Machine Learning – a few useful things to know

Print date: 21 September, 2021

Author: Berton Koen
Prepared for: Axians Summer School

File name: C:\temp\SummerSchool\CourseNotes\SummerSchool-Handouts-20210921-V01.docx

Table of Contents

About this document

This document provides detailed information supporting and complementary to the lecture ““ML under the bonnet “on machine learning algorithms.

It is a collation of machine learning theory and implementation of machine learning algorithms in Java or Python.

How to use this document

The document is conceived both as a reference and introduction document. It is composed from a personal vantage and addresses the reader directly.

Who should use this document?

Anyone interested in acquiring a quick insight on machine learning (ML) algorithms in particular the transformation of mathematical models into programming code.

What is the purpose of the document?

There is ample information available on machine learning algorithms. The majority of the documentation focuses either on the information theory of machine learning or on the development of machine learning applications by relying on readily available machine learning libraries. This document attempts to reconcile both the mathematics and mechanics by providing readable source code for each algorithm discussed.

The major objective of the document however is to convey the notion that writing your own machine learning source code is a viable exercise and a gratifying experience.

How this document is organised

These handouts are structured as follows

- The Machine Learning process and data science methods
- Open source artificial intelligence and machine learning tools
- Clustering (K-Means and DBSCAN)
- Classifiers (Bayes, K Nearest Neighbors, Decision trees)
- Regression (Linear regression, Logistic regression, hyperplanes and Support Vector Machines, Neural Networks and Convolution Neural Networks)
- Meta algorithms (bagging and ensemble learning)
- Feature Engineering

Chapter 1. Introduction

1.1. Summary

Machine Learning and Artificial Intelligence are emerging technologies. Innovative and appealing applications, based on ML and AI, have recently become available. For example: medical image processing, fraud detection, product recommendation, speech and face recognition, self-driving cars, personal assistants, etc.

This lecture aims to provide a peek under the bonnet of these advanced applications.

1.2. Ideas conveyed

The handouts develop the ideas on ML in the following way.

The concept of ML and AI is to automate the detection the rules to which a system responds via a learning process. This learning process majorly relies on the availability of enough, high-quality and adequately labelled (or classified) data.

The idea upon which of AI and ML are based is not really a novel one. 'Old-school' statistical regression analysis is a genuine ML system at dates back as far as the beginning of 1800 (Legendre, Gauss, Pearson, etc.). Neural network research started as early as the 1930's (Pitts).

Evolution can be observed in the development of algorithms. Their logic was initially rule based, then statistical reasoning was introduced, eventually leading into machine learning. Currently artificial intelligence algorithms are available.

Two major learning approaches prevail: supervised and unsupervised training.

AI and ML applications often – yet not always – require a substantial amount of data (and processing power). For example, Facebook's visual recognition software uses data obtained via "crowd-sourcing", i.e. photographic material is submitted and is labelled - for free - by its users (and by paid hands).

A model is a simplified representation of the reality. Under the bonnet of AI and ML "lurk" mathematical models capable of learning from data that has purposely been submitted for training. Once the model has been trained, its logic can be applied to process similar types of data either for classification, regression or clustering purposes.

It is important to make the distinction between creating (often also referred to as training) the model and using the model.

The training phase of an ML application requires reliable data. The extraction and preparation of data is an underrated and often overlooked discipline in ML. Data must be representative and should be free of bias. This lecture will provide details on the Feature Engineering process, i.e. activities for gathering and constructing data fit to be used by ML. Typically, this involves normalizing and scaling data.

ML and AI applications are built via a standardized process, in which the training and validation step have a predominant role. This lecture will briefly discuss the CRISP and TDSM methods, which are used by data scientists.

ML and AI requires extensive computing power. The abundantly available processing power is one of the major enablers of the recent evolutions in AI and ML, as well as the vested strategic interests that nations and corporations currently have in AI. Debates on the ethics of AI and ML are ongoing in the public domain and contemporary society. Albeit that these topics are interesting to explore and discuss, this lecture will not dwell on elaborate on them.

1.3. Associated documents

Ref	Title	Author
01	Machine Learning for developers (Packt)	Rodolfo Bonnin
02	Machine Learning in Action (O'Reilly)	Peter Harrington
03	Deep learning: Practical Neural Networks with Java (PAckt)	Fabio Soares e.a.
04	From traditional to Modern AI (IBM)	Kevin Yavuz
05	Predictive modelling (IBM)	Mark Freeman
06	Support vector machines succinctly (internet)	Alexandre Kowalczyk
07	Machine Learning for Dummies	John Paul Mueller and Luca Massaron

1.4. Document history

Version	Date	Comment
01	30 April 2019	Draft version
02	02 May 2019	Published
03	20 September 2021	Enhanced

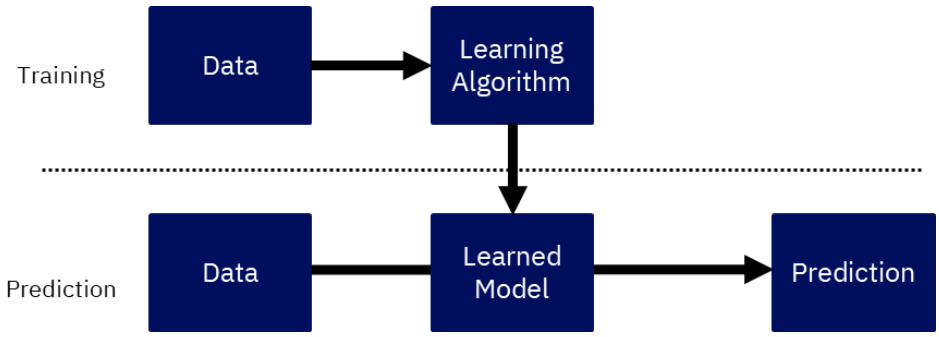
Chapter 2. General concepts

2.1. Definition

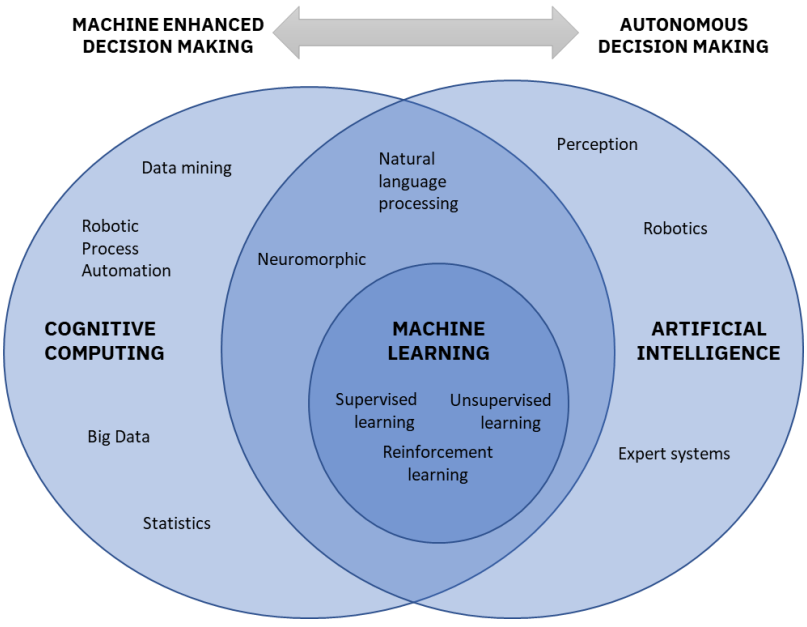
Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence.

Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. [Wikipedia]

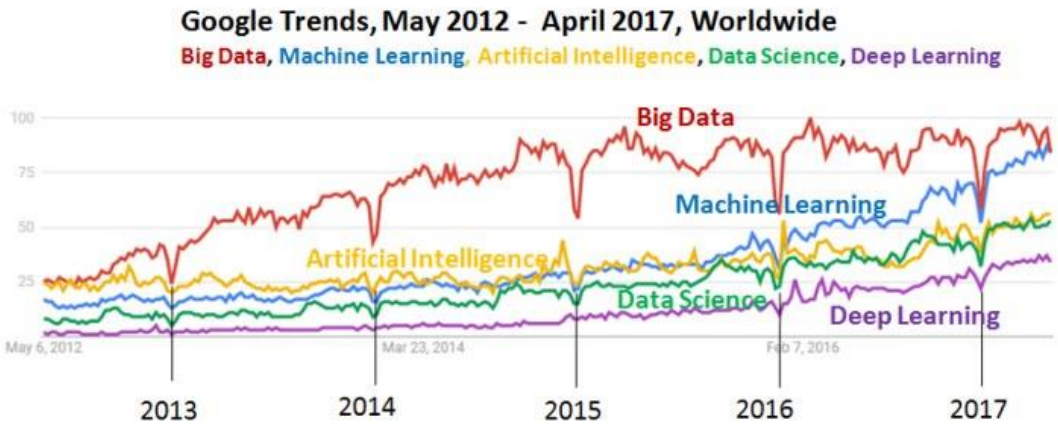
The core machine learning process (or workflow) is the following:



2.2. Cognitive and AI Landscape overview

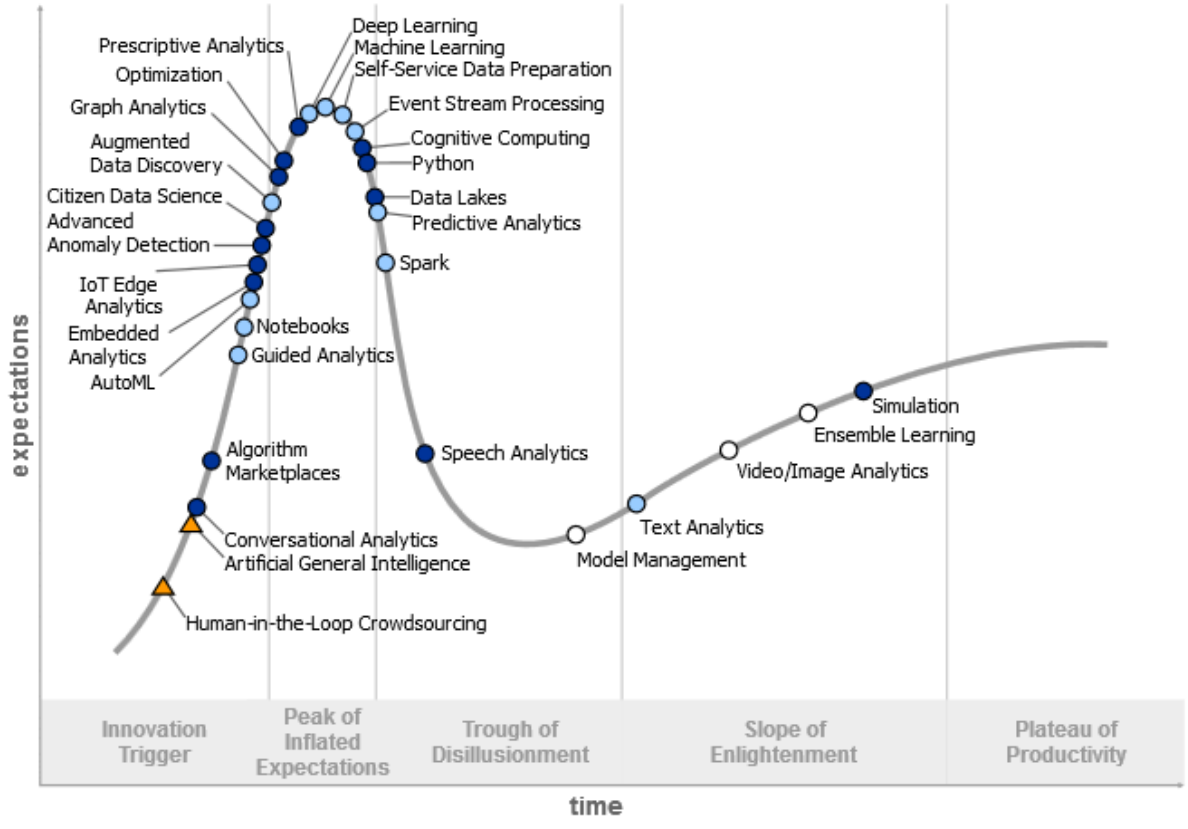


2.3. Market share and trends of ML



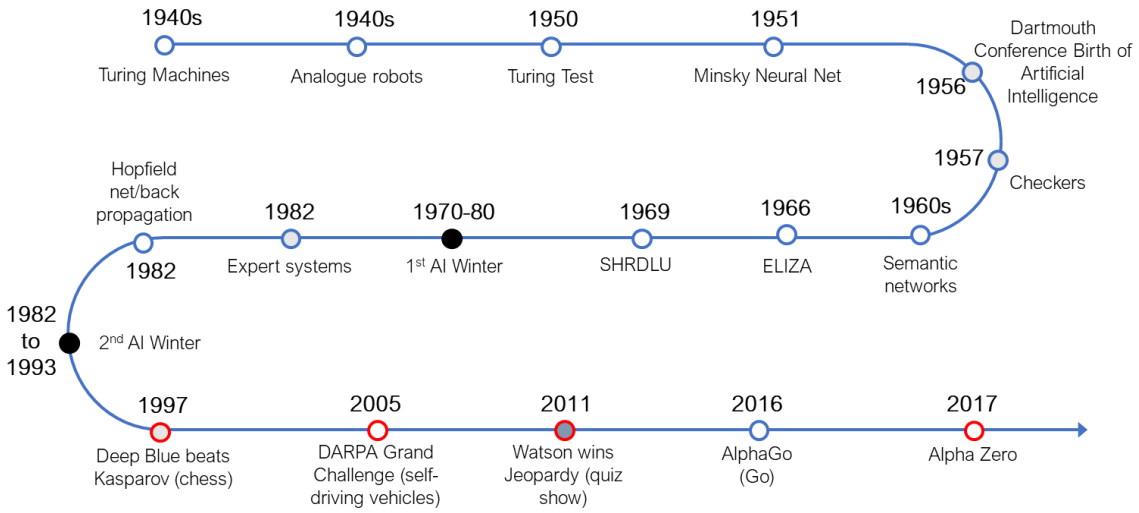
Source : http://www.kdnuggets.com/images/google-trends-big-data-machine-learning-ai-ds-dl-april-2017.jpg

2.4. Hype cycle



Source: Gartner 2018

2.5. A brief history of Artificial Intelligence and Machine Learning



The very beginnings of AI start with the definition of Turing/von Neumann machines as universal computers - could such machines become intelligent? AI was really beyond the reach of the early computers such as Colossus at Bletchley Park and ENIAC, but Turing would go on to define the famous Turing Test.

Around the same time, the first autonomous robots were being built. These were analogue robots that used vacuum tubes or valves as amplifiers to turn signals from sensors into currents that could drive motors. The inventor of these robots was William Grey Walter. He was a neurophysicist who identified the Alpha, Delta and Theta waves. In 1948 he built two analogue robots called Elmer and Elsie to better understand organic brains.

In 1951 Minsky invented the Neural Net – at this stage neural nets were feed forward only (so no memory capability or ability to process a sequence of inputs). IBM would run a simple neural net on an IBM 704 computer in 1956, just before the 'birth of AI' Dartmouth Conference.

1957 – Checkers/Draughts (see previous slide)

1960s – saw the invention of semantic networks as a new way of storing data in a way that more closely resembled the real world (as opposed to tables) – semantic networks are now popularized as graph databases and languages e.g. neo4j, Gremlin, Tinkerpop, SPARQL, OWL/RDF

In 1966 ELIZA – one of the first chatterbots – was invented. The program was used as a 'therapist' and understood nothing of the conversation, but cleverly manipulated the user's input using Natural Language Processing (NLP) to come back with appropriate questions or comments. Such chatterbot technology has been significantly augmented over time and now forms the basis for more elaborate capabilities such as Pandorabots.

In 1969 SHRDLU build on the ELIZA capability with a 50-word NLP program that focused on manipulating objects e.g. "put the red block on the blue one". In some ways it was the first simple interactive fiction as later popularized by Infocom etc.

1970-1980 ELIZA and SHRDLU were very limited, however. They operated in their own finite worlds and extending them to real understanding of language or manipulation of real environments via robot arms proved impossibly hard. At the same time Minsky had doubts about the capabilities of his neural nets. Thus, began the first AI Winter…

Then in the era of shoulder pads and synth pop, AI came back! Expert systems (hand-built rules in rules engines with basic NLP) solved a few real-world problems. In addition, the invention of Hopfield nets (neural nets with back propagation) overcame a previous limitation of neural nets (the lack of any kind of memory) and thus there was a resurgence in AI funding and interest.

Alas, the expert systems (and indeed the experts who provided the rules for them) turned out not to be quite as robust as thought. The systems complex, brittle and hard to maintain. And thus, began the second period of disillusionment with AI and a second winter followed.
Then in the early 1990s, things look up again as compute power enabled Bayesian nets, Hidden Markhov Models, evolutionary algorithms and neural nets to be built on a big enough scale to start making real progress in terms of accuracy. IBM published the first continuous speech recognition system for PCs in 1996. You still had to speak clearly and slowly, but compute power and probability were helping deal with the fuzziness of the real world.

We're now in the era of Big Data and Deep learning where enormous strides been made in the last 20 years – the remainder of this presentation focusses on some key landmark examples.

