INTRODUCTION TO MACHINE LEARNING

Dr. Kamakshaiah Musunuru

About the Author

Dr. M. Kamakshaiah is a open source software evangelist, enterprise solutions architect and academic of data science and analytics. His teaching interests are IT practices in business, MIS, Data Science, Business Analytics including functional analytics related to marketing, finance, HRM, quality and operations. He also teaches theoretical concepts like multivariate analytics, numerical simulations & optimization, machine learning & AI using few programming languages like R, Python, Java. Internet of Things and big data analytics (Hadoop) are his all time favorites for teaching.

He has taught abroad for two years and credits two country visits. He has developed few free and open source software solutions meant for corporate practitioners, academics, scholars and students engaging in data science and analytics. All his software applications are available from his Github portal `https://github.com/Kamakshaiah`.

Foreword

I started writing this book for one simple reason that I got to teach machine learning for business analytics students. Actually I started preserving every little code, chunk by chunk, as and when I struck with this subject. Though I thought it as notes at first but the stuff that I gathered became rather more significant to compile it as a book.

I turned into data analyst due to my being accidentally exposed to open source software when I was in Ethiopia. I got to use GNU/Linux to keep my personal computer away from virus infection at the university. I crashed Windows a couple of times when I was working in office at the University, eventually I found GNU/Linux as a wonderful solution to address virus issues. My first experience of GNU/Linux, perhaps, is *Jaunty Jackalope*, as I suppose. This must be Ubuntu 9.04, an LTS version, if I am not wrong. However, my involvement in GNU/Linux became a serious affair though *Karmic Koala.* [1] I still remember few of my colleagues used to visit my home for OS installation in those days of stay at Ethiopia.

My first encounter with analytics software is R. R is *lingua franca* of statistics. I taught "business statistics" several times during

[1] *Karmic Koala* is Ubuntu 9.10. Visit `https://wiki.ubuntu.com/Releases` for more information on Ubuntu releases.

my stint as academic. Once, I notice this little yet uppercase R letter over there while I was browsing for statistical software in Ubuntu *package manager*. R changed not only my understanding of Statistics but the very way of learning the same. I addicted to R so much so that for every pretty little calculations I used to refer R manuals. I must be the first academic to introduce R in south Indian business management curriculum, yet remained unknown to others. Today, R is one of the best tools for statistical analysis and practice of quantitative techniques.

I came across Python just as in same way as I discovered R in Ubuntu. Python was one of the default programming languages for many things in Ubuntu. At first I did not know the power of Python for everything I learned it was only by self study. All my learning is from online resources. However, I could produce thousands of students majored in data analysts through my formal classes. All that I learned and taught is only by my self study, I mean, through very informal personal learning.

I am writing all this not to show myself as a valiant learner, but to demonstrate how can a novice and a naive enthusiast like me can learn programming tools like R and Python. Today, I am offering a couple of courses to teach Hadoop and IoT, that is all by passion. So, I would like to give confidence to the reader that you don't need any formal learning in computer science or IT to learn data science and adopt it as a profession. However, you need tons and tons of patience and passion.

This book is meant for beginners of machine learning and practice. It has 5 chapters each section represents a unique concept of data analytics. This book may be useful for both practitioners and academics to acquire knowledge of machine learning accompanied with Python practice. The first chapter *introduction*, deals with very short information related to basics of machine learning. Chapter I has information related to installation of Python in Linux, Windows and few other OSes. Chapter 2 deals with *Supervised learning* and this chapter has information related to various supervised machine learning algorithms such as ... Chapter 3, *Unsuptervised learning* deals with algorithms such as ... Chapter 4, *Deep lerning*, deals with different types of techniques related to ...

As far as coding is concerned; those code sections where there exist left-bar such as the below

```
statement 1
statement 2
statement 3
```

represents a ***script***. A script is a plain code file in which there exists program statements. There are other code sections where each statement is preceded by python prompt (`>>>`). These code sections are meant for testing. These sections are useful either to evaluate a Python statement or provide evidence for logic.

All the code chunks used in this book are provided though my github portal with a project name *PfDSaA*. Feel free to visit the site `https://github.com/Kamakshaiah/PfDSaA` and download required *.py* files for practice.

One last note is that this book is meant for both data scientists and analysts. The learner need to have basics of Python. However, little logical thinking together with passion and patience are required. The book is written in such a way that even a person having no knowledge on Python can pick up and become maverick at the end of reading. This book covers from very basic details ranging from installation to creating packages. I am not covering very advanced concepts such as software and applications development due to one reason to keep this book reasonable for both beginners and advanced users.

Happy reading ⋯

Author
Dr. M. Kamakshaiah

Contents

Chapter 1

Introduction

Machine learning (ML) is a field of inquiry devoted to understanding and building methods that "learn", that is, methods that leverage data to improve performance on some set of tasks. [1] It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. [2] Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain. [3] In its application across business problems, machine learning is also referred to as predictive analytics.

Machine learning programs can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks. For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step. [4]

The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid. This can then be used as training data for the computer to improve the algorithm(s) it uses to determine correct answers. For example, to train a system for the task of digital character recognition, the MNIST dataset of handwritten digits has often been used.

1.0.1 History and relationships to other fields

The term machine learning was coined in 1959 by *Arthur Samuel*, an IBM employee and pioneer in the field of computer gaming and artificial intelligence. [5] [6] Also the synonym self-teaching computers were used in this time period. [7]

By the early 1960s an experimental "learning machine" with punched tape memory, called *Cybertron*, had been developed by *Raytheon Company* to analyze sonar signals, electrocardiograms and speech patterns using rudimentary reinforcement learning.

It was repetitively trained by a human operator/teacher to recognize patterns and equipped with a "goof" button to cause it to re-evaluate incorrect decisions. A representative book on research into machine learning during the 1960s was *Nilsson's* book on Learning Machines, dealing mostly with machine learning for pattern classification. Interest related to pattern recognition continued into the 1970s, as described by Duda and Hart in 1973. [8] In 1981 a report was given on using teaching strategies so that a neural network learns to recognize 40 characters (26 letters, 10 digits, and 4 special symbols) from a computer terminal.

Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." [9] This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?".

Modern day machine learning has two objectives, one is to classify data based on models which have been developed, the other purpose is to make predictions for future outcomes based on these models. A hypothetical algorithm specific to classifying data may use computer vision of moles coupled with supervised learning in order to train it to classify the cancerous moles. A machine learning algorithm for stock trading may inform the trader of future potential predictions.

1.0.2 Artificial intelligence

As a scientific endeavor, machine learning grew out of the quest for artificial intelligence. In the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what was then termed "neural networks"; these were mostly *perceptrons* and other models that were later found to be reinventions of the generalized linear models of statistics.[10] Probabilistic reasoning was also employed, especially in automated medical diagnosis. [11]

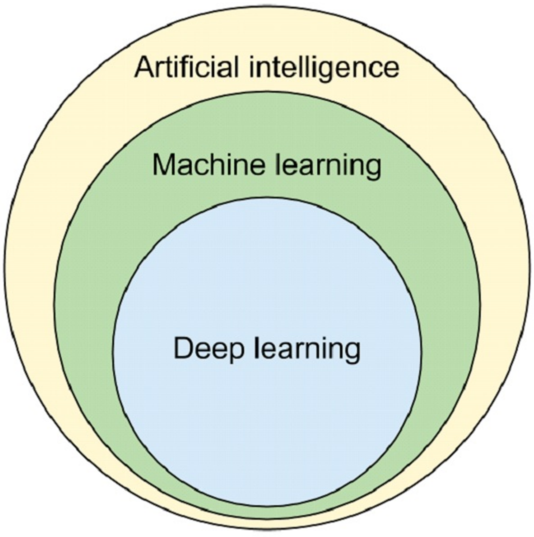

Figure 1.1: AI vs ML

However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation.? By 1980, expert systems had come to dominate AI, and statistics was out of favor. [12] Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information retrieval. Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as “connectionism”, by researchers from other disciplines including *Hopfield, Rumelhart* and *Hinton.* Their main success came in the mid-1980s with the reinvention of *backpropagation.*

Machine learning (ML), reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics, fuzzy logic, and probability theory.

The difference between ML and AI is frequently misunderstood. ML learns and predicts based on passive observations, whereas AI implies an agent interacting with the environment to learn and take actions that maximize its chance of successfully achieving its goals. [13]

As of 2020, many sources continue to assert that ML remains a subfield of AI. Others have the view that not all ML is part of AI, but only an “intelligent subset” of ML should be considered AI.

1.0.3 Data mining

Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as “unsupervised learning” or as a *preprocessing* step to improve learner accuracy. Much of the confusion between these two research communities comes from the basic assumptions they work with. *In machine learning, performance is usually evaluated with respect to the ability to reproduce known knowledge, while in knowledge discovery and data mining (KDD) the key task is the discovery of previously unknown knowledge.* Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by other supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

1.0.4 Optimization

Machine learning also has intimate ties to optimization. Many learning problems are formulated as minimization of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances. For example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the preassigned labels of a set of examples.

1.0.5 Generalization

The difference between optimization and machine learning arises from the goal of generalization. While optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples. Characterizing the generalization of various learning algorithms is an active topic of current research, especially for deep learning algorithms.

1.0.6 Statistics

Machine learning and statistics are closely related fields in terms of methods, but distinct in their principal goal. Statistics draws population inferences from a sample, while machine learning finds generalizable predictive patterns. [14] According to *Michael*

I. Jordan, the ideas of machine learning, from methodological principles to theoretical tools, have had a long pre-history in statistics.[15] He also suggested the term data science as a placeholder to call the overall field. *Leo Breiman* distinguished two statistical modeling paradigms. Data model and algorithmic model, wherein "algorithmic model" means more or less the machine learning algorithms like Random forest. Some statisticians have adopted methods from machine learning, leading to a combined field that they call statistical learning.[16]

1.1 Categories of Machine Learning

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

1. *Supervised learning*: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.

2. *Unsupervised learning*: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

3. *Reinforcement learning*: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize.

1.1.1 Supervised learning

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs.[17] The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs.[18] An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.[19]

Types of supervised-learning algorithms include active learning, classification and regression.[20] Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. As an example, for a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email.

Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

1.1.2 Unsupervised learning

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics, such as finding the probability density function. [21] Though unsupervised learning encompasses other domains involving summarizing and explaining data features.

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated, for example, by internal compactness, or the similarity between members of the same cluster, and separation, the difference between clusters. Other methods are based on estimated density and graph connectivity.

1.1.3 Semi-supervised learning

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy. In weakly supervised learning, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.

1.1.4 Reinforcement learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In machine learning, the environment is typically represented as a Markov decision process (MDP). Many reinforcement learning algorithms use dynamic programming techniques.[22] Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent.

