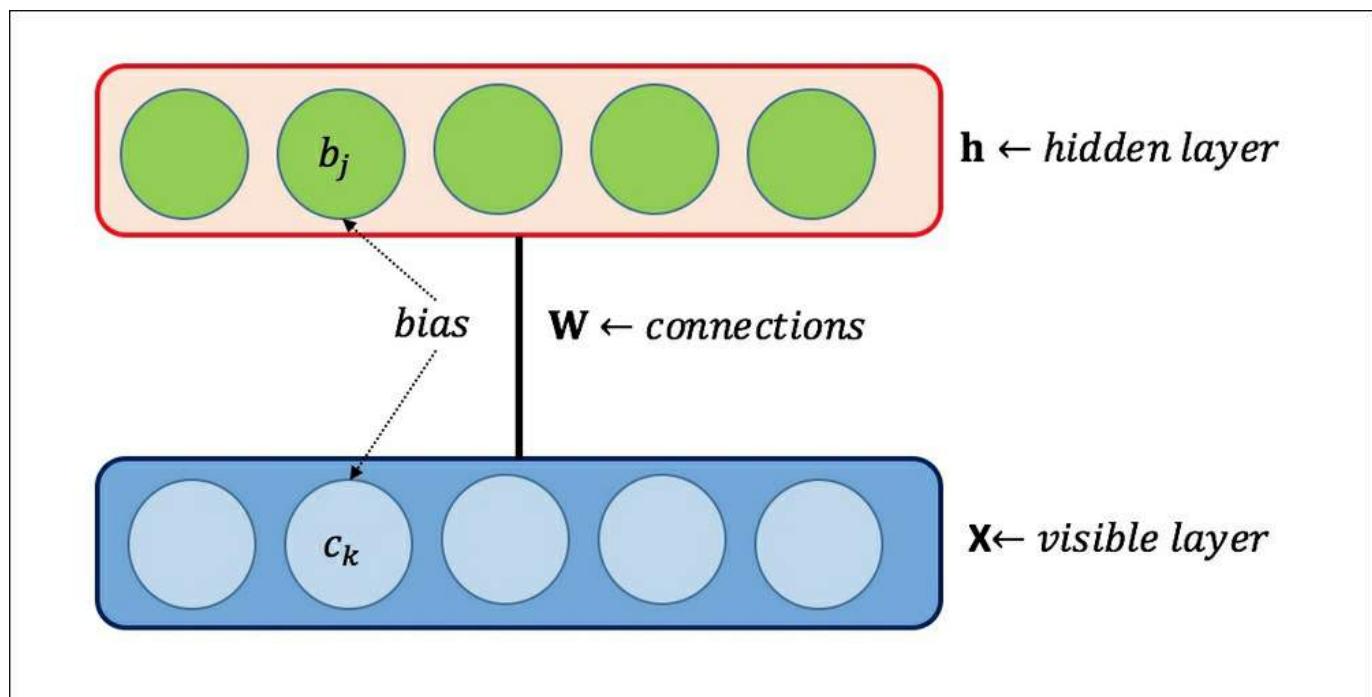


Figure 6: Connection between the visible layer and hidden layer.

The Markov network view of the same in scalar form can be represented using all the pairwise factors, as shown in the following figure. This also makes it clear why it is called a "restricted" Boltzmann machine as there is no connection among units within a given hidden layer or in the visible layers:

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \prod_j \prod_k \exp(W_{j,k} h_j x_k) \prod_j \exp(c_j x_j) \prod_k \exp(b_k h_k)$$

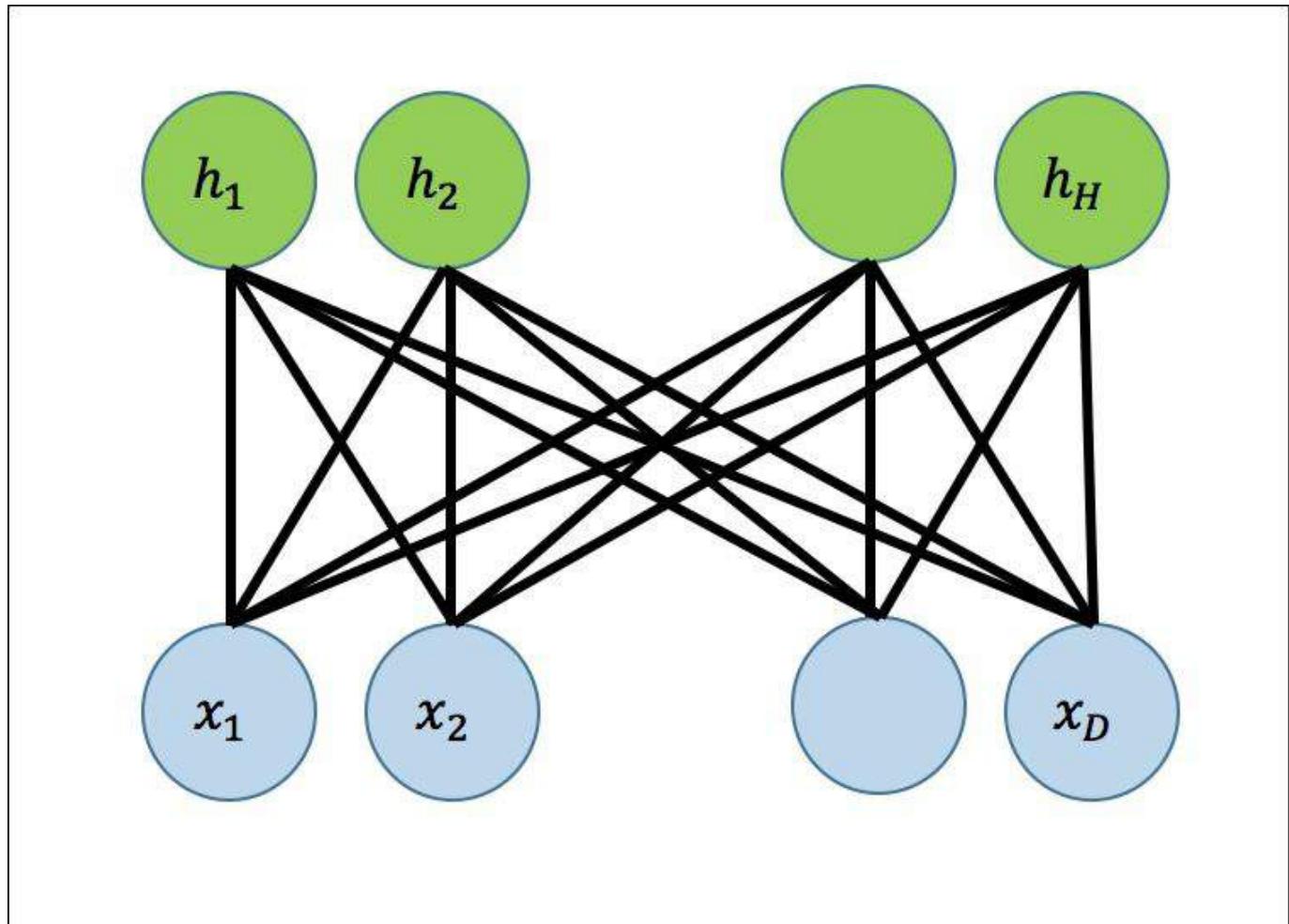


Figure 7: Input and hidden layers as scalars

We have seen that the whole probability distribution function $p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h}))/Z$ is intractable. We will now derive the basic conditional probability distributions for x, h .

Conditional distribution

Although computing the whole $p(x, h)$ is intractable, the conditional distribution of $p(x|h)$ or $p(h|x)$ can be easily defined and shown to be a Bernoulli distribution and tractable:

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$$

$$p(h_j = 1|\mathbf{x}) = \text{sigmoid}(b_j + \mathbf{W}_j \mathbf{x})$$

$$= \frac{1}{1 + \exp(-(b_j + \mathbf{W}_j \mathbf{x}))}$$

Similarly, being symmetric and undirected:

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$$

$$p(x_k = 1|\mathbf{h}) = \text{sigmoid}(c_k + \mathbf{h}^T \mathbf{W}_k)$$

$$= \frac{1}{1 + \exp(-(\mathbf{c}_k + \mathbf{h}^T \mathbf{W}_k))}$$

Free energy in RBM

The distribution of input or the observed variable is:

$$p(\mathbf{x}) = \sum_h p(\mathbf{x}, \mathbf{h})$$

$$p(\mathbf{x}) = \sum_h \exp(-E(\mathbf{x}, \mathbf{h})) / Z$$

$$p(\mathbf{x}) = \frac{\exp\left(\mathbf{c}^T \mathbf{x} + \sum_{h=1}^H \log\left(1 + \exp(b_j + \mathbf{W}_j \mathbf{x})\right)\right)}{Z}$$

$$p(\mathbf{x}) = \frac{\exp(-\mathcal{F}(\mathbf{x}))}{Z}$$

The function $\mathcal{F}(\mathbf{x})$ is called free energy.

Training the RBM

RBMs are trained using the optimization objective of minimizing the average

negative log-likelihood over the entire training data. This can be represented as:

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{T} \sum_t -\log p(\mathbf{x}^t) \text{ where } \theta \text{ are the parameters}$$

The optimization is carried out by using stochastic gradient descent:

$$\frac{\partial -\log p(\mathbf{x}^t)}{\partial \theta} = E_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^t, \mathbf{h})}{\partial \theta} \Big| \mathbf{x}^t \right] - E_{\mathbf{x}, \mathbf{h}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right]$$

$$E_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^t, \mathbf{h})}{\partial \theta} \Big| \mathbf{x}^t \right]$$

The term $E_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^t, \mathbf{h})}{\partial \theta} \Big| \mathbf{x}^t \right]$ is called the "positive phase" and the term

$$E_{\mathbf{x}, \mathbf{h}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right]$$

is called the "negative phase" because of how they affect the probability distributions—the positive phase, because it increases the probability of training data by reducing the free energy, and the negative phase, as it decreases the probability of samples generated by the model.

It has been shown that the overall gradient is difficult to compute analytically because of the "negative phase", as it is computing the expectation over all possible configurations of the input data under the distribution formed by the model and making it intractable!

To make the computation tractable, estimation is carried out using a fixed number of model samples and they are referred to as "negative particles" denoted by N .

The gradient can be now written as the approximation:

$$\frac{\partial -\log p(\mathbf{x}^t)}{\partial \theta} \approx \frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta} - \frac{1}{N} \sum_{\tilde{\mathbf{x}} \in \mathcal{N}} \frac{\partial F(\tilde{\mathbf{x}})}{\partial \theta}$$

Where particles $\tilde{\mathbf{x}}$ are sampled using some sampling techniques such as the Monte Carlo method.

Sampling in RBM

Gibbs sampling is often the technique used to generate samples and learn the probability of $p(x, h)$ in terms of $p(x|h)$ and $p(h|x)$, which are relatively easy to compute, as shown previously.

Gibbs sampling for joint sampling of N random variables $S = (S_1, \dots, S_N)$ is done using N sampling sub-steps of the form $S_i \sim p(S_i | S_{-i})$ where S_{-i} contains samples up to and excluding step S_i . Graphically, this can be shown as follows:

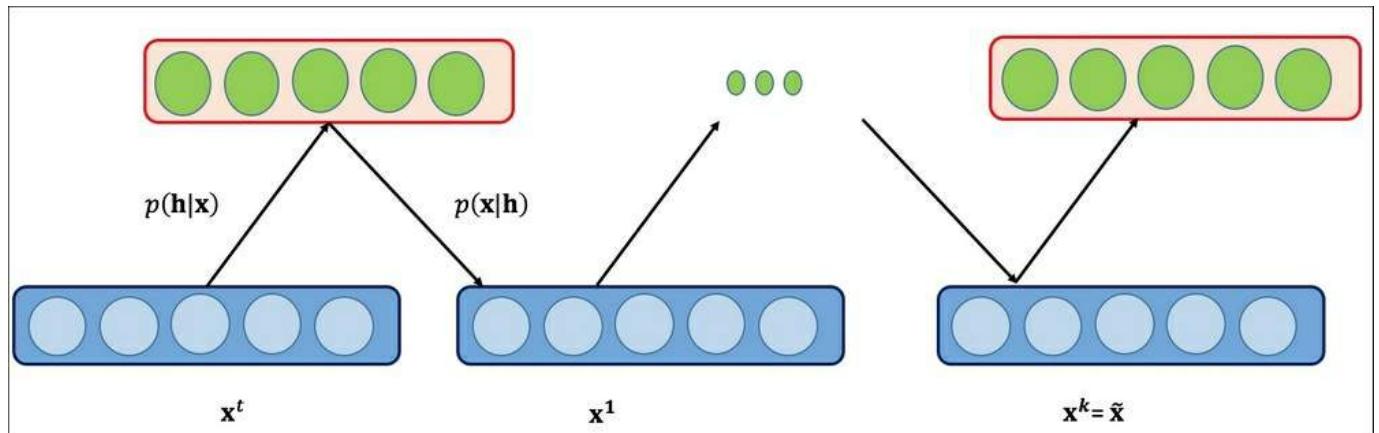


Figure 8: Graphical representation of sampling done between hidden and input layers.

As $t \rightarrow \infty$ it can be shown that the sampling represents the actual distribution $p(x, h)$.

Contrastive divergence

Contrastive divergence (CD) is a trick used to expedite the Gibbs sampling process described previously so it stops at step k of the process rather than continuing for a long time to guarantee convergence. It has been seen that even $k=1$ is reasonable and gives good performance (*References* [10]).

Inputs and outputs

These are the inputs to the algorithm:

- Training dataset
- Number of steps for Gibbs sampling, k
- Learning rate α
- The output is the set of updated parameters

How does it work?

The complete training pseudo-code using CD with the free energy function and partial derivatives can be given as:

1. For each instance in training x^t :
 1. Generate a negative particle \tilde{x} using k steps of Gibbs Sampling.
 2. Update the parameters:

$$W \leftarrow W + \alpha \left(h(x^t) x^{t^T} - h(\tilde{x}) \tilde{x}^T \right)$$

$$b \leftarrow b + \alpha \left(h(x^t) - h(\tilde{x}) \right)$$

$$c \leftarrow c + \alpha \left(x^t - \tilde{x} \right)$$

Persistent contrastive divergence

Persistent contrastive divergence is another trick used to compute the joint

probability $p(x, h)$. In this method, there is a single chain that does not reinitialize after every observed sample to find the negative particle \tilde{x} . It persists its state and parameters are updated just through running these k states by using the particle from the previous step.

Autoencoders

An autoencoder is another form of unsupervised learning technique in neural networks. It is very similar to the feed-forward neural network described at the start with the only difference being it doesn't generate a class at output, but tries to replicate the input at the output layer (*References [12 and 23]*). The goal is to have hidden layer(s) capture the latent or hidden information of the input as features that can be useful in unsupervised or supervised learning.

Definition and mathematical notations

A single hidden layer example of an Autoencoder is shown in the following figure:

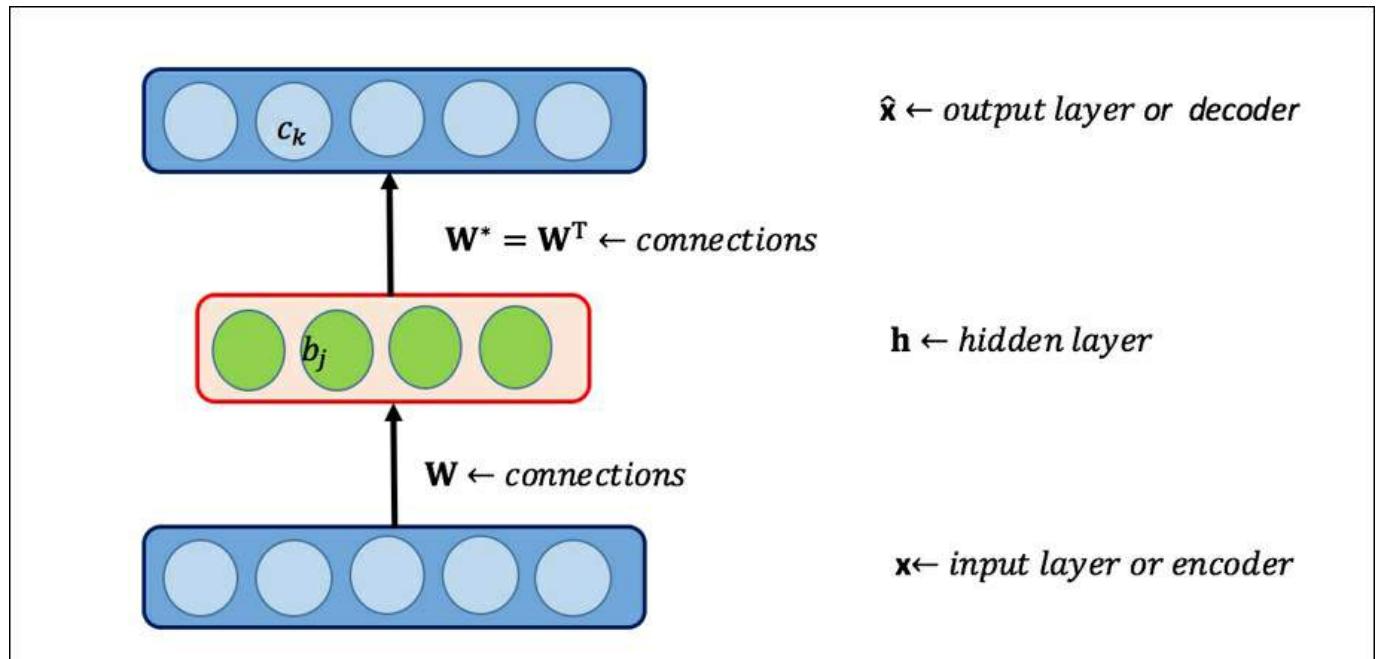


Figure 9: Autoencoder flow between layers

The input layer and the output layer have the same number of neurons similar as feed-forward, corresponding to the input vector, x . Each hidden layer can have greater, equal, or fewer neurons than the input or output layer and an activation

function that does a non-linear transformation of the signal. It can be seen as using the unsupervised or latent hidden structure to "compress" the data effectively.

The encoder or input transformation of the data by the hidden layer is given by:

$$\mathbf{h}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x}))$$

And the decoder or output transformation of the data by the output layer is given by:

$$\hat{\mathbf{x}} = o(\hat{\mathbf{a}}(\mathbf{x}))$$

Generally, a sigmoid function with linear transformation of signals as described in the neural network section is popularly used in the layers:

$$\mathbf{h}(\mathbf{x}) = \text{sigma}(\mathbf{b} + \mathbf{W}\mathbf{x}) \quad \text{and} \quad \hat{\mathbf{x}} = \text{sigma}(c + \mathbf{W} * \mathbf{h}(\mathbf{x}))$$

Loss function

The job of the loss function is to reduce the training error as before so that an optimization process such as a stochastic gradient function can be used.

In the case of binary valued input, the loss function is generally the average cross-entropy given by:

$$l(f(\mathbf{x})) = \sum_k (x_k \log \hat{x}_k + (1 - x_k) \log (1 - \hat{x}_k)) \quad \text{where } k = \text{dimension}$$

It can be easily verified that, when the input signal and output signal match either 0 or 1, the error is 0. Similarly, for real-valued input, a squared error is used:

$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (x_k - \hat{x}_k)^2$$

The gradient of the loss function that is needed for the stochastic gradient procedure is similar to the feed-forward neural network and can be shown through derivation for both real-valued and binary as follows:

$$\nabla_{\hat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x})) = \hat{\mathbf{x}}^{(t)} - \mathbf{x}^{(t)}$$

Parameter gradients are obtained by back-propagating the $\nabla_{\hat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x}))$ exactly as in the neural network.

Limitations of Autoencoders

Autoencoders have some known drawbacks that have been addressed by specialized architectures that we will discuss in the sections to follow. These limitations are:

When the size of the Autoencoder is equal to the number of neurons in the input, there is a chance that the weights learned by the Autoencoders are just the identity vectors and that the whole representation simply passes on the inputs exactly as outputs with zero loss. Thus, they emulate "rote learning" or "memorization" without any generalization.

When the size of the Autoencoder is greater than the number of neurons in the input, the configuration is called an "overcomplete" hidden layer and can have similar problems to the ones mentioned previously. Some of the units can be turned off and others can become identity making it just the copy unit.

When the size of the Autoencoder is less than the number of neurons in the input, known as "undercomplete", the latent structure in the data or important hidden components can be discovered.

Denoising Autoencoder

As mentioned previously, when the Autoencoder has a hidden layer size greater than or equal to that of the input, it is not guaranteed to learn the weights and can become simply a unit switch to copy input to output. This issue is addressed by the Denoising Autoencoder. Here there is another layer added between input and the hidden layer. This layer adds some noise to the input using either a well-known

distribution $p(\tilde{\mathbf{x}}|\mathbf{x})$ or using stochastic noise such as turning a bit to 0 in binary input. This "noisy" input then goes through learning from the hidden layer to the output layer exactly like the Autoencoder. The loss function of the Denoising Autoencoder compares the output with the actual input. Thus, the added noise and the larger hidden layer enable either learning latent structures or adding/removing redundancy to produce the exact signal at the output. This architecture—where non-zero features at the noisy layer generate features at the hidden layer that are themselves transformed by the activation layer as the signal advances forward—lends a robustness and implicit structure to the learning process (*References [15]*).

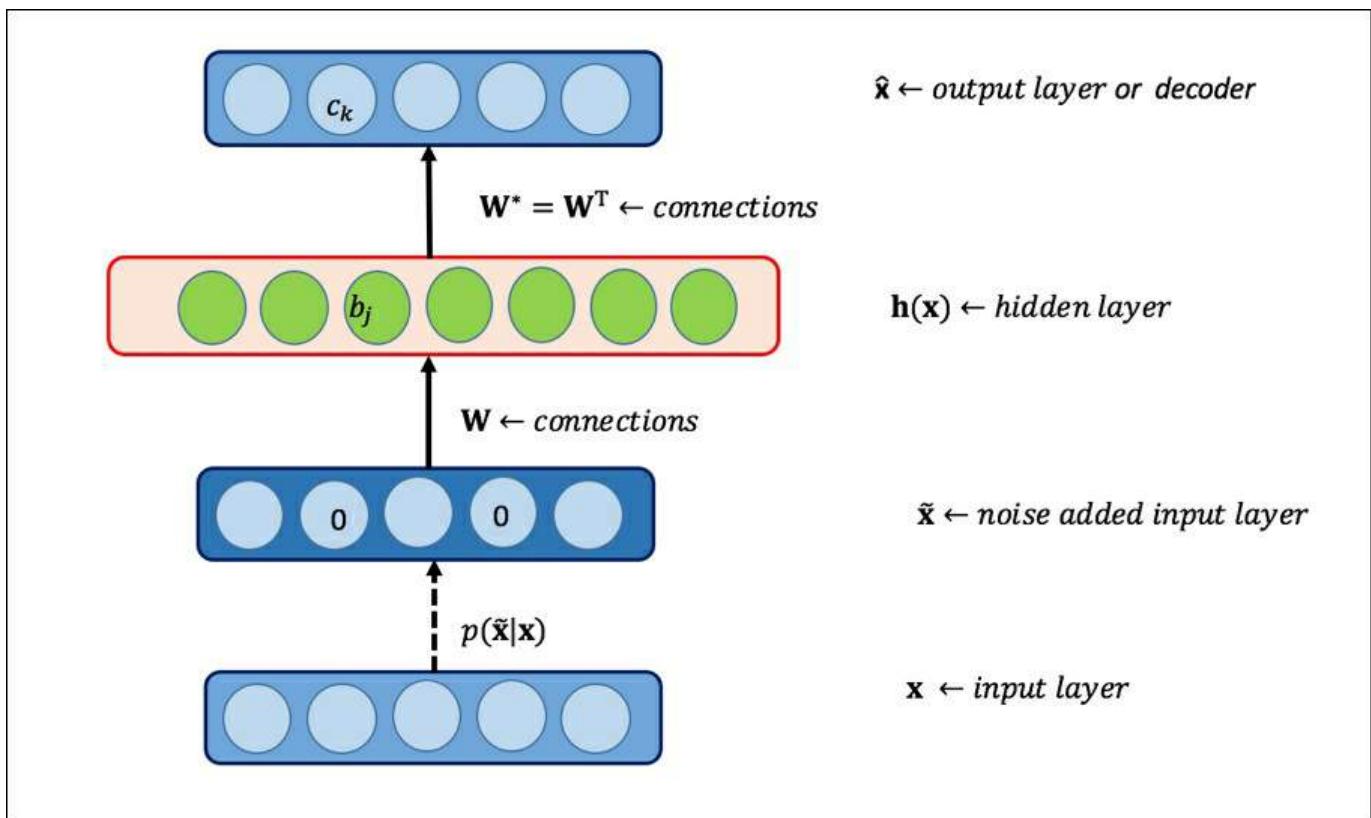


Figure 10: Denoising Autoencoder

Unsupervised pre-training and supervised fine-tuning

As we discussed in the issues section on neural networks, the issue with over-training arises especially in deep learning as the number of layers, and hence parameters, is large. One way to account for over-fitting is to do data-specific regularization. In this section, we will describe the "unsupervised pre-training" method done in the hidden layers to overcome the issue of over-fitting. Note that this is generally the "initialization process" used in many deep learning algorithms.

The algorithm of unsupervised pre-training works in a layer-wise greedy fashion. As shown in the following figure, one layer of a visible and hidden structure is considered at a given time. The weights of this layer are learned for a few iterations using unsupervised techniques such as RBM, described previously. The output of the hidden layer is then used as a "visible" or "input" layer and the training proceeds to the next, and so on.

Each learning of layers can be thought of as a "feature extraction or feature generation" process. The real data inputs when transformed form higher-level features at a given layer and then are further combined to form much higher-level features, and so on.

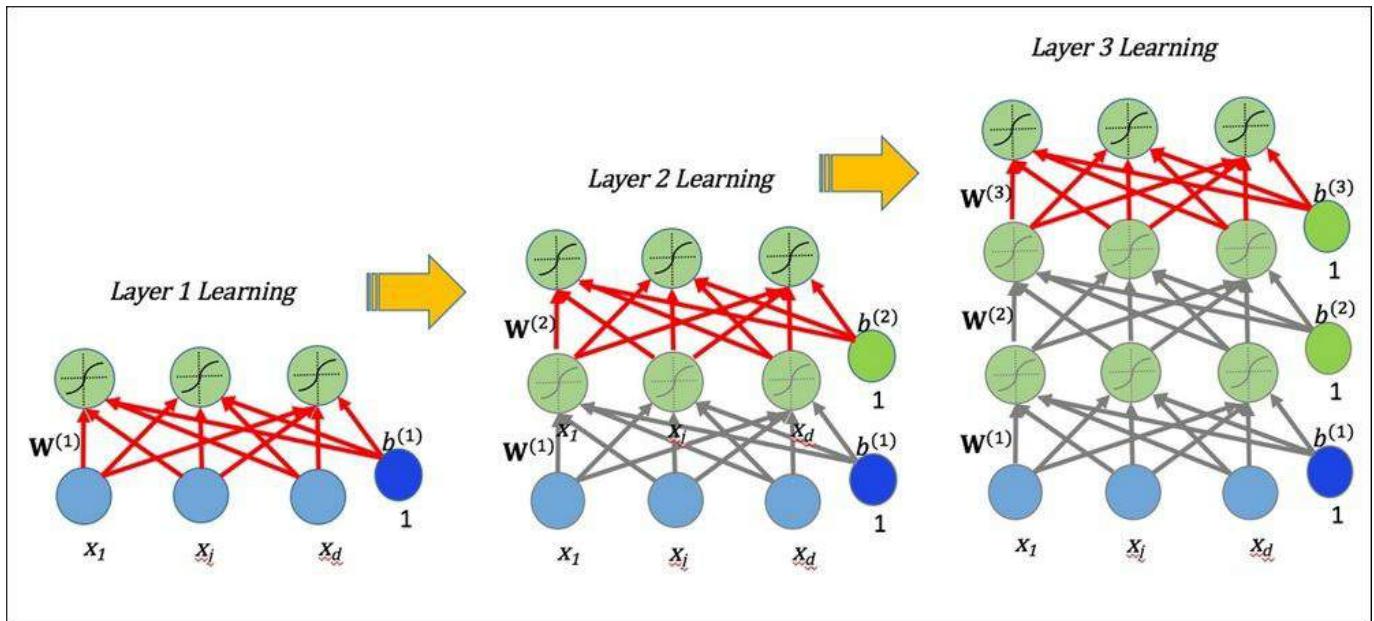


Figure 11: Layer wise incremental learning through unsupervised learning.

Once all the hidden layer parameters are learned in pre-training using unsupervised techniques as described previously, a supervised fine-tuning process follows. In the supervised fine-tuning process, a final output layer is added and, just like in a neural network, training is done with forward and backward propagation. The idea is that most weights or parameters are almost fully tuned and only need a small change for producing a discriminative class mapping at the output.

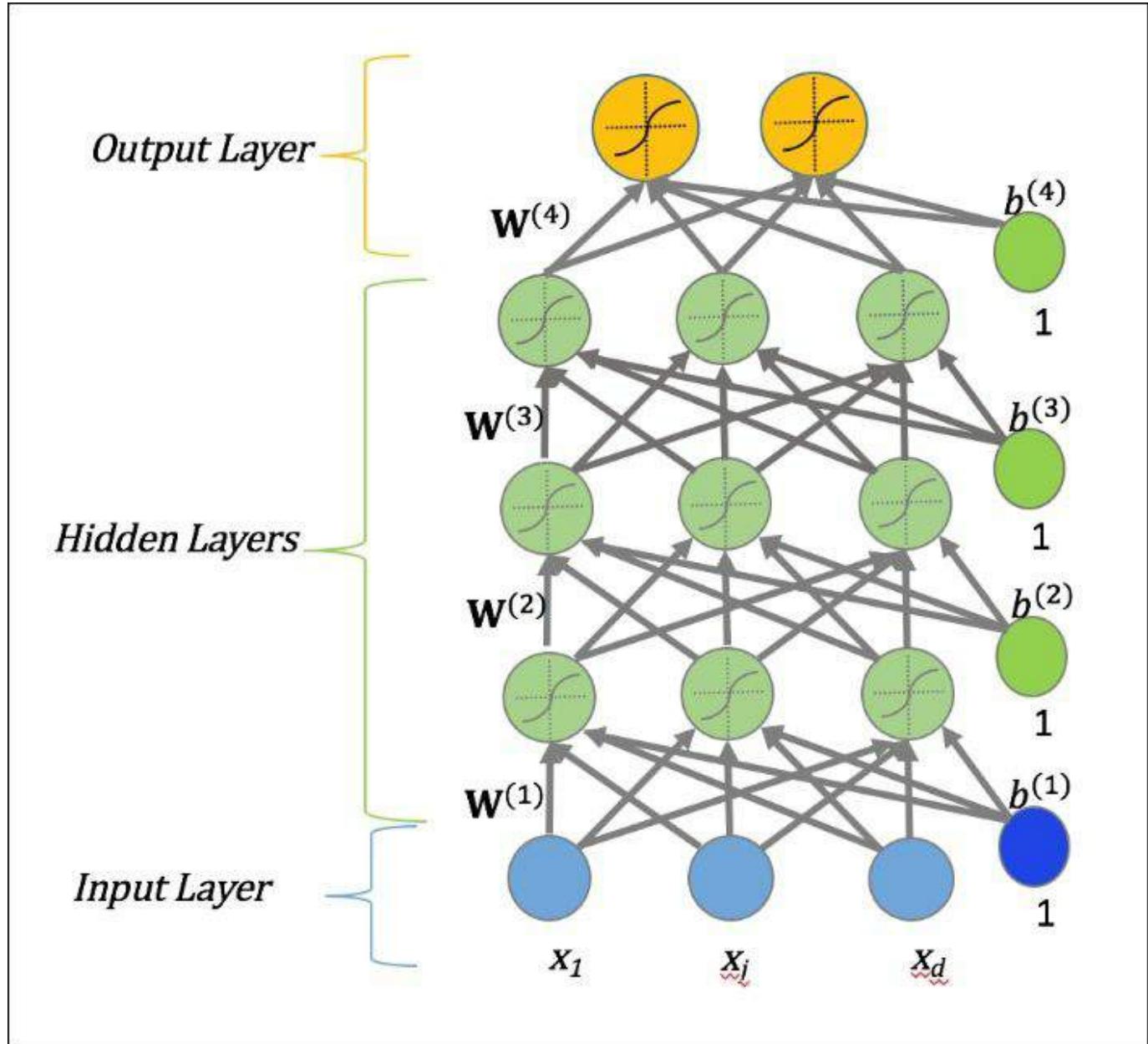


Figure 12: Final tuning or supervised learning.

Deep feed-forward NN

A deep feed-forward neural network involves using the stages pre-training, and fine-tuning.

Depending on the unsupervised learning technique used—RBM, Autoencoders, or Denoising Autoencoders—different algorithms are formed: Stacked RBM, Stacked Autoencoders, and Stacked Denoising Autoencoders, respectively.

Input and outputs

Given an architecture for the deep feed-forward neural net, these are the inputs for training the network:

- Number of layers L
- Dataset without labels D
- Dataset with labels D
- Number of training iterations n

How does it work?

The generalized learning/training algorithm for all three is given as follows:

1. For layers $l=1$ to L (Pre-Training):

$$\mathcal{D} = \left\{ \mathbf{h}^{l-1}(\mathbf{x}^{(t)}) \right\}_{t=1}^T$$

1. Dataset without Labels

2. Perform Step-wise Layer Unsupervised Learning (RBM, Autoencoders, or Denoising Autoencoders)

3. Finalize the parameters $\mathbf{W}^l, \mathbf{b}^l$ from the preceding step

2. For the output layer ($L+1$) perform random initialization of parameters $\mathbf{W}^{L+1}, \mathbf{b}^{L+1}$.

3. For layers $l=1$ to $L+1$ (Fine-Tuning):

$$\mathcal{D} = \left\{ \mathbf{h}^{l-1}(\mathbf{x}^{(t)}), y^{(t)} \right\}_{t=1}^T$$

1. Dataset with Labels

2. Use the pre-initialized weights from 1. ($\mathbf{W}^l, \mathbf{b}^l$).

3. Perform forward-backpropagation for n iterations.

Deep Autoencoders

Deep Autoencoders have many layers of hidden units, which shrink to a very small dimension and then symmetrically grow to the input size.

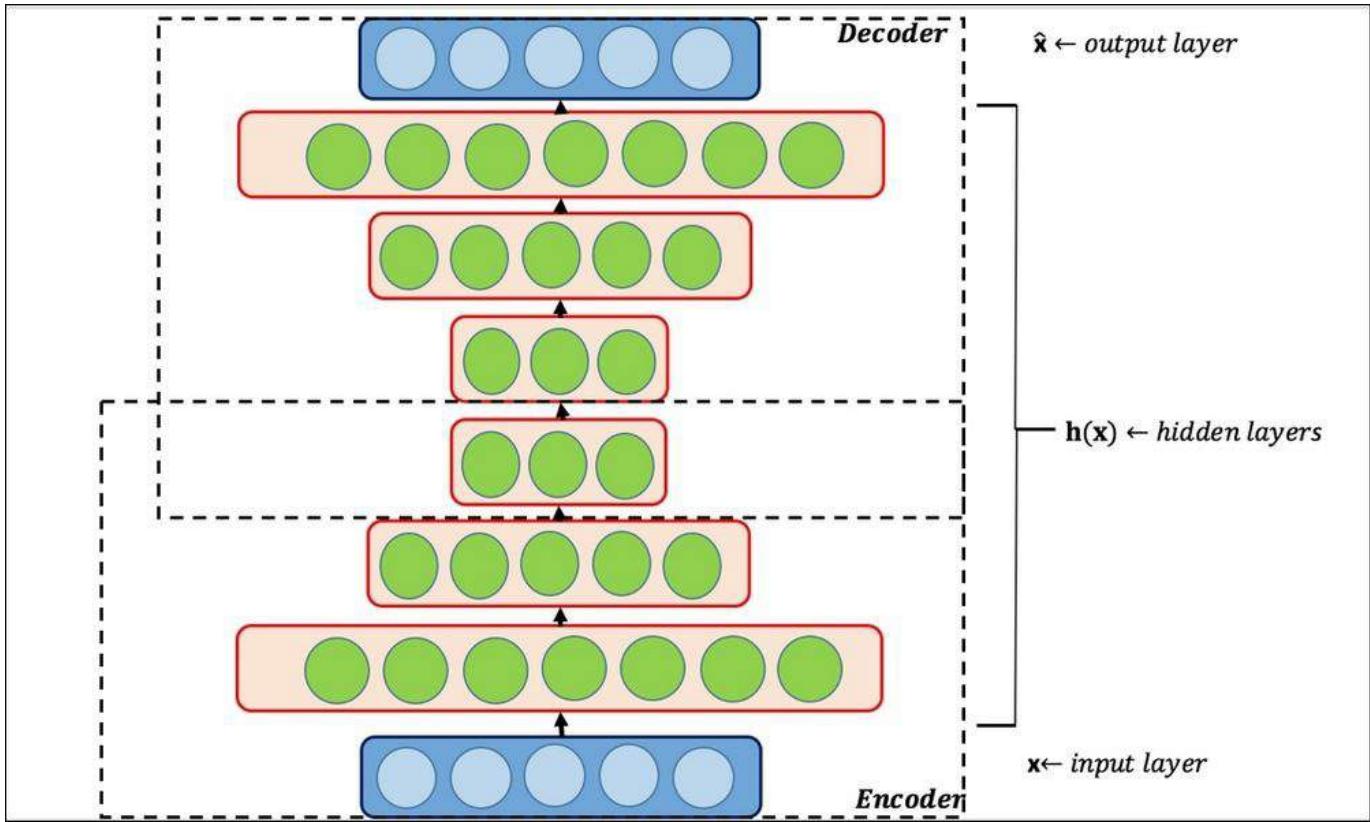


Figure 13: Deep Autoencoders

The idea behind Deep Autoencoders is to create features that capture latent complex structures of input using deep networks and at the same time overcome the issue of gradients and underfitting due to the deep structure. It was shown that this methodology generated better features and performed better than PCA on many datasets (*References [13]*).

Deep Autoencoders use the concept of pre-training, encoders/decoders, and fine-tuning to perform unsupervised learning:

In the pre-training phase, the RBM methodology is used to learn greedy stepwise parameters of the encoders, as shown in the following figure, for initialization:

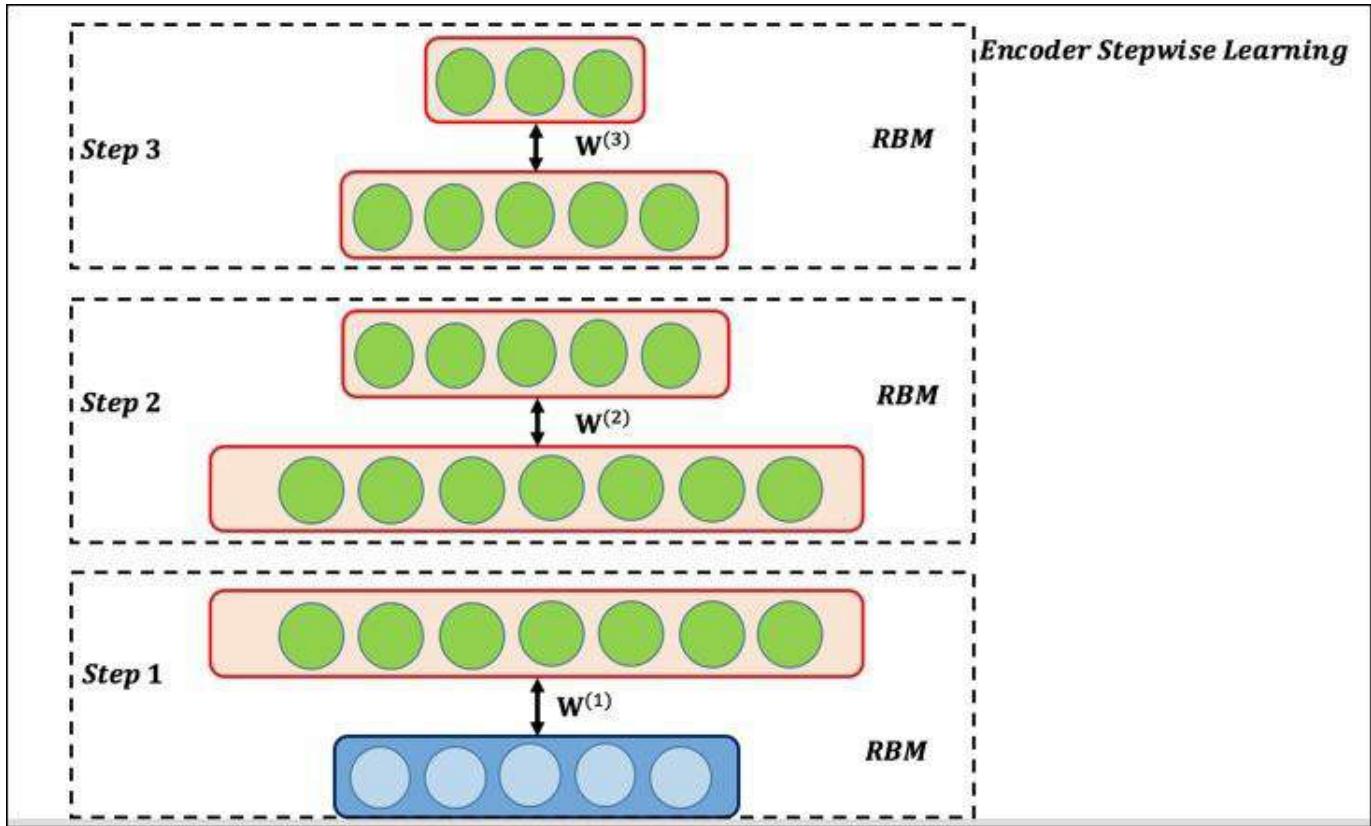


Figure 14: Stepwise learning in RBM

In the unfolding phase the same parameters are symmetrically applied to the decoder network for initialization.

Finally, fine-tuning backpropagation is used to adjust the parameters across the entire network.

Deep Belief Networks

Deep Belief Networks (DBNs) are the origin of the concept of unsupervised pre-training (*References [9]*). Unsupervised pre-training originated from DBNs and then was found to be equally useful and effective in the feed-forward supervised deep networks.

Deep belief networks are not supervised feed-forward networks, but a generative model to generate data samples.

Inputs and outputs

The input layer is the instance of data, represented by one neuron for each input feature. The output of a DBN is a reconstruction of the input from a hierarchy of learned features of increasingly greater abstraction.

How does it work?

How a DBN learns the joint distribution of the input data is explained here using a three-layer DBN architecture as an example.

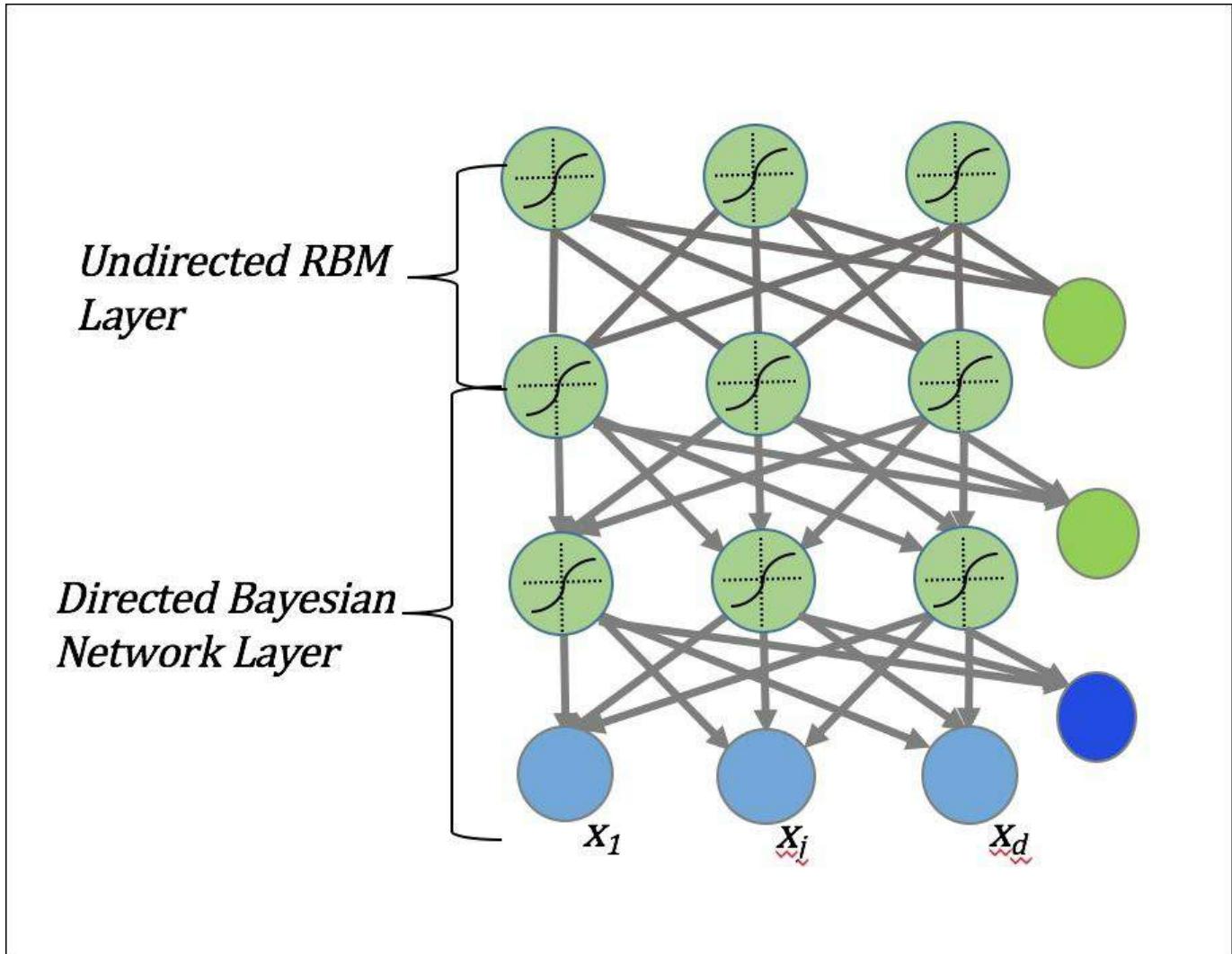


Figure 15: Deep belief network

The three-hidden-layered DBN as shown has a first layer of undirected RBM connected to a two-layered Bayesian network. The Bayesian network with a sigmoid activation function is called a sigmoid Bayesian network (SBN).

The goal of a generative model is to learn the joint distribution as given by
 $p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)})$

$$p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) p(\mathbf{x} | \mathbf{h}^{(1)})$$

RBM computation as seen before gives us:

$$p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \exp\left(\mathbf{h}^{(2)} \mathbf{W}^3 \mathbf{h}^{(2)} + \mathbf{b}^{2^T} \mathbf{h}^{(2)} + \mathbf{b}^{3^T} \mathbf{h}^{(3)}\right) / Z$$

The Bayesian Network in the next two layers is:

$$p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) = \exp \prod_j p(h_j^{(1)} | \mathbf{h}^{(2)})$$

$$p(\mathbf{x} | \mathbf{h}^{(1)}) = \prod_i p(x_i | \mathbf{h}^{(1)})$$

For binary data:

$$p(h_j^{(1)} = 1 | \mathbf{h}^{(2)}) = \text{sigm}\left(\mathbf{b}^{(2)} + \mathbf{W}^{(2)^T} \mathbf{h}^{(2)}\right)$$

$$p(h_j^{(1)} = 1 | \mathbf{h}^{(2)}) = \text{sigm}\left(\mathbf{b}^{(2)} + \mathbf{W}^{(2)^T} \mathbf{h}^{(2)}\right)$$

Deep learning with dropouts

Another technique used to overcome the "overfitting" issues mentioned in deep neural networks is using the dropout technique to learn the parameters. In the next sections, we will define, illustrate, and explain how deep learning with dropouts works.

Definition and mathematical notation

The idea behind dropouts is to "cripple" the deep neural network structure by stochastically removing some of the hidden units as shown in the following figure after the parameters are learned. The units are set to 0 with the dropout probability generally set as $p=0.5$

The idea is similar to adding noise to the input, but done in all the hidden layers. When certain features (or a combination of features) are removed stochastically, the neural network has to learn latent features in a more robust way, without the interdependence of some features.

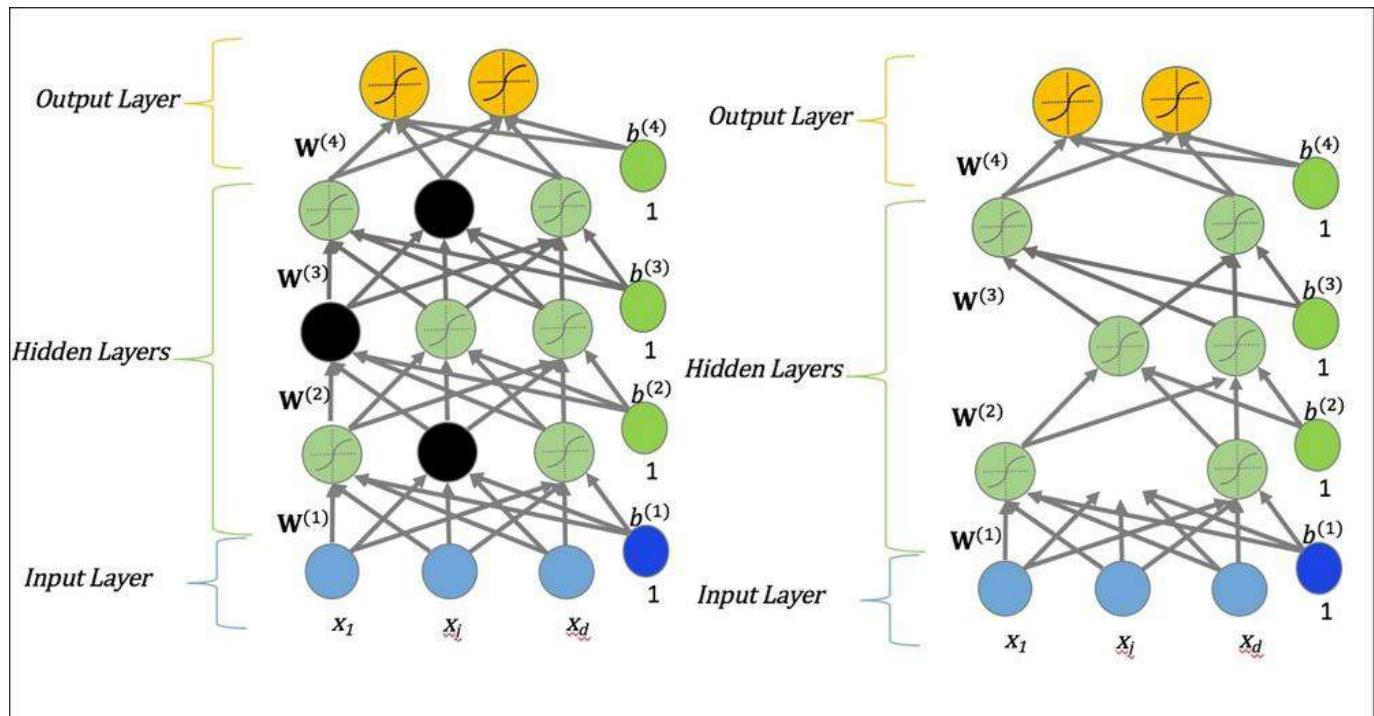


Figure 16: Deep learning with dropout indicated by dropping certain units with dark shading.

Each hidden layer is represented by $h^k(x)$ where k is the layer. The pre-activation for layer $0 < k < l$ is given by:

$$\mathbf{a}^k(x) = \mathbf{b}^k(x) + \mathbf{W}^k \mathbf{h}^{k-1}(x)$$

The hidden layer activation for $l < k < l$. Binary masks are represented by \mathbf{m}^k at each hidden layer:

$$\mathbf{h}^k(x) = \mathbf{g}(\mathbf{a}^k(x)) \odot \mathbf{m}^k$$

The final output layer activation is:

$$\mathbf{h}^{l+1}(x) = \mathbf{o}(\mathbf{a}^{l+1}(x))$$

Inputs and outputs

For training with dropouts, inputs are:

- Network architecture
- Training dataset
- Dropout probability p (typically 0.5)

The output is a trained deep neural net that can be applied for predictive use.

How does it work?

We will now describe the different parts of how deep learning with dropouts works.

Learning Training and testing with dropouts

The backward propagation learning of weights and biases from the output loss function using gradients is very similar to traditional neural network learning. The

only difference is that masks are applied appropriately as follows:

Compute the output gradient before activation:

$$= \nabla_{a^{l+1}(\mathbf{x})} - \log(f(\mathbf{x})_y) = -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

For hidden layers $k=l+1$ to 1 :

Compute the gradient of hidden layer parameters:

$$\nabla_{\mathbf{W}^k} - \log(f(\mathbf{x})_y) = (\nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y)) \mathbf{h}^{k-1}(\mathbf{x})^T$$

$$\nabla_{b^k} - \log(f(\mathbf{x})_y) = \nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y)$$

\mathbf{h}^{k-1} computation has taken into account the binary mask \mathbf{m}^{k-1} applied.

Compute the gradient of the hidden layer below the current:

$$\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) = \mathbf{W}^k (\nabla_{a^k(\mathbf{x})} - \log(f(\mathbf{x})_y))$$

Compute the gradient of the layer below before activation:

$$\begin{aligned} & \left(\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) \right)^T \nabla_{a^{k-1}(\mathbf{x})} \mathbf{h}^k(\mathbf{x}) \\ &= \left(\nabla_{\mathbf{h}^{k-1}(\mathbf{x})} - \log(f(\mathbf{x})_y) \right) \odot [.., g'(a^{k-1}\mathbf{x}_j) ...] \odot \mathbf{m}^{k-1} \end{aligned}$$

When testing the model, we cannot use the binary mask as it is stochastic; the "expectation" value of the mask is used. If the dropout probability is $p=0.5$, the same value 0.5 is used as the expectation for the unit at test or model application time.

Sparse coding

Sparse coding is another neural network used for unsupervised learning and feature generation (*References* [22]). It works on the principle of finding latent structures in high dimensions that capture the patterns, thus performing feature extraction in addition to unsupervised learning.

Formally, for every input $\mathbf{x}^{(t)}$ a latent representation $\mathbf{h}^{(t)}$ is learned, which has a sparse representation (most values are 0 in the vector). This is done by optimization using the following objective function:

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^T \min_{\mathbf{h}^{(t)}} \left(\frac{1}{2} \left\| \mathbf{x}^{(t)} - \mathbf{D}\mathbf{h}^{(t)} \right\|_2^2 + \lambda \left\| \mathbf{h}^{(t)} \right\|_1 \right)$$

$$\frac{1}{2} \left\| \mathbf{x}^{(t)} - \mathbf{D}\mathbf{h}^{(t)} \right\|_2^2$$

Where the first term $\frac{1}{2} \left\| \mathbf{x}^{(t)} - \mathbf{D}\mathbf{h}^{(t)} \right\|_2^2$ is to control the reconstruction error and the second term, which uses a regularizer λ , is for sparsity control. The matrix \mathbf{D} is also known as a Dictionary as it has equivalence to words in a dictionary and $\mathbf{h}^{(t)}$ is similar to word frequency; together they capture the impact of words in extracting patterns when performing text mining.

Convolutional Neural Network

Convolutional Neural Networks or CNNs have become prominent and are widely used in the computer vision domain. Computer vision involves processing images/videos for capturing knowledge and patterns. Annotating images, classifying images/videos, correcting them, story-telling or describing images, and so on, are some of the broad applications in computer vision [16].

Computer vision problems most generally have to deal with unstructured data that can be described as:

Inputs that are 2D images with single or multiple color channels or 3D videos that are high-dimensional vectors.

The features in these 2D or 3D representations have a well-known spatial topology, a hierarchical structure, and some repetitive elements that can be exploited.

The images/videos have a large number of transformations or variants based on factors such as illumination, noise, and so on. The same person or car can look different based on several factors.

Next, we will describe some building blocks used in CNNs. We will use simple images such as the letter X of the alphabet to explain the concept and mathematics involved. For example, even though the same character X is represented in different ways in the following figure due to translation, scaling, or distortion, the human eye can easily read it as X, but it becomes tricky for the computer to see the pattern. The images are shown with the author's permission (*References* [19]):

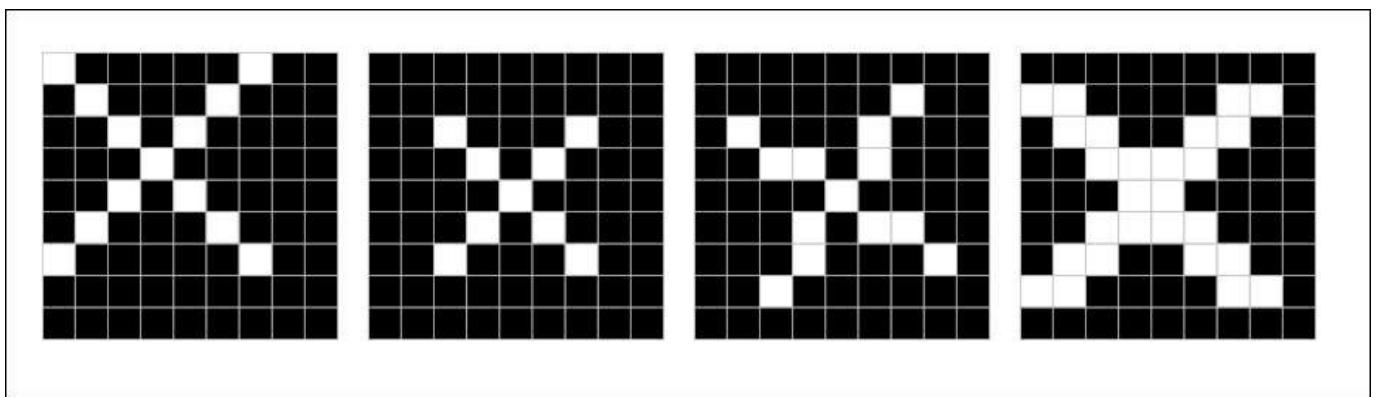


Figure 17: Image of character X represented in different ways.

The following figure illustrates how a simple grayscale image of X has common features such as a diagonal from top left, a diagonal from top right, and left and right intersecting diagonals repeated and combined to form a larger X:

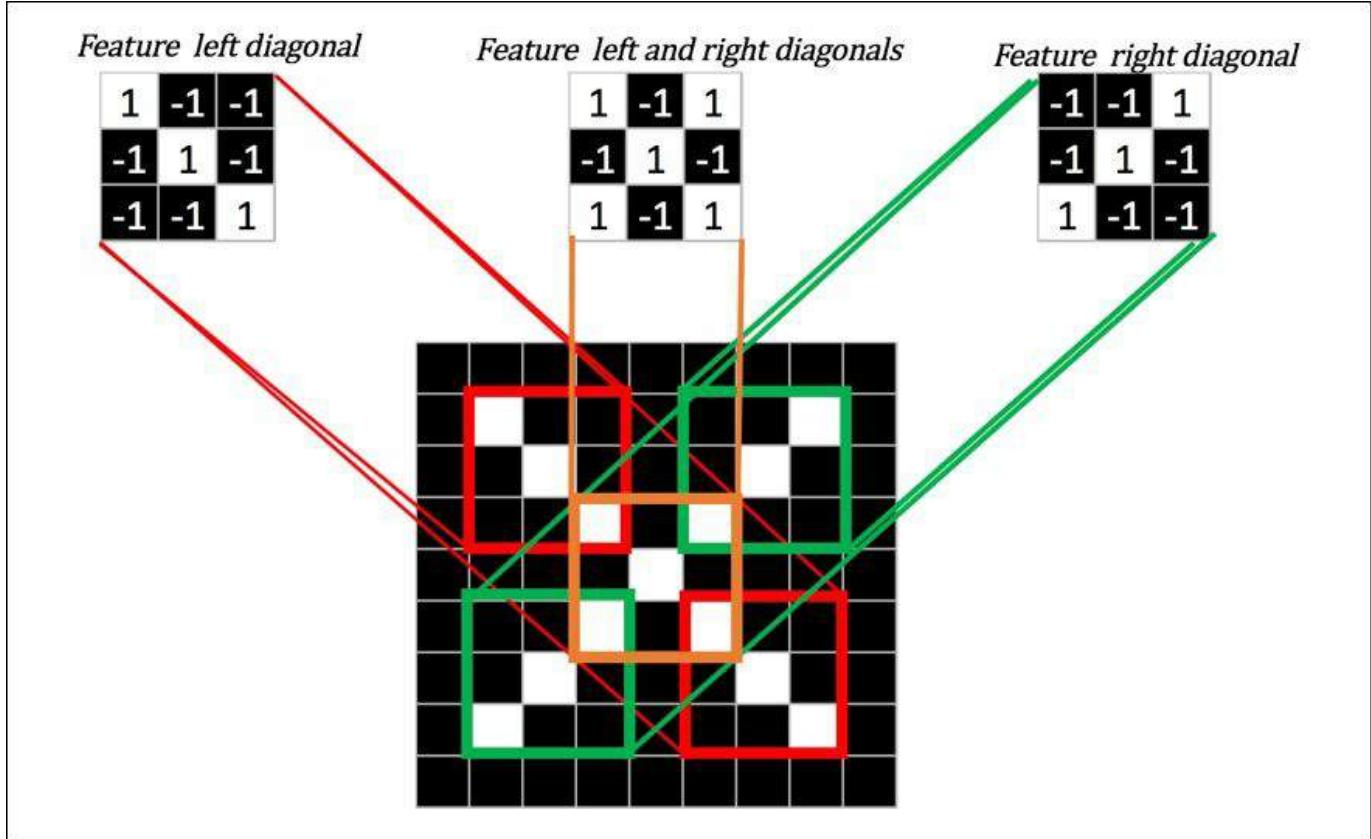


Figure 18: Common features represented in the image of character X.

Local connectivity

This is the simple concept of dividing the whole image into "patches" or "recipient fields" and giving each patch to the hidden layers. As shown in the figure, instead of 9 X 9 pixels of the complete sample image, a 3 X 3 patch of pixels from the top left goes to the first hidden unit, the overlapping second patch goes to second, and so on.

Since the fully connected hidden layer would have a huge number of parameters, having smaller patches completely reduces the parameter or high-dimensional space problem!

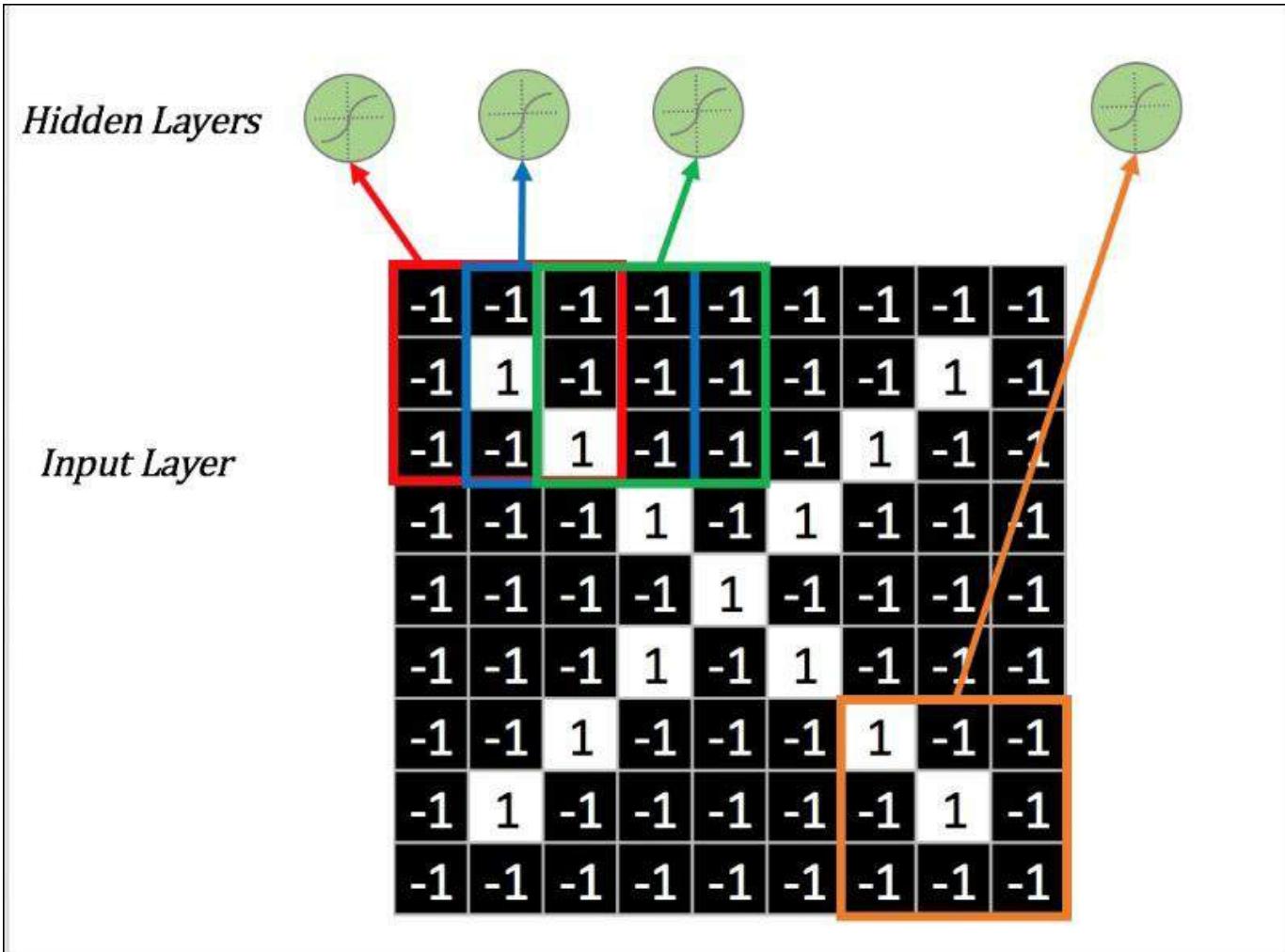
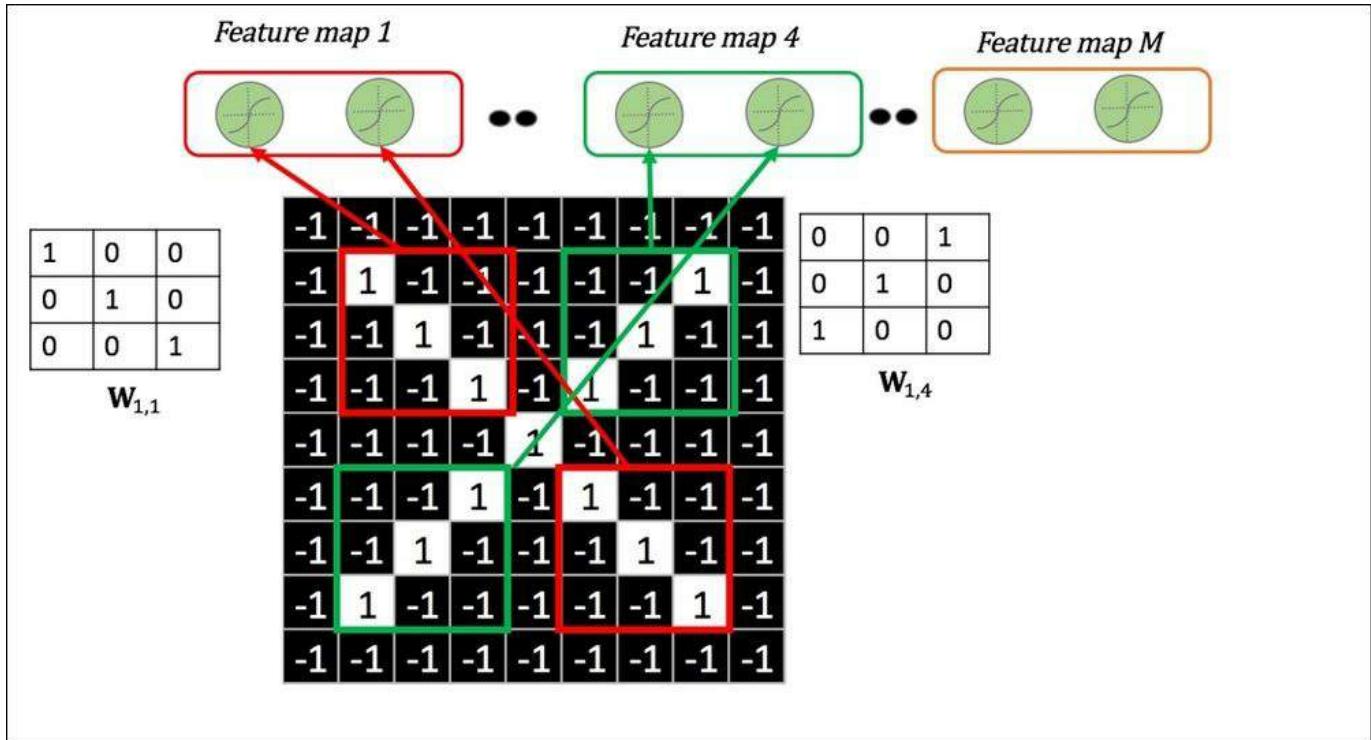


Figure 19: Concept of patches on the whole image.

Parameter sharing

The concept of parameter sharing is to construct a weight matrix that can be reused over different patches or recipient fields as constructed in the preceding figure in the local sharing. As shown in the following figure, the Feature map with same parameters $\mathbf{W}_{1,1}$ and $\mathbf{W}_{1,4}$ creates two different feature maps, Feature Map 1 and 4, both capturing the same features, that is, diagonal edges on either side. Thus, feature maps capture "similar regions" in the images and further reduce the dimensionality of the input space.