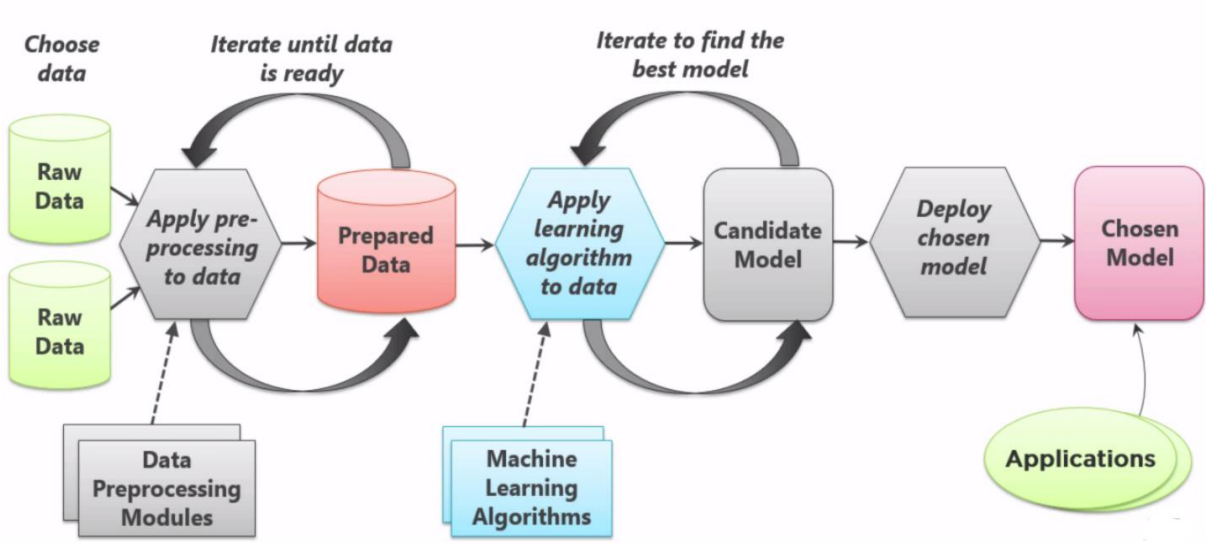
[Source: IBM presentation]

Chapter 3. The learning proces

3.1. Summary

The learning process can be deconstructed in the following major phases

- Defining and understanding a problem to be addressed
- Data retrieval, pre-processing and feature engineering
- Definition, training and evaluating of a learned model
- Understand the results and metrics produced by a model and gather feedback
- (if applicable) select between various models
- Deploy the model that has been chosen.



3.2. Data science methods

CRISP-DM and TDSM are methods used by data scientists for creating and rolling-out machine learning, artificial intelligence or data mining applications. These methods formalize the learning process, i.e. roles, task, deliverables, etc. are defined. Both methods are tool and vendor agnostic.

3.2.1. CRISP-DM

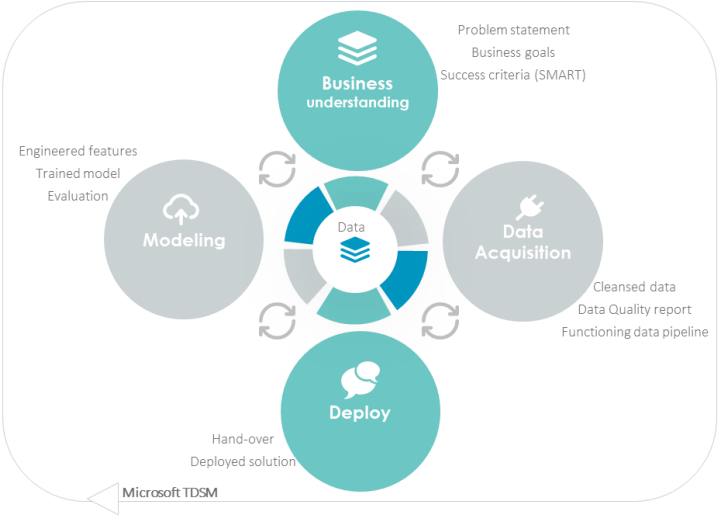
Cross-industry standard process for data mining, known as CRISP-DM is an open standard process model that describes common approaches used by data mining experts [Wikipedia].

You should easily be able to distinguish the learning process phases discussed in the previous section.

NOTE - The modelling workflow in the IBM SPSS Modeler tool closely follows the CRISP-DM steps.

3.2.2. TDSM

Team Data Science Method (TDSM) is the data mining method proposed by Microsoft. It is the method I used on my recent client engagements.



3.3. Understand

This is the least formal phase in the learning process. One attempts to gather an understanding of the (business) problem and (business) variable to predict. This is achieved through interviews, workshops, reading documents (a.k.a. desktop research) or any other technique that is deemed appropriate.

An important objective of this phase is to define the success (or acceptance) criterions, e.g. “we want our e-commerce system to display five products which are often purchased in combination with the buyer’s initial selections within 100 milliseconds after the customer hovers over a picture”. Such criterions tend to be expressed in a SMART (Specific, Measurable, Achievable, Relevant and Time-bound) way.

It is imperative that in “Understand” phase the trusted sources of data are identified. The data extracted from these sources will later be used to train our ML model. It is important to have high quality training data, given that an ML model is leveraging extensively on the training data submitted.

3.4. Data acquisition and preparation

3.4.1. Data acquisition

In this phase we will gather insight on the structure of the data that will be used, we will fetch samples (or the entire) set of data and modify (enhance) the data so that it can be used to train a test our ML model.

It is good practice to always get a “look and feel” of the data, by visualizing the data (e.g. scatter diagrams, histograms, boxplots, etc.), running summary statistics or just peek at it.

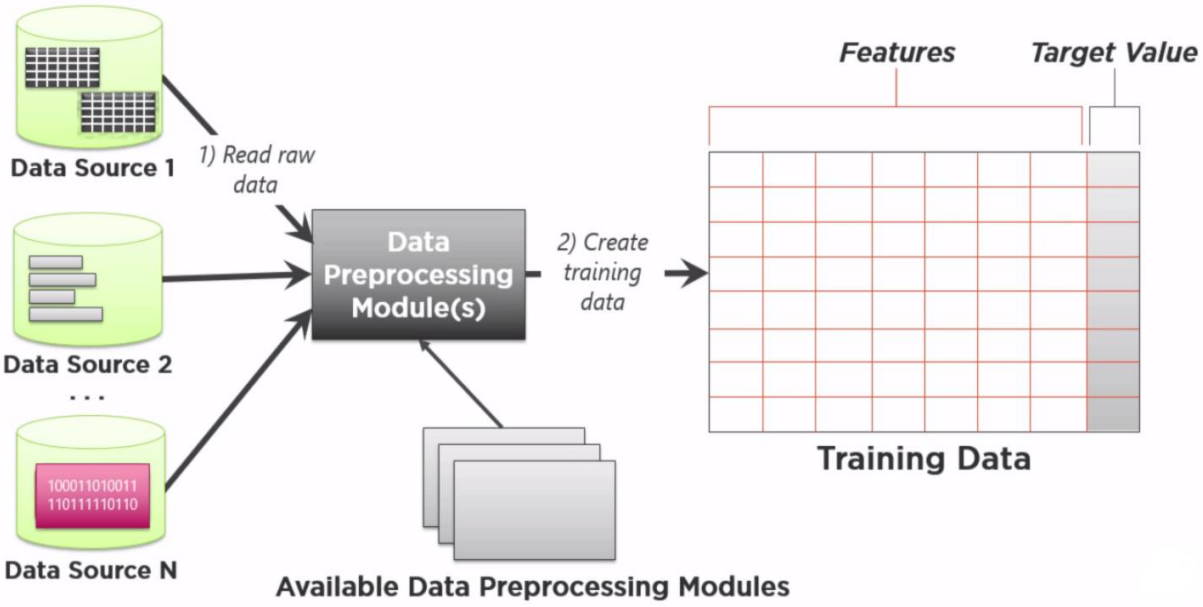
This phase formally relies on data (or information management) disciplines, which are complementary to ML such as: Extraction Transformation and Load (ETL), Data Profiling, Data Cleansing and Feature Engineering

3.4.2. Process

The follows steps and activities are performed during this phase.

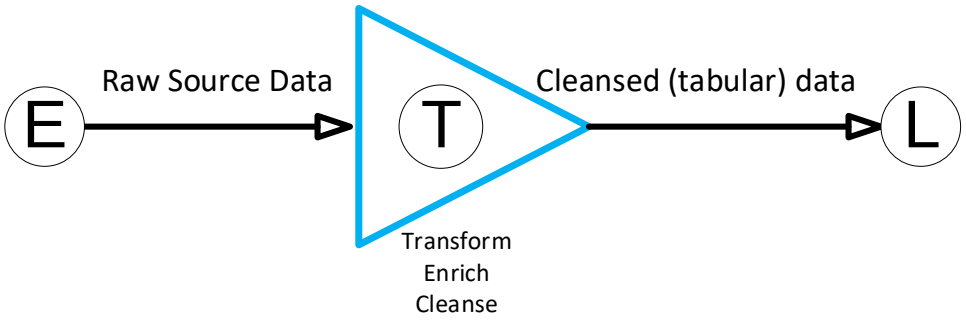
- Ingest the data, e.g. extract and load a copy of the data in the ML environment
- Explore the data (a.k.a. profiling the data)
    - Audit data in a data quality report
    - Visualize and summarize data
- Restructure (transform, reshape, enrich) the data (if necessary)
- Cleanse the data
- Set up the data pipeline, either batch based or real-time, so that the above can be repeated in a consistent manner.

Take away: A feature is a dimension. Features are the variables (hence represented by x) which are used in a model to predict a target value (or label).



NOTE - ML and AI have a specific terminology or jargon. One ingests data – for example - instead of just loading it into a database. Natural Language Processing type of ML "curates" data instead of just pre-processing it.

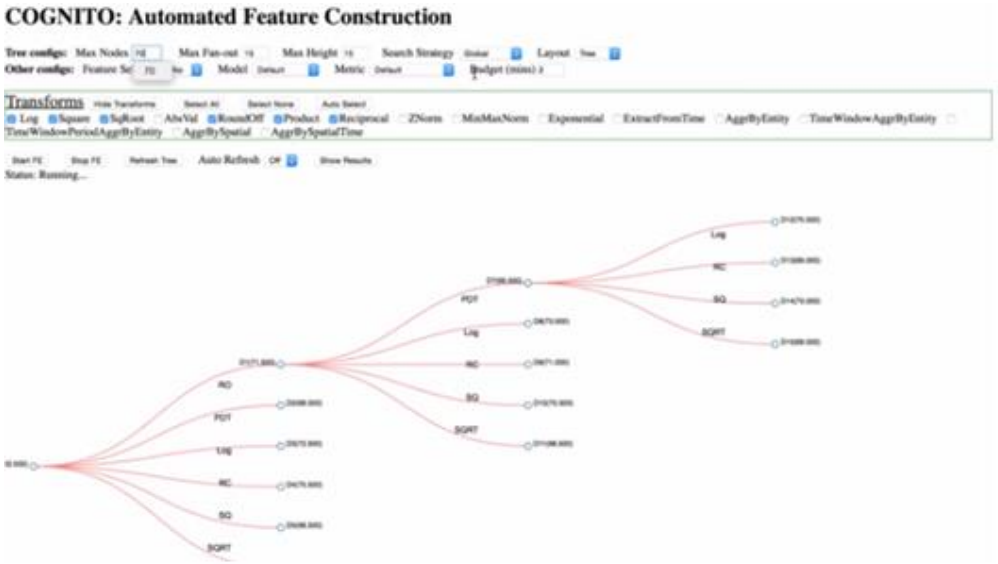
NOTE - The processes defined above constitute a vast area in the IT Information Management discipline. Needless to say, that IBM offers a broad portfolio of infrastructure, tools and services in this area.



3.4.3. Feature engineering

Feature engineering is the process of using domain knowledge, expertise or creative insights to create features (attributes, classes, data) that make machine-learning algorithms work. Feature engineering is fundamental to machine learning and is both difficult and expensive. [Wikipedia]

Automated feature engineering, replacing the manual approach, is therefore an emerging information management discipline. To this purpose, IBM has developed the "Cognito" tool (https://researcher.watson.ibm.com/researcher/view_group.php?id=7500)



3.4.3.1. Feature engineering standard approach

Feature Engineering consists of applying simple transformations that map the unbalanced raw data into standardized format. These are so called affine transformations. The integrity (coherence) of the data is maintained, but the stochastic (random, statistic) properties are improved. This will ensure that the data can be use in an optimal manner by the model or feature transformation (discretization or binning, changing numerical data into classes, changing linear scale to a logarithmic scale, etc.)

Possible mathematical transformations comprise:

- Discretization(binning): turn numerical data into categorical.
- Square, exponentiation or logarithm.
- Scaling: convert variables to comparable scales (inches to centimeters)
- reshaping the data distribution to a format closer to a normal distribution through normalization (harmonization or uniformization)
- Fourier coefficients and wavelets (signal analysis).

NOTE - In addition missing data might need to be detected and - if deemed appropriate – synthesized (e.g. created from scratch or using a default value or by applying a predefined business rule).

3.4.4. Example

The sample data used in these handouts is provided in ARFF format. An Attribute-Relation File Format file is an ASCII text file that describes a list of instances sharing a set of attributes.

ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. The ARFF syntax is very limited and rather self-explanatory. More information to be found at https://www.cs.waikato.ac.nz/~ml/weka/arff.html

In order to give you an impression to what data to be used for training purposes resembles, an extract of the “abalone” sample data has been included. An abalone is a seashell, also known as a sea-ear. The sample data combines several features of an abalone.

```
%https://archive.ics.uci.edu/ml/machine-learning-
databases/abalone/abalone.names
@RELATION abalone
@ATTRIBUTE Length         NUMERIC
@ATTRIBUTE Diameter       NUMERIC
@ATTRIBUTE Height         NUMERIC
@ATTRIBUTE Whole weight   NUMERIC
@ATTRIBUTE Shucked weight NUMERIC
@ATTRIBUTE Viscera weight NUMERIC
@ATTRIBUTE Shell weight   NUMERIC
@ATTRIBUTE Rings          NUMERIC
@ATTRIBUTE Gender         {M,F,I}

@DATA
0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15,M
0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7,M
```

```
0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9,F
0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10,M
0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7,I
0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,8,I
0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,20,F
0.545,0.425,0.125,0.768,0.294,0.1495,0.26,16,F
0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,9,M
0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,19,F
< abridged for legibility purposes>
```

NOTE – There are several sets of coherent and meaningful sample data sets available on the internet which can be used freely to develop and test ML algorithms. Places to start looking for sample data are GitHub, Kaggle.com and the UC Irvine Machine Learning Repository. Alternatively, you can look for sample data provided by public sector organizations through various Open Data initiatives.

3.4.5. Tips and tricks

3.4.5.1. Normalization

In order to normalize a data, it is common practice to apply the following simple transformation.

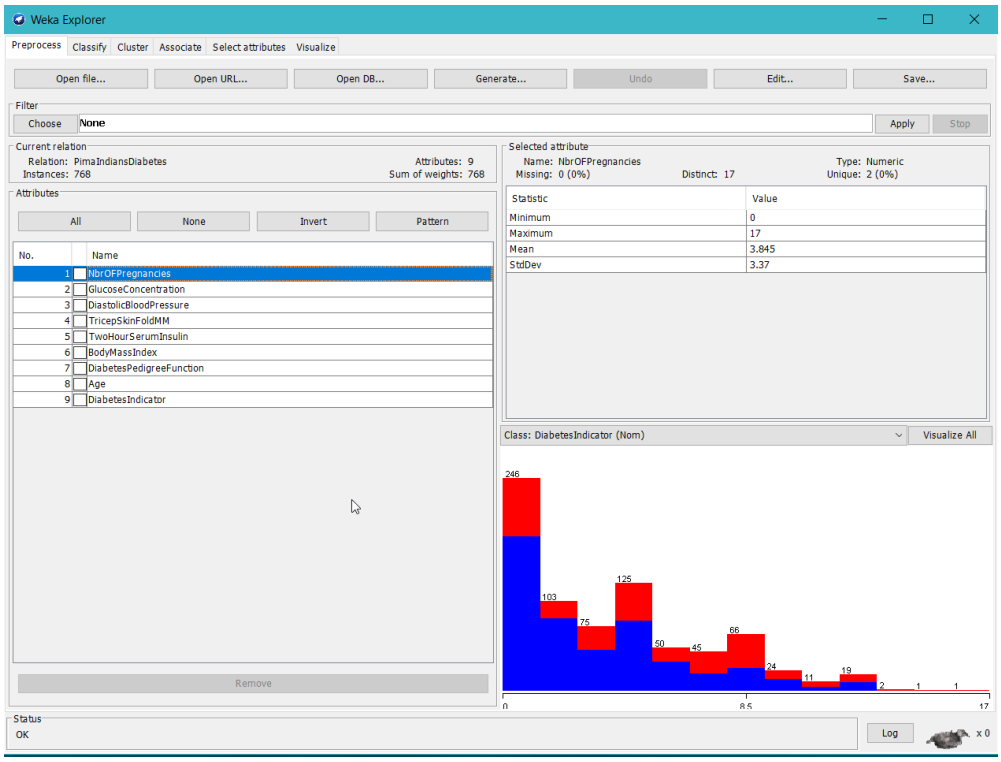
$$z = \frac{x - \mu}{\sigma}$$

Where

- μ is the mean of the values of that particular attribute of sample data
- σ is the standard deviation of the values of that attribute of sample data

For example, the Pima Indians Diabetes data set consist of several medical predictor variables and one target variable (diagnosed with diabetes or not). Independent variables include the number of pregnancies the patient has had, BMI, insulin level, age, weight, number of tricepskin folds, etc. These are all independent features, i.e. there is no (perceived) real correlation across and within an attribute set.

The screenshot from the WEKA tool shows the distribution of the number of pregnancies distributed over 12 bins, you can also see the various other attributes. In order to make the set of attributes more coherent, WEKA enables to normalizes the data (by applying the above formula?) to each measured value prior to testing a model.



3.4.5.2. Distance

Most – if not all – ML algorithms are based on interpreting the distance between data points or vectors. You might remember from your mathematical training that the distance between vectors of dimension higher than three can also be calculated (see formulas further on in this section).

It might be counter-intuitive to accept that even these non-physical distances can be used in a meaningful way. It is important to distinguish the difference between usage of such distances and the actual physical meaning these distances might possibly have. Just keep in mind that N-dimensional distances are a great and wonderful trick.