1.2 Machine learning process

Machine learning is the process of imparting intelligence to machines seems daunting and impossible. But it is actually really easy. It can be broken down into 7 major steps.

1.2.1 Collecting Data

Machines initially learn from the data that you give them. It is of the utmost importance to collect reliable data so that machine learning model can find the correct patterns. The quality of the data that was fed in to the machine will determine how accurate the model is. Incorrect or outdated data, gives rise to wrong outcomes or predictions which are not relevant.

It is always better to procure data from a reliable source, as it will directly affect the outcome of the model. Good data is relevant, contains very few missing and repeated values, and has a good representation of the various subcategories/classes present. Given the problem that needs to be solved, then investigation to obtain data need to be done. The quality and quantity of information is very important since it will directly impact how well or badly the model will work. At times data may be available from existing database or it needs to be created from scratch. If it is a small project you can create a spreadsheet that will later be easily exported as a CSV file. It is also common to use the web scraping technique to automatically collect information from various sources such as APIs.

1.2.2 Preparing the Data

Once data is collected visualizing data becomes a priority. Correlations between the different characteristics need to be assessed. It will be necessary to make a selection of characteristics since these characteristics will impact the execution times and the results. PCA is useful for reducing dimensions if necessary. Additionally, balance the amount of data for each class so that it is significant as the learning may be biased towards a type of response and when your model tries to generalize knowledge it will fail. You must also separate the data into two groups: one for *training* and the other for *model evaluation* which can be divided approximately in a ratio of 80/20 but it can vary depending on the case and the volume of data. At this stage, data needs to be processed through normalization, eliminating duplicates, and making error corrections.

1.2.3 Choose the model

A machine learning model determines the output you get after running a machine learning algorithm on the collected data. It is important to choose a model which is relevant to the task at hand. Over the years, scientists and engineers developed various models suited for different tasks like speech recognition, image recognition, prediction, etc. Apart from this, model needs to be chosen based on type of data i.e. categorical & non-categorical, numerical & non-numerical etc.

There are several models available for machine learning practice. Mostly they are algorithms available for different methods such as classification, prediction, linear regression, clustering, K-Nearest Neighbor, Deep Learning, Neural Networks, and more. There are various models to be used depending on the data that is going to be processed such as images, sound, text, and numerical values. In the following table, there are few models and their applications that can be applied in projects.

Model	Applications
Logistic Regression	Price prediction
Fully connected networks	Classification
Convolutional Neural Networks	Image processing
Recurrent Neural Networks	Voice recognition
Random Forest	Fraud Detection
Reinforcement Learning	Learning by trial and error
Generative Models	Image creation
K-means	Segmentation
k-Nearest Neighbors	Recommendation systems
Bayesian Classifiers	Spam and noise filtering

Table 1.1: ML Algorithms and applications

1.2.4 Training the Model

Training is the most important step in machine learning. In training, data will be passed to the machine learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.

Data need to be trained smoothly to see an incremental improvement in the prediction rate. Remember to initialize the weights of your model randomly. The weights are the values that multiply or affect the relationships between the inputs and outputs. These weights are automatically adjusted by the selected algorithm while getting trained.

1.2.5 Evaluating the Model

After training the model, the model need to be tested for the performance. This is done by testing the performance of the model on previously unseen data. The unseen data used is the testing set that was split earlier. If testing was done on the same data which is used for training, that will not get an accurate measure, as the model is already used to the data, and finds the same patterns in it, as it previously did. This will give you disproportionately high accuracy. When used on testing data, it is possible to get accurate measure of how the model will perform and its speed. If the accuracy is less than or equal to 50%, that model will not be useful since it would be like tossing a coin to make decisions. If the precision is 90% or more, it means the model is a best one to rely upon.

1.2.6 Parameter Tuning

Once the model was created and evaluated, it should be verfied that whether the accuracy can be improved or not. This is done by tuning the parameters present in your model. Parameters are the variables in the model that the programmer generally decides. The accuracy can be maximum at a particular value of the parameter. Parameter tuning refers to finding these values.

At times a model can be overfitting or underfitting if the evaluation did not obtain good predictions and precision is not the minimum desired. It is possible to increase the number of training times as in *epochs*. Another important parameter is the one known as the “learning rate”, which is usually a value that multiplies the gradient to gradually bring it closer to the global or local minimum to minimize the cost of the function.

Increasing your values by 0.1 units from 0.001 is not the same as this can significantly affect the model execution time. Indicate the maximum error allowed for the model. At times it can take a few minutes to hours, and even days, to train teh machine. These parameters are often called *Hyperparameters*. This "tuning" is still more of an art than a science and will improve as you experiment. There are usually many parameters to adjust and when combined they can trigger all the options. Each algorithm has its own parameters to adjust. To name a few more, in Artificial Neural Networks (ANNs) architecture the number of hidden layers need to be decided and gradually to be tested with how many neurons each layer can accommodate. This will be a work of great effort and patience to give good results.

1.2.7 Making Predictions

In the end, the model will be used on the data to make predictions accurately. Usually, predictions are not a matter of concern, but accuracies will be calculated based on *confusion matrix*. A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa. The name stems from the fact that whether the system is confusing two classes (i.e. commonly mislabeling one as another).

1.3 Train, validate & test data

In machine learning, a common task is the study and construction of algorithms that can learn from and make predictions on data.[23] Such algorithms function by making data-driven predictions or decisions, through building a mathematical model from input data. These input data used to build the model are usually divided in multiple data sets. In particular, three data sets are commonly used in different stages of the creation of the model: *training, validation* and *test* sets.

The model is initially fit on a training data set, which is a set of examples used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model.[24] The model (e.g. a naive Bayes classifier) is trained on the training data set using a supervised learning method, for example using optimization methods such as gradient descent or stochastic gradient descent. In practice, the training data set often consists of pairs of an input vector (or scalar) and the corresponding output vector (or scalar), where the answer key is commonly denoted as the target (or label). The current model is run with the training data set and produces a result, which is then compared with the target, for each input vector in the training data set. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.

Successively, the fitted model is used to predict the responses for the observations in a second data set called the validation data set. The validation data set provides an unbiased evaluation of a model fit on the training data set while tuning the model's hyperparameters (e.g. the number of hidden units layers and layer widths in a neural network). Validation datasets can be used for regularization by early stopping (stopping training when the error on the validation data set increases, as this is a sign of over-fitting to the training data set). This simple procedure is complicated in practice by the fact that the validation dataset's error may fluctuate during training, producing multiple local minima. This complication has led to the creation of many *ad hoc* rules for deciding when over-fitting has truly begun.

Finally, the test data set is a data set used to provide an unbiased evaluation of a final model fit on the training data set. If the data in the test data set has never been used in training (for example in cross-validation), the test data set is also called a holdout data set. The term "validation set" is sometimes used instead of "test set" in some literature (e.g., if the original data set was partitioned into only two subsets, the test set might be referred to as the validation set). Deciding the sizes and strategies for data set division in training, test and validation sets is very dependent on the problem and data available.

1.3.1 Training data set

A training data set is a data set of examples used during the learning process and is used to fit the parameters (e.g., weights) of, for example, a classifier.[25] For classification tasks, a supervised learning algorithm looks at the training data set to determine, or learn, the optimal combinations of variables that will generate a good predictive model.[26] The goal is to produce a trained (fitted) model that generalizes well to new, unknown data. The fitted model is evaluated using "new" examples from the held-out datasets (validation and test datasets) to estimate the model s accuracy in classifying new data. To reduce the risk of issues such as over-fitting, the examples in the validation and test datasets should not be used to train the model. Most approaches that search through training data for empirical relationships tend to overfit the data, meaning that they can identify and exploit apparent relationships in the training data that do not hold in general.

1.3.2 Validation data set

A validation data set is a data-set of examples used to tune the hyperparameters (i.e. the architecture) of a classifier. It is sometimes also called the development set or the "dev set". An example of a hyperparameter for artificial neural networks includes the number of hidden units in each layer. [27] It, as well as the testing set (as mentioned below), should follow the same probability distribution as the training data set.

In order to avoid overfitting, when any classification parameter needs to be adjusted, it is necessary to have a validation data set in addition to the training and test datasets. For example, if the most suitable classifier for the problem is sought, the training data set is used to train the different candidate classifiers, the validation data set is used to compare their performances and decide which one to take and, finally, the test data set is used to obtain the performance characteristics such as *accuracy, sensitivity, specificity, F-measure*, and so on. The validation data set functions as a hybrid: it is training data used for testing, but neither as part of the low-level training nor as part of the final testing.

1.3.3 Test data set

A test data set is a data set that is independent of the training data set, but that follows the same probability distribution as the training data set. If a model fit to the training data set also fits the test data set well, minimal overfitting has taken place (see figure below). A better fitting of the training data set as opposed to the test data set usually points to over-fitting.

A test set is therefore a set of examples used only to assess the performance (i.e. generalization) of a fully specified classifier. To do this, the final model is used to predict classifications of examples in the test set. Those predictions are compared to the examples' true classifications to assess the model's accuracy.

In a scenario where both validation and test datasets are used, the test data set is typically used to assess the final model that is selected during the validation process. In the case where the original data set is partitioned into two subsets (training and test datasets), the test data set might assess the model only once (e.g., in the holdout method). Note that some sources advise against such a method. However, when using a method such as cross-validation, two partitions can be sufficient and effective since results are averaged after repeated rounds of model training and testing to help reduce bias and variability.

1.3.4 Cross validation

Cross-validation, sometimes called rotation estimation or out-of-sample testing, is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance. In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance.

Method

Assume a model with one or more unknown parameters, and a data set to which the model can be fit (the training data set). The fitting process optimizes the model parameters to make the model fit the training data as well as possible. If an independent sample of validation data is taken from the same population as the training data, it will generally turn out that the model does not fit the validation data as well as it fits the training data. The size of this difference is likely to be large especially when the size of the training data set is small, or when the number of parameters in the model is large. Cross-validation is a way to estimate the size of this effect.

In linear regression, there exist real response values *y1, ..., yn*, and n p-dimensional vector covariates *x1, ..., xn.* The components of the vector *xi* are denoted *xi1, ..., xip.* If least squares is used to fit a function in the form of a hyperplane $\hat{y} = a + \beta^T x$ to the data $(xi, yi)1 \leq i \leq n$, then the fit can be assessed using the mean squared error (MSE). The MSE for given estimated parameter values a and β on the training set $(xi, yi)1 \leq i \leq n$ is defined as:

$$\begin{aligned}\text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - a - \boldsymbol{\beta}^T \mathbf{x}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - a - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2\end{aligned}$$

If the model is correctly specified, it can be shown under mild assumptions that the expected value of the MSE for the training set is $(n?p?1)/(n+p+1) < 1$ times the expected value of the MSE for the validation set (the expected value is taken over the distribution of training sets). Thus, a fitted model and computed MSE on the training set will result in an optimistically biased assessment of how well the model will fit an independent data set. This biased estimate is called the in-sample estimate of the fit, whereas the cross-validation estimate is an out-of-sample estimate.