Discrete convolution

We will explain the steps in discrete convolution, taking a simple contrived example with simplified mathematics to illustrate the operation.

Suppose the kernel representing the diagonal feature is scanned over the entire image as a patch of 3×3 . If this kernel lands on the self-same feature in the input image and we have to compute the center value through what we call the convolution operator, we get the exact value of 1 because of the matching as shown:

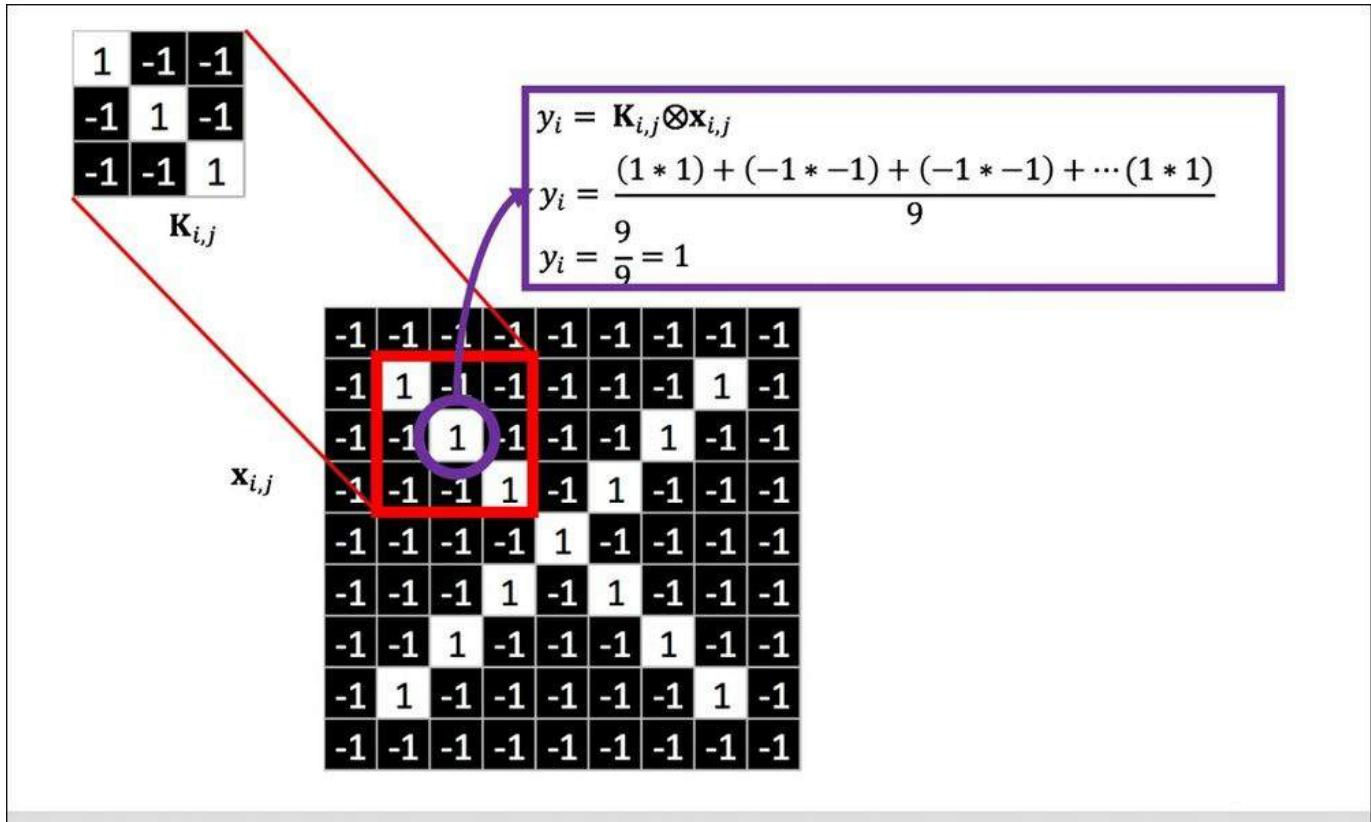


Figure 21: Discrete convolution step.

The entire image when run through this kernel and convolution operator gives a matrix of values as follows:

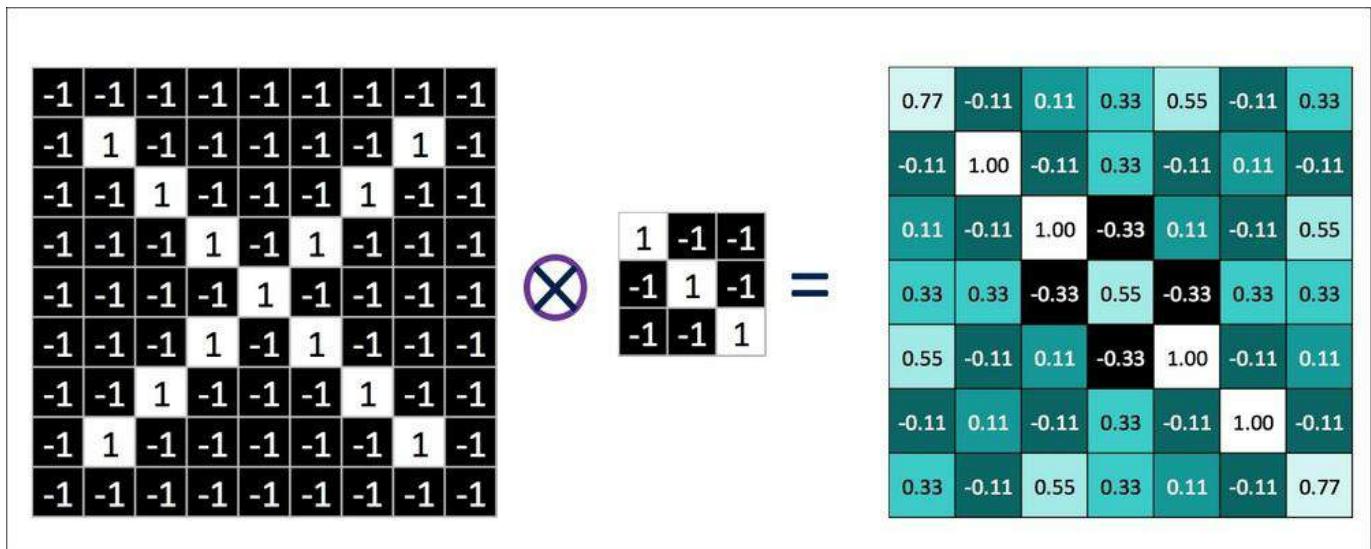


Figure 22: Transformation of the character image after a kernel and convolution operator.

We can see how the left diagonal feature gets highlighted by running this scan. Similarly, by running other kernels, as shown in the following figure, we can get a "stack of filtered images":

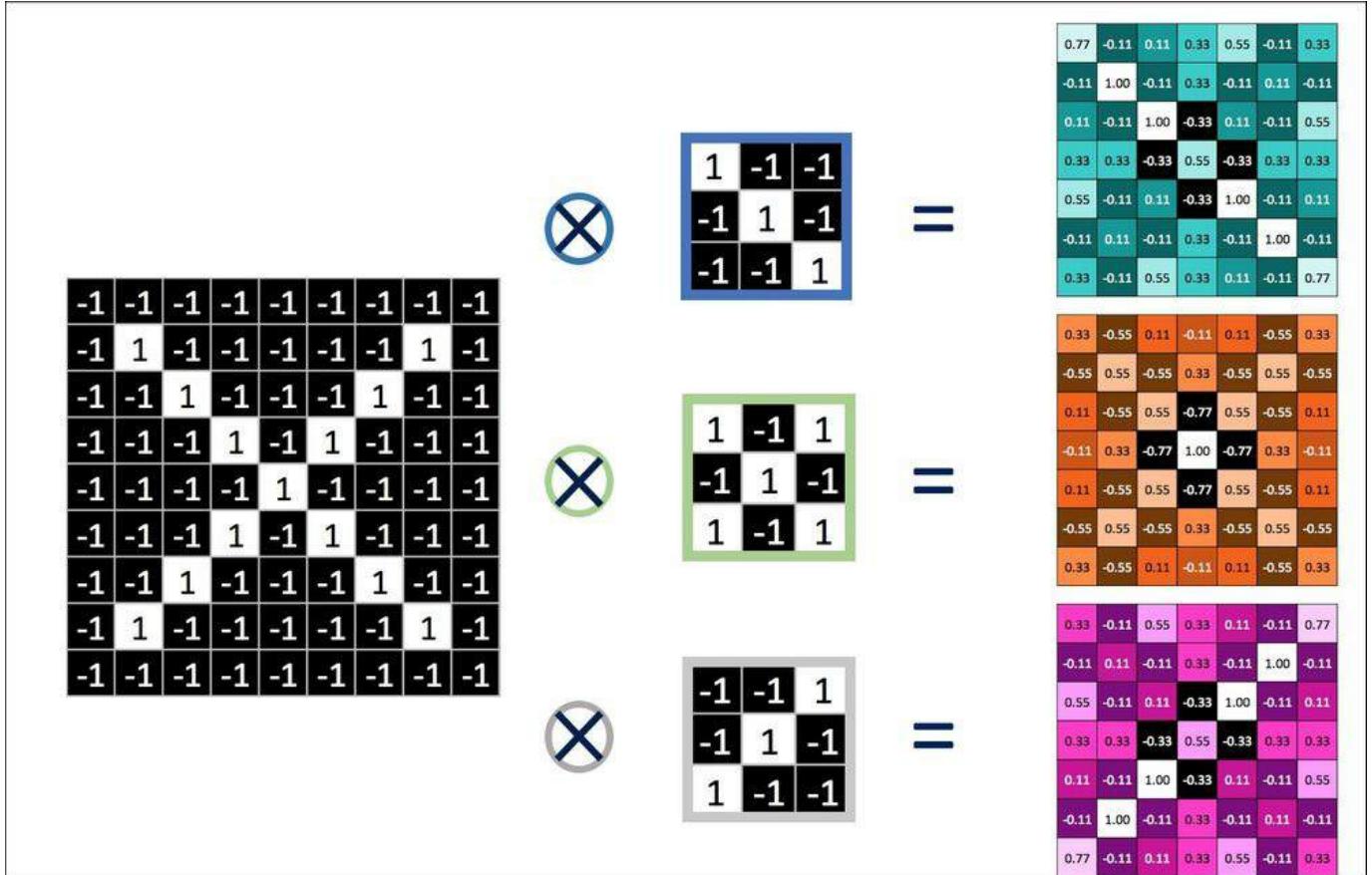


Figure 23: Different features run through the kernel giving a stack of images.

Each cell in the filtered images can be given as:

$$y_j = g_j \tanh\left(\sum_i \mathbf{K}_{i,j} \mathbf{x}_i\right)$$

$\mathbf{K}_{i,j}$ = convolution kernel

\mathbf{x}_i = feature map or patch g_j = scaling factor

Pooling or subsampling

Pooling or subsampling works on the stack of filtered images to further shrink the image or compress it, while keeping the pattern as-is. The main steps carried out in pooling are:

1. Pick a window size (for example, 2 X 2) and a stride size (for example, 2).
2. Move the window over all the filtered images at stride.
3. At each window, pick the "maximum" value.

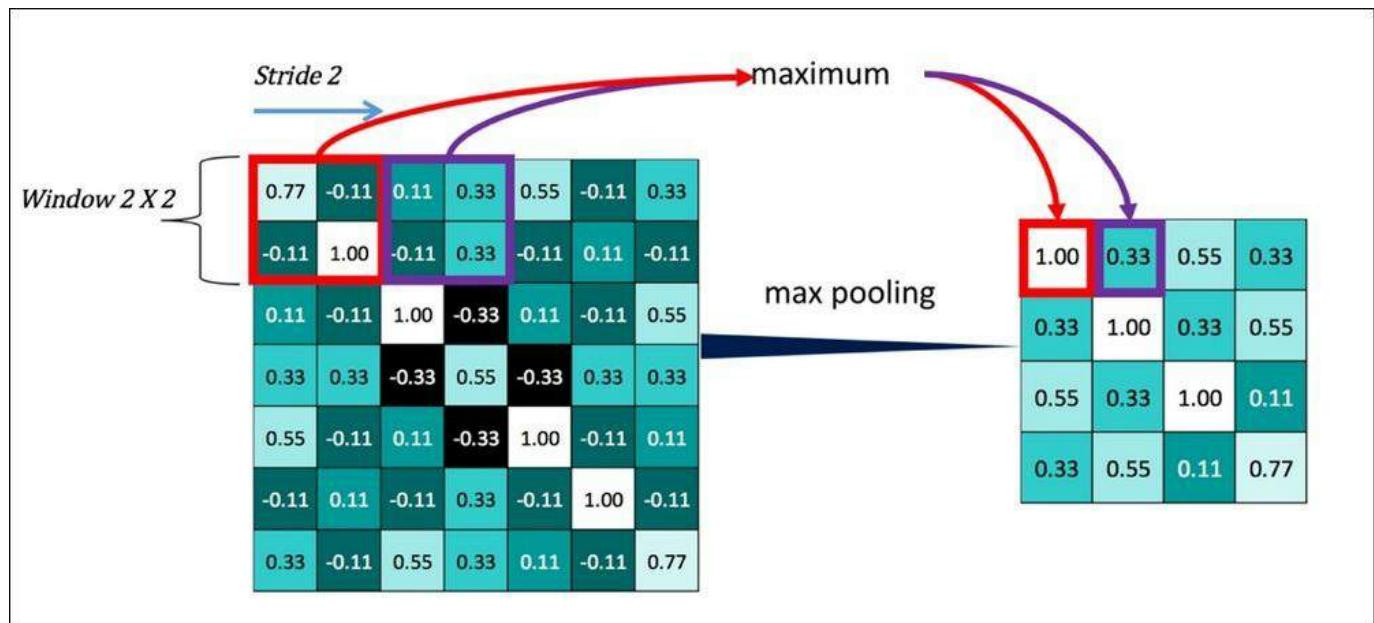


Figure 24: Max pooling, done using a window size of 2 X 2 and stride of 2, computes cell values with maximum for first as 1.0, 0.33 for next, and so on.

Pooling also plays an important part where the same features if moved or scaled can still be detected due to the use of maximum. The same set of stacked filtered images gets transformed into pooled images as follows:

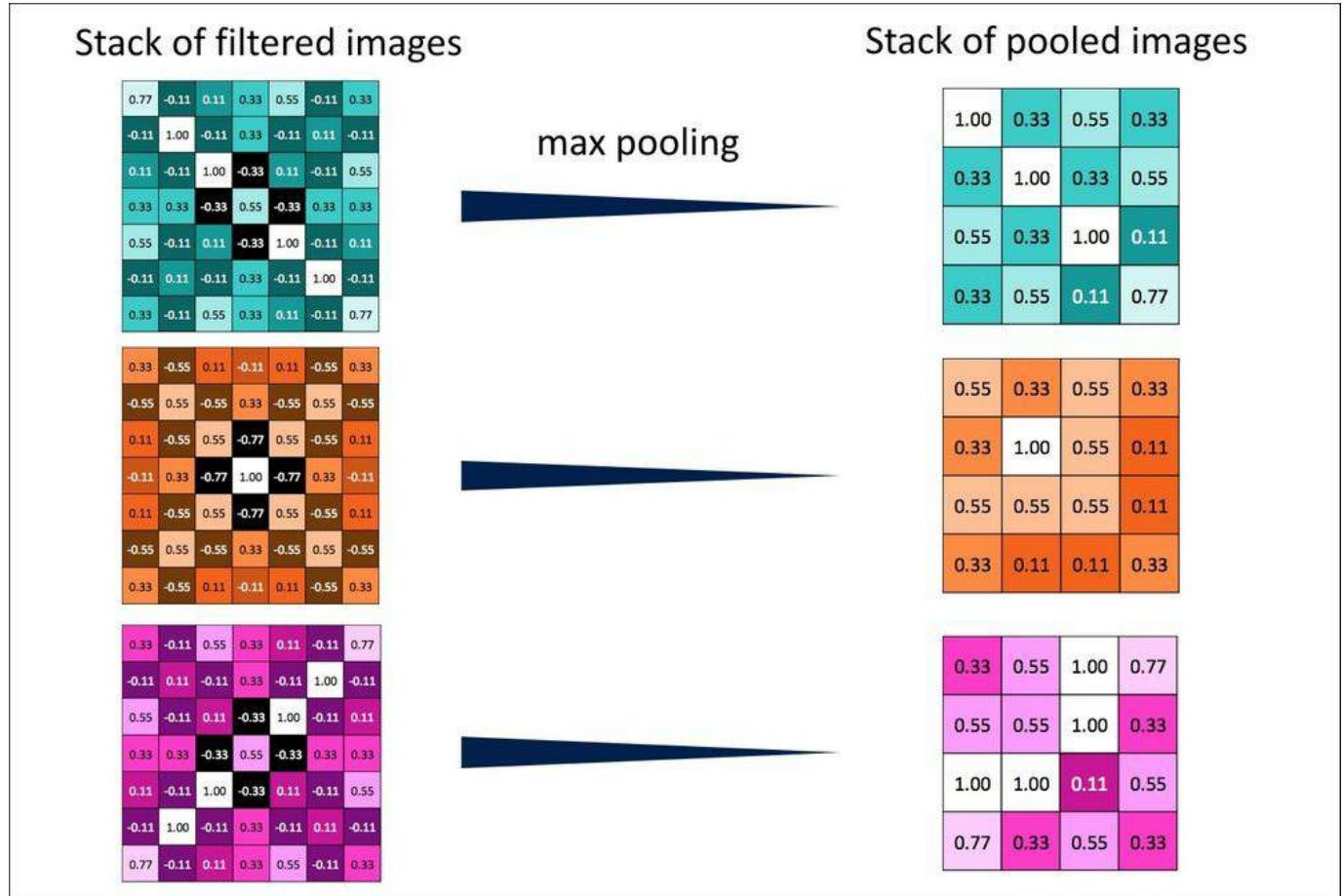


Figure 25: Transformation showing how a stack of filtered images is converted to pooled images.

Normalization using ReLU

As we discussed in the building blocks of deep learning, ReLUs remove the negative by squashing it to 0 and keep the positives as-is. They also play an important role in gradient computation in the backpropagation, removing the vanishing gradient issue of vanishing gradient.

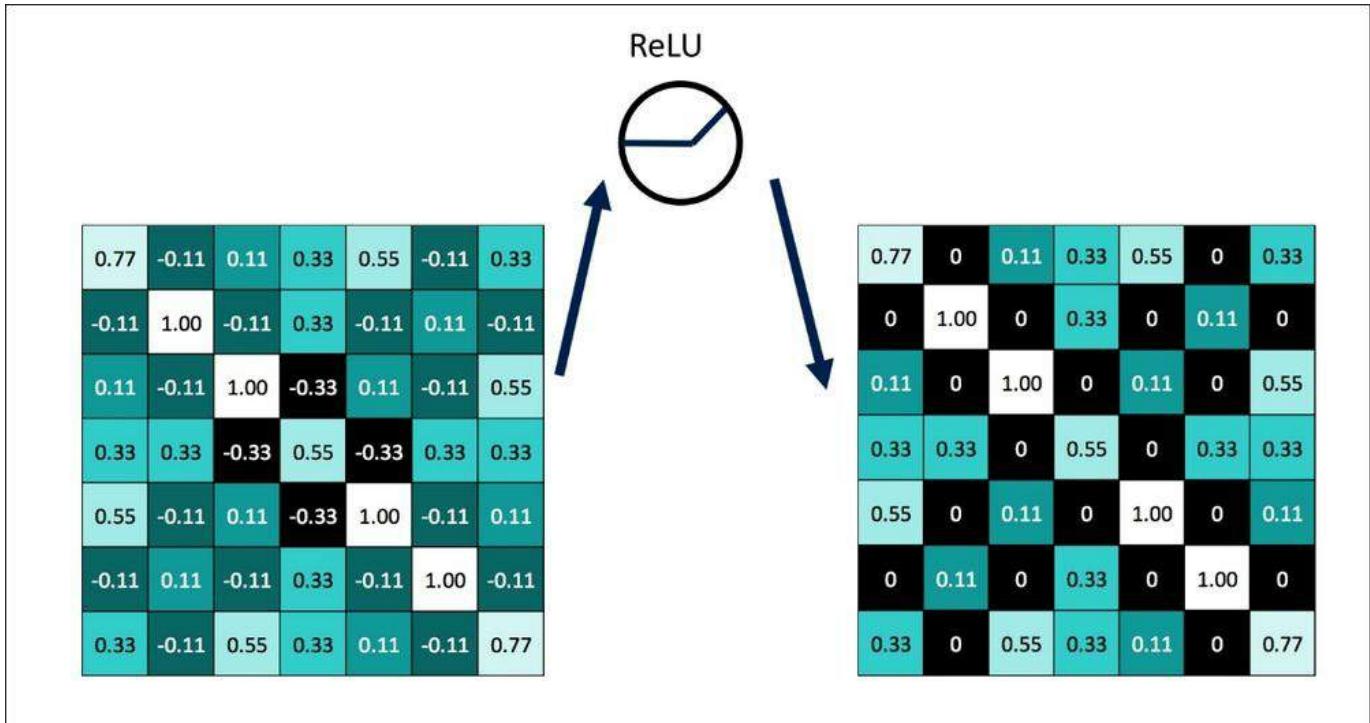


Figure 26: Transformation using ReLU.

CNN Layers

In this section, we will put together the building blocks discussed earlier to form the complete picture of CNNs. Combining the layers of convolution, ReLU, and pooling to form a connected network yielding shrunken images with patterns captured in the final output, we obtain the next composite building block, as shown in the following figure:

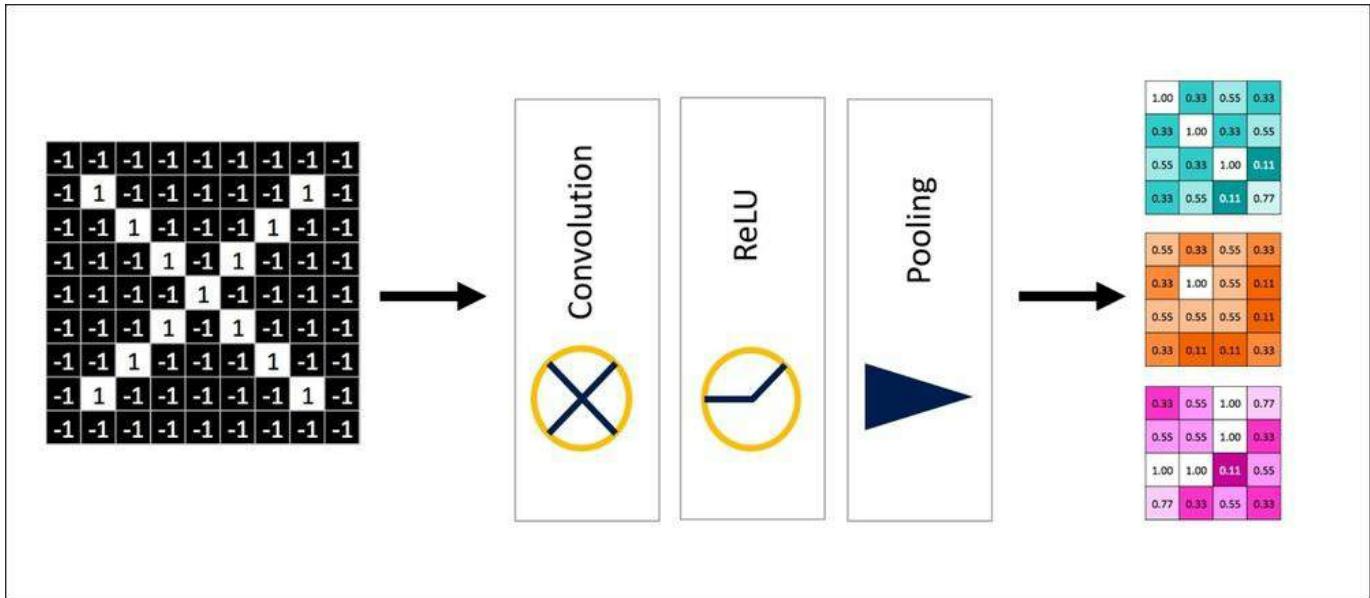


Figure 27: Basic Unit of CNN showing a combination of Convolution, ReLU, and Pooling.

Thus, these layers can be combined or "deep-stacked", as shown in the following figure, to form a complex network that gives a small pool of images as output:



Figure 28: Deep-stacking the basic units repeatedly to form CNN layers.

The output layer is a fully connected network as shown, which uses a voting technique and learns the weights for the desired output. The fully connected output

layer can be stacked too.

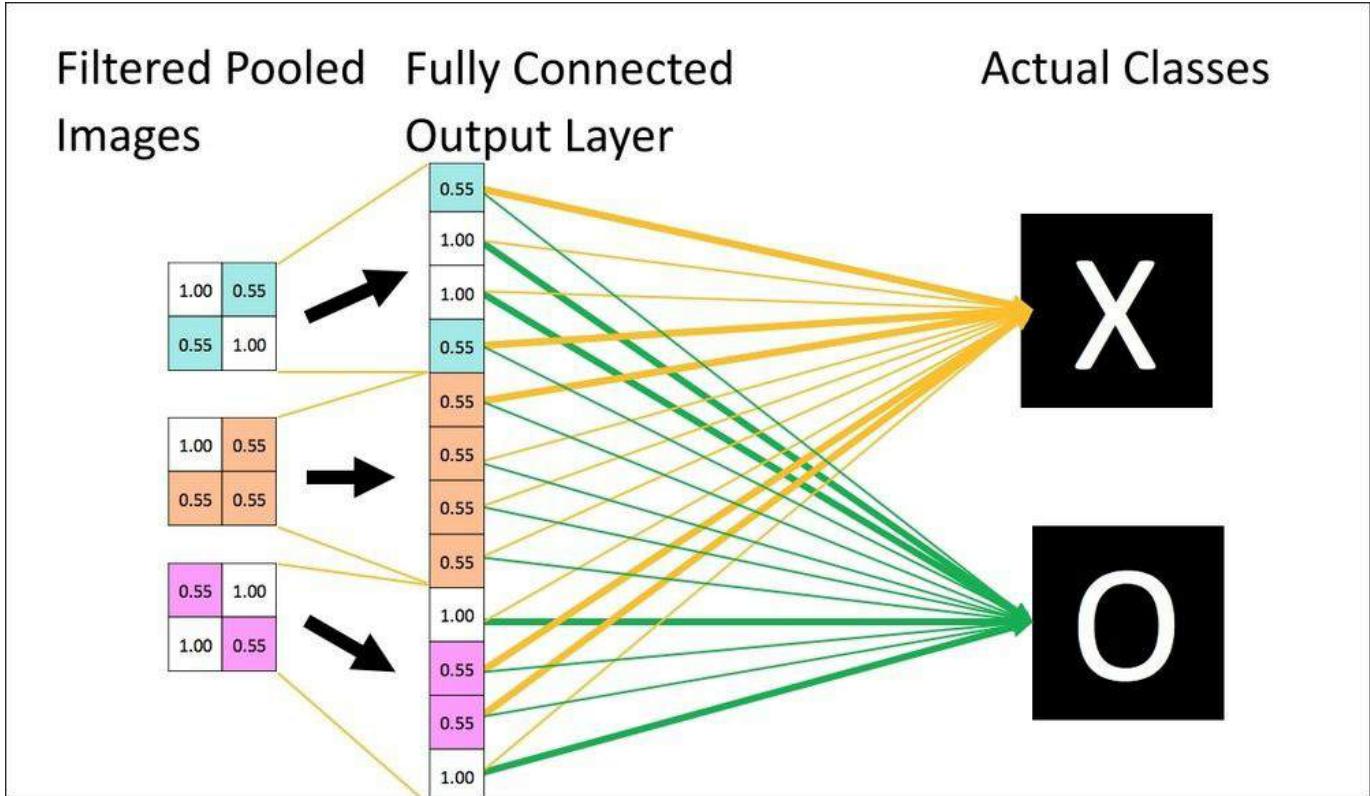


Figure 29: Fully connected layer as output of CNN.

Thus, the final CNNs can be completely illustrated as follows:

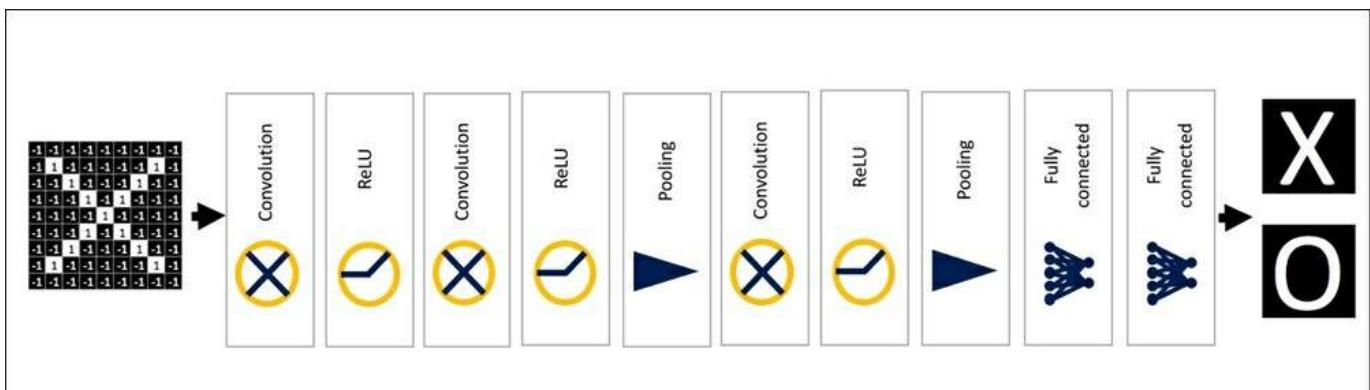


Figure 30: CNNs with all layers showing inputs and outputs.

As before, gradient descent is selected as the learning technique using the loss functions to compute the difference and propagate the error backwards.

CNN's can be used in other domains such as voice pattern recognition, text mining, and so on, if the mapping of the data to the "image" can be successfully done and "local spatial" patterns exist. The following figure shows one of the ways of mapping sound and text to images for CNN usage:

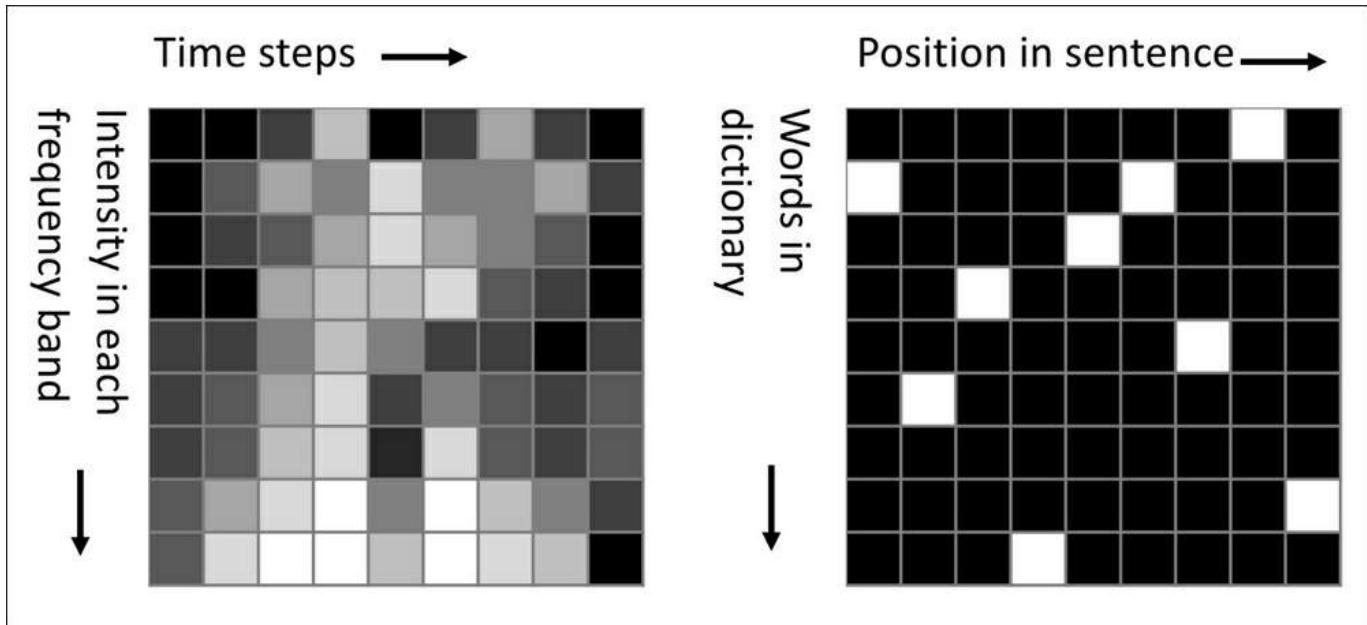


Figure 31: Illustration of mapping between temporal data, such as voice to spatial data, to an image.

Recurrent Neural Networks

Normal deep networks are used when you have finite inputs and there is no interdependence between the input examples or instances. When there are variable length inputs and there are temporal dependencies between them, that is, sequence related data, neural networks must be modified to handle such data. Recurrent Neural Networks (RNN) are examples of neural networks that are used widely to solve such problems, and we will discuss them in the following sections. RNNs are used in many sequence-related problems such as text mining, language modeling, bioinformatics data modeling, and so on, to name a few areas that fit this meta-level description (*References [18 and 21]*).

Structure of Recurrent Neural Networks

We will describe the simplest unit of the RNN first and then show how it is combined to understand it functionally and mathematically and illustrate how different components interact and work.

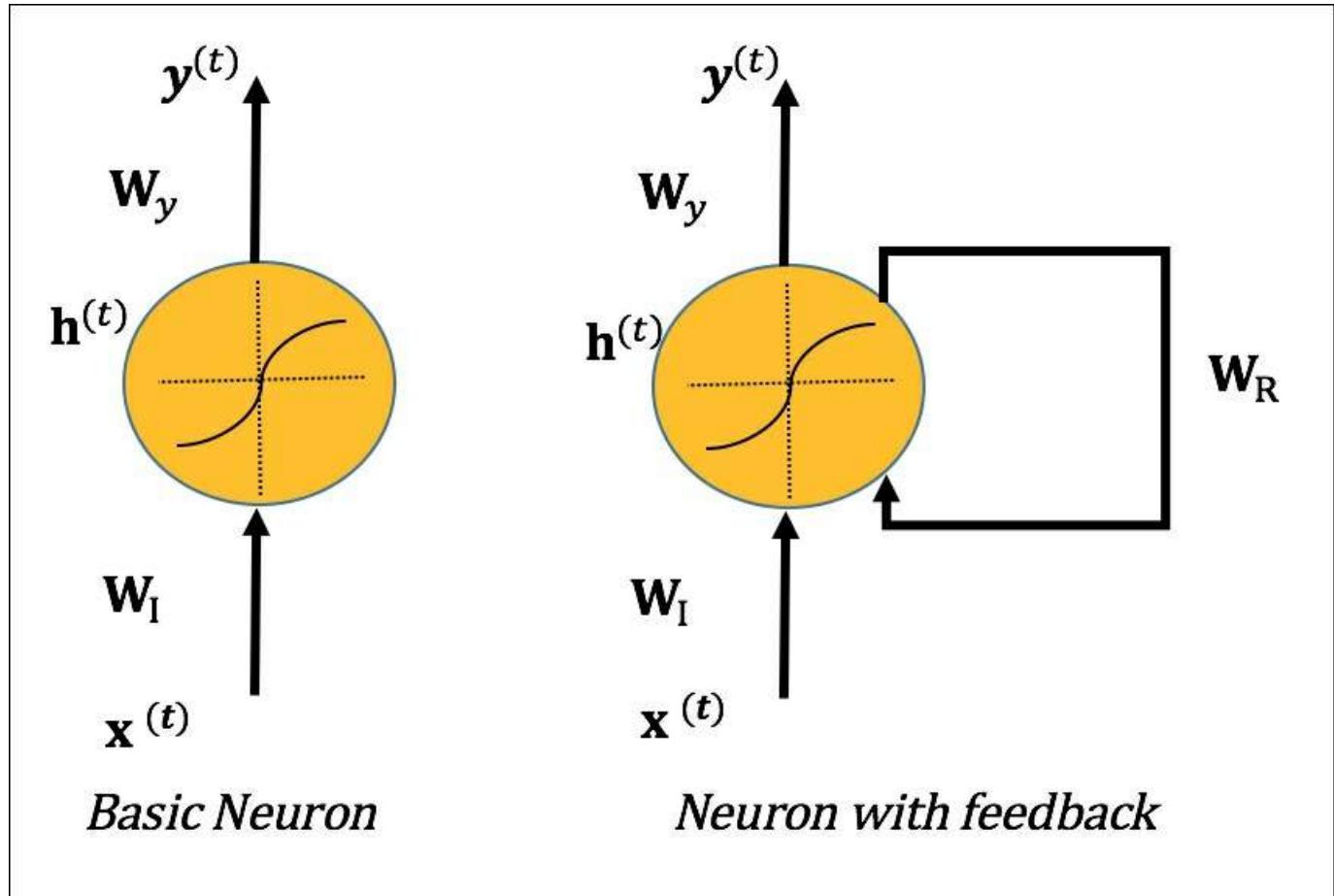


Figure 32: Difference between an artificial neuron and a neuron with feedback.

Let's consider the basic input, a neuron with activation, and its output at a given time t :

$$\mathbf{h}^{(t)} = g_h \left(\mathbf{W}_I \mathbf{x}^{(t)} + b_h \right)$$

$$\mathbf{y}^{(t)} = g_y \left(\mathbf{W}_y \mathbf{h}^{(t)} + b_y \right)$$

A neuron with feedback keeps a matrix \mathbf{W}_R to incorporate previous output at time $t-1$ and the equations are:

$$\mathbf{h}^{(t)} = g_h \left(\mathbf{W}_I \mathbf{x}^{(t)} + \mathbf{W}_R \mathbf{h}^{(t-1)} + b_h \right)$$

$$\mathbf{y}^{(t)} = g_y \left(\mathbf{W}_y \mathbf{h}^{(t)} + b_y \right)$$

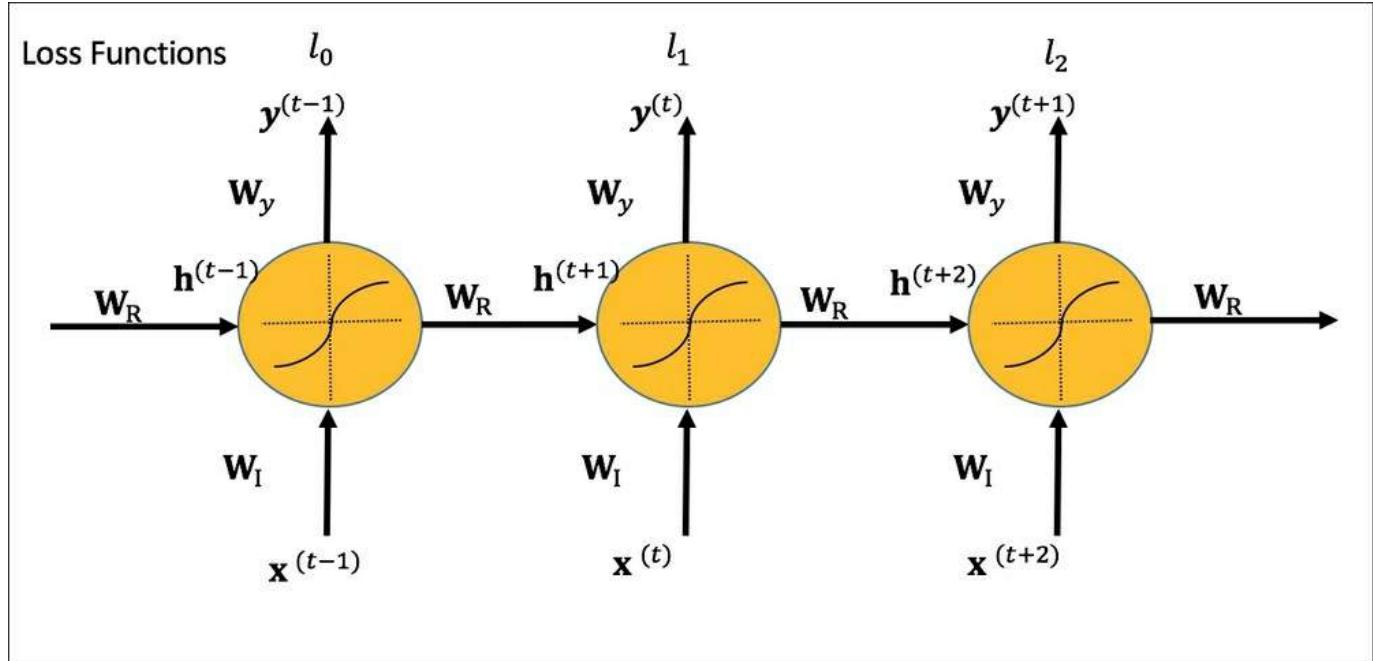


Figure 33: Chain of neurons with feedbacks connected together.

The basic RNN stacks the structure of hidden units as shown with feedback connected from the previous layer. At activation at time t , it depends not only on $\mathbf{x}^{(t)}$ as input, but also on the previous unit given by $\mathbf{W}_R \mathbf{h}^{(t-1)}$. The weights in the feedback connection of RNN are generally the same across all the units, \mathbf{W}_R . Also, instead of emitting output at the very end of the feed-forward neural network, each unit continuously emits an output that can be used in the loss function calculation.

Learning and associated problems in RNNs

Working with RNNs presents some challenges that are specific to them but there are common problems that are also encountered in other types of neural net.

1. The gradient used from the output loss function at any time t of the unit has dependency going back to the first unit or $t=0$, as shown in the following figure. This is because the partial derivative at the unit is dependent on the previous unit, since:

$$\mathbf{h}^{(t)} = g_h \left(\mathbf{W}_I \mathbf{x}^{(t)} + \mathbf{W}_R \mathbf{h}^{(t-1)} + b_h \right)$$

Backpropagation through time (BPTT) is the term used to illustrate the process.

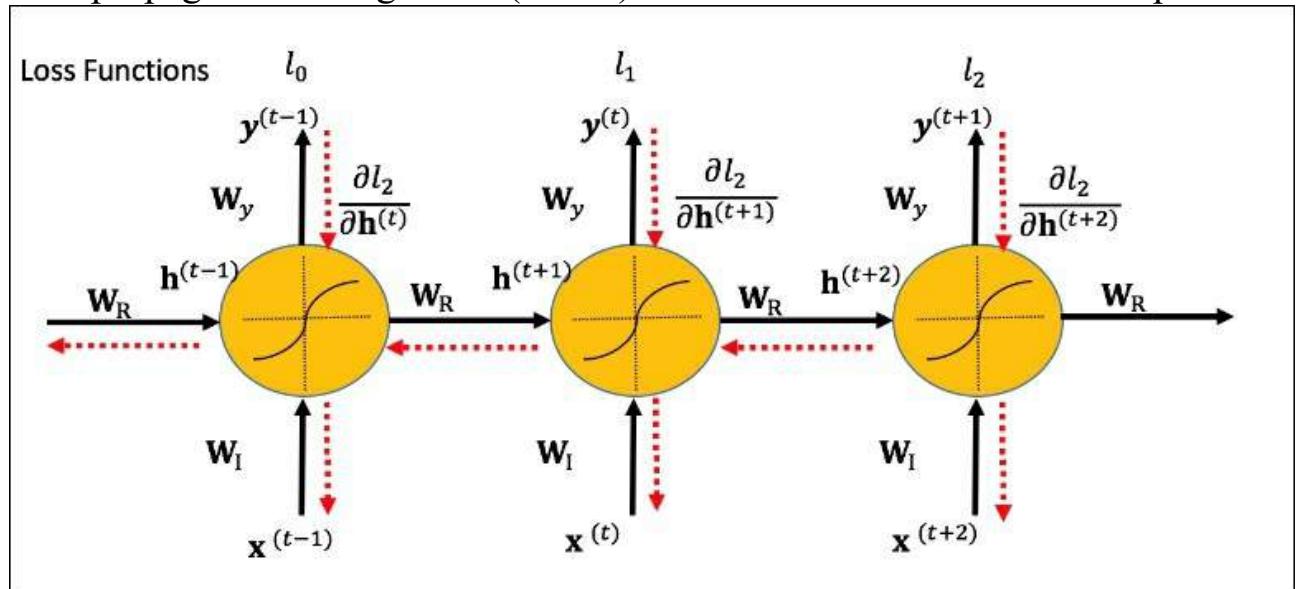


Figure 34: Backpropagation through time.

2. Similar to what we saw in the section on feed-forward neural networks, the cases of exploding and vanishing gradient become more pronounced in RNNs due to the connectivity of units as discussed previously.
3. Some of the solutions for exploding gradients are:
 1. Truncated BPTT is a small change to the BPTT process. Instead of propagating the learning back to time $t=0$, it is truncated to a fixed time backward to $t=k$.
 2. Gradient Clipping to cut the gradient above a threshold when it shoots up.
 3. Adaptive Learning Rate. The learning rate adjusts itself based on the feedback and values.
4. Some of the solutions for vanishing gradients are:
 1. Using ReLU as the activation function; hence the gradient will be 1.
 2. Adaptive Learning Rate. The learning rate adjusts itself based on the feedback and values.
 3. Using extensions such as Long Short Term Memory (LSTM) and Gated Recurrent Units (GRUs), which we will describe next.

There are many applications of RNNs, for example, in next letter predictions, next word predictions, language translation, and so on.

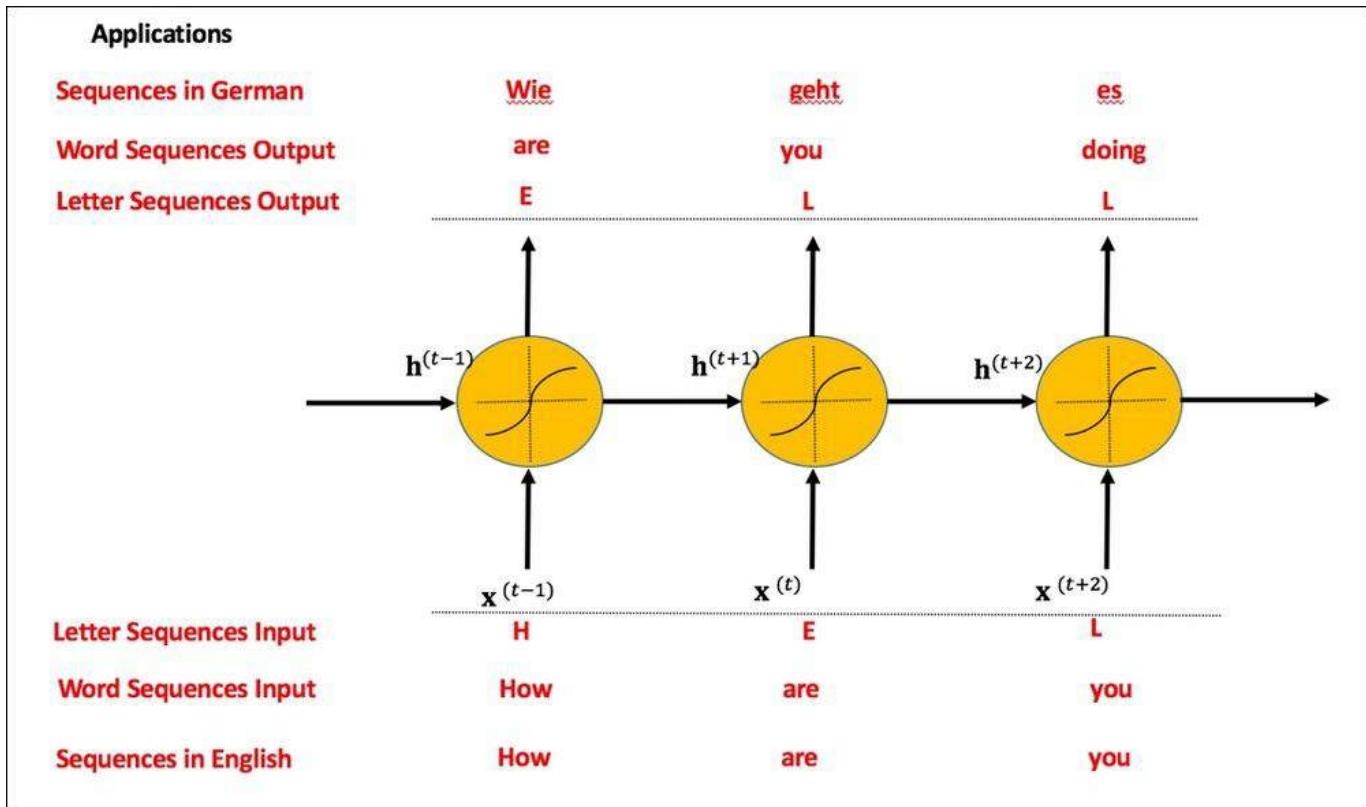


Figure 35: Showing some applications in next letter/word predictions using RNN structures.

Long Short Term Memory

One of the neural network architectures or modifications to RNNs that addresses the issue of vanishing gradient is known as long short term memory or LSTM. We will explain some building blocks of LSTM and then put it together for our readers.

The first modification to RNN is to change the feedback learning matrix to 1, that is, $\mathbf{W}_R = 1$, as shown in the following figure:

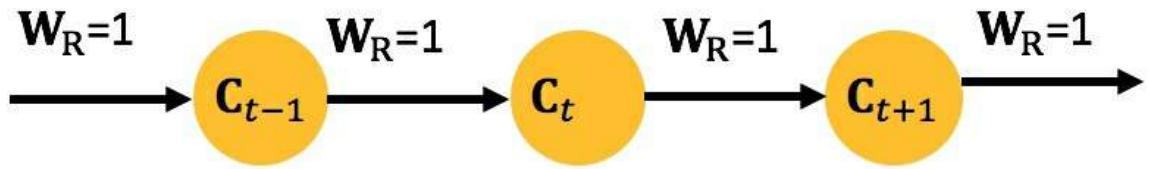


Figure 36: Building blocks of LSTM where the feedback matrix is set to 1.

This will ensure the inputs from older cell or memory units are passed as-is to the next unit. Hence some modifications are needed.

The output gate, as shown in the following figure, combines two computations. The first is the output from the individual unit, passed through an activation function, and the second is the output of the older unit that has been passed through a sigmoid using scaling.

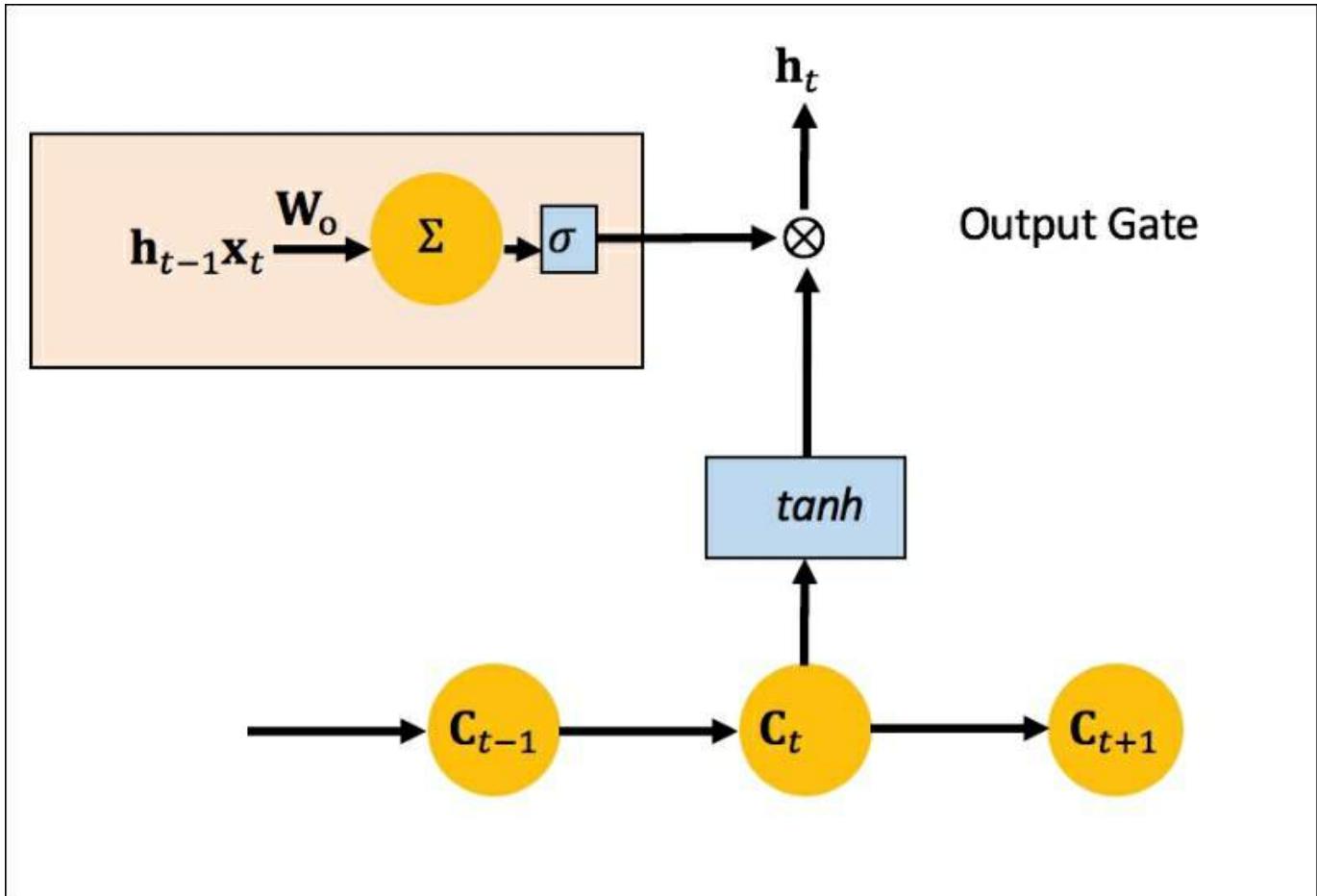


Figure 37: Building block Output Gate for LSTM.

Mathematically, the output gate at the unit is given by:

$$\mathbf{h}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1} \mathbf{x}_t] + b_0) \odot \tanh(\mathbf{C}_t)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

The forget gate is between the two memory units. It generates 0 or 1 based on

learned weights and transformations. The forget gate is shown in the following figure:

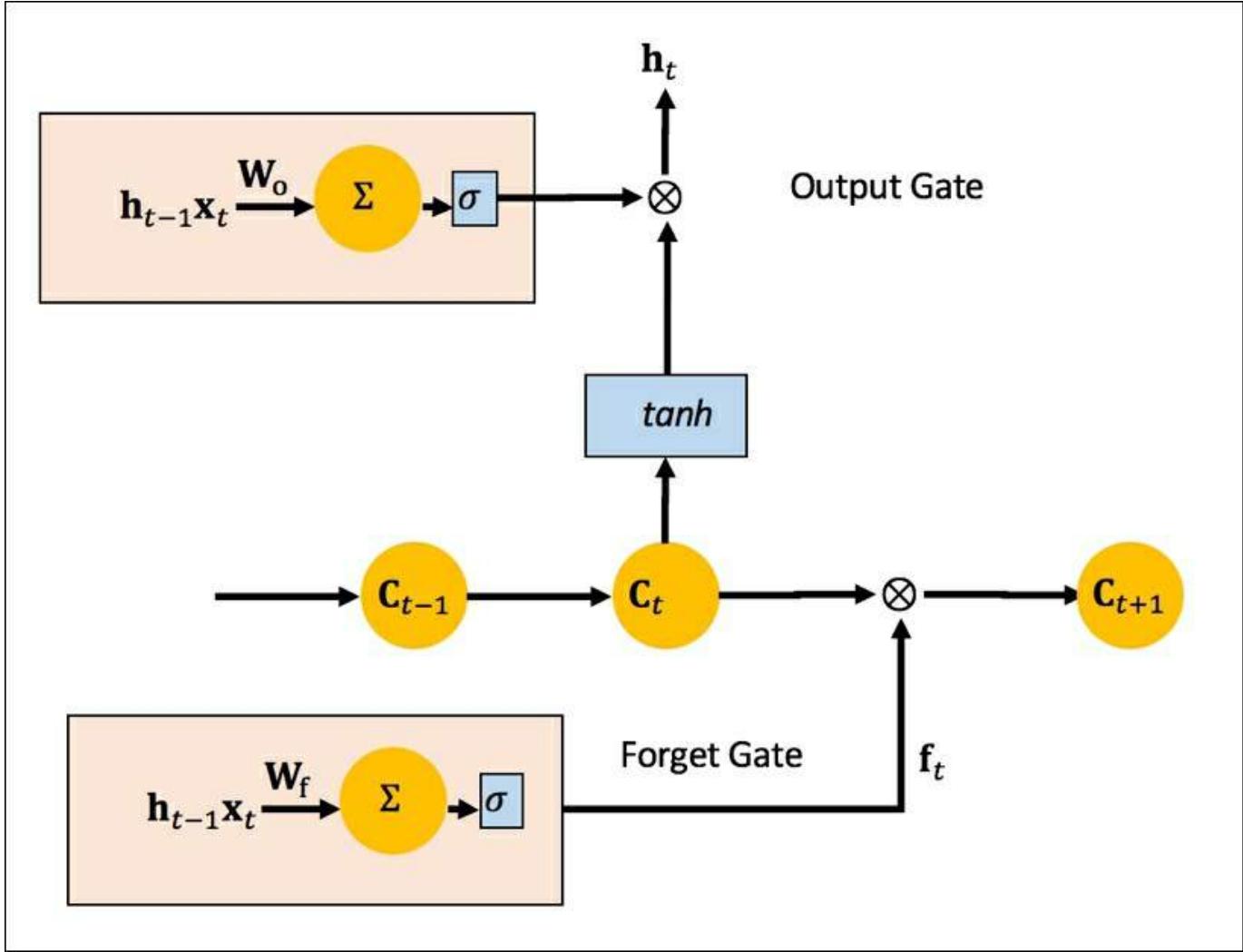


Figure 38: Building block Forget Gate addition to LSTM.

Mathematically, $\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1} \mathbf{x}_t] + \mathbf{b}_f)$ can be seen as the representation of the forget gate. Next, the input gate and the new gate are combined, as shown in the following figure:

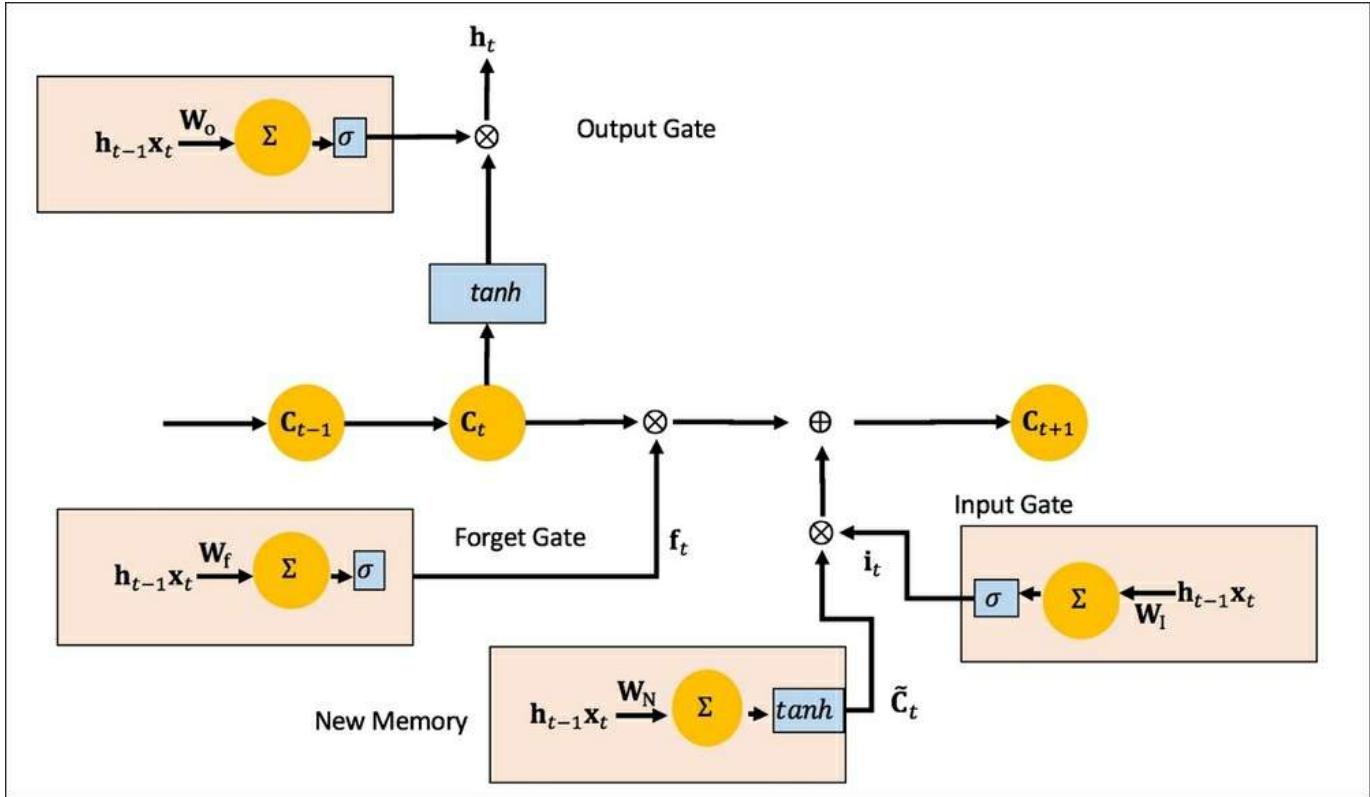


Figure 39: Building blocks New Gate and Input Gate added to complete LSTM.

The new memory generation unit uses the current input x_t and the old state h_{t-1} through an activation function and generates a new memory C_t . The input gate combines the input and the old state and determines whether the new memory or the input should be preserved.

Thus, the update equation looks like this:

$$C_{t+1} = f_t \odot C_t + i_t \odot \tilde{C}_t$$

Gated Recurrent Units

Gated Recurrent Units (GRUs) are simplified LSTMs with modifications. Many of the gates are simplified by using one "update" unit as follows:

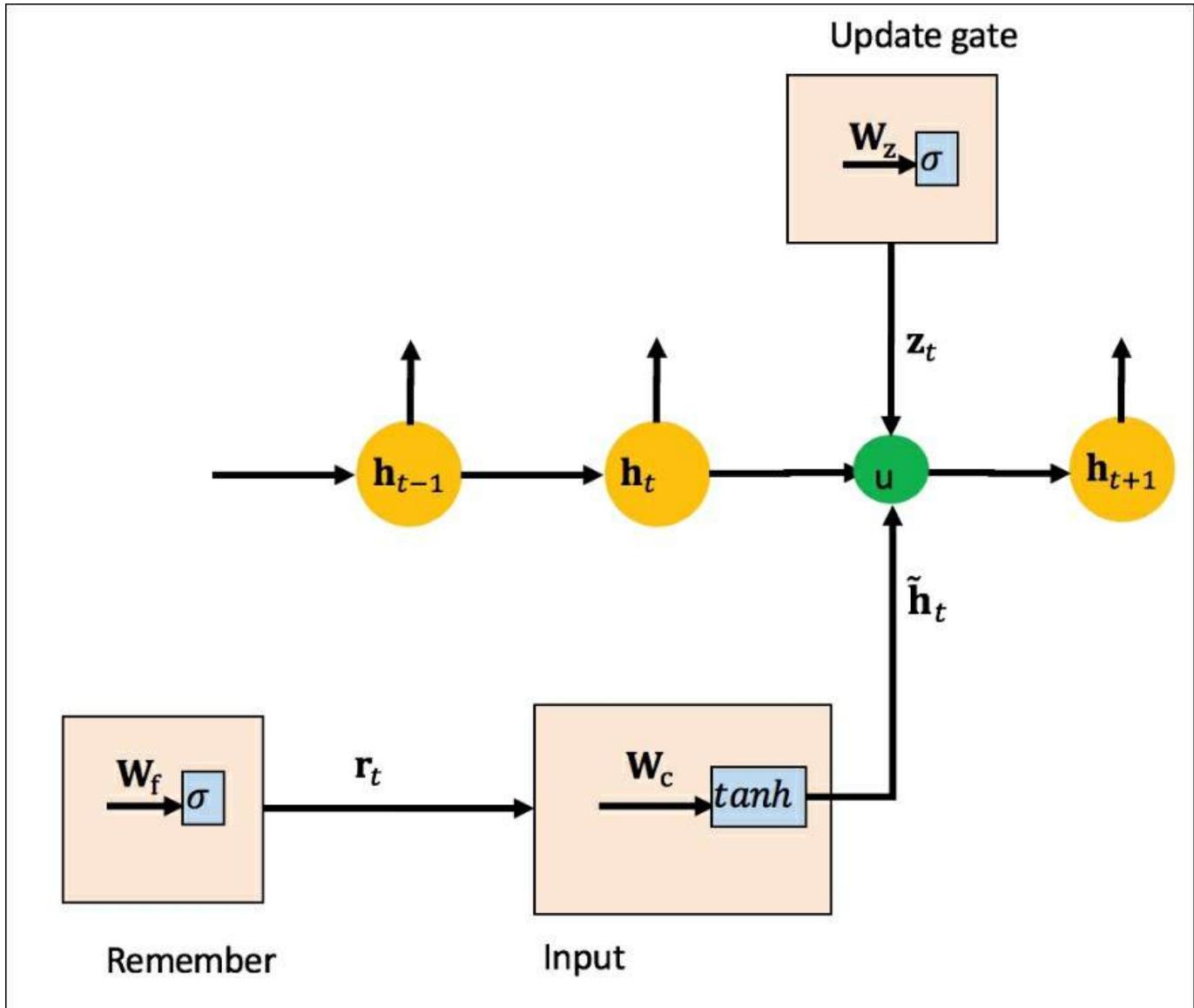


Figure 40: GRUs with Update unit.

The changes made to the equations are:

$$z_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + b_z)$$

$$\mathbf{r}_t = \sigma\Big(\mathbf{W}_f\cdot\big[\mathbf{h}_{t-1}, \mathbf{x}_t\big] + b_f\Big)$$

$$\mathbf{h}_t^{\infty}=\tanh\Big(\mathbf{W}_c\cdot\big[\mathbf{r}_t\mathbf{h}_{t-1}, \mathbf{x}_t\big] + b_c\Big)$$

$$\mathbf{h}_t^{\infty}=\tanh\Big(\mathbf{W}_c\cdot\big[\mathbf{r}_t\mathbf{h}_{t-1}, \mathbf{x}_t\big] + b_c\Big)$$

$$\mathbf{h}_t = \left(1-\mathbf{z}_t\right) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \mathbf{h}_t^{\infty}$$

Case study

Several benchmarks exist for image classification. We will use the MNIST image database for this case study. When we used MNIST in [Chapter 3](#), Unsupervised Machine Learning Techniques with clustering and outlier detection techniques, each pixel was considered a feature. In addition to learning from the pixel values as in previous experiments, with deep learning techniques we will also be learning new features from the structure of the training dataset. The deep learning algorithms will be trained on 60,000 images and tested on a 10,000-image test dataset.

Tools and software

In this chapter, we introduce the open-source Java framework for deep learning called DeepLearning4J (DL4J). DL4J has libraries implementing a host of deep learning techniques and they can be used on distributed CPUs and GPUs.

DeepLearning4J: <https://deeplearning4j.org/index.html>

We will illustrate the use of some DL4J libraries in learning from the MNIST training images and apply the learned models to classify the images in the test set.

Business problem

Image classification is a particularly attractive test-bed to evaluate deep learning networks. We have previously encountered the MNIST database, which consists of greyscale images of handwritten digits. This time, we will show how both unsupervised and supervised deep learning techniques can be used to learn from the same dataset. The MNIST dataset has 28-by-28 pixel images in a single channel. These images are categorized into 10 labels representing the digits 0 to 9. The goal is to train on 60,000 data points and test our deep learning classification algorithm on the remaining 10,000 images.

Machine learning mapping

This includes supervised and unsupervised methods applied to a classification problem in which there are 10 possible output classes. Some techniques use an initial pre-training stage, which is unsupervised in nature, as we have seen in the preceding sections.

Data sampling and transform

The dataset is available at:

<https://yann.lecun.com/exdb/mnist>

In the experiments in this case study, the MNIST dataset has been standardized such that pixel values in the range 0 to 255 have been normalized to values from 0.0 to 1.0. The exception is in the experiment using stacked RBMs, where the training and test data have been binarized, that is, set to 1 if the standardized value is greater than or equal to 0.3 and 0 otherwise. Each of the 10 classes is equally represented in both the training set and the test set. In addition, examples are shuffled using a random number generator seed supplied by the user.

Feature analysis

The input data features are the greyscale values of the pixels in each image. This is the raw data and we will be using the deep learning algorithms to learn higher-level features out of the raw pixel values. The dataset has been prepared such that there are an equal number of examples of each class in both the training and the test sets.

Models, results, and evaluation

We will perform different experiments starting with simple MLP, Convolutional Networks, Variational Autoencoders, Stacked RBMS, and DBNs. We will walk through important parts of code that highlight the network structure or specialized tunings, give parameters to help readers, reproduce the experiments, and give the results for each type of network.

Basic data handling

The following snippet of code shows:

How to generically read data from a CSV with a structure enforced by delimiters.

How to iterate the data and get records.