3.4.5.2.1. Euclidian

In N-dimensional space the Euclidean distance between 2 vectors a and b is defined as follows

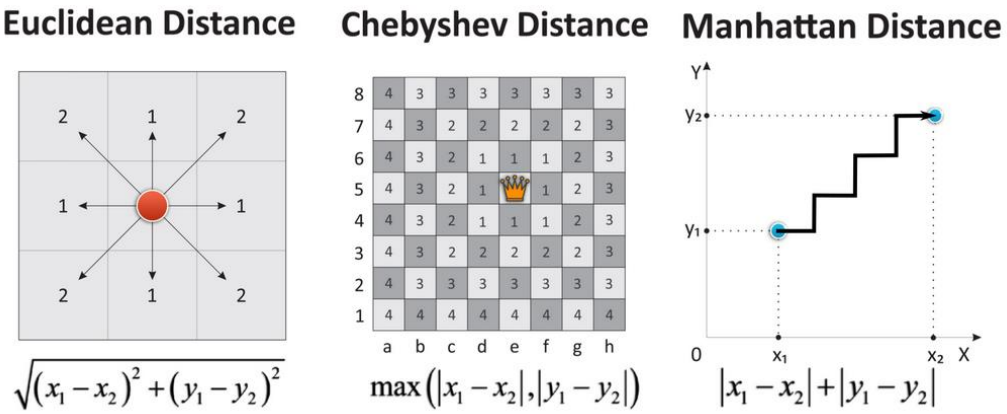
$$euclidian\ d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + .. + (a_n - b_n)^2}$$

$$euclidian\ d(a,b) = \sqrt{\sum_{i=1}^{N} (a_i - b_i)^2}$$

Example: the a vector might contain be data collected on the length of a person and the b vector the matching weights.

3.4.5.2.2. Other types of distances

There are various types of distance formulas: Euclidian, Absolute sum or Manhattan, etc. The Euclidian distance most frequently used.



Source : reference document [01]

Example: x might be data collected on the length of a person and y its weight.

3.4.5.3. Covariance

Covariance is a measure of the joint variability of two variables. If the greater values of one variable mainly correspond with the greater values of the other variable, and vice versa.

$$COV_{xy} = \frac{1}{N}\sum(x_i - \bar{x})(y_i - \bar{y})$$

Where

- $\bar{\bar{x}}$ is the mean of the variable x
- $\bar{\bar{y}}$ is the mean of the variable y
- N the number of samples

Example: x might be data collected on the length of a person and y its weight.

3.4.5.4. Pearson correlation

The Pearson Correlation coefficient is a measure of the linear correlation between two variables x and y. It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation.

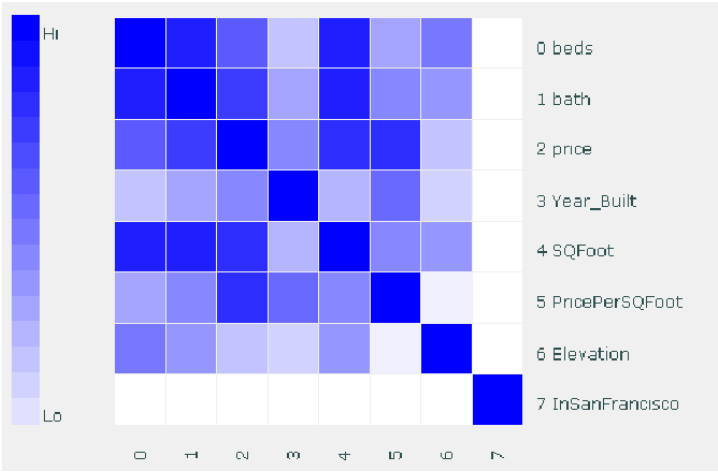
$$R_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

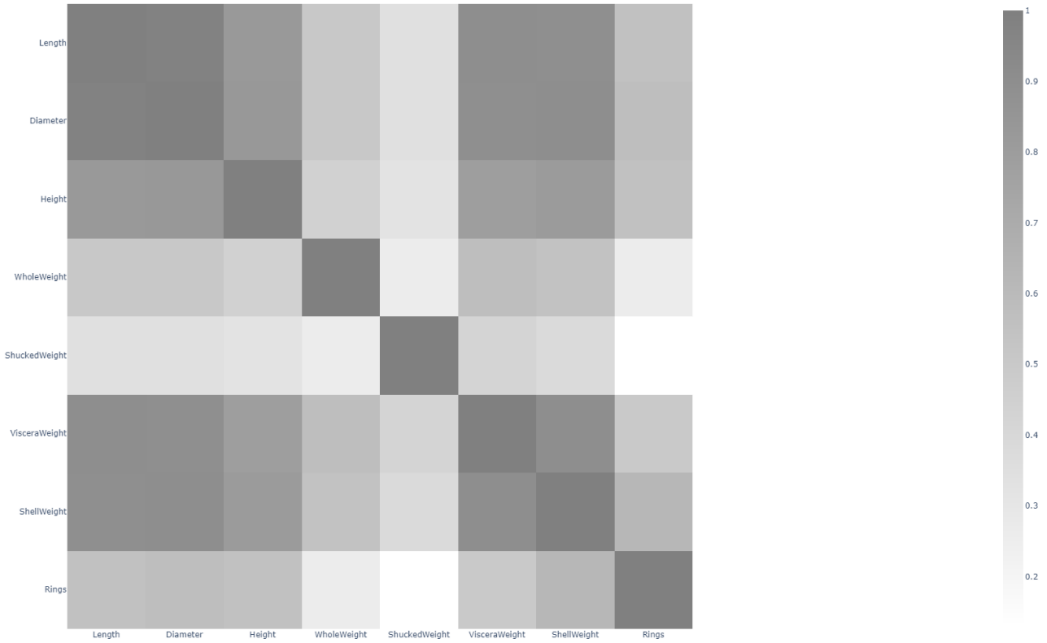
Where

- $\bar{\bar{x}}$ is the mean of the variable x
- $\bar{\bar{y}}$ is the mean of the variable y

- The denominator is actually the multiplication of the standard deviation of x and standard deviation of y

The Pearson Correlation coefficient enables us to create correlation diagrams. When you assemble a feature set you might want to remove features which have a strong correlation. Below the correlation in the San Francisco housing sample data set (created by cbrTekStraktor) and the Abalone sample data (crated by python/plotly)





3.4.5.5. Dimensionality reduction

The training effort of a model is greatly impacted by the number of dimensions (or features). It is therefore common practice to reduce the number of features to a limited number.

The covariance matrix enables to identify features (or dimensions) which are correlated. In order to reduce the training effort of model one can reduce the number of dimensions by keeping one of the correlating features.

Principal Component Analysis (PCA) is another approach. The purpose of PCA is to find a set of features which have the highest impact on the performance (accuracy, recall) of a model. The idea is to transform/reduce the set of N features into a smaller number of features, which still result in the same model output. PCA relies on calculating the Eigen matrix, which is a notoriously complex.

In image processing, Haralick features are used to dimensionality reduction. To compute the Haralick features, the image gray-levels are reduced, via a process called quantization. Haralick texture features are calculated from a Gray Level Co-occurrence Matrix, (GLCM), a matrix that counts the co-occurrence of neighboring gray levels in the image. The GLCM is a square matrix that has the dimension of the number of gray levels N in the region of interest.

3.4.5.6. Entropy

A common summary statistic used in Information Management is “entropy” (see your thermodynamics courses). Entropy measures the “randomness” or “chaos” of a set of variables.

The idea is to define “how much information is present in a set of data”.

$$entropy\ (S) = -\sum_{c=1}^{K} p_c\ LN(\ p_c)$$

Where

- S is the sample of data, comprising K different classes, e.g. sunny, overcast, etc.
- p_c is the proportion of times the category c (e.g. sunny) occurs in the sample S

Examples

- Entropy 0: all samples have the same category. So, the lower the entropy the less chaos (just as in thermodynamics)
- Higher entropy means more random (highest value is LN (p_c)
- The entropy of [1,1,1,1,1] is 0
- The entropy of [1,1,2,2,2] is 0,97
- The entropy of [1,2,3,4,5] is 2,3

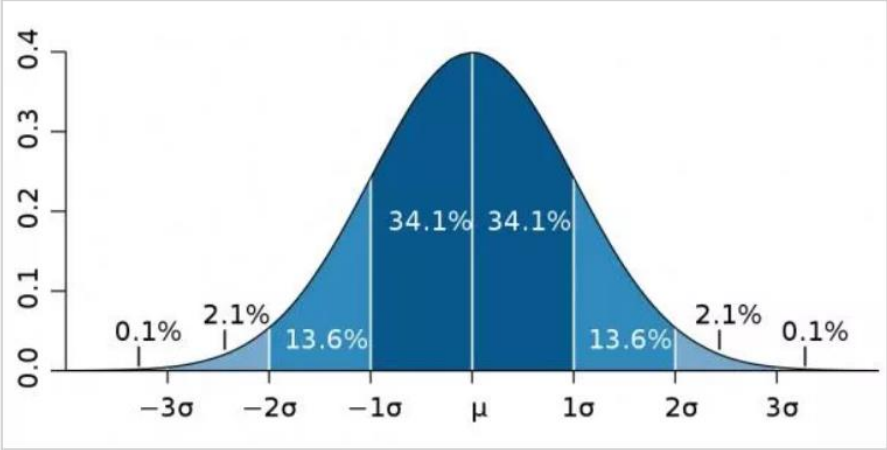
NOTE – The Decision Tree ML algorithm is based on splitting a data set into subsets on a division boundary of the subset (or branch) with the lowest entropy, i.e. increasing the information in a sample by splitting it in subsamples with less variation.

3.4.5.7. Anomaly and outlier detection

[Wikipedia] In data analysis, anomaly detection (also outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data.

When you remove the anomalies form a data set the training will be more accurate.

Observation - In the case of a normal distribution almost 70% (68.2) of the data is situated between the boundaries of the mean and plus and minus the standard deviation. 99% is situated beyond the mean plus or minus 3 times the standard deviation.



Therefore, you can use histograms and boxplots to determine the outliers in particular for a single feature.

DB-Scan (see the section on clustering) can also be used to identify outliers, this also applies to multiple dimensional data sets.

NOTE – scikit-learn comprises a set of routines to perform outlier detection

See : https://scikit-learn.org/0.17/auto_examples/covariance/plot_outlier_detection.html

3.5. Algorithms (model – representation)

3.5.1. Definition

Machine learning is all about defining a model. A model is an abstraction representing and simplifying reality, allowing us to solve real-life problems. In ML we create a model in such a manner that it can be learn from data.

3.5.2. Components of an ML algorithm

There are 3 important components in a ML algorithm

- Model. A model must be represented in some formal language that the computer can handle.
- Evaluation. An evaluation function is needed to distinguish good model from bad ones.
- Optimization. We need a method to search among the model in the language for the highest-scoring one.

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

3.5.3. Learning approaches

There are 3 major learning approaches

- Supervised learning, which uses labelled training data
- Unsupervised learning, which discovers patterns in unlabeled data
- Reinforcement learning via feedback or reward.

The following definitions are sourced from Wikipedia.

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data and consists of a set of training examples. Each training example has one or more inputs and a desired output, also known as a

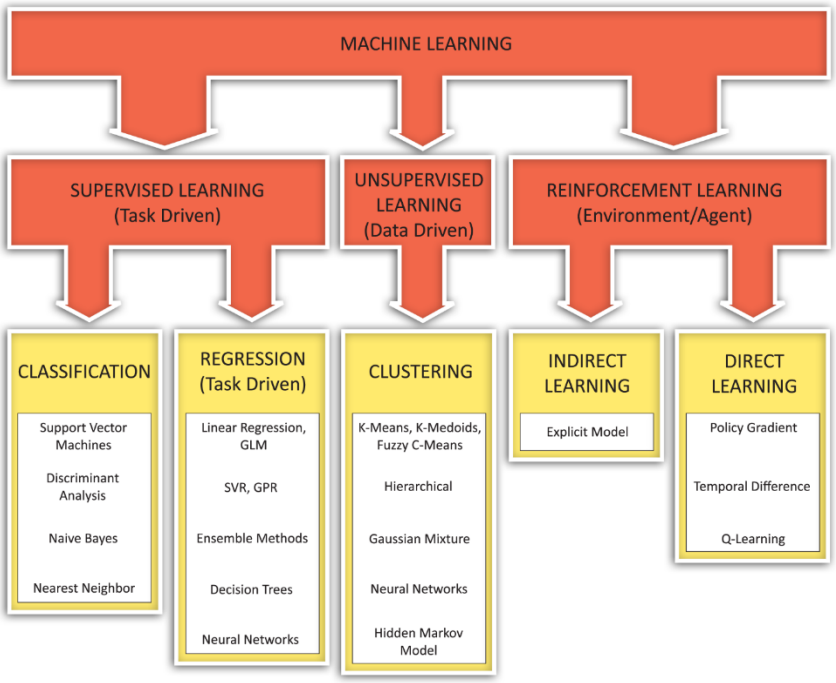
supervisory signal. In the case of semi-supervised learning algorithms, some of the training examples are missing the desired output. In the mathematical model, each training example is represented by an array or vector, and the training data by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. 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.

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 labelled, classified or categorized

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment 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.

3.5.4. Model categories

There are quite a few ML model types available nowadays. Model types are categorized by their learning approach.

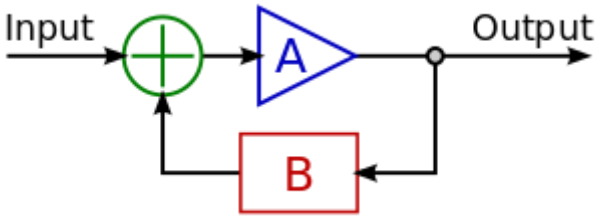


[Diagram source: Machine learning for developers]

3.5.5. Model components

3.5.5.1. Feedback loop

An ML model is a classic control system with feedback mechanism. This is a well-studied topic in the electronics domain. ML leverages on the expertise therein compiled.



The differentiator is that an ML model uses data to train itself. ML models are seldom expressed by a static or deterministic formula, ML models have an iterative and converging nature.

3.5.6. Evaluation

This section briefly discusses the approach to partition sample data and provides formulas that enable to measure the performance of a model (or algorithm).

The evaluation of ML model is a dedicated research domain. Various methods and strategies have been developed. These are beyond the scope of these handouts.

3.5.6.1. Dataset partitioning

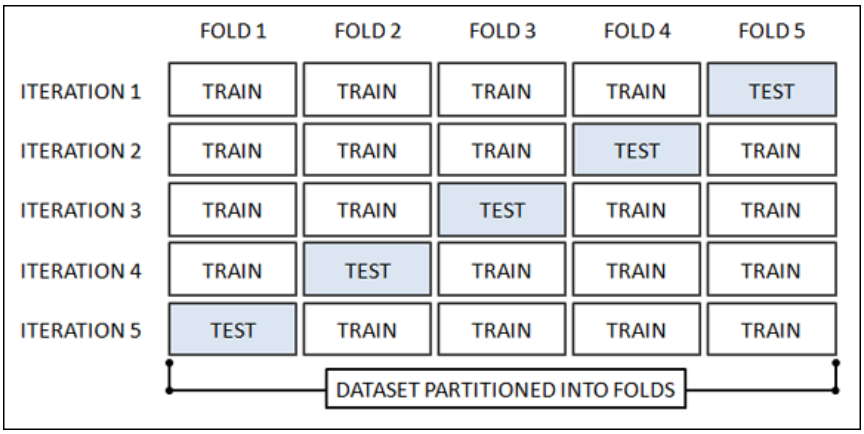
In ML the training, test and evaluation sets of data need to be clearly separated (segregated).

Often the 70/20/10 approach is used, i.e. 70% of the data is used for training, 20% for testing of a model and 10% for cross model testing.

People sometimes refer to in-sample and out-of-sample data instead of train and test data.

The common approach is to select sample rows within a set in a random manner. You should however make sure that the test set distribution is more or less the same as the training distribution, and that the same sequential ordering occurs in both sets.

This is however only one of the possible approaches. Cross-validation is based on a predefined fixed number of folds or portions or parts. It relies on randomly splitting the available data into a number k of folds (subsets) of equal size. Each portion is used in turn as a test set and the others are used for training. Each iteration uses a different fold as a test. The process continues until all the folds are used once as a test set. Every time you keep track of the accuracy and other performance indicators of your model (see next section).



3.5.6.2. Model performance approach

Generalization is the capability to learn from training data and the likelihood that the trained model can be applied to all other data. Test data therefore becomes essential to figuring out whether learning from data is possible, and to what extent.

It is more important to determine the error on the test data set than the error on the training set. The error on the test set shows how the model reacts to new or previously not submitted data. Obviously, a test set does not cover all possible samples. A low test set error is not necessarily an indication of a low test error. Test errors are therefore by "convention" limited to the difference between the expected value and predicted value in the test set.

NOTE - In the same vein. ML practitioners foster the assumption that training and test data is representative for any future set of data. You must be aware that this an assumption might prove to be far from correct. You should therefore define a strategy to verify the conformance of new or fresh data before submitting it to a previously trained ML model.

3.5.6.3. Evaluation of Regression based learning algorithms

This section provides an overview of the common metrics used to calculate the performance (or quality) of a regression model.

3.5.6.3.1. Mean Absolute Error

$$MAE = \frac{1}{N}\sum_{i=0}^{N-1}|y_i - predicted(y_i)|$$

It is mean of the absolute value of the difference between the actual category and the predicted category.