Since in linear regression it is possible to directly compute the factor $(n?p?1)/(n+p+1)$ by which the training MSE underestimates the validation MSE under the assumption that the model specification is valid, cross-validation can be used for checking whether the model has been overfitted, in which case the MSE in the validation set will substantially exceed its anticipated value. (Cross-validation in the context of linear regression is also useful in that it can be used to select an optimally regularized cost function.) In most other regression procedures (e.g. logistic regression), there is no simple formula to compute the expected out-of-sample fit. Cross-validation is, thus, a generally applicable way to predict the performance of a model on unavailable data using numerical computation in place of theoretical analysis.

1.3.5 Overfitting/Underfitting a Model

As we know, in machine learning the data is usually split into two subsets: *training data* and *testing data*, and fit our model on the train data, in order to make predictions on the test data. As a result, there are two issues while fitting a model. They are *overfitting* or *underfitting*. These issues must be tackled in order to maintain accuracy

Overfitting

Overfitting means that model is beign trained “too well”. This usually happens when the model is too complex (i.e. too many features/variables compared to the number of observations). This model will be very accurate on the training data but will probably be very not accurate on untrained or new data. It is because this model is not generalized leading to bad results or prediction.

Basically, when this happens, the model learns or describes the "noise" in the training data instead of the actual relationships between variables in the data. This noise, obviously, is not part in of any new dataset, and cannot be applied to it.

Underfitting

In contrast to overfitting, when a model is underfitted, it means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data. This is usually the result of a very simple model. This happens due to lack of enough predictors/independent variables. It could also happen when, for example, we fit a linear model (like linear regression) to data that is not linear. It almost goes without saying that this model will have poor predictive ability (on training data and can t be generalized to other data). It is worth noting the underfitting is not as prevalent as overfitting. Nevertheless, avoiding both of those problems in data analysis yields good outcomes.

1.3.6 Python practice

Scikit-Learn library and specifically the `train_test_split` method is used to split arrays or matrices into random train and test subsets. Following code snippet shows as to how packages can be imported. The another package is *numpy* which is used to simulate data sets required for our task.

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

In the above snippet there are two packages that were imported namely *numpy* and *sklearn*. The package *numpy* is useful for data simulations. The function `train_test_split` from the package *sklearn* is useful for train, test and split tasks. Below code snippet performs the required tasks.

```
>>> X_train, X_test, y_train, y_test = train_test_split(X
                                       ↪ , y, test_size=0.33,
                                       ↪ random_state=42)
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_train
[2, 0, 3]
>>> y_test
[1, 4]
```

Let's try fitting data using Linear Regression and then test our data with cross validations.

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
print(model.score(X_test, y_test))
print(predictions)
```

The output would be

```
>>> model.score(X_test, y_test)
1.0

>>> predictions
array([1., 4.])
```

Compare `predictions` with `y_test` values. Perfect predictions. Isn't it? There are methods to know about intercept and coefficients. That is not required for this task. [28]. As far as cross validation is concerned, it is not necessary, because it is a perfect fit as we know (score is 1). However, it can be done as shown below for practice

```
>>> scores = cross_val_score(model, X, y, cv=2)
>>> print("Cross-validated scores:", scores)
Cross-validated scores: [1. 1.]
>>> predictions = cross_val_predict(model, X, y, cv=2)
>>> for i in predictions:
...     print(i)
...
3.885780586188048e-16
1.0000000000000002
2.0000000000000004
3.0
4.0
```

Perfect predictions.

Exercises

1. Learn data simulations using Python. There are number of ways to make data sets quickly using Python programming. Appendix 1 [6] shows few methods to perform data simulations. Create few data variables such as numeric and non-numeric and try to create data sets using *pandas* library.

2. Take a sample data set with certain valid variables. Perform linear regression and explain regression analysis with the help of summary measures. [For instance, you can use *diabetes* data set available from *sklearn* library.]

3. Perform test, split and train tasks on simulated data sets.

4. Plot regression fits using your own data sets using simulation methods.

5. Demonstrate model fit, model score and model predictions using imported data sets. Explain cross validation and accuracy of models.

6. Use any data sets of your interest and calculate AIC, BIC for regression models.

Notes

[1]Mitchell, Tom (1997). Machine Learning. New York: McGraw Hill. ISBN 0-07-042807-7.

[2]Friedman, Jerome H. (1998). "Data Mining and Statistics: What's the connection?". Computing Science and Statistics. 29 (1): 3 9.

[3]Zhou, Victor (2019-12-20). "Machine Learning for Beginners: An Introduction to Neural Networks". Medium. Retrieved 2021-08-15.

[4]Ethem Alpaydin (2020). Introduction to Machine Learning (Fourth ed.). MIT. pp. xix, 1 3, 13 18. ISBN 978-0262043793.

[5]Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. 3 (3): 210 229.

[6]R. Kohavi and F. Provost, "Glossary of terms," Machine Learning, vol. 30, no. 2 3, pp. 271 274, 1998.

[7]Gerovitch, Slava (9 April 2015). "How the Computer Got Its Revenge on the Soviet Union". Nautilus. Retrieved 19 September 2021.

[8]Duda, R., Hart P. Pattern Recognition and Scene Analysis, Wiley Interscience, 1973

[9]Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 978-0-07-042807-2.

[10]Sarle, Warren (1994). "Neural Networks and statistical models". CiteSeerX 10.1.1.27.699.

[11]Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. ISBN 978-0137903955.

[12]Langley, Pat (2011). "The changing science of machine learning". Machine Learning. 82 (3): 275 279. doi:10.1007/s10994-011-5242-y

[13]Alpaydin, Ethem (2010). Introduction to Machine Learning. MIT Press. p. 9. ISBN 978-0-262-01243-0.

[14]Bzdok, Danilo; Altman, Naomi; Krzywinski, Martin (2018). "Statistics versus Machine Learning". Nature Methods. 15 (4): 233 234.

[15]Michael I. Jordan (2014-09-10). "statistics and machine learning". reddit. Retrieved 2014-10-01.

[16]Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani (2013). An Introduction to Statistical Learning. Springer. p. vii.

[17]Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (Third ed.). Prentice Hall. ISBN 9780136042594.

[18]Mohri, Mehryar; Rostamizadeh, Afshin; Talwalkar, Ameet (2012). Foundations of Machine Learning. The MIT Press. ISBN 9780262018258.

[19]Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN

978-0-07-042807-2.

[20]Alpaydin, Ethem (2010). Introduction to Machine Learning. MIT Press. p. 9. ISBN 978-0-262-01243-0.

[21]Jordan, Michael I.; Bishop, Christopher M. (2004). "Neural Networks". In Allen B. Tucker (ed.). Computer Science Handbook, Second Edition (Section VII: Intelligent Systems). Boca Raton, Florida: Chapman Hall/CRC Press LLC. ISBN 978-1-58488-360-9.

[22]van Otterlo, M.; Wiering, M. (2012). Reinforcement learning and markov decision processes. Reinforcement Learning. Adaptation, Learning, and Optimization. Vol. 12. pp. 3 42. doi:10.1007/978-3-642-27645-$3_1 . ISBN 978 - 3 - 642 - 27644 - 6$.

[23]Ron Kohavi; Foster Provost (1998). "Glossary of terms". Machine Learning. 30: 271 274.

[24]James, Gareth (2013). An Introduction to Statistical Learning: with Applications in R. Springer. p. 176.

[25]Ripley, B.D. (1996) Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press, p. 354

[26]Larose, D. T.; Larose, C. D. (2014). Discovering knowledge in data : an introduction to data mining. Hoboken: Wiley.

[27]Ripley, B.D. (1996) Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press, p. 354

[28]There are ways to access measures. Suppose, if the model object is `fit` The intercept and coefficients can be retrieved using `fit.intercept_` and `fit.coef_`

Chapter 2

Supervised Learning

Supervised learning (SL) is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias). This statistical quality of an algorithm is measured through the so-called generalization error.[29] [30]

To solve a given problem of supervised learning, one has to perform the following steps:

1. *Determine the type of training examples.* Before doing

anything else, the user should decide what kind of data is to be used as a training set. In the case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, an entire sentence of handwriting or perhaps a full paragraph of handwriting.

2. *Gather a training set.* The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

3. *Determine the input feature representation of the learned function.* The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.

4. *Determine the structure of the learned function and corresponding learning algorithm.* For example, the engineer may choose to use support-vector machines or decision trees.

5. *Complete the design.* Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.

6. *Evaluate the accuracy of the learned function.* After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

2.1 Algorithms

The most widely used learning algorithms are:

1. Support-vector machines
2. Linear regression
3. Logistic regression
4. Naive Bayes
5. Linear discriminant analysis
6. Decision trees
7. K-nearest neighbor algorithm
8. Neural networks (Multilayer perceptron)
9. Similarity learning

2.2 Applications

1. Bioinformatics
2. Cheminformatics
3. Quantitative structure activity relationship
4. Database marketing
5. Handwriting recognition
6. Information retrieval
7. Learning to rank
8. Information extraction recognition in computer vision character recognition detection
9. Pattern recognition
10. Speech recognition

11. Supervised learning is a special case of downward causation in biological systems
12. Landform classification using satellite imagery

2.3 Logistic regression

In statistics logistic regression is used to model the probability of a certain class or event. I will be focusing more on the basics and implementation of the model, and not go too deep into the math part. Logistic regression is similar to linear regression because both of these involve estimating the values of parameters used in the prediction equation based on the given training data. Linear regression predicts the value of some continuous, dependent variable. Whereas logistic regression predicts the probability of an event or class that is dependent on other factors. Thus the output of logistic regression always lies between 0 and 1. Because of this property it is commonly used for classification purpose.

2.3.1 Logistic Model

Consider a model with features *x1, x2, x3 ... xn*. Let the binary output be denoted by *Y*, that can take the values 0 or 1. Let *p* be the probability of *Y = 1*, we can denote it as *p = P(Y=1)*. The mathematical relationship between these variables can be denoted as:

$$\log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{2.1}$$

Here the term *p/(1-p)* is known as the odds and denotes the likelihood of the event taking place. Thus *ln(p/(1-p))* is known as the log odds and is simply used to map the probability that lies between 0 and 1 to a range between $(-\infty, +\infty)$. The terms *b0, b1, b2 ...* are parameters (or weights) that we will estimate during training. So this is just the basic math behind what we are going to do. We are interested in the probability p in this equation. So we simplify the equation to obtain the value of p:

1. The log term *ln* on the LHS can be removed by raising the RHS as a power of e:

$$\frac{p}{1-p} = b^{\beta_0+\beta_1 x_1+\beta_2 x_2}. \tag{2.2}$$

2. Now we can easily simplify to obtain the value of p:

$$p = \frac{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}}{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}+1} = \frac{1}{1+b^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}} = S_b(\beta_0+\beta_1 x_1+\beta_2 x_2). \tag{2.3}$$

This actually turns out to be the equation of the Sigmoid Function which is widely used in other machine learning applications. The Sigmoid Function is given by:

$$\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}} \tag{2.4}$$

Now we will be using the above derived equation to make our predictions. Before that we will train our model to obtain the values of our parameters *b0, b1, b2 ...* that result in least error. This is where the error or loss function comes in.

2.3.2 Loss Function

The loss is basically the error in our predicted value. In other words it is a difference between our predicted value and the actual value. We will be using the L2 Loss Function to calculate the error. Theoretically you can use any function to calculate the error. This function can be broken down as:

1. Let the actual value be y_i. Let the value predicted using our model be denoted as $\hat{y_i}$. Find the difference between the actual and predicted value.

2. Square this difference.
3. Find the sum across all the values in training data.

$$L = \sum_{i=1}^{n} (y + i - \bar{y_i})^2 \tag{2.5}$$

Now that we have the error, we need to update the values of our parameters to minimize this error. This is where the "learning" actually happens, since our model is updating itself based on it s previous output to obtain a more accurate output in the next step. Hence with each iteration our model becomes more and more accurate. We will be using the Gradient Descent Algorithm to estimate our parameters. Another commonly used algorithm is the *Maximum Likelihood Estimation.*

2.3.3 The Gradient Descent Algorithm

You might know that the partial derivative of a function at it s minimum value is equal to *0.* So gradient descent basically uses this concept to estimate the parameters or weights of our model by minimizing the loss function. For simplicity, for the rest of this tutorial let us assume that our output depends only on a single feature *x.* So we can rewrite our equation as:

$$y_i = p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{2.6}$$

Thus we need to estimate the values of weights *b0* and *b1* using our given training data.

1. Initially let *b0=0* and *b1=0.* Let L be the learning rate. The learning rate controls by how much the values of b0 and b1 are updated at each step in the learning process. Here let *L=0.001.*

2. Calculate the partial derivative with respect to b0 and b1. The value of the partial derivative will tell us how far the loss function is from it s minimum value. It is a measure of how much our weights need to be updated to attain minimum or ideally *0* error. In case you have more than one feature, you need to calculate the partial derivative for each weight *b0, b1 ... bn* where n is the number of features.

$$D_{b0} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times y_i \times (1-\bar{y_i}) D_{b1} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times y_i \times (1-\bar{y_i}) \times x_i \tag{2.7}$$

3. Next we update the values of *b0* and *b1*:

$$b_0 = b_0 - L \times D_{b0} b_1 = b_1 - L \times D_{b1} \tag{2.8}$$

4. We repeat this process until our loss function is a very small value or ideally reaches 0 (meaning no errors and 100% accuracy). The number of times we repeat this learning process is known as iterations or epochs.

2.3.4 Python practice

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegressionCV
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegressionCV(cv=5, random_state=0).fit(
                                     ↪ X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :]).shape
(2, 3)
>>> clf.score(X, y)
0.98...
```