How to shuffle data in memory and create training/testing or validation sets:

```
RecordReader recordReader = new ]  
CSVRecordReader(numLinesToSkip,delimiter);  
recordReader.initialize(new FileSplit(new  
ClassPathResource(fileName).getFile()));  
DataSetIterator iterator = new  
RecordReaderDataSetIterator(recordReader,batchSize,labelIndex,numClasse  
s);  
DataSet allData = iterator.next();  
allData.shuffle();  
SplitTestAndTrain testAndTrain =  
allData.splitTestAndTrain(trainPercent);  
DataSet trainingData = testAndTrain.getTrain();  
DataSet testData = testAndTrain.getTest();
```

DL4J has a specific MNIST wrapper for handling the data that we have used, as shown in the following snippet:

```
DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize, true,  
randomSeed);  
DataSetIterator mnistTest = new MnistDataSetIterator(batchSize, false,  
randomSeed);
```

Multi-layer perceptron

In the first experiment, we will use a basic multi-layer perceptron with an input

layer, one hidden layer, and an output layer. A detailed list of parameters that are used in the code is given here:

Parameters used for MLP

Parameter	Variable	Value
Number of iterations	m	1
Learning rate	rate	0.0015
Momentum	momentum	0.98
L2 regularization	regularization	0.005
Number of rows in input	numRows	28
Number of columns in input	numColumns	28
Layer 0 output size, Layer 1 input size	outputLayer0, inputLayer1	500
Layer 1 output size, Layer 2 input size	outputLayer1, inputLayer2	300
Layer 2 output size, Layer 3 input size	outputLayer2, inputLayer3	100
Layer 3 output size,	outputNum	10

Code for MLP

In the listing that follows, we can see how we first configure the MLP by passing in the hyperparameters using the Builder pattern.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
.seed(randomSeed)
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT) // use SGD
.iterations(m) //iterations
.activation(Activation.RELU)//activation function
.weightInit(WeightInit.XAVIER)//weight initialization
.learningRate(rate) //specify the learning rate
.updater(Updater.NESTEROVS).momentum(momentum)//momentum
.regularization(true).l2(rate * regularization) //
.list()
```

```

.layer(0,
new DenseLayer.Builder() //create the first input layer.
.nIn(numRows * numColumns)
.nOut(firstOutput)
.build())
.layer(1, new DenseLayer.Builder() //create the second input layer
.nIn(secondInput)
.nOut(secondOutput)
.build())
.layer(2, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD)
//create hidden layer
.activation(Activation.SOFTMAX)
.nIn(thirdInput)
.nOut(numberOfOutputClasses)
.build())
.pretrain(false).backprop(true) //use backpropagation to adjust weights
.build();

```

Training, evaluation, and testing the MLP are shown in the following snippet. Notice the code that initializes the visualization backend enabling you to monitor the model training in your browser, particularly the model score (the training error after each iteration) and updates to parameters:

```

MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(5)); //print the score
with every iteration
//Initialize the user interface backend
UIServer uiServer = UIServer.getInstance();
//Configure where the network information (gradients, activations,
score vs. time etc) is to be stored
//Then add the StatsListener to collect this information from the
network, as it trains
StatsStorage statsStorage = new InMemoryStatsStorage();
//Alternative: new FileStatsStorage(File) - see UIStorageExample
int listenerFrequency = 1;
net.setListeners(new StatsListener(statsStorage, listenerFrequency));
//Attach the StatsStorage instance to the UI: this allows the contents
of the StatsStorage to be visualized
uiServer.attach(statsStorage);
log.info("Train model....");
for( int i=0; i<numEpochs; i++ ){
log.info("Epoch " + i);
model.fit(mnistTrain);
}
log.info("Evaluate model....");
Evaluation eval = new Evaluation(numberOfOutputClasses);

```

```

while(mnistTest.hasNext()) {
    DataSet next = mnistTest.next();
    INDArray output = model.output(next.getFeatureMatrix()); //get the
    networks prediction
    eval.eval(next.getLabels(), output); //check the prediction against the
    true class
}
log.info(eval.stats());

```

The following plots show the training error against training iteration for the MLP model. This curve should decrease with iterations:

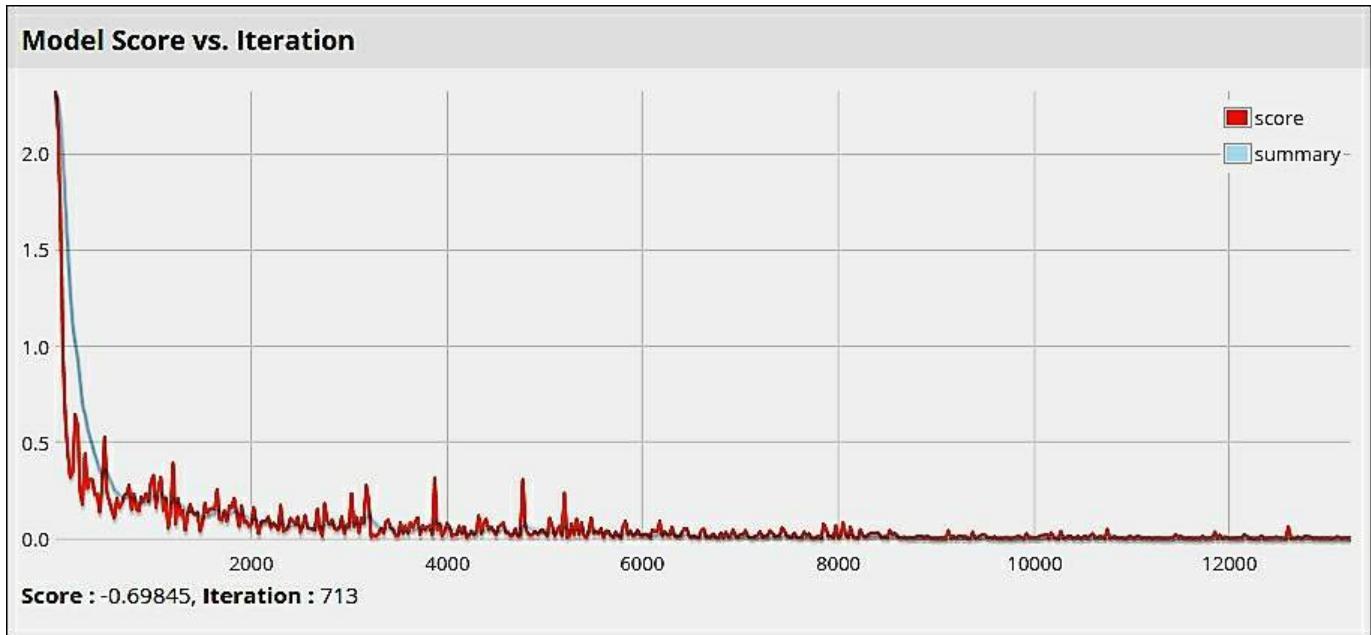


Figure 41: Training error as measured with number of iterations of training for the MLP model.

In the following figure, we see the distribution of parameters in Layer 0 of the MLP as well as the distribution of updates to the parameters. These histograms should have an approximately Gaussian (Normal) shape, which indicates good convergence. For more on how to use charts to tune your model, see the DL4J Visualization page (<https://deeplearning4j.org/visualization>):

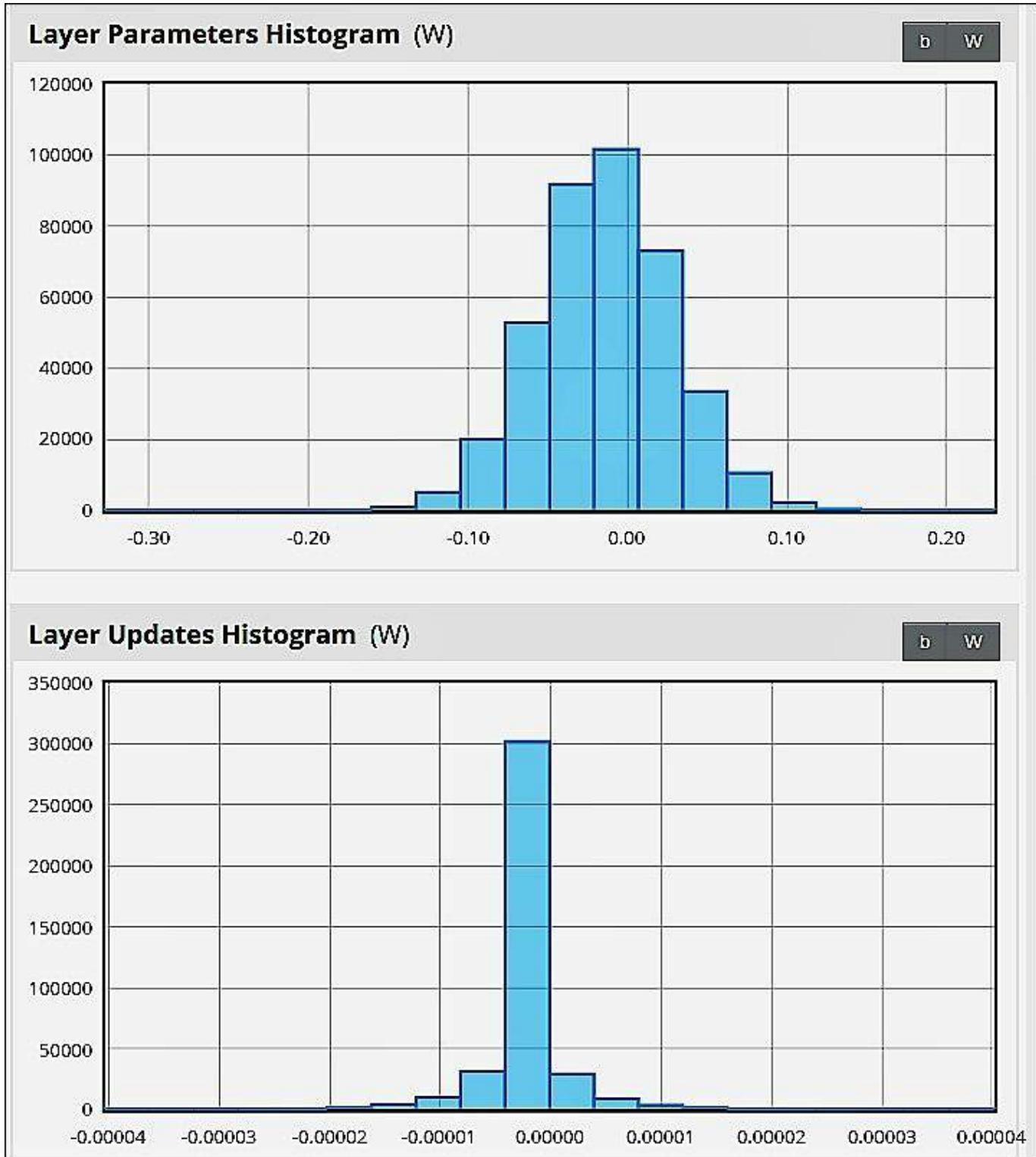


Figure 42: Histograms showing Layer parameters and update distribution.

Convolutional Network

In the second experiment, we configured a Convolutional Network (ConvNet) using the built-in MultiLayerConfiguration. The architecture of the network consists of a total of five layers, as can be seen from the following code snippet. Following the input layer, two convolution layers with 5-by-5 filters alternating with Max pooling layers are followed by a fully connected dense layer using the ReLu activation layer, ending with Softmax activation in the final output layer. The optimization algorithm used is Stochastic Gradient Descent, and the loss function is Negative Log Likelihood.

The various configuration parameters (or hyper-parameters) for the ConvNet are given in the table.

Parameters used for ConvNet

Parameter	Variable	Value
Seed	seed	123
Input size	numRows, numColumns	28, 28
Number of epochs	numEpochs	10
Number of iterations	iterations	1
L2 regularization	regularization	0.005
Learning rate	learningRate	0.1
Momentum	momentum	0.9
Convolution filter size	xsize, ysize	5, 5
Convolution layers stride size	x, y	1, 1
Number of input channels	numChannels	1
Subsampling layer stride size	sx, sy	2, 2
Layer 0 output size	nOut0	20
Layer 2 output size	nOut1	50

Layer 4 output size	nOut2	500
Layer 5 output size	outputNum	10

Code for CNN

As you can see, configuring multi-layer neural networks with the DL4J API is similar whether you are building MLPs or CNNs. Algorithm-specific configuration is simply done in the definition of each layer.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
.seed(seed)
.iterations(iterations) .regularization(true).l2(regularization)
.learningRate(learningRate)
.weightInit(WeightInit.XAVIER)
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
.updater(Updater.NESTEROVS) .momentum(momentum)
.list()
.layer(0, new ConvolutionLayer.Builder(xsize, ysize)
.nIn(nChannels)
.stride(x, y)
.nOut(nOut0)
.activation(Activation.IDENTITY)
.build())
.layer(1, new SubsamplingLayer
.Builder(SubsamplingLayer.PoolingType.MAX)
.kernelSize(width, height)
.stride(sx,sy)
.build())
.layer(2, new ConvolutionLayer.Builder(xsize, ysize)
.stride(x, y)
.nOut(nOut2)
.activation(Activation.IDENTITY)
.build())
.layer(3, new SubsamplingLayer
.Builder(SubsamplingLayer.PoolingType.MAX)
.kernelSize(width, height)
.stride(sx,sy)
.build())
.layer(4, new DenseLayer.Builder()
.activation(Activation.RELU)
.nOut(nOut4).build())
.layer(5, new OutputLayer.
Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
.nOut(outputNum)
.activation(Activation.SOFTMAX)
```

```

.build())
.setInputType(InputType.convolutionalFlat(numRows, numColumns, 1))
.backprop(true).pretrain(false).build();

```

Variational Autoencoder

In the third experiment, we configure a Variational Autoencoder as the classifier.

Parameters used for the Variational Autoencoder

The parameters used to configure the VAE are shown in the table.

Parameter	Variable	Values
Seed for RNG	rngSeed	12345
Number of iterations	Iterations	1
Learning rate	learningRate	0.001
RMS decay	rmsDecay	0.95
L2 regularization	regularization	0.0001
Output layer size	outputNum	10
VAE encoder layers size	vaeEncoder1, vaeEncoder2	256, 256
VAE decoder layers size	vaeDecoder1, vaeDecoder2	256, 256
Size of latent variable space	latentVarSpaceSize	128

Code for Variational Autoencoder

We have configured two layers each of encoders and decoders and are reconstructing the input using a Bernoulli distribution.

```

MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
.seed(rngSeed)
.iterations(iterations)
.optimizationAlgo(
OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
.learningRate(learningRate)

```

```

.updater(Updater.RMSPROP).rmsDecay(rmmsDecay)
.weightInit(WeightInit.XAVIER)
.regularization(true).l2(regulaization)
.list()
.layer(0, new VariationalAutoencoder.Builder()
.activation(Activation.LEAKYRELU)
.encoderLayerSizes(vaeEncoder1, vaeEncoder2) //2
encoder layers
.decoderLayerSizes(vaeDecoder1, vaeDecoder2) //2
decoder layers
.pzxActivationFunction("identity") //p(z|x) activation function
.reconstructionDistribution(new
BernoulliReconstructionDistribution(Activation.SIGMOID.getActivationFunction())) //Bernoulli distribution for p(data|z) (binary or 0 to 1
data only)
.nIn(numRows * numColumns) //Input size
.nOut(latentVarSpaceSize) //Size of the latent variable space: p(z|x).
.build())
.layer(1, new
OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD).activation(Activation.SOFTMAX)
.nIn(latentVarSpaceSize).nOut(outputNum).build())
.pretrain(true).backprop(true).build();

```

DBN

The parameters used in DBN are shown in the following table:

Parameter	Variable	Value
Input data size	numRows, numColumns	28, 28
Seed for RNG	seed	123
Number of training iterations	iterations	1
Momentum	momentum	0.5
Layer 0 (input)	numRows * numColumns	28 * 28
Layer 0 (output)	nOut0	500
Layer 1 (input, output)	nIn1, nOut1	500, 250
Layer 2 (input, output)	nIn2, nOut2	250, 200
Layer 3 (input, output)	nIn3, outputNum	200, 10

Configuring the DBN using the DL4J API is shown in the example used in this case study. The code for the configuration of the network is shown here.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
.seed(seed)
.gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
.gradientNormalizationThreshold(1.0)
.iterations(iterations)
.updater(Updater.NESTEROVS)
.momentum(momentum)
.optimizationAlgo(OptimizationAlgorithm.CONJUGATE_GRADIENT)
.list()
.layer(0, new RBM.Builder().nIn(numRows*numColumns).nOut(nOut0)
.weightInit(WeightInit.XAVIER).lossFunction(LossFunction.KL_DIVERGENCE)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())
.layer(1, new RBM.Builder().nIn(nIn1).nOut(nOut1)
.weightInit(WeightInit.XAVIER).lossFunction(LossFunction.KL_DIVERGENCE)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())
.layer(2, new RBM.Builder().nIn(nIn2).nOut(nOut2)
.weightInit(WeightInit.XAVIER).lossFunction(LossFunction.KL_DIVERGENCE)
.visibleUnit(RBM.VisibleUnit.BINARY)
.hiddenUnit(RBM.HiddenUnit.BINARY)
.build())
.layer(3, new OutputLayer.Builder().nIn(nIn3).nOut(outputNum)
.weightInit(WeightInit.XAVIER).activation(Activation.SOFTMAX)
.build())
.pretrain(true).backprop(true)
.build();
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(listenerFreq));
```

Parameter search using Arbiter

DeepLearning4J provides a framework for fine-tuning hyper-parameters by taking the burden of hand-tuning away from the modeler; instead, it allows the specification of the parameter space to search. In the following example code snippet, the configuration is specified using a MultiLayerSpace instead of a MultiLayerConfiguration object, in which the ranges for the hyper-parameters are specified by means of ParameterSpace objects in the Arbiter DL4J package for the

parameters to be tuned:

```
ParameterSpace<Double> learningRateHyperparam = new  
ContinuousParameterSpace(0.0001, 0.1); //Values will be generated  
uniformly at random between 0.0001 and 0.1 (inclusive)  
ParameterSpace<Integer> layerSizeHyperparam = new  
IntegerParameterSpace(16,256); //Integer values will be  
generated uniformly at random between 16 and 256 (inclusive)  
MultiLayerSpace hyperparameterSpace = new MultiLayerSpace.Builder()  
//These next few options: fixed values for all models  
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)  
.iterations(1)  
.regularization(true)  
.l2(0.0001)  
//Learning rate: this is something we want to test different values for  
.learningRate(learningRateHyperparam)  
.addLayer( new DenseLayerSpace.Builder()  
//Fixed values for this layer:  
.nIn(784) //Fixed input: 28x28=784 pixels for MNIST  
.activation("relu")  
//One hyperparameter to infer: layer size  
.nOut(layerSizeHyperparam)  
.build())  
.addLayer( new OutputLayerSpace.Builder())  
//nIn: set the same hyperparameter as the nOut for the last layer.  
.nIn(layerSizeHyperparam)  
//The remaining hyperparameters: fixed for the output layer  
.nOut(10)  
.activation("softmax")  
.lossFunction(LossFunctions.LossFunction.MCXENT)  
.build())  
.pretrain(false).backprop(true).build();
```

Results and analysis

The results of evaluating the performance of the four networks on the test data are given in the following table:

	MLP	ConvNet	VAE	DBN
Accuracy	0.9807	0.9893	0.9743	0.7506
Precision	0.9806	0.9893	0.9742	0.7498
Recall	0.9805	0.9891	0.9741	0.7454

F1 score	0.9806	0.9892	0.9741	0.7476
----------	--------	--------	--------	--------

The goal of the experiments was not to match benchmark results in each of the neural network structures, but to give a comprehensive architecture implementation in the code with detailed parameters for the readers to explore.

Tuning the hyper-parameters in deep learning Networks is quite a challenge and though Arbiter and online resources such as gitter (<https://gitter.im/deeplearning4j/deeplearning4j>) help with DL4J, the time and cost of running the hyper-parameter search is quite high as compared to other classification techniques including SVMs.

The benchmark results on the MNIST dataset and corresponding papers are available here:

- <http://yann.lecun.com/exdb/mnist/>
- http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

As seen from the benchmark result, Linear 1 Layer NN gets an error rate of 12% and adding more layers reduces it to about 2. This shows the non-linear nature of the data and the need for a complex algorithm to fit the patterns.

As compared to the benchmark best result on neural networks ranging from a 2.5% to 1.6% error rate, our results are very much comparable with the 2% error rate.

Most of the benchmark results show Convolutional Network architectures having error rates in the range of 1.1% to 0.5% and our hyper-parameter search has matched the best of those models with an error rate of just under 1.1%.

Our results for DBN fall far short of the benchmarks at just over 25%. There is no reason to doubt that further tuning can improve performance bringing it to the range of 3-5%.

Summary

The history of Deep Learning is intimately tied to the limitations of earlier attempts at using neural networks in machine learning and AI, and how these limitations were overcome with newer techniques, technological improvements, and the availability of vast amounts of data.

The perceptron is the basic neural network. Multi-layer networks are used in supervised learning and are built by connecting several hidden layers of neurons to propagate activations forward and using backpropagation to reduce the training error. Several activation functions are used, most commonly, the sigmoid and tanh functions.

The problems of neural networks are vanishing or exploding gradients, slow training, and the trap of local minima.

Deep learning successfully addresses these problems with the help of several effective techniques that can be used for unsupervised as well as supervised learning.

Among the building blocks of deep learning networks are Restricted Boltzmann Machines (RBM), Autoencoders, and Denoising Autoencoders. RBMs are two-layered undirected networks that are able to extract high-level features from their input. Contrastive divergence is used to speed up the training. Autoencoders are also deep learning networks used in unsupervised learning—they attempt to replicate the input by first encoding learned features in the encoding layer and then reconstructing the input via a set of decoding layers. Denoising Autoencoders address some limitations of Autoencoders, which can sometimes cause them to trivially learn the identity function.

Deep learning networks are often pretrained in an unsupervised fashion and then their parameters are fine-tuned via supervised fine-tuning. Stacked RBMs or Autoencoders are used in the pretraining phase and the fine-tuning is typically accomplished with a softmax activation in the output layer in the case of classification.

Deep Autoencoders are good at learning complex latent structures in data and are used in unsupervised learning by employing pre-training and fine-tuning with Autoencoder building blocks. Deep Belief Networks (DBN) are generative models

that can be used to create more samples. It is constructed using a directed Bayesian network with an undirected RBM layer on top. Overfitting in deep learning networks can be addressed by learning with dropouts, where some nodes in the network are randomly "turned off".

Convolutional Neural Networks (CNNs) have a number of applications in computer vision. CNNs can learn patterns in the data translation-invariant and robust to linear scaling in the data. They reduce the dimensionality of the data using convolution filters and pooling layers and can achieve very effective results in classification tasks. A use case involving the classification of digital images is presented.

When the data arrives as sequences and there are temporal relationships among data, Recurrent Neural Networks (RNN) are used for modeling. RNNs use feedback from previous layers and emit output continually. The problem of vanishing and exploding gradients recurs in RNNs, and are addressed by several modifications to the architecture, such as Long Short Term Memory (LSTM) and Gated Recurrent Networks (GRU).

In this chapter's case study, we present the experiments done with various deep learning networks to learn from MNIST handwritten digit image datasets. Results using MLP, ConvNet, Variational Autoencoder, and Stacked RBM are presented.

We think that deep neural networks are able to approximate a significant and representative sub-set of key structures that the underlying data is based on. In addition, the hierachic structures of the data can be easily captured with the help of different hidden layers. Finally, the invariance against rotation, translation, and the scale of images, for instance, is the last key elements of the performance of deep neural networks. The invariance allows us to reduce the number of possible states to be captured by the neural network (*References [19]*).

References

1. Behnke, S. (2001). Learning iterative image reconstruction in the neural abstraction pyramid. *International Journal of Computational Intelligence and Applications*, 1(4), 427–438.
2. Behnke, S. (2002). Learning face localization using hierarchical recurrent networks. In *Proceedings of the 12th international conference on artificial neural networks* (pp. 1319–1324).
3. Behnke, S. (2003). Discovering hierarchical speech features using convolutional non-negative matrix factorization. In *Proceedings of the international joint conference on neural networks*, vol. 4 (pp. 2758–2763).
4. Behnke, S. (2003). LNCS, Lecture notes in computer science: Vol. 2766. *Hierarchical neural networks for image interpretation*. Springer. Behnke, S. (2005). Face localization and tracking in the neural abstraction pyramid. *Neural Computing and Applications*, 14(2), 97–103.
5. Casey, M. P. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6), 1135–1178.
6. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.
7. Goller, C.; Küchler, A (1996). ""Learning task-dependent distributed representations by backpropagation through structure"". *Neural Networks*, IEEE. doi:10.1109/ICNN.1996.548916
8. Hochreiter, Sepp. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02): 107–116, 1998.
9. G. E. Hinton, S. Osindero, and Y. The (2006). "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554.
10. G. E. Hinton (2002). "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, pp. 1771–1800.
11. G. E. Hinton and R. R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507.
12. Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. *Advances in Neural Information Processing Systems*, 6, 3–10.
13. Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. (2007). "Greedy layer-

- wise training of deep networks," in Advances in Neural Information Processing Systems 19 (NIPS'06) pp. 153–160.
- 14. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio (2007). "An empirical evaluation of deep architectures on problems with many factors of variation," in Proc. 24th Int. Conf. Machine Learning (ICML'07) pp. 473–480.
 - 15. P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008), "Extracting and composing robust features with denoising autoencoders," in Proc. 25th Int. Conf. Machine Learning (ICML'08), pp. 1096–1103.
 - 16. F.-J. Huang and Y. LeCun (2006). "Large-scale learning with SVM and convolutional nets for generic object categorization," in Proc. Computer Vision and Pattern Recognition Conf. (CVPR'06).
 - 17. F. A. Gers, N. N. Schraudolph, and J. Schmidhuber (2003). Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*.
 - 18. Kyunghyun Cho et. al (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.
<https://arxiv.org/pdf/1406.1078.pdf>.
 - 19. https://brohrer.github.io/how_convolutional_neural_networks_work.html
 - 20. Henry W. Lin, Max Tegmark, David Rolnick (2016). Why does deep and cheap learning work so well? <https://arxiv.org/abs/1608.08225>
 - 21. Mike Schuster and Kuldip K. Paliwal (1997). Bidirectional Recurrent Neural Networks, *Trans. on Signal Processing*.
 - 22. H Lee, A Battle, R Raina, AY Ng (2007). Efficient sparse coding algorithms, In Advances in Neural Information Processing Systems
 - 23. Bengio Y. (2009). Learning deep architectures for AI, Foundations and Trends in Machine Learning 1(2) pages 1-127.

Chapter 8. Text Mining and Natural Language Processing

Natural language processing (NLP) is ubiquitous today in various applications such as mobile apps, ecommerce websites, emails, news websites, and more. Detecting spam in e-mails, characterizing e-mails, speech synthesis, categorizing news, searching and recommending products, performing sentiment analysis on social media brands—these are all different aspects of NLP and mining text for information.

There has been an exponential increase in digital information that is textual in content—in the form of web pages, e-books, SMS messages, documents of various formats, e-mails, social media messages such as tweets and Facebook posts, now ranges in exabytes (an exabyte is 1,018 bytes). Historically, the earliest foundational work relying on automata and probabilistic modeling began in the 1950s. The 1970s saw changes such as stochastic modeling, Markov modeling, and syntactic parsing, but their progress was limited during the 'AI Winter' years. The 1990s saw the emergence of text mining and a statistical revolution that included ideas of corpus statistics, supervised Machine Learning, and human annotation of text data. From the year 2000 onwards, with great progress in computing and Big Data, as well as the introduction of sophisticated Machine Learning algorithms in supervised and unsupervised learning, the area has received rekindled interest and is now among the hottest topics in research, both in academia and the R&D departments of commercial enterprises. In this chapter, we will discuss some aspects of NLP and text mining that are essential in Machine Learning.

The chapter begins with an introduction to the key areas within NLP, and it then explains the important processing and transformation steps that make the documents more suitable for Machine Learning, whether supervised or unsupervised. The concept of topic modeling, clustering, and named entity recognition follow, with brief descriptions of two Java toolkits that offer powerful text processing capabilities. The case study for this chapter uses another widely-known dataset to demonstrate several techniques described here through experiments using the tools KNIME and Mallet.

The chapter is organized as follows:

- NLP, subfields, and tasks:
 - Text categorization
 - POS tagging
 - Text clustering
 - Information extraction and named entity recognition
 - Sentiment analysis
 - Coreference resolution
 - Word-sense disambiguation
 - Machine translation
 - Semantic reasoning and inferencing
 - Summarization
 - Questions and answers
 - Issues with mining and unstructured data
- Text processing components and transformations:
 - Document collection and standardization
 - Tokenization
 - Stop words removal
 - Stemming/Lemmatization
 - Local/Global dictionary
 - Feature extraction/generation
 - Feature representation and similarity
 - Feature selection and dimensionality reduction
- Topics in text mining:
 - Topic modeling
 - Text clustering
 - Named entity recognition
 - Deep learning and NLP
- Tools and usage:
 - Mallet
 - KNIME
- Case study

NLP, subfields, and tasks

Information about the real world exists in the form of structured data, typically generated by automated processes, or unstructured data, which, in the case of text, is created by direct human agency in the form of the written or spoken word. The process of observing real-world situations and using either automated processes or having humans perceive and convert that information into understandable data is very similar in both structured and unstructured data. The transformation of the observed world into unstructured data involves complexities such as the language of the text, the format in which it exists, variances among different observers in interpreting the same data, and so on. Furthermore, the ambiguity caused by the syntax and semantics of the chosen language, subtlety in expression, the context in the data, and so on, make the task of mining text data very difficult.

Next, we will discuss some high-level subfields and tasks that involve NLP and text mining. The subject of NLP is quite vast, and the following topics is in no way comprehensive.

Text categorization

This field is one of the most well-established, and in its basic form classifies documents with unstructured text data into predefined categories. This can be viewed as a direct extension of supervised Machine Learning in the unstructured text world, learning from historic documents to predict categories of unseen documents in the future. Basic methods in spam detection in e-mails or news categorization are among some of the most prominent applications of this task.

Spam detection	Text categorization
Email Body: We need to setup a meeting	✓ Madonna to perform in NYC ENTERTAINMENT
Email Body: Forget Hair Fall forever!! Contact ...	✗ NASDAQ down by 20 points. BUSINESS
	President Obama signs a treaty. NEWS

Figure 1: Text Categorization showing classification into different categories

Part-of-speech tagging (POS tagging)

Another subtask in NLP that has seen a lot of success is associating parts-of-speech of the language—such as nouns, adjectives, verbs—to words in a text, based on context and relationship to adjacent words. Today, instead of manual POS tagging, automated and sophisticated POS taggers perform the job.

Part-of-speech (POS) tagging						
DT	ADJ	NN	VB	PP	DT	NN
The grand jury commented on a number						

Figure 2: POS Tags associated with segment of text

Text clustering

Clustering unstructured data for organization, retrieval, and groupings based on similarity is the subfield of text clustering. This field is also well-developed with advancements in different clustering and text representations suited for learning.

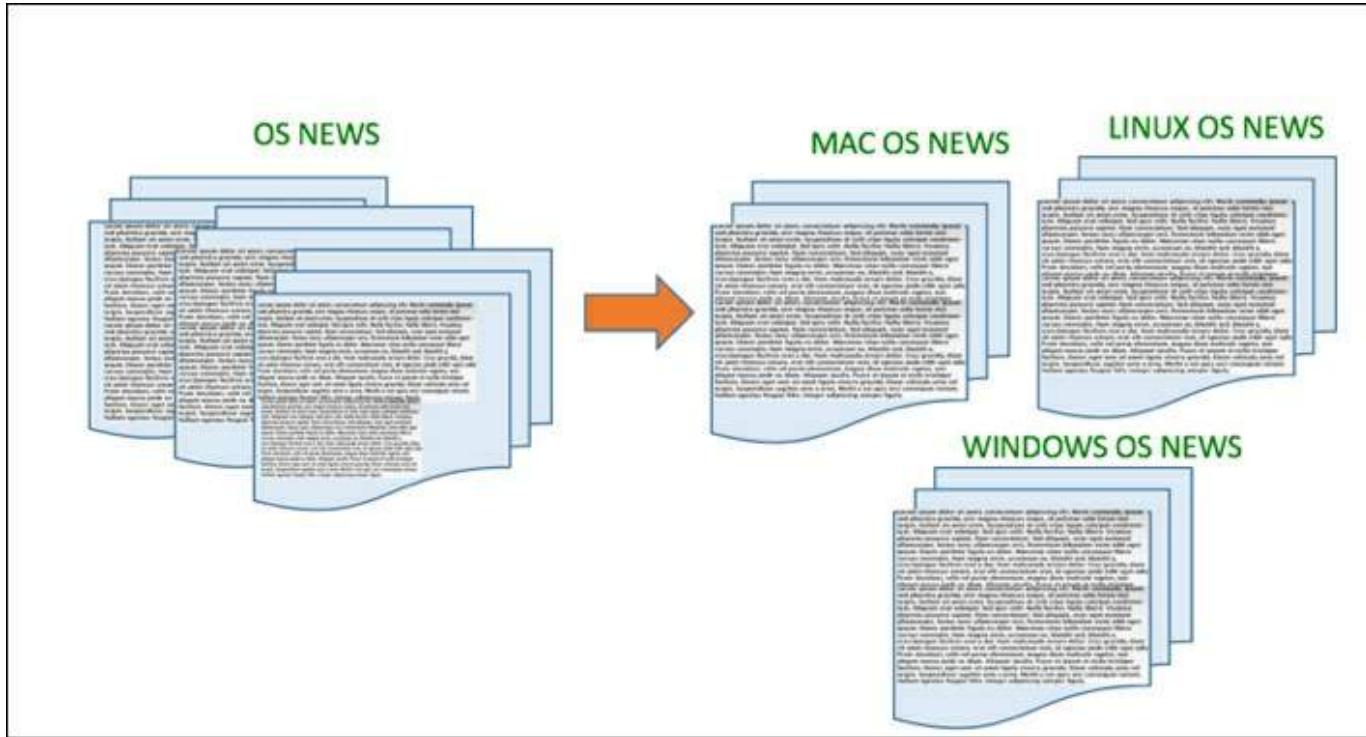


Figure 3: Clustering of OS news documents to various OS specific clusters

Information extraction and named entity recognition

The task of extracting specific elements, such as time, location, organization, entities, and so on, comes under the topic of information extraction. Named entity recognition is a sub-field that has wide applications in different domains, from reviews of historical documents to bioinformatics with gene and drug information.

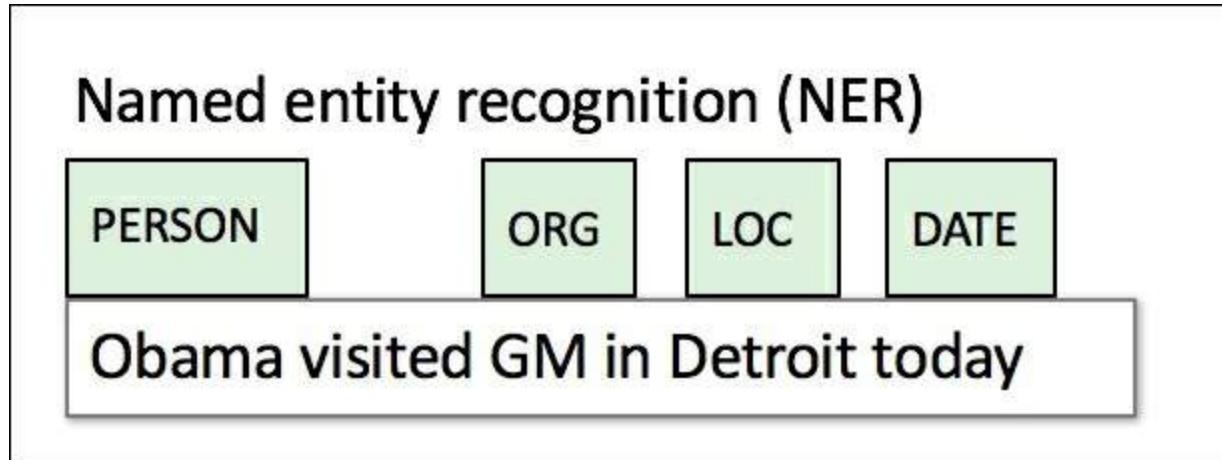


Figure 4: Named Entity Recognition in a sentence

Sentiment analysis and opinion mining

Another sub-field in the area of NLP involves inferring the sentiments of observers in order to categorize them with an understandable metric or to give insights into their opinions. This area is not as advanced as some of the ones mentioned previously, but much research is being done in this direction.

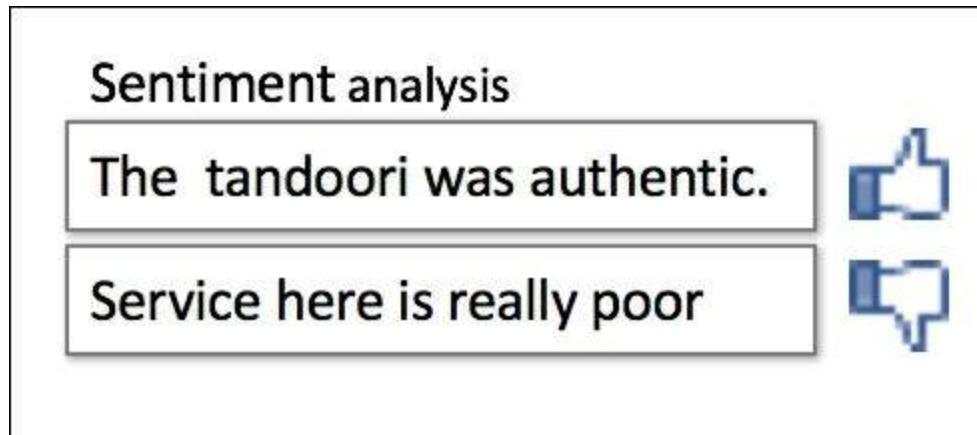


Figure 5: Sentiment Analysis showing positive and negative sentiments for sentences

Coreference resolution

Understanding references to multiple entities existing in the text and disambiguating that reference is another popular area of NLP. This is considered as a stepping stone in doing more complex tasks such as question answering and summarization, which will be discussed later.

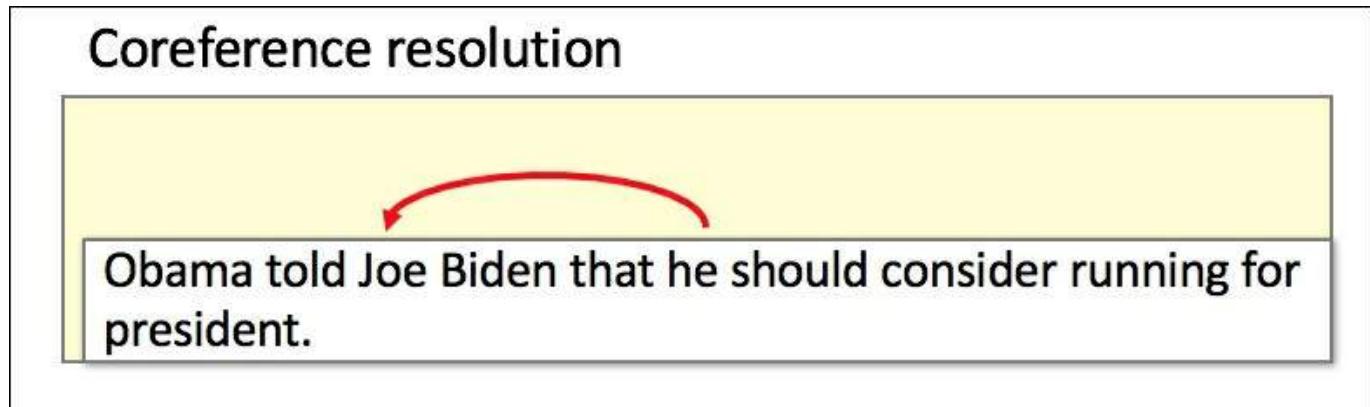


Figure 6: Coreference resolution showing how pronouns get disambiguated

Word sense disambiguation

In a language such as English, since the same word can have multiple meanings based on the context, deciphering this automatically is an important part of NLP, and the focus of **word sense disambiguation (WSD)**.

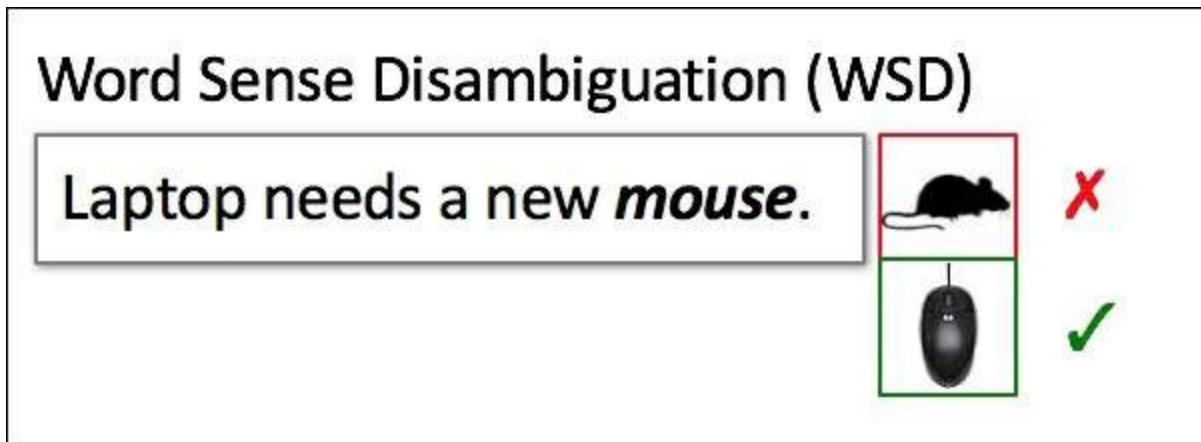


Figure 7: Showing how word "mouse" is associated with right word using the context

Machine translation

Translating text from one language to another or from speech to text in different languages is broadly covered in the area of **machine translation (MT)**. This field has made significant progress in the last few years, with the usage of Machine Learning algorithms in supervised, unsupervised, and semi-supervised learning. Deep learning with techniques such as LSTM has been proved to be the most effective technique in this area, and is widely used by Google for its translation.



Figure 8: Machine Translation showing English to Chinese conversion

Semantic reasoning and inferencing

Reasoning, deriving logic, and inferencing from unstructured text is the next level of advancement in NLP.

Semantic Inferencing

Data:

Marie Curie, was the first woman to win Nobel prize in 1903 in Physics.

Question:

Has any woman won Nobel in any field before 1903?

Inference: No

Figure 9: Semantic Inferencing answering complex questions

Text summarization

A subfield that is growing in popularity in NLP is the automated summarization of large documents or passages of text to a small representative text that can be easily understood. This is one of the budding research areas in NLP. Search engines' usage of summaries, multi-document summarizations for experts, and so on, are some of the applications that are benefiting from this field.

Automating question and answers

Answering questions posed by humans in natural language, ranging from questions specific to a certain domain to generic, open-ended questions is another emerging field in the area of NLP.

Issues with mining unstructured data

Humans can read, parse, and understand unstructured text/documents more easily than computer-based programs. Some of the reasons why text mining is more complicated than general supervised or unsupervised learning are given here:

- Ambiguity in terms and phrases. The word *bank* has multiple meanings, which a human reader can correctly associate based on context, yet this requires preprocessing steps such as POS tagging and word sense disambiguation, as we have seen. According to the Oxford English Dictionary, the word *run* has no fewer than 645 different uses in the verb form alone and we can see that such words can indeed present problems in resolving the meaning intended (between them, the words run, put, set, and take have more than a thousand meanings).
- Context and background knowledge associated with the text. Consider a sentence that uses a neologism with the suffix *gate* to signify a political scandal, as in, *With cries for impeachment and popularity ratings in a nosedive, Russiagate finally dealt a deathblow to his presidency.* A human reader can surmise what is being referred to by the coinage *Russiagate* as something that recalls the sense of high-profile intrigue, by association via an affix, of another momentous scandal in US political history, *Watergate*. This is particularly difficult for a machine to make sense of.
- Reasoning, that is, inferencing from documents is very difficult as mapping unstructured information to knowledge bases is itself a big hurdle.
- Ability to perform supervised learning needs labeled training documents and based on the domain, performing labeling on the documents can be time consuming and costly.

Text processing components and transformations

In this section, we will discuss some common preprocessing and transformation steps that are done in most text mining processes. The general concept is to convert the documents into structured datasets with features or attributes that most Machine Learning algorithms can use to perform different kinds of learning.

We will briefly describe some of the most used techniques in the next section. Different applications of text mining might use different pieces or variations of the components shown in the following figure:

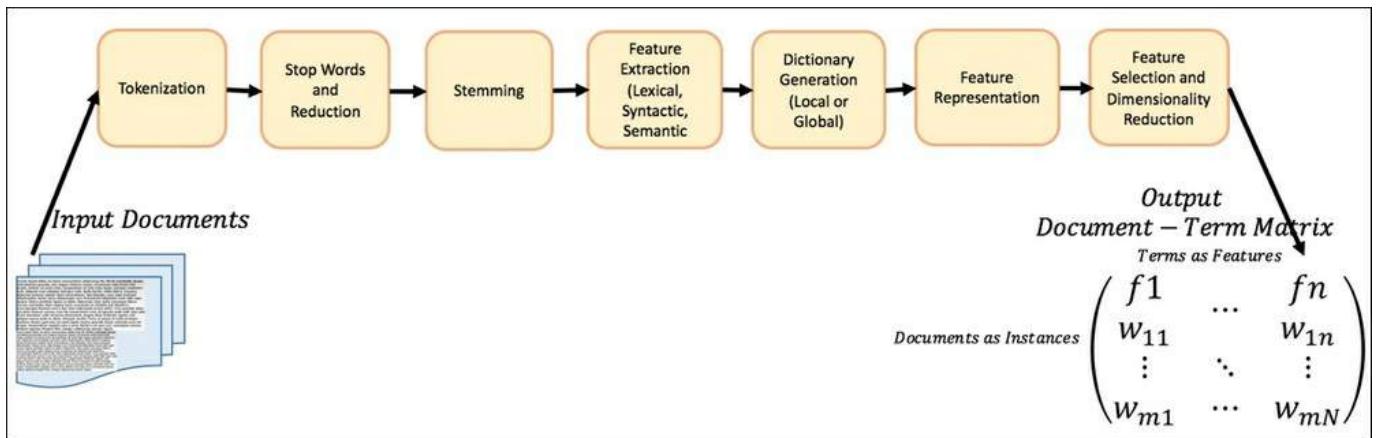


Figure 10: Text Processing components and the flow

Document collection and standardization

One of the first steps in most text mining applications is the collection of data in the form of a body of documents—often referred to as a *corpus* in the text mining world. These documents can have predefined categorization associated with them or it can simply be an unlabeled corpus. The documents can be of heterogeneous formats or standardized into one format for the next process of tokenization. Having multiple formats such as text, HTML, DOCs, PDGs, and so on, can lead to many complexities and hence one format, such as XML or **JavaScript Object Notation (JSON)** is normally preferred in most applications.

Inputs and outputs

Inputs are vast collections of homogeneous or heterogeneous sources and outputs are a collection of documents standardized into one format, such as XML.

How does it work?

Standardization involves ensuring tools and formats are agreed upon based on the needs of the application:

1. Agree to a standard format, such as XML, with predefined tags that provide information on meta-attributes of documents (`<author>`, `<title>`, `<date>`, and so on) and actual content, such as `<document>`.
2. Most document processors can either be transformed into XML or transformation code can be written to perform this.

Tokenization

The task of tokenization is to extract words or meaningful characters from the text containing a stream of these words. For example, the text *The boy stood up. He then ran after the dog* can be tokenized into tokens such as {the, boy, stood, up, he, ran, after, the, dog}.

Inputs and outputs

An input is a collection of documents in a well-known format as described in the last section and an output is a document with tokens of words or characters as needed in the application.

How does it work?

Any automated system for tokenization must address the particular challenges presented by the language(s) it is expected to handle:

- In languages such as English, tokenization is relatively simple due to the presence of white space, tabs, and newline for separating the words.
- There are different challenges in each language—even in English, abbreviations such as *Dr.*, alphanumeric characters (*B12*), different naming schemes (*O'Reilly*), and so on, must be tokenized appropriately.
- Language-specific rules in the form of if-then instructions are written to extract tokens from the documents.

Stop words removal

This involves removing high frequency words that have no discriminatory or predictive value. If every word can be viewed as a feature, this process reduces the dimension of the feature vector by a significant number. Prepositions, articles, and pronouns are some of the examples that form the stop words that are removed without affecting the performance of text mining in many applications.

Inputs and outputs

An input is a collection of documents with the tokens extracted and an output is a collection of documents with tokens reduced by removing stop words.

How does it work?

There are various techniques that have evolved in the last few years ranging from manually precompiled lists to statistical elimination using either term-based or mutual information.

- The most commonly used technique for many languages is a manually precompiled list of stop words, including prepositions (in, for, on), articles (a, an, the), pronouns (his, her, they, their), and so on.
- Many tools use Zipf's law (*References [3]*), where high frequency words, singletons, and unique terms are removed. Luhn's early work (*References [4]*), as represented in the following figure 11, shows thresholds of the upper bound and lower bound of word frequency, which give us the significant words that can be used for modeling:

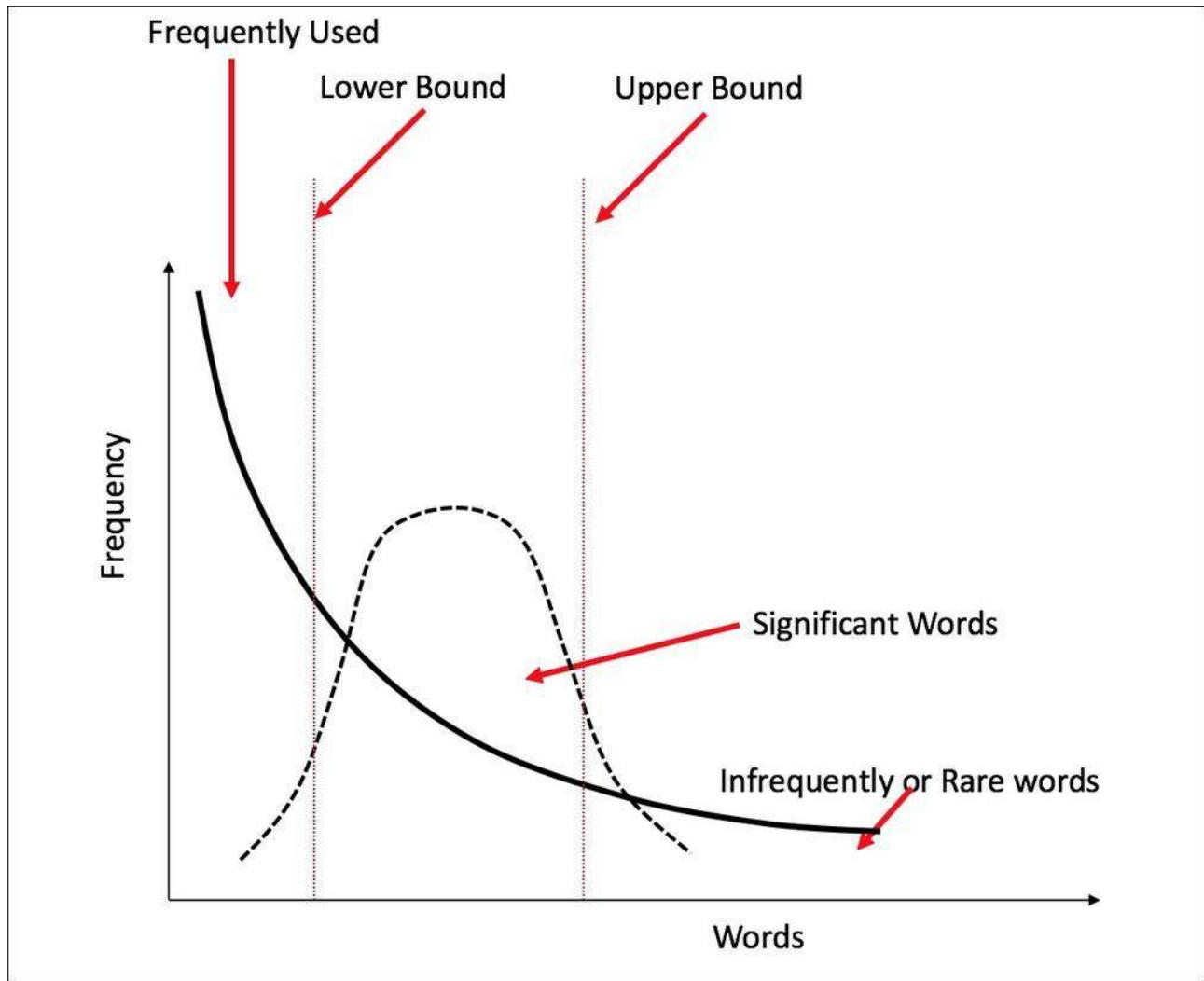


Figure 11: Word Frequency distribution, showing how frequently used, significant and rare words exist in corpus

Stemming or lemmatization

The idea of normalizing tokens of similar words into one is known as stemming or lemmatization. Thus, reducing all occurrences of "talking", "talks", "talked", and so on in a document to one root word "talk" in the document is an example of stemming.

Inputs and outputs

An input is documents with tokens and an output is documents with reduced tokens normalized to their stem or root words.

How does it work?

1. There are basically two types of stemming: inflectional stemming and stemming to the root.
2. Inflectional stemming generally involves removing affixes, normalizing the verb tenses and removing plurals. Thus, "ships" to "ship", "is", "are" and "am" to "be" in English.
3. Stemming to the root is generally a more aggressive form than inflectional stemming, where words are normalized to their roots. An example of this is "applications", "applied", "reapply", and so on, all reduced to the root word "apply".
4. Lovin's stemmer was one of the first stemming algorithms (*References [1]*). Porter's stemming, which evolved in the 1980s with around 60 rules in 6 steps, is still the most widely used form of stemming (*References [2]*).
5. Present-day applications drive a wide range of statistical techniques based on stemming, including those using n-grams (a contiguous sequence of n items, either letters or words from a given sequence of text), **hidden Markov models (HMM)**, and context-sensitive stemming.

Local/global dictionary or vocabulary?

Once the preprocessing task of converting documents into tokens is performed, the next step is the creation of a corpus or vocabulary as a single dictionary, using all the tokens from all documents. Alternatively, several dictionaries are created based on category, using specific tokens from fewer documents.

Many applications in topic modeling and text categorization perform well when dictionaries are created per topic/category, which is known as a local dictionary. On the other hand, many applications in document clustering and information extraction perform well when one single global dictionary is created from all the document tokens. The choice of creating one or many specific dictionaries depends on the core NLP task, as well as on computational and storage requirements.

Feature extraction/generation

A key step in converting the document(s) with unstructured text is to transform them into datasets with structured features, similar to what we have seen so far in Machine Learning datasets. Extracting features from text so that it can be used in Machine Learning tasks such as supervised, unsupervised, and semi-supervised learning depends on many factors, such as the goals of the applications, domain-specific requirements, and feasibility. There are a wide variety of features, such as words, phrases, sentences, POS-tagged words, typographical elements, and so on, that can be extracted from any document. We will give a broad range of features that are commonly used in different Machine Learning applications.

Lexical features

Lexical features are the most frequently used features in text mining applications. Lexical features form the basis for the next level of features. They are the simple character- or word- level features constructed without trying to capture information about intent or the various meanings associated with the text. Lexical features can be further broken down into character-based features, word-based features, part-of-speech features, and taxonomies, for example. In the next section, we will describe some of them in greater detail.

Character-based features

Individual characters (unigram) or a sequence of characters (n-gram) are the simplest forms of features that can be constructed from the text document. The bag of characters or unigram characters have no positional information, while higher order n-grams capture some amount of context and positional information. These features can be encoded or given numeric values in different ways, such as binary 0/1 values, or counts, for example, as discussed later in the next section.

Let us consider the memorable Dr. Seuss rhyme as the text content—"the Cat in the Hat steps onto the mat". While the bag-of-characters (1-gram or unigram features) will generate unique characters {"t", "h", "e", "c", "a", "i", "n", "s", "p", "o", "n", "m"} as features, the 3-gram features are {"\sCa", "\sHa", "\sin", "\sma", "\son", "\sst", "\sth", "Cat", "Hat", "at\s", "e\sC", "e\sH", "e\sm", "eps", "he\s", "in\s", "mat", "n\st", "nto", "o\st", "ont", "ps\s", "s\so", "ste", "t\si", "t\ss", "tep", "the", "to\s"}. As can be seen, as "n" increases, the number of features increases exponentially and soon becomes unwieldy. The advantage of n-grams is that at the cost of increasing

the total number of features, the assembled features often seem to capture combinations of characters that are more interesting than the individual characters themselves.

Word-based features

Instead of generating features from characters, features can similarly be constructed from words in a unigram and n-gram manner. These are the most popular feature generation techniques. The unigram or 1-word token is also known as the bag of words model. So the example of "the Cat in the Hat steps onto the mat" when considered as unigram features is {"the", "Cat", "in", "Hat", "steps", "onto", "mat"}. Similarly, bigram features on the same text would result in {"the Cat", "Cat in", "in the", "the Hat", "Hat step", "steps onto", "onto the", "the mat"}. As in the case of character-based features, by going to higher "n" in the n-grams, the number of features increases, but so does the ability to capture word sense via context.

Part-of-speech tagging features

The input is text with words and the output is text where every word is associated with the grammatical tag. In many applications, part-of-speech gives a context and is useful in identification of named entities, phrases, entity disambiguation, and so on. In the example "the Cat in the Hat steps onto the mat" , the output is {"the\Det", "Cat\Noun", "in\Prep", "the\Det", "Hat\Noun", "steps\Verb", "onto\Prep", "the\Det", "mat\Noun"}. Language specific rule-based taggers or Markov chain-based probabilistic taggers are often used in this process.

Taxonomy features

Creating taxonomies from the text data and using it to understand relationships between words is also useful in different contexts. Various taxonomical features such as hypernyms, hyponyms, is-member, member-of, is-part, part-of, antonyms, synonyms, acronyms, and so on, give lexical context that proves useful in searches, retrieval, and matching in many text mining scenarios.

Syntactic features

The next level of features that are higher than just characters or words in text documents are the syntax-based features. Syntactical representation of sentences in text is generally in the form of syntax trees. Syntax trees capture nodes as terms used in sentences, and relationships between the nodes are captured as links. Syntactic features can also capture more complex features about sentences and usage—such as

aggregates—that can be used for Machine Learning. It can also capture statistics about syntax trees—such as sentences being left-heavy, right-heavy, or balanced—that can be used to understand signatures of different content or writers.

Two sentences can have the same characters and words in the lexical analysis, but their syntax trees or intent could be completely different. Breaking the sentences in the text into different phrases—**Noun Phrase (NP)**, **Prepositional Phrase (PP)**, **Verbal (or Gerund) Phrase (VP)**, and so on—and capturing phrase structure trees for the sentences are part of this processing task. The following is the syntactic parse tree for our example sentence:

```
(S (NP (NP the cat)
        (PP in
            (NP the hat)))
    (VP steps
        (PP onto
            (NP the mat))))
```

Syntactic Language Models (SLM) are about determining the probability of a sequence of terms. Language model features are used in machine translation, spelling correction, speech translation, summarization, and so on, to name a few of the applications. Language models can additionally use parse trees and syntax trees in their computation as well.

The chain rule is applied to compute the joint probability of terms in the sentence:

$$P(w_1, w_2, \dots, w_n) = \prod_i P(w_i | w_1, w_2, \dots, w_{i-1})$$

In the example "the cat in the hat steps onto the mat":

$$\begin{aligned} P(\text{the}, \text{cat}, \text{in}, \text{the}, \text{hat}, \text{steps}, \text{onto}, \text{the}, \text{mat}) \\ = P(\text{the}) \times P(\text{cat} \setminus \text{the}) \times P(\text{in} \setminus \text{the cat}) \dots \\ \times P(\text{mat} \setminus \text{the, cat, in, the, hat, steps, onto, the})) \end{aligned}$$

Generally, the estimation of the probability of long sentences based on counts using any corpus is difficult due to the need for many examples of such sentences. Most language models use the Markov assumption of independence and n-grams (2-5 words) in practical implementations (*References [8]*).

Semantic features

Semantic features attempt to capture the "meaning" of the text, which is then used for different applications of text mining. One of the simplest forms of semantic features is the process of adding annotations to the documents. These annotations, or metadata, can have additional information that describes or captures the intent of the text or documents. Adding tags using collaborative tagging to capture tags as keywords describing the text is a common semantic feature generation process.

Another form of semantic feature generation is the process of ontological representation of the text. Generic and domain specific ontologies that capture different relationships between objects are available in knowledge bases and have well-known specifications, such as Semantic Web 2.0. These ontological features help in deriving complex inferencing, summarization, classification, and clustering tasks in text mining. The terms in the text or documents can be mapped to "concepts" in ontologies and stored in knowledge bases. These concepts in ontologies have semantic properties, and are related to other concepts in a number of ways, such as generalization/specialization, member-of/isAMember, association, and so on, to name a few. These attributes or properties of concepts and relationships can be further used in search, retrieval, and even in predictive modeling. Many semantic features use the lexical and syntactic processes as pre-cursors to the semantic process and use the outputs, such as nouns, to map to concepts in ontologies, for example. Adding the concepts to an existing ontology or annotating it with more concepts makes the structure more suitable for learning. For example, in the "the cat in the .." sentence, "cat" has properties such as {age, eats, ...} and has different relationships, such as { "isA Mammal", "hasChild", "hasParent", and so on}.

Feature representation and similarity

Lexical, syntactic, and semantic features, described in the last section, often have representations that are completely different from each other. Representations of the same feature type, that is, lexical, syntactic, or semantic, can differ based on the computation or mining task for which they are employed. In this section, we will describe the most common lexical feature-based representation known as vector space models.

Vector space model

The **vector space model (VSM)** is a transformation of the unstructured document to a numeric vector representation where terms in the corpus form the dimensions of the vector and we use some numeric way of associating value with these dimensions.

As discussed in the section on dictionaries, a corpus is formed out of unique words and phrases from the entire collection of documents in a domain or within local sub-categories of one. Each of the elements of such a dictionary are the dimensions of the vector. The terms—which can be single words or phrases, as in n-grams—form the dimensions and can have different values associated with them in a given text/document. The goal is to capture the values in the dimensions in a way that reflects the relevancy of the term(s) in the entire corpus (*References [11]*). Thus, each document or file is represented as a high-dimensional numeric vector. Due to the sparsity of terms, the numeric vector representation has a sparse representation in numeric space. Next, we will give some well-known ways of associating values to these terms.

Binary

This is the simplest form of associating value to the terms, or dimensions. In binary form, each term in the corpus is given a 0 or 1 value based on the presence or absence of the term in the document. For example, consider the following three documents:

- Document 1: "The Cat in the Hat steps onto the mat"
- Document 2: "The Cat sat on the Hat"
- Document 3: "The Cat loves to step on the mat"

After preprocessing by removing stop words {on, the, in, onto} and stemming {love/loves, steps/step} using a unigram or bag of words, {cat, hat, step, mat, sat,

`love`} are the features of the corpus. Each document is now represented in a binary vector space model as follows:

Terms	cat	hat	step	mat	sat	love
Document 1	1	1	1	1	0	0
Document 2	1	1	0	0	1	0
Document 3	1	0	1	1	0	1

Term frequency (TF)

In **term frequency (TF)**, as the name suggests, the frequency of terms in the entire document forms the numeric value of the feature. The basic assumption is that the higher the frequency of the term, the greater the relevance of that term for the document. Counts of terms or normalized counts of terms are used as values in each column of terms:

$$tf(t) = \text{count}(D, t)$$

The following table gives term frequencies for the three documents in our example:

TF / Terms	cat	hat	step	mat	sat	love
Document 1	1	1	1	1	0	0
Document 2	1	1	0	0	1	0
Document 3	1	0	1	1	0	1

Inverse document frequency (IDF)

Inverse document frequency (IDF) has various flavors, but the most common way of computing it is using the following:

$$Idf(t) = \log\left(\frac{N}{n_j}\right)$$

Here, $N = \text{total number of documents}$, $n_j = \text{number of documents containing term } j$. IDF favors mostly those terms that occur relatively infrequently in the documents. Some empirically motivated improvements to IDF have also been proposed in the research (*References* [7]).

TF for our example corpus:

Terms	cat	hat	step	mat	sat	love
N/nj	3/3	3/2	3/2	3/2	3/1	3/1
IDF	0.0	0.40	0.40	0.40	1.10	1.10

Term frequency-inverse document frequency (TF-IDF)

Combining both term frequencies and inverse document frequencies in one metric, we get the term frequency-inverse document frequency values. The idea is to value those terms that are relatively uncommon in the corpus (high IDF), but are reasonably relevant for the document (high TF). TF-IDF is the most common form of value association in many text mining processes:

$$tfIdf(t) = tf(t) \times Idf(t)$$

This gives us the TF-IDF for all the terms in each of the documents:

TF-IDF/Terms	cat	hat	step	mat	sat	love
Document 1	0.0	0.40	0.40	0.40	1.10	1.10

Document 2	0.0	0.40	0.0	0.0	1.10	0.0
Document 3	0.0	0.0	0.40	0.40	0.0	1.10

Similarity measures

Many techniques in supervised, unsupervised, and semi-supervised learning use "similarity" measures in their underlying algorithms to find similar patterns or to separate different patterns. Similarity measures are tied closely to the representation of the data. In the VSM representation of documents, the vectors are very high dimensional and sparse. This poses a serious issue in most traditional similarity measures for classification, clustering, or information retrieval. Angle-based similarity measures, such as cosine distances or Jaccard coefficients, are more often used in practice. Consider two vectors represented by \mathbf{t}_1 and \mathbf{t}_2 corresponding to two text documents.

Euclidean distance

This is the L2 norm in the feature space of the documents:

$$d_{EUC}(\mathbf{t}_1, \mathbf{t}_2) = \left[(\mathbf{t}_1 - \mathbf{t}_2) \cdot (\mathbf{t}_1 - \mathbf{t}_2) \right]^{1/2}$$

Cosine distance

This angle-based similarity measure considers orientation between vectors only and not their lengths. It is equal to the cosine of the angle between the vectors. Since the vector space model is a positive space, cosine distance varies from 0 (orthogonal, no common terms) to 1 (all terms are common to both, but not necessarily with the same term frequency):

$$S_{cos}(\mathbf{t}_1, \mathbf{t}_2) = \frac{\mathbf{t}_1 \cdot \mathbf{t}_2}{\|\mathbf{t}_1\| \|\mathbf{t}_2\|}$$

Pairwise-adaptive similarity

This measure the distance in a reduced feature space by only considering the features that are most important in the two documents:

$$d_{PAIR}(\mathbf{t}_1, \mathbf{t}_2) = \frac{\mathbf{t}_{1,K} \cdot \mathbf{t}_{2,K}}{\|\mathbf{t}_{1,K}\| \|\mathbf{t}_{2,K}\|}$$

Here, $\mathbf{t}_{i,k}$ is a vector formed from a subset of the features of \mathbf{t}_i ($i = 1, 2$) containing the union of the K largest features appearing in \mathbf{t}_1 and \mathbf{t}_2 .

Extended Jaccard coefficient

This measure is computed as a ratio of the shared terms to the union of the terms between the documents:

$$S_{EJ}(\mathbf{t}_1, \mathbf{t}_2) = \frac{\mathbf{t}_1 \cdot \mathbf{t}_2}{\mathbf{t}_1 \cdot \mathbf{t}_1 + \mathbf{t}_2 \cdot \mathbf{t}_2 - \mathbf{t}_1 \cdot \mathbf{t}_2}$$

Dice coefficient

The Dice coefficient is given by the following:

$$S_{DICE}(\mathbf{t}_1, \mathbf{t}_2) = \frac{2(\mathbf{t}_1 \cdot \mathbf{t}_2)}{\mathbf{t}_1 \cdot \mathbf{t}_1 + \mathbf{t}_2 \cdot \mathbf{t}_2}$$

Feature selection and dimensionality reduction

The goal is the same as in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning* and [Chapter 3](#), *Unsupervised Machine Learning Techniques*. The problem of the curse of dimensionality becomes even more pronounced with text mining and high dimensional features.

Feature selection

Most feature selection techniques are supervised techniques that depend on the labels or the outcomes for scoring the features. In the majority of cases, we perform filter-based rather than wrapper-based feature selection, due to the lower performance cost. Even among filter-based methods, some, such as those involving multivariate techniques such as **Correlation based Feature selection (CFS)**, as described in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning*, can be quite costly or result in suboptimal performance due to high dimensionality (*References [9]*).

Information theoretic techniques

As shown in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning*, filter-based univariate feature selection methods, such as **Information gain (IG)** and **Gain Ratio (GR)**, are most commonly used once preprocessing and feature extraction is done.

In their research, Yang and Pederson (*References [10]*) clearly showed the benefits of feature selection and reduction using IG to remove close to 98% of terms and yet improve the predictive capability of the classifiers.

Many of the information theoretic or entropy-based methods have a stronger influence resulting from the marginal probabilities of the tokens. This can be an issue when the terms have equal conditional probability $P(t|class)$, the rarer terms may have better scores than the common terms.

Statistical-based techniques

?² feature selection is one of the most common statistical-based techniques employed to perform feature selection in text mining. ?² statistics, as shown in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning*, give the independence relationship between the tokens in the text and the classes.

It has been shown that χ^2 statistics for feature selection may not be effective when there are low-frequency terms (*References [19]*).

Frequency-based techniques

Using the term frequency or the document frequency described in the section on feature representation, a threshold can be manually set, and only terms above or below a certain threshold can be allowed used for modeling in either classification or clustering tasks. Note that **term frequency (TF)** and **document frequency (DF)** methods are biased towards common words while some of the information theoretic or statistical-based methods are biased towards less frequent words. The choice of selection of features depends on the domain, the particular application of predictive learning, and more importantly, on how models using these features are evaluated, especially on the unseen dataset.

Dimensionality reduction

Another approach that we saw in [Chapter 3, Unsupervised Machine Learning Techniques](#), was to use unsupervised techniques to reduce the features using some form of transformation to decide their usefulness.

Principal component analysis (PCA) computes a covariance or correlation matrix from the document-term matrix. It transforms the data into linear combinations of terms in the inputs in such a way that the transformed combination of features or terms has higher discriminating power than the input terms. PCA with cut-off or thresholding on the transformed features, as shown in [Chapter 3, Unsupervised Machine Learning Techniques](#), can bring down the dimensionality substantially and even improve or give comparable performance to the high dimensional input space. The only issue with using PCA is that the transformed features are not interpretable, and for domains where understanding which terms or combinations yield better predictive models, this technique has some limitations.

Latent semantic analysis (LSA) is another way of using the input matrix constructed from terms and documents and transforming it into lower dimensions with latent concepts discovered through combinations of terms used in documents (*References [5]*). The following figure captures the process using the **singular value decomposition (SVD)** method for factorizing the input document-term matrix:

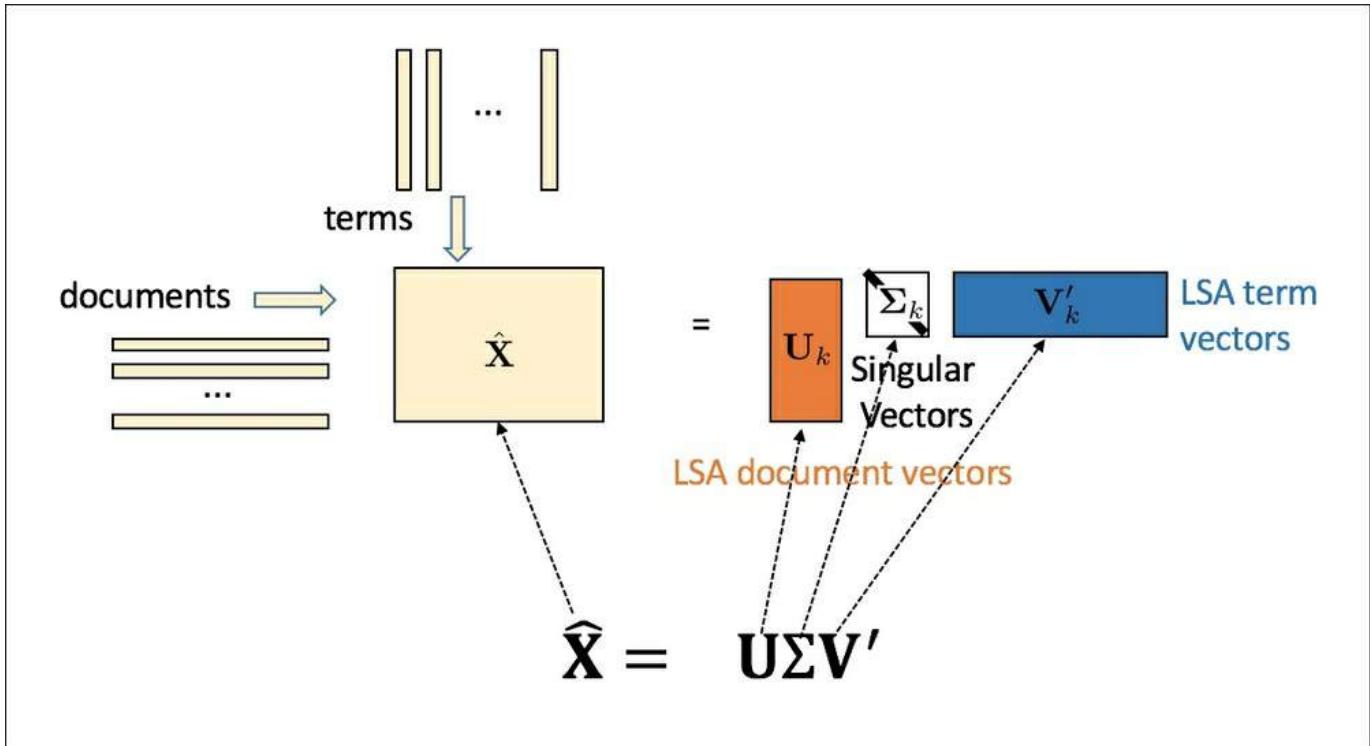


Figure 12: SVD factorization of input document-terms into LSA document vectors and LSA term vectors

LSA has been shown to be a very effective way of reducing the dimensions and also of improving the predictive performance in models. The disadvantage of LSA is that storage of both vectors U and V is needed for performing retrievals or queries. Determining the lower dimension k is hard and needs some heuristics similar to k-means discussed in [Chapter 3, Unsupervised Machine Learning Techniques](#).

Topics in text mining

As we saw in the first section, the area of text mining and performing Machine Learning on text spans a wide range of topics. Each topic discussed has some customizations to the mainstream algorithms, or there are specific algorithms that have been developed to perform the task called for in that area. We have chosen four broad topics, namely, text categorization, topic modeling, text clustering, and named entity recognition, and will discuss each in some detail.

Text categorization/classification

The text classification problem manifests itself in different applications, such as document filtering and organization, information retrieval, opinion and sentiment mining, e-mail spam filtering, and so on. Similar to the classification problem discussed in [Chapter 2, Practical Approach to Real-World Supervised Learning](#), the general idea is to train on the training data with labels and to predict the labels of unseen documents.

As discussed in the previous section, the preprocessing steps help to transform the unstructured document collection into well-known numeric or categorical/binary structured data arranged in terms of a document-term matrix. The choice of performing some preprocessing steps, such as stemming or customizing stop words, depends on the data and applications. The feature choice is generally basic lexical features, n-grams of words as terms, and only in certain cases do we use the entire text as a string without breaking it into terms or tokens. It is common to use binary feature representation or frequency-based representation for document-term structured data. Once this transformation is complete, we do feature selection using univariate analysis, such as information gain or chi-square, to choose discriminating features above certain thresholds of scores. One may also perform feature transformation and dimensionality reduction such as PCA or LSA in many applications.

There is a wide range in the choice of classifiers once we get structured data from the preceding process. In the research as well as in commercial applications, we see the use of most of the common modeling techniques, including linear (linear regression, logistic regression, and so on), non-linear (SVM, neural networks, KNN), generative (naïve bayes, bayesian networks), interpretable (decision trees, rules), and ensemble-based (bagging, boosting, random forest) classifiers. Many algorithms use similarity or distance metrics, of which cosine distance is the most popular choice. In certain classifiers, such as SVM, the string representation of the document can be used as is, with the right choice of string kernels and similarity-based metrics on strings to compute the dot products.

Validation and evaluation methods are similar to supervised classification methodologies—splitting the data into train/validation/test, training on training data, tuning parameters of algorithm(s) on validation data, and estimating the performance of the models on hold-out or test data.

Since most of text classification involves a large number of documents, and the target classes are rare, the metrics used for evaluation, tuning, or choosing algorithms are in most cases precision, recall, and F-score measure, as follows:

$$precision(p) = \frac{Count(\{relevant \cap retrieved\})}{Count(retrieved)}$$

$$recall(r) = \frac{Count(\{relevant \cap retrieved\})}{Count(relevant)}$$

$$F-score = \frac{2}{1/precision + 1/recall}$$

Topic modeling

A topic is a distribution over a fixed vocabulary. Topic modeling can be defined as an ability to capture different core ideas or themes in various documents. This has a wide range of applications, such as the summarization of documents, understanding reasons for sentiments, trends, the news, and many others. The following figure shows how topic modeling can discern a user-specified number k of topics from a corpus and then, for every document, assign proportions representing how much of each topic is found in the document:

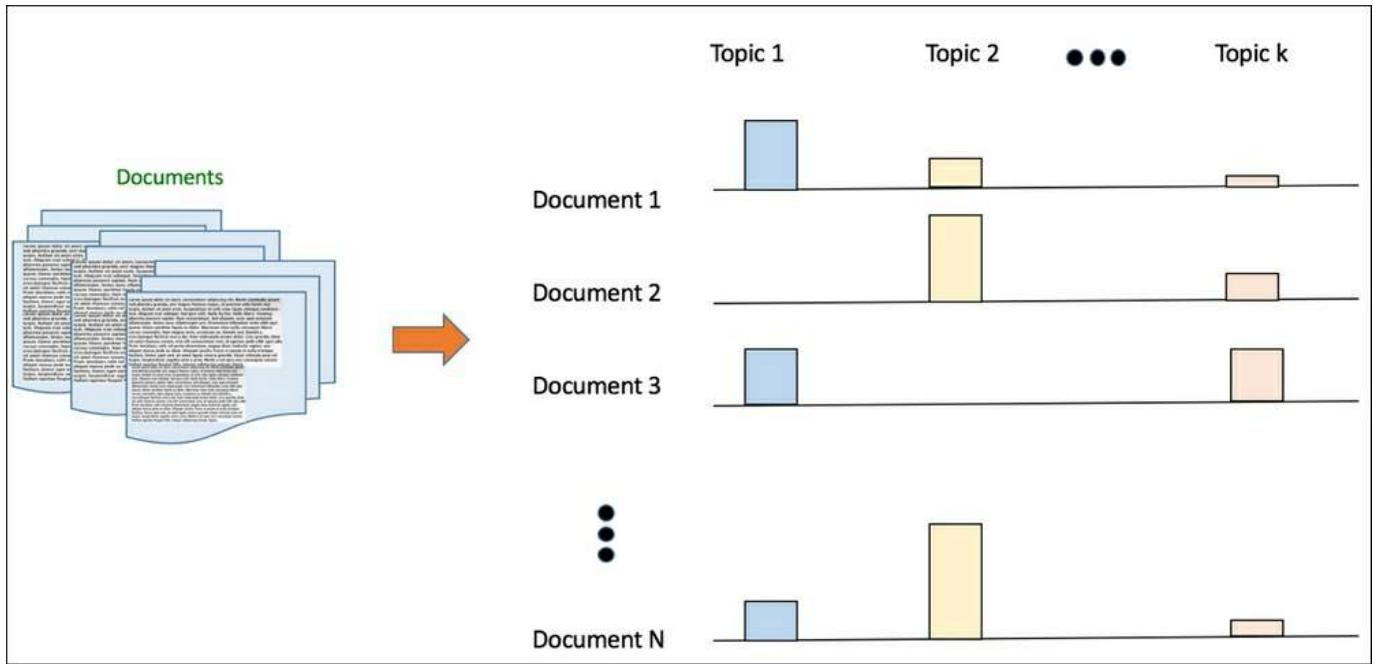


Figure 13: Probabilistic topic weights assignment for documents

There are quite a few techniques for performing topic modeling using supervised and unsupervised learning in the literature (*References [13]*). We will discuss the most common technique known as **probabilistic latent semantic index (PLSI)**.

Probabilistic latent semantic analysis (PLSA)

The idea of PLSA, as in the LSA for feature reduction, is to find latent concepts hidden in the corpus by discovering the association between co-occurring terms and treating the documents as mixtures of these concepts. This is an unsupervised technique, similar to dimensionality reduction, but the idea is to use it to model the

mixture of topics or latent concepts in the document (*References* [12]).

As shown in the following figure, the model may associate terms occurring together often in the corpus with a latent concept, and each document can then be said to exhibit that topic to a smaller or larger extent:

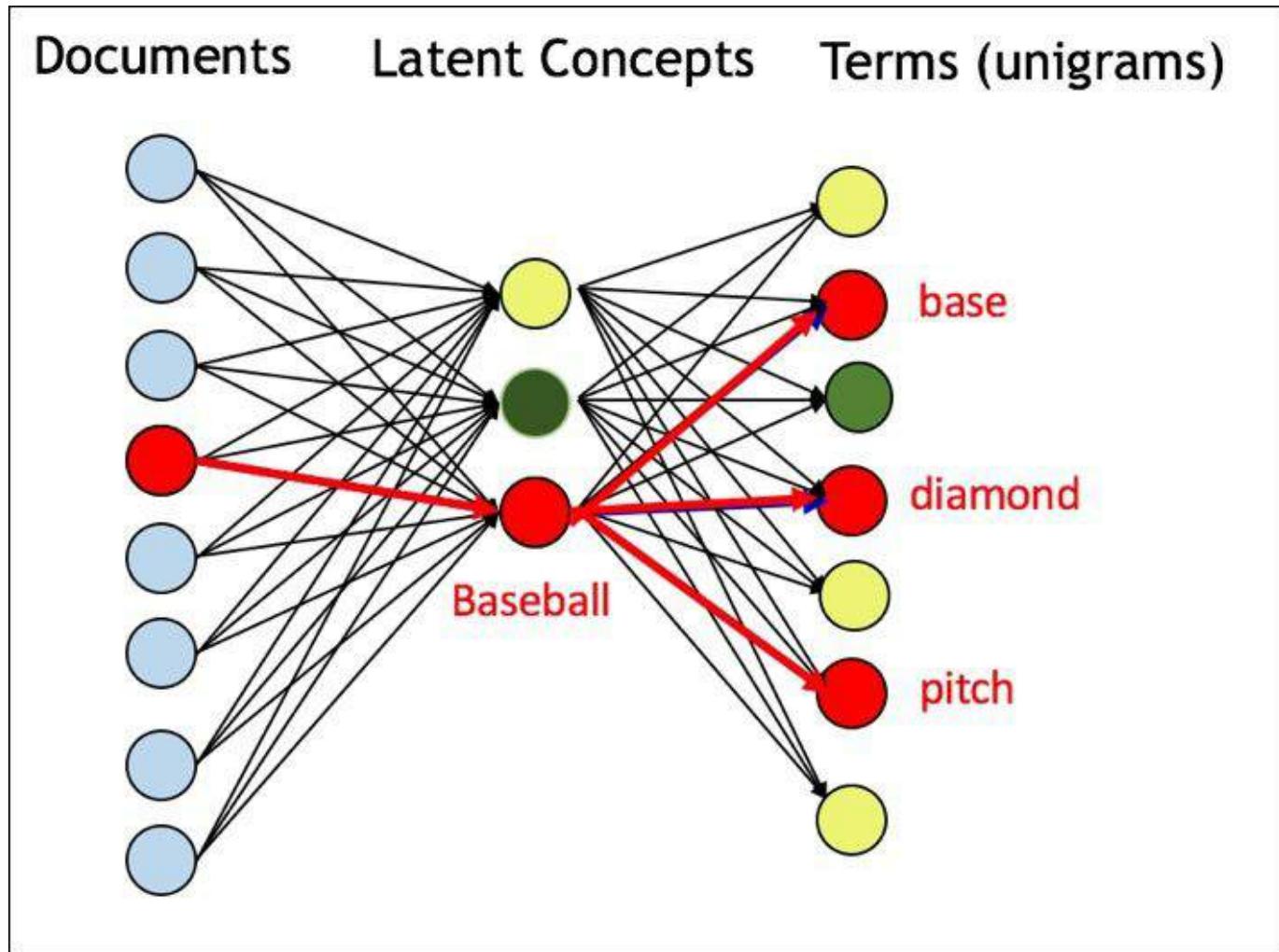


Figure 14: Latent concept of Baseball capturing the association between documents and related terms

Input and output

The inputs are:

- A collection of documents following a certain format and structure. We will give the notation:

$$c = \{d_1, d_2, \dots, d_n\}$$

- The number of topics that need to be modeled or discovered as k .

The output is:

- k topics identified $T = \{T_1, T_2, \dots, T_k\}$.
- For each document, coverage of the topic given in the document d_i can be written as $\{p_{i1}, p_{i2}, \dots, p_{ik}\}$, where p_{ij} is the probability of the document d_i covering the topic T_j .

How does it work?

Implementations of PLSA generally follow the steps described here:

1. Basic preprocessing steps as discussed previously, such as tokenization, stop words removal, stemming, dictionary of words formation, feature extraction (unigrams or n-grams, and so on), and feature selection (unsupervised techniques) are carried out, if necessary.
2. The problem can be reduced to estimating the distribution of terms in a document, and, given the distribution, choosing the topic based on the maximum terms corresponding to the topic.
3. Introducing a "latent variable" z helps us to select whether the term belongs to a topic. Note that z is not "observed", but we assume that it is related to picking the term from the topic. Thus, the probability of the term t given the document d can be expressed in terms of this latent variable as:

$$p(t | d) = \sum_z p(t | z) p(z | d)$$

4. By using two sets of variables (z , θ) the equation can be written as:

$$p(t | d) = \sum_z p(t | z; \theta) p(d; \pi)$$

Here, $p(t|z; \theta)$ is the probability of latent concepts in terms and $p(z|d; \pi)$ is the probability of latent concepts in document-specific mixtures.

5. Using log-likelihood to estimate the parameters to maximize:

$$l(\theta, \pi; N) = \sum_{d,t} \text{count}(d, t) \log \left(\sum_z p(t|z; \theta) p(z|d; \pi) \right)$$

6. Since this equation involves nonconvex optimization, the EM algorithm is often used to find the parameters iteratively until convergence is reached or the total number of iterations are completed (*References [6]*):

1. The E-step of the EM algorithm is used to determine the posterior probability of the latent concepts. The probability of term t occurring in the document d , can be explained by the latent concept z as:

$$p(z|d, t) = \frac{p(z|d; \pi) p(t|z; \theta)}{\sum_{z'} p(t|z'; \theta) p(z'|d; \pi)}$$

2. The M-step of the EM algorithm uses the values obtained from the E-step, that is, $p(z|d, t)$ and does parameter estimation as:

$$p(t|z; \theta) \propto \sum_d \text{count}(d, t) p(z|d, t)$$

3. $\sum_d \text{count}(d, t) p(z|d, t)$ = how often the term t is associated with concept z :

$$p(z|d, t) \propto \sum_t \text{count}(d, t) p(z|d, t)$$

4. $\sum_t \text{count}(d, t) p(z | d, t)$ = how often document d is associated with concept z .

Advantages and limitations

The advantages and limitations are as follows:

- Though widely used, PLSA has some drawbacks that have been overcome by more recent techniques.
- The unsupervised nature of the algorithm and its general applicability allows it to be used in a wide variety of similar text mining applications, such as clustering documents, associating topics related to authors/time, and so on.
- PLSA with EM algorithms, as discussed in previous chapters, face the problem of getting "stuck in local optima", unlike other global algorithms, such as evolutionary algorithms.
- PLSA algorithms can only do topic identification in known documents, but cannot do any predictive modeling. PLSA has been generalized and is known as **latent dirichlet allocation (LDA)** to overcome this (*References [14]*).

Text clustering

The goal of clustering, as seen in [Chapter 3](#), *Unsupervised Machine Learning Techniques*, is to find groups of data, text, or documents that are similar to one another within the group. The granularity of unstructured data can vary from small phrases or sentences, paragraphs, and passages of text to a collection of documents. Text clustering finds its application in many domains, such as information retrieval, summarization, topic modeling, and document classification in unsupervised situations, to name a few. Traditional techniques in clustering can be employed once the unstructured text data is transformed into structured data via preprocessing. The difficulty with traditional clustering techniques is the high-dimensional and sparse nature of the dataset obtained using the transformed document-term matrix representation. Many traditional clustering algorithms work only on numeric values of features. Because of this constraint, categorical or binary representation of terms cannot be used and often TF or TF-IDF are used for representation of the document-term matrix.

In this section, we will discuss some of the basic processes and techniques in clustering. We will start with pre-processing and transformations and then discuss some techniques that are widely used and the modifications made to them.

Feature transformation, selection, and reduction

Most of the pre-processing steps discussed in this section are normally used to get either unigram or n-gram representation of terms in documents. Dimensionality reduction techniques, such as LSA, are often employed to transform the features into smaller latent space.

Clustering techniques

The techniques for clustering in text include probabilistic models, as well as those that use distance-based methods, which are familiar to us from when we learned about structured data. We will also discuss **Non-negative Matrix Factorization (NNMF)** as an effective technique with good performance and interpretability.

Generative probabilistic models

There is commonality between topic modeling and text clustering in generative methods. As shown in the following figure, clustering associates a document with a

single cluster (generally), compared to topic modeling where each document can have a probability of coverage in multiple topics. Every word in topic modeling can be generated by multiple topics in an independent manner, whereas in clustering all the words are generated from the same cluster:

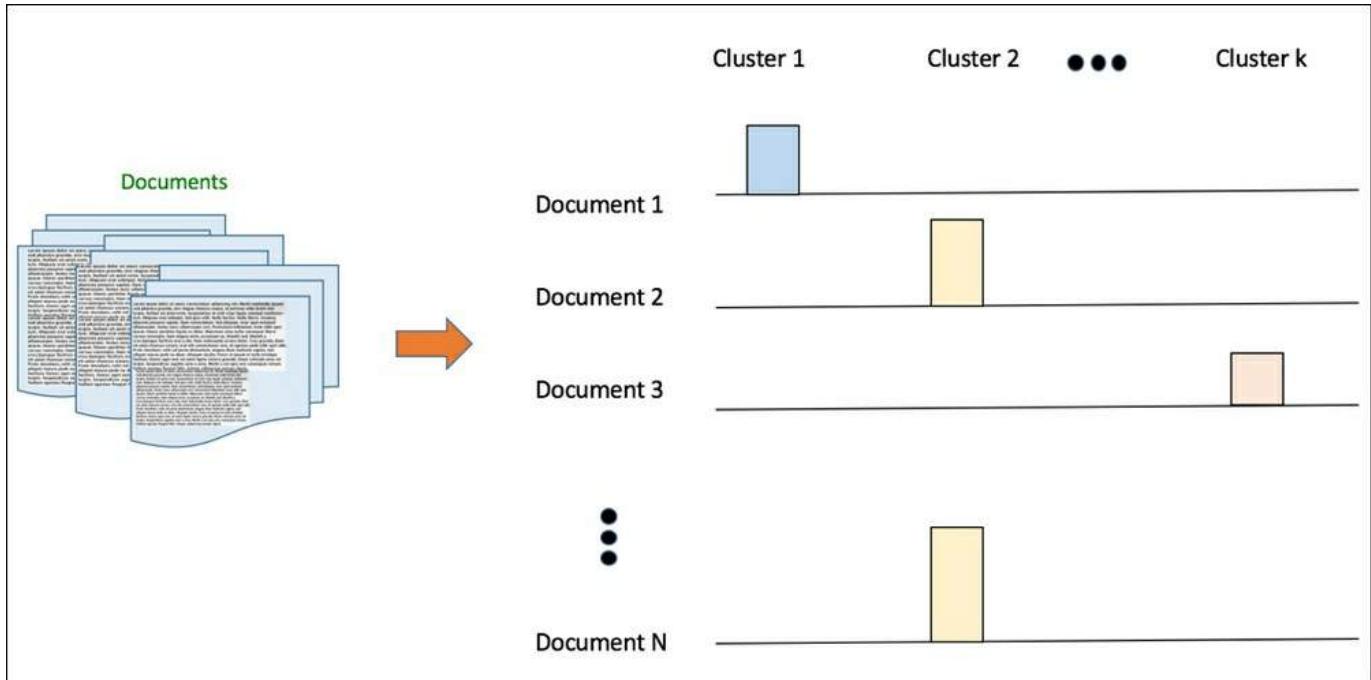


Figure 15: Exclusive mapping of documents to K-Clusters

Mathematically, this can be explained using two topics $T = \{T_1, T_2\}$ and two clusters $c = \{c_1, c_2\}$.

In clustering, the likelihood of the document can be given as:

$$p(d) = p(c_1)p(d|c_1) + p(c_2)p(d|c_2)$$

If the document has, say, L terms, this can be further expanded as:

$$p(d) = p(t_1, t_2 \dots t_L) = p(c_1) \prod_{i=1}^L p(t_i | c_1) + p(c_2) \prod_{i=1}^L p(t_i | c_2)$$

Thus, once you "assume" a cluster, all the words come from that cluster. The product of all terms is computed, followed by the summation across all clusters.

In topic modeling, the likelihood of the document can be given as:

$$p(d) = p(t_1, t_2 \dots t_L) = \prod_{i=1}^L [p(T_1) p(t_i | T_1) + p(T_2) p(t_i | T_2)]$$

Thus, each term t_i can be picked independently from topics and hence summation is done inside and the product is done outside.

Input and output

The inputs are:

- A collection of documents following a certain format and structure expressed with the following notation:

$$D = \{d_1, d_2, \dots, d_n\}$$

- The number of clusters that need to be modeled or discovered as k .

The output is:

- k clusters identified $c = \{c_1, c_2, \dots, c_k\}$.
- For each document, $p(d_i)$ is mapped to one of the clusters k .

How does it work?

Here are the steps:

1. Basic preprocessing steps as discussed previously, such as tokenization, stop words removal, stemming, dictionary of words formation, feature extraction

(unigrams or n-grams, and so on) of terms, feature transformations (LSA), and even feature selection. Let t be the terms in the final feature set; they correspond to the dictionary or vocabulary \mathbb{V} .

2. Similar to PLSA, we introduce a "latent variable", z , which helps us to select whether the document belonging to the cluster falls in the range of $z = \{1, 2, \dots, k\}$ corresponding to the clusters. Let the γ parameter be the parameter we estimate for each latent variable such that $p(\gamma_i)$ corresponds to the probability of cluster $z = i$.
3. The probability of a document belonging to a cluster is given by $p(\gamma_i)$, and every term in the document generated from that cluster is given by $p(t|\gamma_i)$. The likelihood equation can be written as:

$$p(d | \Lambda) = \sum_{i=1}^k p(\theta_i) \prod_{t \in \text{vp}} (t | \theta_i)^{\text{count}(t,d)} ; \Lambda = \{\theta_i | i = 1..k\}$$

Note that instead of going through the documents, it is rewritten with terms t in the vocabulary \mathbb{V} raised to the number of times that term appears in the document.

4. Perform the EM algorithm in a similar way to the method we used previously to estimate the parameters as follows:

1. The E-step of the EM algorithm is used to infer the cluster from which the document was generated:

$$p(z_d = i | d) \propto p(\theta_i) \prod_{t \in \mathbb{V}} p(t | \theta_i)^{\text{count}(t,d)} ; \sum_{i=1}^k p(z_d = i | d) = 1$$

2. The M-step of the EM algorithm is used to re-estimate the parameters using the result of the E-step, as shown here:

$$p(\theta_i) \alpha \sum_{j=1}^N p(z_{dj} = i | d_j) ; \sum_{i=1}^k p(\theta_i) = 1$$

$$p(t | \theta_i) \propto \sum_{j=1}^N count(t, d_j) p(z_{d_j} = i | d_j); \sum_{t \in \mathbb{V}} p(t | \theta_i) = 1$$

5. The final probability estimate for each document can be done using either the maximum likelihood or by using a Bayesian algorithm with prior probabilities,

as shown here: $c_d = \operatorname{argmax}_i (p(d | \theta_i))$ or

$$c_d = \operatorname{argmax}_i (p(d | \theta_i) p(\theta_i))$$

Advantages and limitations

- Generative-based models have similar advantages to LSA and PLSA, where we get a probabilistic score for documents in clusters. By applying domain knowledge or priors using cluster size, the assignments can be further fine-tuned.
- The disadvantages of the EM algorithm having to do with getting stuck in local optima and being sensitive to the starting point are still true here.

Distance-based text clustering

Most distance-based clustering algorithms rely on the similarity or the distance measure used to determine how far apart instances are from each other in feature space. Normally in datasets with numeric values, Euclidean distance or its variations work very well. In text mining, even after converting unstructured text to structured features of terms with numeric values, it has been found that the cosine and Jaccard similarity functions perform better.

Often, Agglomerative or Hierarchical clustering, discussed in [Chapter 3, Unsupervised Machine Learning Techniques](#), are used, which can merge documents based on similarity, as discussed previously. Merging the documents or groups is often done using single linkage, group average linkage, and complete linkage techniques. Agglomerative clustering also results in a structure that can be used for information retrieval and the searching of documents.

The partition-based clustering techniques k-means and k-medoids accompanied by h a suitable similarity or distance method are also employed. The issue with k-means, as indicated in the discussion on clustering techniques, is the sensitivity to starting

conditions along with computation space and time. k-medoids are sensitive to the sparse data structure and also have computation space and time constraints.

Non-negative matrix factorization (NMF)

Non-negative matrix factorization is another technique used to factorize a large data-feature matrix into two non-negative matrices, which not only perform the dimensionality reduction, but are also easier to inspect. NMF has gained popularity for document clustering, and many variants of NMF with different optimization functions have now been shown to be very effective in clustering text (*References [15]*).

Input and output

The inputs are:

- A collection of documents following a certain format and structure given by the notation:

$$D = \{d_1, d_2, \dots, d_n\}$$

- Number of clusters that need to be modeled or discovered as k .

The output is:

- k clusters identified $c = \{c_1, c_2, \dots, c_k\}$ with documents assigned to the clusters.

How does it work?

The mathematical details and interpretation of NMF are given in the following:

1. The basic idea behind NMF is to factorize the input matrix using low-rank approximation, as follows:

$$\mathbf{A} \cong \mathbf{A}_k = \mathbf{W}_k \mathbf{H}_k$$

2. A non-linear optimization function is used as:

$$\min \|\mathbf{A} - \mathbf{WH}\|; \mathbf{W}, \mathbf{H} \geq \mathbf{0}$$

This is convex in W or H , but not in both, resulting in no guarantees of a global minimum. Various algorithms that use constrained least squares, such as mean-square error and gradient descent, are used to solve the optimization function.

3. The interpretation of NMF, especially in understanding the latent topics based on terms, makes it very useful. The input $A_{m \times n}$ of terms and documents, can be represented in low rank approximation as $W_{m \times k} H_{k \times n}$ matrices, where $W_{m \times k}$ is the term-topic representation whose columns are NMF basis vectors. The non zero elements of column 1 of W , given by W_1 , correspond to particular terms. Thus, the w_{ij} can be interpreted as a basis vector W_i about the terms j . The H_{i1} can be interpreted as how much the document given by doc 1 has affinity towards the direction of the topic vector W_i .
4. From the paper (*References [18]*) it was clearly shown how the basis vectors obtained for the medical abstracts, known as the Medlars dataset, creates highly interpretable basis vectors. The highest weighted terms in these basis vectors directly correspond to the concept, for example, W_1 corresponds to the topic related to "heart" and W_5 is related to "developmental disability".

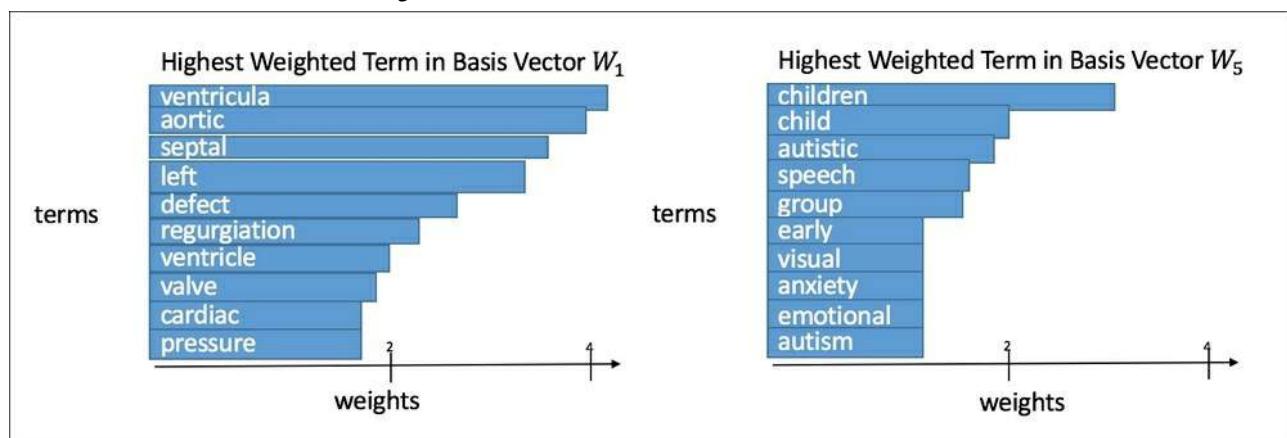


Figure 16: From Langville et al (2006) showing some basis vectors for medical datasets for interpretability.

Advantages and limitations

NMF has been shown to be almost equal in performance with top algorithms, such as LSI, for information retrieval and queries:

- Scalability, computation, and storage is better in NMF than in LSA or LSI using

SVD.

- NMF has a problem with optimization not being global and getting stuck in local minima.
- NMF generation of factors depends on the algorithms for optimization and the parameters chosen, and is not unique.

Evaluation of text clustering

In the case of labeled datasets, all the external measures discussed in [Chapter 3](#), *Unsupervised Machine Learning Techniques*, such as F-measure and Rand Index are useful in evaluating the clustering techniques. When the dataset doesn't have labels, some of the techniques described as the internal measures, such as the Davies–Bouldin Index, R-Squared, and Silhouette Index, can be used.

The general good practice is to adapt and make sure similarity between the documents, as discussed in this section, is used for measuring closeness, remoteness, and spread of the cluster when applied to text mining data. Similarly usage depends on the algorithm and some relevance to the problem too. In distance-based partition algorithms, the similarity of the document can be computed with the mean vector or the centroid. In hierarchical algorithms, similarity can be computed with most similar or dissimilar documents in the group.

Named entity recognition

Named entity recognition (NER) is one of the most important topics in information retrieval for text mining. Many complex mining tasks, such as the identification of relations, annotations of events, and correlation between entities, use NER as the initial component or basic preprocessing step.

Historically, manual rules-based and regular expression-based techniques were used for entity recognition. These manual rules relied on basic pre processing, using POS tags as features, along with hand-engineered features, such as the presence of capital words, usage of punctuations prior to words, and so on.

Statistical learning-based techniques are now used more for NER and its variants. NER can be mapped to sequence labeling and prediction problems in Machine Learning. BIO notation, where each entity type T has two labels B-T and I-T corresponding to beginning and intermediate, respectively, is labeled, and learning involves finding the pattern and predicting it in unseen data. The O represents an outside or unrelated entity in the sequence of text. The entity type T is further classified into Person, Organization, Data, and location in the most basic form.

In this section, we will discuss the two most common algorithms used: generative-based hidden Markov models and discriminative-based maximum entropy models.

Though we are discussing these algorithms in the context of Named Entity Recognition, the same algorithms and processes can be used for other NLP tasks such as POS Tagging, where tags are associated with a sequence rather than associating the NER classes.

Hidden Markov models for NER

Hidden Markov models, as explained in [Chapter 6, Probabilistic Graph Modeling](#), are the sequence-based generative models that assume an underlying distribution that generates the sequences. The training data obtained by labeling sequences with the right NER classes can be used to learn the distribution and parameters, so that for unseen future sequences, effective predictions can be performed.

Input and output

The training data consists of text sequences $x = \{x_1, x_2, \dots, x_n\}$ where each x_i is a

word in the text sequence and labels for each word are available as $y = \{y_1, y_2, \dots, y_n\}$. The algorithm generates a model so that on testing on unseen data, the labels for new sequences can be generated.

How does it work?

1. In the simplest form, a Markov assumption is made, which is that the hidden states and labels of the sequences are only dependent on the previous state. An adaptation to the sequence of words with labels is shown in the following figure:

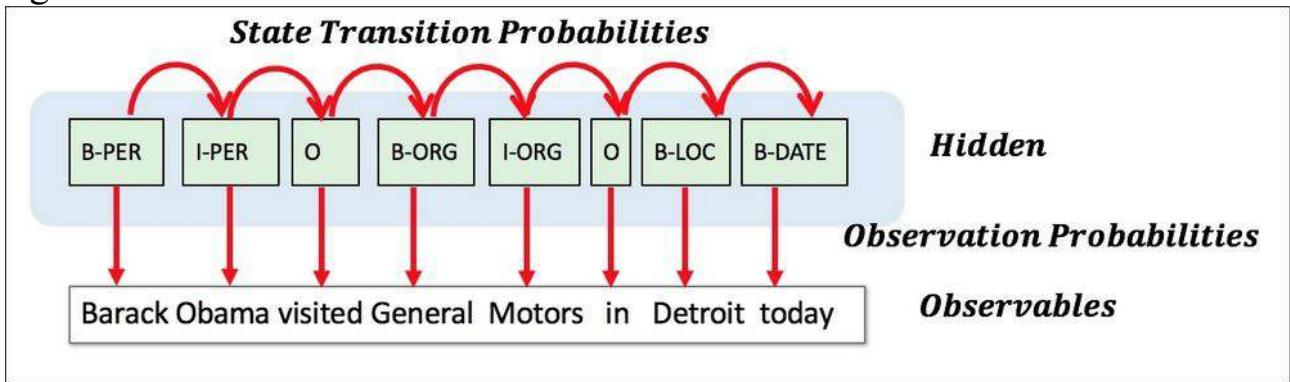


Figure 17: Text sequence and labels corresponding to NER in Hidden Markov Chain

2. The HMM formulation of the sequence classification helps in estimating the joint probability maximized on training data:

$$p(\mathbf{x}, \mathbf{y}) = \prod_i p(x_i | y_i) p(y_i | y_{i-1})$$

3. Each y_i is assumed to be generated based on y_{i-1} and x_i . The first word in the entity is generated conditioned on current and previous labels, that is, y_i and y_{i-1} . If the instance is already a Named entity, then conditioning is only on previous instances, that is, x_{i-1} . Outside words such as "visited" and "in" are considered "not a name class".
4. The HMM formulation with the forward-backward algorithm can be used to determine the likelihood of a sequence of observations with parameters learned

from the training data.

Advantages and limitations

The advantages and limitations are as follows:

- HMMs are good for short sequences, as shown, with one word or term and independence assumption. For sequences with entities that have a longer span, the results will violate these assumptions.
- The HMM needs a large set of training data to estimate the parameters.

Maximum entropy Markov models for NER

Maximum entropy Markov model (MEMM) is a popular NER technique that uses the concept of Markov chains and maximum entropy models to learn and predict the named entities (*References* [16] and [17]).

Input and output

The training data consists of text sequences $x = \{x_1, x_2, \dots, x_n\}$ where each x_i is a word in the text sequence and labels for each word are available as $y = \{y_1, y_2, \dots, y_n\}$. The algorithm generates models so that, on testing on unseen data, the labels for new sequences can be generated.

How does it work?

The following illustrates how the MEMM method is used for learning named entities.

1. The features in MEMM can be word features or other types of features, such as "isWordCapitalized", and so on, which gives it a bit more context and improves performance compared to HMM, where it is only word-based.
2. Next, let us look at a maximum entropy model known as a MaxEnt model, which is an exponential probabilistic model, but which can also be seen as a multinomial logistic regression model. In basic MaxEnt models, given the features $\{f_1, f_2, \dots, f_N\}$ and classes c_1, c_2, \dots, c_C , weights for these features are learned $\{w_{c1}, w_{c2}, \dots, w_{cN}\}$ per class using optimization methods from the training data, and the probability of a particular class can be estimated as:

$$p(c|x) = \frac{\exp\left(\sum_{i=0}^N w_{ci} f_i\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^N w_{c'i} f_i\right)}$$

3. The feature f_i is formally written as $f_i(c, x)$, which means the feature f_i for class c and observation x . The $f_i(c, x)$ is generally binary with values 1/0 in most NER models. Thus, it can be written as:

$$p(c|x) = \frac{\exp\left(\sum_{i=0}^N w_{ci} f_i(c, x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^N w_{c'i} f_i(c, x)\right)}$$

Maximum likelihood based on the probability of prediction across class can be used to select a single class:

$$\hat{c} = \operatorname{argmax}_{c \in C} p(c|x)$$

4. For every word, we use the current word, the features from "nearby" words, and the predictions on the nearby words to create a joint probability model. This is also called local learning as the chunks of test and distribution are learned around local features corresponding to the word.

Mathematically, we see how a discriminative model is created from current word and last prediction as:

$$\hat{c} = \operatorname{argmax}_C p(y|x)$$

$$\hat{c} = \operatorname{argmax}_C \prod_i p(y_i | y_{i-1}, x_i)$$

Generalizing for the k features:

$$\hat{c} = \operatorname{argmax}_C \prod_i p(y_i | y_{i-k}^{i-1}, x_{i-1})$$

5. Thus, in MEMM we compute the probability of the state, which is the class in NER, and even though we condition on prediction of nearby words given by y_{i-1} , in general we can use more features, and that is the advantage over the HMM model discussed previously:

$$p(y_i | x_i) = \prod_i p(y_i | y_{i-1}, x_i)$$

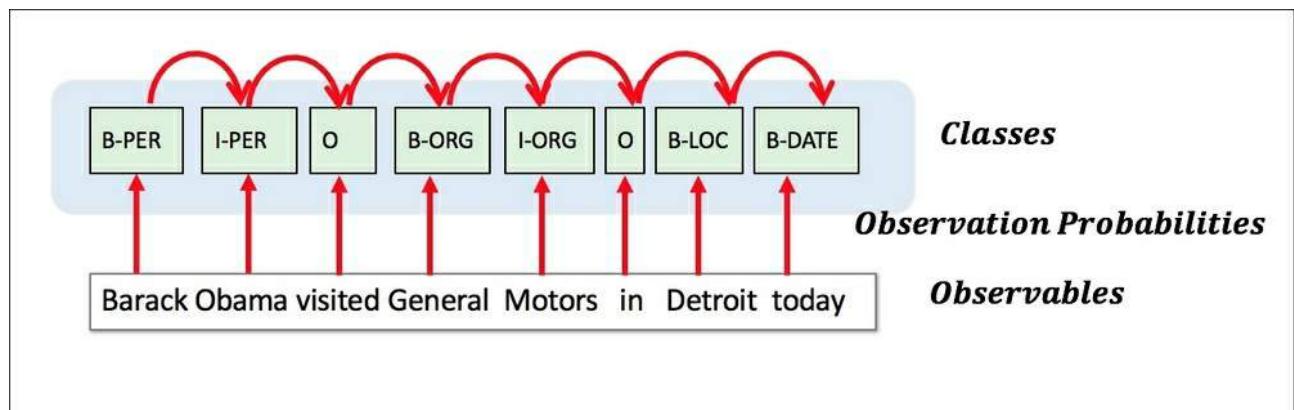


Figure 18 : Text Sequences and observation probabilities with labels

6. The Viterbi algorithm is used to perform the estimation of class for the word or decoding/inferencing in HMM, that is, to get estimates for $p(y_i|y_{i-1}, X_i)$
7. Finally, the MaxEnt model is used to estimate the weights as before using the

optimization methods for state changes in general:

$$p(y_i | y_{i-1}, x_i) = \frac{\exp\left(\sum_{i=0}^N w_{ci} f_i(x_i, y_i, y_{i-1})\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^N w_{ci} f_i(x_i, y_i, y_{i-1})\right)}$$

Advantages and limitations

- MEMM has more flexibility in using features that are not just word-based or even human-engineered, giving it more richness and enabling its models to be more predictive.
- MEMM can have a range of more than just close words, giving it an advantage of detection over larger spans compared to HMM.

Deep learning and NLP

In the last few years, Deep Learning and its application to various areas of NLP has shown huge success and is considered the cutting edge of technology these days. The main advantage of using Deep Learning lies in a small subset of tools and methods, which are useful in a wide variety of NLP problems. It solves the basic issue of feature engineering and carefully created manual representations by automatically learning them, and thus solves the issue of having a large number of language experts dealing with a wide range of problems, such as text classification, sentiment analysis, POS tagging, and machine translation, to name a few. In this section, we will try to cover important concepts and research in the area of Deep Learning and NLP.

In his seminal paper, Bengio introduced one of the most important building blocks for deep learning known as word embedding or word vector (*References [20]*). Word embedding can be defined as a parameterized function that maps words to a high dimensional vector (usually 25 to 500 dimensions based on the application).

Formally, this can be written as $W : \text{words} \rightarrow \mathbb{R}^n$.

For example, $W(\text{"cat"}) = (0.2, -0.4, 0.3, \dots, 0.6)$ and $W(\text{"hat"}) = (0.1, 0.2, 0.7, \dots, 0.1)$, and so on.

A neural network (R) whose inputs are the words from sentences or n-grams of sentences with binary classification, such as whether the sequence of words in n-grams are valid or not, is used to train and learn the W and R :

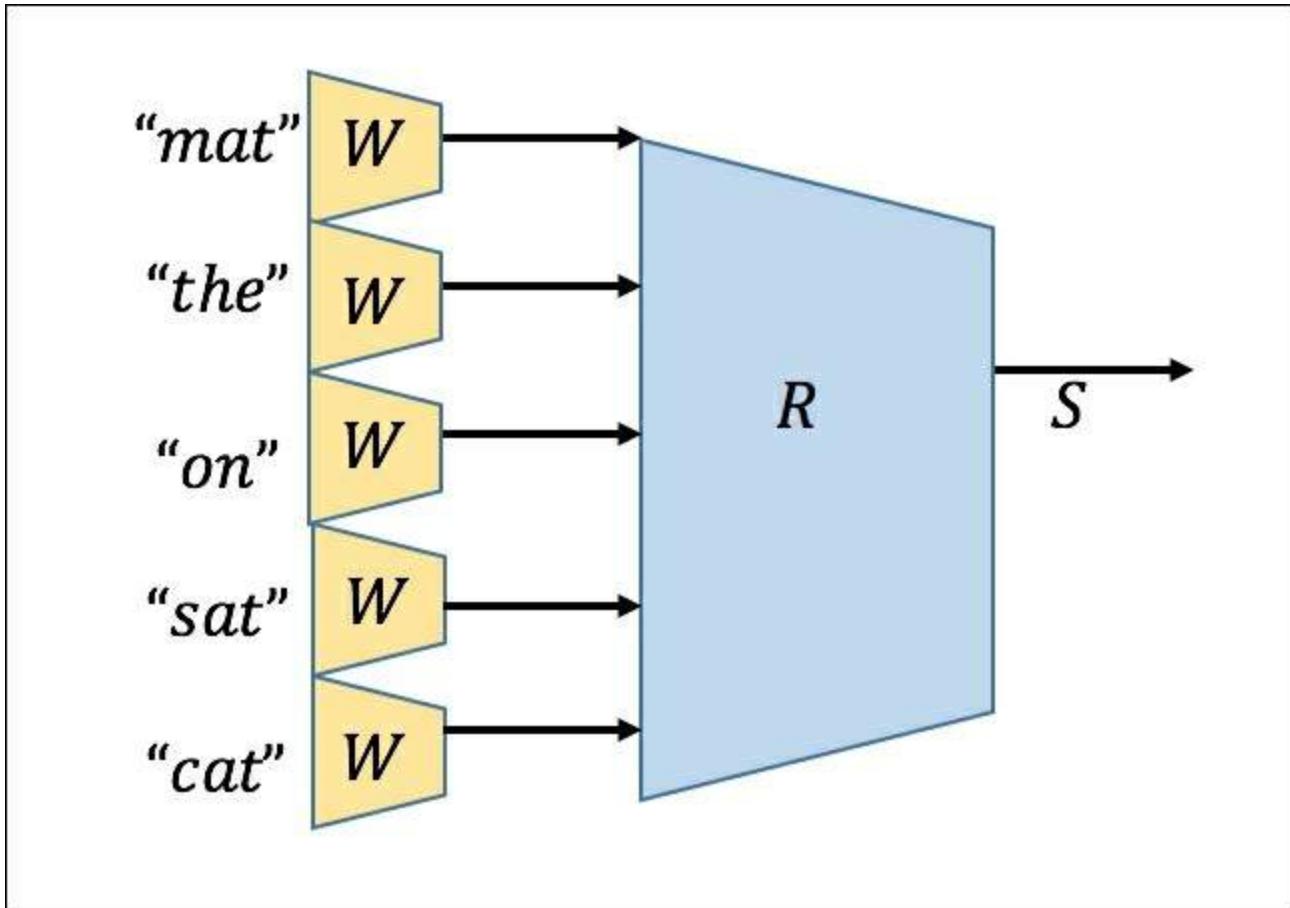


Figure 19: A modular neural network learning 5-gram words for valid–invalid classification

For example:

- $R(W(cat), W(sat), W(on), W(the), W(mat)) = 1(\text{valid})$
- $R(W(cat), W(sat), W(on), W(the), W(mat)) = 0 (\text{Invalid})$

The idea of training these sentences or n-grams is not only to learn the correct structure of the phrases, but also the right parameters for W and R . The word embeddings can also be projected on to a lower dimensional space, such as a 2D space, using various linear and non linear feature reduction/visualization techniques introduced in [Chapter 3, Unsupervised Machine Learning Techniques](#), which humans can easily visualize. This visualization of word embeddings in two dimensions using techniques such as t-SNE discovers important information about the closeness of words based on semantic meaning and even clustering of words in the area, as shown in the following figure:

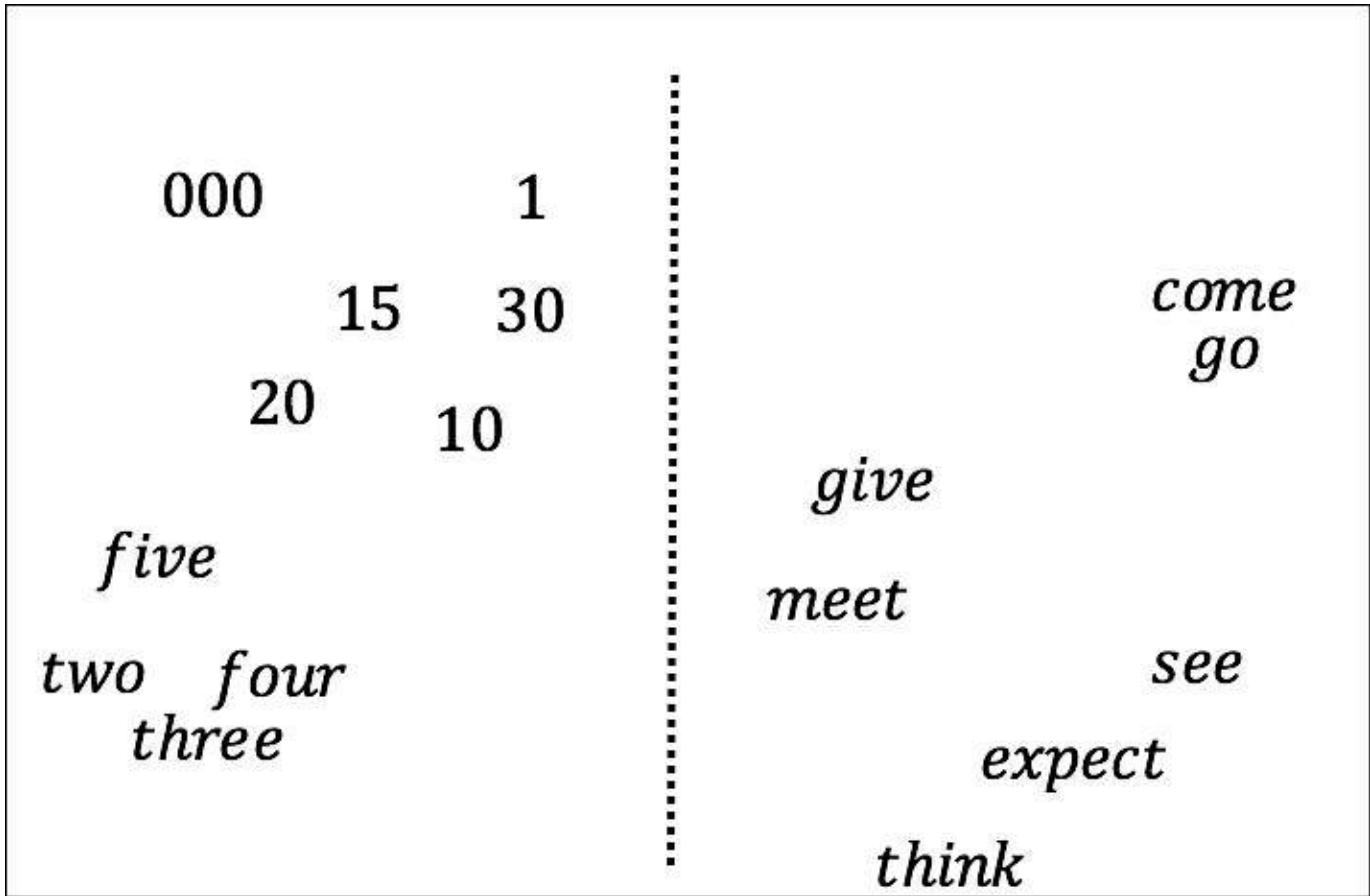


Figure 20: t-SNE representation of a small section of the entire word mappings. Numbers in Roman numerals and words are shown clustered together on the left, while semantically close words are clustered together on the right.

Further extending the concepts, both Collobert and Mikolov showed that the side effects of learning the word embeddings can be very useful in a variety of NLP tasks, such as similar phrase learning (for example, $W("the\ color\ is\ red") \approx W("the\ color\ is\ yellow")$), finding synonyms (for example, $W("nailed") \approx W("smashed")$), analogy mapping (for example, $W("man") \approx W("woman")$ then $W("king") \approx W("queen")$), and even complex relationship mapping (for example, $W("Paris") \approx W("France")$ then $W("Tokyo") \approx W("Japan")$) (References [21 and 22]).

The extension of word embedding concepts towards a generalized representation that helps us reuse the representation with various NLP problems (with minor extensions) has been the main reason for many recent successes of Deep Learning in NLP. Socher, in his research, extended the word embeddings concept to produce bilingual

word embeddings, that is, embed words from two different languages, such as Chinese (Mandarin) and English into a shared space (*References* [23]). By learning two language word embeddings independently of each other and then projecting them in a same space, his work gives us interesting insights into word similarities across languages that can be extended for Machine Translation. Socher also did interesting work on projecting the images learned from CNNs with the word embedding in to the same space for associating words with images as a basic classification problem (*References* [24]). Google, around the same time, has also been working on similar concepts, but at a larger scale for word-image matching and learning (*References* [26]).

Extending the word embedding concept to have combiners or association modules that can help combine the words, words-phrases, phrases-phrases in all combinations to learn complex sentences has been the idea of Recursive Neural Networks. The following figure shows how complex association (*((the cat)(sat(on (the mat))))*) can be learned using Recursive Neural Networks. It also removes the constraint of a "fixed" number of inputs in neural networks because of the ability to recursively combine:

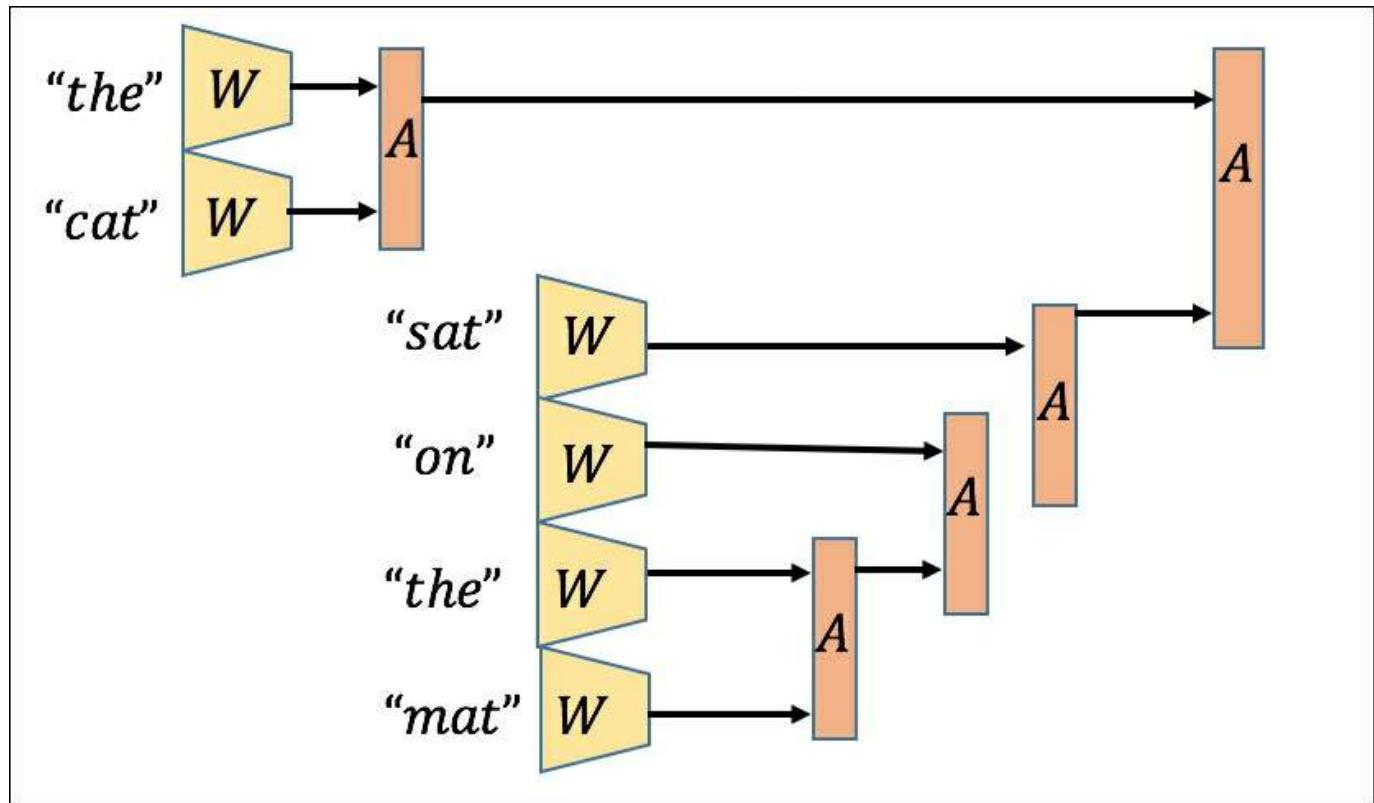


Figure 21: Recursive Neural Network showing how complex phrases can be learned.

Recursive Neural Networks have been showing great promise in NLP tasks, such as sentiment analysis, where association of one negative word at the start of many positive words has an overall negative impact on sentences, as shown in the following figure:

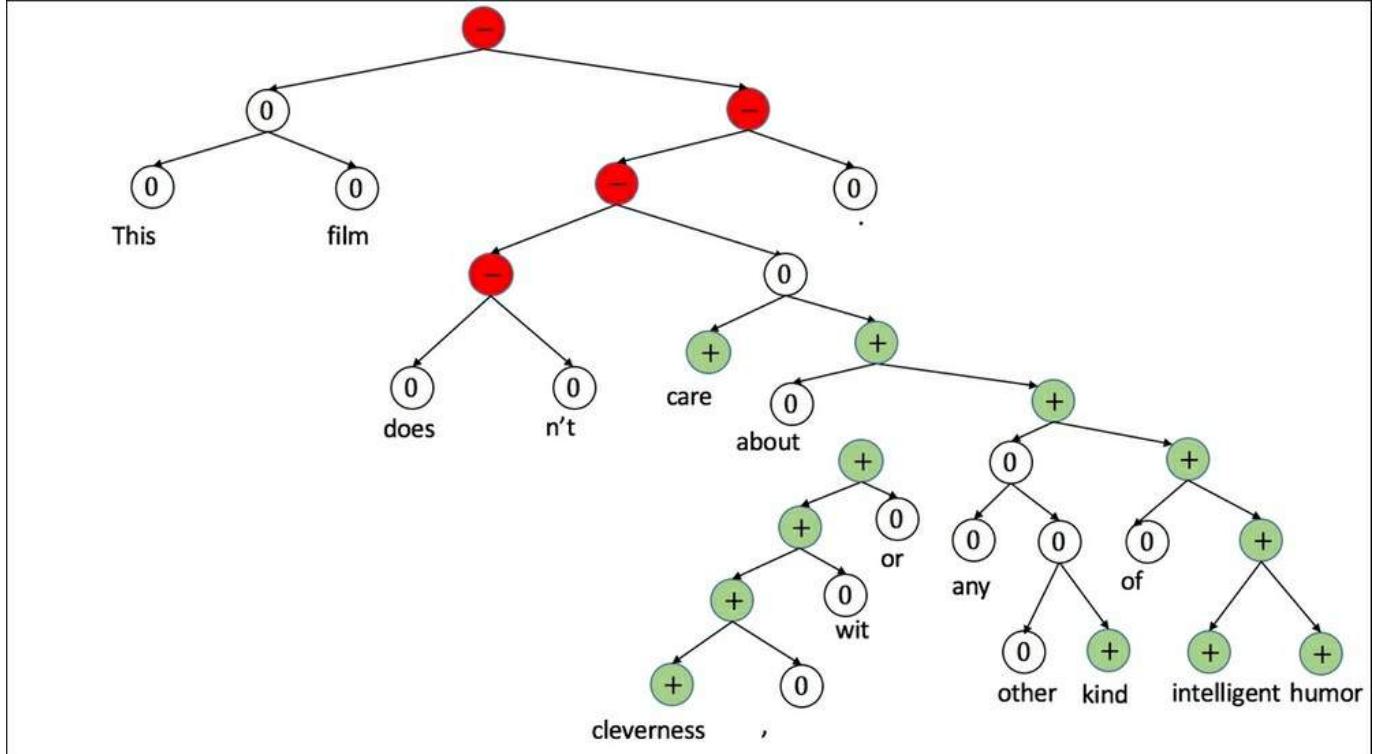


Figure 22: A complex sentence showing words with negative (as red circle), positive (green circle), and neutral (empty with 0) connected through RNN with overall negative sentiment.

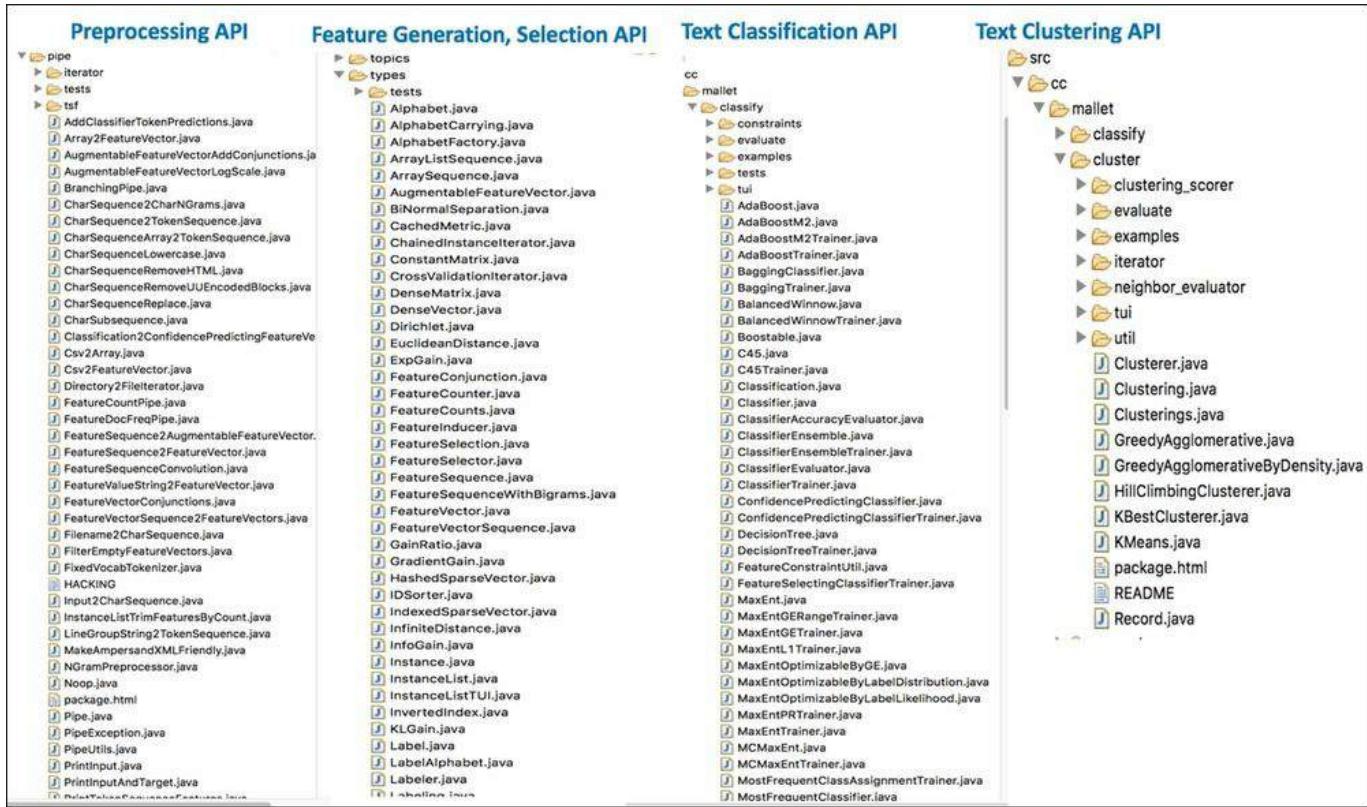
The concept of recursive neural networks is now extended using the building blocks of encoders and decoders to learn reversible sentence representation—that is, reconstructing the original sentence with roughly the same meaning from the input sentence (*References [27]*). This has become the central core theme behind Neural Machine Translation. Modeling conversations using the encoder-decoder framework of RNNs has also made huge breakthroughs (*References [28]*).

Tools and usage

We will now discuss some of the most well-known tools and libraries in Java that are used in various NLP and text mining applications.

Mallet

Mallet is a Machine Learning toolkit for text written in Java, which comes with several natural language processing libraries, including those some for document classification, sequence tagging, and topic modeling, as well as various Machine Learning algorithms. It is open source, released under CPL. Mallet exposes an extensive API (see the following screenshots) to create and configure sequences of "pipes" for pre-processing, vectorizing, feature selection, and so on, as well as to extend implementations of classification and clustering algorithms, plus a host of other text analytics and Machine Learning capabilities.



Sequence Learning and Tagging API

```
└── fst
    ├── confidence
    ├── semi_supervised
    └── tests
        ├── CacheStateIndicator.java
        ├── CRF.java
        ├── CRFCacheStateIndicator.java
        ├── CRFOptimizableByBatchLabelLikelihood.java
        ├── CRFOptimizableByGradientValues.java
        ├── CRFOptimizableByLabelLikelihood.java
        ├── CRFTrainerByL1LabelLikelihood.java
        ├── CRFTrainerByLabelLikelihood.java
        ├── CRFTrainerByStochasticGradient.java
        ├── CRFTrainerByThreadedLabelLikelihood.java
        ├── CRFTrainerByValueGradients.java
        ├── CRFWriter.java
        ├── FeatureTransducer.java
        ├── HMM.java
        ├── HMMTrainerByLikelihood.java
        ├── InstanceAccuracyEvaluator.java
        ├── LabelDistributionEvaluator.java
        ├── MaxLattice.java
        ├── MaxLatticeDefault.java
        ├── MaxLatticeFactory.java
        ├── MEMM.java
        ├── MEMMTrainer.java
        ├── MultiSegmentationEvaluator.java
        ├── NoopTransducerTrainer.java
        └── package.html

        PerClassAccuracyEvaluator.java
        Segment.java
        SegmentationEvaluator.java
        ShallowTransducerTrainer.java
        SimpleTagger.java
        SumLattice.java
        SumLatticeBeam.java
        SumLatticeConstrained.java
        SumLatticeDefault.java
        SumLatticeFactory.java
        SumLatticeScaling.java
        ThreadedOptimizable.java
        TokenAccuracyEvaluator.java
        Transducer.java
        TransducerEvaluator.java
        TransducerTrainer.java
```

Information Extraction API

```
└── src
    └── cc
        ├── mallet
            ├── classify
            ├── cluster
            ├── examples
            └── extract
                ├── pipe
                └── test
                    ├── AccuracyCoverageEvaluator.java
                    ├── BIOTokenizationFilter.java
                    ├── BIOTokenizationFilterWithTokenIndices.java
                    ├── ConfidenceTokenizationFilter.java
                    ├── CRFExtractor.java
                    ├── DefaultTokenizationFilter.java
                    ├── DocumentExtraction.java
                    ├── DocumentViewer.java
                    ├── ExactMatchComparator.java
                    ├── Extraction.java
                    ├── ExtractionConfidenceEstimator.java
                    ├── ExtractionEvaluator.java
                    ├── Extractor.java
                    ├── Field.java
                    ├── FieldCleaner.java
                    ├── FieldComparator.java
                    ├── HierarchicalTokenizationFilter.java
                    ├── LabeledSpan.java
                    ├── LabeledSpans.java
                    ├── LatticeViewer.java
                    └── package.html

                    PerDocumentF1Evaluator.java
                    PerFieldF1Evaluator.java
                    PunctuationIgnoringComparator.java
                    Record.java
                    RegexFieldCleaner.java
                    Span.java
                    StringSpan.java
                    StringTokenizer.java
                    Tokenization.java
                    TokenizationFilter.java
                    TransducerExtractionConfidenceEstimator.java
```

Topic Mining API