3.5.6.3.2. Median Absolute error

$$MedianAE = median\{|y_0 - predicted(y_0)|, |y_1 - predicted(y_1)|, .. , |y_n - predicted(y_n)|\}$$

3.5.6.3.3. Mean Square Error

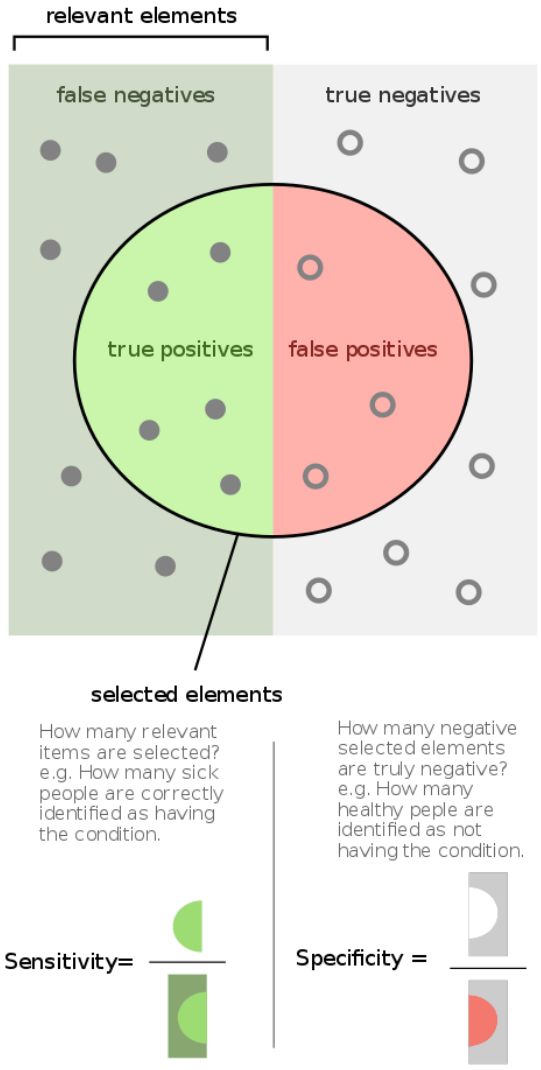
This is the measure which is most often used.

$$MSE = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - predicted(y_i))^2$$

3.5.6.4. Evaluation of classifiers algorithms

3.5.6.4.1. True positives and similar

- True positive (TP) is a positive sample which is correctly classified
- True negative: (TN) a negative sample correctly classified
- False positive (FP). A negative sample classified as positive. Also known as a Type 1 error and sometimes referred to as a false alert or fall-out. It that can be undone without too much impact. Stated as: The positive hypothesis is rejected. For example, when the model which is predicting an illness gives you a patient who is healthy.
- False negative (FN) a positive sample classified as negative. Also known as a Type 2 error. This is an inconvenient error: Stated as: The negative hypothesis is rejected. For example, if you are, looking for an illness and a patient is diagnosed to be not ill, whereas the patient is in fact unfortunately ill.



Source: Wikipedia https://en.wikipedia.org/wiki/Sensitivity_and_specificity

NOTE – the above is a counter-intuitive diagram. It takes a while to grasp it. The Left side are the positive cases, which obviously comprise the true positives which were identified, but also the false negatives which were detected (erroneously).

3.5.6.4.2. Accuracy

The most common and important classifier metric is the accuracy

$$Accuracy = \frac{Number\ of\ correctly\ classified\ samples}{Total\ number\ of\ samples}$$

Or

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

3.5.6.4.3. Confusion matrix

A False Positive and false negative are not the same (an ill person diagnosed healthy is different from a healthy person diagnosed ill). For this reason, a confusion matrix is created.

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

Source: Wikipedia https://en.wikipedia.org/wiki/Sensitivity_and_specificity

3.5.6.4.4. Precision

$$Precision = \frac{TP}{TP + FP}$$

The degree of correctly classified positive samples over the samples that should meet the positive hypothesis.

3.5.6.4.5. Recall

Recall is sometimes called Sensitivity (see the Wikipedia diagram)

$$Recall = \frac{TP}{TP + FN}$$

The degree of correctly classified positive samples over all correctly classified samples.

3.5.6.4.6. F-Measure

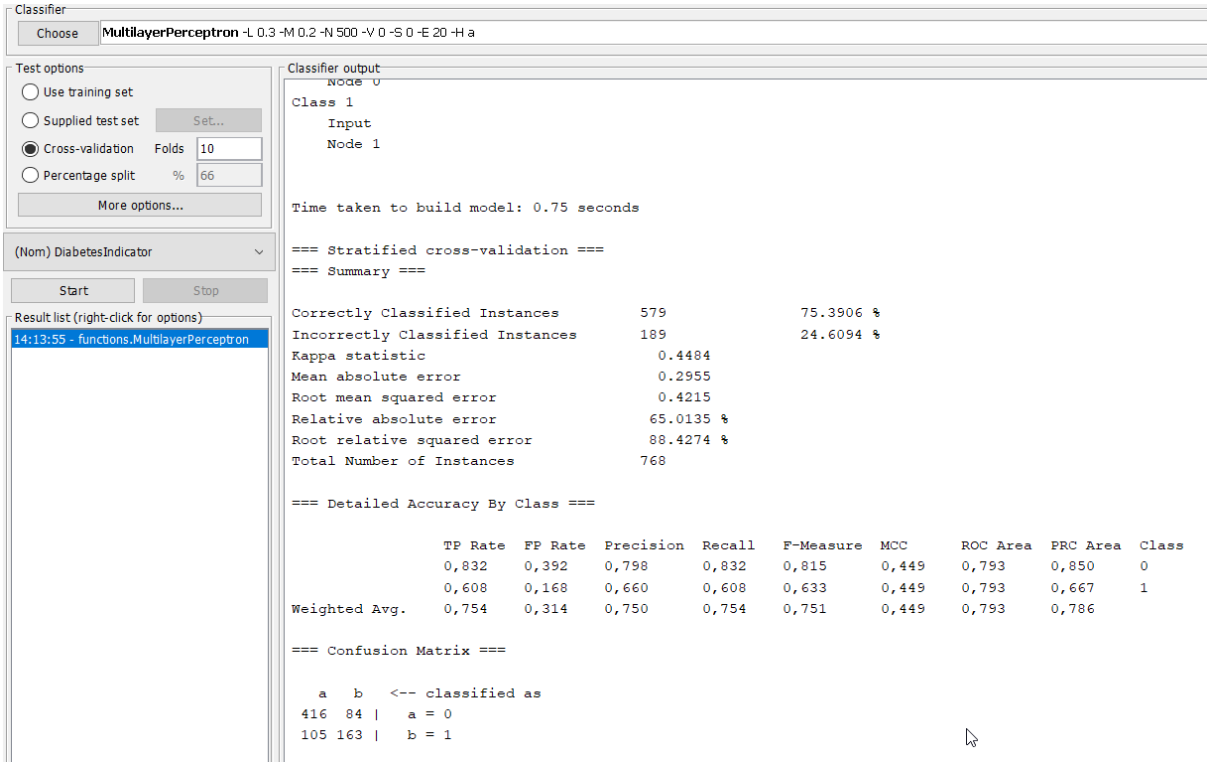
This is the weighted harmonic mean of precision and recall. I have provided the simplified formula

$$F\_Measure = \frac{Precison\,.\,Recall}{Precison + Recall}$$

3.5.6.4.7. Global classification report

The accuracy, precision, recall and F-Measure are often combined into a single report.

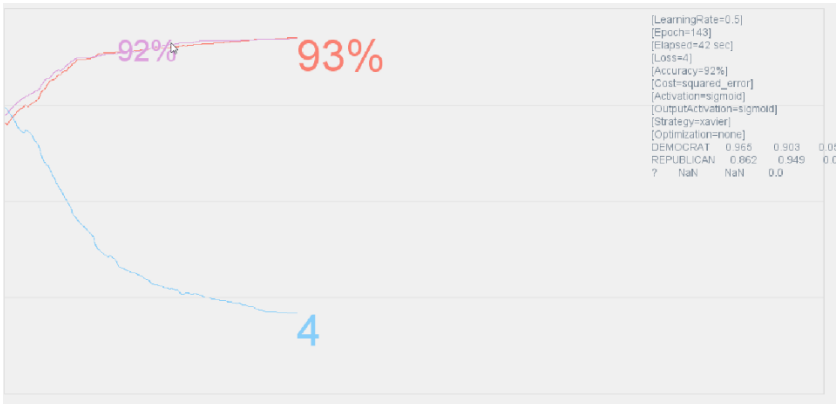
For example, the result of running the perceptron algorithm in WEKA on the PimaIndian test set looks as follows.



3.5.6.4.8. Learning curve

The learning curve is a diagram that depicts the relationship of the number of samples on the accuracy of a model. It is used during the development of a model. Its purpose is to determine the impact of the number of samples on the model. Rule of the thumb is that at "some moment", the learning curve should be stable, i.e. the model becomes independent of the number of samples.

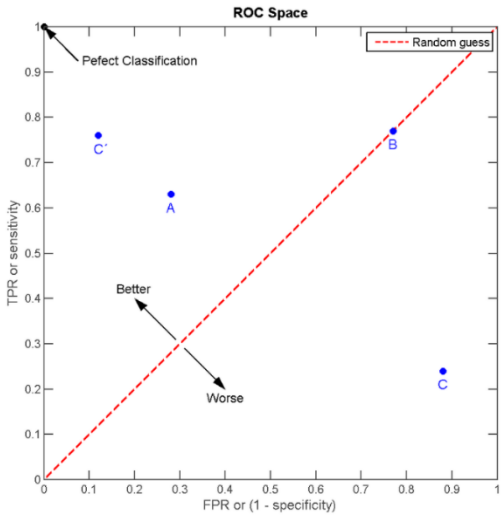
The red line on the diagram below depicts the accuracy (y-axis) per number of epochs or runs (x-axis). The pink line is the accuracy when the model is applied to the test data. The blue line shows the evolution of the loss function. The model predicts the voting behavior of a US Senator.



3.5.6.4.9. ROC Curve

A receiver operating characteristic curve (ROC) is a graphical plot that illustrates the diagnostic ability of a classifier as the model or algorithm. The curve was designed by RADAR technicians during the second world war, for obvious reasons.

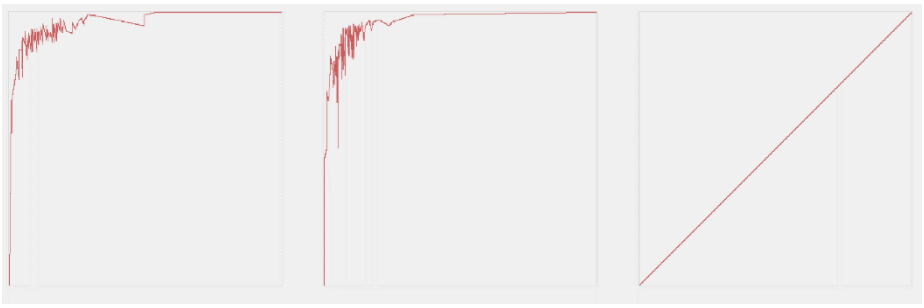
- The X axis is the False Positive Rate: FP / (FP + TN) (or specificity – see the Wikipedia diagram)
- The Y axis is the True Positive Rate: TP / (TP + FN) (so the recall)



The diagram (from Wikipedia) shows the result of four models (the blue dots). You want your models to have many True Positives and few false positives.

NOTE – You can also create ROC curve to assess the impact of the number of samples.

The diagram below shows the ROC curve of the model in the previous section. First segment is the ROC for predicting to vote in favor democratic oriented proposal, the second segment for republican and the third segment covers of proposals which could not be labeled.



3.5.6.5. Evaluation of clustering algorithms

3.5.6.5.1. Silhouette coefficient

This is the ratio of the average distance of a point to all other points and the minimum distance of that point to all other points.

3.5.7. Optimization

Optimization is how you search the space of represented models to obtain better evaluations. This is the way you expect to traverse the landscape to find the promised land of ideal models.

3.5.7.1. Loss function

An important concept in ML modelling is the “Loss”, “Cost’ or “Error” function. The cost function is what truly drives the success of a machine learning algorithm.

The Loss function defines the difference (or digression) of the results of a model and the expected or real value.

An ML model uses the Loss function to minimize the differences in an iterative manner. ML therefore aims to determine “local minima”.

JUMP OF FAITH - Loss functions are therefore chosen in such a manner that they have a convex shape and differentiable. I will skip the math. The bottom line is that is that a local minimum can be found if a function is convex (and conversely a maximum if concave).

Most often used loss functions are

- Mean Squared Error (MSE), in regression and classification ML
- Cross Entropy, in neural networks

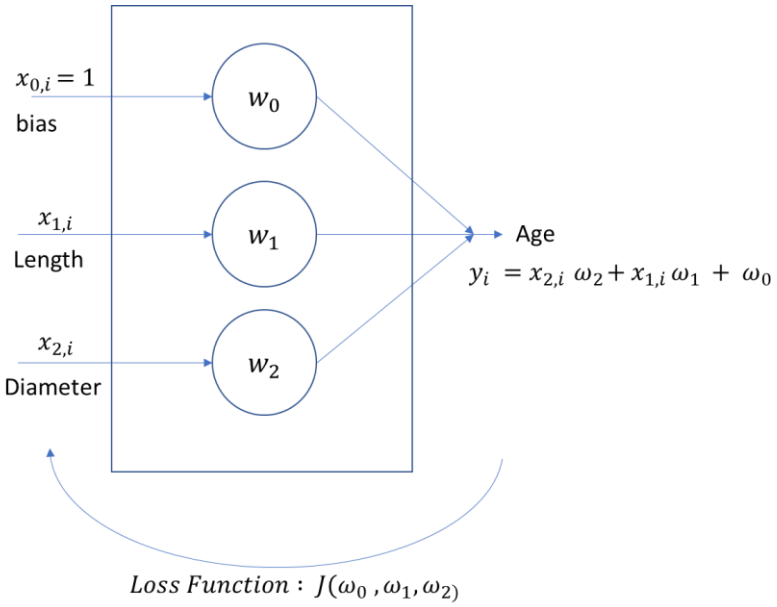
3.5.7.1.1. Gradient descent

A commonly used approach is “gradient descent”. Gradient descent works out a solution by starting from a random solution when given a set of parameters (a data matrix made of features and a response). It then proceeds in various iterations using the feedback from the cost function, thus changing its parameters with values that gradually improve the initial random solution and lower the error.

Assume that we want to create a model that predicts the gender of an abalone seashell based on the height and diameter of the shell via a one-degree equation.

The following picture shows the typical notation of such a model for predicting a CONTINUOS target value. (see end of this section for predicting a categorical result).

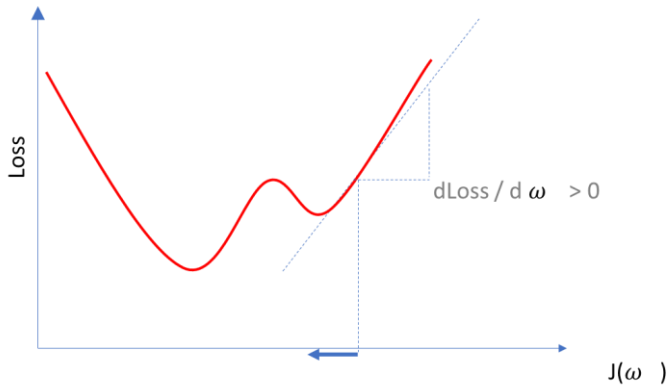
- There N samples (i=1..N)
- The lengths are stored in a vector (of size N) and are called $x_{1,i}$
- The diameters are stored in a vector (of size N) and are called $x_{2,i}$
- There is also a bias, a constant without a variable, but in order to simplify the calculation we introduce $x_{0,i}$, which is always set to 1
- The results (shoesize) are stored in a vector (of size N) and are called y_i
- The model will try to find the value of the constants ω_2 , ω_1 and ω_0 (a bit confusing, these are also called weights) to predict the gender
- The model is expressed as a polynomial of the first degree $y_i = x_{1,i}\omega_1 + \omega_0$




```
@RELATION abalone

@ATTRIBUTE Length          NUMERIC
@ATTRIBUTE Diameter        NUMERIC
@ATTRIBUTE Age             NUMERIC

@DATA
0.455,0.365,0.095,1
0.35,0.265,2
0.53,0.42,4
0.44,0.3655M
0.33,0.2556I
0.425,0.3,7
0.53,0.415,1
0.545,0.425,2
0.475,0.37,3
0.55,0.44,4
0.525,0.38,2
```

We also define a loss function J (which is to differentiable and convex)

This Loss function has 3 dimensions, so in effect it is a 3D-volume. Let's assume that we have a loss function with one dimension. The loss function might look as follows.



The above diagram depicts the result of the loss function for every of the possible weights. The diagram shows a single dimensional loss function, L = f (w_1). For the sake of this discussion let's assume that this particular Loss function conveniently happens to look like a double dipped quadratic curve with 2 valleys.

Stochastic gradient descent attempts to minimize the loss by continuously adapting its input, i.e. the weights on its linear model function (in AI this called the Net function). The key question obviously is how to determine the value by which weights need to be modified.

In essence this is a global minimalization problem. This can be solved only if the pre-requisite condition that the Loss function differentiable is fulfilled, e.g. a simple loss function: expected minus actual ($\hat{Y}$ – Y) won't do the trick. If a function is differentiable, one can determine the direction in

which the input value should be changed. If the derivative is positive, decrease the weight. Contrary, if the derivative is negative, increase the weight.

And given that this a heuristic approach: repeat until the global minimum is reached.

Stochastic gradient descent just repeats the above process until a minimum is a reached.

One should also look for a Loss function of which the derivative is numerically easy to calculate.

Yet this does not provide a solution for the size of the change to be applied to the weight. This is one of the reasons the concept of a Step size (ML) or Learning Rate (AI) is therefore introduced. A learning rate (symbol η) or Step Size (symbol alpha) is just a value that is multiplied to the derivative. The learning rate will define the size of the steps taken whilst descending.

So, for a single weight, the formula is as follows

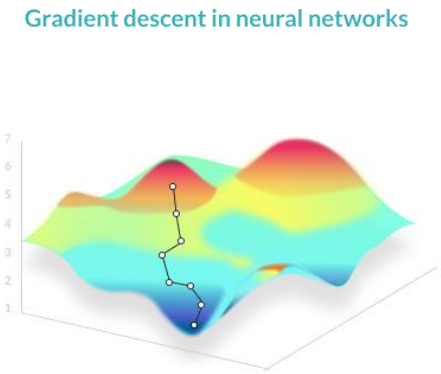
$$w_1' = w_1 - n\,\frac{dLoss}{dw_1}$$

Ultimately, assuming N weights are in a set or vector, we can use the following vector formula.