2.4 K-Nearest Neighbors Algorithm (k-NN)

The k-nearest neighbors algorithm (k-NN) is a non-parametric classification method first developed by *Evelyn Fix* and *Joseph Hodges* in 1951, and later expanded by *Thomas Cover.* [31] [32] It is used for classification and regression. In both cases, the input consists of the k closest training examples in data set. The output depends on whether *k-NN* is used for classification or regression:

1. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.
2. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically. [33] [34]

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

Suppose we have pairs $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ taking values in $R^d \times \{1, 2\}$, where Y is the class label of X, so that $X|Y = r \sim P_r$ for $r = 1, 2$ (and probability distributions P_r. Given some norm $\|\cdot\|$ on R^d and a point $x \in R^d$, let $(X_{(1)}, Y_{(1)}), \ldots, (X_{(n)}, Y_{(n)})$ be a reordering of the training data such that $\|X_{(1)} - x\| \leq \cdots \leq \|X_{(n)} - x\|$.

2.4.1 Algorithm

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, *k* is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has been employed with correlation coefficients, such as Pearson and Spearman, as a metric. Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

A drawback of the basic ”majority voting” classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number. One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. K-NN can then be applied to the SOM.

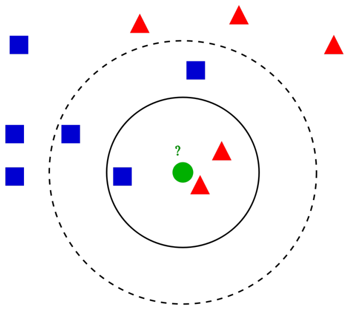

Figure 2.1: Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If $k = 3$ (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

2.4.2 Python practice

Demonstrate the resolution of a regression problem using a k-Nearest Neighbor and the interpolation of the target using both barycenter and constant weights.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn import neighbors

np.random.seed(0)
X = np.sort(5 * np.random.rand(40, 1), axis=0)
T = np.linspace(0, 5, 500)[:, np.newaxis]
y = np.sin(X).ravel()

# Add noise to targets
y[::5] += 1 * (0.5 - np.random.rand(8))

# Fit regression model
n_neighbors = 5

for i, weights in enumerate(['uniform', 'distance']):
    knn = neighbors.KNeighborsRegressor(n_neighbors,
                                      ↪ weights=weights)
    y_ = knn.fit(X, y).predict(T)

    plt.subplot(2, 1, i + 1)
    plt.scatter(X, y, color='darkorange', label='data')
    plt.plot(T, y_, color='navy', label='prediction')
    plt.axis('tight')
    plt.legend()
    plt.title("KNeighborsRegressor (k = %i, weights = '%s
                                      ↪ ')" % (n_neighbors,
                                      ↪ weights))

plt.tight_layout()
plt.show()
```

Figure 2.2: KNN for both *Uniform* and *distance* based weights.

2.5 Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

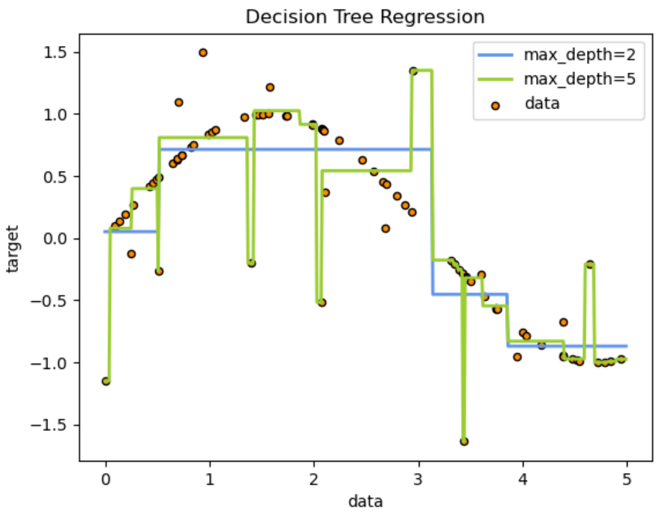

Figure 2.3: Decision Tree Regression

Some advantages of decision trees are:

1. Simple to understand and to interpret. Trees can be visualised.
2. Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
3. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the

tree.

4. Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.

5. Able to handle multi-output problems.

6. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

7. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

8. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

1. Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

2. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

3. Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.

4. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

5. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

6. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

2.5.1 Classification

`DecisionTreeClassifier` is a class capable of performing multi-class classification on a dataset. As with other classifiers, this class takes as input two arrays: an array X, sparse or dense, of shape $(n\_samples, n\_features)$ holding the training samples, and an array Y of integer values, shape $(n\_samples,)$, holding the class labels for the training samples:

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

After being fitted, the model can then be used to predict the class of samples:

```
>>> clf.predict([[2., 2.]])
array([1])
```

In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf:

```
>>> clf.predict_proba([[2., 2.]])
array([[0., 1.]])
```

`DecisionTreeClassifier` is capable of both binary (where the labels are *[-1, 1]*) classification and multiclass (where the labels are *[0,…, K-1]*) classification. Using the Iris dataset, we can construct a tree as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
```

Once trained, you can plot the tree with the `plot_tree` function:

```
>>> tree.plot_tree(clf)
```

The method `export_graphviz` in *Graphviz* package is useful to export the tree. If you use the conda package manager, the graphviz binaries and the python package can be installed with `conda install python-graphviz`. [35] Alternatively binaries for graphviz can be downloaded from the graphviz project homepage, and the Python wrapper installed from pypi with *pip install graphviz*. Below is an example graphviz export of the above tree trained on the entire iris dataset; the results are saved in an output file *iris.pdf*:

```
>>> import graphviz
>>> dot_data = tree.export_graphviz(clf, out_file=None)
>>> graph = graphviz.Source(dot_data)
>>> graph.render("iris")
```

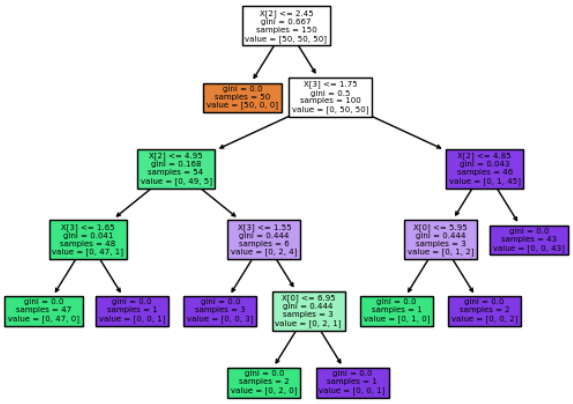

Figure 2.4: Decision Fit on Iris Data Set

The `export_graphviz` exporter also supports a variety of aesthetic options, including coloring nodes by their class (or value for regression) and using explicit variable and class names if desired. Jupyter notebooks also render these plots inline automatically:

```
>>> dot_data = tree.export_graphviz(clf, out_file=None,
...                          feature_names=iris.feature_names
    ↪ ,
...                          class_names=iris.target_names,
...                          filled=True, rounded=True,
...                          special_characters=True)
>>> graph = graphviz.Source(dot_data)
>>> graph
```

2.5.2 Regression

Decision trees can also be applied to regression problems, using the `DecisionTreeRegressor` class. As in the classification setting, the fit method will take as argument arrays X and y, only that in this case y is expected to have floating point values instead of integer values:

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```

2.6 Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are:

1. Effective in high dimensional spaces.

2. Still effective in cases where number of dimensions is greater than the number of samples.

3. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

4. *Versatile*: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

1. If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.

2. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

2.6.1 Classification

SVC, NuSVC and *LinearSVC* are classes capable of performing binary and multi-class classification on a dataset. *SVC* and *NuSVC* are similar methods, but accept slightly different sets of parameters and have different mathematical formulations (see section Mathematical formulation). On the other hand, *LinearSVC* is another (faster) implementation of Support Vector Classification for the case of a linear kernel. Note that *LinearSVC* does not accept parameter kernel, as this is assumed to be linear. It also lacks some of the attributes of *SVC* and *NuSVC*, like `support_`. As other classifiers, SVC, NuSVC and *LinearSVC* takes two input arrays. An array X with a shape of $(n\_samples, n\_features)$ holding the training samples, and an array y of class labels (strings or integers), of shape $(n\_samples)$.