```
├── classify
├── cluster
├── examples
├── extract
├── fst
├── grmm
├── optimize
├── pipe
├── regression
├── share
└── topics
    └── tui
        ├── DMRLoader.java
        ├── EvaluateTopics.java
        ├── HierarchicalLDAUI.java
        ├── InferTopics.java
        ├── TopicTrainer.java
        ├── Vectors2Topics.java
        ├── AbstractTopicReports.java
        ├── DMROptimizable.java
        ├── DMRTopicModel.java
        ├── HierarchicalDA.java
        ├── HierarchicalPAM.java
        ├── JSONTopicReports.java
        ├── LabeledLDA.java
        ├── LDA.java
        ├── LDAHyper.java
        ├── LDAsStream.java
        ├── MarginalProbEstimator.java
        ├── MultinomialHMM.java
        ├── NPTopicModel.java
        ├── PAM4L.java
        ├── ParallelTopicModel.java
        ├── PolylingualTopicModel.java
        ├── RTopicModel.java
        ├── SimpleDA.java
        ├── TopicalNGrams.java
        ├── TopicAssignment.java
        ├── TopicInferencer.java
        ├── TopicModelDiagnostics.java
        ├── TopicReports.java
        ├── WeightedTopicModel.java
        ├── WordEmbeddingRunnable.java
        ├── WordEmbeddings.java
        └── WorkerRunnable.java
```

KNIME

KNIME is an open platform for analytics with Open GL licensing with a number of powerful tools for conducting all aspects of data science. The Text Processing module is available for separate download from KNIME Labs. KNIME has an intuitive drag and drop UI with downloadable examples available from their workflow server.

Note

KNIME: <https://www.knime.org/>

KNIME Labs: <https://tech.knime.org/knime-text-processing>

The platform includes a Node repository that contains all the necessary tools to compose your workflow with a convenient nesting of nodes that can easily be reused by copying and pasting. The execution of the workflows is simple. Debugging errors can take some getting used to, so our recommendation is to take the text mining example, use a different dataset as input, and make the workflow execute without errors. This is the quickest way to get familiar with the platform.

Topic modeling with mallet

We will now illustrate the usage of API and Java code to implement Topic Modeling to give the user an illustration on how to build a text learning pipeline for a problem in Java:

```
//create pipeline
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
    // Pipes: lowercase, tokenize, remove stopwords, map to features
pipeList.add( new CharSequenceLowercase() );
pipeList.add( new CharSequence2TokenSequence(Pattern.compile("\\"p{L}
[\"p{L}\"p{P}]+"p{L}")) );
pipeList.add( new TokenSequenceRemoveStopwords(new
File("stopReuters/en.txt"), "UTF-8", false, false, false) );
//add all
pipeList.add( new TokenSequence2FeatureSequence() );
InstanceList instances = new InstanceList (new SerialPipes(pipeList));
//read the file
Reader fileReader = new InputStreamReader(new FileInputStream(new
File(reutersFile)), "UTF-8");
instances.addThruPipe(new CsvIterator (fileReader,
Pattern.compile("^(\\"S*)[\"s,]*(\\"S*)[\"s,]*(.*)$"),
3, 2, 1)); // name fields, data, label
```

ParallelTopicModel in Mallet has an API with parameters such as the number of topics, alpha, and beta that control the underlying parameter for tuning the LDA using Dirichlet distribution. Parallelization is very well supported, as seen by the increased number of threads available in the system:

```
ParallelTopicModel model = new ParallelTopicModel(10, 1.0, 0.01); //10
topics using LDA method
model.addInstances(instances); //add instances
model.setNumThreads(3); //parallelize with threading
model.setNumIterations(1000); //gibbs sampling iterations
model.estimate(); //perform estimation of probability
```

Topic and term association is shown in the following screenshot as the result of running the ParallelTopicModel in Mallet. Clearly, the top terms and association of the topics are very well discovered in many cases, such as the classes of exec, acq, wheat, crude, corn, and earning:

Topic (class)	Probability	Terms(Weights)
0(exc)	0.091	bank (1990) pct (1540) market (1250) rate (1093) rates (897) min (852) stg (723) money (676) banks (626) billion (622)
1(exc)	0.097	trade (1733) u.s (1718) japan (1017) japanese (695) dollar (681) foreign (583) told (522) economic (485) exchange (479) government (462)
2(acq)	0.229	cts (4879) min (4855) net (4802) loss (4271) shr (4024) dtrs (2684) march (2405) rev (2321) qtr (2307) profit (2264)
3	0.120	shares (1576) pct (1135) stock (1083) offer (1050) company (1010) dtrs (993) march (798) share (795) group (673) common (626)
4(wheat)	0.065	tonnes (2104) min (1721) wheat (1298) corn (720) u.s (647) grain (568) pct (532) march (528) april (463) export (432)
5	0.041	u.s (466) gulf (437) oil (361) iran (249) iranian (246) march (201) shipping (188) ships (181) strike (167) spokesman (164)
6(crude)	0.050	oil (2082) crude (579) min (499) prices (489) gas (470) dtrs (466) opec (408) bpd (385) barrels (354) production (340)
7(corn)	0.046	u.s (918) wheat (472) farm (391) trade (356) washington (341) corn (340) bill (330) agriculture (327) march (289) program (254)
8(earning)	0.121	min (3374) dtrs (2395) billion (1883) year (1639) pct (1553) company (870) quarter (770) earnings (666) march (663) share (569)
9(acq)	0.139	min (1055) company (946) march (878) corp (871) dtrs (796) sale (508) sell (421) unit (407) acquisition (391) agreement (377)

Business problem

The Reuters corpus labels each document with one of 10 categories. The aim of the experiments in this case study is to employ the techniques of text processing learned in this chapter to give structure to these documents using vector space modeling. This is done in three different ways, and four classification algorithms are used to train and make predictions using the transformed dataset in each of the three cases. The open source Java analytics platform KNIME was used for text processing and learning.

Machine Learning mapping

Among the learning techniques for unstructured data, such as text or images, classification of the data into different categories given a training set with labels is a supervised learning problem. However, since the data is unstructured, some statistical or information theoretic means are necessary to extract learnable features from the data. In the design of this study, we performed feature representation and selection on the documents before using linear, non-linear, and ensemble methods for classification.

Data collection

The dataset used in the experiments is a version of the Reuters-21578 Distribution 1.0 Text Categorization Dataset available from the UCI Machine Learning Repository:

Note

Reuters-21578 dataset: <https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection>

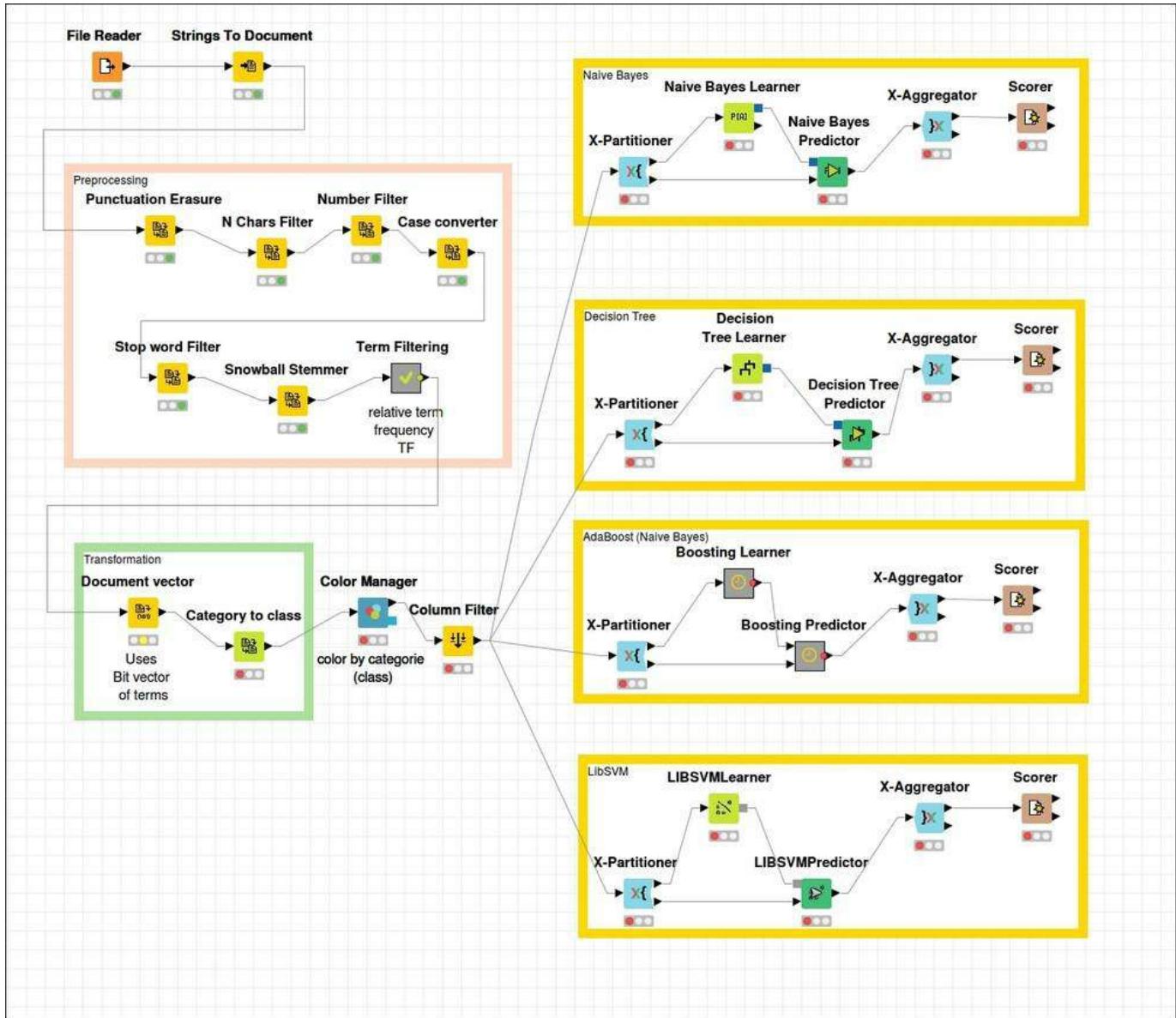
This dataset is a Modified-Apte split containing 9,981 documents, each with a class label indicating the category of the document. There are 10 distinct categories in the dataset.

Data sampling and transformation

After importing the data file, we performed a series of pre-processing steps in order to enrich and transform the data before training any models on the documents. These steps can be seen in the screenshot of the workflow created in KNIME. They include:

- Punctuation erasure
- N char filtering (removes tokens less than four characters in length)
- Number filtering
- Case conversion – convert all to lower case
- Stop word filtering
- Stemming

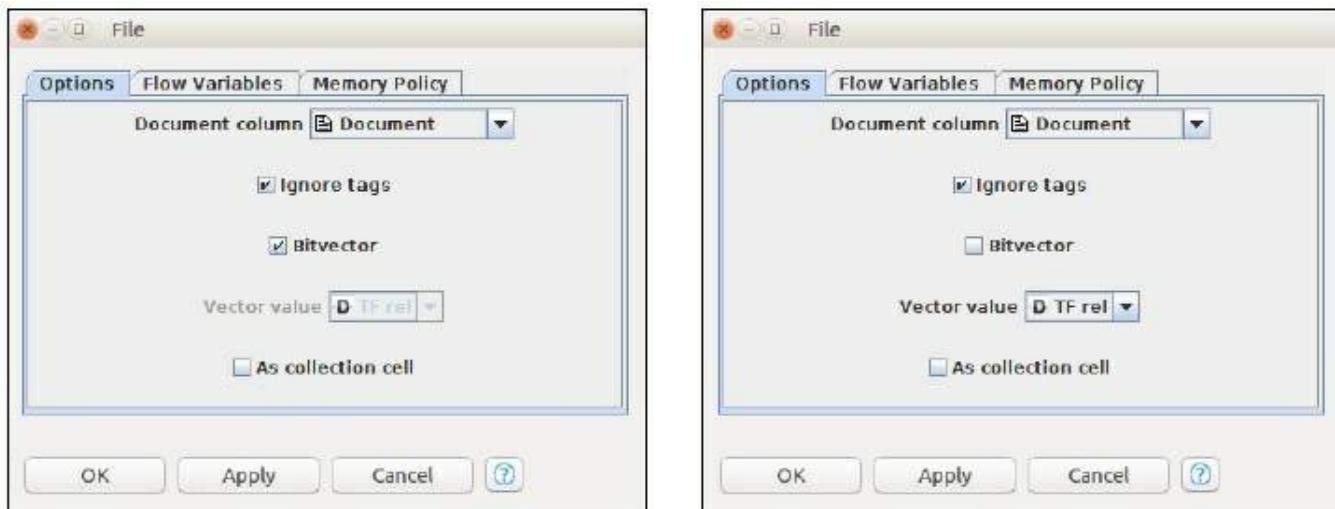
Prior to the learning step, we sampled the data randomly into a 70-30 split for training and testing, respectively. We used five-fold cross-validation in each experiment.



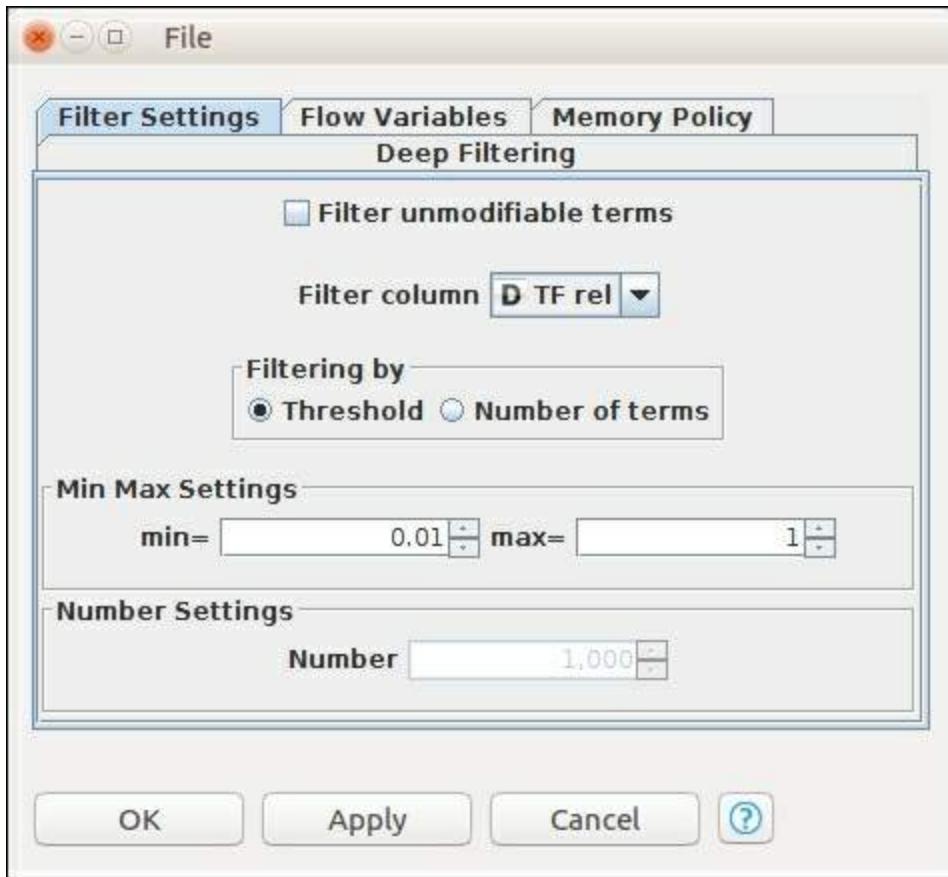
The preceding screenshot shows the workflow for the first experiment set, which uses a binary vector of features. Data import is followed by a series of pre processing nodes, after which the dataset is transformed into a document vector. After adding back the target vector, the workflow branches out into four classification tasks, each using a five-fold cross-validation setup. Results are gathered in the Scorer node.

Feature analysis and dimensionality reduction

We conducted three sets of experiments in total. In the first set, after pre processing, we used binary vectorization of the terms, which adds a representation indicating whether or not a term appeared in the document:



In the second experiment, we used the values for relative **Term Frequency (TF)** for each term, resulting in a value between 0 and 1.



In the third, we performed feature selection by filtering out terms that had a relative TF score of less than 0.01.

Models, results, and evaluation

For each of the three sets of experiments, we used two linear classifiers (naïve Bayes and SVM using linear kernel) and two non-linear classifiers (Decision Tree and AdaBoost with Naïve Bayes as base learner). In text mining classification, precision/recall metrics are generally chosen as the evaluation metric over accuracy, which is more common in traditional, balanced classification problems.

The results from the three sets of experiments are given in the tables. Scores are averages over all the classes:

Binary Term Vector:

Classifier	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
Naïve Bayes	0.5079	0.5281	0.5079	0.9634	0.5087	0.7063	0.6122
Decision Tree	0.4989	0.5042	0.4989	0.9518	0.5013	0.7427(2)	0.6637(2)
AdaBoost(NB)	0.5118(2)	0.5444(2)	0.5118(2)	0.9665(2)	0.5219(2)	0.7285	0.6425
LibSVM	0.6032(1)	0.5633(1)	0.6032(1)	0.9808(1)	0.5768(1)	0.8290(1)	0.7766(1)

Relative TF vector:

Classifier	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
Naïve Bayes	0.4853	0.5480(2)	0.4853	0.9641	0.5113(2)	0.7248	0.6292
Decision Tree	0.4947(2)	0.4954	0.4947(2)	0.9703(2)	0.4950	0.7403(2)	0.6612(2)
AdaBoost(NB)	0.4668	0.5326	0.4668	0.9669	0.4842	0.6963	0.6125
LibSVM	0.6559(1)	0.6651(1)	0.6559(1)	0.9824(1)	0.6224(1)	0.8433(1)	0.7962(1)

Relative TF vector with threshold filtering (rel TF > 0.01):

Classifier	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa

Naïve Bayes	0.4689	0.5456(2)	0.4689	0.9622	0.4988	0.7133	0.6117	
Decision Tree	0.5008(2)	0.5042	0.5008(2)	0.9706(2)	0.5022(2)	0.7439(2)	0.6657(2)	
AdaBoost(NB)	0.4435	0.4992	0.4435	0.9617	0.4598	0.6870	0.5874	
LibSVM	0.6438(1)	0.6326(1)	0.6438(1)	0.9810(1)	0.6118(1)	0.8313(1)	0.7806(1)	

Analysis of text processing results

The analysis of results obtained from our experiments on the Reuters dataset is presented here with some key observations:

- As seen in the first table, with the binary representation of terms, Naïve Bayes scores around 0.7, which indicates that the features generated have good discriminating power. AdaBoost on the same configuration of Naïve Bayes further improves all the metrics, such as precision, recall, F1-measure, and accuracy, by about 2%, indicating the advantage of boosting and meta-learning.
- As seen in the first table, non-linear classifiers, such as Decision Tree, do only marginally better than linear Naïve Bayes in most metrics. SVM with a linear classifier increases accuracy by 17% over linear Naïve Bayes and has better metrics similarly in almost all measures. SVM and kernels, which have no issues with higher dimensional data, the curse of text classification, are thus one of the better algorithms for modeling, and the results confirm this.
- Changing the representation from binary to TF improves many measures, such as accuracy, for linear Naïve Bayes (from 0.70 to 0.72) and SVM (0.82 to 0.84). This indeed confirms that TF-based representation in many numeric-based algorithms, such as SVM. AdaBoost performance with Naïve Bayes drops in most metrics when the underlying classifier Bayes gets stronger in performance, as shown in many theoretical and empirical results.
- Finally, by reducing features using threshold $TF > 0.01$, as used here, we get almost similar or somewhat reduced performance in most classifiers, indicating that although certain terms seem rare, they have discriminating power, and reducing them has a negative impact.

Summary

A large proportion of information in the digital world is textual. Text mining and NLP are areas concerned with extracting information from this unstructured form of data. Several important sub areas in the field are active topics of research today and an understanding of these areas is essential for data scientists.

Text categorization is concerned with classifying documents into pre-determined categories. Text may be enriched by annotating words, as with POS tagging, in order to give it more structure for subsequent processing tasks to act on. Unsupervised techniques such as clustering can be applied to documents as well. Information extraction and named entity recognition help identify information-rich specifics such as location, person or organization name, and so on. Summarization is another important application for producing concise abstracts of larger documents or sets of documents. Various ambiguities of language and semantics such as context, word sense, and reasoning make the tasks of NLP challenging.

Transformations of the contents of text include tokenization, stop words removal, and word stemming, all of which prepare the corpus by standardizing the content so Machine Learning techniques can be applied productively. Next, lexical, semantic, and syntactic features are extracted so numerical values can represent the document structure more conventionally with a vector space model. Similarity and distance measures can then be applied to effectively compare documents for sameness. Dimensionality reduction is key due to the large number of features that are typically present. The details of the techniques for topic modeling, PLSA and text clustering, and named entity recognition are described in this chapter. Finally, the recent techniques employing deep learning in various fields of NLP are introduced to the readers.

Mallet and KNIME are two open source Java-based tools that provide powerful NLP and Machine Learning capabilities. The case study examines performance of different classifiers on the Reuters corpus using KNIME.

References

1. J. B. Lovins (1968). *Development of a stemming algorithm*, Mechanical Translation and Computer Linguistic, vol.11, no.1/2, pp. 22-31.
2. Porter M.F, (1980). *An algorithm for suffix stripping*, Program; 14, 130-137.
3. ZIPF, H.P., (1949). *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, Cambridge, Massachusetts.
4. LUHN, H.P., (1958). *The automatic creation of literature abstracts'*, IBM Journal of Research and Development, 2, 159-165.
5. Deerwester, S., Dumais, S., Furnas, G., & Landauer, T. (1990), *Indexing by latent semantic analysis*, Journal of the American Society for Information Sciences, 41, 391–407.
6. Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977), *Maximum likelihood from incomplete data via the EM algorithm*. Journal of the Royal Statistic Society, Series B, 39(1), 1–38.
7. Greiff, W. R. (1998). *A theory of term weighting based on exploratory data analysis*. In 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, NY. ACM.
8. P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J/ C. Lai (1992), *Class-based n-gram models of natural language*, Computational Linguistics, 18, 4, 467-479.
9. T. Liu, S. Lin, Z. Chen, W.-Y. Ma (2003), *An Evaluation on Feature Selection for Text Clustering*, ICML Conference.
10. Y. Yang, J. O. Pederson (1995). *A comparative study on feature selection in text categorization*, ACM SIGIR Conference.
11. Salton, G. & Buckley, C. (1998). *Term weighting approaches in automatic text retrieval*. Information Processing & Management, 24(5), 513–523.
12. Hofmann, T. (2001). *Unsupervised learning by probabilistic latent semantic analysis*. Machine Learning Journal, 41(1), 177–196.
13. D. Blei, J. Lafferty (2006). *Dynamic topic models*. ICML Conference.
14. D. Blei, A. Ng, M. Jordan (2003). *Latent Dirichlet allocation*, Journal of Machine Learning Research, 3: pp. 993–1022.
15. W. Xu, X. Liu, and Y. Gong (2003). *Document-Clustering based on Non-negative Matrix Factorization*. Proceedings of SIGIR'03, Toronto, CA, pp. 267-273,
16. Dud'ik M. and Schapire (2006). R. E. *Maximum entropy distribution estimation with generalized regularization*. In Lugosi, G. and Simon, H. (Eds.), COLT, Berlin, pp. 123– 138, Springer-Verlag,.

17. McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). *Maximum Entropy Markov Models for Information Extraction and Segmentation*. In ICML, pp. 591–598..
18. Langville, A. N, Meyer, C. D., Albright, R. (2006). *Initializations for the Nonnegative Factorization*. KDD, Philadelphia, USA
19. Dunning, T. (1993). *Accurate Methods for the Statistics of Surprise and Coincidence*. Computational Linguistics, 19, 1, pp. 61-74.
20. Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin (2003). *A Neural Probabilistic Language Model*. Journal of Machine Learning Research.
21. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. (2011). *Natural language processing (almost) from scratch*. Journal of Machine Learning Research, 12:2493–2537.
22. T. Mikolov, K. Chen, G. Corrado and J. Dean (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv:1301.3781v1.
23. R. Socher, Christopher Manning, and Andrew Ng. (2010). *Learning continuous phrase representations and syntactic parsing with recursive neural networks*. In NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning.
24. R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. (2011). *Semi-supervised recursive autoencoders for predicting sentiment distributions*. In EMNLP.
25. M. Luong, R. Socher and C. Manning (2013). *Better word representations with recursive neural networks for morphology*. CONLL.
26. A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al (2013). *Devise: A deep visual-semantic embedding model*. In NIPS Proceedings.
27. Léon Bottou (2011). From Machine Learning to Machine Reasoning.
<https://arxiv.org/pdf/1102.1808v3.pdf>.
28. Cho, Kyunghyun, et al (2014). *Learning phrase representations using rnn encoder-decoder for statistical machine translation*. arXiv preprint arXiv:1406.1078.

Chapter 9. Big Data Machine Learning

– The Final Frontier

In recent years, we have seen an exponential growth in data generated by humans and machines. Varied sources, including home sensors, healthcare-related monitoring devices, news feeds, conversations on social media, images, and worldwide commerce transactions—an endless list—contribute to the vast volumes of data generated every day.

Facebook had 1.28 billion daily active users in March 2017 sharing close to four million pieces of unstructured information as text, images, URLs, news, and videos (Source: Facebook). 1.3 billion Twitter users share approximately 500 million tweets a day (Source: Twitter). **Internet of Things (IoT)** sensors in lights, thermostats, sensor in cars, watches, smart devices, and so on, will grow from 50 billion to 200 billion by 2020 (Source: IDC estimates). YouTube users upload 300 hours of new video content every five minutes. Netflix has 30 million viewers who stream 77,000 hours of video daily. Amazon has sold approximately 480 million products and has approximately 244 million customers. In the financial sector, the volume of transactional data generated by even a single large institution is enormous—approximately 25 million households in the US have Bank of America, a major financial institution, as their primary bank, and together produce petabytes of data annually. Overall, it is estimated that the global Big Data industry will be worth 43 billion US dollars in 2017 (Source: www.statista.com).

Each of the aforementioned companies and many more like them face the real problem of storing all this data (structured and unstructured), processing the data, and learning hidden patterns from the data to increase their revenue and to improve customer satisfaction. We will explore how current methods, tools and technology can help us learn from data in Big Data-scale environments and how as practitioners in the field we must recognize challenges unique to this problem space.

This chapter has the following structure:

- What are the characteristics of Big Data?
- Big Data Machine Learning
 - General Big Data Framework:
 - Big data cluster deployments frameworks

- HortonWorks Data Platform (HDP)
 - Cloudera CDH
 - Amazon Elastic MapReduce (EMR)
 - Microsoft HDInsight
- Data acquisition:
 - Publish-subscribe framework
 - Source-sink framework
 - SQL framework
 - Message queueing framework
 - Custom framework
- Data storage:
 - Hadoop Distributed File System (HDFS)
 - NoSQL
- Data processing and preparation:
 - Hive and Hive Query Language (HQL)
 - Spark SQL
 - Amazon Redshift
 - Real-time stream processing
- Machine Learning
- Visualization and analysis
- Batch Big Data Machine Learning
 - H2O:
 - Machine learning in H2O
 - Tools and usage
 - Case study
 - Business problems
 - Machine Learning mapping
 - Data collection
 - Data sampling and transformation
 - Experiments, results, and analysis
 - Spark MLlib:
 - Spark architecture
 - Machine Learning in MLlib
 - Tools and usage
 - Experiments, results, and analysis
- Real-time Big Data Machine Learning
 - Scalable Advanced Massive Online Analysis (SAMOA):

- SAMOA architecture
- Machine Learning algorithms
- Tools and usage
- Experiments, results, and analysis
- The future of Machine Learning

What are the characteristics of Big Data?

There are many characteristics of Big Data that are different than normal data. Here we highlight them as four *Vs* that characterize Big Data. Each of these makes it necessary to use specialized tools, frameworks, and algorithms for data acquisition, storage, processing, and analytics:

- **Volume:** One of the characteristic of Big Data is the size of the content, structured or unstructured, which doesn't fit the storage capacity or processing power available on a single machine and therefore needs multiple machines.
- **Velocity:** Another characteristic of Big Data is the rate at which the content is generated, which contributes to volume but needs to be handled in a time sensitive manner. Social media content and IoT sensor information are the best examples of high velocity Big Data.
- **Variety:** This generally refers to multiple formats in which data exists, that is, structured, semi-structured, and unstructured and furthermore, each of them has different forms. Social media content with images, video, audio, text, and structured information about activities, background, networks, and so on, is the best example of where data from various sources must be analyzed.
- **Veracity:** This refers to a wide variety of factors such as noise, uncertainty, biases, and abnormality in the data that must be addressed, especially given the volume, velocity, and variety of data. One of the key steps, as we will discuss in the context of Big Data Machine Learning, is processing and cleaning such "unclean" data.

Many have added other characteristics such as value, validity, and volatility to the preceding list, but we believe they are largely derived from the previous four.

Big Data Machine Learning

In this section, we will discuss the general flow and components that are required for Big Data Machine Learning. Although many of the components, such as data acquisition or storage, are not directly related to Machine Learning methodologies, they inevitably have an impact on the frameworks and processes. Giving a complete catalog of the available components and tools is beyond the scope of this book, but we will discuss the general responsibilities of the tasks involved and give some information on the techniques and tools available to accomplish them.

General Big Data framework

The general Big Data framework is illustrated in the following figure:

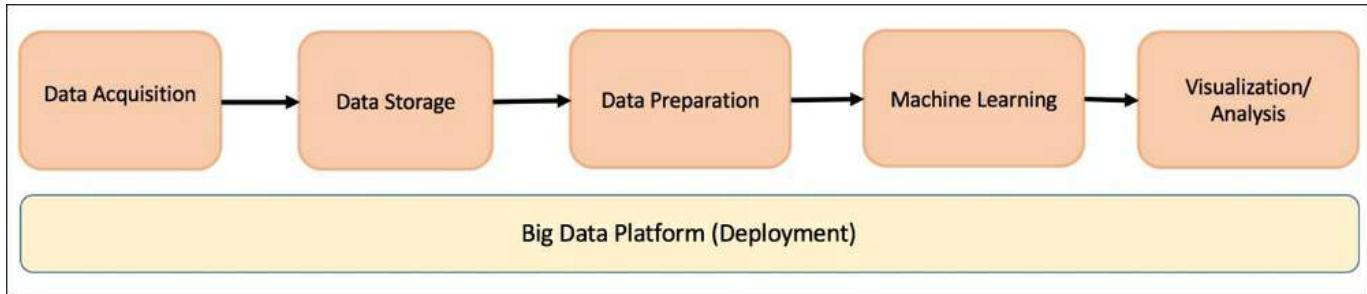


Figure 1: Big data framework

The choice of how the Big Data framework is set up and deployed in the cluster is one of the decisions that affects the choice of tools, techniques, and cost. The data acquisition or collection component is the first step and it consists of several techniques, both synchronous and asynchronous, to absorb data into the system. Various techniques ranging from publish-subscribe, source-sink, relational database queries, and custom data connectors are available in the components.

Data storage choices ranging from distributed filesystems such as HDFS to non-relational databases (NoSQL) are available based on various other functional requirements. NoSQL databases are described in the section on *Data Storage*.

Data preparation, or transforming the large volume of stored data so that it is consumable by the Machine Learning analytics, is an important processing step. This has some dependencies on the frameworks, techniques, and tools used in storage. It also has some dependency on the next step: the choice of Machine Learning analytics/frameworks that will be used. There are a wide range of choices for processing frameworks that will be discussed in the following sub-section.

Recall that, in batch learning, the model is trained simultaneously on a number of examples that have been previously collected. In contrast to batch learning, in real-time learning model training is continuous, each new instance that arrives becoming part of a dynamic training set. See [Chapter 5, Real-Time Stream Machine Learning](#) for details. Once the data is collected, stored, and transformed based on the domain

requirements, different Machine Learning methodologies can be employed, including batch learning, real-time learning, and batch-real-time mixed learning. Whether one selects supervised learning, unsupervised learning, or a combination of the two also depends on the data, the availability of labels, and label quality. These will be discussed in detail later in this chapter.

The results of analytics during the development stage as well as the production or runtime stage also need to be stored and visualized for humans and automated tasks.

Big Data cluster deployment frameworks

There are many frameworks that are built on the core Hadoop (*References [3]*) open source platform. Each of them provides a number of tools for the Big Data components described previously.

Hortonworks Data Platform

Hortonworks Data Platform (HDP) provides an open source distribution comprising various components in its stack, from data acquisition to visualization. Apache Ambari is often the user interface used for managing services and provisioning and monitoring clusters. The following screenshot depicts Ambari used for configuring various services and the health-check dashboard:

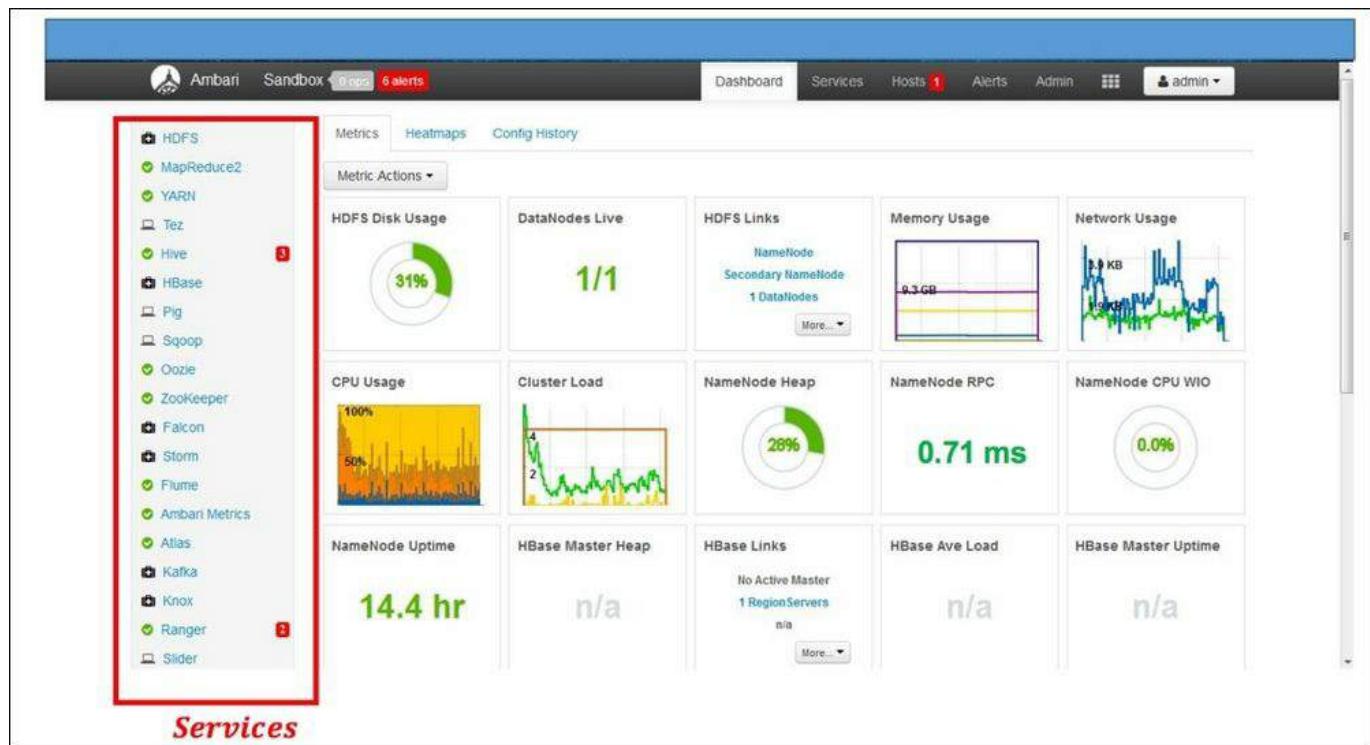


Figure 2: Ambari dashboard user interface

Cloudera CDH

Like HDP, Cloudera CDH (*References [4]*) provides similar services and Cloudera Services Manager can be used in a similar way to Ambari for cluster management and health checks, as shown in the following screenshot:

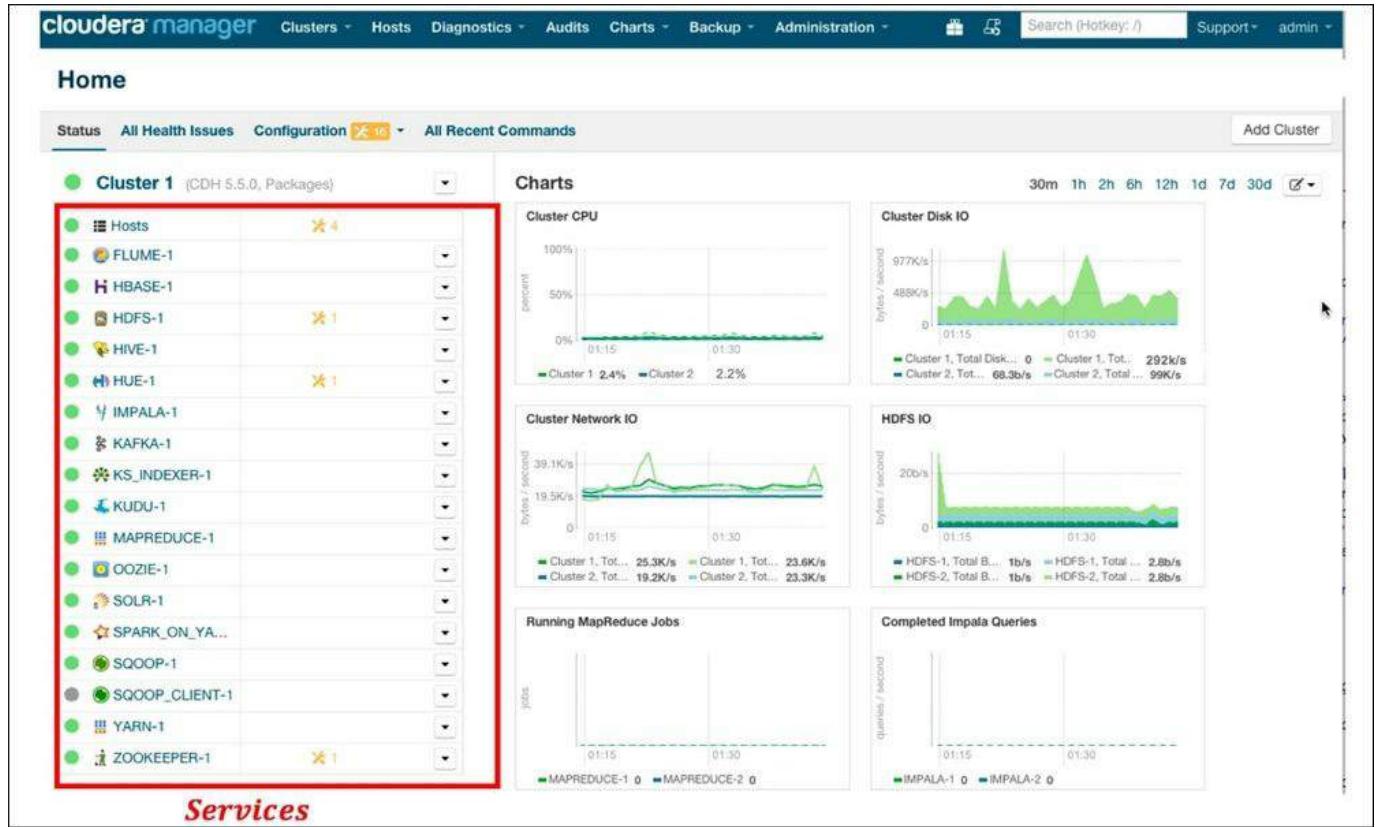


Figure 3: Cloudera Service Manager user interface

Amazon Elastic MapReduce

Amazon Elastic MapReduce (EMR) (*References [5]*) is another Big Data cluster, platform similar to HDP and Cloudera, which supports a wide variety of frameworks. EMR has two modes—**cluster mode** and **step execution mode**. In cluster mode, you choose the Big Data stack vendor EMR or MapR and in step execution mode, you give jobs ranging from JARs to SQL queries for execution. In the following screenshot, we see the interface for configuring a new cluster as well

as defining a new job flow:

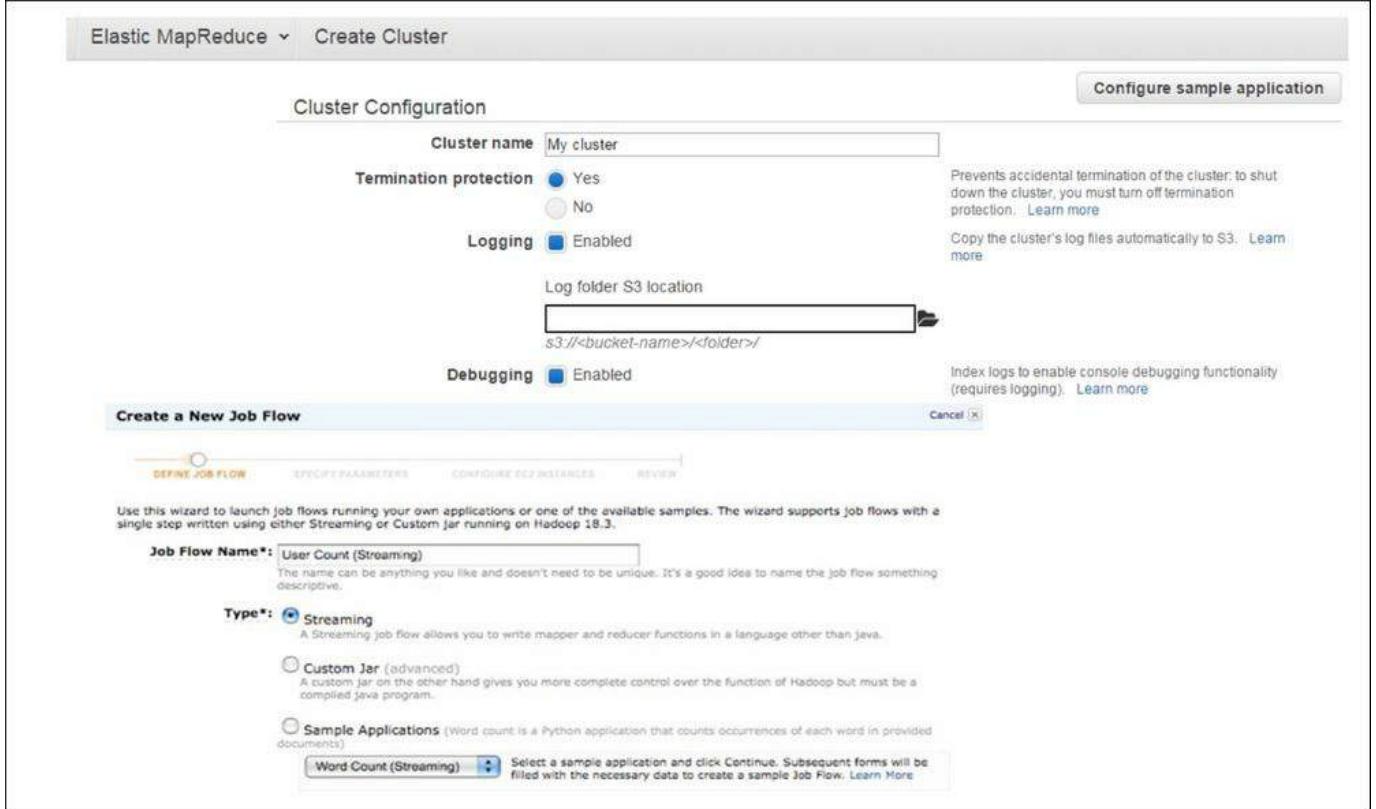


Figure 4: Amazon Elastic MapReduce cluster management user interface

Microsoft Azure HDInsight

Microsoft Azure HDInsight (*References [6]*) is another platform that allows cluster management with most of the services that are required, including storage, processing, and Machine Learning. The Azure portal, as shown in the following screenshot, is used to create, manage, and help in learning the statuses of the various components of the cluster:

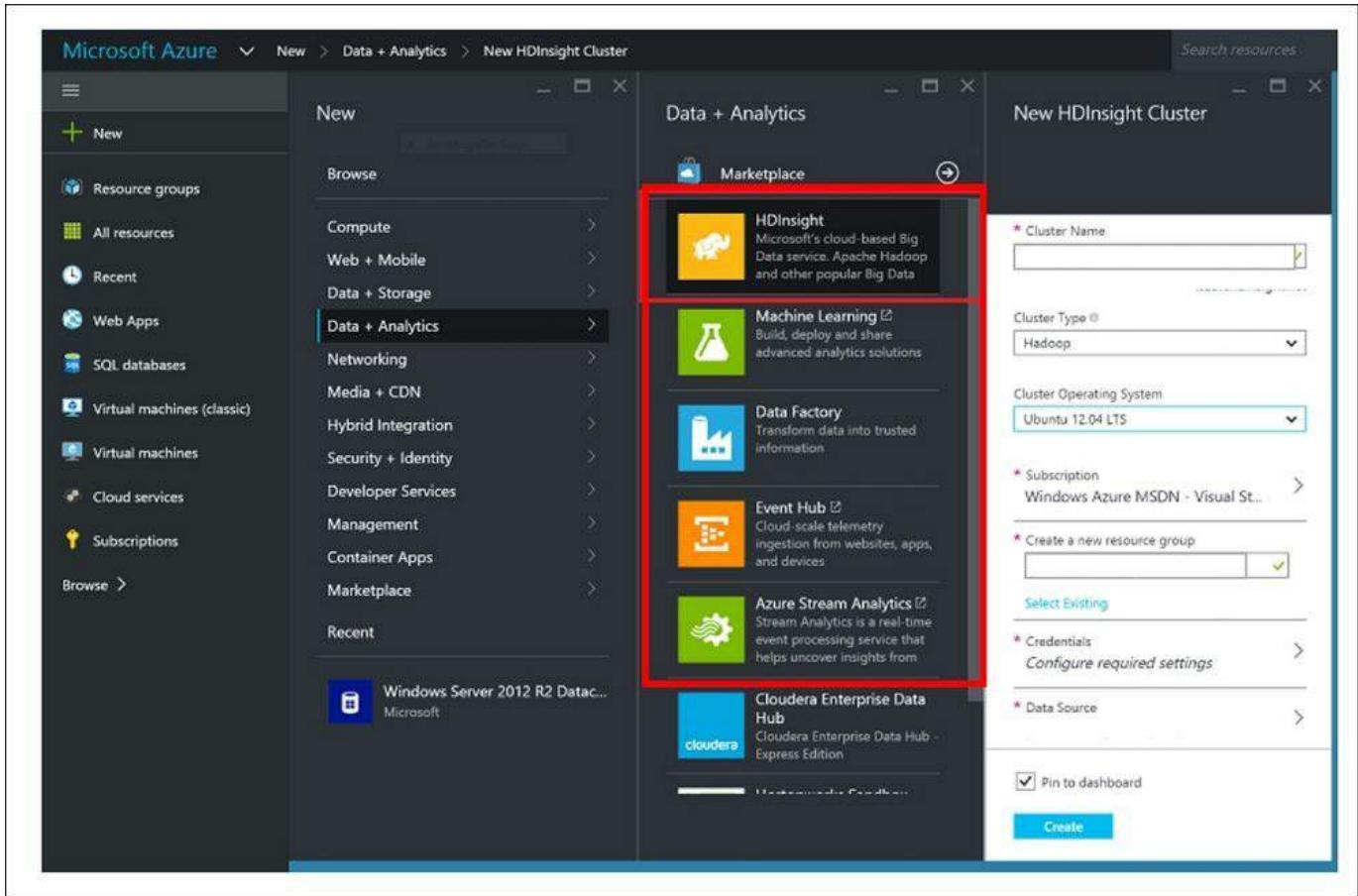


Figure 5: Microsoft Azure HDInsight cluster management user interface

Data acquisition

In the Big Data framework, the acquisition component plays an important role in collecting the data from disparate source systems and storing it in Big Data storage. Based on types of source and volume, velocity, functional, and performance-based requirements, there are a wide variety of acquisition frameworks and tools. We will describe a few of the most well-known frameworks and tools used to give readers some insight.

Publish-subscribe frameworks

In publish-subscribe based frameworks, the publishing source pushes the data in different formats to the broker, which has different subscribers waiting to consume them. The publisher and subscriber are unaware of each other, with the broker mediating in between.

Apache Kafka (*References [9]*) and **Amazon Kinesis** are two well-known implementations that are based on this model. Apache Kafka defines the concepts of publishers, consumers, and topics—on which things get published and consumed—and a broker to manage the topics. Amazon Kinesis is built on similar concepts with producers and consumers connected through Kinesis streams, which are similar to topics in Kafka.

Source-sink frameworks

In source-sink models, sources push the data into the framework and the framework pushes the system to the sinks. Apache Flume (*References [7]*) is a well-known implementation of this kind of framework with a variety of sources, channels to buffer the data, and a number of sinks to store the data in the Big Data world.

SQL frameworks

Since many traditional data stores are in the form of SQL-based RDBMS, SQL-based frameworks provide a generic way to import the data from RDBMS and store it in Big Data, mainly in the HDFS format. Apache Sqoop (*References [10]*) is a well-known implementation that can import data from any JDBC-based RDBMS and store it in HDFS-based systems.

Message queueing frameworks

Message queueing frameworks are push-pull based frameworks similar to publisher-subscriber systems. Message queues separate the producers and consumers and can store the data in the queue, in an asynchronous communication pattern. Many protocols have been developed on this such as Advanced Message Queueing Protocol (AMQP) and ZeroMQ Message Transfer Protocol (ZMTP). RabbitMQ, ZeroMQ, Amazon SQS, and so on, are some well-known implementations of this framework.

Custom frameworks

Specialized connectors for different sources such as IoT, HTTP, WebSockets, and so on, have resulted in many specific connectors such as Amazon IoT Hub, REST-connectors, WebSocket, and so on.

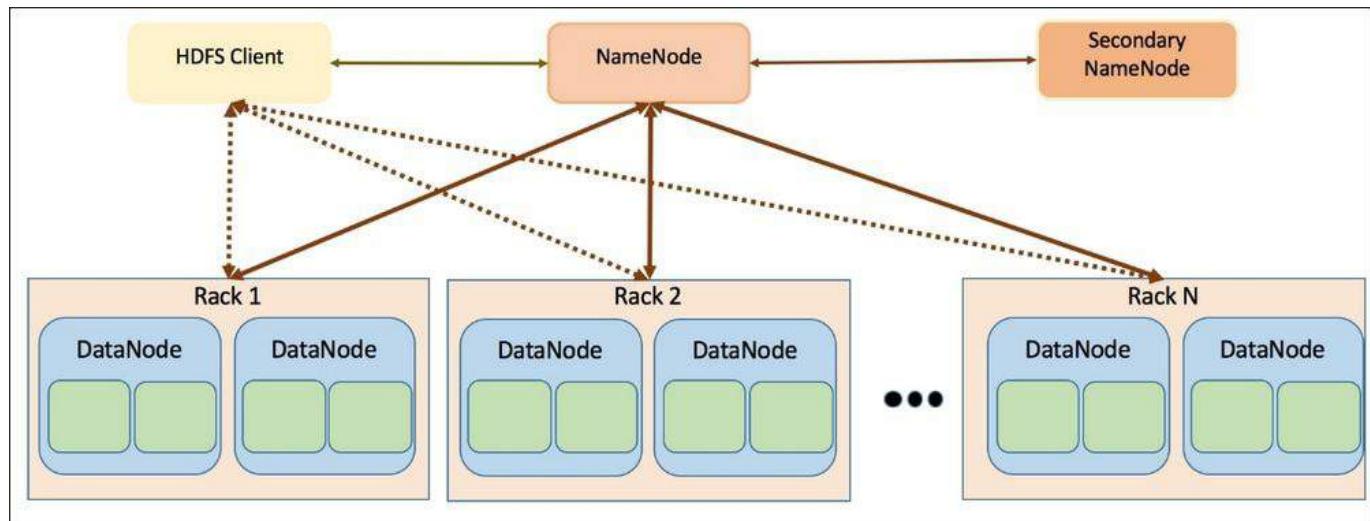
Data storage

The data storage component plays a key part in connecting the acquisition and the

rest of the components together. Performance, impact on data processing, cost, high-availability, ease of management, and so on, should be taken into consideration while deciding on data storage. For pure real-time or near real-time systems there are in-memory based frameworks for storage, but for batch-based systems there are mainly distributed File Systems such as HDFS or NoSQL.

HDFS

HDFS can run on a large cluster of nodes and provide all the important features such as high-throughput, replications, fail-over, and so on.



The basic architecture of HDFS has the following components:

- **NameNode:** The HDFS client always sends the request to the NameNode, which keeps the metadata of the file while the real data is distributed in blocks on the DataNodes. NameNodes are only responsible for handling opening and closing a file while the remaining interactions of reading, writing, and appending happen between clients and the data nodes. The NameNode stores the metadata in two files: `fsimage` and `edit` files. The `fsimage` contains the filesystem metadata as a snapshot, while `edit` files contain the incremental changes to the metadata.
- **Secondary NameNode:** Secondary NameNode provides redundancy to the metadata in the NameNode by keeping a copy of the `fsimage` and `edit` files at every predefined checkpoint.
- **DataNode:** DataNodes manage the actual blocks of data and facilitate read-write operation on these datablocks. DataNodes keep communicating with the

NameNodes using heartbeat signals indicating they are alive. The data blocks stored in DataNodes are also replicated for redundancy. Replication of the data blocks in the DataNodes is governed by the rack-aware placement policy.

NoSQL

Non-relational databases, also referred to as NoSQL databases, are gaining enormous popularity in the Big Data world. High throughput, better horizontal scaling, improved performance on retrieval, and storage at the cost of weaker consistency models are notable characteristics of most NoSQL databases. We will discuss some important forms of NoSQL database in this section along with implementations.

Key-value databases

Key-value databases are the most prominent NoSQL databases used mostly for semi-structured or unstructured data. As the name suggests, the structure of storage is quite basic, with unique keys associating the data values that can be of any type including string, integer, double precision, and so on—even BLOBS. Hashing the keys for quick lookup and retrieval of the values together with partitioning the data across multiple nodes gives high throughput and scalability. The query capabilities are very limited. Amazon DynamoDB, Oracle NoSQL, MemcacheDB, and so on, are examples of key-value databases.

Document databases

Document databases store semi-structured data in the form of XML, JSON, or YAML documents, to name some of the most popular formats. The documents have unique keys to which they are mapped. Though it is possible to store documents in key-value stores, the query capabilities offered by document stores are greater as the primitives making up the structure of the document—which may include names or attributes—can also be used for retrieval. When the data is ever-changing and has variable numbers or lengths of fields, document databases are often a good choice. Document databases do not offer join capabilities and hence all information needs to be captured in the document values. MongoDB, ElasticSearch, Apache Solr, and so on, are some well-known implementations of document databases.

Columnar databases

The use of columns as the basic unit of storage with name, value, and often timestamp, differentiates columnar databases from traditional relational databases. Columns are further combined to form column families. A row is indexed by the row key and has multiple column families associated with the row. Certain rows can use

only column families that are populated, giving it a good storage representation in sparse data. Columnar databases do not have fixed schema-like relational databases; new columns and families can be added at any time, giving them a significant advantage. **HBase**, **Cassandra**, and **Parquet** are some well-known implementations of columnar databases.

Graph databases

In many applications, the data has an inherent graph structure with nodes and links. Storing such data in graph databases makes it more efficient for storage, retrieval, and queries. The nodes have a set of attributes and generally represent entities, while links represent relationships between the nodes that can be directed or undirected. **Neo4J**, **OrientDB**, and **ArangoDB** are some well-known implementations of graph databases.

Data processing and preparation

The data preparation step involves various preprocessing steps before the data is ready to be consumed by analytics and machine learning algorithms. Some of the key tasks involved are:

- **Data cleansing:** Involves everything from correcting errors, type matching, normalization of elements, and so on, on the raw data.
- **Data scraping and curating:** Converting data elements and normalizing the data from one structure to another.
- **Data transformation:** Many analytical algorithms need features that are aggregates built on raw or historical data. Transforming and computing those extra features are done in this step.

Hive and HQL

Apache Hive (*References [11]*) is a powerful tool for performing various data preparation activities in HDFS systems. Hive organizes the underlying HDFS data a of structure that is similar to relational databases. HQL is like SQL and helps in performing various aggregates, transformations, cleanup, and normalization, and the data is then serialized back to HDFS. The logical tables in Hive are partitioned across and sub-divided into buckets for speed-up. Complex joins and aggregate queries in Hive are automatically converted into MapReduce jobs for throughput and speed-up.

Spark SQL

Spark SQL, which is a major component of Apache Spark (*References [1]* and *[2]*), provides SQL-like functionality—similar to what HQL provides—for performing changes to the Big Data. Spark SQL can work with underlying data storage systems such as Hive or NoSQL databases such as Parquet. We will touch upon some aspects of Spark SQL in the section on Spark later.

Amazon Redshift

Amazon Redshift provides several warehousing capabilities especially on Amazon EMR setups. It can process petabytes of data using its **massively parallel processing (MPP)** data warehouse architecture.

Real-time stream processing

In many Big Data deployments, processing and performing the transformations specified previously must be done on the stream of data in real time rather than from stored batch data. There are various **Stream Processing Engines (SPE)** such as Apache Storm (*References [12]*) and Apache Samza, and in-memory processing engines such as Spark-Streaming that are used for stream processing.

Machine Learning

Machine learning helps to perform descriptive, predictive, and prescriptive analysis on Big Data. There are two broad extremes that will be covered in this chapter:

- Machine learning can be done on batch historical data and then the learning/models can be applied to new batch/real-time data
- Machine learning can be done on real-time data and applied simultaneously to the real-time data

Both topics are covered at length in the remainder of this chapter.

Visualization and analysis

With batch learning done at modeling time and real-time learning done at runtime, predictions—the output of applying the models to new data—must be stored in some data structure and then analyzed by the users. Visualization tools and other reporting tools are frequently used to extract and present information to the users. Based on the domain and the requirements of the users, the analysis and visualization can be static, dynamic, or interactive.

Lightning is a framework for performing interactive visualizations on the Web with different binding APIs using REST for Python, R, Scala, and JavaScript languages.

Pygal and Seaborn are Python-based libraries that help in plotting all possible charts and graphs in Python for analysis, reporting, and visualizations.

Batch Big Data Machine Learning

Batch Big Data Machine Learning involves two basic steps, as discussed in [Chapter 2, Practical Approach to Real-World Supervised Learning](#), [Chapter 3, Unsupervised Machine Learning Techniques](#), and [Chapter 4, Semi-Supervised and Active Learning](#): learning or training data from historical datasets and applying the learned models to unseen future data. The following figure demonstrates the two environments along with the component tasks and some technologies/frameworks that accomplish them:

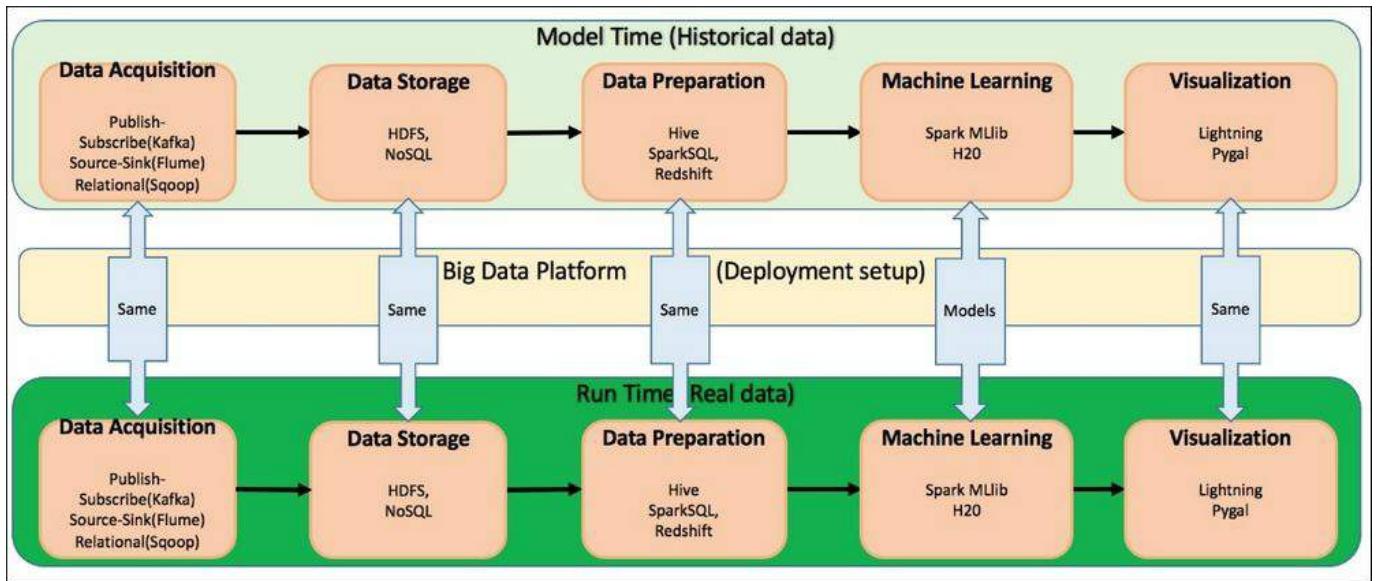


Figure 6: Model time and run time components for Big Data and providers

We will discuss two of the most well-known frameworks for doing Machine Learning in the context of batch data and will use the case study to highlight either the code or tools to perform modeling.

H2O as Big Data Machine Learning platform

H2O (*References* [13]) is a leading open source platform for Machine Learning at Big Data scale, with a focus on bringing AI to the enterprise. The company was founded in 2011 and counts several leading lights in statistical learning theory and optimization among its scientific advisors. It supports programming environments in multiple languages. While the H2O software is freely available, customer service and custom extensions to the product can be purchased.

H2O architecture

The following figure gives a high-level architecture of H2O with important components. H2O can access data from various data stores such as HDFS, SQL, NoSQL, and Amazon S3, to name a few. The most popular deployment of H2O is to use one of the deployment stacks discussed earlier with Spark or to run it in a H2O cluster itself.

The core of H2O is an optimized way of handling Big Data in memory, so that iterative algorithms that go through the same data can be handled efficiently and achieve good performance. Important Machine Learning algorithms in supervised and unsupervised learning are implemented specially to handle horizontal scalability across multiple nodes and JVMs. H2O provides not only its own user interface, known as flow, to manage and run modeling tasks, but also has different language bindings and connector APIs to Java, R, Python, and Scala.

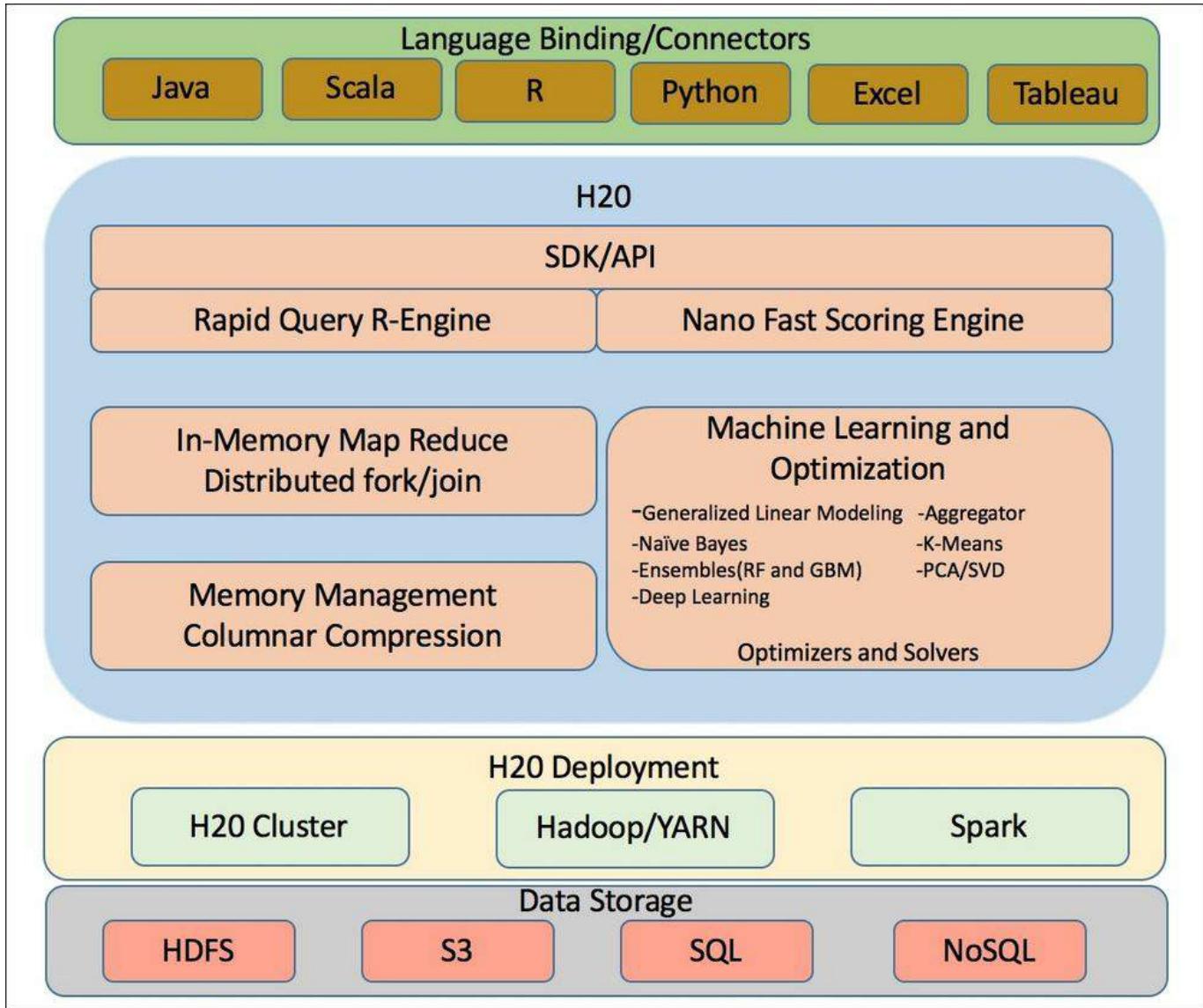


Figure 7: H2O high level architecture

Most Machine Learning algorithms, optimization algorithms, and utilities use the concept of fork-join or MapReduce. As shown in *Figure 8*, the entire dataset is considered as a **Data Frame** in H2O, and comprises vectors, which are features or columns in the dataset. The rows or instances are made up of one element from each Vector arranged side-by-side. The rows are grouped together to form a processing unit known as a **Chunk**. Several chunks are combined in one JVM. Any algorithmic or optimization work begins by sending the information from the topmost JVM to fork on to the next JVM, then on to the next, and so on, similar to the map operation in MapReduce. Each JVM works on the rows in the chunks to establish the task and

finally the results flow back in the reduce operation:

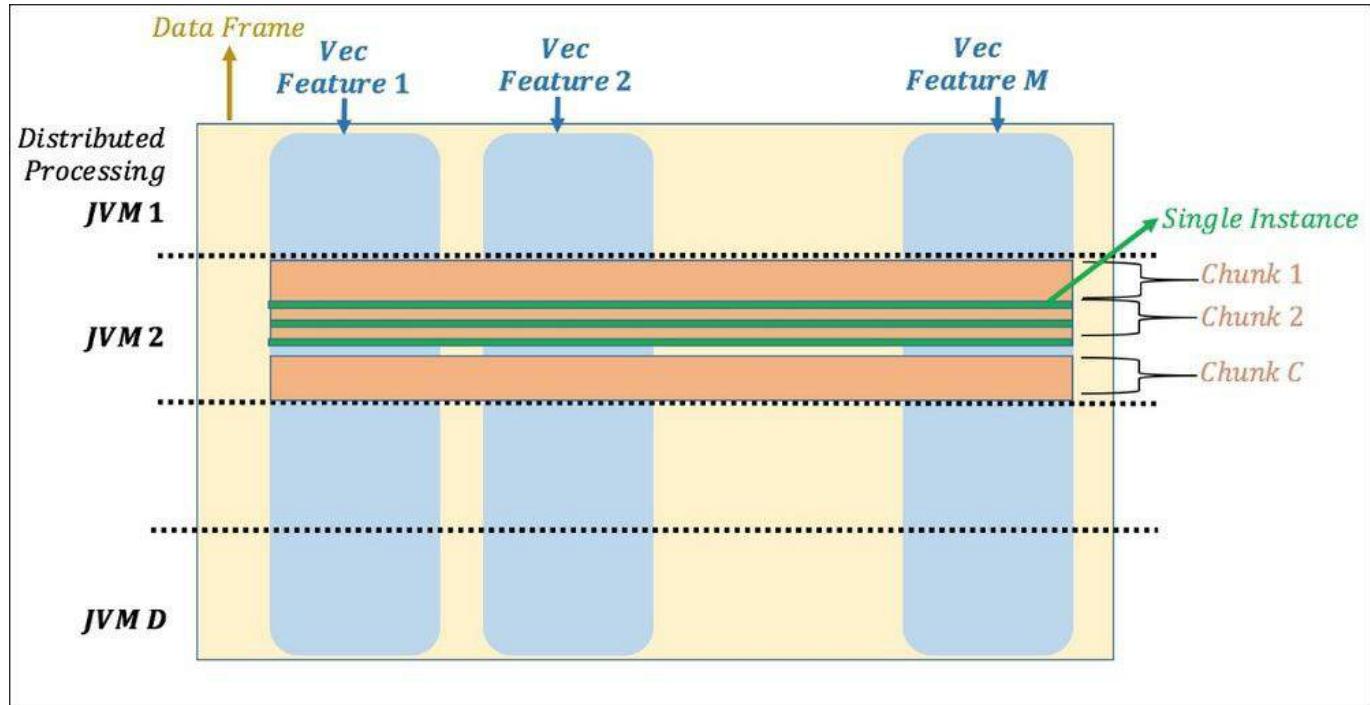


Figure 8: H2O distributed data processing using chunking

Machine learning in H2O

The following figure shows all the Machine Learning algorithms supported in H2O v3 for supervised and unsupervised learning:

Data Science Algorithms

Supervised Learning

- Generalized Linear Modeling (GLM)
- Gradient Boosting Machine (GBM)
- Deep Learning
- Distributed Random Forest
- Naive Bayes
- Ensembles (Stacking)

Unsupervised Learning

- Generalized Low Rank Models (GLRM)
- K-Means Clustering
- Principal Components Analysis (PCA)

Figure 9: H2O v3 Machine learning algorithms

Tools and usage

H2O Flow is an interactive web application that helps data scientists to perform various tasks from importing data to running complex models using point and click and wizard-based concepts.

H2O is run in local mode as:

```
java -Xmx6g -jar h2o.jar
```

The default way to start Flow is to point your browser and go to the following URL: <http://192.168.1.7:54321/>. The right-side of Flow captures every user action performed under the tab **OUTLINE**. The actions taken can be edited and saved as named flows for reuse and collaboration, as shown in *Figure 10*:

Figure 10: H2O Flow in the browser

Figure 11 shows the interface for importing files from the local filesystem or HDFS and displays detailed summary statistics as well as next actions that can be performed on the dataset. Once the data is imported, it gets a data frame reference in the H2O framework with the extension of .hex. The summary statistics are useful in understanding the characteristics of data such as **missing**, **mean**, **max**, **min**, and so on. It also has an easy way to transform the features from one type to another, for example, numeric features with a few unique values to categorical/nominal types known as `enum` in H2O.

The actions that can be performed on the datasets are:

1. Visualize the data.
 2. Split the data into different sets such as training, validation, and testing.
 3. Build supervised and unsupervised models.
 4. Use the models to predict.
 5. Download and export the files in various formats.

The screenshot shows the H2O AI Platform's interface for importing files. At the top, there's a search bar and a results section indicating 2 files found. Below that, a list of files is shown with checkboxes for selection. One file, 'covertype-train-sample.csv', is selected and highlighted with a red dashed border. This selection leads to a detailed view of its column summaries, which includes a table with columns for label, type, missing values, zeros, and various statistical measures like min, max, mean, and sigma. The table also lists actions for each column, such as 'Convert to enum'. The overall interface is clean with a white background and a grid-based layout.

Figure 11: Importing data as frames, summarizations, and actions that can be performed

Building supervised or unsupervised models in H2O is done through an interactive screen. Every modeling algorithm has its parameters classified into three sections: basic, advanced, and expert. Any parameter that supports hyper-parameter searches for tuning the model has a checkbox grid next to it, and more than one parameter value can be used.

Some basic parameters such as **training_frame**, **validation_frame**, and **response_column**, are common to every supervised algorithm; others are specific to model types, such as the choice of solver for GLM, the activation function for deep learning, and so on. All such common parameters are available in the basic section. Advanced parameters are settings that afford greater flexibility and control to the modeler if the default behavior must be overridden. Several of these parameters are

also common across some algorithms—two examples are the choice of method for assigning the fold index (if cross-validation was selected in the basic section), and selecting the column containing weights (if each example is weighted separately), and so on.

Expert parameters define more complex elements such as how to handle the missing values, model-specific parameters that need more than a basic understanding of the algorithms, and other esoteric variables. In *Figure 12*, GLM, a supervised learning algorithm, is being configured with 10-fold cross-validation, binomial (two-class) classification, efficient LBFGS optimization algorithm, and stratified sampling for cross-validation split:



Figure 12: Modeling algorithm parameters and validations

The model results screen contains a detailed analysis of the results using important evaluation charts, depending on the validation method that was used. At the top of the screen are possible actions that can be taken, such as to run the model on unseen data for prediction, download the model as POJO format, export the results, and so on.

Some of the charts are algorithm-specific, like the scoring history that shows how the

training loss or the objective function changes over the iterations in GLM—this gives the user insight into the speed of convergence as well as into the tuning of the iterations parameter. We see the ROC curves and the Area Under Curve metric on the validation data in addition to the gains and lift charts, which give the cumulative capture rate and cumulative lift over the validation sample respectively.

Figure 13 shows **SCORING HISTORY**, **ROC CURVE**, and **GAINS/LIFT** charts for GLM on 10-fold cross-validation on the `CoverType` dataset:

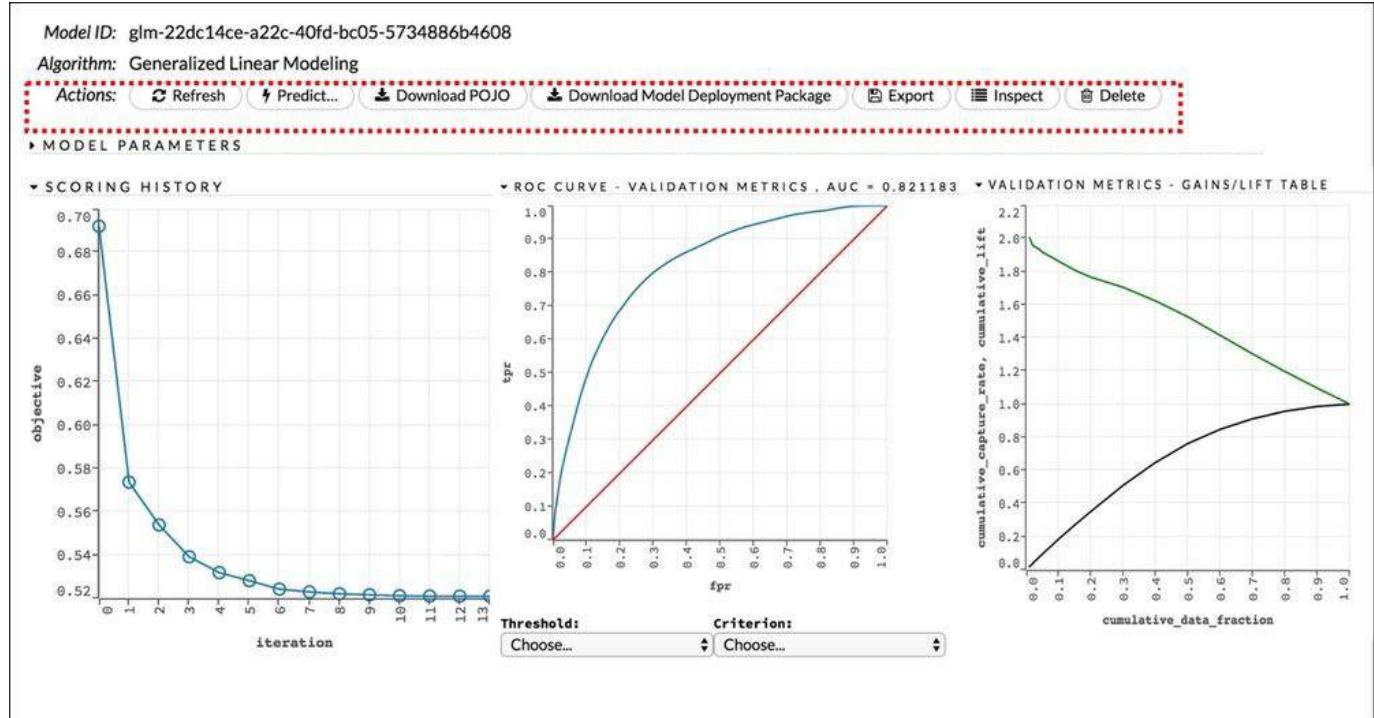


Figure 13: Modeling and validation ROC curves, objective functions, and lift/gain charts

The output of validation gives detailed evaluation measures such as accuracy, AUC, err, errors, f1 measure, MCC (Mathews Correlation Coefficient), precision, and recall for each validation fold in the case of cross-validation as well as the mean and standard deviation computed across all.

▼ OUTPUT - CROSS-VALIDATION METRICS SUMMARY													
	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid	cv_6_valid	cv_7_valid	cv_8_valid	cv_9_valid	cv_mean	cv_sd
accuracy	0.7411992	0.0020919477	0.73774326	0.7429334	0.7448235	0.7381085	0.7430922	0.7459533	0.73845685	0.7381071	0.7431965	0.7405446	0.0020919477
auc	0.8211171	0.0011220425	0.81931686	0.82393146	0.8218799	0.82080626	0.8215902	0.82210237	0.8197568	0.8186006	0.82291764	0.8211171	0.0011220425
err	0.25880077	0.0020919477	0.26225677	0.2570666	0.25517648	0.26189148	0.25690782	0.25404665	0.26154318	0.26189288	0.2568035	0.25880077	0.0020919477
err_count	13532.7	181.7846	13801.0	13487.0	13396.0	13616.0	13398.0	13372.0	13657.0	13664.0	13362.0	13532.7	181.7846
f0point5	0.7140929	0.0025237158	0.70960116	0.714942	0.71712357	0.71209085	0.7179527	0.71962744	0.7101482	0.7093464	0.7170292	0.7140929	0.0025237158
f1	0.75218046	0.001344509	0.75220394	0.755657	0.7514657	0.75472856	0.7509017	0.75085706	0.7507892	0.7490542	0.753505	0.75218046	0.001344509
f2	0.79461336	0.004083693	0.8002491	0.80128944	0.7892624	0.8027975	0.78702044	0.78492016	0.79636425	0.7934711	0.79389083	0.79461336	0.004083693
lift_top_group	2.0038874	0.0143965315	1.9922546	2.040354	2.012063	1.9808673	1.9791358	2.0059962	1.9857894	2.0164146	2.0331487	2.0038874	0.0143965315
logloss	0.5208205	0.0013281262	0.5231053	0.51792485	0.51960087	0.5214477	0.5202088	0.5187702	0.5226515	0.52344406	0.5189758	0.5208205	0.0013281262
max_per_class_error	0.33552998	0.009792185	0.3514494	0.34064615	0.31948155	0.3546882	0.32055646	0.31098375	0.34422165	0.34093636	0.32979506	0.33552998	0.009792185
mcc	0.49442172	0.002846507	0.4905058	0.499489	0.4993631	0.49085903	0.49499753	0.49987337	0.4996272	0.48938003	0.49728892	0.49442172	0.002846507
mean_per_class_accuracy	0.7450862	0.0018750997	0.74219555	0.7471276	0.7485821	0.7418539	0.7462687	0.74928785	0.7428648	0.74259645	0.7467569	0.7450862	0.0018750997
mean_per_class_error	0.2549138	0.0018750997	0.25780442	0.2528724	0.25147185	0.2581461	0.25373128	0.25079215	0.25713524	0.25740358	0.2532431	0.2549138	0.0018750997
mse	0.17293216	5.2172324E-4	0.17375539	0.1715886	0.17254509	0.17318714	0.17266512	0.17245334	0.17356905	0.17399642	0.17212443	0.17293216	5.2172324E-4
null_deviance	72367.18	215.07042	72833.43	72612.14	72618.93	72002.8	72178.836	72816.945	72254.55	72179.43	72019.37	72367.18	215.07042
precision	0.69079375	0.003936701	0.6837827	0.6901516	0.6959211	0.68624496	0.6975475	0.700212	0.6854135	0.6851335	0.6946126	0.69079375	0.003936701
r2	0.30661532	0.0021007422	0.30340376	0.3120748	0.30770847	0.30620724	0.3077754	0.30816406	0.30394053	0.30199826	0.31000972	0.30661532	0.0021007422
recall	0.82570237	0.0066806898	0.8358405	0.83490133	0.81664586	0.83839595	0.8130939	0.8093995	0.82995117	0.82612926	0.8233089	0.82570237	0.0066806898
residual_deviance	54468.254	192.93034	55055.793	54345.855	54554.977	54221.18	54258.812	54611.98	54582.586	54620.344	54006.695	54468.254	192.93034
rmse	0.4158502	6.274607E-4	0.41683978	0.41423255	0.41538548	0.4161576	0.4155299	0.415275	0.4166162	0.41712877	0.41487882	0.4158502	6.274607E-4
specificity	0.66447	0.009792185	0.64855057	0.65935385	0.68051845	0.64531183	0.67944354	0.6890163	0.65577835	0.65906364	0.67020494	0.66447	0.009792185

Figure 14: Validation results and summary

The prediction action runs the model using unseen held-out data to estimate the out-of-sample performance. Important measures such as errors, accuracy, area under curve, ROC plots, and so on, are given as the output of predictions that can be saved or exported.

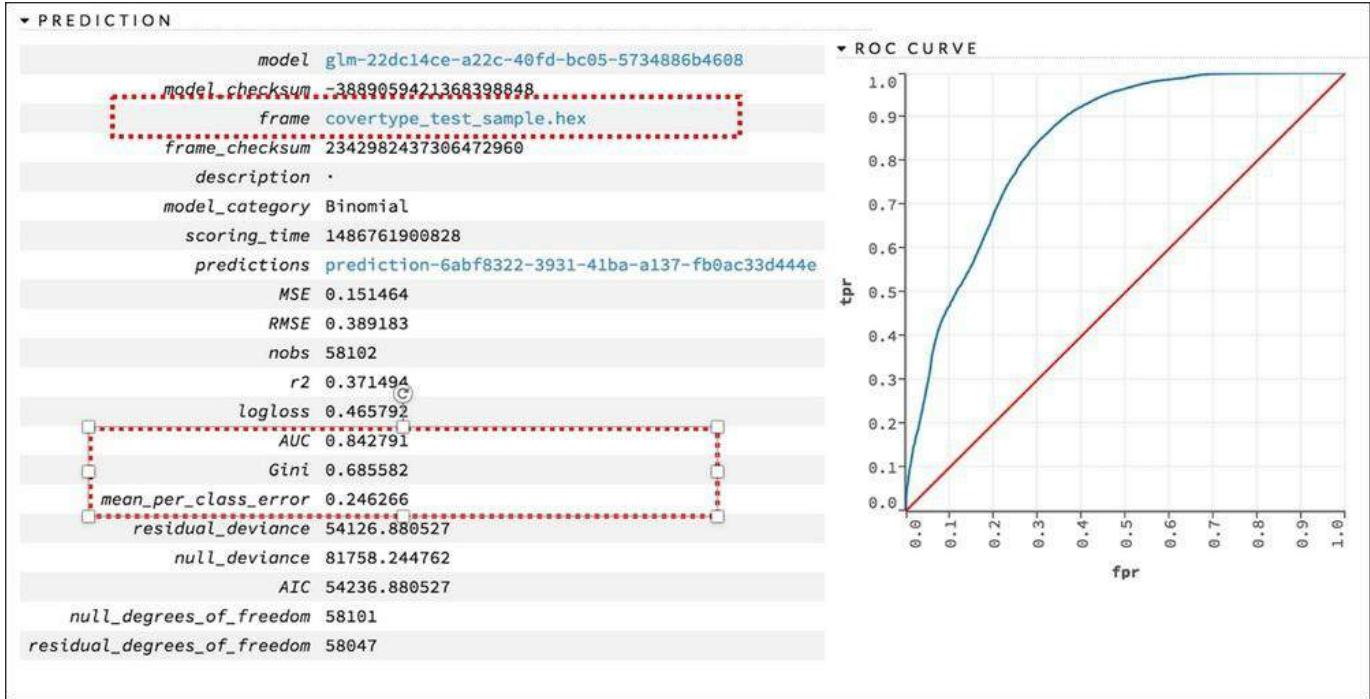


Figure 15: Running test data, predictions, and ROC curves

Case study

In this case study, we use the CoverType dataset to demonstrate classification and clustering algorithms from H2O, Apache Spark MLlib, and SAMOA Machine Learning libraries in Java.

Business problem

The CoverType dataset available from the UCI machine learning repository (<https://archive.ics.uci.edu/ml/datasets/Covertype>) contains unscaled cartographic data for 581,012 cells of forest land 30 x 30 m² in dimension, accompanied by actual forest cover type labels. In the experiments conducted here, we use the normalized version of the data. Including one-hot encoding of two categorical types, there are a total of 54 attributes in each row.

Machine Learning mapping

First, we treat the problem as one of classification using the labels included in the dataset and perform several supervised learning experiments. With the models generated, we make predictions about the forest cover type of an unseen held out test dataset. For the clustering experiments that follow, we ignore the data labels, determine the number of clusters to use, and then report the corresponding cost using various algorithms implemented in H2O and Spark MLLib.

Data collection

This dataset was collected using cartographic measurements only and no remote sensing. It was derived from data originally collected by the **US Forest Service (USFS)** and the **US Geological Survey (USGS)**.

Data sampling and transformation

Train and test data—The dataset was split into two sets in the ratio 20% for testing and 80% for training.

The categorical Soil Type designation was represented by 40 binary variable attributes. A value of 1 indicates the presence of a soil type in the observation; a 0 indicates its absence.

The wilderness area designation is likewise a categorical attribute with four binary columns, with 1 indicating presence and 0 absence.

All continuous value attributes have been normalized prior to use.

Experiments, results, and analysis

In the first set of experiments in this case study, we used the H2O framework.

Feature relevance and analysis

Though H2O doesn't have explicit feature selection algorithms, many learners such as GLM, random forest, GBT, and so on, give feature importance metrics based on training/validation of the models. In our analysis, we have used GLM for feature selection, as shown in *Figure 16*. It is interesting that the feature **Elevation** emerges as the most discriminating feature along with some categorical features that are converted into numeric/binary such as **Soil_Type2**, **Soil_Type4**, and so on. Many of the soil type categorical features have no relevance and can be dropped from the modeling perspective.

Learning algorithms included in this set of experiments were: **Generalized Linear Models (GLM)**, **Gradient Boosting Machine (GBM)**, **Random Forest (RF)**, **Naïve Bayes (NB)**, and **Deep Learning (DL)**. The deep learning model supported by H2O is the **multi-layered perceptron (MLP)**.

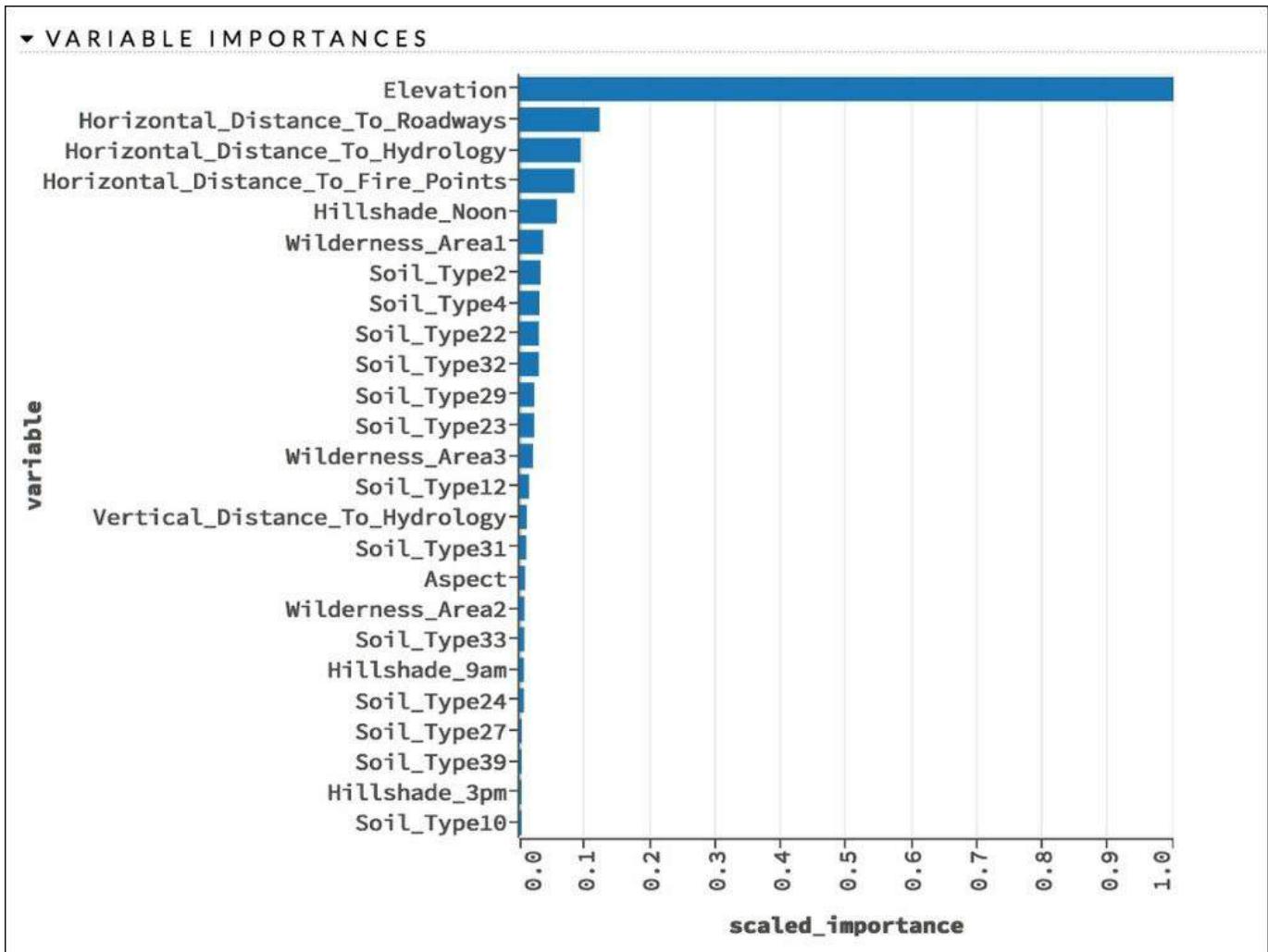


Figure 16: Feature selection using GLM

Evaluation on test data

The results using all the features are shown in the table:

Algorithm	Parameters	AUC	Max Accuracy	Max F1	Max Precision	Max Recall	Max Specificity
GLM	Default	0.84	0.79	0.84	0.98	1.0(1)	0.99
GBM	Default	0.86	0.82	0.86	1.0(1)	1.0(1)	1.0(1)
Random Forest (RF)	Default	0.88(1)	0.83(1)	0.87(1)	0.97	1.0(1)	0.99
Naïve Bayes (NB)	Laplace=50	0.66	0.72	0.81	0.68	1.0(1)	0.33

Deep Learning (DL)	Rect,300, 300,Dropout	0.	0.78	0.83	0.88	1.0(1)	0.99
Deep Learning (DL)	300, 300,MaxDropout	0.82	0.8	0.84	1.0(1)	1.0(1)	1.0(1)

The results after removing features not scoring well in feature relevance were:

Algorithm	Parameters	AUC	Max Accuracy	Max F1	Max Precision	Max Recall	Max Specificity
GLM	Default	0.84	0.80	0.85	1.0	1.0	1.0
GBM	Default	0.85	0.82	0.86	1.0	1.0	1.0
Random Forest (RF)	Default	0.88	0.83	0.87	1.0	1.0	1.0
Naïve Bayes (NB)	Laplace=50	0.76	0.74	0.81	0.89	1.0	0.95
Deep Learning (DL)	300,300, RectDropout	0.81	0.79	0.84	1.0	1.0	1.0
Deep Learning (DL)	300, 300, MaxDropout	0.85	0.80	0.84	0.89	0.90	1.0

Table 1: Model evaluation results with all features included

Analysis of results

The main observations from an analysis of the results obtained are quite instructive and are presented here.

1. The feature relevance analysis shows how the **Elevation** feature is a highly discriminating feature, whereas many categorical attributes converted to binary features, such as **SoilType_10**, and so on, have near-zero to no relevance.
2. The results for experiments with all features included, shown in *Table 1*, clearly indicate that the non-linear ensemble technique Random Forest is the best algorithm as shown by the majority of the evaluation metrics including accuracy, F1, AUC, and recall.
3. *Table 1* also highlights the fact that whereas the faster, linear Naive Bayes algorithm may not be best-suited, GLM, which also falls in the category of linear algorithms, demonstrates much better performance—this points to some inter-dependence among features!

4. As we saw in [Chapter 7](#), *Deep Learning*, algorithms used in deep learning typically need a lot of tuning; however, even with a few small tuning changes, the results from DL are comparable to Random Forest, especially with MaxDropout.
5. *Table 2* shows the results of all the algorithms after removing low-relevance features from the training set. It can be seen that Naive Bayes—which has the most impact due to multiplication of probabilities based on the assumption of independence between features—gets the most benefit and highest uplift in performance. Most of the other algorithms such as Random Forest have inbuilt feature selection as we discussed in [Chapter 2](#), *Practical Approach to Real-World Supervised Learning*, and as a result removing the unimportant features has little or no effect on their performance.

Spark MLlib as Big Data Machine Learning platform

Apache Spark, started in 2009 at AMPLab at UC Berkley, was donated to Apache Software Foundation in 2013 under Apache License 2.0. The core idea of Spark was to build a cluster computing framework that would overcome the issues of Hadoop, especially for iterative and in-memory computations.

Spark architecture

The Spark stack as shown in *Figure 17* can use any kind of data stores such as HDFS, SQL, NoSQL, or local filesystems. It can be deployed on Hadoop, Mesos, or even standalone.

The most important component of Spark is the Spark Core, which provides a framework to handle and manipulate the data in a high-throughput, fault-tolerant, and scalable manner.

Built on top of Spark core are various libraries each meant for various functionalities needed in processing data and doing analytics in the Big Data world. Spark SQL gives us a language for performing data manipulation in Big Data stores using a querying language very much like SQL, the *lingua franca* of databases. Spark GraphX provides APIs to perform graph-related manipulations and graph-based algorithms on Big Data. Spark Streaming provides APIs to handle real-time operations needed in stream processing ranging from data manipulations to queries on the streams.

Spark-MLLib is the Machine Learning library that has an extensive set of Machine Learning algorithms to perform supervised and unsupervised tasks from feature selection to modeling. Spark has various language bindings such as Java, R, Scala, and Python. MLLib has a clear advantage running on top of the Spark engine, especially because of caching data in memory across multiple nodes and running MapReduce jobs, thus improving performance as compared to Mahout and other large-scale Machine Learning engines by a significant factor. MLLib also has other advantages such as fault tolerance and scalability without explicitly managing it in the Machine Learning algorithms.

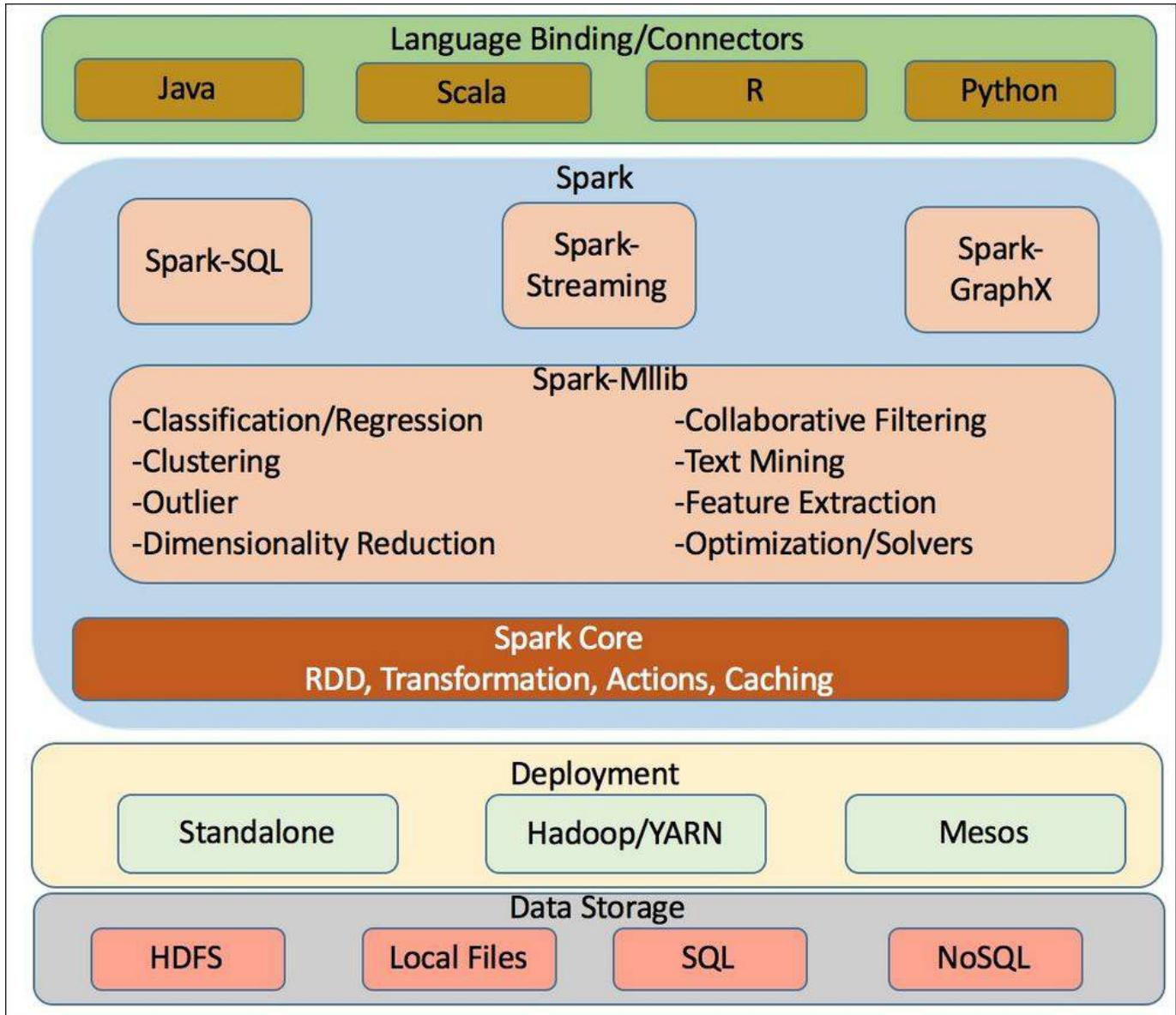


Figure 17: Apache Spark high level architecture

The Spark core has the following components:

- **Resilient Distributed Datasets (RDD)**: RDDs are the basic immutable collection of objects that Spark Core knows how to partition and distribute across the cluster for performing tasks. RDDs are composed of "partitions", dependent on parent RDDs and metadata about data placement.
- Two distinct operations are performed on RDDs:
 - **Transformations**: Operations that are lazily evaluated and transform one RDD into another. Lazy evaluation defers evaluation as long as possible,

which makes some resource optimizations possible.

- **Action:** The actual operation that triggers transformations and returns output values
- **Lineage graph:** The pipeline or flow of data describing the computation for a particular task, including different RDDs created in transformations and actions is known as the lineage graph of the task. The lineage graph plays a key role in fault tolerance.

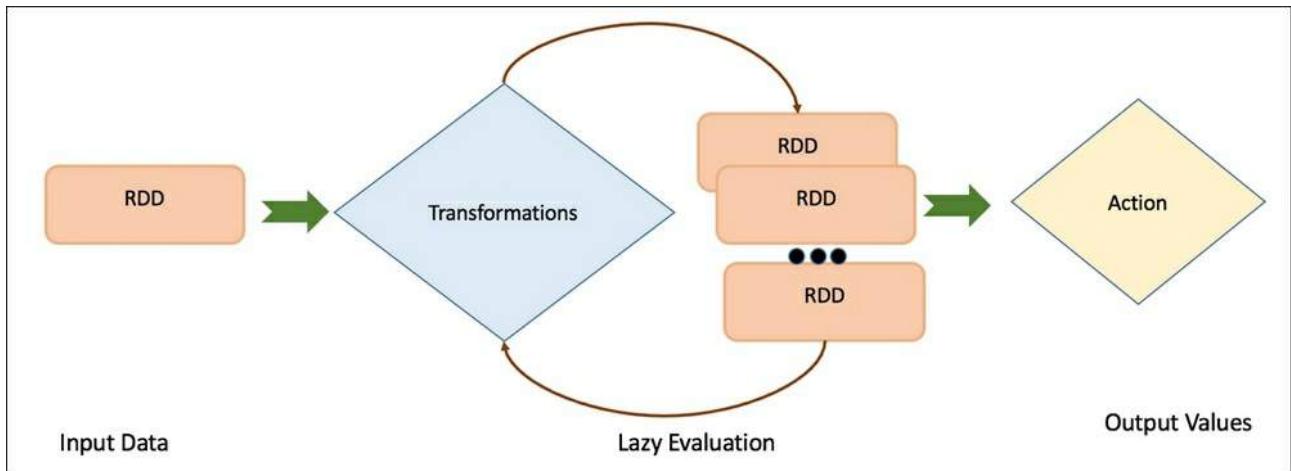


Figure 18: Apache Spark lineage graph

Spark is agnostic to the cluster management and can work with several implementations—including YARN and Mesos—for managing the nodes, distributing the work, and communications. The distribution of tasks in Transformations and Actions across the cluster is done by the scheduler, starting from the driver node where the Spark context is created, to the many worker nodes as shown in *Figure 19*. When running with YARN, Spark gives the user the choice of the number of executors, heap, and core allocation per JVM at the node level.

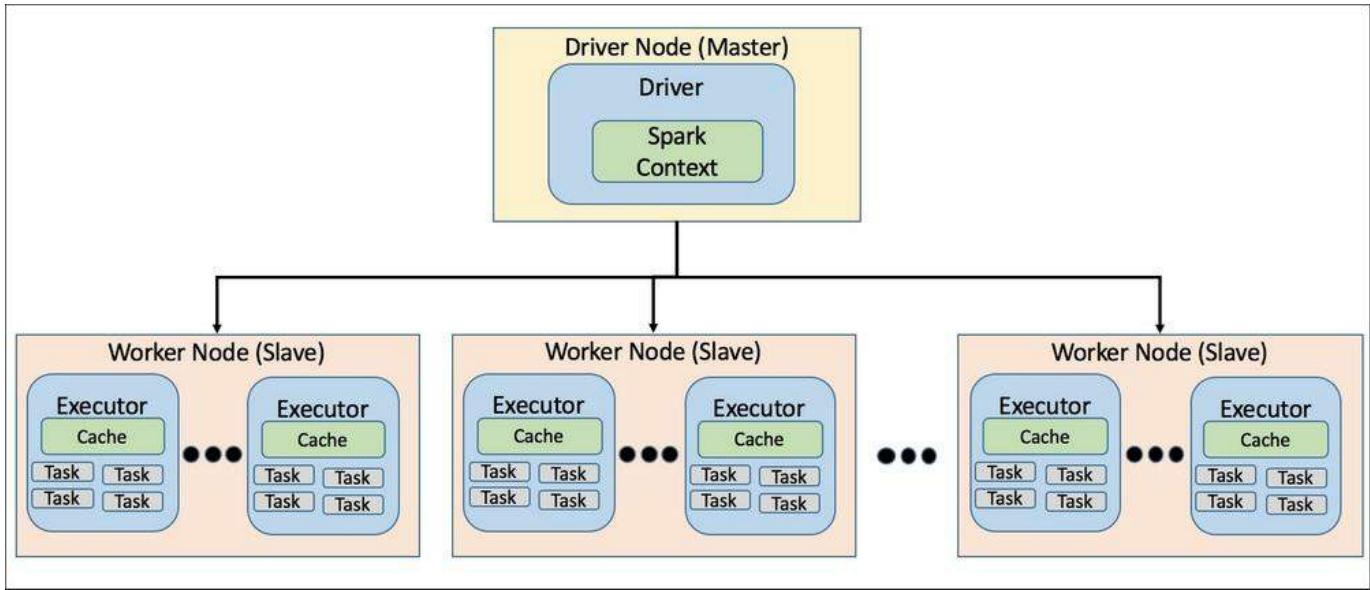


Figure 19: Apache Spark cluster deployment and task distribution

Machine Learning in MLlib

Spark MLlib has a comprehensive Machine Learning toolkit, offering more algorithms than H2O at the time of writing, as shown in *Figure 20*:

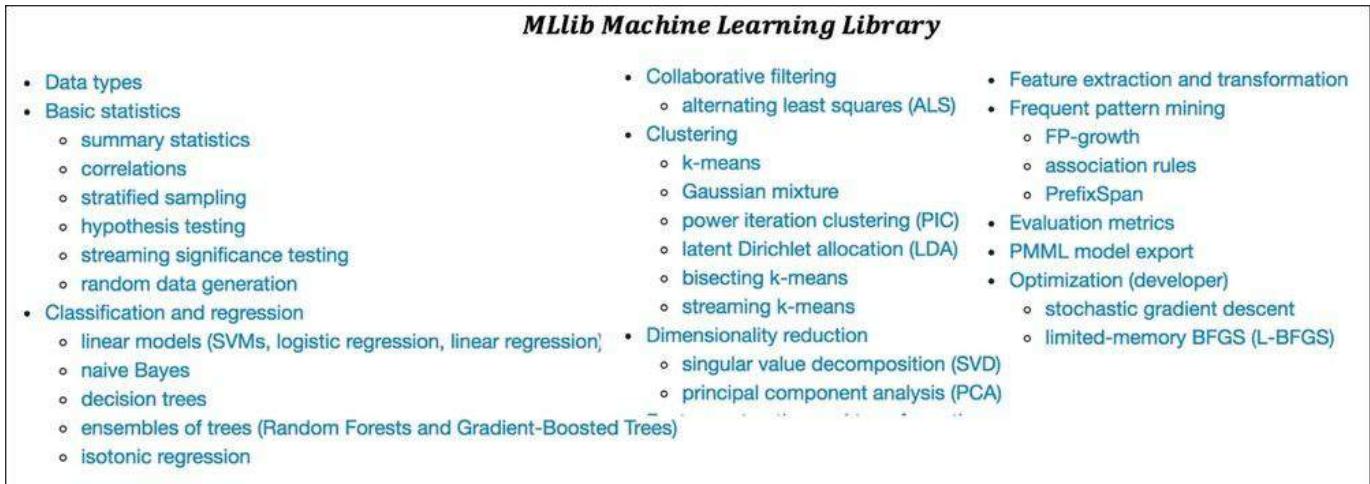


Figure 20: Apache Spark MLlib machine learning algorithms

Many extensions have been written for Spark, including Spark MLlib, and the user

community continues to contribute more packages. You can download third-party packages or register your own at <https://spark-packages.org/>.

Tools and usage

Spark MLlib provides APIs for other languages in addition to Java, including Scala, Python, and R. When a `SparkContext` is created, it launches a monitoring and instrumentation web console at port 4040, which lets us see key information about the runtime, including scheduled tasks and their progress, RDD sizes and memory use, and so on. There are also external profiling tools available for use.

Experiments, results, and analysis

The business problem we tackled here is the same as the one described earlier for experiments using H2O. We employed five learning algorithms using MLlib, in all. The first was k-Means with all features using a k value determined from computing the cost—specifically, the **Sum of Squared Errors (SSE)**—over a large number of values of k and selecting the "elbow" of the curve. Determining the optimal value of k is typically not an easy task; often, evaluation measures such as silhouette are compared in order to pick the best k . Even though we know the number of classes in the dataset is 7, it is instructive to see where experiments like this lead if we pretend we did not have labeled data. The optimal k found using the elbow method was 27. In the real world, business decisions may often guide the selection of k .

In the following listings, we show how to use different models from the MLlib suite to do cluster analysis and classification. The code is based on examples available in the MLlib API Guide (<https://spark.apache.org/docs/latest/mllib-guide.html>). We use the normalized UCI `CoverType` dataset in CSV format. Note that it is more natural to use `spark.sql.Dataset` with the newer `spark.ml` package, whereas the `spark.mllib` package works more closely with `JavaRDD`. This provides an abstraction over RDDs and allows for optimization of the transformations beneath the covers. In the case of most unsupervised learning algorithms, this means the data must be transformed such that the dataset to be used for training and testing should have a column called `features` by default that contains all the features of an observation as a vector. A `VectorAssembler` object can be used for this transformation. A glimpse into the use of ML pipelines, which is a way to chain tasks together, is given in the source code for training a Random Forest classifier.

k-Means

The following code fragment for the k-Means experiment uses the algorithm from the `org.apache.spark.ml.clustering` package. The code includes minimal boilerplate for setting up the `SparkSession`, which is the handle to the Spark runtime. Note that eight cores have been specified in local mode in the setup:

```
SparkSession spark = SparkSession.builder()
    .master("local[8]")
    .appName("KMeansExpt")
    .getOrCreate();

// Load and parse data
String filePath =
"/home/kchoppella/book/Chapter09/data/covtypeNorm.csv";
// Selected K value
int k = 27;

// Loads data.
Dataset<Row> inDataset = spark.read()
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .option("inferSchema", true)
    .load(filePath);
ArrayList<String> inputColsList = new ArrayList<String>
((Arrays.asList(inDataset.columns())));

//Make single features column for feature vectors
inputColsList.remove("class");
String[] inputCols =
inputColsList.parallelStream().toArray(String[]::new);

//Prepare dataset for training with all features in "features" column
VectorAssembler assembler = new
VectorAssembler().setInputCols(inputCols).setOutputCol("features");
Dataset<Row> dataset = assembler.transform(inDataset);

KMeans kmeans = new KMeans().setK(k).setSeed(1L);
KMeansModel model = kmeans.fit(dataset);

// Evaluate clustering by computing Within Set Sum of Squared Errors.
double SSE = model.computeCost(dataset);
System.out.println("Sum of Squared Errors = " + SSE);

spark.stop();
```

The optimal value for the number of clusters was arrived at by evaluating and plotting the sum of squared errors for several different values and choosing the one

at the elbow of the curve. The value used here is 27.

k-Means with PCA

In the second experiment, we used k-Means again, but first we reduced the number of dimensions in the data through PCA. Again, we used a rule of thumb here, which is to select a value for the PCA parameter for the number of dimensions such that at least 85% of the variance in the original dataset is preserved after the reduction in dimensionality. This produced 16 features in the transformed dataset from an initial 54, and this dataset was used in this and subsequent experiments. The following code shows the relevant code for PCA analysis:

```
int numDimensions = 16
PCAModel pca = new PCA()
    .setK(numDimensions)
    .setInputCol("features")
    .setOutputCol("pcaFeatures")
    .fit(dataset);

Dataset<Row> result = pca.transform(dataset).select("pcaFeatures");
KMeans kmeans = new KMeans().setK(k).setSeed(1L);
KMeansModel model = kmeans.fit(dataset);
```

Bisecting k-Means (with PCA)

The third experiment used MLlib's Bisecting k-Means algorithm. This algorithm is similar to a top-down hierarchical clustering technique where all instances are in the same cluster at the outset, followed by successive splits:

```
// Trains a bisecting k-Means model.
BisectingKMeans bkm = new BisectingKMeans().setK(k).setSeed(1);
BisectingKMeansModel model = bkm.fit(dataset);
```

Gaussian Mixture Model

In the next experiment, we used MLlib's **Gaussian Mixture Model (GMM)**, another clustering model. The assumption inherent to this model is that the data distribution in each cluster is Gaussian in nature, with unknown parameters. The same number of clusters is specified here, and default values have been used for the maximum number of iterations and tolerance, which dictate when the algorithm is considered to have converged:

```
GaussianMixtureModel gmm = new GaussianMixture()
    .setK(numClusters)
    .fit(result);
```

```

// Output the parameters of the mixture model
for (int k = 0; k < gmm.getK(); k++) {
    String msg = String.format("Gaussian
%d:\nweight=%f\nmu=%s\nsigma=\n%s\n\n",
        k, gmm.weights() [k], gmm.gaussians() [k].mean(),
        gmm.gaussians() [k].cov());
    System.out.printf(msg);
    writer.write(msg + "\n");
    writer.flush();
}

```

Random Forest

Finally, we ran Random Forest, which is the only available ensemble learner that can handle multi-class classification. In the following code, we see that this algorithm needs some preparatory tasks to be performed prior to training. Pre-processing stages are composed into a pipeline of Transformers and Estimators. The pipeline is then used to fit the data. You can learn more about Pipelines on the Apache Spark website (<https://spark.apache.org/docs/latest/ml-pipeline.html>):

```

// Index labels, adding metadata to the label column.
// Fit on whole dataset to include all labels in index.
StringIndexerModel labelIndexer = new StringIndexer()
    .setInputCol("class")
    .setOutputCol("indexedLabel")
    .fit(dataset);
// Automatically identify categorical features, and index them.
// Set maxCategories so features with > 2 distinct values are treated
as continuous since we have already encoded categoricals with sets of
binary variables.
VectorIndexerModel featureIndexer = new VectorIndexer()
    .setInputCol("features")
    .setOutputCol("indexedFeatures")
    .setMaxCategories(2)
    .fit(dataset);

// Split the data into training and test sets (30% held out for
testing)
Dataset<Row>[] splits = dataset.randomSplit(new double[] {0.7, 0.3});
Dataset<Row> trainingData = splits[0];
Dataset<Row> testData = splits[1];

// Train a RF model.
RandomForestClassifier rf = new RandomForestClassifier()
    .setLabelCol("indexedLabel")
    .setFeaturesCol("indexedFeatures")
    .setImpurity("gini")

```

```

.setMaxDepth(5)
.setNumTrees(20)
.setSeed(1234);

// Convert indexed labels back to original labels.
IndexToString labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(labelIndexer.labels());

// Chain indexers and RF in a Pipeline.
Pipeline pipeline = new Pipeline()
  .setStages(new PipelineStage[] {labelIndexer, featureIndexer, rf,
labelConverter});

// Train model. This also runs the indexers.
PipelineModel model = pipeline.fit(trainingData);

// Make predictions.
Dataset<Row> predictions = model.transform(testData);

// Select example rows to display.
predictions.select("predictedLabel", "class", "features").show(5);

// Select (prediction, true label) and compute test error.
MulticlassClassificationEvaluator evaluator = new
MulticlassClassificationEvaluator()
  .setLabelCol("indexedLabel")
  .setPredictionCol("prediction");

evaluator.setMetricName("accuracy");
double accuracy = evaluator.evaluate(predictions);
System.out.printf("Accuracy = %f\n", accuracy);

```

The sum of squared errors for the experiments using k-Means and Bisecting k-Means are given in the following table:

Algorithm	k	Features	SSE
k-Means	27	54	214,702
k-Means(PCA)	27	16	241,155
Bisecting k-Means(PCA)	27	16	305,644

Table 3: Results with k-Means

The GMM model was used to illustrate the use of the API; it outputs the parameters of the Gaussian mixture for every cluster as well as the cluster weight. Output for all the clusters can be seen at the website for this book.

For the case of Random Forest these are the results for runs with different numbers of trees. All 54 features were used here:

Number of trees	Accuracy	F1 measure	Weighted precision	Weighted recall
15	0.6806	0.6489	0.6213	0.6806
20	0.6776	0.6470	0.6191	0.6776
25	0.5968	0.5325	0.5717	0.5968
30	0.6547	0.6207	0.5972	0.6547
40	0.6594	0.6272	0.6006	0.6594

Table 4: Results for Random Forest

Analysis of results

As can be seen from *Table 3*, there is a small increase in cost when fewer dimensions are used after PCA with the same number of clusters. Varying k with PCA might suggest a better k for the PCA case. Notice also that in this experiment, for the same k , Bisecting K-Means with PCA-derived features has the highest cost of all. The stopping number of clusters used for Bisecting k-Means has simply been picked to be the one determined for basic k-Means, but this need not be so. A similar search for k that yields the best cost may be done independently for Bisecting k-Means.

In the case of Random Forest, we see the best performance when using 15 trees. All trees have a depth of three. This hyper-parameter can be varied to tune the model as well. Even though Random Forest is not susceptible to over-fitting due to accounting for variance across trees in the training stages, increasing the value for the number of trees beyond an optimum number can degrade performance.

Real-time Big Data Machine Learning

In this section, we will discuss the real-time version of Big Data Machine Learning where data arrives in large volumes and is changing at a rapid rate at the same time. Under these conditions, Machine Learning analytics cannot be applied *per* the traditional practice of "batch learning and deploy" (*References [14]*).

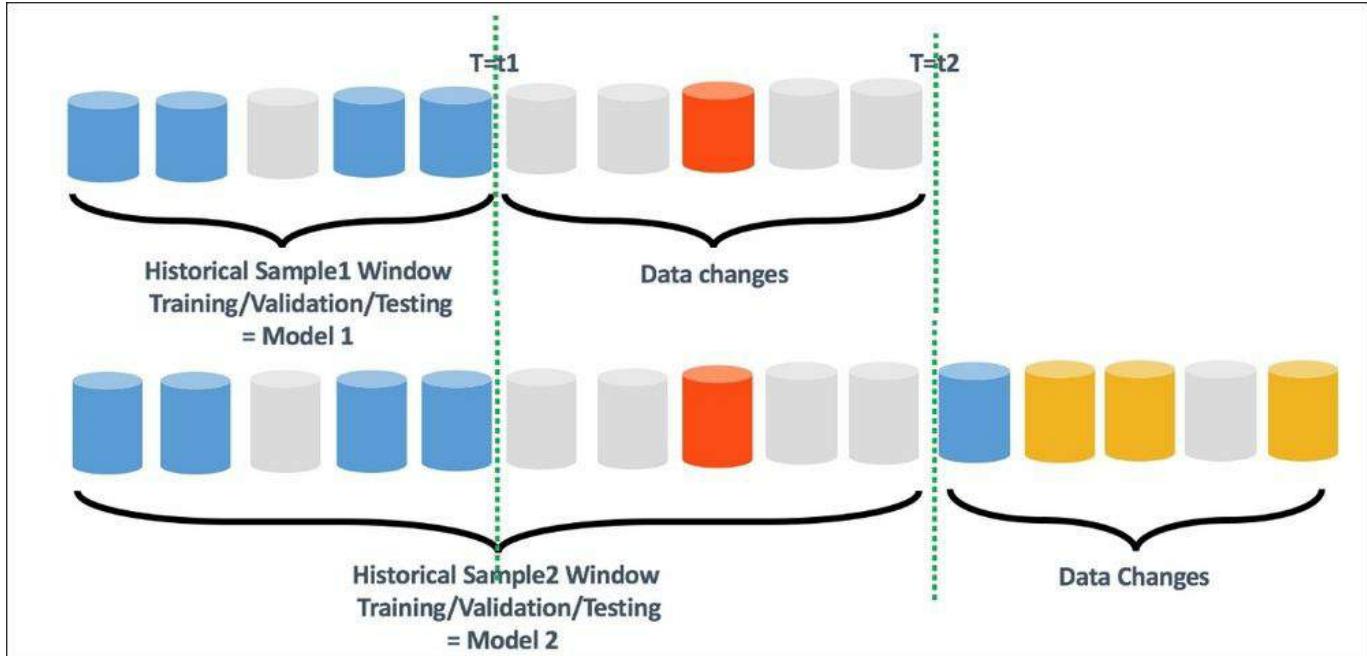


Figure 21: Use case for real-time Big Data Machine Learning

Let us consider a case where labeled data is available for a short duration, and we perform the appropriate modeling techniques on the data and then apply the most suitable evaluation methods on the resulting models. Next, we select the best model and use it for predictions on unseen data at runtime. We then observe, with some dismay, that model performance drops significantly over time. Repeating the exercise with new data shows a similar degradation in performance! What are we to do now? This quandary, combined with large volumes of data motivates the need for a different approach: real-time Big Data Machine Learning.

Like the batch learning framework, the real-time framework in big data may have similar components up until the data preparation stage. When the computations involved in data preparation must take place on streams or combined stream and batch data, we require specialized computation engines such as **Spark Streaming**. Like stream computations, Machine Learning must work across the cluster and perform different Machine Learning tasks on the stream. This adds an additional

layer of complexity to the implementations of single machine multi-threaded streaming algorithms.

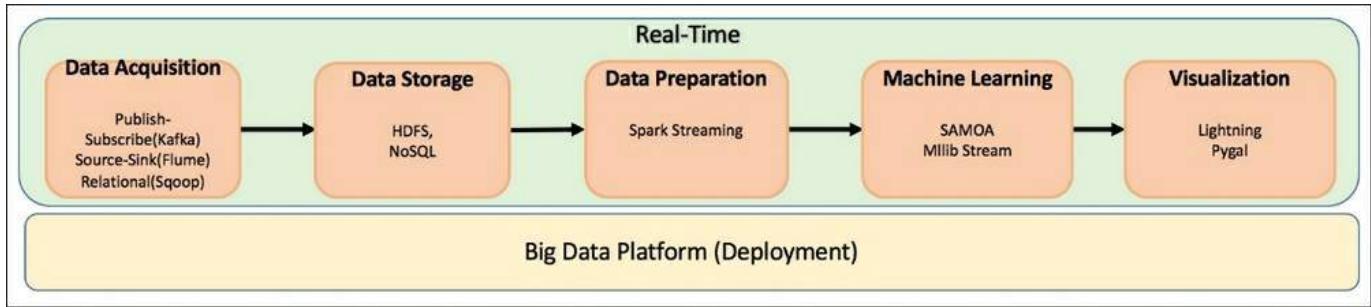


Figure 22: Real-time big data components and providers

SAMOA as a real-time Big Data Machine Learning framework

For a single machine, in [Chapter 5, Real-Time Stream Machine Learning](#), we discussed the MOA framework at length. SAMOA is the distributed framework for performing Machine Learning on streams.

At the time of writing, SAMOA is an incubator-level open source project with Apache 2.0 license and good integration with different stream processing engines such as **Apache Storm**, **Samza**, and **S4**.

SAMOA architecture

The SAMOA framework offers several key streaming services to an extendable set of stream processing engines, with existing implementations for the most popular engines of today.

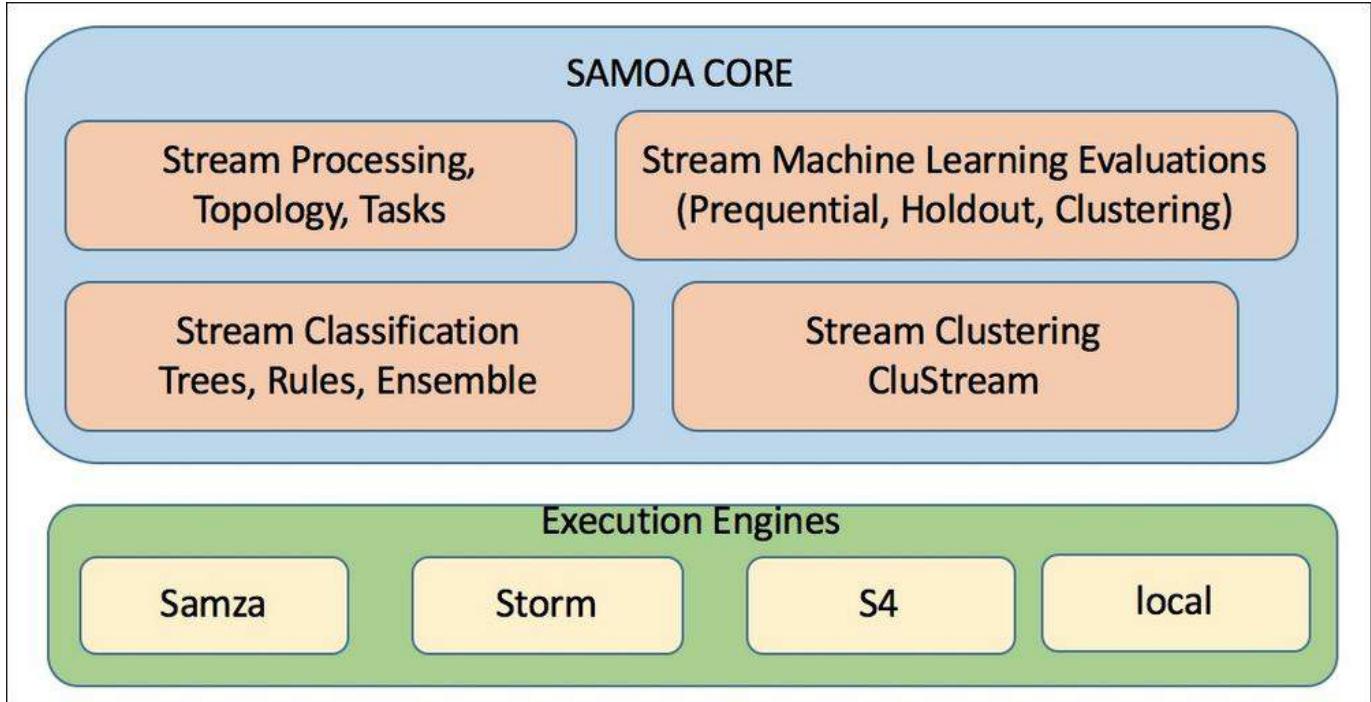


Figure 23: SAMOA high level architecture

`TopologyBuilder` is the interface that acts as a factory to create different components and connect them together in SAMOA. The core of SAMOA is in building processing elements for data streams. The basic unit for processing consists of `ProcessingItem` and the `Processor` interface, as shown in *Figure 24*. `ProcessingItem` is an encapsulated hidden element, while `Processor` is the core implementation where the logic for handling streams is coded.

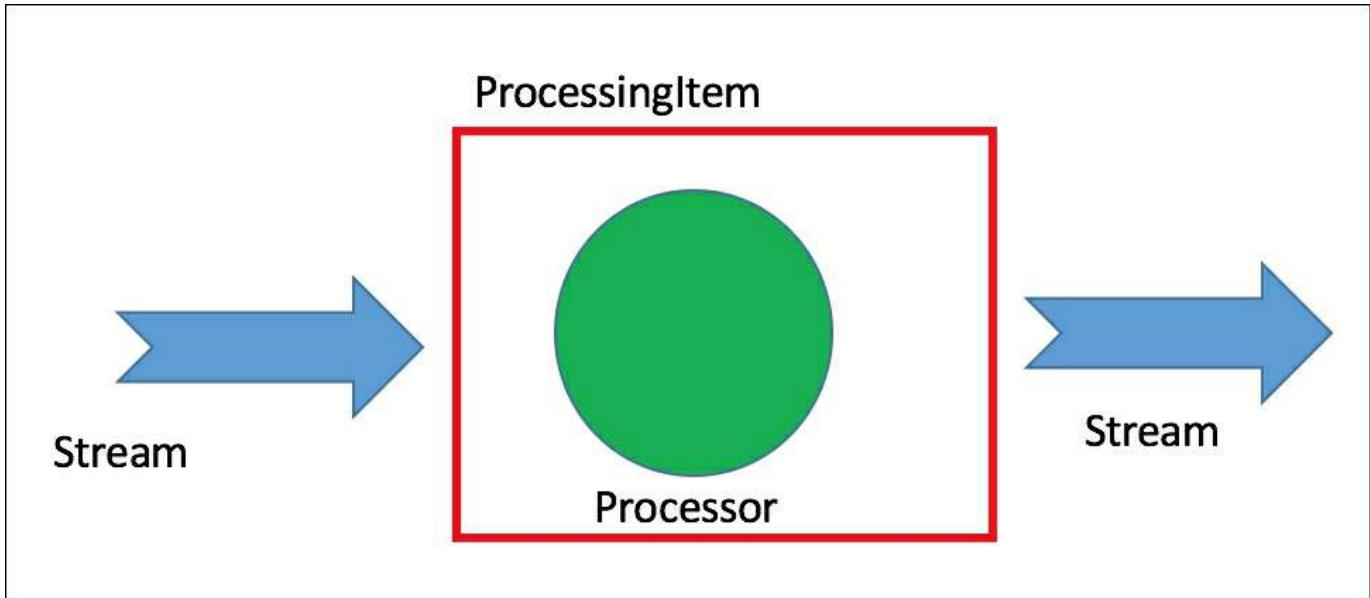


Figure 24: SAMOA processing data streams

Stream is another interface that connects various **Processors** together as the source and destination created by `TopologyBuilder`. A Stream can have one source and multiple destinations. Stream supports three forms of communication between source and destinations:

- **All:** In this communication, all messages from source are sent to all the destinations
- **Key:** In this communication, messages with the same keys are sent to the same processors
- **Shuffle:** In this communication, messages are randomly sent to the processors

All the messages or events in SAMOA are implementations of the interface `ContentEvent`, encapsulating mostly the data in the streams as a value and having some form of key for uniqueness.

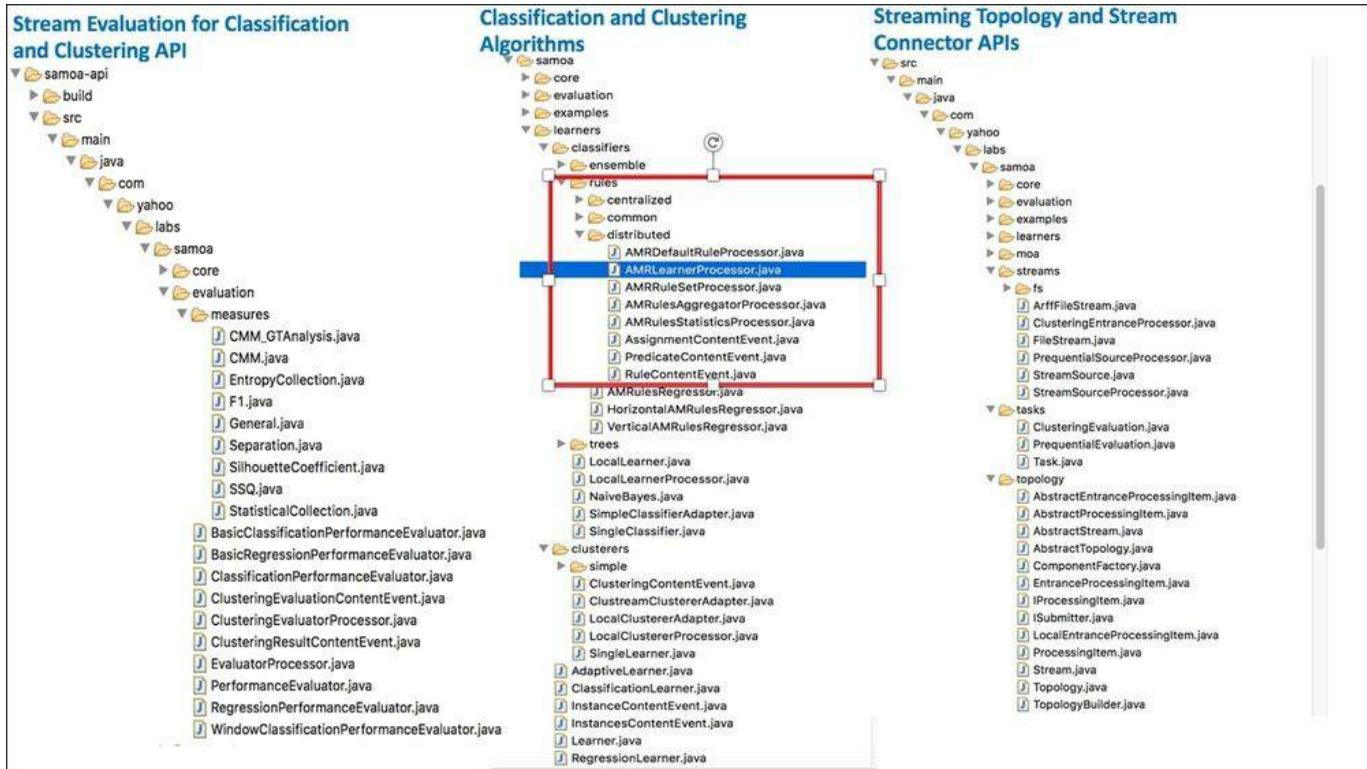
Each stream processing engine has an implementation for all the key interfaces as a plugin and integrates with SAMOA. The Apache Storm implementations `StormTopology`, `StormStream`, and `StormProcessingItem`, and so on are shown in the API in *Figure 25*.

Task is another unit of work in SAMOA, having the responsibility of execution. All

the classification or clustering evaluation and validation techniques such as prequential, holdout, and so on, are implemented as Tasks.

Learner is the interface for implementing all Supervised and Unsupervised Learning capability in SAMOA. Learners can be local or distributed and have different extensions such as `ClassificationLearner` and `RegressionLearner`.

Machine Learning algorithms



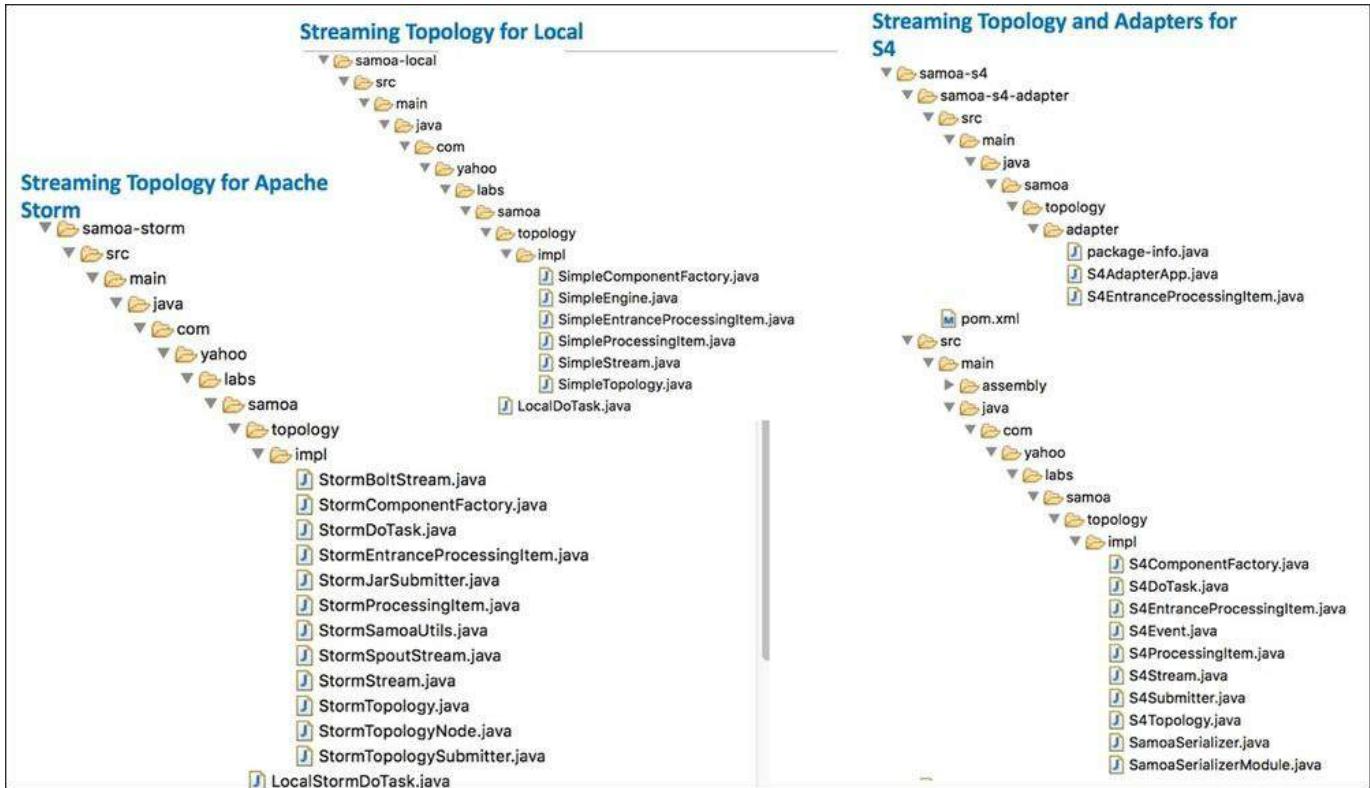


Figure 25: SAMOA machine learning algorithms

Figure 25 shows the core components of the SAMOA topology and their implementation for various engines.

Tools and usage

We continue with the same business problem as before. The command line to launch the training job for the covtype dataset is:

```
bin/samoa local target/SAMOA-Local-0.3.0-SNAPSHOT.jar
"PrequentialEvaluation -l classifiers.ensemble.Bagging
 -s (ArffFileStream -f covtype-train.csv.arff) -f 10000"
```

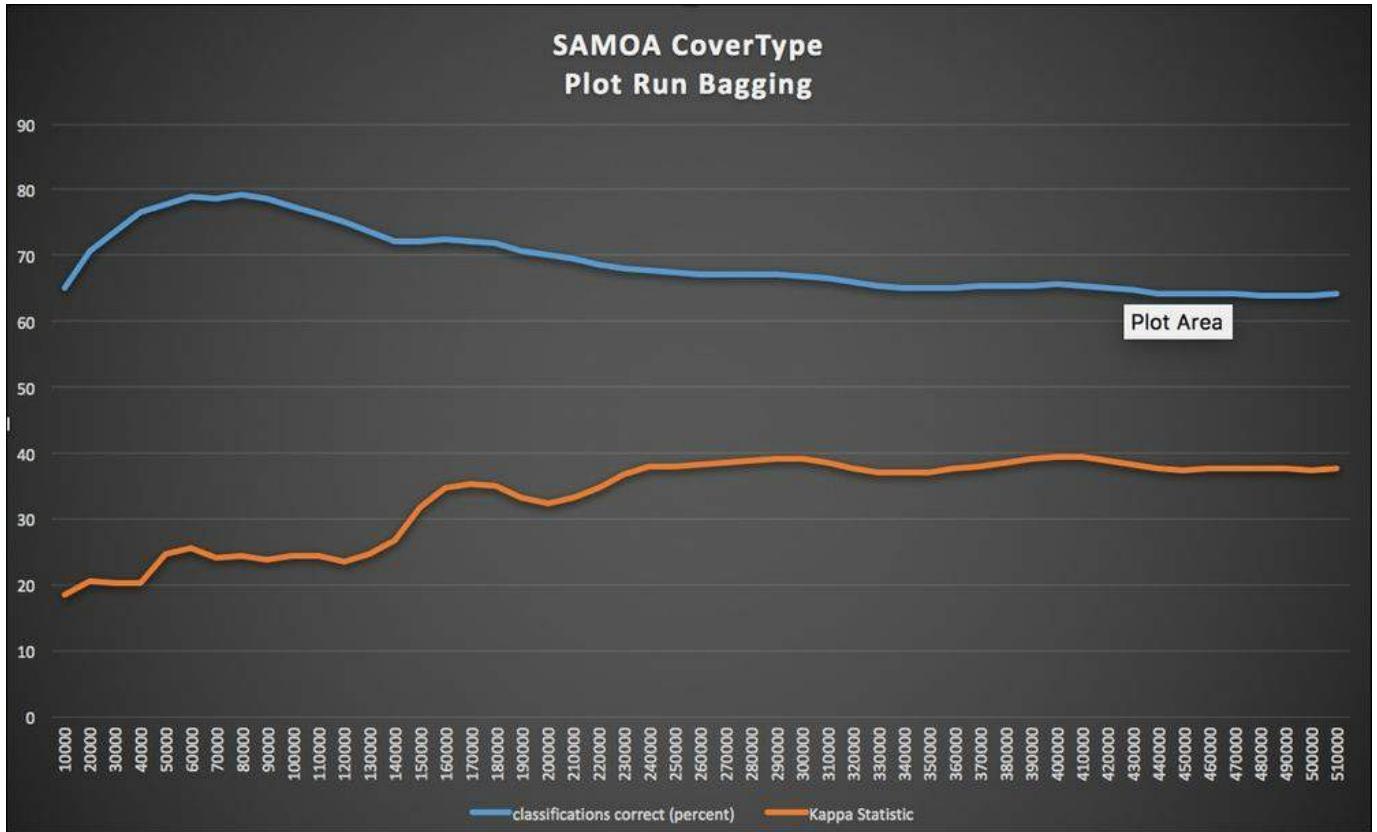


Figure 25: Bagging model performance

When running with Storm, this is the command line:

```
bin/samoa storm target/SAMOA-Storm-0.3.0-SNAPSHOT.jar
"PrequentialEvaluation -l classifiers.ensemble.Bagging
-s (ArffFileStream -f covtype-train.csv.arff) -f 10000"
```

Experiments, results, and analysis

The results of experiments using SAMOA as a stream-based learning platform for Big Data are given in *Table 5*.

Algorithm	Best Accuracy	Final Accuracy	Final Kappa Statistic	Final Kappa Temporal Statistic
Bagging	79.16	64.09	37.52	-69.51
Boosting	78.05	47.82	0	-1215.1
VerticalHoeffdingTree	83.23	67.51	44.35	-719.51

AdaptiveBagging	81.03	64.64	38.99	-67.37
-----------------	-------	-------	-------	--------

Table 5: Experimental results with Big Data real-time learning using SAMOA

Analysis of results

From an analysis of the results, the following observations can be made:

- *Table 5* shows that the popular non-linear decision tree-based VHDT on SAMOA is the best performing algorithm according to almost all the metrics.
- The adaptive bagging algorithm performs better than bagging because it employs Hoeffding Adaptive Trees in the implementation, which are more robust than basic online stream bagging.
- The online boosting algorithm with its dependency on the weak learners and no adaptability ranked the lowest as expected.
- The bagging plot in *Figure 25* shows a nice trend of stability achieved as the number of examples increased, validating the general consensus that if the patterns are stationary, more examples lead to robust models.

The future of Machine Learning

The impact of Machine Learning on businesses, social interactions, and indeed, our day-to-day lives today is undeniable, though not always immediately obvious. In the near future, it will be ubiquitous and inescapable. According to a report by McKinsey Global Institute published in December 2016 (*References [15]*), there is a vast unexploited potential for data and analytics in major industry sectors, especially healthcare and the public sector. Machine Learning is one of the key technologies poised to help exploit that potential. More compute power is at our disposal than ever before. More data is available than ever before, and we have cheaper and greater storage capacity than ever before.

Already, the unmet demand for data scientists has spurred changes to college curricula worldwide, and has caused an increase of 16% a year in wages for data scientists in the US, in the period 2012-2014. The solution to a wide swathe of problems is within reach with Machine Learning, including resource allocation, forecasting, predictive analytics, predictive maintenance, and price and product optimization.

The same McKinsey report emphasizes the increasing role of Machine Learning, including deep learning in a variety of use cases across industries such as agriculture,

pharma, manufacturing, energy, media, and finance. These scenarios run the gamut: predict personalized health outcomes, identify fraud transactions, optimize pricing and scheduling, personalize crops to individual conditions, identify and navigate roads, diagnose disease, and personalize advertising. Deep learning has great potential in automating an increasing number of occupations. Just improving natural language understanding would potentially cause a USD 3 trillion impact on global wages, affecting jobs like customer service and support worldwide.

Giant strides in image and voice recognition and language processing have made applications such as personal digital assistants commonplace, thanks to remarkable advances in deep learning techniques. The symbolism of AlphaGO's success in defeating Lee Sedol, alluded to in the opening chapter of this book, is enormous, as it is a vivid example of how progress in artificial intelligence is besting our own predictions of milestones in AI advancement. Yet this is the tip of the proverbial iceberg. Recent work in areas such as transfer learning offers the promise of more broadly intelligent systems that will be able to solve a wider range of problems rather than narrowly specializing in just one. General Artificial Intelligence, where AI can develop objective reasoning, proposes a methodology to solve a problem, and learn from its mistakes, is some distance away at this point, but check back in a few years and that distance may well have shrunk beyond our current expectations!

Increasingly, the confluence of transformative advances in technologies incrementally enabling each other spells a future of dizzying possibilities that we can already glimpse around us. The role of Machine Learning, it would appear, is to continue to shape that future in profound and extraordinary ways. Of that, there is little doubt.

Summary

The final chapter of this book deals with Machine Learning adapted to what is arguably one of the most significant paradigm shifts in information management and analytics to have emerged in the last few decades—Big Data. Much as many other areas of computer science and engineering have seen, AI—and Machine Learning in particular—has benefited from innovative solutions and dedicated communities adapting to face the many challenges posed by Big Data.

One way to characterize Big Data is by volume, velocity, variety, and veracity. This demands a new set of tools and frameworks to conduct the tasks of effective analytics at large.

Choosing a Big Data framework involves selecting distributed storage systems, data preparation techniques, batch or real-time Machine Learning, as well as visualization and reporting tools.

Several open source deployment frameworks are available including Hortonworks Data Platform, Cloudera CDH, Amazon Elastic MapReduce, and Microsoft Azure HDInsight. Each provides a platform with components supporting data acquisition, data preparation, Machine Learning, evaluation, and visualization of results.

Among the data acquisition components, publish-subscribe is a model offered by Apache Kafka (*References [8]*) and Amazon Kinesis, which involves a broker mediating between subscribers and publishers. Alternatives include source-sink, SQL, message queueing, and other custom frameworks.

With regard to data storage, several factors contribute to the proper choice for whatever your needs may be. HDFS offers a distributed File System with robust fault tolerance and high throughput. NoSQL databases also offer high throughput, but generally with weak guarantees on consistency. They include key-value, document, columnar, and graph databases.

Data processing and preparation come next in the flow, which includes data cleaning, scraping, and transformation. Hive and HQL provide these functions in HDFS systems. SparkSQL and Amazon Redshift offer similar capabilities. Real-time stream processing is available from products such as Storm and Samza.

The learning stage in Big Data analytics can include batch or real-time data.

A variety of rich visualization and analysis frameworks exist that are accessible from multiple programming environments.

Two major Machine Learning frameworks on Big Data are H2O and Apache Spark MLLib. Both can access data from various sources such as HDFS, SQL, NoSQL, S3, and others. H2O supports a number of Machine Learning algorithms that can be run in a cluster. For Machine Learning with real-time data, SAMOA is a big data framework with a comprehensive set of stream-processing capabilities.

The role of Machine Learning in the future is going to be a dominant one, with a wide-ranging impact on healthcare, finance, energy, and indeed on most industries. The expansion in the scope of automation will have inevitable societal effects.

Increases in compute power, data, and storage per dollar are opening up great new vistas to Machine Learning applications that have the potential to increase productivity, engender innovation, and dramatically improve living standards the world over.

References

1. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, *Ion Stoica:Spark: Cluster Computing with Working Sets*. HotCloud 2010
2. Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, *Ion Stoica:Apache Spark: a unified engine for Big Data processing*. Commun. ACM 59(11): 56-65 (2016)
3. Apache Hadoop: <https://hadoop.apache.org/>.
4. Cloudera: <http://www.cloudera.com/>.
5. Hortonworks: <http://hortonworks.com/>.
6. Amazon EC2: <http://aws.amazon.com/ec2/>.
7. Microsoft Azure: <http://azure.microsoft.com/>.
8. Apache Flume: <https://flume.apache.org/>.
9. Apache Kafka: <http://kafka.apache.org/>.
10. Apache Sqoop: <http://sqoop.apache.org/>.
11. Apache Hive: <http://hive.apache.org/>.
12. Apache Storm: <https://storm.apache.org/>.
13. H2O: <http://h2o.ai/>.
14. Shahrivari S, Jalili S. *Beyond batch processing: towards real-time and streaming Big Data*. Computers. 2014;3(4):117–29.
15. *MGI, The Age of Analytics*—Executive Summary
<http://www.mckinsey.com/~media/McKinsey/Business%20Functions/McKinsey-The-Age-of-Analytics-Full-report.ashx>.

Appendix A. Linear Algebra

Linear algebra is of primary importance in machine learning and it gives us an array of tools that are especially handy for the purpose of manipulating data and extracting patterns from it. Moreover, when data must be processed in batches as in much machine learning, great runtime efficiencies are gained from using the "vectorized" form as an alternative to traditional looping constructs when implementing software solutions in optimization or data pre-processing or any number of operations in analytics.

We will consider only the domain of real numbers in what follows. Thus, a vector $\vec{v} \in \mathbb{R}^n$ represents an array of n real-valued numbers. A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a two-dimensional array of m rows and n columns of real-valued numbers.

Some key concepts from the foundation of linear algebra are presented here.

Vector

The vector \mathbf{x} (lowercase, bold, by convention; equivalently, \vec{x}) can be thought of as a point in n -dimensional space. Conventionally, we mean column-vector when we say vector. The *transpose* of a column vector is a *row* vector with the same number of elements, arranged in a single row.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

$$\Rightarrow \mathbf{x}^T = [x \ x_2 \ \dots \ x_n]$$

Scalar product of vectors

Also known as the dot product, the scalar product is defined for two vectors of equal length. The result of the operation is a scalar value and is obtained by summing over the products of the corresponding elements of the vectors. Thus, given vectors \mathbf{x} and \mathbf{y} :

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

The dot product $\mathbf{x}^T \mathbf{y}$ is given as:

$$\mathbf{x}^T \mathbf{y} = [x_1 \ x_2] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = x_1 y_1 + x_2 y_2$$

Matrix

A matrix is a two-dimensional array of numbers. Each element can be indexed by its row and column position. Thus, a 3 x 2 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

Transpose of a matrix

Swapping columns for rows in a matrix produces the transpose. Thus, the transpose of \mathbf{A} is a 2×3 matrix:

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

Matrix addition

Matrix addition is defined as element-wise summation of two matrices with the same shape. Let \mathbf{A} and \mathbf{B} be two $m \times n$ matrices. Their sum \mathbf{C} can be written as follows:

$$C_{i,j} = A_{i,j} + B_{i,j}$$

Scalar multiplication

Multiplication with a scalar produces a matrix where each element is scaled by the scalar value. Here \mathbf{A} is multiplied by the scalar value d :

$$\mathbf{D}_{i,j} = d\mathbf{A}_{i,j}$$

Matrix multiplication

Two matrices \mathbf{A} and \mathbf{B} can be multiplied if the number of columns of \mathbf{A} equals the number of rows of \mathbf{B} . If \mathbf{A} has dimensions $m \times n$ and \mathbf{B} has dimensions $n \times p$, then the product \mathbf{AB} has dimensions $m \times p$:

$$\mathbf{C}_{i,j} = \mathbf{AB} = \sum_k \mathbf{A}_{ik} \mathbf{B}_{kj}$$

Properties of matrix product

Distributivity over addition: $A(B + C) = AB + AC$

Associativity: $A(BC) = (AB)C$

Non-commutativity: $AB \neq BA$

Vector dot-product is commutative: $\mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$

Transpose of product is product of transposes: $(AB)^T = A^T B^T$

Linear transformation

There is a special importance to the product of a matrix and a vector in linear algebra. Consider the product of a 3×2 matrix \mathbf{A} and a 2×1 vector \mathbf{x} producing a 3×1 vector \mathbf{y} :

$$\mathbf{y} = \mathbf{Ax} \Rightarrow \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} =$$

$$\begin{bmatrix} x_1 a_{11} + x_2 a_{12} \\ x_1 a_{21} + x_2 a_{22} \\ x_1 a_{31} + x_2 a_{32} \end{bmatrix}$$

$$= \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$= x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

(C)

$$= \begin{bmatrix} [a_{11}, a_{12}] \cdot \vec{x} \\ [a_{21}, a_{22}] \cdot \vec{x} \\ [a_{31}, a_{32}] \cdot \vec{x} \end{bmatrix}$$

(R)

It is useful to consider two views of the preceding matrix-vector product, namely, the column picture (C) and the row picture (R). In the column picture, the product can be seen as a linear combination of the column vectors of the matrix, whereas the row picture can be thought of as the dot products of the rows of the matrix with the vector \vec{x} .

Matrix inverse

The product of a matrix with its inverse is the Identity matrix. Thus:

$$\mathbf{A}^{-1} \mathbf{A} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} = \mathbf{I}$$

The matrix inverse, if it exists, can be used to solve a system of simultaneous equations represented by the preceding vector-matrix product equation. Consider a system of equations:

$$x_1 + 2x_2 = 3$$

$$3x_1 + 9x_2 = 21$$

This can be expressed as an equation involving the matrix-vector product:

$$\begin{bmatrix} 1 & 2 \\ 3 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 21 \end{bmatrix}$$

We can solve for the variables x_1 and x_2 by multiplying both sides by the matrix inverse:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 & -2/3 \\ -1 & 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ 21 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

The matrix inverse can be calculated by different methods. The reader is advised to view Prof. Strang's MIT lecture: bit.ly/10vmKcL.

Eigendecomposition

Matrices can be decomposed to factors that can give us valuable insight into the transformation that the matrix represents. Eigenvalues and eigenvectors are obtained as the result of eigendecomposition. For a given square matrix \mathbf{A} , an eigenvector is a non-zero vector that is transformed into a scaled version of itself when multiplied by the matrix. The scalar multiplier is the eigenvalue. All scalar multiples of an eigenvector are also eigenvectors:

$$\mathbf{A} \mathbf{v} = \lambda \mathbf{v}$$

In the preceding example, \mathbf{v} is an eigenvector and λ is the eigenvalue.

The eigenvalue equation of matrix \mathbf{A} is given by:

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = 0$$

The non-zero solution for the eigenvalues is given by the roots of the characteristic polynomial equation of degree n represented by the determinant:

$$|\mathbf{A} - \lambda \mathbf{I}| = (\lambda_1 - \lambda)(\lambda_2 - \lambda)(\lambda_3 - \lambda) \dots (\lambda_n - \lambda)$$

The eigenvectors can then be found by solving for \mathbf{v} in $\mathbf{Av} = \lambda \mathbf{v}$.

Some matrices, called diagonalizable matrices, can be built entirely from their eigenvectors and eigenvalues. If Λ is the diagonal matrix with the eigenvalues of matrix \mathbf{A} on its principal diagonal, and \mathbf{Q} is the matrix whose columns are the eigenvectors of \mathbf{A} :

$$\Lambda = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{v}^{(1)} & \mathbf{v}^{(2)} & \mathbf{v}^{(3)} & \dots & \mathbf{v}^{(n)} \end{bmatrix}$$

Then $\mathbf{A} = \mathbf{Q} \Lambda \mathbf{Q}^{-1}$.

Positive definite matrix

If a matrix has only positive eigenvalues, it is called a **positive definite matrix**. If the eigenvalues are positive or zero, the matrix is called a **positive semi-definite matrix**. With positive definite matrices, it is true that:

$$\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$$

Singular value decomposition (SVD)

SVD is a decomposition of any rectangular matrix \mathbf{A} of dimensions $n \times p$ and is written as the product of three matrices:

$$\mathbf{A}_{n \times p} = \mathbf{U}_{n \times n} \mathbf{S}_{n \times p} \mathbf{V}^T_{p \times p}$$

\mathbf{U} is defined to be $n \times n$, \mathbf{S} is a diagonal $n \times p$ matrix, and \mathbf{V} is $p \times p$. \mathbf{U} and \mathbf{V} are orthogonal matrices; that is:

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}_{n \times n} \text{ and } \mathbf{V}^T \mathbf{V} = \mathbf{I}_{p \times p}$$

The diagonal values of \mathbf{S} are called the singular values of \mathbf{A} . Columns of \mathbf{U} are called left singular vectors of \mathbf{A} and those of \mathbf{V} are called right singular vectors of \mathbf{A} . The left singular vectors are orthonormal eigenvectors of $\mathbf{A}^T \mathbf{A}$ and the right singular vectors are orthonormal eigenvectors of $\mathbf{A} \mathbf{A}^T$.

The SVD representation expands the original data into a coordinate system such that the covariance matrix is a diagonal matrix.

Appendix B. Probability

Essential concepts in probability are presented here in brief.

Axioms of probability

Kolmogorov's axioms of probability can be stated in terms of the sample space S of possible events, $E_1, E_2, E_3, \dots, E_n$ and the real-valued probability $P(E)$ of an event E . The axioms are:

1. $P(E) \geq 0$ for all $E \in S$
2. $P(S) = 1$
3. if $A \cap B = \emptyset$ then $P(A \cup B) = P(A) + P(B)$

Together, these axioms say that probabilities cannot be negative numbers—impossible events have zero probability—no events outside the sample space are possible as it is the universe of possibilities under consideration, and that the probability of either of two mutually exclusive events occurring is equal to the sum of their individual probabilities.

Bayes' theorem

The probability of an event E conditioned on evidence X is proportional to the prior probability of the event and the likelihood of the evidence given that the event has occurred. This is Bayes' Theorem:

$$P(E|X) = \frac{P(X|E)P(E)}{P(X)}$$

$P(X)$ is the normalizing constant, which is also called the marginal probability of X . $P(E)$ is the prior, and $P(X|E)$ is the likelihood. $P(E|X)$ is also called the posterior probability.

Bayes' Theorem expressed in terms of the posterior and prior odds is known as Bayes' Rule.

Density estimation

Estimating the hidden probability density function of a random variable from sample data randomly drawn from the population is known as density estimation. Gaussian mixtures and kernel density estimates are examples used in feature engineering, data modeling, and clustering.

Given a probability density function $f(X)$ for a random variable X , the probabilities associated with the values of X can be found as follows:

$$P(a < X < b) = \int_b^a f(x) dx \text{ for all } a < b$$

Density estimation can be parametric, where it is assumed that the data is drawn from a known family of distributions and $f(x)$ is estimated by estimating the parameters of the distribution, for example, μ and σ^2 in the case of a normal distribution. The other approach is non-parametric, where no assumption is made about the distribution of the observed data and the data is allowed to determine the form of the distribution.

Mean

The long-run average value of a random variable is known as the expectation or mean. The sample mean is the corresponding average over the observed data.

In the case of a discrete random variable, the mean is given by:

$$\mu = \sum xP(x)$$

For example, the mean number of pips turning up on rolling a single fair die is 3.5.

For a continuous random variable with probability density function $f(x)$, the mean is:

$$\mu = \int_{-\infty}^{\infty} xf(x)dx$$

Variance

Variance is the expectation of the square of the difference between the random variable and its mean.

In the discrete case, with the mean defined as previously discussed, and with the probability mass function $p(x)$, the variance is:

$$Var(X) = \sigma^2 = \sum_{i=1}^n p_i \cdot (x_i - \mu)^2 = \sum_{i=1}^n p_i x_i^2 - \mu^2$$

In the continuous case, it is as follows:

$$Var(X) = \sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 f(x) dx = \int_{-\infty}^{\infty} x^2 f(x) dx - \mu^2$$

Some continuous distributions do not have a mean or variance.

Standard deviation

Standard deviation is a measure of how spread out the data is in relation to its mean value. It is the square root of variance, and unlike variance, it is expressed in the same units as the data. The standard deviation in the case of discrete and continuous random variables are given here:,

- Discrete case:

$$\sigma = \sqrt{\sum_{i=1}^n p_i \cdot (x_i - \mu)^2}$$

- Continuous case:

$$\sigma = \sqrt{\int (x - \mu)^2 f(x) dx}$$

Gaussian standard deviation

The standard deviation of a sample drawn randomly from a larger population is a biased estimate of the population standard deviation. Based on the particular distribution, the correction to this biased estimate can differ. For a Gaussian or

normal distribution, the variance is adjusted by a value of $\frac{n}{n-1}$.

Per the definition given earlier, the biased estimate s is given by:

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

In the preceding equation, \bar{x} is the sample mean.

The unbiased estimate, which uses Bessel's correction, is:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Covariance

In a joint distribution of two random variables, the expectation of the product of the deviations of the random variables from their respective means is called the covariance. Thus, for two random variables X and Y , the equation is as follows:

$$\text{cov}(X, Y) = E[(X - \mu_x)(Y - \mu_y)]$$

$$= E[XY] - \mu_x \mu_y$$

If the two random variables are independent, then their covariance is zero.

Correlation coefficient

When the covariance is normalized by the product of the standard deviations of the two random variables, we get the correlation coefficient $\rho_{X,Y}$, also known as the Pearson product-moment correlation coefficient:

$$\rho_{X,Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

The correlation coefficient can take values between -1 and 1 only. A coefficient of +1 means a perfect increasing linear relationship between the random variables. -1 means a perfect decreasing linear relationship. If the two variables are independent of each other, the Pearson's coefficient is 0.

Binomial distribution

Discrete probability distribution with parameters **n** and **p**. A random variable is a binary variable, with the probability of outcome given by **p** and **1 – p** in a single trial. The probability mass function gives the probability of **k** successes out of **n** independent trials.

Parameters: n, k

PMF:

$$f(k | n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

This is the Binomial coefficient.

Mean: $E[X] = np$

Variance: $Var(X) = np(1 - p)$

Poisson distribution

The Poisson distribution gives the probability of the number of occurrences of an event in a given time period or in a given region of space.

Parameter λ , is the average number of occurrences in a given interval. The probability mass function of observing k events in that interval is

PMF:

$$P(k | \lambda) = e^{-\lambda} \frac{\lambda^k}{k!}$$

Mean: $E[X] = \lambda$

Variance: $Var(X) = \lambda$

Gaussian distribution

The Gaussian distribution, also known as the normal distribution, is a continuous probability distribution. Its probability density function is expressed in terms of the mean and variance as follows:

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Mean: μ

Standard deviation: σ

Variance: σ^2

The standard normal distribution is the case when the mean is 0 and the standard deviation is 1. The PDF of the standard normal distribution is given as follows:

$$\varphi(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$$

Central limit theorem

The central limit theorem says that when you have several independent and identically distributed random variables with a distribution that has a well-defined mean and variance, the average value (or sum) over a large number of these observations is approximately normally distributed, irrespective of the parent distribution. Furthermore, the limiting normal distribution has the same mean as the parent distribution and a variance equal to the underlying variance divided by the sample size.

Given a random sample $X_1, X_2, X_3 \dots X_n$ with $\mu = E[X_i]$ and $\sigma^2 = Var(X_i)$, the

sample mean: $\bar{X} = \sum_{i=1}^n X_i$

$$N\left(n\mu, \frac{\sigma^2}{n}\right).$$

is approximately normal

There are several variants of the central limit theorem where independence or the constraint of being identically distributed are relaxed, yet convergence to the normal distribution still follows.

Error propagation

Suppose there is a random variable X , which is a function of multiple observations each with their own distributions. What can be said about the mean and variance of X given the corresponding values for measured quantities that make up X ? This is the problem of error propagation.

Say x is the quantity to be determined via observations of variables u, v , and so on:

$$x = f(u, v, \dots)$$

Let us assume that:

$$\bar{x} = f(\bar{u}, \bar{v}, \dots)$$

The uncertainty in x in terms of the variances of u, v , and so on, can be expressed by the variance of x :

$$\sigma_x^2 = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_i (x_i - \bar{x})^2$$

From the Taylor expansion of the variance of x , we get the following:

$$\sigma_x^2 = \sigma_u^2 \left(\frac{\partial f}{\partial u} \right)_{\bar{u}}^2 + \sigma_v^2 \left(\frac{\partial f}{\partial v} \right)_{\bar{v}}^2 + 2\sigma_{uv}^2 \frac{\partial f}{\partial u} \Big|_{\bar{u}} \frac{\partial f}{\partial v} \Big|_{\bar{v}} + \dots$$

Here, σ_{uv}^2 is the covariance.

Similarly, we can determine the propagation error of the mean. Given N measurements with x_i with uncertainties characterized by s_i , the following can be written:

$$\bar{x} = \frac{1}{N} (x_1 + x_2 + x_3 + \dots + x_n) = \frac{1}{N} \sum x_i$$

With:

$$s_i^2 = \sum_i s_i^2 \left(\frac{\partial \bar{x}}{\partial x_i} \right)_{\bar{x}}^2,$$

These equations assume that the covariance is 0.

Suppose $s_i = s$ – that is, all observations have the same error.

$$s_{\bar{x}}^2 = \sum_i s_i^2 \left(\frac{\partial \bar{x}}{\partial x_i} \right)_{\bar{x}}^2$$

Then,

$$\frac{\partial \bar{x}}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\frac{1}{N} \sum_j x_j \right) = \frac{1}{N} \left(\frac{\partial x_j}{\partial x_i} \delta_{ij} \right)$$

$$s_{\bar{x}}^2 = \sum_i s^2 \left(\frac{1}{N} \right)^2$$

$$= \frac{s^2}{N}$$

Therefore,

Appendix D. Bibliography

This course is a blend of text and quizzes, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Java for Data Science, Richard M. Reese, Jennifer L. Reese*
- *Machine Learning in Java, Boštjan Kaluža*
- *Mastering Java Machine Learning, Dr. Uday Kamath, Krishna Choppella*

Index

A

- A/B tests
 - URL / [Importance of evaluation](#)
- Abstract-C / [Abstract-C](#)
- activation function
 - about / [Perceptron](#)
- active learning
 - about / [Active learning](#)
 - representation / [Representation and notation](#)
 - notation / [Representation and notation](#)
 - scenarios / [Active learning scenarios](#)
 - approaches / [Active learning approaches](#)
 - uncertainty sampling / [Uncertainty sampling](#)
 - version space sampling / [Version space sampling](#)
- active learning, case study
 - about / [Case study in active learning](#)
 - tools / [Tools and software](#)
 - software / [Tools and software](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine learning mapping](#)
 - data collection / [Data Collection](#)
 - data sampling / [Data sampling and transformation](#)
 - data transformation / [Data sampling and transformation](#)
 - feature analysis / [Feature analysis and dimensionality reduction](#)
 - dimensionality reduction / [Feature analysis and dimensionality reduction](#)
 - models / [Models, results, and evaluation](#)
 - results / [Models, results, and evaluation](#)
 - evaluation / [Models, results, and evaluation](#)
 - results, pool-based scenarios / [Pool-based scenarios](#)
 - results, stream-based scenarios / [Stream-based scenarios](#)
 - results, analysis / [Analysis of active learning results](#)
- activity recognition
 - about / [Introducing activity recognition](#)
 - mobile phone sensors / [Mobile phone sensors](#)
 - activity-recognition pipeline / [Activity recognition pipeline](#)

- plan / [The plan](#)
- AdaBoost M1 method
 - about / [Choosing a classification algorithm](#)
- ADaptable sliding WINdow (ADWIN)
 - about / [Sliding windows](#)
- adaptation methods
 - about / [Adaptation methods](#)
 - explicit adaptation / [Explicit adaptation](#)
 - implicit adaptation / [Implicit adaptation](#)
- Advanced Message Queueing Protocol (AMQP) / [Message queueing frameworks](#)
- advanced modelling
 - with ensembles / [Advanced modeling with ensembles](#)
 - ensembleLibrary package, using / [Before we start](#)
 - data, pre-processing / [Data pre-processing](#)
 - attribute selection / [Attribute selection](#)
 - model selection / [Model selection](#)
 - performance, evaluation / [Performance evaluation](#)
- affinity analysis
 - about / [Affinity analysis](#)
 - cross-industry applications / [Other applications in various areas](#)
- affinity propagation
 - about / [Affinity propagation](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- agglomerative clustering
 - about / [Clustering](#)
- algorithms, comparing
 - McNemars Test / [McNemar's Test](#)
 - Wilcoxon signed-rank test / [Wilcoxon signed-rank test](#)
- Amazon Elastic MapReduce (EMR) / [Amazon Elastic MapReduce](#)
- Amazon Kinesis / [Publish-subscribe frameworks](#)
- Amazon Machine Learning / [Machine learning as a service](#)
- Amazon Redshift / [Amazon Redshift](#)
- analysis types

- about / [Analysis types](#)
 - pattern analysis / [Pattern analysis](#)
 - transaction analysis / [Transaction analysis](#)
- Android Device Monitor
 - about / [Collecting training data](#)
- Android Studio
 - installing / [Installing Android Studio](#)
 - URL / [Installing Android Studio](#)
- Angle-based Outlier Degree (ABOD) / [How does it work?](#)
- anomalous behaviour detection
 - about / [Suspicious and anomalous behavior detection](#)
 - unknown-unknowns / [Unknown-unknowns](#)
- anomalous pattern detection
 - about / [Anomalous pattern detection](#)
 - analysis types / [Analysis types](#)
 - plan recognition / [Plan recognition](#)
- anomaly detection
 - about / [Outlier or anomaly detection](#)
- anomaly detection, in time series data
 - about / [Anomaly detection in time series data](#)
 - histogram-based anomaly detection / [Histogram-based anomaly detection](#)
 - data, loading / [Loading the data](#)
 - histograms, creating / [Creating histograms](#)
 - density based k-nearest neighbours / [Density based k-nearest neighbors](#)
- anomaly detection, in website traffic
 - about / [Anomaly detection in website traffic](#)
 - dataset, using / [Dataset](#)
- ANOVA test / [ANOVA test](#)
- Apache Kafka / [Publish-subscribe frameworks](#)
- Apache Mahout
 - about / [Apache Mahout](#)
 - configuring / [Getting Apache Mahout](#)
 - configuring, in Eclipse with Maven plugin / [Configuring Mahout in Eclipse with the Maven plugin](#)
- Apache Spark
 - about / [Apache Spark](#)
 - URL / [Apache Spark](#)
- Apache Storm / [SAMOA as a real-time Big Data Machine Learning framework](#)

- Application Portfolio Management (APM)
 - about / [IT Operations Analytics](#)
- Applied Machine Learning
 - workflow / [Applied machine learning workflow](#)
- Approx Storm / [Approx Storm](#)
- Apriori
 - about / [Weka](#)
- Apriori algorithm
 - about / [Apriori algorithm](#)
 - used, for discovering shopping patterns / [Apriori](#)
- ArangoDB / [Graph databases](#)
- artificial neural networks
 - about / [Artificial neural networks](#)
- association analysis / [Machine learning – types and subtypes](#)
- association rule learning
 - about / [Association rule learning](#)
 - Apriori algorithm / [Apriori algorithm](#)
 - FP-growth algorithm / [FP-growth algorithm](#)
- association rule learning, basic concepts
 - database, of transactions / [Database of transactions](#)
 - itemset / [Itemset and rule](#)
 - rule / [Itemset and rule](#)
 - support / [Support](#)
 - confidence / [Confidence](#)
- autoencoder
 - about / [Autoencoder](#)
- Autoencoders
 - about / [Autoencoders](#)
 - mathematical notations / [Definition and mathematical notations](#)
 - loss function / [Loss function](#)
 - limitations / [Limitations of Autoencoders](#)
 - denoising / [Denoising Autoencoder](#)
- axioms of probability / [Axioms of probability](#)

B

- bag-of-word (BoW)
 - about / [Working with text data](#)
- Balanced Iterative Reducing and Clustering Hierarch (BIRCH)
 - about / [Hierarchical based and micro clustering](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- basic modelling
 - about / [Basic modeling](#)
 - models, evaluating / [Evaluating models](#)
 - naive Bayes baseline, implementing / [Implementing naive Bayes baseline](#)
- basic naive Bayes classifier baseline
 - about / [Basic naive Bayes classifier baseline](#)
 - data, obtaining / [Getting the data](#)
 - data, loading / [Loading the data](#)
- Batch Big Data Machine Learning
 - about / [Batch Big Data Machine Learning](#)
 - used, as H2O / [H2O as Big Data Machine Learning platform](#)
- Bayesian information score (BIC) / [Measures to evaluate structures](#)
- Bayesian networks
 - about / [Bayesian networks](#)
 - representation / [Bayesian networks, Representation](#)
 - inference / [Bayesian networks, Inference](#)
 - learning / [Bayesian networks, Learning](#)
- Bayes theorem
 - about / [Bayes' theorem](#)
 - density, estimation / [Density estimation](#)
 - mean / [Mean](#)
 - variance / [Variance](#)
 - standard deviation / [Standard deviation](#)
 - Gaussian standard deviation / [Gaussian standard deviation](#)
 - covariance / [Covariance](#)
 - correlation coefficient / [Correlation coefficient](#)
 - binomial distribution / [Binomial distribution](#)

- Poisson distribution / [Poisson distribution](#)
- Gaussian distribution / [Gaussian distribution](#)
- central limit theorem / [Central limit theorem](#)
- error propagation / [Error propagation](#)
- BBC dataset
 - URL / [BBC dataset](#)
- Bernoulli distribution / [Random variables, joint, and marginal distributions](#)
- big data
 - dealing with / [Dealing with big data](#)
 - volume / [Dealing with big data](#)
 - velocity / [Dealing with big data](#)
 - variety / [Dealing with big data](#)
- Big Data
 - characteristics / [What are the characteristics of Big Data?](#)
 - volume / [What are the characteristics of Big Data?](#)
 - velocity / [What are the characteristics of Big Data?](#)
 - variety / [What are the characteristics of Big Data?](#)
 - veracity / [What are the characteristics of Big Data?](#)
- big data application
 - architecture / [Big data application architecture](#)
- Big Data cluster deployment frameworks
 - about / [Big Data cluster deployment frameworks](#)
 - Hortonworks Data Platform (HDP) / [Hortonworks Data Platform](#)
 - Cloudera CDH / [Cloudera CDH](#)
 - Amazon Elastic MapReduce (EMR) / [Amazon Elastic MapReduce](#)
 - Microsoft Azure HDInsight / [Microsoft Azure HDInsight](#)
- Big Data framework
 - about / [General Big Data framework](#)
 - cluster deployment frameworks / [Big Data cluster deployment frameworks](#)
 - data acquisition / [Data acquisition](#)
 - data storage / [Data storage](#)
 - data preparation / [Data processing and preparation](#)
 - data processing / [Data processing and preparation](#)
 - machine learning / [Machine Learning](#)
 - visualization / [Visualization and analysis](#)
 - analysis / [Visualization and analysis](#)
- Big Data Machine Learning
 - about / [Big Data Machine Learning](#)

- framework / [General Big Data framework](#)
 - Big Data framework / [General Big Data framework](#)
 - Spark MLlib / [Spark MLlib as Big Data Machine Learning platform](#)
- BigML / [Machine learning as a service](#)
- binomial distribution / [Binomial distribution](#)
- Book-Crossing dataset
 - URL / [Book ratings dataset](#)
 - BX-Users file / [Book ratings dataset](#)
 - BX-Books file / [Book ratings dataset](#)
 - BX-Book-Ratings file / [Book ratings dataset](#)
- book-recommendation engine
 - building / [Building a recommendation engine](#)
 - book ratings dataset, using / [Book ratings dataset](#)
 - data, loading / [Loading the data](#)
 - data, loading from file / [Loading data from file](#)
 - data, loading from database / [Loading data from database](#)
 - in-memory database, creating / [In-memory database](#)
 - collaborative filtering, implementing / [Collaborative filtering](#)
 - custom rules, adding / [Adding custom rules to recommendations](#)
 - evaluation / [Evaluation](#)
 - online learning engine / [Online learning engine](#)
 - content-based filtering, implementing / [Content-based filtering](#)
- boosting
 - about / [Boosting](#)
 - algorithm input / [Algorithm inputs and outputs](#)
 - algorithm output / [Algorithm inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- bootstrap aggregating (bagging)
 - about / [Bootstrap aggregating or bagging](#)
 - algorithm inputs / [Algorithm inputs and outputs](#)
 - algorithm outputs / [Algorithm inputs and outputs](#)
 - working / [How does it work?](#)
 - Random Forest / [Random Forest](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Broyden-Fletcher-Goldfarb-Shanno (BFGS) / [How does it work?](#)

- Business Intelligence (BI) / [What is not machine learning?](#)
- business problem / [Business problem](#), [Business problem](#)

C

- Canova library
 - URL / [Loading the data](#)
- case study
 - about / [Case study](#)
 - business problem / [Business problem](#)
 - machine learning mapping / [Machine learning mapping](#)
 - data sampling and transformation / [Data sampling and transformation](#)
 - feature analysis / [Feature analysis](#)
 - Models, results, and evaluation / [Models, results, and evaluation](#)
 - results, analysis / [Analysis of results](#)
- case study, with CoverType dataset
 - about / [Case study](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine Learning mapping](#)
 - data collection / [Data collection](#)
 - data sampling / [Data sampling and transformation](#)
 - data transformation / [Data sampling and transformation](#)
 - Big Data Machine Learning, used as Spark MLlib / [Spark MLlib as Big Data Machine Learning platform](#)
- Cassandra
 - about / [Big data application architecture](#)
 - URL / [Big data application architecture](#)
 - / [Columnar databases](#)
- cc.mallet.pipe package
 - Input2CharSequence pipeline / [Pre-processing text data](#)
 - CharSequenceRemoveHTML pipeline / [Pre-processing text data](#)
 - MakeAmpersandXMLFriendly pipeline / [Pre-processing text data](#)
 - TokenSequenceLowercase pipeline / [Pre-processing text data](#)
 - TokenSequence2FeatureSequence pipeline / [Pre-processing text data](#)
 - TokenSequenceNGrams pipeline / [Pre-processing text data](#)
- central limit theorem / [Central limit theorem](#)
- Chebyshev distance
 - about / [Java machine learning](#)
- Chi-Squared feature / [Statistical approach](#)
- Chunk / [H2O architecture](#)
- classification

- about / [Classification](#), [Classification](#), [Formal description and notation](#)
 - decision trees learning / [Decision tree learning](#)
 - probabilistic classifiers / [Probabilistic classifiers](#)
 - kernel methods / [Kernel methods](#)
 - artificial neural networks / [Artificial neural networks](#)
 - ensemble learning / [Ensemble learning](#)
 - evaluating / [Evaluating classification](#)
 - precision / [Precision and recall](#)
 - recall / [Precision and recall](#)
 - Roc curves / [Roc curves](#)
 - data, using / [Data](#)
 - data, loading / [Loading data](#)
 - feature selection / [Feature selection](#)
 - learning algorithms, selecting / [Learning algorithms](#)
 - data, classifying / [Classify new data](#)
 - evaluation / [Evaluation and prediction error metrics](#)
 - prediction error metrics / [Evaluation and prediction error metrics](#)
 - confusion matrix, examining / [Confusion matrix](#)
 - algorithm, selecting / [Choosing a classification algorithm](#)
- classification algorithms
 - weka.classifiers.rules.ZeroR / [Choosing a classification algorithm](#)
 - weka.classifiers.trees.RandomTree / [Choosing a classification algorithm](#)
 - weka.classifiers.trees.RandomForest / [Choosing a classification algorithm](#)
 - weka.classifiers.lazy.IBk / [Choosing a classification algorithm](#)
 - weka.classifiers.functions.MultilayerPerceptron / [Choosing a classification algorithm](#)
 - weka.classifiers.bayes.NaiveBayes / [Choosing a classification algorithm](#)
 - weka.classifiers.meta.AdaBoostM1 / [Choosing a classification algorithm](#)
 - weka.classifiers.meta.Bagging / [Choosing a classification algorithm](#)
- Classification and Regression Trees (CART) / [Decision Trees](#)
- classifier
 - building / [Building a classifier](#)
 - spurious transitions, reducing / [Reducing spurious transitions](#)
 - plugging, into mobile app / [Plugging the classifier into a mobile app](#)
- class implementation
 - reference link / [Loading the data](#)
- class unbalance / [Class unbalance](#)
- Clique tree or junction tree algorithm, Bayesian networks

- about / [Clique tree or junction tree algorithm](#)
 - input and output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages and limitations / [Advantages and limitations](#)
- Cloudera CDH / [Cloudera CDH](#)
- Cluster-based Local Outlier Factor (CBLOF)
 - about / [How does it work?](#)
- clustering
 - about / [Clustering, Clustering, Clustering](#)
 - algorithms / [Clustering algorithms](#)
 - evaluation / [Evaluation](#)
 - spectral clustering / [Spectral clustering](#)
 - affinity propagation / [Affinity propagation](#)
 - using, for incremental unsupervised learning / [Incremental unsupervised learning using clustering](#)
 - evaluation techniques / [Validation and evaluation techniques](#)
 - validation techniques / [Validation and evaluation techniques](#)
 - stream cluster evaluation, key issues / [Key issues in stream cluster evaluation](#)
 - evaluation measures / [Evaluation measures](#)
- clustering-based methods
 - about / [Clustering-based methods](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- clustering algorithms
 - implementing / [Clustering algorithms](#)
 - about / [Clustering algorithms](#)
 - k-means / [k-Means](#)
 - DBSCAN / [DBSCAN](#)
 - mean shift / [Mean shift](#)
 - Gaussian mixture modeling (GMM) / [Expectation maximization \(EM\) or Gaussian mixture modeling \(GMM\)](#)
 - expectation maximization (EM) / [Expectation maximization \(EM\) or Gaussian mixture modeling \(GMM\)](#)
 - hierarchical clustering / [Hierarchical clustering](#)

- self-organizing maps (SOM) / [Self-organizing maps \(SOM\)](#)
- clustering evaluation
 - about / [Clustering validation and evaluation](#)
 - internal evaluation measures / [Clustering validation and evaluation](#), [Internal evaluation measures](#)
 - external evaluation measures / [Clustering validation and evaluation](#), [External evaluation measures](#)
- Clustering Features (CF) / [Hierarchical based and micro clustering](#)
- Clustering Feature Tree (CF Tree) / [Hierarchical based and micro clustering](#)
- clustering techniques
 - about / [Clustering techniques](#)
 - generative probabilistic models / [Generative probabilistic models](#)
 - distance-based text clustering / [Distance-based text clustering](#)
 - non-negative Matrix factorization (NMF) / [Non-negative matrix factorization \(NMF\)](#)
- Clustering Trees (CT) / [Hierarchical based and micro clustering](#)
- clustering validation
 - about / [Clustering validation and evaluation](#)
- Cluster Mapping Measures (CMM)
 - about / [Cluster Mapping Measures \(CMM\)](#)
 - mapping phase / [Cluster Mapping Measures \(CMM\)](#)
 - penalty phase / [Cluster Mapping Measures \(CMM\)](#)
- cluster mode / [Amazon Elastic MapReduce](#)
- cluster SSL
 - about / [Cluster and label SSL](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - limitations / [Advantages and limitations](#)
 - advantages / [Advantages and limitations](#)
- CluStream
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- co-training SSL
 - about / [Co-training SSL or multi-view SSL](#)

- inputs / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- collaborative filtering
 - about / [Collaborative filtering](#)
 - implementing, with book-recommendation engine / [Collaborative filtering](#)
 - user-based / [User-based filtering](#)
 - item-based / [Item-based filtering](#)
- columnar databases / [Columnar databases](#)
- Comma Separated Value (CSV)
 - about / [Loading the data](#)
- competitions
 - about / [Competitions](#)
- concept drift
 - about / [Concept drift and drift detection](#)
- conditional probability distribution (CPD) / [Factor types, Definition](#)
- Conditional random fields (CRFs) / [Conditional random fields](#)
- confusion matrix / [Confusion matrix and related metrics](#)
- conjugate gradient optimization algorithm
 - building / [Building a single-layer regression model](#)
- connectivity-based outliers (COF) / [How does it work?](#)
- content-based filtering
 - about / [Content-based filtering](#)
 - implementing, with book-recommendation engine / [Content-based filtering](#)
- contrastive divergence (CD) / [Contrastive divergence](#)
- Contrastive Divergence algorithm
 - about / [Restricted Boltzmann machine](#)
- Convolutional Neural Network (CNN)
 - about / [Deep convolutional networks](#)
- Convolutional Neural Networks (CNN)
 - about / [Convolutional Neural Network](#)
 - local connectivity / [Local connectivity](#)
 - parameter sharing / [Parameter sharing](#)
 - discrete convolution / [Discrete convolution](#)
 - Pooling or Subsampling / [Pooling or subsampling](#)
 - ReLU / [Normalization using ReLU](#)

- coreference resolution / [Coreference resolution](#)
- Core Motion framework, iOS
 - URL / [Mobile phone sensors](#)
- correlation-based feature selection (CFS) / [Correlation-based feature selection \(CFS\)](#)
- Correlation based Feature selection (CFS)
 - about / [Feature selection](#)
- correlation coefficient
 - about / [Correlation coefficient](#)
[/ Correlation coefficient](#)
- cosine distance
 - about / [Content-based filtering](#)
- Cosine distance / [Cosine distance](#)
- cost function
 - about / [Supervised learning](#)
- Coursera
 - URL / [Online courses](#)
- covariance / [Covariance](#)
- CoverType dataset
 - reference link / [Business problem](#)
- cross-industry applications, of affinity analysis
 - about / [Other applications in various areas](#)
 - medical diagnosis / [Medical diagnosis](#)
 - protein sequences / [Protein sequences](#)
 - census data / [Census data](#)
 - customer relationship management (CRM) / [Customer relationship management](#)
 - IT Operations Analytics / [IT Operations Analytics](#)
- cross-validation
 - about / [Cross-validation](#)
- Cross Industry Standard Process (CRISP)
 - about / [Process](#)
- Cross Industry Standard Process for Data Mining (CRISP-DM)
 - about / [CRISP-DM](#)
- CrowdANALYTIX
 - URL / [Competitions](#)
- CSVLoader class
 - URL / [Loading the data](#)

- cumulative sum (CUSUM) / [CUSUM and Page-Hinckley test](#)
- curse of dimensionality
 - about / [The curse of dimensionality](#)
- customer relationship database
 - about / [Customer relationship database](#)
 - challenge / [Challenge](#)
 - dataset / [Dataset](#)
 - evaluation / [Evaluation](#)
- custom frameworks / [Custom frameworks](#)

D

- (DBSCAN)
 - about / [DBSCAN](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- D-Separation, Bayesian networks / [D-Separation](#)
- data
 - about / [Data and problem definition](#)
- data acquisition
 - about / [Data acquisition](#)
 - publish-subscribe frameworks / [Publish-subscribe frameworks](#)
 - source-sink frameworks / [Source-sink frameworks](#)
 - SQL frameworks / [SQL frameworks](#)
 - message queueing frameworks / [Message queueing frameworks](#)
 - custom frameworks / [Custom frameworks](#)
- data analysis
 - about / [Data analysis](#)
 - label analysis / [Label analysis](#)
 - features analysis / [Features analysis](#)
- data and problem definition
 - about / [Data and problem definition](#)
- Data and problem definition
 - measurement scales / [Measurement scales](#)
- data cleaning
 - about / [Data cleaning](#)
- data collection
 - about / [Data collection](#)
 - data, observing / [Find or observe data](#)
 - data, searching / [Find or observe data](#)
 - data, generating / [Generate data](#)
 - traps, sampling / [Sampling traps](#)
 - from mobile phone / [Collecting data from a mobile phone](#)
 - Android Studio, installing / [Installing Android Studio](#)
 - data collector, loading / [Loading the data collector](#)

- training data, collecting / [Collecting training data](#)
 - mapping / [Data collection](#)
- data collector
 - loading / [Loading the data collector](#)
 - URL / [Loading the data collector](#)
 - feature extraction / [Feature extraction](#)
- data distribution sampling
 - about / [Data distribution sampling](#)
 - working / [How does it work?](#)
 - model change / [Expected model change](#)
 - error reduction / [Expected error reduction](#)
 - advantages / [Advantages and limitations](#)
 - disadvantages / [Advantages and limitations](#)
- Data Frame / [H2O architecture](#)
- data management
 - about / [Data management](#)
- Data Mining
 - URL / [Websites and blogs](#)
- Data Mining Research
 - URL / [Websites and blogs](#)
- data pre-processing
 - about / [Data pre-processing](#)
 - data cleaning / [Data cleaning](#)
 - missing values, filling / [Fill missing values](#)
 - outliers, removing / [Remove outliers](#)
 - data transformation / [Data transformation](#)
 - data reduction / [Data reduction](#)
- data preparation
 - key tasks / [Data processing and preparation](#)
 - HQL / [Hive and HQL](#)
 - Hive / [Hive and HQL](#)
 - Spark SQL / [Spark SQL](#)
 - Amazon Redshift / [Amazon Redshift](#)
 - real-time stream processing / [Real-time stream processing](#)
- data preprocessing
 - about / [Data transformation and preprocessing](#)
- data processing
 - key tasks / [Data processing and preparation](#)

- HQL / [Hive and HQL](#)
- Hive / [Hive and HQL](#)
- Spark SQL / [Spark SQL](#)
- Amazon Redshift / [Amazon Redshift](#)
- real-time stream processing / [Real-time stream processing](#)
- data quality analysis
 - about / [Data quality analysis](#)
 - / [Data quality analysis](#)
- data reduction
 - about / [Data reduction](#)
- data sampling
 - about / [Data sampling](#), [Data sampling and transformation](#)
 - need for / [Is sampling needed?](#)
 - undersampling / [Undersampling and oversampling](#)
 - oversampling / [Undersampling and oversampling](#)
 - stratified sampling / [Stratified sampling](#)
 - techniques / [Training, validation, and test set](#)
 - experiments / [Experiments, results, and analysis](#)
 - results / [Experiments, results, and analysis](#)
 - analysis / [Experiments, results, and analysis](#), [Feature relevance and analysis](#)
 - feature relevance / [Feature relevance and analysis](#)
 - test data, evaluation / [Evaluation on test data](#)
 - results, analysis / [Analysis of results](#)
- data science
 - about / [Machine learning and data science](#)
- Data Science Central
 - URL / [Websites and blogs](#)
- Data Science CS109 (Harvard) by John A. Paulson
 - URL / [Online courses](#)
- data scientist
 - about / [Machine learning and data science](#)
- dataset rebalancing
 - about / [Dataset rebalancing](#)
- datasets
 - about / [Datasets](#)
 - used, in machine learning / [Datasets used in machine learning](#)
 - structured data / [Datasets used in machine learning](#)

- transaction data / [Datasets used in machine learning](#)
 - market data / [Datasets used in machine learning](#)
 - unstructured data / [Datasets used in machine learning](#)
 - sequential data / [Datasets used in machine learning](#)
 - graph data / [Datasets used in machine learning](#)
- datasets, machine learning
 - about / [Datasets](#)
 - UC Irvine (UCI) database / [Datasets](#)
 - Tunedit / [Datasets](#)
 - Mldata.org / [Datasets](#)
 - KDD Challenge Datasets / [Datasets](#)
 - Kaggle / [Datasets](#)
- data storage
 - about / [Data storage](#)
 - HDFS / [HDFS](#)
 - NoSQL / [NoSQL](#)
- data transformation
 - about / [Data transformation](#), [Data transformation and preprocessing](#), [Data sampling and transformation](#)
 - feature, construction / [Feature construction](#)
 - missing values, handling / [Handling missing values](#)
 - outliers, handling / [Outliers](#)
 - discretization / [Discretization](#)
 - data sampling / [Data sampling](#)
 - training / [Training, validation, and test set](#)
 - validation / [Training, validation, and test set](#)
 - test set / [Training, validation, and test set](#)
- Davies-Bouldin index / [Davies-Bouldin index](#)
 - Silhouettes index / [Silhouette's index](#)
- Decision and Predictive Analytics (ADAPA) / [Predictive Model Markup Language](#)
- decision trees
 - about / [Underfitting and overfitting](#)
- Decision Trees
 - about / [Decision Trees](#)
 - algorithm input / [Algorithm inputs and outputs](#)
 - algorithm output / [Algorithm inputs and outputs](#)
 - working / [How does it work?](#)

- advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- decision trees learning
 - about / [Decision tree learning](#)
- Deep Autoencoders
 - about / [Deep Autoencoders](#)
- deep belief network
 - building / [Building a deep belief network](#)
- deep belief networks
 - about / [Artificial neural networks](#)
- Deep Belief Networks (DBN)
 - inputs and outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
- Deep Belief Networks (DBNs)
 - about / [Restricted Boltzmann machine](#)
- deep convolutional networks
 - about / [Deep convolutional networks, MNIST dataset](#)
- Deep feed-forward NN
 - about / [Deep feed-forward NN](#)
 - input and outputs / [Input and outputs](#)
 - working / [How does it work?](#)
- deep learning
 - about / [Deep learning](#)
 - building blocks / [Building blocks for deep learning](#)
 - Rectified linear activation function / [Rectified linear activation function](#)
 - Restricted Boltzmann Machines / [Restricted Boltzmann Machines](#)
 - Autoencoders / [Autoencoders](#)
 - Unsupervised pre-training and supervised fine-tuning / [Unsupervised pre-training and supervised fine-tuning](#)
 - Deep feed-forward NN / [Deep feed-forward NN , How does it work?](#)
 - Deep Autoencoders / [Deep Autoencoders](#)
 - Deep Belief Networks (DBN) / [Deep Belief Networks, Inputs and outputs](#)
 - Dropouts / [Deep learning with dropouts, Definition and mathematical notation](#)
 - sparse coding / [Sparse coding](#)
 - Convolutional Neural Network (CNN) / [Convolutional Neural Network](#)
 - Convolutional Neural Network (CNN) layers / [CNN Layers](#)
 - Recurrent Neural Networks (RNN) / [Recurrent Neural Networks](#)

- Deep Learning
 - about / [Deep learning and NLP](#)
- Deep Learning (DL) / [Feature relevance and analysis](#)
- deep learning, case study
 - about / [Case study](#)
 - tools and software / [Tools and software](#)
 - business problem / [Business problem](#)
 - machine learning mapping / [Machine learning mapping](#)
 - feature analysis / [Feature analysis](#)
 - models, results and evaluation / [Models, results, and evaluation](#)
 - basic data handling / [Basic data handling](#)
 - multi-layer Perceptron / [Multi-layer perceptron](#)
 - MLP, parameters / [Multi-layer perceptron](#)
 - MLP, code for / [Code for MLP](#)
 - Convolutional Network / [Convolutional Network](#)
 - Convolutional Network, code for / [Code for CNN](#)
 - Variational Autoencoder / [Variational Autoencoder](#), [Code for Variational deep learning, case study](#)
[Variational Autoencoder](#)[Autoencoder](#)
 - DBN / [DBN](#)
 - parameter search, Arbiter used / [Parameter search using Arbiter](#)
 - results and analysis / [Results and analysis](#)
- Deeplearning4j
 - about / [Deeplearning4j](#)
 - URL / [Deeplearning4j](#)
 - org.deeplearning4j.base / [Deeplearning4j](#)
 - org.deeplearning4j.berkeley / [Deeplearning4j](#)
 - org.deeplearning4j.clustering / [Deeplearning4j](#)
 - org.deeplearning4j.datasets / [Deeplearning4j](#)
 - org.deeplearning4j.distributions / [Deeplearning4j](#)
 - org.deeplearning4j.eval / [Deeplearning4j](#)
 - org.deeplearning4j.exceptions / [Deeplearning4j](#)
 - org.deeplearning4j.models / [Deeplearning4j](#)
 - org.deeplearning4j.nn / [Deeplearning4j](#)
 - org.deeplearning4j.optimize / [Deeplearning4j](#)
 - org.deeplearning4j.plot / [Deeplearning4j](#)
 - org.deeplearning4j.rng / [Deeplearning4j](#)
 - org.deeplearning4j.util / [Deeplearning4j](#)
- DeepLearning4J

- URL / [Machine learning – tools and datasets](#)
 - about / [Machine learning – tools and datasets](#)
- deeplearning4java
 - about / [Deeplearning4j](#)
 - obtaining / [Getting DL4J](#)
- delta rule
 - about / [Perceptron](#)
- density
 - estimation / [Density estimation](#)
- density-based methods / [Outliers](#)
 - about / [Density-based methods](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- density based algorithm
 - about / [Density based](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- descriptive quality analysis
 - about / [Descriptive data analysis](#)
 - basic label analysis / [Basic label analysis](#)
 - basic feature analysis / [Basic feature analysis](#)
- detection methods
 - model evolution, monitoring / [Monitoring model evolution](#)
 - distribution changes, monitoring / [Monitoring distribution changes](#)
- Deviance-Threshold Measure / [Measures to evaluate structures](#)
- Dice coefficient / [Dice coefficient](#)
- dimensionality reduction
 - about / [Feature relevance analysis and dimensionality reduction, Feature analysis and dimensionality reduction](#)
 - notation / [Notation](#)
 - linear models / [Linear methods](#)
 - nonlinear methods / [Nonlinear methods](#)

- PCA / [PCA](#)
- random projections / [Random projections](#)
- ISOMAP / [ISOMAP](#)
- observation / [Observations on feature analysis and dimensionality reduction](#)
- / [Dimensionality reduction](#)
- Directed Acyclic Graph (DAG) / [Definition](#)
- directory
 - text data, importing / [Importing from directory](#)
- Direct Update of Events (DUE) / [Direct Update of Events \(DUE\)](#)
- Dirichlet distribution / [Prior and posterior using the Dirichlet distribution](#)
- Discrete Fourier Transform (DFT)
 - about / [Activity recognition pipeline](#)
- discretization
 - about / [Discretization](#)
 - by binning / [Discretization](#)
 - by frequency / [Discretization](#)
 - by entropy / [Discretization](#)
- distance-based clustering
 - for outlier detection / [Distance-based clustering for outlier detection](#)
- distance-based methods / [Outliers](#)
 - about / [Distance-based methods](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- distance measures
 - Euclidean distances / [Euclidean distances](#)
 - non-Euclidean distances / [Non-Euclidean distances](#)
- distribution changes, monitoring
 - about / [Monitoring distribution changes](#)
 - Welchs test / [Welch's t test](#)
 - Kolmogorov-Smirnovs test / [Kolmogorov-Smirnov's test](#)
 - Page-Hinckley test / [CUSUM and Page-Hinckley test](#)
 - cumulative sum (CUSUM) / [CUSUM and Page-Hinckley test](#)
- divide-and-conquer strategy
 - about / [FP-growth algorithm](#)

- document collection
 - about / [Document collection and standardization](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
- document databases / [Document databases](#)
- document frequency (DF) / [Frequency-based techniques](#)
- double evaluateLeftToRight method
 - Instances heldOutDocuments component / [Evaluating a model](#)
 - int numParticles component / [Evaluating a model](#)
 - boolean useResampling component / [Evaluating a model](#)
 - PrintStream docProbabilityStream component / [Evaluating a model](#)
- drift detection
 - about / [Concept drift and drift detection](#)
 - data management / [Data management](#)
 - partial memory / [Partial memory](#)
- drift detection method (DDM) / [Drift Detection Method or DDM](#)
- DrivenData / [Competitions](#)
- DropConnect neural network
 - about / [MNIST dataset](#)
- Dropouts
 - about / [Deep learning with dropouts](#)
 - definition and mathematical notation / [Definition and mathematical notation](#), [How does it work?](#)
 - training with / [Learning Training and testing with dropouts](#)
 - testing with / [Learning Training and testing with dropouts](#)
- DSGuide
 - URL / [Websites and blogs](#)
- Dunns Indices / [Dunn's Indices](#)
- dynamic time wrapping (DTW)
 - about / [Java machine learning](#)

E

- early drift detection method (EDDM) / [Early Drift Detection Method or EDDM](#)
- Eclipse
 - Apache Mahout, configuring with Maven plugin / [Configuring Mahout in Eclipse with the Maven plugin](#)
- Eclipse IDE
 - using / [Before you start](#)
- Edit distance
 - about / [Non-Euclidean distances](#)
- eigendecomposition / [Eigendecomposition](#)
- elbow method
 - about / [Clustering](#)
- ELEC dataset / [Data collection](#)
- elimination-based inference, Bayesian networks
 - about / [Elimination-based inference](#)
 - variable elimination algorithm / [Variable elimination algorithm](#)
 - input and output / [Input and output](#)
 - VE algorithm, advantages / [Advantages and limitations](#)
- Elki
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
- EM (Diagonal Gaussian Model Factory) / [Clustering models, results, and evaluation](#)
- email spam dataset
 - URL / [E-mail spam dataset](#)
- email spam detection
 - about / [E-mail spam detection](#)
 - email spam dataset, collecting / [E-mail spam dataset](#)
 - default pipeline, creating / [Feature generation](#)
 - training / [Training and testing](#)
 - testing / [Training and testing](#)
 - model performance, evaluating / [Model performance](#)
- embedded approach / [Embedded approach](#)
- energy efficiency dataset
 - URL / [Loading the data](#)
- ensambleSel.setOptions () method
 - -L </path/to/modelLibrary> option / [Model selection](#)

- -W </path/to/working/directory> option / [Model selection](#)
 - -B <numModelBags> option / [Model selection](#)
 - -E <modelRatio> option / [Model selection](#)
 - -V <validationRatio> option / [Model selection](#)
 - -H <hillClimbIterations> option / [Model selection](#)
 - -I <sortInitialization> option / [Model selection](#)
 - -X <numFolds> option / [Model selection](#)
 - -P <hillclimbMetrc> option / [Model selection](#)
 - -A <algorithm> option / [Model selection](#)
 - -R option / [Model selection](#)
 - -G option / [Model selection](#)
 - -O option / [Model selection](#)
 - -S <num> option / [Model selection](#)
 - -D option / [Model selection](#)
- ensemble algorithms
 - about / [Ensemble algorithms](#)
 - weighted majority algorithm (WMA) / [Weighted majority algorithm](#)
 - online bagging algorithm / [Online Bagging algorithm](#)
 - online boosting algorithm / [Online Boosting algorithm](#)
- ensemble learning
 - about / [Ensemble learning](#), [Ensemble learning and meta learners](#)
 - types / [Ensemble learning and meta learners](#)
 - bootstrap aggregating (bagging) / [Bootstrap aggregating or bagging](#)
 - boosting / [Boosting](#)
- ensembleLibrary package
 - using / [Before we start](#)
 - URL / [Before we start](#)
- ensembles
 - used, for advanced modelling / [Advanced modeling with ensembles](#)
- Ensemble Selection algorithm
 - about / [Advanced modeling with ensembles](#)
- environmental sensors
 - about / [Mobile phone sensors](#)
- error propagation / [Error propagation](#)
- error reduction
 - variance reduction / [Variance reduction](#)
 - density weighted methods / [Density weighted methods](#)
- Euclidean distance / [Euclidean distance](#)

- Euclidean distances
 - about / [Euclidean distances](#)
- EUPLv1.1
 - URL / [Machine learning – tools and datasets](#)
- evaluate() method, parameters
 - RecommenderBuilder / [Evaluation](#)
 - DataModelBuilder / [Evaluation](#)
 - DataModel / [Evaluation](#)
 - trainingPercentage / [Evaluation](#)
 - evaluationPercentage / [Evaluation](#)
- evaluation
 - about / [Generalization and evaluation](#)
- evaluation criteria
 - accuracy / [Evaluation criteria](#)
 - balanced accuracy / [Evaluation criteria](#)
 - Area under ROC curve (AUC) / [Evaluation criteria](#)
 - Kappa statistic (K) / [Evaluation criteria](#)
 - Kappa Plus statistic / [Evaluation criteria](#)
- evaluation measures, clustering
 - Cluster Mapping Measures (CMM) / [Cluster Mapping Measures \(CMM\)](#)
 - V-Measure / [V-Measure](#)
 - other measures / [Other external measures](#)
 - purity / [Other external measures](#)
 - entropy / [Other external measures](#)
 - Recall / [Other external measures](#)
 - F-Measure / [Other external measures](#)
 - Precision / [Other external measures](#)
- Exact Storm / [Exact Storm](#)
- Expectation Maximization (EM) / [Advantages and limitations](#)
 - about / [Expectation maximization \(EM\) or Gaussian mixture modeling \(GMM\), How does it work?, How does it work?](#)
- Expectation Maximization (EM) clustering
 - about / [Clustering](#)
- exploitation
 - about / [Exploitation versus exploration](#)
- exploration
 - about / [Exploitation versus exploration](#)
- extended Jaccard Coefficient / [Extended Jaccard coefficient](#)

- external evaluation measures
 - about / [External evaluation measures](#)
 - Rand index / [Rand index](#)
 - F-Measure / [F-Measure](#)
 - normalized mutual information index (NMI) / [Normalized mutual information index](#)

F

- F-Measure / [F-Measure](#)
- False Positive Rate (FPR) / [Confusion matrix and related metrics](#)
- feature analysis
 - about / [Feature analysis and dimensionality reduction](#)
 - notation / [Notation](#)
 - observation / [Observations on feature analysis and dimensionality reduction](#)
- feature evaluation techniques
 - about / [Feature evaluation techniques](#)
 - filter approach / [Filter approach](#)
 - wrapper approach / [Wrapper approach](#)
 - embedded approach / [Embedded approach](#)
- Feature extraction
 - about / [Building a machine learning application](#)
- feature extraction/generation
 - about / [Feature extraction/generation](#)
 - lexical features / [Lexical features](#)
 - syntactic features / [Syntactic features](#)
 - semantic features / [Semantic features](#)
- feature map
 - about / [Deep convolutional networks](#)
- feature relevance analysis
 - about / [Feature relevance analysis and dimensionality reduction](#)
 - feature search techniques / [Feature search techniques](#)
 - feature evaluation techniques / [Feature evaluation techniques](#)
- features
 - construction / [Feature construction](#)
- feature search techniques / [Feature search techniques](#)
- feature selection
 - about / [Data reduction, Feature selection](#)
 - Information theoretic techniques / [Information theoretic techniques](#)
 - statistical-based techniques / [Statistical-based techniques](#)
 - frequency-based techniques / [Frequency-based techniques](#)
- feedforward neural networks
 - about / [Feedforward neural networks](#)
- file

- text data, importing / [Importing from file](#)
- filter approach
 - about / [Filter approach](#)
 - univariate feature selection / [Univariate feature selection](#)
 - multivariate feature selection / [Multivariate feature selection](#)
- Fine Needle Aspirate (FNA) / [Datasets and analysis](#)
- flow of influence, Bayesian networks / [Flow of influence](#)
- Fourier transform
 - reference link / [Activity recognition pipeline](#)
- FP-Growth
 - about / [Weka](#)
- FP-growth algorithm
 - about / [FP-growth algorithm](#)
 - used, for discovering shopping patterns / [FP-growth](#)
- FP-tree structure
 - about / [FP-growth algorithm](#)
- fraud detection, of insurance claims
 - about / [Fraud detection of insurance claims](#)
 - dataset, using / [Dataset](#)
 - suspicious patterns, modelling / [Modeling suspicious patterns](#)
- frequent pattern (FP)
 - about / [FP-growth algorithm](#)
- Friedmans test / [Friedman's test](#)

G

- Gain charts / [Gain charts and lift curves](#)
- Gain Ratio (GR) / [Information theoretic techniques](#)
- Gaussian distribution / [Gaussian distribution](#)
- Gaussian Mixture Model (GMM) / [Gaussian Mixture Model](#)
- Gaussian mixture modeling (GMM)
 - about / [Expectation maximization \(EM\) or Gaussian mixture modeling \(GMM\)](#), [How does it work?](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Gaussian Radial Basis Kernel / [How does it work?](#)
- Gaussian standard deviation / [Gaussian standard deviation](#)
- Geeking with Greg
 - URL / [Websites and blogs](#)
- generalization
 - about / [Generalization and evaluation](#)
 - underfitting / [Underfitting and overfitting](#)
 - overfitting / [Underfitting and overfitting](#)
 - test set / [Train and test sets](#)
 - train set / [Train and test sets](#)
 - cross-validation / [Cross-validation](#)
 - leave-one-out validation / [Leave-one-out validation](#)
 - stratification / [Stratification](#)
- Generalized Linear Models (GLM) / [Feature relevance and analysis](#)
- Generalized Sequential Patterns (GSP)
 - about / [Weka](#)
- generative probabilistic models
 - about / [Generative probabilistic models](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- Generative Stochastic Networks (GSNs)

- about / [Restricted Boltzmann machine](#)
- Gibbs parameterization / [Gibbs parameterization](#)
- Gibbs sampling
 - about / [Restricted Boltzmann machine](#)
- Gini index / [How does it work?](#)
- GNU General Public License (GNU GPL)
 - about / [Weka](#)
- Google Prediction API / [Machine learning as a service](#)
- Gradient Boosting Machine (GBM) / [Feature relevance and analysis](#)
- graph
 - concepts / [Graph concepts](#)
 - structure and properties / [Graph structure and properties](#)
 - subgraphs and cliques / [Subgraphs and cliques](#)
 - path / [Path, trail, and cycles](#)
 - trail / [Path, trail, and cycles](#)
 - cycles / [Path, trail, and cycles](#)
- graph data / [Datasets used in machine learning](#)
- graph databases / [Graph databases](#)
- Graphics Processing Unit (GPU)
 - reference link / [Build a Multilayer Convolutional Network](#)
 - about / [Build a Multilayer Convolutional Network](#)
- graph mining / [Machine learning – types and subtypes](#)
- GraphX
 - about / [Apache Spark, Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
- grid based algorithm
 - about / [Grid based](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)

H

- H2O
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
 - as Big Data Machine Learning platform / [H2O as Big Data Machine Learning platform](#)
 - architecture / [H2O architecture](#)
 - machine learning / [Machine learning in H2O](#)
 - tools / [Tools and usage](#)
 - usage / [Tools and usage](#)
- Hadoop
 - about / [Big data application architecture](#)
 - URL / [Big data application architecture](#)
- Hadoop Distributed File System (HDFS)
 - about / [Apache Spark](#)
- Hamming distance
 - about / [Non-Euclidean distances](#)
- HBase
 - about / [Big data application architecture](#)
 - URL / [Big data application architecture](#)