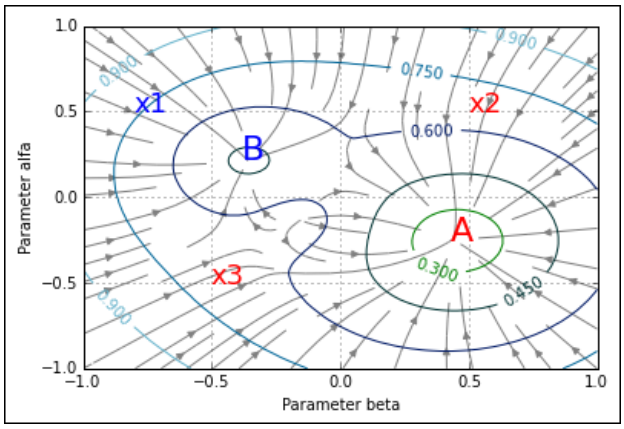
$$\vec{W}' = \vec{W} - \eta\,\frac{d\overrightarrow{Loss}}{d\vec{W}}$$

NOTE – A vector will be symbolized in this document by an uppercase character or a word with a superscript arrow: $\vec{V}$ a matrix by an uppercase: M

The discussion is identical for multi-dimensional loss functions. Below an example of a two-dimensional Loss function, L = f (w_1, w_2), with the added twist of multiple local minima occurring.



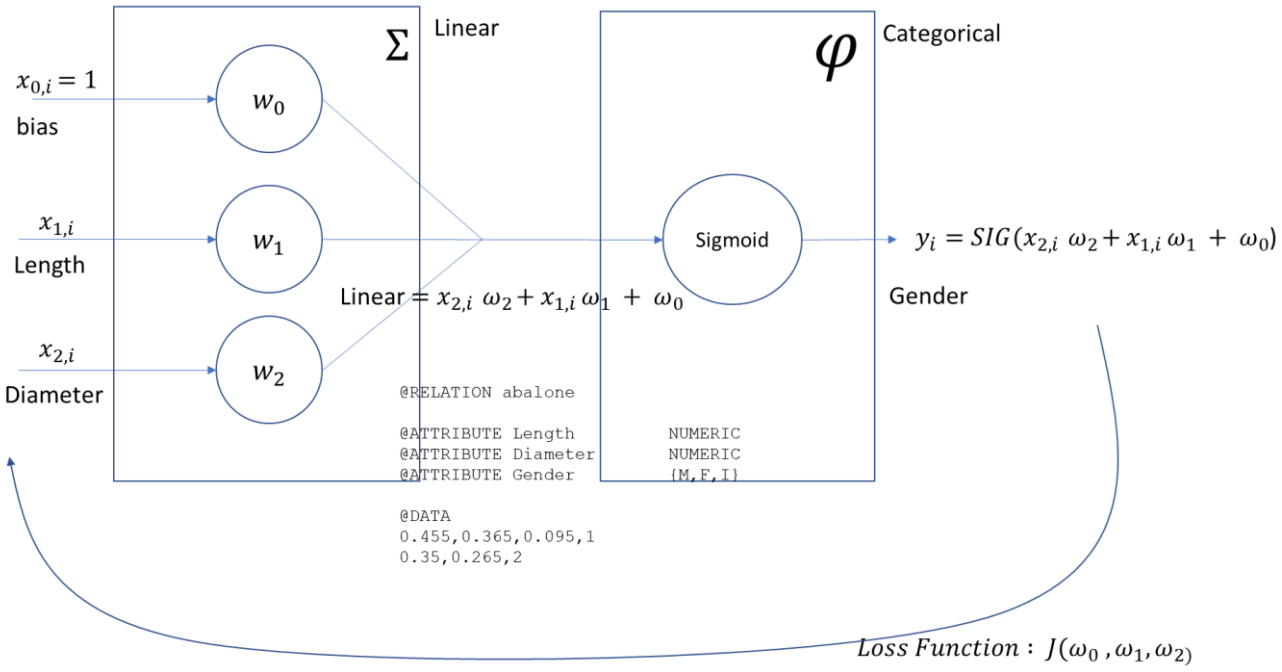
The picture below is a bird-eye's view on a loss function plane (or with a bit of imagination on a mountain landscape with two valleys) shows how a different starting point can make the difference. Starting point x1 ends toward a local minimum, whereas points x2 and x3 reach the global minimum. Fortunately, for us researchers have developed ML algorithms that ensure that a single local minimum can be found.



Source: reference document [07]

The following picture shows the typical notation of such a model for predicting a CATEGORICAL target value. An additional step is introduced in order to transform the linear weighted function into a function that produces a nearly categorical result. This is the activation function (phi). A possible activation function is the Sigmoid function.

The gradient descend approach is identical. See section on Logistic Regression.



3.5.8. Mathematical convenience functions

ADVISE - ML Models are obscure and complex at first sight. You will often see that mathematical "convenience" functions are applied, i.e. expert tricks carefully chosen by mathematical researchers to simplify or approximate/accelerate the calculus. This type of mathematical bravado is challenging and often disconcerting (it certainly took time for me getting used to). The algorithms however are mostly straightforward and (as you will soon discover) can be implemented in the simple procedural programming languages like java, python, etc. Most of the techniques can also be visualized easily.

Chapter 4. Tools

4.1. Summary

This is a short section on tools you can use (at no cost) to gather a hands-downs experience in ML.

4.2. Weka

Weka is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization. It is very intuitive to use. I use it for prototyping and visualizing data.
https://www.cs.waikato.ac.nz/ml/weka/

4.3. Java

Plain old Java is well suited for implement ML algorithms. Java unfortunately has no out of the box constructs or objects for managing matrix or vector based numerical data. You will need to create your own library for managing these components (or leverage an existing one). Java source code also tends to be quite voluminous, just think about the getter/setters.

4.4. Python

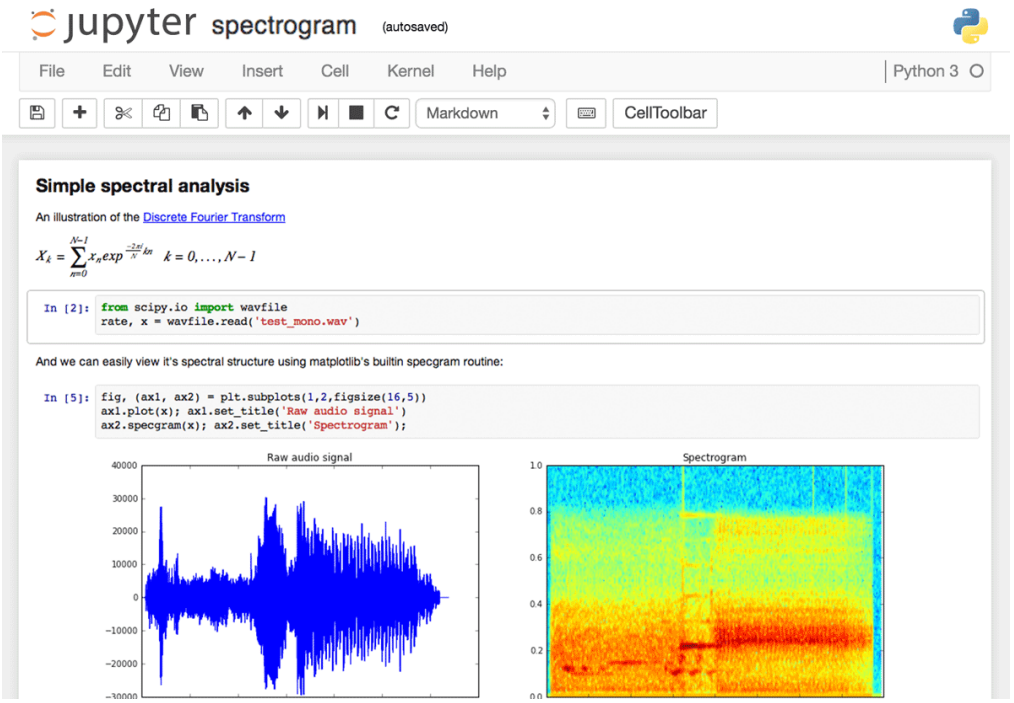
Python is a general-purpose scripting language. It possesses a very simple syntax with great extensibility. Most of the ML practitioners prefer to use Python. Python provides constructs and objects for managing matrix and vector data. Python is a high-level language. Python syntax looks like pseudo-code. This enables to code at a faster rate than Java.

There a various of libraries available for mathematical, statistical, machine learning, data management, etc. For example, scikit (math), pandas (data manipulation), seaborn (visuals) and NumPy (machine learning, statistics, artificial intelligence).

4.5. Jupyter notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter notebooks are documents consisting of intertwined cells of code, graphics, or formatted text, resulting in a very versatile and powerful research environment. All these elements are wrapped in a convenient web interface that interacts with Python.

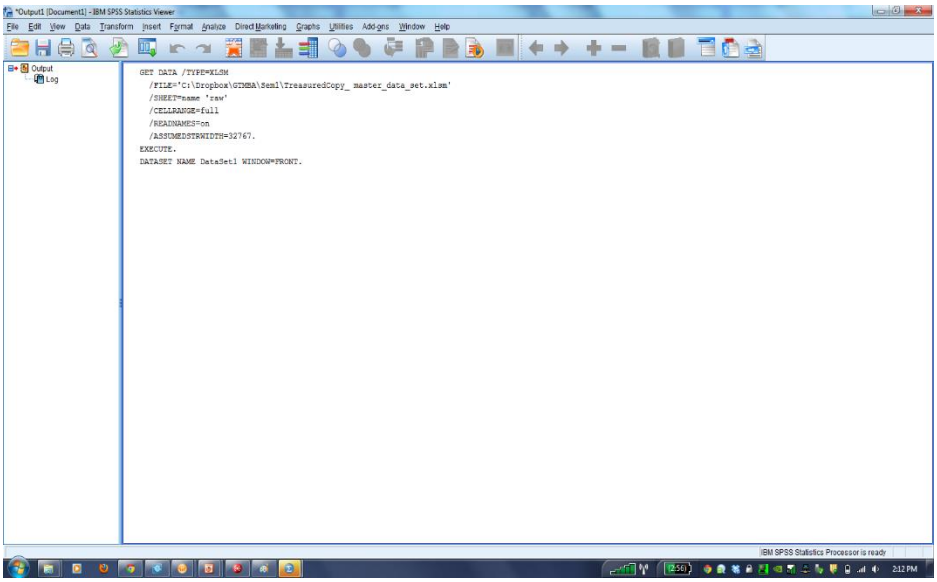


4.6. IBM SPSS

SPSS stands for Statistical Package for the Social Sciences. SPSS is used for survey authoring and deployment (IBM SPSS Data Collection), data mining (IBM SPSS Modeler), text analytics, statistical analysis, and collaboration and deployment (batch and automated scoring services).

Statistics included in the base software:

- Descriptive statistics: Cross tabulation, Frequencies, Descriptives, Explore, Descriptive Ratio Statistics
- Bivariate statistics: Means, t-test, ANOVA, Correlation (bivariate, partial, distances), Nonparametric tests, Bayesian
- Prediction for numerical outcomes: Linear regression
- Prediction for identifying groups: Factor analysis, cluster analysis (two-step, K-means, hierarchical), Discriminant
- Geo spatial analysis, simulation
- R extension (GUI), Python



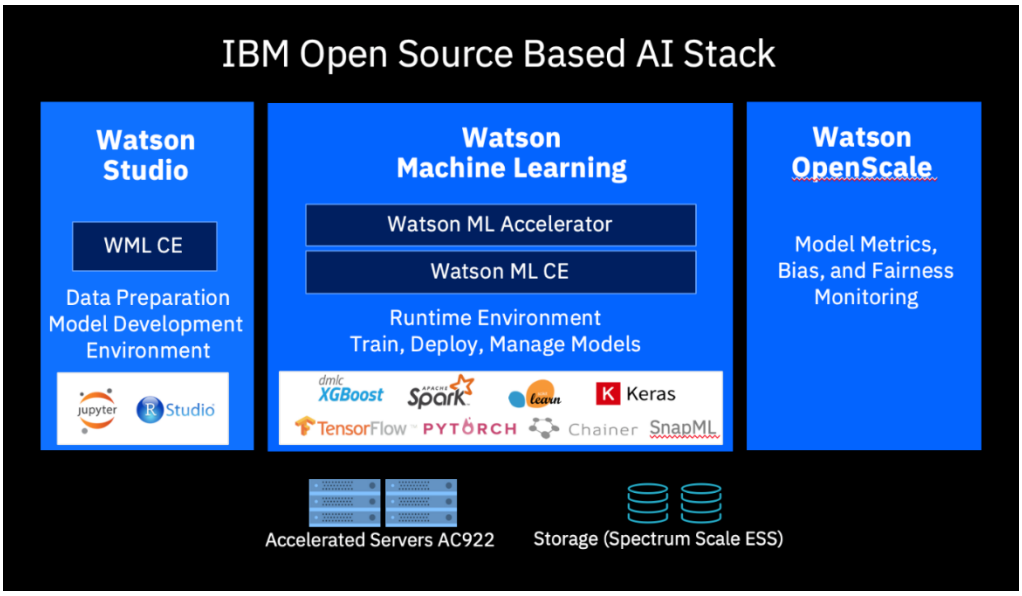
NOTE – SPSS offers academic licenses. See https://developer.ibm.com/

4.7. Platforms

Various vendors provide an ML or AI Platform. Examples are

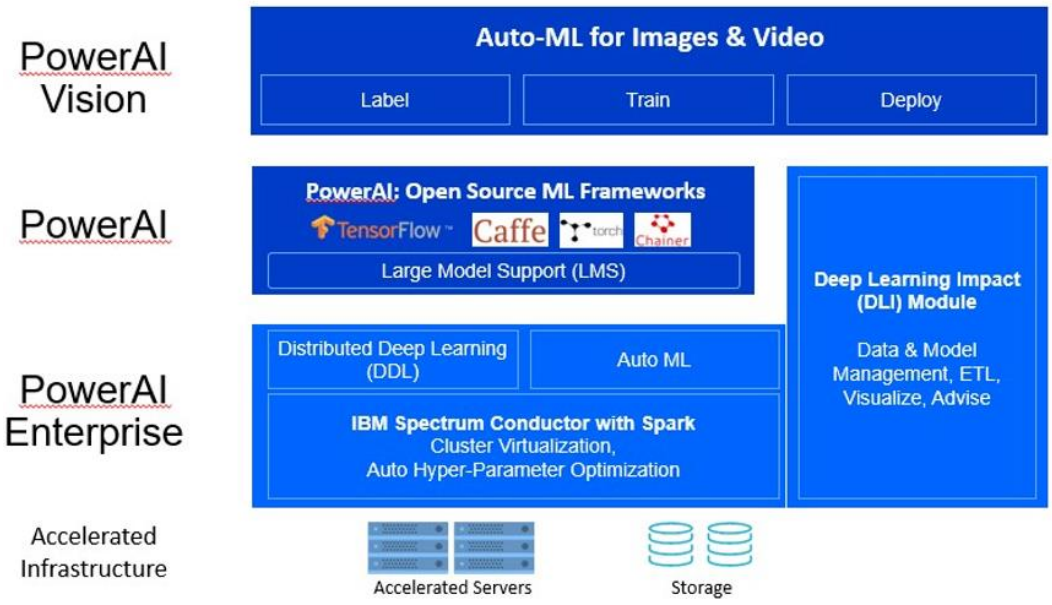
- Keras is open-source and free. It contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML.
- TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks
  NOTE – recent versions (2020) of TensorFlow embed the Keras platform.
- IBM Cloud and Bluemix

4.7.1. IBM Watson AI Stack



IBM is (finally) bringing together all its machine learning tools and putting them under the Watson brand. This integrated software suite gives enterprise data scientists an end-to-end developer tools. Watson Studio is an integrated development environment based on Jupyter notebooks and R Studio. Watson ML is a set of machine and deep learning libraries and model and data management. Watson OpenScale is an AI model monitoring and bias and fairness detection. All this is bundled in Kubernetes and can therefore be made available on IBM Cloud or private cloud infrastructure.

The software formerly known as PowerAI and PowerAI Enterprise will continue to be developed by the IBM Cognitive Systems division (SWG). Power AI enterprise is not open source.



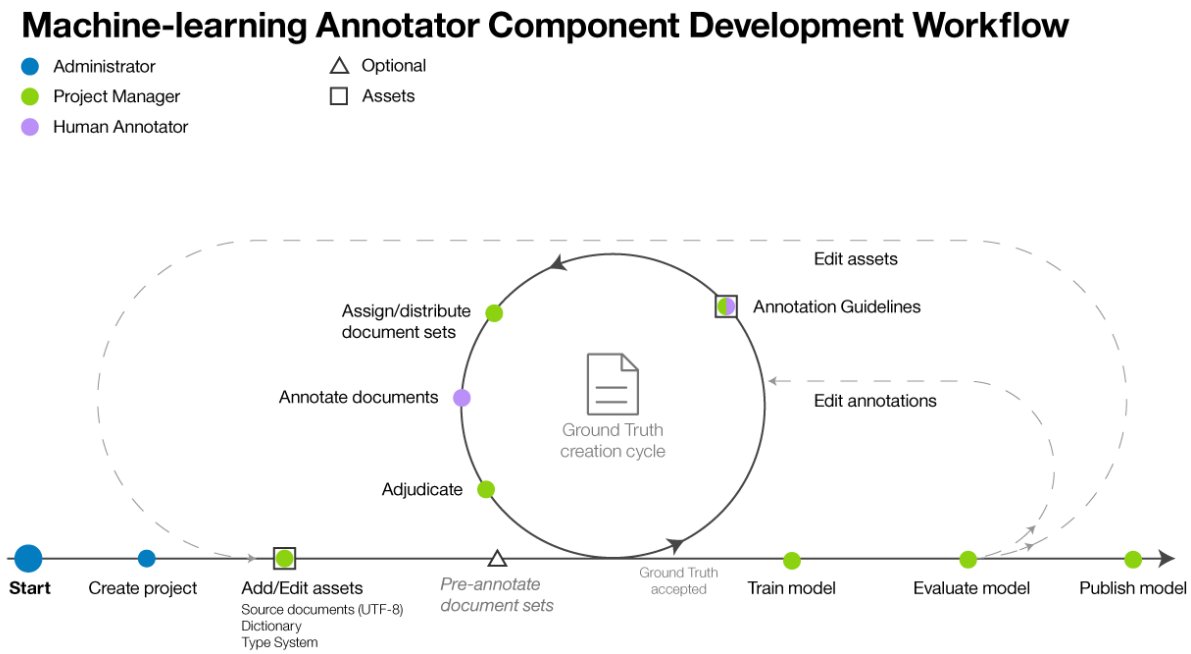
In particular OpenScale is a novel offering. The importance of fair and ethical correct AI is gaining attention.