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()
```

After being fitted, the model can then be used to predict new values:

```
>>> clf.predict([[2., 2.]])
array([1])
```

SVMs decision function (detailed in the Mathematical formulation) depends on some subset of the training data, called the support vectors. Some properties of these support vectors can be found in attributes *support_vectors_*, *support_* and *n_support_*:

```
>>> # get support vectors
>>> clf.support_vectors_
array([[0., 0.],
       [1., 1.]])
>>> # get indices of support vectors
>>> clf.support_
array([0, 1]...)
>>> # get number of support vectors for each class
>>> clf.n_support_
array([1, 1]...)
```

2.6.2 Regression

The method of Support Vector Classification can be extended to solve regression problems. This method is called Support Vector Regression. The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

There are three different implementations of Support Vector Regression: *SVR*, *NuSVR* and *LinearSVR*. *LinearSVR* provides a faster implementation than *SVR* but only considers the linear kernel, while *NuSVR* implements a slightly different formulation than *SVR* and *LinearSVR*. As with classification classes, the fit method will take as argument vectors X, y, only that in this case y is expected to have floating point values instead of integer values:

```
>>> from sklearn import svm
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> regr = svm.SVR()
>>> regr.fit(X, y)
SVR()
>>> regr.predict([[1, 1]])
array([1.5])
```

2.7 Na ve Bayes Algorithm

2.8 Random Forest

2.9 Rule based learning

2.9.1 Apriori Algorithm

Notes

[29]Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall ISBN 9780136042594.

[30]Cabannes, Vivien; Rudi, Alessandro; Bach, Francis (2021). ”Fast rates in Structured Prediction”. COLT. arXiv:2102.00760.

[31]Fix, Evelyn; Hodges, Joseph L. (1951). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties (PDF) (Report). USAF School of Aviation Medicine, Randolph Field, Texas.

[32]Altman, Naomi S. (1992). ”An introduction to kernel and nearest-neighbor nonparametric regression” (PDF). The American Statistician. 46 (3): 175 185. doi:10.1080/00031305.1992.10475879. hdl:1813/31637.

[33]Piryonesi S. Madeh; El-Diraby Tamer E. (2020-06-01). ”Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems”. Journal of Transportation Engineering, Part B: Pavements. 146 (2): 04020022.

[34]Hastie, Trevor. (2001). The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations. Tibshirani, Robert., Friedman, J. H. (Jerome H.). New York: Springer. ISBN 0-387-95284-5. OCLC 46809224.

[35]Visit `https://www.graphviz.org/` for more details.

Chapter 3

Unsupervised Learning

Unsupervised learning (UL) is a type of algorithm that learns patterns from untagged data. The hope is that, through mimicry, the machine is forced to build a compact internal representation of its world and then generate imaginative content. In contrast to supervised learning (SL) where data is tagged by a human, e.g. as “car” or “fish” etc, UL exhibits self-organization that captures patterns as neuronal predilections or probability densities. The other levels in the supervision spectrum are reinforcement learning where the machine is given only a numerical performance score as its guidance, and semi-supervised learning where a smaller portion of the data is tagged. Two broad methods in UL are Neural Networks and Probabilistic Methods. [36]

3.1 Approaches

Some of the most common algorithms used in unsupervised learning include: (1) Clustering, (2) Anomaly detection, (3) Neural Networks, and (4) Approaches for learning latent variable models. Each approach uses several methods as follows.

1. *Clustering methods* include: hierarchical clustering, k-means, mixture models, DBSCAN, and OPTICS algorithm
2. *Anomaly detection methods* include: Local Outlier Factor, and Isolation Forest
3. Approaches for learning latent variable models such as Expectation maximization algorithm (EM), Method of moments, and Blind signal separation techniques (Principal component analysis, Independent component analysis, Non-negative matrix factorization, Singular value decomposition)
4. *Neural Networks methods* include: Autoencoders, Deep Belief Nets, Hebbian Learning, Generative adversarial networks, and Self-organizing map

3.2 Clustering

Cluster analysis was originated in anthropology by Driver and Kroeber in 1932 and introduced to psychology by Joseph Zubin in 1938 and Robert Tryon in 1939 and famously used by Cattell beginning in 1943 for trait theory classification in personality psychology. [37] [38] [39] [40]

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It is a main task of exploratory data analysis, and a common technique for statistical data analysis, used in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances between cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including parameters such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

Besides the term clustering, there are a number of terms with similar meanings, including automatic classification, numerical taxonomy, botryology, typological analysis, and community detection. The subtle differences are often in the use of the results: while in data mining, the resulting groups are the matter of interest, in automatic classification the resulting discriminative power is of interest.

3.2.1 Algorithms for clustering

Clustering algorithms can be categorized based on their cluster model. The following overview will only list the most prominent examples of clustering algorithms, as there are possibly over 100 published clustering algorithms. Not all provide models for their clusters and can thus not easily be categorized. An overview of algorithms explained in Wikipedia can be found in the list of statistics algorithms.

There is no objectively "correct" clustering algorithm, but as it was noted, "clustering is in the eye of the beholder." The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. An algorithm that is designed for one kind of model will generally fail on a data set that contains a radically different kind of model. For example, k-means cannot find non-convex clusters. [41] Following are some of the algorithms available for clustering.

1. *Connectivity models*: for example, hierarchical clustering builds models based on distance connectivity.
2. *Centroid models*: for example, the k-means algorithm represents each cluster by a single mean vector.
3. *Distribution models*: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the expectation-maximization algorithm.
4. *Density models*: for example, DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
5. *Subspace models*: in biclustering (also known as co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.
6. *Group models*: some algorithms do not provide a refined model for their results and just provide the grouping information.
7. *Graph-based models*: a clique, that is, a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques, as in the HCS clustering algorithm.

8. *Signed graph models*: Every path in a signed graph has a sign from the product of the signs on the edges. Under the assumptions of balance theory, edges may change sign and result in a bifurcated graph. The weaker "clusterability axiom" (no cycle has exactly one negative edge) yields results with more than two clusters, or subgraphs with only positive edges.

9. *Neural models:* the most well known unsupervised neural network is the self-organizing map and these models can usually be characterized as similar to one or more of the above models, and including subspace models when neural networks implement a form of Principal Component Analysis or Independent Component Analysis.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example, a hierarchy of clusters embedded in each other. Clusterings can be roughly distinguished as:

1. *Hard clustering*: each object belongs to a cluster or not

2. *Soft clustering (fuzzy clustering)*: each object belongs to each cluster to a certain degree (for example, a likelihood of belonging to the cluster)

There are also finer distinctions possible, for example:

1. *Strict partitioning clustering*: each object belongs to exactly one cluster

2. *Strict partitioning clustering with outliers*: objects can also belong to no cluster, and are considered outliers Overlapping clustering (also: alternative clustering, multi-view clustering): objects may belong to more than one cluster; usually involving hard clusters

3. *Hierarchical clustering*: objects that belong to a child cluster also belong to the parent cluster

4. *Subspace clustering*: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap

3.3 K-Means clustering

Also known as centroid-based clustering. In centroid-based clustering clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k, k-means clustering gives a formal definition as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized.

The optimization problem itself is known to be NP-hard, and thus the common approach is to search only for approximate solutions. A particularly well known approximate method is *Lloyd's algorithm*, often just referred to as "k-means algorithm". It does however only find a local optimum, and is commonly run multiple times with different random initializations. Variations of k-means often include such optimizations as choosing the best of multiple runs, but also restricting the centroids to members of the data set (k-medoids), choosing medians (k-medians clustering), choosing the initial centers less randomly (k-means++) or allowing a fuzzy cluster assignment (fuzzy c-means).

Most k-means-type algorithms require the number of clusters - k - to be specified in advance, which is considered to be one of the biggest drawbacks of these algorithms. Furthermore, the algorithms prefer clusters of approximately similar size, as they will always assign an object to the nearest centroid. This often leads to incorrectly cut borders of clusters (which is not surprising since the algorithm optimizes cluster centers, not cluster borders).

K-means has a number of interesting theoretical properties. First, it partitions the data space into a structure known as a Voronoi diagram. Second, it is conceptually close to nearest neighbor classification, and as such is popular in machine learning. Third, it can be seen as a variation of model based clustering, and Lloyd's algorithm as a variation of the Expectation-maximization algorithm for this model discussed below.

Centroid-based clustering problems such as *k-means* and *k-medoids* are special cases of the uncapacitated, metric facility location problem, a canonical problem in the operations research and computational geometry communities. In a basic facility location problem (of which there are numerous variants that model more elaborate settings), the task is to find the best warehouse locations to optimally service a given set of consumers. One may view "warehouses" as cluster centroids and "consumer locations" as the data to be clustered. This makes it possible to apply the well-developed algorithmic solutions from the facility location literature to the presently considered centroid-based clustering problem.

3.4 K-Means clustering using Python

Clustering of unlabeled data can be performed with the module `sklearn.cluster`. Each clustering algorithm comes in two variants: a *class,* that implements the `fit` method to learn the clusters on train data, and a *function*, that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the `labels_` attribute.

3.4.1 clustering algorithms in ***scikit-learn***

1. K-Means
2. Affinity propagation
3. Mean-shift

4. Spectral clustering
5. Ward hierarchical clustering
6. Agglomerative clustering
7. DBSCAN
8. OPTICS
9. Gaussian mixtures
10. BIRCH

The *KMeans* algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields. The k-means algorithm divides a set of samples into disjoint clusters , each described by the mean of the samples in the cluster. The means are commonly called the cluster centroids ; note that they are not, in general, points from , although they live in the same space. The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

$$\sum_{i=0}^{n} \frac{min}{\mu_{j \in c}} (||x_i - \mu_i||^2) \tag{3.1}$$

Inertia can be recognized as a measure of how internally coherent clusters are. It suffers from various drawbacks. [42]. The k-means problem is solved using either *Lloyd s* or *Elkan s* algorithm. **The average complexity is given by *O(k n T)*, where *n* is the number of samples and *T* is the number of iteration.** The worst case complexity is given by $O(n^{k+2/p})$ with n = samples, p = features. In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in *local minima.* That s why it can be useful to restart it several times. If the algorithm stops before fully converging (because of `tol` or `max_iter`), `labels_` and `cluster_centers_` will not be consistent, i.e. the `cluster_centers_` will not be the means of the points in each cluster. Also, the estimator will reassign `labels_` after the last iteration to make `labels_` consistent with predict on the training set.

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

3.5 Hierarchical clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

1. *Agglomerative*: This is a “bottom-up” approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

2. *Divisive*: This is a “top-down” approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a *dendrogram.* [43] [44]

The standard algorithm for hierarchical agglomerative clustering (HAC) has a time complexity of $\mathcal{O}(n^3)$ and requires $\Omega(n^2)$ memory, which makes it too slow for even medium data sets. However, for some special cases, optimal efficient agglomerative methods (of complexity $\mathcal{O}(n^2)$) are known: SLINK for single-linkage and CLINK for complete-linkage clustering. With a heap, the runtime of the general case can be reduced to $\mathcal{O}(n^2 \log n)$, an improvement on the aforementioned bound of $\mathcal{O}(n^3)$, at the cost of further increasing the memory requirements. In many cases, the memory overheads of this approach are too large to make it practically usable. Except for the special case of single-linkage, none of the algorithms (except exhaustive search in $\mathcal{O}(2^n)$ can be guaranteed to find the optimum solution. Divisive clustering with an exhaustive search is $\mathcal{O}(2^n)$, but it is common to use faster heuristics to choose splits, such as k-means. [45] [46]

In order to decide which clusters should be combined (for agglomerative), or where a cluster should be split (for divisive), a measure of dissimilarity between sets of observations is required. In most methods of hierarchical clustering, this is achieved by use of an appropriate metric (a measure of distance between pairs of observations), and a linkage criterion which specifies the dissimilarity of sets as a function of the pairwise distances of observations in the sets.

The choice of an appropriate metric will influence the shape of the clusters, as some elements may be relatively closer to one another under one metric than another. For example, in two dimensions, under the Manhattan distance metric, the distance between the origin (0,0) and (0.5, 0.5) is the same as the distance between the origin and (0, 1), while under the Euclidean distance metric the latter is strictly greater. Some commonly used metrics for hierarchical clustering are:

Names	Formula
Euclidean distance	$\|a-b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$
Squared Euclidean distance	$\|a-b\|_2^2 = \sum_i (a_i - b_i)^2$
Manhattan distance	$\|a-b\|_1 = \sum_i \lvert a_i - b_i \rvert$
Maximum distance	$\|a-b\|_\infty = \max_i \lvert a_i - b_i \rvert$
Mahalanobis distance	$\sqrt{(a-b)^\top S^{-1}(a-b)}$ where S is the Covariance matrix

Table 3.1: Distance metric for HAC.

3.5.1 Python practice

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample. See the Wikipedia page for more details. The AgglomerativeClustering object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The linkage criteria determines the metric used for the merge strategy:

1. Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

2. Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.

3. Average linkage minimizes the average of the distances between all observations of pairs of clusters.

4. Single linkage minimizes the distance between the closest observations of pairs of clusters.

AgglomerativeClustering can also scale to large number of samples when it is used jointly with a connectivity matrix, but is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.