 - / [Columnar databases](#)
- HDFS
 - about / [HDFS](#)
- HDFS, components / [HDFS](#)
 - NameNode / [HDFS](#)
 - Secondary NameNode / [HDFS](#)
 - DataNode / [HDFS](#)
- Hidden layer
 - about / [Feedforward neural networks](#)
- Hidden layer, issues
 - vanishing gradients problem / [Feedforward neural networks](#)
 - overfitting / [Feedforward neural networks](#)
- Hidden Markov models
 - about / [Hidden Markov models for NER](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)

- advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- hidden Markov models (HMM)
 - about / [Apache Mahout](#)
[/ How does it work?](#)
- Hidden Markov models (HMM) / [Hidden Markov models](#)
- Hidden Markov Models (HMMs)
 - about / [Transaction analysis](#)
- hierarchical clustering
 - about / [Clustering, Hierarchical clustering](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - single linkage / [How does it work?](#)
 - complete linkage / [How does it work?](#)
 - average linkage / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- high-dimensional-based methods
 - about / [High-dimensional-based methods](#)
 - inputs / [Inputs and outputs](#)
 - ouputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- histogram-based anomaly detection
 - about / [Histogram-based anomaly detection](#)
- Hive / [Hive and HQL](#)
- Hoeffding Trees (HT) / [Hoeffding trees or very fast decision trees \(VFDT\)](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - limitations / [Advantages and limitations](#)
 - advantages / [Advantages and limitations](#)
- Horse Colic Classification, case study
 - about / [Case Study – Horse Colic Classification](#)
 - reference link / [Case Study – Horse Colic Classification](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine learning mapping](#)

- data analysis / [Data analysis](#)
- supervised learning, experiments / [Supervised learning experiments](#)
- analysis / [Results, observations, and analysis](#)
- Hortonworks Data Platform (HDP) / [Hortonworks Data Platform](#)
- Hotspot
 - about / [Weka](#)
- HQL / [Hive and HQL](#)
- hybrid approach
 - about / [Hybrid approach](#)
- Hyperbolic tangent (/ [Hyperbolic tangent \("tanh"\) function](#)
- hyperplane / [How does it work?](#)

I

- I-Map, Bayesian networks / [I-Map](#)
- IBM Research team
 - about / [Advanced modeling with ensembles](#)
- IBM Watson Analytics / [Machine learning as a service](#)
- image classification
 - about / [Image classification](#)
 - deeplearning4java / [Deeplearning4j](#)
 - MNIST dataset / [MNIST dataset](#)
 - data, loading / [Loading the data](#)
 - models, building / [Building models](#)
- ImageNet
 - about / [Introducing image recognition](#)
 - URL / [Deep convolutional networks](#)
- image recognition
 - about / [Introducing image recognition](#)
 - neural networks / [Neural networks](#)
- incremental learning / [Machine learning – types and subtypes](#)
- incremental supervised learning
 - about / [Incremental supervised learning](#)
 - modeling techniques / [Modeling techniques](#)
 - validation / [Validation, evaluation, and comparisons in online setting](#)
 - evaluation / [Validation, evaluation, and comparisons in online setting](#)
 - comparisons, in online setting / [Validation, evaluation, and comparisons in online setting](#)
 - model validation techniques / [Model validation techniques](#)
- incremental unsupervised learning
 - clustering, using / [Incremental unsupervised learning using clustering](#)
 - modeling techniques / [Modeling techniques](#)
- independent, identical distributions (i.i.d.) / [Monitoring model evolution](#)
- Independent Component Analysis (ICA) / [Advantages and limitations](#)
- inference, Bayesian networks
 - about / [Inference](#)
 - elimination-based inference / [Elimination-based inference](#)
 - propagation-based techniques / [Propagation-based techniques](#)
 - sampling-based techniques / [Sampling-based techniques](#)
- inferencing / [Machine learning – types and subtypes](#), [Semantic reasoning and](#)

inferencing

- influence space (IS) / [How does it work?](#)
- information extraction / [Information extraction and named entity recognition](#)
- Information gain (IG) / [Information theoretic techniques](#)
- Infrastructure as a Service (IaaS) / [Machine learning in the cloud](#)
- Input layer
 - about / [Feedforward neural networks](#)
- insurance claims
 - fraud detection / [Fraud detection of insurance claims](#)
- internal evaluation measures
 - about / [Internal evaluation measures](#)
 - compactness / [Internal evaluation measures](#)
 - separation / [Internal evaluation measures](#)
 - notation / [Notation](#)
 - R-Squared / [R-Squared](#)
 - Dunns Indices / [Dunn's Indices](#)
 - Davies-Bouldin index / [Davies-Bouldin index](#)
- Internet of things (IoT) / [Machine learning applications](#)
- Interquartile Ranges (IQR) / [Outliers](#)
- interval data
 - about / [Measurement scales](#)
- Intrusion Detection (ID)
 - about / [Transaction analysis](#)
- inverse document frequency (IDF) / [Inverse document frequency \(IDF\)](#)
- Isomap / [Advantages and limitations](#)
- item-based analysis
 - about / [User-based and item-based analysis](#)
- item-based collaborative filtering
 - about / [Item-based filtering](#)
- iterative reweighted least squares (IRLS) / [How does it work?](#)

J

- Jaccard distance
 - about / [Non-Euclidean distances](#)
- Java
 - need for / [The need for Java](#)
- Java-ML packages
 - net.sf.javaml.classification / [Java machine learning](#)
 - net.sf.javaml.clustering / [Java machine learning](#)
 - net.sf.javaml.core / [Java machine learning](#)
 - net.sf.javaml.distance / [Java machine learning](#)
 - net.sf.javaml.featureselection / [Java machine learning](#)
 - net.sf.javaml.filter / [Java machine learning](#)
 - net.sf.javaml.matrix / [Java machine learning](#)
 - net.sf.javaml.sampling / [Java machine learning](#)
 - net.sf.javaml.tools / [Java machine learning](#)
 - net.sf.javaml.utils / [Java machine learning](#)
- java -Xmx16g
 - about / [Performance evaluation](#)
- Java API packages, Weka
 - weka.associations / [Weka](#)
 - weka.classifiers / [Weka](#)
 - weka.clusterers / [Weka](#)
 - weka.core / [Weka](#)
 - weka.datagenerators / [Weka](#)
 - weka.estimators / [Weka](#)
 - weka.experiment / [Weka](#)
 - weka.filters / [Weka](#)
 - weka.gui / [Weka](#)
- Java Class Library for Active Learning (JCLAL)
 - URL / [Machine learning – tools and datasets](#)
 - about / [Machine learning – tools and datasets](#)
 - reference link / [Tools and software](#)
- Java machine learning (Java-ML)
 - about / [Java machine learning](#)
 - URL / [Java machine learning](#)
- JavaScript Object Notation (JSON) / [Document collection and standardization](#)
- JKKernelMachines (Transductive SVM) / [Tools and software](#)

- joint distribution / [Random variables, joint, and marginal distributions](#), [Factor types](#)

K

- k-means
 - about / [k-Means](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- k-means clustering
 - about / [Clustering](#)
- k-nearest neighbors
 - about / [Underfitting and overfitting](#)
- k-Nearest Neighbors (k-NN) / [Outliers](#)
- K-Nearest Neighbors (KNN)
 - about / [K-Nearest Neighbors \(KNN\)](#)
 - algorithm input / [Algorithm inputs and outputs](#)
 - algorithm output / [Algorithm inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Kaggle / [Competitions](#)
 - about / [Datasets](#)
 - URL / [Datasets](#)
- KDD Challenge Datasets
 - URL / [Datasets](#)
 - about / [Datasets](#)
- KDD Cup
 - URL / [Getting the data](#)
- KDnuggets / [Machine learning as a service](#)
 - URL / [Websites and blogs](#)
- KEEL
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
- KEEL (Knowledge Extraction based on Evolutionary Learning)
 - about / [Tools and software](#)
 - reference link / [Tools and software](#)
- kernel density estimation (KDE) / [How does it work?](#)

- kernel methods
 - about / [Kernel methods](#)
- Kernel Principal Component Analysis (KPCA)
 - about / [Kernel Principal Component Analysis \(KPCA\)](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- kernel trick / [How does it work?](#)
- key-value databases / [Key-value databases](#)
- Key Performance Indicators (KPIs) / [What is not machine learning?](#)
- Knime
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
- KNIME
 - about / [KNIME](#)
 - references / [KNIME](#)
- known-knowns
 - about / [Unknown-unknowns](#)
- known-unknowns
 - about / [Unknown-unknowns](#)
- Kohonen networks / [How does it work?](#)
- Kolmogorov-Smirnovs test / [Kolmogorov-Smirnov's test](#)
- Kubat / [Widmer and Kubat](#)
- Kullback-Leibler (KL) / [How does it work?](#)

L

- label SSL
 - about / [Cluster and label SSL](#)
 - output / [Inputs and outputs](#)
 - input / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Latent Dirichlet
 - about / [MALLET](#)
- Latent Dirichlet Allocation
 - about / [Topic modeling](#)
- Latent Dirichlet Allocation (LDA)
 - about / [Modeling](#)
/ [Advantages and limitations](#)
- latent semantic analysis (LSA) / [Dimensionality reduction](#)
- learning
 - techniques / [Training, validation, and test set](#)
- learning, Bayesian networks
 - goals / [Learning](#)
 - parameters / [Learning parameters](#)
 - Maximum likelihood estimation (MLE) / [Maximum likelihood estimation for Bayesian networks](#)
 - Bayesian parameter, estimation / [Bayesian parameter estimation for Bayesian network](#)
 - Dirichlet distribution / [Prior and posterior using the Dirichlet distribution](#)
 - structures / [Learning structures](#)
 - structures, evaluating / [Measures to evaluate structures](#)
 - structures, learning / [Methods for learning structures, Advantages and limitations](#)
 - constraint-based techniques / [Constraint-based techniques](#)
 - advantages and limitations / [Advantages and limitations](#)
 - search and score-based techniques / [Search and score-based techniques, How does it work?, Advantages and limitations](#)
- leave-one-out validation
 - about / [Leave-one-out validation](#)
- lemmatization

- about / [Stemming or lemmatization](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
- Leveraging Bagging (LB) / [Supervised learning experiments](#)
- lexical features
 - about / [Lexical features](#)
 - character-based features / [Character-based features](#)
 - word-based features / [Word-based features](#)
 - part-of-speech tagging features / [Part-of-speech tagging features](#)
 - taxonomy features / [Taxonomy features](#)
- lift curves / [Gain charts and lift curves](#)
- linear algorithm
 - online linear models, with loss functions / [Online linear models with loss functions](#)
 - Online Naive Bayes / [Online Naïve Bayes](#)
- Linear Discriminant Analysis (LDA)
 - about / [Modeling](#)
 - reference link / [Evaluating a model](#)
- Linear Embedding (LLE) / [How does it work?](#)
- linear models / [Linear models](#)
 - Linear Regression / [Linear Regression](#)
 - Naive Bayes / [Naïve Bayes](#)
 - Logistic Regression / [Logistic Regression](#)
 - about / [Linear methods](#)
 - principal component analysis (PCA) / [Principal component analysis \(PCA\)](#)
 - random projections (RP) / [Random projections \(RP\)](#)
 - Multidimensional Scaling (MDS) / [Multidimensional Scaling \(MDS\)](#)
- Linear Regression
 - about / [Linear Regression](#)
 - algorithm input / [Algorithm input and output](#)
 - algorithm output / [Algorithm input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- linear regression
 - about / [Linear regression](#)
- Lloyds algorithm

- working / [How does it work?](#)
- Local Outlier Factor (LOF)
 - about / [Histogram-based anomaly detection](#)
[/ How does it work?](#)
- LOF algorithm
 - URL / [Density based k-nearest neighbors](#)
- logical datasets
 - about / [Training, validation, and test set](#)
- Logistic Regression
 - about / [Logistic Regression](#)
 - algorithm input / [Algorithm input and output](#)
 - algorithm output / [Algorithm input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)

M

- machine learning / [Machine Learning](#)
 - about / [Machine learning and data science](#)
 - advantages / [What kind of problems can machine learning solve?](#)
 - supervised learning / [What kind of problems can machine learning solve?, Machine learning – types and subtypes](#)
 - unsupervised learning / [What kind of problems can machine learning solve?](#)
 - reinforcement learning / [What kind of problems can machine learning solve?, Machine learning – types and subtypes](#)
 - in real life / [Machine learning in real life](#)
 - noisy data / [Noisy data](#)
 - class unbalance / [Class unbalance](#)
 - feature selection / [Feature selection is hard](#)
 - model chaining / [Model chaining](#)
 - evaluation / [Importance of evaluation](#)
 - models, in production / [Getting models into production](#)
 - models, maintaining / [Model maintenance](#)
 - in cloud / [Machine learning in the cloud](#)
 - as service / [Machine learning as a service](#)
 - history / [Machine learning – history and definition](#)
 - definition / [Machine learning – history and definition](#)
 - relationship with / [Machine learning – history and definition](#)
 - concepts / [Machine learning – concepts and terminology](#)
 - terminology / [Machine learning – concepts and terminology](#)
 - types / [Machine learning – types and subtypes](#)
 - subtypes / [Machine learning – types and subtypes](#)
 - semi-supervised learning / [Machine learning – types and subtypes](#)
 - graph mining / [Machine learning – types and subtypes](#)
 - probabilistic graph modeling / [Machine learning – types and subtypes](#)
 - inferencing / [Machine learning – types and subtypes](#)
 - time-series forecasting / [Machine learning – types and subtypes](#)
 - association analysis / [Machine learning – types and subtypes](#)
 - stream learning / [Machine learning – types and subtypes](#)
 - incremental learning / [Machine learning – types and subtypes](#)
 - datasets, used / [Datasets used in machine learning](#)
 - practical issues / [Practical issues in machine learning](#)

- roles / [Machine learning – roles and process](#)
 - process / [Machine learning – roles and process](#)
 - tools / [Machine learning – tools and datasets](#)
 - datasets / [Machine learning – tools and datasets](#)
 - mapping / [Machine learning mapping](#), [Machine learning mapping](#)
 - in H2O / [Machine learning in H2O](#)
 - future / [The future of Machine Learning](#)
- machine learning application
 - building / [Building a machine learning application](#)
 - traditional machine learning / [Traditional machine learning architecture](#)
 - big data, dealing with / [Dealing with big data](#)
- machine learning applications / [Machine learning applications](#)
- Machine Learning for Language Toolkit (MALLET)
 - about / [MALLET](#)
 - URL / [MALLET](#)
- machine learning libraries
 - about / [Machine learning libraries](#)
 - Waikato Environment for Knowledge Analysis (Weka) / [Weka](#)
 - Java machine learning (Java-ML) / [Java machine learning](#)
 - Apache Mahout / [Apache Mahout](#)
 - Apache Spark / [Apache Spark](#)
 - Deeplearning4j / [Deeplearning4j](#)
 - Machine Learning for Language Toolkit (MALLET) / [MALLET](#)
 - comparing / [Comparing libraries](#)
- Machine learning mastery
 - URL / [Websites and blogs](#)
- machine translation (MT) / [Machine translation](#)
- Mahalanobis distance
 - about / [Non-Euclidean distances](#), [Java machine learning](#)
- Mahout interfaces, abstractions
 - DataModel / [Collaborative filtering](#)
 - UserSimilarity / [Collaborative filtering](#)
 - ItemSimilarity / [Collaborative filtering](#)
 - UserNeighborhood / [Collaborative filtering](#)
 - Recommender / [Collaborative filtering](#)
- Mahout libraries
 - org.apache.mahout.cf.taste / [Apache Mahout](#)
 - org.apache.mahout.classifier / [Apache Mahout](#)

- org.apache.mahout.clustering / [Apache Mahout](#)
- org.apache.mahout.common / [Apache Mahout](#)
- org.apache.mahout.ep / [Apache Mahout](#)
- org.apache.mahout.math / [Apache Mahout](#)
- org.apache.mahout.vectorizer / [Apache Mahout](#)
- Mallet
 - installing / [Installing Mallet](#)
 - URL / [Installing Mallet, Machine learning – tools and datasets](#)
 - reference link / [Pre-processing text data](#)
 - about / [Machine learning – tools and datasets](#)
- mallet
 - topic modeling / [Topic modeling with mallet](#)
- MALLET, packages
 - cc.mallet.classify / [MALLET](#)
 - cc.mallet.cluster / [MALLET](#)
 - cc.mallet.extract / [MALLET](#)
 - cc.mallet.fst / [MALLET](#)
 - cc.mallet.grmm / [MALLET](#)
 - cc.mallet.optimize / [MALLET](#)
 - cc.mallet.pipe / [MALLET](#)
 - cc.mallet.topics / [MALLET](#)
 - cc.mallet.types / [MALLET](#)
 - cc.mallet.util / [MALLET](#)
- Manhattan distance
 - about / [Java machine learning](#)
- manifold learning
 - about / [Manifold learning](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- marginal distribution / [Random variables, joint, and marginal distributions](#)
- market basket analysis (MBA)
 - about / [Market basket analysis](#)
 - item affinity / [Market basket analysis](#)
 - identification, of driver items / [Market basket analysis](#)

- trip classification / [Market basket analysis](#)
 - store-to-store comparison / [Market basket analysis](#)
 - revenue optimization / [Market basket analysis](#)
 - marketing / [Market basket analysis](#)
 - operations optimization / [Market basket analysis](#)
 - affinity analysis / [Affinity analysis](#)
- market data / [Datasets used in machine learning](#)
- Markov blanket / [Markov blanket](#)
- Markov chain
 - about / [Restricted Boltzmann machine](#)
- Markov chains
 - about / [Markov chains](#)
 - Hidden Markov models (HMM) / [Hidden Markov models](#)
 - Hidden Markov models (HMM), portable path / [Most probable path in HMM](#)
 - Hidden Markov models (HMM), posterior decoding / [Posterior decoding in HMM](#)
- Markov networks (MN) / [Markov networks and conditional random fields](#)
 - representation / [Representation](#)
 - parameterization / [Parameterization](#)
 - Gibbs parameterization / [Gibbs parameterization](#)
 - factor graphs / [Factor graphs](#)
 - log-linear models / [Log-linear models](#)
 - independencies / [Independencies](#)
 - global / [Global](#)
 - Pairwise Markov / [Pairwise Markov](#)
 - Markov blanket / [Markov blanket](#)
 - inference / [Inference](#)
 - learning / [Learning](#)
 - Conditional random fields (CRFs) / [Conditional random fields](#)
- Markov random field (MRF) / [Markov networks and conditional random fields](#)
- massively parallel processing (MPP) / [Amazon Redshift](#)
- Massive Online Analysis (MOA)
 - about / [Tools and software](#)
 - references / [Tools and software](#)
 - reference link / [Analysis of stream learning results](#)
- mathematical transformation
 - of feature / [Outliers](#)

- matrix
 - about / [Matrix](#)
 - transpose / [Transpose of a matrix](#)
 - addition / [Matrix addition](#)
 - scalar multiplication / [Scalar multiplication](#)
 - multiplication / [Matrix multiplication](#)
- matrix product, properties
 - about / [Properties of matrix product](#)
 - linear transformation / [Linear transformation](#)
 - matrix inverse / [Matrix inverse](#)
 - eigendecomposition / [Eigendecomposition](#)
 - positive definite matrix / [Positive definite matrix](#)
- Maven plugin
 - Apache Mahout, configuring with / [Configuring Mahout in Eclipse with the Maven plugin](#)
- maximum entropy Markov model (MEMM)
 - about / [Maximum entropy Markov models for NER](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- Maximum Likelihood Estimates (MLE) / [How does it work?](#)
- Maximum likelihood estimation (MLE) / [Maximum likelihood estimation for Bayesian networks](#)
- McNemars Test / [McNemar's Test](#)
- McNemar test / [Comparing algorithms and metrics](#)
- mean / [Mean](#)
- mean absolute error
 - about / [Mean absolute error](#)
- mean shift
 - about / [Mean shift](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- mean squared error

- about / [Mean squared error](#)
- measurement scales
 - about / [Measurement scales](#)
 - nominal data / [Measurement scales](#)
 - ordinal data / [Measurement scales](#)
 - interval data / [Measurement scales](#)
 - ratio data / [Measurement scales](#)
- meta learners
 - about / [Ensemble learning and meta learners](#)
- Micro Clustering based Algorithm (MCOD) / [Micro Clustering based Algorithm \(MCOD\)](#)
- Microsoft Azure HDInsight / [Microsoft Azure HDInsight](#)
- Microsoft Azure Machine Learning / [Machine learning as a service](#)
- Min-Max Normalization / [Outliers](#)
- minimal redundancy maximal relevance (mRMR) / [Minimal redundancy maximal relevance \(mRMR\)](#)
- Minimum Covariant Determinant (MCD) / [How does it work?](#)
- Minimum Description Length (MDL) / [How does it work?](#)
- Minkowski distance
 - about / [Java machine learning](#)
- missing values
 - filling / [Fill missing values](#)
 - handling / [Handling missing values, Results, observations, and analysis](#)
- Mixed National Institute of Standards and Technology (MNIST) / [Data quality analysis](#)
- Mldata.org
 - about / [Datasets](#)
 - URL / [Datasets](#)
- MLlib API library
 - org.apache.spark.mllib.classification / [Apache Spark](#)
 - org.apache.spark.mllib.clustering / [Apache Spark](#)
 - org.apache.spark.mllib.linalg / [Apache Spark](#)
 - org.apache.spark.mllib.optimization / [Apache Spark](#)
 - org.apache.spark.mllib.recommendation / [Apache Spark](#)
 - org.apache.spark.mllib.regression / [Apache Spark](#)
 - org.apache.spark.mllib.stat / [Apache Spark](#)
 - org.apache.spark.mllib.tree / [Apache Spark](#)
 - org.apache.spark.mllib.util / [Apache Spark](#)

- MNIST database
 - reference link / [Data collection](#)
- MNIST dataset
 - about / [MNIST dataset](#)
- MOA
 - about / [Machine learning – tools and datasets](#)
- mobile app
 - classifier, plugging into / [Plugging the classifier into a mobile app](#)
- mobile phone
 - data, collecting / [Collecting data from a mobile phone](#)
- mobile phone sensors
 - about / [Mobile phone sensors](#)
 - motion sensors / [Mobile phone sensors](#)
 - environmental sensors / [Mobile phone sensors](#)
 - position sensors / [Mobile phone sensors](#)
 - URL, for Android / [Mobile phone sensors](#)
 - URL, for Windows Phone / [Mobile phone sensors](#)
- model
 - chaining / [Model chaining](#)
 - in production / [Getting models into production](#)
 - maintenance / [Model maintenance](#)
 - building / [Model building](#)
 - linear models / [Linear models](#)
- model assesment
 - about / [Model assessment, evaluation, and comparisons](#), [Model assessment](#)
- model comparison
 - about / [Model assessment, evaluation, and comparisons](#), [Model comparisons](#)
 - algorithms, comparing / [Comparing two algorithms](#)
 - multiple algorithms, comparing / [Comparing multiple algorithms](#)
- model evaluation
 - about / [Model assessment, evaluation, and comparisons](#)
- model evaluation metrics
 - about / [Model evaluation metrics](#), [Model evaluation metrics](#)
 - confusion matrix / [Confusion matrix and related metrics](#)
 - PRC curve / [ROC and PRC curves](#)
 - ROC curve / [ROC and PRC curves](#)
 - Gain charts / [Gain charts and lift curves](#)

- lift curves / [Gain charts and lift curves](#)
 - Confusion Metrics, evaluation / [Evaluation on Confusion Metrics](#)
 - ROC curves / [ROC Curves, Lift Curves, and Gain Charts](#)
 - Lift Curves / [ROC Curves, Lift Curves, and Gain Charts](#)
 - Gain Charts / [ROC Curves, Lift Curves, and Gain Charts](#)
- model evolution, monitoring
 - Kubat / [Widmer and Kubat](#)
 - drift detection method (DDM) / [Drift Detection Method or DDM](#)
 - early drift detection method (EDDM) / [Early Drift Detection Method or EDDM](#)
- model evolution, monitoring
 - about / [Monitoring model evolution](#)
 - Widmer / [Widmer and Kubat](#)
 - early drift detection method (EEDM) / [Early Drift Detection Method or EDDM](#)
- modeling techniques
 - linear algorithm / [Linear algorithms](#)
 - non-linear algorithms / [Non-linear algorithms](#)
 - ensemble algorithms / [Ensemble algorithms](#)
 - partition based algorithm / [Partition based](#)
 - hierarchical clustering / [Hierarchical based and micro clustering](#)
 - micro clustering / [Hierarchical based and micro clustering](#)
 - density based algorithm / [Density based](#)
 - grid based algorithm / [Grid based](#)
- models
 - building / [Building models](#)
 - single layer regression model, building / [Building a single-layer regression model](#)
 - deep belief network, building / [Building a deep belief network](#)
 - Multilayer Convolutional Network, building / [Build a Multilayer Convolutional Network](#)
 - non-linear models / [Non-linear models](#)
 - ensemble learning / [Ensemble learning and meta learners](#)
 - meta learners / [Ensemble learning and meta learners](#)
 - clustering analysis / [Observations and clustering analysis](#)
 - observations / [Observations and clustering analysis](#)
- model validation techniques
 - about / [Model validation techniques](#)

- prequential evaluation / [Prequential evaluation](#)
 - holdout evaluation / [Holdout evaluation](#)
 - controlled permutations / [Controlled permutations](#)
 - evaluation criteria / [Evaluation criteria](#)
 - algorithms, versus metrics / [Comparing algorithms and metrics](#)
- MongoDB
 - about / [Big data application architecture](#)
 - URL / [Big data application architecture](#)
- most probable explanation (MPE) / [MAP queries and marginal MAP queries](#)
- motion sensors
 - about / [Mobile phone sensors](#)
- Mozilla Thunderbird
 - about / [E-mail spam detection](#)
- Multi-layered neural network
 - inputs / [Inputs, neurons, activation function, and mathematical notation](#)
 - neuron / [Inputs, neurons, activation function, and mathematical notation](#)
 - activation function / [Inputs, neurons, activation function, and mathematical notation](#)
 - mathematical notation / [Inputs, neurons, activation function, and mathematical notation](#)
 - about / [Multi-layered neural network](#)
 - structure and mathematical notations / [Structure and mathematical notations](#)
 - activation functions / [Activation functions in NN](#)
 - training / [Training neural network](#)
- multi-layered perceptron (MLP) / [Feature relevance and analysis](#)
- Multi-layer feed-forward neural network
 - about / [Multi-layer feed-forward neural network](#)
- multi-view SSL
 - about / [Co-training SSL or multi-view SSL](#)
- Multidimensional Scaling (MDS)
 - about / [Multidimensional Scaling \(MDS\)](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Multilayer Convolutional Network

- about / [Building models](#)
- building / [Build a Multilayer Convolutional Network](#)
- multinomial distribution / [Random variables, joint, and marginal distributions](#)
- multiple algorithms, comparing
 - ANOVA test / [ANOVA test](#)
 - Friedmans test / [Friedman's test](#)
- multivariate feature analysis
 - about / [Multivariate feature analysis](#)
 - scatter plots / [Multivariate feature analysis](#)
 - ScatterPlot Matrix / [Multivariate feature analysis](#)
 - parallel plots / [Multivariate feature analysis](#)
- multivariate feature selection
 - about / [Multivariate feature selection](#)
 - minimal redundancy maximal relevance (mRMR) / [Minimal redundancy maximal relevance \(mRMR\)](#)
 - correlation-based feature selection (CFS) / [Correlation-based feature selection \(CFS\)](#)
- myrunscollector package
 - Globals.java class / [Loading the data collector](#)
 - CollectorActivity.java class / [Loading the data collector](#)
 - SensorsService.java class / [Loading the data collector](#)

N

- Naive Bayes
 - about / [Underfitting and overfitting, Naïve Bayes](#)
 - algorithm input / [Algorithm input and output](#)
 - algorithm output / [Algorithm input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- Naive Bayes (NB) / [Feature relevance and analysis](#)
- naive Bayes baseline
 - implementing / [Implementing naive Bayes baseline](#)
- named entity recognition / [Information extraction and named entity recognition](#)
- named entity recognition (NER)
 - about / [Named entity recognition](#)
 - Hidden Markov models / [Hidden Markov models for NER](#)
 - Maximum entropy Markov model (MEMM) / [Maximum entropy Markov models for NER](#)
- natural language processing (NLP)
 - about / [NLP, subfields, and tasks, Deep learning and NLP](#)
 - text categorization / [Text categorization](#)
 - part-of-speech tagging (POS tagging) / [Part-of-speech tagging \(POS tagging\)](#)
 - text clustering / [Text clustering](#)
 - information extraction / [Information extraction and named entity recognition](#)
 - named entity recognition / [Information extraction and named entity recognition](#)
 - sentiment analysis / [Sentiment analysis and opinion mining](#)
 - opinion mining / [Sentiment analysis and opinion mining](#)
 - coreference resolution / [Coreference resolution](#)
 - Word sense disambiguation (WSD) / [Word sense disambiguation](#)
 - machine translation (MT) / [Machine translation](#)
 - semantic reasoning / [Semantic reasoning and inferencing](#)
 - inferencing / [Semantic reasoning and inferencing](#)
 - text summarization / [Text summarization](#)
 - question, automating / [Automating question and answers](#)
 - answers, automating / [Automating question and answers](#)

- Nemenyi test / [Comparing algorithms and metrics](#)
- Neo4j
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
 - URL, for licensing / [Machine learning – tools and datasets](#)
- Neo4J / [Graph databases](#)
- neural network
 - about / [Underfitting and overfitting](#)
- neural network, training
 - about / [Training neural network](#)
 - empirical risk minimization / [Empirical risk minimization](#)
 - parameter initialization / [Parameter initialization](#)
 - loss function / [Loss function](#)
 - gradients / [Gradients](#)
 - feed forward and backpropagation / [Feed forward and backpropagation, How does it work?](#)
- neural networks
 - about / [Neural networks](#)
 - perceptron / [Perceptron](#)
 - feedforward neural networks / [Feedforward neural networks](#)
 - autoencoder / [Autoencoder](#)
 - Restricted Boltzman machine / [Restricted Boltzmann machine](#)
 - deep convolutional networks / [Deep convolutional networks](#)
 - limitations / [Limitations of neural networks, Vanishing gradients, local optimum, and slow training](#)
- No Free Lunch Theorem (NFLT) / [Model building](#)
- nominal data
 - about / [Measurement scales](#)
- non-Euclidean distance
 - about / [Non-Euclidean distances](#)
- non-linear algorithms
 - about / [Non-linear algorithms](#)
 - Hoeffding Trees (HT) / [Hoeffding trees or very fast decision trees \(VFDT\)](#)
 - very fast decision trees (VFDT) / [Hoeffding trees or very fast decision trees \(VFDT\)](#)
- non-linear models
 - about / [Non-linear models](#)
 - Decision Trees / [Decision Trees](#)

- K-Nearest Neighbors (KNN) / [K-Nearest Neighbors \(KNN\)](#)
 - support vector machines (SVM) / [Support vector machines \(SVM\)](#)
- non-negative Matrix factorization (NMF)
 - about / [Non-negative matrix factorization \(NMF\)](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitation / [Advantages and limitations](#)
- Non-negative Matrix Factorization (NNMF) / [Clustering techniques](#)
- nonlinear methods
 - about / [Nonlinear methods](#)
 - Kernel Principal Component Analysis (KPCA) / [Kernel Principal Component Analysis \(KPCA\)](#)
 - manifold learning / [Manifold learning](#)
- normalization
 - about / [Outliers](#)
 - Min-Max Normalization / [Outliers](#)
 - Z-Score Normalization / [Outliers](#)
- Normalized mutual information (NMI) / [Normalized mutual information index](#)
- NoSQL
 - about / [NoSQL](#)
 - key-value databases / [Key-value databases](#)
 - document databases / [Document databases](#)
 - columnar databases / [Columnar databases](#)
 - graph databases / [Graph databases](#)
- notations, supervised learning
 - about / [Formal description and notation](#)
 - instance / [Formal description and notation](#)
 - label / [Formal description and notation](#)
 - binary classification / [Formal description and notation](#)
 - regression / [Formal description and notation](#)
 - dataset / [Formal description and notation](#)
- Noun Phrase (NP)
 - about / [Syntactic features](#)

O

- one-class SVM
 - about / [One-class SVM](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- online bagging algorithm
 - about / [Online Bagging algorithm](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- online boosting algorithm
 - about / [Online Boosting algorithm](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- online courses
 - about / [Online courses](#)
- Online k-Means
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- online learning engine
 - about / [Online learning engine](#)
- online linear models, with loss functions
 - inputs / [Inputs and outputs](#)
 - ouputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)

- limitataions / [Advantages and limitations](#)
- Online Naive Bayes
 - about / [Online Naïve Bayes](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- OpenMarkov
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
[/ OpenMarkov](#)
- opinion mining / [Sentiment analysis and opinion mining](#)
- Oracle Database Online Documentation
 - URL / [Dataset](#)
- ordinal data
 - about / [Measurement scales](#)
- OrientDB / [Graph databases](#)
- outlier algorithms
 - about / [Outlier algorithms](#)
 - statistical-based / [Outlier algorithms](#), [Statistical-based](#)
 - distance-based / [Outlier algorithms](#)
 - density-based / [Outlier algorithms](#)
 - clustering-based / [Outlier algorithms](#)
 - high-dimension-based / [Outlier algorithms](#)
 - distance-based methods / [Distance-based methods](#)
 - density-based methods / [Density-based methods](#)
 - clustering-based methods / [Clustering-based methods](#)
 - high-dimensional-based methods / [High-dimensional-based methods](#)
 - one-class SVM / [One-class SVM](#)
- outlier detection
 - about / [Outlier or anomaly detection](#)
 - outlier algorithms / [Outlier algorithms](#)
 - outlier evaluation techniques / [Outlier evaluation techniques](#)
 - used, for unsupervised learning / [Unsupervised learning using outlier detection](#)
 - partition-based clustering / [Partition-based clustering for outlier detection](#)
 - input / [Inputs and outputs](#), [Inputs and outputs](#)

- output / [Inputs and outputs](#), [Inputs and outputs](#)
 - working / [How does it work?](#), [How does it work?](#)
 - advantages / [Advantages and limitations](#), [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#), [Advantages and limitations](#)
 - Exact Storm / [Exact Storm](#)
 - Abstract-C / [Abstract-C](#)
 - Direct Update of Events (DUE) / [Direct Update of Events \(DUE\)](#)
 - Micro Clustering based Algorithm (MCOD) / [Micro Clustering based Algorithm \(MCOD\)](#)
 - Approx Storm / [Approx Storm](#)
 - validation techniques / [Validation and evaluation techniques](#)
 - evaluation techniques / [Validation and evaluation techniques](#)
- outlier evaluation techniques
 - about / [Outlier evaluation techniques](#)
 - supervised evaluation / [Supervised evaluation](#)
 - unsupervised evaluation / [Unsupervised evaluation](#)
 - technique / [Unsupervised evaluation](#)
- outlier models
 - observation / [Observations and analysis](#)
 - analysis / [Observations and analysis](#)
- outliers
 - removing / [Remove outliers](#)
 - handling / [Outliers](#)
 - detecting, in data / [Outliers](#)
 - IQR / [Outliers](#)
 - distance-based methods / [Outliers](#)
 - density-based methods / [Outliers](#)
 - mathematical transformation, of feature / [Outliers](#)
 - handling, robust statistical algorithms used / [Outliers](#)
- Output layer
 - about / [Feedforward neural networks](#)
- overfits
 - about / [Applied machine learning workflow](#)
- overfitting
 - about / [Underfitting and overfitting](#)
- oversampling / [Undersampling and oversampling](#)

P

- p-norm distance
 - about / [Euclidean distances](#)
- Page-Hinckley test / [CUSUM and Page-Hinckley test](#)
- Paired-t test / [Paired-t test](#)
- pairwise-adaptive similarity / [Pairwise-adaptive similarity](#)
- PAPI
 - URL / [Machine learning as a service](#)
- parallel plots / [Multivariate feature analysis](#)
- Parquet / [Columnar databases](#)
- part-of-speech (POS)
 - about / [Working with text data](#)
- part-of-speech tagging (POS tagging) / [Part-of-speech tagging \(POS tagging\)](#)
- partial memory
 - about / [Partial memory](#)
 - full memory / [Full memory](#)
 - detection methods / [Detection methods](#)
 - adaptation methods / [Adaptation methods](#)
- partition based algorithm
 - Online k-Means / [Online k-Means](#)
- pattern analysis
 - about / [Pattern analysis](#)
- Pearson coefficient
 - about / [Content-based filtering](#)
- Pearson correlation coefficient
 - about / [Java machine learning](#)
- perceptron
 - about / [Artificial neural networks, Introducing image recognition, Perceptron](#)
- peristalsis / [Visualization analysis](#)
- Phrase (VP)
 - about / [Syntactic features](#)
- Pipelines
 - reference link / [Random Forest](#)
- plan recognition
 - about / [Plan recognition](#)
- Poisson distribution / [Poisson distribution](#)

- Polynomial Kernel / [How does it work?](#)
- Portable Format for Analytics (PFA) / [Predictive Model Markup Language](#)
- position sensors
 - about / [Mobile phone sensors](#)
- positive definite matrix / [Positive definite matrix](#)
- positive semi-definite matrix / [Positive definite matrix](#)
- PRC curve / [ROC and PRC curves](#)
- Pre-processing phase
 - about / [Building a machine learning application](#)
- precision
 - about / [Precision and recall](#)
- Prediction.IO / [Machine learning as a service](#)
- predictive apriori
 - about / [Weka](#)
- Predictive Model Markup Language (PMML)
 - about / [Predictive Model Markup Language](#)
- Prepositional Phrase (PP)
 - about / [Syntactic features](#)
- Principal Component Analysis (PCA)
 - about / [Histogram-based anomaly detection](#)
 - about / [Embedded approach](#)
- principal component analysis (PCA)
 - about / [Principal component analysis \(PCA\)](#)
 - inputs / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advanatges / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
 - [Dimensionality reduction](#)
- Principal component analysis (PCA)
 - about / [Data reduction](#)
- principal components / [How does it work?](#)
- Principal Components Analysis (PCA)
 - about / [Kernel methods](#)
- probabilistic classifiers
 - about / [Probabilistic classifiers](#)
- probabilistic graphical models (PGM) / [Machine learning – tools and datasets](#)
- probabilistic graph modeling / [Machine learning – types and subtypes](#)

- probabilistic latent semantic analysis (PLSA)
 - about / [Probabilistic latent semantic analysis \(PLSA\)](#)
 - input / [Input and output](#)
 - output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- probabilistic latent semantic index (PLSI) / [Topic modeling](#)
- Probabilistic Principal Component Analysis (PPCA) / [Advantages and limitations](#)
- probability
 - about / [Probability revisited](#)
 - concepts / [Concepts in probability](#)
 - conditional probability / [Conditional probability](#)
 - Chain rule and Bayes' theorem / [Chain rule and Bayes' theorem](#)
 - random variables / [Random variables, joint, and marginal distributions](#)
 - joint / [Random variables, joint, and marginal distributions](#)
 - marginal distributions / [Random variables, joint, and marginal distributions](#)
 - marginal independence / [Marginal independence and conditional independence](#)
 - conditional independence / [Marginal independence and conditional independence](#)
 - factors / [Factors](#)
 - factors, types / [Factor types](#)
 - distribution queries / [Distribution queries](#)
 - probabilistic queries / [Probabilistic queries](#)
 - MAP queries / [MAP queries and marginal MAP queries](#)
 - marginal MAP queries / [MAP queries and marginal MAP queries](#)
- process, machine learning
 - about / [Process](#)
 - business problem, identifying / [Process](#)
 - mapping / [Process](#)
 - data collection / [Process](#)
 - data quality analysis / [Process](#)
 - data sampling / [Process](#)
 - transformation / [Process](#)
 - feature analysis / [Process](#)
 - feature selection / [Process](#)

- modeling / [Process](#)
- model evaluation / [Process](#)
- model selection / [Process](#)
- model deployment / [Process](#)
- model performance, monitoring / [Process](#)
- processors / [SAMOA architecture](#)
- Propagation-based techniques, Bayesian networks
 - about / [Propagation-based techniques](#)
 - belief propagation / [Belief propagation](#)
 - factor graph / [Factor graph](#)
 - factor graph, messaging in / [Messaging in factor graph](#)
 - input and output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages and limitations / [Advantages and limitations](#)
- publish-subscribe frameworks / [Publish-subscribe frameworks](#)

Q

- Query by Committee (QBC)
 - about / [Query by Committee \(QBC\)](#)
- Query by disagreement (QBD)
 - about / [Query by disagreement \(QBD\)](#)

R

- R-Squared / [R-Squared](#)
- Radial Basis Function (RBF) / [Inputs and outputs](#)
- Rand index / [Rand index](#)
- Random Forest / [Random Forest](#)
- Random Forest (RF) / [Feature relevance and analysis](#), [Random Forest](#)
- random projections (RP)
 - about / [Random projections \(RP\)](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- RapidMiner
 - about / [Machine learning – tools and datasets](#), [Case Study – Horse Colic Classification](#)
 - URL / [Machine learning – tools and datasets](#)
 - experiments / [RapidMiner experiments](#)
 - visualization analysis / [Visualization analysis](#)
 - feature selection / [Feature selection](#)
 - model process flow / [Model process flow](#)
 - model evaluation metrics / [Model evaluation metrics](#)
- ratio data
 - about / [Measurement scales](#)
- real-time Big Data Machine Learning
 - about / [Real-time Big Data Machine Learning](#)
 - SAMOA / [SAMOA as a real-time Big Data Machine Learning framework](#)
 - machine learning algorithms / [Machine Learning algorithms](#)
 - tools / [Tools and usage](#)
 - usage / [Tools and usage](#)
 - experiments / [Experiments, results, and analysis](#)
 - results / [Experiments, results, and analysis](#)
 - analysis / [Experiments, results, and analysis](#)
 - results, analysis / [Analysis of results](#)
- real-time stream processing / [Real-time stream processing](#)
- real-world case study
 - about / [Real-world case study](#)

- tools / [Tools and software](#)
 - software / [Tools and software](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine learning mapping](#)
 - data collection / [Data collection](#)
 - data quality analysis / [Data quality analysis](#)
 - data sampling / [Data sampling and transformation](#)
 - data transformation / [Data sampling and transformation](#)
 - feature analysis / [Feature analysis and dimensionality reduction](#)
 - dimensionality reduction / [Feature analysis and dimensionality reduction](#)
 - models, clustering / [Clustering models, results, and evaluation](#)
 - results / [Clustering models, results, and evaluation](#), [Outlier models, results, and evaluation](#)
 - evaluation / [Clustering models, results, and evaluation](#), [Outlier models, results, and evaluation](#)
 - outlier models / [Outlier models, results, and evaluation](#)
- reasoning, Bayesian networks
 - patterns / [Reasoning patterns](#)
 - causal or predictive reasoning / [Causal or predictive reasoning](#)
 - evidential or diagnostic reasoning / [Evidential or diagnostic reasoning](#)
 - intercausal reasoning / [Intercausal reasoning](#)
 - combined reasoning / [Combined reasoning](#)
- recall
 - about / [Precision and recall](#)
- receiver operating characteristics (ROC) / [Machine learning – concepts and terminology](#)
- Receiver Operating Characteristics (ROC)
 - about / [Roc curves](#)
- recommendation engine
 - basic concepts / [Basic concepts](#)
 - key concepts / [Key concepts](#)
 - user-based analysis / [User-based and item-based analysis](#)
 - item-based analysis / [User-based and item-based analysis](#)
 - similarity, calculating / [Approaches to calculate similarity](#)
 - exploitation / [Exploitation versus exploration](#)
 - exploration / [Exploitation versus exploration](#)
 - book-recommendation engine, building / [Building a recommendation engine](#)

- Recurrent neural networks (RNN)
 - about / [Recurrent Neural Networks](#)
 - structure / [Structure of Recurrent Neural Networks](#)
 - learning / [Learning and associated problems in RNNs](#)
 - issues / [Learning and associated problems in RNNs](#)
 - Long short term memory (LSTM) / [Long Short Term Memory](#)
 - Gated Recurrent Units (GRUs) / [Gated Recurrent Units](#)
- regression
 - about / [Regression](#), [Underfitting and overfitting](#), [Regression](#), [Formal description and notation](#)
 - linear regression / [Linear regression](#)
 - evaluating / [Evaluating regression](#)
 - mean squared error / [Mean squared error](#)
 - mean absolute error / [Mean absolute error](#)
 - correlation coefficient / [Correlation coefficient](#)
 - data, loading / [Loading the data](#)
 - attributes, analyzing / [Analyzing attributes](#)
 - model, building / [Building and evaluating regression model](#)
 - model, evaluating / [Building and evaluating regression model](#)
 - tips / [Tips to avoid common regression problems](#)
- regression model
 - evaluating / [Building and evaluating regression model](#)
 - building / [Building and evaluating regression model](#)
 - linear regression / [Linear regression](#)
 - regression trees / [Regression trees](#)
- regression trees
 - about / [Regression trees](#)
- regularization
 - about / [Regularization](#)
 - L2 regularization / [L2 regularization](#)
 - L1 regularization / [L1 regularization](#)
- reinforcement learning
 - about / [What kind of problems can machine learning solve?](#) / [Machine learning – types and subtypes](#)
- representation, Bayesian networks
 - about / [Representation](#)
 - definition / [Definition](#)
- resampling / [Is sampling needed?](#)

- Resilient Distributed Dataset (RDD)
 - about / [Apache Spark](#)
- Resilient Distributed Datasets (RDD) / [Spark architecture](#)
- Restricted Boltzman machine
 - about / [Restricted Boltzmann machine](#)
- restricted Boltzmann machine
 - about / [Artificial neural networks](#)
- restricted Boltzmann machines (RBM)
 - about / [Deeplearning4j](#)
- Restricted Boltzmann Machines (RBM)
 - about / [Restricted Boltzmann Machines](#)
 - definition and mathematical notation / [Definition and mathematical notation](#)
 - Conditional distribution / [Conditional distribution](#)
 - free energy / [Free energy in RBM](#)
 - training / [Training the RBM](#)
 - sampling / [Sampling in RBM](#)
 - contrastive divergence / [Contrastive divergence , How does it work?](#)
 - persistent contrastive divergence / [Persistent contrastive divergence](#)
- ROC curve / [ROC and PRC curves](#)
- Roc curves
 - about / [Roc curves](#)
- roles, machine learning
 - about / [Roles](#)
 - business domain expert / [Roles](#)
 - data engineer / [Roles](#)
 - project manager / [Roles](#)
 - data scientist / [Roles](#)
 - machine learning expert / [Roles](#)
- RuleSetModel / [Predictive Model Markup Language](#)

S

- SAMOA
 - about / [Machine learning – tools and datasets](#), [SAMOA as a real-time Big Data Machine Learning framework](#)
 - URL / [Machine learning – tools and datasets](#)
 - architecture / [SAMOA architecture](#)
- sampling
 - about / [Machine learning – concepts and terminology](#), [Sampling](#)
 - uniform random sampling / [Machine learning – concepts and terminology](#)
 - stratified random sampling / [Machine learning – concepts and terminology](#)
 - cluster sampling / [Machine learning – concepts and terminology](#)
 - systematic sampling / [Machine learning – concepts and terminology](#)
- sampling-based techniques, Bayesian networks
 - about / [Sampling-based techniques](#)
 - forward sampling with rejection / [Forward sampling with rejection](#), [How does it work?](#)
- Samza / [SAMOA as a real-time Big Data Machine Learning framework](#)
- scalar product
 - of vectors / [Scalar product of vectors](#)
- Scale Invariant Feature Transform (SIFT)
 - about / [Introducing image recognition](#)
- ScatterPlot Matrix / [Multivariate feature analysis](#)
- scatter plots / [Multivariate feature analysis](#)
- score function
 - about / [Supervised learning](#)
- self-organizing maps (SOM)
 - about / [Self-organizing maps \(SOM\)](#)
 - inputs / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- self-training SSL
 - about / [Self-training SSL](#)
 - inputs / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)

- advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
 - Query by Committee (QBC) / [Query by Committee \(QBC\)](#)
- semantic features / [Semantic features](#)
- semantic reasoning / [Semantic reasoning and inferencing](#)
- semi-supervised learning / [Machine learning – types and subtypes](#)
- Semi-Supervised Learning (SSL)
 - about / [Semi-supervised learning](#)
 - representation / [Representation, notation, and assumptions](#)
 - notation / [Representation, notation, and assumptions](#)
 - assumptions / [Representation, notation, and assumptions](#)
 - assumptions, to be true / [Representation, notation, and assumptions](#)
 - techniques / [Semi-supervised learning techniques](#)
 - self-training SSL / [Self-training SSL](#)
 - multi-view SSL / [Co-training SSL or multi-view SSL](#)
 - co-training SSL / [Co-training SSL or multi-view SSL](#)
 - label SSL / [Cluster and label SSL](#)
 - cluster SSL / [Cluster and label SSL](#)
 - transductive graph label propagation / [Transductive graph label propagation](#)
 - transductive SVM (TSVM) / [Transductive SVM \(TSVM\)](#)
 - advanatages / [Advantages and limitations](#)
 - disadvanatages / [Advantages and limitations](#)
 - data distribution sampling / [Data distribution sampling](#)
- Semi-Supervised Learning (SSL), case study
 - about / [Case study in semi-supervised learning](#)
 - tools / [Tools and software](#)
 - software / [Tools and software](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine learning mapping](#)
 - data collection / [Data collection](#)
 - data quality, analysis / [Data quality analysis](#)
 - data sampling / [Data sampling and transformation](#)
 - data transformation / [Data sampling and transformation](#)
 - datasets / [Datasets and analysis](#)
 - datasets, analysis / [Datasets and analysis](#)
 - feature analysis, results / [Feature analysis results](#)
 - experiments / [Experiments and results](#)

- results / [Experiments and results](#)
 - analysis / [Analysis of semi-supervised learning](#)
- sentiment analysis / [Sentiment analysis and opinion mining](#)
- sequential data / [Datasets used in machine learning](#)
- shrinking methods
 - embedded approach / [Embedded approach](#)
- Sigmoid function / [Sigmoid function](#)
- Sigmoid Kernel / [How does it work?](#)
- Silhouettes index / [Silhouette's index](#)
- similar items
 - searching / [Find similar items](#)
- similarity calculation
 - about / [Approaches to calculate similarity](#)
 - collaborative filtering / [Collaborative filtering](#)
 - content-based filtering / [Content-based filtering](#)
 - hybrid approach / [Hybrid approach](#)
- similarity measures
 - about / [Similarity measures](#)
 - Euclidean distance / [Euclidean distance](#)
 - Cosine distance / [Cosine distance](#)
 - pairwise-adaptive similarity / [Pairwise-adaptive similarity](#)
 - extended Jaccard Coefficient / [Extended Jaccard coefficient](#)
 - Dice coefficient / [Dice coefficient](#)
- SimRank
 - about / [Non-Euclidean distances](#)
- single layer regression model
 - building / [Building a single-layer regression model](#)
- Singular value decomposition (SVD)
 - about / [Data reduction](#)
- Singular Value Decomposition (SVD) / [Advantages and limitations](#)
- singular value decomposition (SVD) / [Dimensionality reduction, Singular value decomposition \(SVD\)](#)
- sliding windows
 - about / [Sliding windows](#)
- SMILE
 - reference link / [Tools and software](#)
- Smile
 - URL / [Machine learning – tools and datasets](#)

- about / [Machine learning – tools and datasets](#)
- software / [Tools and software](#)
- source-sink frameworks / [Source-sink frameworks](#)
- Spark-MLLib
 - about / [Machine learning – tools and datasets](#)
 - URL / [Machine learning – tools and datasets](#)
- Spark core, components
 - Resilient Distributed Datasets (RDD) / [Spark architecture](#)
 - Lineage graph / [Spark architecture](#)
- Spark MLlib
 - used, as Big Data Machine Learning / [Spark MLlib as Big Data Machine Learning platform](#)
 - architecture / [Spark architecture](#)
 - machine learning / [Machine Learning in MLlib](#)
 - tools / [Tools and usage](#)
 - usage / [Tools and usage](#)
 - experiments / [Experiments, results, and analysis](#)
 - results / [Experiments, results, and analysis](#)
 - analysis / [Experiments, results, and analysis](#)
 - reference link / [Experiments, results, and analysis](#)
 - k-Means / [k-Means](#)
 - k-Means, with PCA / [k-Means with PCA](#)
 - k-Means with PCA, bisecting / [Bisecting k-Means \(with PCA\)](#)
 - Gaussian Mixture Model (GMM) / [Gaussian Mixture Model](#)
 - Random Forest / [Random Forest](#)
 - results, analysis / [Analysis of results](#)
- Spark SQL / [Spark SQL](#)
- Spark Streaming
 - about / [Apache Spark, Real-time Big Data Machine Learning](#)
- sparse coding
 - about / [Sparse coding](#)
- spatio-temporal patterns
 - about / [Transaction analysis](#)
- Spearman's footrule distance
 - about / [Java machine learning](#)
- spectral clustering
 - about / [Spectral clustering](#)
 - input / [Inputs and outputs](#)

- output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- SQL frameworks / [SQL frameworks](#)
- stacked autoencoders
 - about / [Autoencoder](#)
- standard deviation / [Standard deviation](#)
- standardization
 - about / [Document collection and standardization](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
- standards and markup languages
 - about / [Standards and markup languages](#)
- Statistical-based
 - about / [Statistical-based](#)
 - input / [Inputs and outputs](#)
 - outputs / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Statistics 110 (Harvard) by Joe Blitzstein
 - URL / [Online courses](#)
- stemming / [Stemming or lemmatization](#)
- step execution mode / [Amazon Elastic MapReduce](#)
- Stochastic Gradient Descent (SGD)
 - about / [How does it work?](#)
 - / [Supervised learning experiments](#)
- stop words removal
 - about / [Stop words removal](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
- stratification
 - about / [Stratification](#)
- stratified sampling / [Stratified sampling](#)
- stream / [SAMOA architecture](#)

- stream computational technique
 - about / [Basic stream processing and computational techniques](#), [Stream computations](#)
 - frequency count / [Stream computations](#)
 - point queries / [Stream computations](#)
 - distinct count / [Stream computations](#)
 - mean / [Stream computations](#)
 - standard deviation / [Stream computations](#)
 - correlation coefficient / [Stream computations](#)
 - sliding windows / [Sliding windows](#)
 - sampling / [Sampling](#)
- stream learning / [Machine learning – types and subtypes](#)
- stream learning, case study
 - about / [Case study in stream learning](#)
 - tools / [Tools and software](#)
 - software / [Tools and software](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine learning mapping](#)
 - data collection / [Data collection](#)
 - data sampling / [Data sampling and transformation](#)
 - data transformation / [Data sampling and transformation](#)
 - feature analysis / [Feature analysis and dimensionality reduction](#)
 - dimensionality reduction / [Feature analysis and dimensionality reduction](#)
 - models / [Models, results, and evaluation](#)
 - results / [Models, results, and evaluation](#)
 - evaluation / [Models, results, and evaluation](#)
 - supervised learning experiments / [Supervised learning experiments](#)
 - concept drift experiments / [Concept drift experiments](#)
 - clustering experiments / [Clustering experiments](#)
 - outlier detection experiments / [Outlier detection experiments](#)
 - results, analysis / [Analysis of stream learning results](#)
- Stream Processing Engines (SPE) / [Real-time stream processing](#)
- stream processing technique
 - about / [Basic stream processing and computational techniques](#)
- structured data
 - sequential data / [Datasets used in machine learning](#)
- Structure Score Measure / [Measures to evaluate structures](#)
- subfields

- about / [NLP, subfields, and tasks](#)
- Subspace Outlier Detection (SOD) / [How does it work?](#)
- Sum of Squared Errors (SSE) / [Clustering models, results, and evaluation, Experiments, results, and analysis](#)
- sum transfer function
 - about / [Perceptron](#)
- supermarket dataset
 - about / [The supermarket dataset](#)
 - shopping patterns, discovering / [Discover patterns](#)
 - shopping patterns, discovering with Apriori algorithm / [Apriori](#)
 - shopping patterns, discovering with FP-growth algorithm / [FP-growth](#)
- supervised learning / [Machine learning – types and subtypes](#)
 - about / [What kind of problems can machine learning solve?, Supervised learning](#)
 - classification / [Classification](#)
 - regression / [Regression](#)
 - experiments / [Supervised learning experiments](#)
 - Weka, experiments / [Weka experiments](#)
 - RapidMiner, experiments / [RapidMiner experiments](#)
 - reference link / [Results, observations, and analysis](#)
 - and unsupervised learning, common issues / [Issues in common with supervised learning](#)
 - assumptions / [Assumptions and mathematical notations](#)
 - mathematical notations / [Assumptions and mathematical notations](#)
- Support Vector Machine (SVM) model / [Predictive Model Markup Language](#)
- Support Vector Machines (SVM)
 - about / [Kernel methods](#)
 - / [How does it work?](#)
- support vector machines (SVM)
 - about / [Support vector machines \(SVM\)](#)
 - algorithm input / [Algorithm inputs and outputs](#)
 - algorithm output / [Algorithm inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- survivorship bias
 - about / [Sampling traps](#)
- suspicious behaviour detection

- about / [Suspicious and anomalous behavior detection](#)
- suspicious pattern detection
 - about / [Suspicious pattern detection](#)
- suspicious patterns, modelling
 - about / [Modeling suspicious patterns](#)
 - vanilla approach / [Vanilla approach](#)
 - dataset rebalancing / [Dataset rebalancing](#)
- SVM
 - about / [Underfitting and overfitting](#)
- Syntactic features
 - about / [Syntactic features](#)
- Syntactic Language Models (SLM)
 - about / [Syntactic features](#)
- Synthetic Minority Oversampling Technique (SMOTE) / [Undersampling and oversampling](#)
- Sample, Explore, Modify, Model, and Assess (SEMMA).
 - about / [SEMMA methodology](#)

T

- target variables
 - churn probability / [Challenge](#)
 - appetency probability / [Challenge](#)
 - upselling probability / [Challenge](#)
- tasks
 - about / [NLP, subfields, and tasks](#)
- term frequency (TF) / [Frequency-based techniques](#)
- Term Frequency (TF) / [Term frequency \(TF\)](#)
- term frequency-inverse document frequency (TF-IDF) / [Term frequency-inverse document frequency \(TF-IDF\)](#)
- Tertius
 - about / [Weka](#)
- test set
 - about / [Train and test sets](#)
- text categorization
 - about / [Text categorization](#)
- text classification
 - about / [Text classification](#)
 - examples / [Text classification](#)
- text clustering / [Text clustering](#)
 - about / [Text clustering](#)
 - feature transformation / [Feature transformation, selection, and reduction](#)
 - selection / [Feature transformation, selection, and reduction](#)
 - reduction / [Feature transformation, selection, and reduction](#)
 - techniques / [Clustering techniques](#)
 - evaluation / [Evaluation of text clustering](#)
- text data
 - extracting / [Working with text data](#)
 - importing / [Importing data](#)
 - importing, from directory / [Importing from directory](#)
 - importing, from file / [Importing from file](#)
 - pre-processing / [Pre-processing text data](#)
- text mining
 - about / [Introducing text mining](#)
 - topic modeling / [Topic modeling](#), [Topic modeling](#)
 - text classification / [Text classification](#)

- topics / [Topics in text mining](#)
 - categorization/classification / [Text categorization/classification](#)
 - clustering / [Text clustering](#)
 - named entity recognition (NER) / [Named entity recognition](#)
 - Deep Learning / [Deep learning and NLP](#)
 - NLP / [Deep learning and NLP](#)
- text processing components
 - about / [Text processing components and transformations](#)
 - document collection / [Document collection and standardization](#)
 - standardization / [Document collection and standardization](#)
 - tokenization / [Tokenization](#)
 - stop words removal / [Stop words removal](#)
 - lemmatization / [Stemming or lemmatization](#)
 - local-global dictionary / [Local/global dictionary or vocabulary?](#)
 - vocabulary / [Local/global dictionary or vocabulary?](#)
 - feature extraction/generation / [Feature extraction/generation](#)
 - feature representation / [Feature representation and similarity](#)
 - similarity / [Feature representation and similarity](#)
 - feature selection / [Feature selection and dimensionality reduction](#)
 - dimensionality reduction / [Feature selection and dimensionality reduction](#)
- text summarization / [Text summarization](#)
- time-series forecasting / [Machine learning – types and subtypes](#)
- time series data
 - anomaly detection / [Anomaly detection in time series data](#)
- tokenization
 - about / [Tokenization](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
- tools / [Tools and software](#)
 - about / [Tools and usage](#)
 - Mallet / [Mallet](#)
 - KNIME / [KNIME](#)
- tools, machine learning
 - RapidMiner / [Machine learning – tools and datasets](#)
 - Weka / [Machine learning – tools and datasets](#)
 - Knime / [Machine learning – tools and datasets](#)
 - Mallet / [Machine learning – tools and datasets](#)

- Elki / [Machine learning – tools and datasets](#)
 - JCLAL / [Machine learning – tools and datasets](#)
 - KEEL / [Machine learning – tools and datasets](#)
 - DeepLearning4J / [Machine learning – tools and datasets](#)
 - Spark-MLLib / [Machine learning – tools and datasets](#)
 - H2O / [Machine learning – tools and datasets](#)
 - MOA/SAMOA / [Machine learning – tools and datasets](#)
 - Neo4j / [Machine learning – tools and datasets](#)
 - GraphX / [Machine learning – tools and datasets](#)
 - OpenMarkov / [Machine learning – tools and datasets](#)
 - Smile / [Machine learning – tools and datasets](#)
- topic modeling
 - about / [Topic modeling, Topic modeling](#)
 - probabilistic latent semantic analysis (PLSA) / [Probabilistic latent semantic analysis \(PLSA\)](#)
 - with mallet / [Topic modeling with mallet](#)
 - business problem / [Business problem](#)
 - machine learning, mapping / [Machine Learning mapping](#)
 - data collection / [Data collection](#)
 - data sampling / [Data sampling and transformation](#)
 - transformation / [Data sampling and transformation](#)
 - feature analysis / [Feature analysis and dimensionality reduction](#)
 - dimensionality reduction / [Feature analysis and dimensionality reduction](#)
 - models / [Models, results, and evaluation](#)
 - results / [Models, results, and evaluation](#)
 - evaluation / [Models, results, and evaluation](#)
 - text processing results, analysis / [Analysis of text processing results](#)
- topic modelling, for BBC news
 - about / [Topic modeling for BBC news](#)
 - BBC dataset, collecting / [BBC dataset](#)
 - modeling / [Modeling](#)
 - model, evaluating / [Evaluating a model](#)
 - model, reusing / [Reusing a model](#)
 - model, saving / [Saving a model](#)
 - model, restoring / [Restoring a model](#)
- traditional machine learning
 - architecture / [Traditional machine learning architecture](#)
- training data

- collecting / [Collecting training data](#)
- Training data
 - about / [Building a machine learning application](#)
- training phases
 - competitive phase / [How does it work?](#)
 - cooperation phase / [How does it work?](#)
 - adaptive phase / [How does it work?](#)
- train set
 - about / [Train and test sets](#)
- transaction analysis
 - about / [Transaction analysis](#)
- transaction data / [Datasets used in machine learning](#)
- transductive graph label propagation
 - about / [Transductive graph label propagation](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- transductive SVM (TSVM)
 - about / [Transductive SVM \(TSVM\)](#)
 - output / [Inputs and outputs](#)
 - input / [Inputs and outputs](#)
 - working / [How does it work?](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- transformations
 - about / [Text processing components and transformations](#)
- Tree augmented network (TAN)
 - about / [Tree augmented network](#)
 - input and output / [Input and output](#)
 - working / [How does it work?](#)
 - advantages and limitations / [Advantages and limitations](#)
- TreeModel / [Predictive Model Markup Language](#)
- Tunedit
 - about / [Datasets](#)
 - URL / [Datasets](#)

U

- UCI machine learning repository
 - URL / [Datasets](#)
- UCI repository
 - reference link / [Data Collection](#)
- UC Irvine (UCI) database
 - about / [Datasets](#)
 - URL / [Datasets](#)
- Udemy
 - URL / [Online courses](#)
- uncertainty sampling
 - about / [Uncertainty sampling](#)
 - working / [How does it work?](#)
 - least confident sampling / [Least confident sampling](#)
 - smallest margin sampling / [Smallest margin sampling](#)
 - label entropy sampling / [Label entropy sampling](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- underfits
 - about / [Applied machine learning workflow](#)
- underfitting
 - about / [Underfitting and overfitting](#)
- undersampling / [Undersampling and oversampling](#)
- univariate feature analysis
 - about / [Univariate feature analysis](#)
 - categorical features / [Categorical features](#)
 - continuous features / [Continuous features](#)
- univariate feature selection
 - information theoretic approach / [Information theoretic approach](#)
 - statistical approach / [Statistical approach](#)
- Universal PMML Plug-in (UPPI) / [Predictive Model Markup Language](#)
- unknown-unknowns
 - about / [Unknown-unknowns](#)
- unnormalized measure / [Factor types](#)
- unstructured data / [Datasets used in machine learning](#)
 - mining, issues / [Issues with mining unstructured data](#)
- unsupervised learning / [Machine learning – types and subtypes](#)

- about / [What kind of problems can machine learning solve?](#), [Unsupervised learning](#)
- similar items, searching / [Find similar items](#)
- clustering / [Clustering](#)
- specific issues / [Issues specific to unsupervised learning](#)
- assumptions / [Assumptions and mathematical notations](#)
- mathematical notations / [Assumptions and mathematical notations](#)
- outlier detection, used / [Unsupervised learning using outlier detection](#)
- usage / [Tools and usage](#)
- user-based analysis
 - about / [User-based and item-based analysis](#)
- user-based collaborative filtering
 - about / [User-based filtering](#)
- US Forest Service (USFS) / [Data collection](#)
- US Geological Survey (USGS) / [Data collection](#)

V

- V-Measure
 - about / [V-Measure](#)
 - Homogeneity / [V-Measure](#)
 - Completeness / [V-Measure](#)
- validation
 - techniques / [Training, validation, and test set](#)
- vanilla approach
 - about / [Vanilla approach](#)
- Variable elimination (VE) algorithm / [Variable elimination algorithm](#)
- variance / [Variance](#)
- vector
 - about / [Vector](#)
 - scalar product / [Scalar product of vectors](#)
- vector space model (VSM)
 - about / [Vector space model](#)
 - binary / [Binary](#)
 - Term Frequency (TF) / [Term frequency \(TF\)](#)
 - inverse document frequency (IDF) / [Inverse document frequency \(IDF\)](#)
 - term frequency-inverse document frequency (TF-IDF) / [Term frequency-inverse document frequency \(TF-IDF\)](#)
- version space sampling
 - about / [Version space sampling](#)
 - Query by disagreement (QBD) / [Query by disagreement \(QBD\)](#)
- very fast decision trees (VFDT) / [Hoeffding trees or very fast decision trees \(VFDT\)](#)
 - output / [Inputs and outputs](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Very Fast K-means Algorithm (VFKM) / [Advantages and limitations](#)
- visualization analysis
 - about / [Visualization analysis](#)
 - univariate feature analysis / [Univariate feature analysis](#)
 - multivariate feature analysis / [Multivariate feature analysis](#)
- Vote Entropy
 - disadvantages / [How does it work?](#)

W

- Waikato Environment for Knowledge Analysis (Weka)
 - about / [Weka](#)
 - URL / [Weka](#)
- web resources and competitions
 - about / [Web resources and competitions](#), [Competitions](#)
 - datasets / [Datasets](#)
 - online courses / [Online courses](#)
 - websites and blogs / [Websites and blogs](#)
 - venues and conferences / [Venues and conferences](#)
- website traffic
 - anomaly detection / [Anomaly detection in website traffic](#)
- weighted linear sum (WLS) / [How does it work?](#)
- weighted linear sum of squares (WSS) / [How does it work?](#)
- weighted majority algorithm (WMA)
 - about / [Weighted majority algorithm](#)
 - input / [Inputs and outputs](#)
 - output / [Inputs and outputs](#)
 - working / [Advantages and limitations](#)
 - advantages / [Advantages and limitations](#)
 - limitations / [Advantages and limitations](#)
- Weka
 - URL / [Machine learning – tools and datasets](#)
 - about / [Machine learning – tools and datasets](#), [Case Study – Horse Colic Classification](#)
 - experiments / [Weka experiments](#)
 - Sample end-to-end process, in Java / [Sample end-to-end process in Java](#)
 - experimenter / [Weka experimenter and model selection](#)
 - model selection / [Weka experimenter and model selection](#)
- weka.classifiers package
 - weka.classifiers.bayes / [Weka](#)
 - weka.classifiers.evaluation / [Weka](#)
 - weka.classifiers.functions / [Weka](#)
 - weka.classifiers.lazy / [Weka](#)
 - weka.classifiers.meta / [Weka](#)
 - weka.classifiers.mi / [Weka](#)
 - weka.classifiers.rules / [Weka](#)

- weka.classifiers.trees / [Weka](#)
- Weka 3.6
 - URL / [Before you start](#)
 - downloading / [Before you start](#)
- Weka Bayesian Network GUI / [Weka Bayesian Network GUI](#)
- WEKA Packages
 - URL / [Before we start](#)
- Welch's test / [Welch's t test](#)
- Widmer / [Widmer and Kubat](#)
- Wilcoxon signed-rank test / [Wilcoxon signed-rank test](#)
- word2vec
 - about / [Working with text data](#)
 - URL / [Working with text data](#)
- Word sense disambiguation (WSD) / [Word sense disambiguation](#)
- workflow, Applied Machine Learning
 - data and problem definition / [Applied machine learning workflow](#)
 - data collection / [Applied machine learning workflow](#)
 - data preprocessing / [Applied machine learning workflow](#)
 - data analysis and modeling / [Applied machine learning workflow](#)
 - evaluation / [Applied machine learning workflow](#)
- wrapper approach / [Wrapper approach](#)

X

- Xiaming Chen
 - URL / [Datasets](#)

Y

- Yahoo traffic dataset
 - URL / [Dataset](#)

Z

- Z-Score Normalization / [Outliers](#)
- ZeroMQ Message Transfer Protocol (ZMTP) / [Message queueing frameworks](#)