[Datanews 10 April 2019] Bedrijven zoals Barco, IBM en Microsoft in België gaan de volgende maanden de ethische richtlijnen die de Europese Commissie wil opleggen rond artificiële intelligentie (AI) uittesten op hun praktische haalbaarheid. Technologiefederatie Agoria heeft de richtlijnen mee vorm gegeven en wil bedrijven helpen bij de toepassing ervan https://datanews.knack.be/ict/nieuws/belgische-bedrijven-gaan-ethische-richtlijnen-ai-uittesten/article-news-1450981.html

More: https://www.agoria.be/nl/Denk-bij-AI-ontwikkeling-vanaf-het-begin-na-over-trustworthiness-en-bias

4.8. Watson Knowledge Studio

Watson Knowledge Studio (WKS) provides the tooling to build annotators for entities, relations, and co-references, for training data using machine learning in a supervised learning manner.



What technology is used by WKS

- Natural language (pre-)processing
- Machine learning algorithms (namely maximum entropy model, e.g. decision trees)
- N-grams (an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus.)

WKS is the successor of the former Vivisimo product, a text mining / text analytics application that uses the Apache SOLR text database. The following logic is performed

- Annotator development using machine learning:
- Creation and curation of ground truth data using human annotation (entities, relations, co-references)
- Statistical machine-learning-based information-extraction system, creating a SIRE Model: Statistical Information and Relation Extraction
- Linguistic Phenomena: Statistical parsing and Semantic-role labelling
- Evaluation of recall and precision
- Integration of other annotation techniques and technologies:
- Simple dictionaries, Rules based approaches (REGEX), Existing SIRE Models
- populates the context with discovered entities)
- Discovery (to extract entities and relations)

Regression

Regression models are used for prediction. There are basically two types of regression

- Quantitative (linear or continuous). Real values are predicted (or extrapolated). Examples are the statistical formulas for first and second-degree regression fitting.
- Qualitative (discrete or nominative). Nominative of classes are predicted. This type of regression is a classifier (see next part).

The origins of regression

What we know today as regression was invented by the cousin of Charles Darwin, Francis Galton. Galton did his first regression in 1877 to estimate the size of pea seeds based on the size of their parents' seeds. Galton performed regression on several things, including the heights of humans. He noticed that if parents were above average in height, their children also tended to be above average but not as much as their parents. The heights of children were regressing toward a mean value. Galton noticed this behavior in several things he studied, and so the technique is called regression, despite the English word having no relationship to predicting numeric values.

Source: Reference document [02].

Chapter 5. Linear Regression

5.1.1. Concept

The idea is to determine a function, the "Regression Equation", which defines the relationship between a set of variables (attributes, features or x's) and a predefined set of results (categories or classes). So, the regression equation for a single variable looks like this:

$$y = w_1 x_1 + w_0 x_0 + \xi$$

Or

$$y = w_1 x_1 + w_0 + \xi$$

- w is often referred to as the "regression weight".
- x_0^0 is equal to 1 and therefore omitted
- ξ (xi) s the tolerance, most of the time it is omitted.
- The regression equation is sometimes referred to as the H function.

NOTE – the above formula is a hyperplane of dimension 1 (see support vector machines).

The multivariable version might look as follows (assuming there are M dimensions)

$$y = w_m x_m + .. + w_1 x_1 + w_0 + \xi$$

NOTE – To add to the confusion. The above regression equation is a first-degree polynomial. The algorithm also works for multi-variable multi-degree polynomial functions. Assume a P degree M multi-variable function in the formula below.

$$y = w_{m-1,p} x_m^p + w_{m-1,p-1} x_{m-1}^{p-1} + w_{m-2,p-2} x_{m-2}^{p-2} + etc + w_{2,} x_2^2 + w_1 x_1^1 + w_{20} x_0^0$$

We will iteratively attempt to find the optimal values for w1 and w0. Therefor we define a "Loss" function, so that we can consecutively attempt to minimize the loss. The concept of a loss function is simple to grasp. It is a technique for evaluating how well the regression equation fits the samples in the dataset. If the predictions are off, the loss function will output a higher error value. A comprehensive Introduction to Loss functions can be found here: https://blog.algorithmia.com/introduction-to-loss-functions/

Gradient Descent Regression uses the L2 Loss function, i.e. the Mean Least Square (the squared difference between the anticipated result and the predicted result, divided by the number of samples).

Approach: per sample you determine the difference between the values predicted and labelled result, squared it and summarize all. H is the Regression Equation.

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - H)^2$$

Assume that we have N samples in our set and a single variable x. *(KB – elsewhere you use M for number of samples – apologies)*

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_i x_{1i} + w_0 x_0))^2$$

By convention one assumes X0 to be equal to 1

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_i x_{1i} + w_0))^2$$

NOTE – Just in case you wondered: the result of the loss function is a single real number (there is 1 value per dimension, so in the above the loss is just a single real number).

Recap, at this stage we have

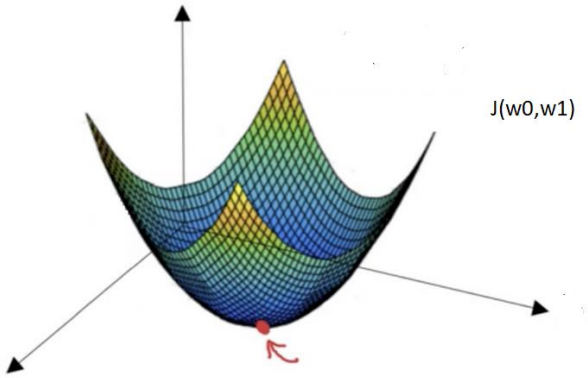
- A known set of samples (or data points) and their anticipated result (label)
- An unknown regression equation that fits the set of data
- A known L2 loss function

A possible approach is **gradient descent**. The idea to find a local minimum for the loss function. Gradient descent determines the "slope" or gradient at a point on the loss function. The reasoning is to descend, i.e. follow the slope (or grade) at that point with a predefined amplitude (or step size). It is like scaling down a mountain via the steepest slope. This step is repeated until the minimum is reached (loss is zero or it is minimal and stable). Fortunately, our L2 Loss function is differentiable and is convex. So, we know there must be a minimum.

For the sake of simplicity: we will limit our regression equation to 1 dimension, first degree polynomial and divide by 2 (just one of those mathematical convenience tricks).

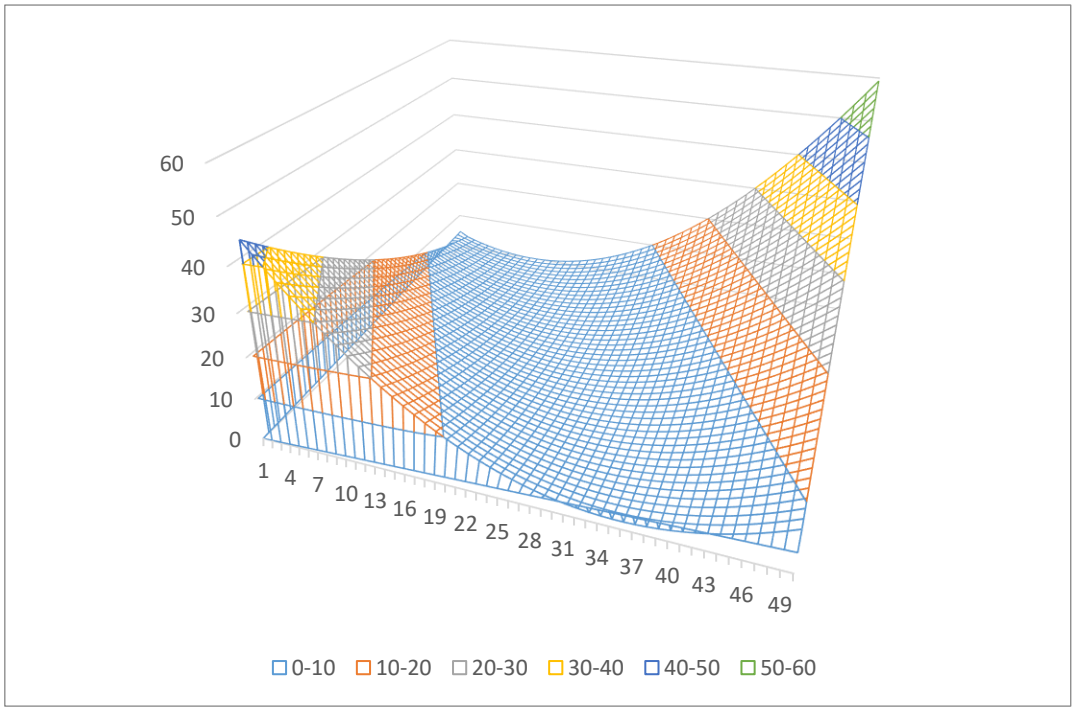
$$J(w_1, w_0) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - (w_1 x_{1i} + w_0))^2$$

The L2 Loss function is quadratic (multiplicative), so its minimum must be a minimum on a convex surface. It might look like the following surface. Picture taken from
https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0



Worked example for y = 2x +3 for a single data point (2,7)

A local minimum occurs at $((2x - 3) - (w_1 1x + w_0)^2$ or $(7 - 2w_1 - w_0)^2$ obviously w_0=3 and w_0=2 form a minimum.



[KB – Apologies the labels on the X axis are incorrect. I need to fix this in Excel]

The gradient is the partial derivative of the loss function for w1 and w0.

$$\nabla J(w_1, w_0) = \nabla \frac{1}{2N} \sum_{i=1}^{N} (y_i - (w_1 x_{1i} + w_0))^2$$

The partial derivative for a first-degree equation a vector of 2 dimensions

$$\nabla J(w_1, w_0) = \begin{pmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \end{pmatrix}$$

A bit of calculus: the partial derivative once for w0 and once for w1 is a vector of 2 elements and looks like this:

$$\begin{cases} \frac{\partial J}{\partial w_0} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - (w_1 x_{1i} + w_0)\right) x^0 \\ \frac{\partial J}{\partial w_1} = \frac{1}{N} \sum_{i=1}^{N} (y_i - (w_1 x_{1i} + w_0)) x_{1i} \end{cases}$$

Mathematicians Introduce a convenience constant alpha at this stage and express the iterative nature (K+1 is the iteration following the kth iteration). This results in the following vector with 2 elements.

$$\begin{cases} w_{0_{k+1}} \leftarrow w_{0_k} - \alpha \frac{1}{N} \sum_{i=1}^{N} (w_1 x_{1i} + w_0 - y_i) 1 \\ w_{1_{k+1}} \leftarrow w_{1_k} - \alpha \frac{1}{N} \sum_{i=1}^{N} (w_1 x_{1i} + w_0 - y_i) x_{1i} \end{cases}$$

The same applies to multi variables regression; so, the general expression is

$$\overline{w_{k+1}} = \overline{w_k} - \alpha \sum \nabla J$$

- $\overline{w}$ is the regression function, expressed as a vector by referring to its weights
- Alpha: the step
- ∇J the partial derivative of the loss function (or gradient), which is a vector of which the number of elements is equal to the degree of the regression equation.

The formula can be interpreted as follows: “The next iteration of the regression function (weight vector) is the regression function of the previous iteration minus the step size multiplied to the sum of all gradients (or partial derivatives) of the loss function” (in the case of Linear Regression the L2 or quadratic loss function).

NOTE – if the model expression is 2nd degree, then the details are as follows (the next section is an example of a second-degree polynomial)

$$w_{0_{k+1}} \leftarrow w_{0_k} - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(w_2 x_{1i}^2 + w_1 x_{1i} + w_0 - y_i \right) 1$$

$$w_{1_{k+1}} \leftarrow w_{1_k} - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(w_2 x_{1i}^2 + w_1 x_{1i} + w_0 - y_i \right) x_{1i}$$

$$w_{2_{k+1}} \leftarrow w_{2_k} - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(w_2 x_{1i}^2 + w_1 x_{1i} + w_0 - y_i \right) x_{1i}^2$$

5.1.2. Worked example

Congrats: you have a gradient descent-based regression formula! So how do we calculate a regression curve using this formula? Transforming this formula into program code is straightforward and looks like this.

5.1.2.1. Program steps

- Initialize the regression weights
- Perform the following steps for a predefined number of cycles
    - For each sample
        - Calculate the result of H using the current weights (one per variable – degree)
        - Calculate the gradient
    - Aggregate the individual gradients and divide by N (number of samples)
    - Update each weight (using the gradient and step size)
    - Calculate the L2 Loss (so you can assess the convergence, i.e. verify whether the delta drops below a predefined threshold).

5.1.2.2. Java implementation

Java implementation for an N degree function

NOTE – double[][] xn= new double[NumberofDataRows][NumberOfWeights] is an array used to speed up the algorithm. It is just a matrix comprising the calculated values for each of the samples' variables, i.e. x_i^n .

So, the feature matrix looks as follows for 3 rows of 1 variable, i.e. a row comprises 1 then the x value and then the square of the x value. (N,3)

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix}$$

The weight matrix is a single row matrix with 3 elements (1,3)

$$[w_0, \omega_1, w_2]$$

The gradient matrix is a single row with 3 elements (1,3)

$$[g_0, g, g_2]$$

The labels (target values) is a matrix of 1 column and 3 rows

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

The calculated results of the model are matrix 1 column and N rows, for the i-th iteration

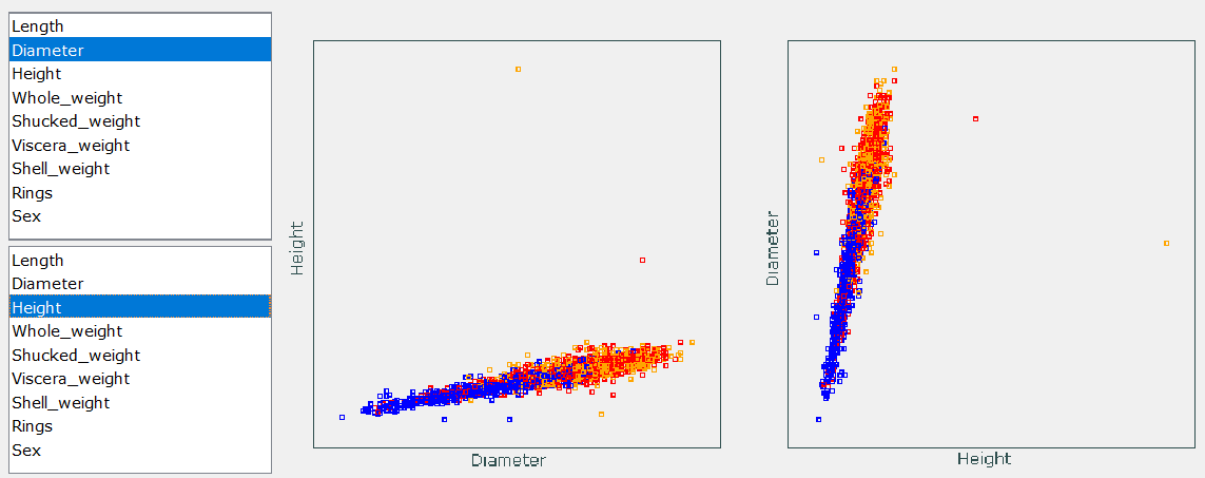
$$\begin{bmatrix} y_{i1} \\ y_{i2} \\ y_{i3} \end{bmatrix} = [w_0, \omega_1, w_2] \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix}$$

Also, the line “double y = data.getValues()[i][1];” might look too restrictive, however the code is conceived in such a manner that it only functions on sample data with 2 features and a bias of 1.

```
	for(int cycle=0;cycle<NbrOfCycles;cycle++)
	{
		// Gradient
		double[] gradient = new double[ weights.length ];
		for( int i=0;i<gradient.length;i++) gradient[i]=0;
		for(int i=0;i<data.getNbrOfRows();i++)
		{
			double y = data.getValues()[i][1];
			//  CURVE => y = w2.(x2^2)  + w1.(x1^1) + w0.(x0^0)  etc
			double yi = 0; // the calculated value of y using the current
weights
			for(int j=0;j<weights.length;j++)
			{
				yi += xn[i][j] * weights[j];
			}
			// calculate the gradient    (y - calculatedY).x power
			for(int j=0;j<weights.length;j++)
			{
				gradient[j] += (y - yi) * xn[i][j];    // postpone 1/N
			}

		}
		for( int i=0;i<gradient.length;i++) gradient[i]=gradient[i] /
data.getNbrOfRows();
		// update weights
```

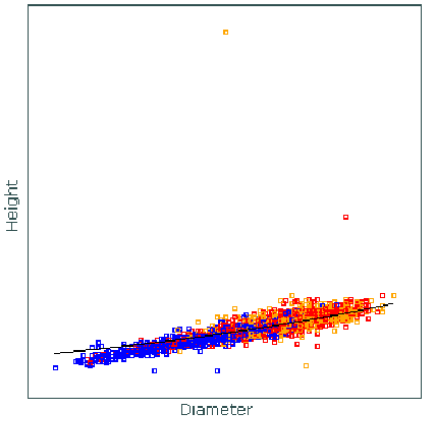
```
            for( int i=0;i<gradient.length;i++)
            {
                  weights[i] = weights[i] + (step * gradient[i] );
            }
            // calculate error with new weights
            double lms = 0;
            for(int i=0;i<data.getNbrOfRows();i++)
            {
                  double y = data.getValues()[i][1];
                  double yi = 0; // the calculated value of y using the current
weights
                  for(int j=0;j<weights.length;j++)
                  {
                        yi += xn[i][j] * weights[j];
                  }
                  lms += Math.pow( (y - yi), (double)2);
            }
            lms = lms / ( 2 * data.getNbrOfRows() );
            lastLMS = lms;
      }
```

Example : Abalon.arff



Regression for 3rd degree curve and single variable (x) regression equation.