```
>>> from sklearn.cluster import AgglomerativeClustering
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> clustering = AgglomerativeClustering().fit(X)
>>> clustering
AgglomerativeClustering()
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
```

3.5.2 Plotting Dendrogram

This example plots the corresponding dendrogram of a hierarchical clustering using AgglomerativeClustering and the dendrogram method available in scipy.

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx -
                                                ↪ n_samples]
```

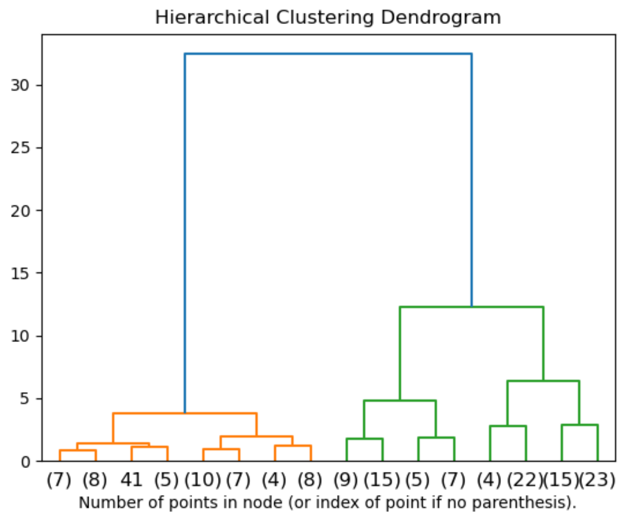

Figure 3.1: Agglomerative dendrogram

```
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


iris = load_iris()
X = iris.data

# setting distance_threshold=0 ensures we compute the
                             ↪ full tree.
model = AgglomerativeClustering(distance_threshold=0,
                             ↪ n_clusters=None)
```

```
model = model.fit(X)
plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point
                              ↪ if no parenthesis).")
plt.show()
```

3.6 Anomaly Detection

3.7 Expectation Maximization (EM) algorithm

3.8 Reinforcement Learning

Notes

[36]Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of Neural Computation. MIT Press. ISBN 978-0262581684.

[37]Driver and Kroeber (1932). ”Quantitative Expression of Cultural Relationships”. University of California Publications in American Archaeology and Ethnology. Quantitative Expression of Cultural Relationships: 211 256 via http://dpg.lib.berkeley.edu.

[38]Zubin, Joseph (1938). ”A technique for measuring like-mindedness”. The Journal of Abnormal and Social Psychology. 33 (4): 508 516. doi:10.1037/h0055441. ISSN 0096-851X.

[39]Tryon, Robert C. (1939). Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality. Edwards Brothers.

[40]Cattell, R. B. (1943). ”The description of personality: Basic traits resolved into clusters”. Journal of Abnormal and Social Psychology. 38 (4): 476 506. doi:10.1037/h0054116.

[41]Estivill-Castro, Vladimir (20 June 2002). ”Why so many clustering algorithms A Position Paper”. ACM SIGKDD Explorations Newsletter. 4 (1): 65 75. doi:10.1145/568574.568575. S2CID 7329935.

[42]See the KMeans documentation at `https://scikit-learn.org/stable/modules/clustering.html#k-means`

[43]Maimon, Oded; Rokach, Lior (2006). ”Clustering methods”. Data Mining and Knowledge Discovery Handbook. Springer. pp. 321 352. ISBN 978-0-387-25465-4.

[44]Nielsen, Frank (2016). ”8. Hierarchical Clustering”. Introduction to HPC with MPI for Data Science. Springer. pp. 195 211. ISBN 978-3-319-21903-5.

[45]R. Sibson (1973). ”SLINK: an optimally efficient algorithm for the single-link cluster method” (PDF). The Computer Journal. British Computer Society. 16 (1): 30 34. doi:10.1093/comjnl/16.1.30.

[46]D. Defays (1977). ”An efficient algorithm for a complete-link method”. The Computer Journal. British Computer Society. 20 (4): 364 6. doi:10.1093/comjnl/20.4.364.

Chapter 4

Introduction to Deep Learning

4.1 Concept

4.2 Artificial Neural Networks

4.2.1 Basic Structure of ANN

4.2.2 Types of ANN

4.2.3 Definition of ANN

4.2.4 Training ANN

Notes

[36]Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of Neural Computation. MIT Press. ISBN 978-0262581684.

[37]Driver and Kroeber (1932). "Quantitative Expression of Cultural Relationships". University

of California Publications in American Archaeology and Ethnology. Quantitative Expression of Cultural Relationships: 211 256 via http://dpg.lib.berkeley.edu.

[38]Zubin, Joseph (1938). "A technique for measuring like-mindedness". The Journal of Abnormal and Social Psychology. 33 (4): 508 516. doi:10.1037/h0055441. ISSN 0096-851X.

[39]Tryon, Robert C. (1939). Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality. Edwards Brothers.

[40]Cattell, R. B. (1943). "The description of personality: Basic traits resolved into clusters". Journal of Abnormal and Social Psychology. 38 (4): 476 506. doi:10.1037/h0054116.

[41]Estivill-Castro, Vladimir (20 June 2002). "Why so many clustering algorithms A Position Paper". ACM SIGKDD Explorations Newsletter. 4 (1): 65 75. doi:10.1145/568574.568575. S2CID 7329935.

[42]See the KMeans documentation at https://scikit-learn.org/stable/modules/clustering.html#k-means

[43]Maimon, Oded; Rokach, Lior (2006). "Clustering methods". Data Mining and Knowledge Discovery Handbook. Springer. pp. 321 352. ISBN 978-0-387-25465-4.

[44]Nielsen, Frank (2016). "8. Hierarchical Clustering". Introduction to HPC with MPI for Data Science. Springer. pp. 195 211. ISBN 978-3-319-21903-5.

[45]R. Sibson (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method" (PDF). The Computer Journal. British Computer Society. 16 (1): 30 34. doi:10.1093/comjnl/16.1.30.

[46]D. Defays (1977). "An efficient algorithm for a complete-link method". The Computer Journal. British Computer Society. 20 (4): 364 6. doi:10.1093/comjnl/20.4.364.

Chapter 5

Applications of Machine Learning

5.1 Sales and Marketing

5.2 Financial Services

5.3 Social Media Analysis

5.4 Fraud Detection

Notes

[36]Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of Neural Computation. MIT Press. ISBN 978-0262581684.

[37]Driver and Kroeber (1932). "Quantitative Expression of Cultural Relationships". University of California Publications in American Archaeology and Ethnology. Quantitative Expression of Cultural Relationships: 211 256 via http://dpg.lib.berkeley.edu.

[38]Zubin, Joseph (1938). "A

technique for measuring like-mindedness". The Journal of Abnormal and Social Psychology. 33 (4): 508 516. doi:10.1037/h0055441. ISSN 0096-851X.

[39]Tryon, Robert C. (1939). Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality. Edwards Brothers.

[40]Cattell, R. B. (1943). "The description of personality: Basic traits resolved into clusters". Journal of Abnormal and Social Psychology. 38 (4): 476 506. doi:10.1037/h0054116.

[41]Estivill-Castro, Vladimir (20 June 2002). "Why so many clustering algorithms A Position Paper". ACM SIGKDD Explorations Newsletter. 4 (1): 65 75. doi:10.1145/568574.568575. S2CID 7329935.

[42]See the KMeans documentation at `https://scikit-learn.org/stable/modules/clustering.html#k-means`

[43]Maimon, Oded; Rokach, Lior (2006). "Clustering methods". Data Mining and Knowledge Discovery Handbook. Springer. pp. 321 352. ISBN 978-0-387-25465-4.

[44]Nielsen, Frank (2016). "8. Hierarchical Clustering". Introduction to HPC with MPI for Data Science. Springer. pp. 195 211. ISBN 978-3-319-21903-5.

[45]R. Sibson (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method" (PDF). The Computer Journal. British Computer Society. 16 (1): 30 34. doi:10.1093/comjnl/16.1.30.

[46]D. Defays (1977). "An efficient algorithm for a complete-link method". The Computer Journal. British Computer Society. 20 (4): 364 6. doi:10.1093/comjnl/20.4.364.

Chapter 6

Appendix 1

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as *list comprehensions, cycle-detecting garbage collection, reference counting*, and *Unicode support*. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020. Python consistently ranks as one of the most popular programming languages in the world.

6.1 Installation

Installing Python is very easy doesn't matter if it is Windows, Linux or MacOS. In this section I am going to explain you installation methods for both Windows and Linux. Though, it is not a big deal as such but helps in working with both *OSes*. Most of the world is still revolving around MS Windows. In desktop computing, Windows is the largest OS that rules the world with roughly 87.07% market share. [47]. The second being *Mac* (9.54%), followed by Linux (2.35%). However, this is only related to desktop computing. Android is, of course, leading in mobile phones and other hand-held gadgets. So, most of the folk who use computers seem to use Windows mostly. So, I thought of including a section for Windows folk. However, most of the tasks that I have shown here or the rest of the book are implemented in Linux platform.

6.1.1 Installing in Linux

Working with Python in Linux is very easy for being one simple reason that you don't need to install at all, if you are using a *debian* system like Ubuntu. It is default programming language which comes automatically through OS installation. Execute the following command in the Terminal, to know the current version. Open the terminal by executing keyboard shortcut key such as *Alt+Ctl+T*, if you are in Linux like OS.

```
python --version
```

There will be two Python distributions, in any Linux, at least in Debian based systems. Same statement if executed in Ubuntu (18.04) will return 2.7. Python by default is 2.7. There is other command for version 3 i.e. `python3 --version`. If you skip `--version`, the terminal gets you . Oh! I didn't tell you what is REPL right! REPL is known as *Read-Eval-Print Loop* in short we refer to REPL. It is *an interactive top level or language shell,* *is a simple, interactive computer programming environment that takes single user inputs (i.e. single expressions), evaluates them, and returns the result to the user.* [48] If your system is Debian based system:

```
sudo apt-get install python3
```

You can also use `python2.7` in stead of python3. There are addeed advantages of having Python 2.7 if not both. This may be owing to the fact that python grew exponentially during 2.7.

6.1.2 Installing in Windows

Installation in Windows is not that difficult. Python is available as executable binaries. Visit https://www.python.org/ which is an official portal for all Python development. One important section that needs your attention is *PyPI* besides *Doc*. I mean *menus* at https://www.python.org/ website.

6.2 Execution

The program is a collection of statements called a module or simply a script. Execution is the way a script is being processed. Usually languages are expected to be operated thorugh terminals or consoles of respective OSes. However, for user ease there are plenty of Editors. Editors helps users while performing user level tasks such as cut, paste, copy etc. Python executes code in two modes; (1) interactive, (2) batch. Observe the following chunk of code.

6.2.1 Interactive

Interactive mode needs either a shell like utility such as Bash in Linux and CMD in Windows. Python has a shell called REPL. REPL gives a Python prompt for user where the statements or commands can be executed.

```
>>> 1+2
3
>>> (1+2)/2
1.5
>>> print("Hello World!")
Hello World!
```

6.2.2 Batch

Now that you know how to start Python, you should also know how to close the same. To quit Python you need to execute `exit()` not `exit`, please be cautious. The other type of execution is called *batch mode*. The primary purpose of batch mode, of course, is to develop modules and applications. Once the module is finished, it may be called as *source code*. However, developing source code and applications is beyond the scope of this book. However, it is better to know various modes of dealing with Python. We need to follow certain discipline to execute code in batch mode. The following are the few steps that might help understand the batch mode execution in Python.

1. Open text editor (such as *gedit* or *notepad*): It is always better to keep a separate directory for all Python related work. You may use `sudo mkdir` *your_directory_name* if it is *debian* OS. In case of Windows; File Explorer -> Home + New Folder or simply execute *Ctl + Shift + N* inside the window. [49]

2. Write Python statements: We will try a program that returns a sum of two values.

3. Close the file and go back to the Terminal.

4. Execute the file by using `python` *`file`*.

The following shows the execution of the code in Linux Terminal.

```
sudo mkdir python_work
sudo gedit myFirstProg.py
```

1. The first statement creates the directory with a name `python_work`.
2. The second statement opens a text editor. [50]
3. You may write your program as shown below

   ```
   a=2
   b=3
   c=a+b
   print("The Sum of %d and %d is %d" %(a, b, c))
   ```

4. Go back to Terminal and execute the following code.

   ```
   python myFirstProg.py
   ```

   The result of the above statment might be *The Sum of 2, 3 is 5.*

Of couse this is done in certain working directory which also can be work space for the tool.

6.3 I/O and file management

Python has nice ways of managing inputs and outputs. The *os* module is useful to know about file system.

```
>>> import os
>>> os.getcwd()
'C:\\Program Files\\Python310'
>>> os.chdir('D:\\Miscel')
>>> os.getcwd()
'D:\\Miscel'
>>> os.listdir()
['django', 'Document.docx', 'markdown-demo', 'Rplots.pdf'
                                ↪ , 'screen shots', 'test.R'
                                ↪ ]
>>> os.mkdir('pythonwork')
>>> os.listdir()
['django', 'Document.docx', 'markdown-demo', 'pythonwork'
                                ↪ , 'Rplots.pdf', 'screen
                                ↪ shots', 'test.R']
>>> os.rmdir('pythonwork')
```

```
>>> os.listdir()
['django', 'Document.docx', 'markdown-demo', 'Rplots.pdf'
                                        ↪ , 'screen shots', 'test.R'
                                        ↪ ]
```

Python use \\ as path separator. The above code snippet shows few tasks such as creating, changing removing directories. Following code snippet shows writing data to a file.

```
>>> file_path = os.path.join('pythonwork', 'test.txt')
>>> file_path
'pythonwork\\test.txt'
```

These statements will create a file with a name *test.txt* in current directory.

```
with open(file_path, 'w') as fw:
    for i in range(5):
        fw.write(str(i))
    print('Done')


1
1
1
1
1
Done
```

Above code writes a linear number series from 1 to 5 to the newly created file.

```
>>> with open(file_path, 'r') as fr:
...     x = fr.read()
...     print('Done')
...
...
Done
>>> x
'01234'
>>> for i in x:
...     int(i)
```

```
...
...
0
1
2
3
4
```

Above code reads the data to Python shell from the same file that was created earlier. It is also possible to save such data as in the form of data structures such as vectors, arrays, dictionaries etc. There is a in-built module called *csv* which helps reading data from CSV files. Suppose if I have some gender based data in certain CSV file called gender.csv in my file system. The data can be read to Python shell using the following code.

```
>>> with open('pythonwork\\gender.csv', 'r') as gf:
...     gender = gf.readlines()
...
...
>>> gender
['male\n', 'female\n', 'male\n', 'female\n', 'female\n',
                                             ↪ 'male\n', 'female\n', '
                                             ↪ male\n', 'female\n', 'male
                                             ↪ \n']
```

The resultant object *gender* is a list. We can print the data nicely ordered in the shell

```
>>> for i in gender:
...     print(i.replace('\n', ''))
...
...
male
female
male
female
female
male
female
male
female
```

```
male
```

CSV files can be structured in rows and columns. At times we may be having more than one column in CSV files. Assume that I have two columns in a file called *data.csv* which represents *gender* and *age*. In such case the file can processed as shown below.

```
>>> with open('pythonwork\\gender.csv', 'r') as gf:
...     data = gf.readlines()
...
...
>>> data
['male,21\n', 'female,74\n', 'male,64\n', 'female,62\n',
                                              ↪ 'female,66\n', 'male,23\n'
                                              ↪ , 'female,30\n', 'male,73\
                                              ↪ n', 'female,62\n', 'male,
                                              ↪ 80\n']
```

The data in the object *data* is not usable. It needs to be separated into two separate lists in order to use the variables.

```
>>> for i in data:
...               a, b = i.split(',')
...               j.append(a)
...               k.append(b)
...               print('Done')
...
...
Done
Done
Done
Done
Done
Done
Done
Done
Done
Done
>>> j
...
```

```
['male', 'female', 'male', 'female', 'female', 'male', '
                                             ↪ female', 'male', 'female',
                                             ↪  'male']
>>> k
...
['21\n', '74\n', '64\n', '62\n', '66\n', '23\n', '30\n',
                                             ↪ '73\n', '62\n', '80\n']
>>> k_ = []
>>> for i in k:
...                 k_.append(i.replace('\n', ''))
>>> k_
...
['21', '74', '64', '62', '66', '23', '30', '73', '62', '
                                             ↪ 80']
```

Let us computes summaries for these two variables.

```
>>> c = 0
>>> d = 0
>>> for i in j:
...                 if i == 'male':
...                     c = c + 1
...                 else:
...                     d = d + 1
...
...
>>> (c, d)
(6, 5)
>>> print('gender: ' + repr(c) + ', ' + 'age: ' + repr(d)
                                             ↪ )
gender: 6, age: 5
```

6.4 Data types & data structures

Data type refers to primitives of programming. For instance, string, integer, float, double, complex are known as primitives. Data structures are combinations of these types. For instance,

```
>>> type(1+2)
<class 'int'>
>>> x = 1; y=2; z=x+y; type(z)
```

```
<class 'int'>
>>> x = 1; y=2; z=x/y; type(z)
<class 'float'>
>>> x = 1000000; y=2000000; z=x/y; type(z)
<class 'float'>
>>> x = 1.5; y=2.5; z=x*y; type(z)
<class 'float'>
>>> my_name = 'kamakshaiah musunuru'
>>> type(my_name)
<class 'str'>
```

Python has sufficiently large number of data structures for various needs of data processing. Following is the list of data structures that are supported in

1. Tuple
2. List
3. Dictionary
4. Set

6.4.1 Tuple

```
>>> x, y = (1, 2)
>>> x; y
1
2
>>> x, y = (1, 2)
>>> x; y
1
2
>>> x = (1, 2, 3, 4, 5)
>>> x[0]
1
>>> x[1]
2
```

Tuples are highly dynamic and easy to handle through index numbers. Tuples are created using *parentheses* to contain values.