Relationship : abalone



Degree : 3
Max Number of Cycles : 10000
Mean Squared Error : 2.895537231673121E-4
Regression fitted curve : 0,06886 (x^3) + 0,11760 (x^2) + 0,16551 x + 0,04613

5.2. Ordinary Least Square Regression

5.2.1. Definition

Another approach to determine a regression equation is via matrix calculation. The Ordinary Least Square is not an ML algorithm per se; it is a discrete formula that uses training data.

5.2.2. Algorithm

Assume the regression equation to be

$$y = w_1 x_1 + w_0 . 1$$

The Least Mean Square formula is used as a Loss function.

$$Loss = \frac{1}{2N} \sum_{i=1}^{N} (y_i - (w_i x_i + w_0))^2$$

The formula can be transformed into matrix calculation by the following formula. Remember that the result of the loss function is a single numeric value, i.e. the result must be a (1,1) matrix)

- X is a matrix of N rows and 2 columns (N,2) (in the above example of a first-degree polynomial)
- Y is a matrix of N rows and 1 column (N,1)
- If you specify W as a matrix of 2 rows and 1 column (2,1), XW will be a matrix of (N,1)
- Why 2N ? In essence you determine the loss function, so by adding a 2 in the denominator, it will level out the result of the first derivative (see below)

$$\frac{1}{2N} (Y - XW)^2$$

$$\frac{1}{2N} (Y - XW) \ (Y - XW)$$

((N,1) – (N,1)) ((N,1) – (N,1)) => (N,1)(N,1) => ?

In order to make the above formula correct for matrix multiplication you will however need to transpose the first segment.

$$Loss = \frac{1}{2N} (Y - XW)^T (Y - XW)$$

(1,2) (2,1) => (1,1)

- X is the **augmented matrix**, dimension is (N,D) or N rows and D columns. D is the degree and N is the number of samples. For example, N samples and a 2nd degree polynomial.

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ .. & .. & .. \\ 1 & x_n & x_N^2 \end{bmatrix}$$

- Y is the label matrix, dimension is (N,1). N rows and 1 column. The label contains the actual y value of the function to be found.

- W is the weight matrix, dimension is (D,1), one row for each degree, i.e. a vector with the number of elements equal to the degree of the regression equation.

Test : ((N,1) – (N,D) (D,1))T ((N,1) – (N,D) (D,1)) => (N,1)T (N,1) => (1,1) qed.

The minimum is to be found where the derivative for W for the loss function is zero

$$\frac{\partial \frac{1}{2}(Y-XW)^2}{\partial w} = \frac{-2}{2N}(Y - XW)X$$

$$0 = \frac{1}{N} X^T (Y - XW)$$

then

$$0 = \frac{1}{N} X^T Y - \frac{1}{N} X^T XW$$

then

$$W = \frac{N}{N} \frac{X^T Y}{X^T X}$$

And eventually we get the Ordinary Least Square formula.

$$W = (X^T X)^{-1} X^T Y$$

CAUTION - There is a prerequisite in order to be able to use the OLS: the matrix $(X^T X)$ must be invertible, i.e. the determinant must be different from zero.

Test: (D,N)(N,D) (D,N)(N,1) => (D,D)(D,1) => (D,1)

5.2.3. Implementation

The follow source code snippet demonstrates the brevity of program when implemented in Python and relying on the NumPy math library.

```
from numpy import *

def loadDataSet(fileName):
    numFeat = len(open(fileName).readline().split('\t')) - 1
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
```

```
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat

def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T
    xTx = xMat.T*xMat
    if linalg.det(xTx) == 0.0:
       print "This matrix is singular, cannot do inverse"
       return
    ws = xTx.I * (xMat.T*yMat)
    return ws
```

Source : Reference document [02]

NOTE - yMat = mat(yArr).T still confuses me. Regardless it is just a matrix with the result class of which the dimensions have been transposed to enable correct multiplication. Matrix multiplication is non-communitative (AB is not BA); so you just need to be careful that the dimensions are aligned. That still leaves you with two possible approaches. One of the approaches will match reality better than the other, e.g. a sample data matrix to store age-weight data of 1000 rows and 2 columns somehow always makes more sense to me than a 2-row matrix with 1000 columns.

Example in ploty, a scatterplot with regression lines for the abalone features (length, shell weight)

Clustering

What is clustering

Clustering or Data Clustering is a way to organize data into groups based on the existing undiscovered natural patterns in a data set.

It is an unsupervised (no dependent variable) data mining methodology.

The goals are to find similarities within groups and maximize differences between groups.

It is most beneficial in a multi-dimensional space, typically 5 to 10 dimensions that can be used to describe the data where visual perception fails.

What is clustering used for

Clustering is used to solve the following problems

- Finding similarities in marketing segments – Determining patterns in customer cohort groups to maximize marketing efforts.
- Product groupings – using product features and sales patterns to segment for store positioning or online sales recommendations.
- Image segmentation - for example using RGB Coordinates to cluster together tokens with high similarity in the feature space.
- Recommender Systems – finds recommendations for likeminded entities and returns recommendations based on those similarities.
- Identifying Crime “Hot Spots” – clustering on geospatial characteristics such as use of physical layout, proximity to services, and land usage.
- Missing values analysis and handling – mean and mode are used from resulting member cluster.
- Detect Outliers – measures the deviation of a record from its peer group and if it exceeds thresholds is considered an outlier.

Chapter 6. K-Means Clustering

6.1. Definition

K-Means is an unsupervised ML algorithm and surprisingly easy to implement. It was an eye opener for me to discover first-hand what such a simple algorithm can achieve.

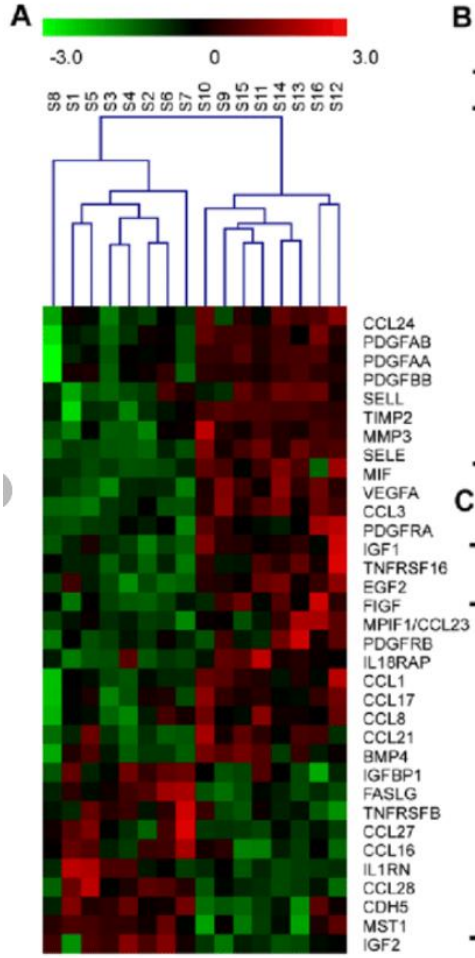
The name is particularly well chosen: the algorithm creates K groups which cluster around a central data element, a.k.a. the centroid.

The centroid is a point (sample, data element, vector, etc.) for which the “distance” to all other points in that data set (or cluster) is the smallest.

6.2. Example

Cluster analysis of antibody-based cytokine microarray in eight healthy individuals with positive Epstein-Barr virus viral capsid antigen (S1-S8) and eight patients with nasopharyngeal carcinoma (NPC) (S9-S16).

A



B

The top 10 up-regulated cytokines in the array

Cytokine	Fold change	*P* value
MIF	5.28	0.005823153
SELE	3.80	8.86286E-06
PDGF-AB	2.87	0.000501708
CCL24	2.68	0.003554355
TIMP-2	2.66	3.99736E-07
VEGF-A	2.48	0.000178298
CCL1	2.41	0.015814008
CCL3	2.32	4.08824E-05
CCL17	2.27	0.022465174
PDGF-AA	2.17	4.6002E-05

C

The expression of cytokines in GSE datasets

Cytokine	GSE12452 fold change		GSE53819 fold change	
MIF	1.94	UP	2.12	UP
CCL3	5.76	UP	2.50	UP
TIMP-2	1.89	UP	1.35	UP
VEGF-A	2.65	UP	1.66	UP
PDGFA	1.48	UP	1.06	UP
SELE	1.02	DOWN	4.98	UP
CCL17	1.00	UP	1.22	DOWN
CCL24	1.10	UP	1.12	DOWN
CCL1	2.27	UP	1.03	DOWN

6.3. Euclidian distance

In most cases, the technique for calculating the Euclidian distance is used

In N-dimensional space the Euclidean distance between 2 vectors a and b is defined as follows:

$$euclidian\ d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + .. + (a_n - b_n)^2}$$

6.4. Algorithm

Step 1. Specify K number of clusters.

Step 2. Initial centers (centroids) are selected at random.

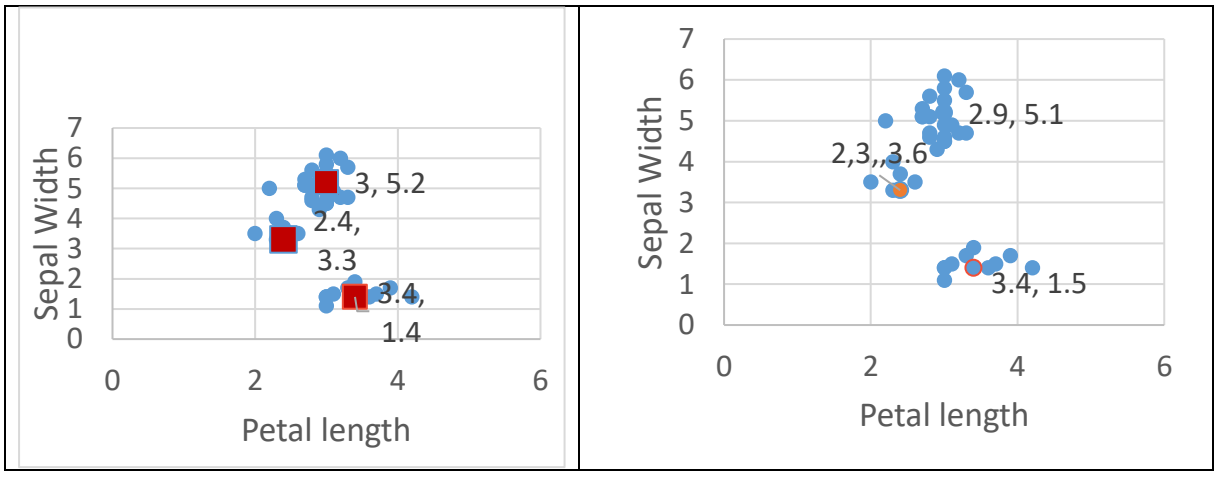
Step 3. For each sample (point) calculate the distance to the K centroids and assign a sample to the cluster of which the distance is the smallest.

Step 4. Centroids are recalculated (find the point in a set which is nearest to all other) once all points have been assigned.

Step 5. Re-estimate the distance of each point to each centroid and move a point to the cluster which is nearest.

Step 6. If there are no points being transferred to another cluster stop, else go to step 4

NOTE – The algorithm is not deterministic: the same set of sample data might converge to a different set of clusters.



Apologies, the pictures relative to the example are not very explanatory. The pictures show K-Means for K=3 clustering on the IRIS data set. You should be able to see how the centroid changes (the minimal distance of a set is displayed).

6.5. Implementation

6.5.1. Java

```
int[] prevCentroids = new int[aantalClusters];
int[] afstand = new int[aantalClusters];
for(int i=0;i<aantalClusters;i++) prevCentroids[i] = centroids[i];
// loop doorheen alle waarden en alloceer
for(int i=0;i<kmset.length;i++)
{
            for(int j=0;j<aantalClusters;j++)
            {
                afstand[j] = centroids[j] - kmset[i].value;
                if( afstand[j] < 0 ) afstand[j] = 0 - afstand[j];
            }
            // zoek kleinste afstand
            int idx=0;
            int min=afstand[idx];
            for(int j=1;j<aantalClusters;j++)
            {
              if( afstand[j] < min ) {
                  min = afstand[j];
                  idx=j;
              }
            }
            kmset[i].cluster = idx;
            // nu de MEAN op die cluster aanpassen -> andere centroid
            int som=0;
            int nn=0;
            for(int z=0;z<kmset.length;z++)
            {
                if( kmset[z].cluster != idx ) continue;
                som += kmset[i].value;
                nn++;
            }
            centroids[idx] = som/nn;
        }
}
```

6.5.2. Scikit-learn

Scikit-learn provides a number of ML routines, including K-Means. The data just needs to be structured as a Pandas DataFrame.

```
kmeans = KMeans(n_clusters=<value of K>, random_state=0).fit( <dataframe> )
```

6.6. Optimizing the number of clusters

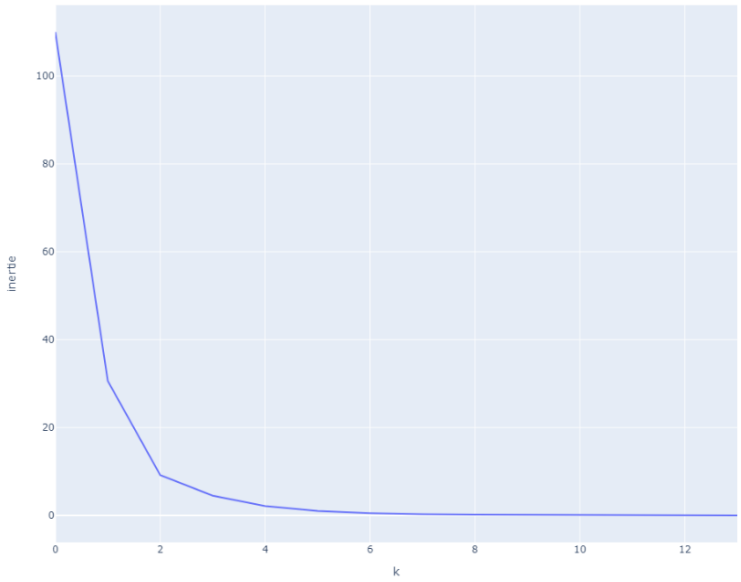
The elbow method is used in determining the number of clusters in a data set.

The method consists of plotting the inertia within the clusters as a function of the number of clusters (K) and picking the elbow (pivotal point) of the curve as the number of clusters to use.

The idea is that at a given number of clusters it is no longer worthwhile increasing the number of clusters.

The inertia of cluster can be defined as the variance of the data in the cluster, i.e. sum of the squared distances of the points to the centroid.

In the following picture the optimal number of clusters would be 2.



NOTE – scikit K-Means function provides the inertia value. You just need to iterate through k and store the result.

```
for k in K:
    km = KMeans(n_clusters=k).fit( <dataframe> )
    InertiaArray.append(km.inertia_)
```

6.7. Vector quantization

K-Means clustering originated from signal processing where it was originally used for vector quantization. Vector quantization is an information reduction technique. Data values are replaced by the mean of the cluster they belong too.

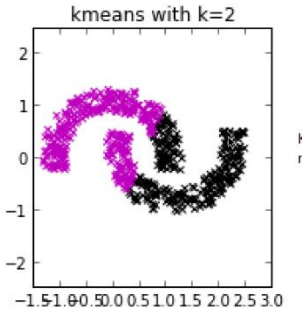
Chapter 7. Other clustering algorithms

7.1. K-Means Hierarchical Clustering

The accuracy of the K Means algorithm depends on the initialization of the centroids. There are various strategies to pick the first set of centroids. The K-Means Hierarchical algorithm solves this issue as follows: Determine K-Means cluster but always for K=2. Then pick the one with the smallest distances and continue.

7.2. Density Based Scan

The K-Means algorithm cannot separate non-convex shapes. In the picture below there are 2 distinct curved shapes, which K-Means cannot not detect.



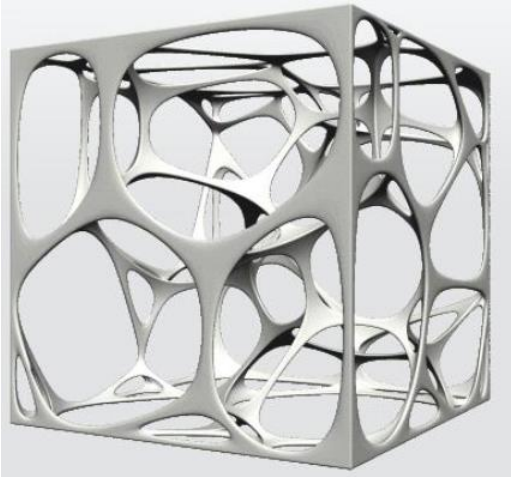
The DBSCAN algorithm solves this shortcoming by defining the minimum number of data points cluster must have within a certain area (density).

[Source Wikipedia] DBSCAN requires two parameters: the neighborhood parameter ε (epsilon) and the minimum number of points required to form a dense region. It starts with an arbitrary starting point that has not been visited. This point's ε-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

If a point is found to be a dense part of a cluster, its ε-neighborhood is also part of that cluster. Hence, all points that are found within the ε-neighborhood are added, as is their own ε-neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

7.3. Voronoi

A Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (seeds) is specified beforehand, and for each seed, there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells. Obviously, this is clustering. Voronoi cells are used for geographically optimization, e.g. optimal distribution of luminance or to create visually appealing structures like the one below.