6.4.2 List

Lists are created using *square brackets* to contain values. Lists has quite a few methods associated with it. Use help('list') to know more about lists.

```
>>> x = list()
...
>>> type(x)
...
<class 'list'>
>>> list_methods = [func for func in dir(list) if
                                  ↪ callable(getattr(list,
                                  ↪ func)) and not func.
                                  ↪ startswith('__')]
...
>>> len(list_methods)
...
11
>>> list_methods
...
['append', 'clear', 'copy', 'count', 'extend', 'index', '
                                  ↪ insert', 'pop', 'remove',
                                  ↪ 'reverse', 'sort']
```

List is a class not a function. There are 11 methods associated with *list* class. How lists are contained?

```
>>> import random
>>> random.randint(1, 2)
2
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> x = list(range(5))
>>> x
[0, 1, 2, 3, 4]
```

List comprehensions

One of the most distinctive features of Python is the *list comprehension*, which can be used to create powerful functionality within a single line of code. However, many developers struggle to fully leverage the more advanced features of a list comprehension in Python. Some programmers even use them too much, which can lead to code that s less efficient and harder to read. List comprehensions provides easy way to manipulate lists. For instance,

```
>>> [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [random.randint(1, 10) for i in range(10)]
...
[7, 5, 3, 4, 4, 4, 9, 7, 7, 5]
>>> [round(random.random()*100, 2) for i in range(10)]
...
[55.18, 71.09, 82.87, 32.67, 3.51, 66.0, 36.21, 26.81, 22
                              ↪ .56, 50.77]
>>>
```

We can even create named lists and code, transform them as shown below.

```
>>> x = [random.randint(1, 2) for i in range(10)]
>>> x
[1, 1, 2, 2, 1, 1, 2, 2, 1, 1]

>>> y = ['male' if x[i] == 1 else 'female' for i in range
                              ↪ (len(x))]
>>> y
['male', 'male', 'female', 'female', 'male', 'male', '
                              ↪ female', 'female', 'male',
                              ↪  'male']
```

6.4.3 Dictionary

The other important data structure is *dictionary*. Dictionary is a special type of data structure that has two important attributes called *keys* and *values*.

```
>>> data_dict = dict(a=1, b=[round(random.random()*100, 2
                              ↪ ) for i in range(5)])
>>> data_dict
{'a': 1, 'b': [92.03, 48.61, 15.63, 91.54, 49.51]}
```

```
>>> data_dict.keys()
dict_keys(['a', 'b'])
>>> data_dict.items()
dict_items([('a', 1), ('b', [92.03, 48.61, 15.63, 91.54,
                             ↪ 49.51])])
>>> data_dict.keys()
dict_keys(['a', 'b'])
>>> data_dict.values()
dict_values([1, [92.03, 48.61, 15.63, 91.54, 49.51]])
```

It is possible to make summaries of contents of dictionaries, but they need to be changed to lists or other structures suitable for computations.

```
>>> import statistics as sts

>>> list(data_dict.values())
[1, [92.03, 48.61, 15.63, 91.54, 49.51]]
>>> list(data_dict.values())[1]
[92.03, 48.61, 15.63, 91.54, 49.51]
>>> sts.mean(list(data_dict.values())[1])
59.464
```

6.4.4 Set

A set can be thought of simply as a well-defined collection of distinct objects, typically called elements or members. Sets are useful to arrange unique elements of any data structure into another data structure. For instance,

```
>>> education = [random.randint(1, 3) for i in range(10)]
...
>>> education
...
[2, 1, 3, 3, 1, 3, 1, 1, 1, 1]
>>> educ_factor = ['primary' if education[i] == 1 else '
                             ↪ secondary' if education[i]
                             ↪  == 2  else 'higher' for i
                             ↪  in range(len(education))]
...
>>> educ_dict = dict(data=education, levels=set(
                             ↪ educ_factor))
```

```
...
>>> educ_dict['data']
...
[2, 1, 3, 3, 1, 3, 1, 1, 1, 1]
>>> educ_dict['levels']
...
{'primary', 'secondary', 'higher'}

>>> list(educ_dict['levels'])[0]
...
'primary'
```

In the above code, *set()* is used to find out unique instances or elements in the variable called *education*. There are so many methods associated with this class set.

```
>>> x = set()
...
>>> type(x)
...
<class 'set'>
```

So *set()* is not a function but a class.

```
>>> method_list = [func for func in dir(set) if callable(
                             ↪ getattr(set, func)) and
                             ↪ not func.startswith("__")]
...
>>> method_list
...
['add', 'clear', 'copy', 'difference', 'difference_update
                             ↪ ', 'discard', '
                             ↪ intersection', '
                             ↪ intersection_update', '
                             ↪ isdisjoint', 'issubset', '
                             ↪ issuperset', 'pop', '
                             ↪ remove', '
                             ↪ symmetric_difference', '
                             ↪ symmetric_difference_update
                             ↪ ', 'union', 'update']
>>> len(method_list)
...
17
```

There are approximately 17 different methods available for *set()* class.

6.5 Functions

Python is multi-paradigm language. It is possible to write functions and routines related to object oriented programming. Functions are defined by a keyword called *def* followed by name of the function and then arguments within parentheses.

```
def mean(x):
    return sum(x)/len(x)

def variance(x):
    n = len(x)
    am = mean(x)
    return sum((x - am) ** 2 for x in x) / (n - 1)

def stdev(x):
    res = variance(x)
    return math.sqrt(res)
```

Above code has three functions and they calculates *arithmetic mean, variance* and *standard deviation* respectively. Now let us test these functions with data simulated using an in-built module called *random.*

```
import random
import math
import statistics as sts

data_x  = [random.randint(10, 100) for i in range(10)]
print(data_x)
print(mean(data_x))
print(sts.mean(data_x))
print(variance(data_x))
print(sts.variance(data_x))
print(stdev(data_x))
print(sts.stdev(data_x))
```

The output will be

```
[54, 49, 59, 93, 26, 40, 40, 78, 31, 63]
53.3
53.3
436.4555555555555
436.4555555555556
20.89151874698332
20.891518746983323
```

Perfect.

6.6 Classes

```
class summaries():

    data = list()

    def __init__(self, data):
        self.data = data

    def mean(self):
        return sum(self.data)/len(self.data)

    def variance(self):
        n = len(self.data)
        am = self.mean()
        return sum((i - am) ** 2 for i in self.data) / (n
                                                   ↪  - 1)

    def stdev(self):
        res = self.variance()
        return math.sqrt(res)
```

Testing

```
import random
import math
import statistics as sts

x  = [random.randint(10, 100) for i in range(10)]
summaries = summaries(x)
print(summaries.mean())
print(summaries.variance())
print(summaries.stdev())
```

Output

```
58.9
892.5444444444444
29.87548232990464
```

Chapter 7

Appendix 2

7.1 Data simulations

Programmers never care for data, they just use data only for testing code. In fact, programmers can create or simulate data sets required for their use. This may sound a bit odd for data analysts for whom data is a collection of rows and columns which is obtained through meticulously defined methods with sacrosanct efforts. For analyst, every thing starts from data. Analysts expects data to be precise and also accurate but it is not true for programmers. For programmers, the data is just bits and bytes. Analysts need to care context or scenarios to which data is linked to. For instance, the variable *age* has certain sense to analysts. However, for a programmer it is just a collection of numbers may be an array, a vector, or at most a data matrix. Programmers try to see these variables as collection of integers or decimals or anything else from the number system. The aim of the programmer is to see the data fits to a model defined through code or *vice versa*. That kind of effort is referred to data science.

A simulation is the imitation of the operation of a real-world process or system over time. Data simulations are highly useful for testing models. Usually simulations required to use models. Statisticians refer them to probability distributions. To start with, a number series can be modeled as linear or non-linear depending on the context. In the same fashion every probability distribution an underlying model in it. For instance, a normally distributed data arise or originate from a process known as "Gaussian". The model for normal distribution is a mathematical representation of that process. As such these models or processes determines the the very occurrence of data points. In this section there are few yet very potential methods that are necessary to deal with few practice exercises given in this text or book.

Following code is available from a companion Github repository at ml-book. The code available from a module called *datasimulations*.

```
import random

def createFactor(cats=['male', 'female'], n=10):
    cats = cats*n
    out = random.sample(cats, n)
    return out

def createVector(start=1, end=10, n=10, m= 0, Type=None):
    if Type=='linear':
        out = list(range(n))
    elif Type=='random':
        out = [round(random.uniform(1, n)*m, 0) for i in
                                            ↪ range(n)]
    else:
        out = [random.randint(1, n)*m for i in range(n)]
    return out
```

There are two methods in the above modules and these methods are plain Python functions (defs). These functions can be used to simulate multivariate data sets using following procedure.

```
>>> from datasimulations import createFactor,
                               ↪ createVector
>>> import random
>>> createFactor(['yes', 'no'], 10)
['no', 'yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'no
                               ↪ ', 'no']
>>> createVector(1, 10, m=100, Type='random')
[220.0, 621.0, 373.0, 344.0, 638.0, 361.0, 292.0, 499.0,
                               ↪ 308.0, 951.0]
```

Finally, creating data sets is very easy using *pandas* package. Following code snippet shows the procedure to create multivariate data set with three different data variables known as *age, education, gender* and a dichotomous target variable called *target*. All these variables are defined as columns in a data set called *df*.

```
age = createVector(1, 10, m=10, Type='random')
age = [round(i, 0) for i in age]
education = gender = createFactor(['primary', 'secondary'
                               ↪ , 'higher'], 10)
gender = createFactor(['yes', 'no'], 10)
target = createVector(1, 2)
df = pd.DataFrame({'age': age, 'gender': gender, '
                               ↪ education':education, '
                               ↪ target': target})
>>> df
    age gender  education  target
0  73.0     no    primary       2
1  64.0     no    primary       2
2  98.0    yes     higher       1
3  58.0    yes  secondary       2
4  14.0     no    primary       1
5  13.0     no    primary       1
6  51.0     no     higher       2
7  30.0    yes    primary       2
8  30.0     no  secondary       1
9  86.0    yes  secondary       1

>>> df['age'].mean()
51.7
```