Reinforcement Learning

Chapter 8. Reinforcement learning

8.1. Summary

This is a section on Reinforcement Learning. It comprises: the concepts of Markov Decision Process, the Bellman formula and Q-Learning algorithm.

8.2. Overview

An overview of the Reinforcement Learning field can be found at https://towardsdatascience.com/reinforcement-learning-an-introduction-to-the-concepts-applications-and-code-ced6fbfd882d

RL is a non-supervised machine learning concept and is based on the idea of a reward signal that the agent uses to evolve to an optimal solution.

Principle: all goals can be described by the maximalization of the anticipated accumulated reward.

Field: RL is about making decision for a move (transition, action, etc.), i.e. when solving mazes, learning to play a game, self-driving cars, etc.

8.3. Markov Decision Process (MDP)

A Markov state is defined by a state in which it is assumed that the current state captures all information from previous states, therefore a transition (or action) on the current state is the same as one would have decided to make that transition by evaluating the entire history of states, i.e. the current state suffices to make a correct decision.

- A Markov process (or Markov chain) is just a sequence of Markov states. Identified by A set of States and a set of actions.
- A Markov Reward process is a Markov Chain, plus a Reward value (a scalar)
- A State Value function, v(S), is the expected long-term value of a state.
- The Bellman Expectation Equation is often used to define a State Value. The Equation takes into account Immediate reward, R(t) and the discounted (gamma) value of the successor state, gamma S(t+1) or s apostrophe

  $$\mathrm{v(s)} = R_s + \gamma \sum_{s'} P \;\; v(s')$$

  Summation of the value estimation (P) of all actions possible in the next state.

An MPD is characterized by the following 5 elements

- S : set of states
- A : set of actions
- P : state transition probability matrix
- R : reward function
- Gamma : discount factor.

8.3.1. Value of an action (Q)

q(s,a) is the value of all actions performed resulting in the current state s.

8.4. Dynamic programming

An MPD can be used to solve dynamic programming problems. There are 2 major uses of an MPD

8.5. Monte Carlo learning

Is used to evaluate the performance (accuracy) of a policy

8.6. Temporal Difference learning

TD learns directly from experience and interaction with the environment. It relies on finetuning a guess of the value function (this is called bootstrapping).

The Q learning algorithm is a TD algorithm. The value of a state is updated (increased/decreased) by the difference of the current value, v(t), and the expected value at t+1, v(t+1)

$$\mathrm{q}(s_t a_t) \mathrel{+}= \alpha \left(r_{t+1} + \gamma MaxArgs_a \, \mathrm{q}(S_{t+1}, a) - \mathrm{q}(s_t a_t) \right)$$

$$\mathrm{V}(S_t) \mathrel{+}= \alpha \left(R_{t+1} + \gamma \mathrm{V}(S_{t+1}) - \mathrm{V}(S_t) \right)$$

So, the value of the next state is estimated using the bellman function (gamma maxargs) and the current value is subtracted to obtain an increment, upon which a learning rate (alpha) is applied.

- Q : q-value or utility of a stet
- S : state (at time t or t+&)
- A: action (at time t or t+1)
- Alpha : learning rate, e.g. 0.7 (higher cause more fluctuation)
- Gamma: discount rate (higher is favor the estimate or utility of next state)
- MaxArgs: maximum of all values over all action in that state

8.7. Q Learning algorithm and gridworld

The Q learning formula is restructured as follows

$$\mathrm{q}(s,a) = (1-\alpha)\mathrm{q}(s,a) + \alpha(R_{t+1} + \gamma MaxArgs_a \, \mathrm{q}(s_{t+1}, a))$$

Typical values

- Alpha = 0.7
- Gamma = 0.99
- R can be used to define a movement cost, i.e. set it to -1
- Epsilon: between 0 and 1

```
GridCellDTO targetCell = null;
targetCell = selectNextAction( explore );

// q(s,a) = (1 - alpha) q(s,a) + alpha ( Reward(t+1) + gamma * Max q(s(t+1),a) )
double maxQSAnext = targetCell.getMaxQSA();
double oldqsa = currentCell.getQSA( lastMovementDirection );
double reward = targetCell.getReward();
double newqsa = (( 1 - alpha) * oldqsa) + alpha * ( reward  + (gamma *maxQSAnext)
);
currentCell.setQSA( lastMovementDirection , newqsa );
```

8.8. Epsilon greedy

A run can be performed by randomly selecting an action (direction) and continue until the target cell is reached. The q values (one per action or direction) are updated as the process evolves. By performing a number of runs, one will reach on optima.

For example, a grid of 3 by 3, starting point left bottom corner and target is right upper corner.

The grid is optimally traverse by following the direction with the greatest q-value.

```
[          7,8110        ]  [          8,9000        ]  [                        ]
[  7,8110          8,9000]  [  7,8110         10,0000]  [                        ]
[          6,7329        ]  [          7,8110        ]  [                        ]

[          7,8110        ]  [          8,9000        ]  [         10,0000        ]
[  6,7329          7,8110]  [  6,7329          8,9000]  [  7,8110          8,9000]
[          5,6656        ]  [          6,7329        ]  [          7,8110        ]

[          6,7329        ]  [          7,8110        ]  [          8,9000        ]
[  5,6656          6,7329]  [  5,6656          7,8110]  [  6,7329          7,8110]
[          5,6656        ]  [          6,7329        ]  [          7,8110        ]
```

Epsilon is a value between 0 and 1. It is used to decide whether the process will explore or exploit its previously gathered knowledge. The decision is just generating a random value and comparing it to the epsilon value. If greater than a random move is made, else the move corresponds to the direction with the largest q-value. (this is performed in the selectNextAction() function in the above Java code)

8.9. Additional info

https://deeplizard.com/learn/video/mo96Nqlo1L8

Also look for the online courses by Jacob Schrum, Southwestern university.

deeplizzard
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3DqhRNvCVVJaA&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=PILQxMbVk2MghibrTeeFyYmQWb7kk5DEFihtQZA_8sw&e=

alpha approach
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3Dmo96Nqlo1L8&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=qNe7749bNwKXuXXwImjXLhwg307qNv1P1MkDEkvw7Bg&e=

python
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3DHGeI30uATws&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=EaB2mvO_HtDXbu8f9u2j-UqwTrsQLbN_uIGCgFO8Wi8&e=

expected values
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3D3T5eCou2erg&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=0uLmt9GlFd3uD-CAizWA_NaTi8g_mJORgJPxlhXPoVY&e=

q value
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3DbHeeaXgqVig&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=lCe5oSxv7WdxnBB-pi1cq_JyiWYHPALy4tMbeKHzrfI&e=

calculation of q value

https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3D1XRahNzA5bE&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=YaGYlsdcD6InuBFKyp9BQNmpr47Lk-F0TAoLk-ZogGM&e=

gamma (learning rate)
https://urldefense.proofpoint.com/v2/url?u=https-3A__www.youtube.com_watch-3Fv-3DXrxgdpduWOU&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=kMTStfUdA_2sHCZBG-Ae91vQMig-R_aOm2otssp8Bzo&m=csILdQ0u1tNX0xbiH_ncBHkOlMke55HypmCvq715C60&s=RlPPdilOPSOAH6HKlktLFU-pSo4oIq3S97tWGSfIdbU&e=

Classifiers

Chapter 9. Decision tree (ID3)

9.1. Summary

The ID3 algorithm is a multi-category classifier. It builds a decision tree from a sample set of examples using an “entropy based” discretization mechanism.

The categories (or classes) created by ID3 are inductive, that is, given a small set of training instances, the specific classes created by ID3 are expected to work for all other sets of data. The distribution of the unknowns must therefore be same as the test cases, otherwise ID3 will misclassify.

ID3 also expects quite a large set of training data.

ID3 is used in rule-induction packages. Some specific applications include medical diagnosis, credit risk assessment of loan applications, equipment malfunctions by their cause, classification of diseases, etc.

Note – A good explanation comprising a worked example on decision trees can be found at https://natmeurer.com/a-simple-guide-to-entropy-based-discretization/

9.2. Origin

Origin. J. Ross Quinlan originally developed ID3 at the University of Sydney. He first presented ID3 in 1975 in his book “Machine Learning”.

9.3. Concept

ID3 uses two concepts

- The Information Entropy (or Information that is potentially available in a sample) to define the amount of information present in a sample
- The Information Gain. Gain measures then increase (or decrease) in Information Entropy in one sample versus another. The one with the highest information (information being the most useful for classification) is selected.

9.3.1. Entropy

$$entropy\ (S) = -\sum_{c=1}^{K} p_c\ LN(\ p_c)$$

Where

- S is the sample of data, comprising K different classes, e.g. sunny, overcast, etc.
- p_c is the proportion of times the category c (e.g. sunny) occurs in the sample S

Example

- Entropy 0 : all samples have the same category. So, the lower the entropy the lesser chaos (just as in the 2nd law of thermodynamics)
- Higher entropy means more random (highest value is LN (p_c)

9.3.1.1. Worked Example

If S is a collection of 14 examples with 9 YES and 5 NO examples, then

Entropy(S) = - (9/14) LN (9/14) - (5/14) LN (5/14) = 0.940

Notice entropy is 0 if all members of S belong to the same class (the data is perfectly classified). The range of entropy is 0 ("perfectly classified") to 1 ("totally random").

9.3.2. Information Gain

Gain(S, A) is information gain of example set S AAAA

$$Gain(\ S_v\ ,V) = \ entropy(S) - \sum(\frac{|S_v|}{|S|}\ \ entropy(S_v))$$

Where

- S is the overall sample
- S_v is a subset of sample S. it contains all records which have a target category v
- |S| is the number of elements in the overall sample
- $|S_v|$ is the number of elements in subset S_v

9.3.2.1. Pseudo code

- Calculate Entropy for the sample
- For each potential split in the sample
    - Calculate Entropy in each potential bin
    - Find the net entropy for your split
    - Calculate entropy gain
- Select the split with the highest entropy gain
- Recursively (or iteratively in some cases) perform the partition on each split until a termination criterion is met
    - Terminate once you reach a specified number of bins
    - Terminate once entropy gain falls below a certain threshold.

9.3.2.2. Example

(KB. I no longer know the source of the example, somewhere on the internet.)

Suppose we want ID3 to decide whether the weather is amenable to playing baseball. Over the course of 2 weeks, data is collected to help ID3 build a decision tree.

The target classification is "should we play baseball?" which can be yes or no.

The weather attributes are outlook, temperature, humidity, and wind speed. They can have the following values:

- outlook = { sunny, overcast, rain }
- temperature = {hot, mild, cool }
- humidity = { high, normal }
- wind = {weak, strong }

Examples of set S are:

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

We need to find which attribute will be the root node in our decision tree. The gain is calculated for all four attributes:

- Gain(S, Outlook) = 0.246
- Gain(S, Temperature) = 0.029
- Gain(S, Humidity) = 0.151
- Gain(S, Wind) = 0.048

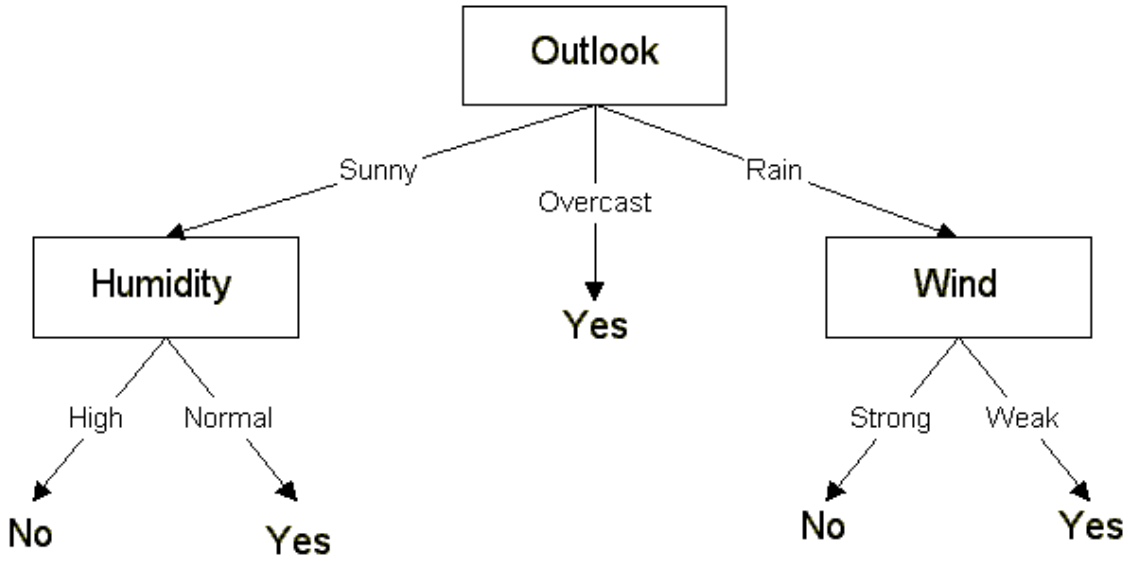
Outlook attribute has the highest gain; therefore it is used as the decision attribute in the root node.

Since Outlook has three possible values, the root node has three branches (sunny, overcast, rain). The next question is "what attribute should be tested at the Sunny branch node?" Since Outlook is used at the root, we only decide on the remaining three attributes: Humidity, Temperature, or Wind.

- Ssunny = {D1, D2, D8, D9, D11} = 5 examples from table 1 with outlook = sunny
- Gain(Ssunny, Humidity) = 0.970
- Gain(Ssunny, Temperature) = 0.570

- Gain(Ssunny, Wind) = 0.019

Humidity has the highest gain; therefore, it is used as the decision node. This process goes on until all data is classified perfectly or we run out of attributes.



9.4. Enhancement

The above algorithm can be extended for multiple features (or attributes). You then simply calculate the Gain per possible sample and per attribute. The leaf is then split on the category with the highest gain.

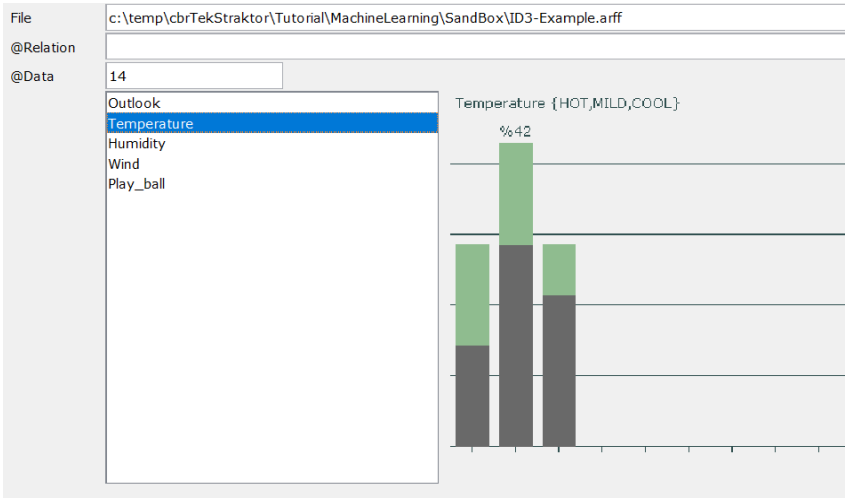
9.5. Implementation

Below is a variant of the ID3. The decision tree is a binary one, i.e. the samples are split in just two branches. The algorithm is essentially an iterative one.

The algorithm also accommodates for collisions, i.e. the same input variables resulting in the same target class. For example, if Sunny then basketball, if Sunny then Baseball. In real-life, contradictory information is often present in data and might actually be correct. When those cases occur, the algorithm then randomly chooses the branch to pick (see the “fippedACoin” flag in the Java source code). I based this on the sound advice of Mr.Yogi Bear : “When in doubt, any one of the two solutions will do”.

9.5.1.1. ARF File

```
@RELATION   ID3_Example

@ATTRIBUTE  Outlook          {Sunny,Overcast,Rain}
@ATTRIBUTE  Temperature      {Hot,Mild,Cool}
@ATTRIBUTE  Humidity         {High,Normal}
@ATTRIBUTE  Wind             {Weak,Strong}
@ATTRIBUTE  Play_ball        {Yes,No}

@DATA
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No
```



9.5.1.2. Result

For what it is worth. The evaluation results of this very small sample set. The accuracy on a test set of only elements is 50%. See the discussion on ROC curves for a statement on the quality of this test approach.

```
- <DecisionTreeModelEvaluation>
  - <TrainedModel>
      <TrainedModelAcurracy>1.0</TrainedModelAcurracy>
      <TrainedModelEntries>12</TrainedModelEntries>
    - <![CDATA[

                          YES        NO ---> Actual
                     ----------  ----------  ----------
          YES         [       9][       0]
          NO          [       0][       3]

                      Precision  Sensitivity    FMeasure
                     ----------  ----------  ----------
          YES         [ 1,000000][  1,000000][  1,000000]
          NO          [ 1,000000][  1,000000][  1,000000]
      ]]>
    </TrainedModel>
  - <TestedModel>
      <TestedModelAcurracy>0.5</TestedModelAcurracy>
      <TestedModelEntries>2</TestedModelEntries>
      <TestSetPercentage>14</TestSetPercentage>
    - <![CDATA[

                          YES        NO ---> Actual
                     ----------  ----------  ----------
          YES         [       0][       1]
          NO          [       0][       1]

                      Precision  Sensitivity    FMeasure
                     ----------  ----------  ----------
          YES         [ 0,000000][       NaN][       NaN]
          NO          [ 1,000000][  0,500000][  0,666667]
      ]]>
    </TestedModel>
</DecisionTreeModelEvaluation>
```

9.5.1.3. Model

The decision tree's nodes and decision criterion (class and pivot point value) are stored in XML.

```
- <DecisionTreeModel>
    - <DecisionTreeModelGeneral>
          <ModelApplicationDomain>General</ModelApplicationDomain>
          <NumberOfNodes>9</NumberOfNodes>
          <NumberOfLevels>4</NumberOfLevels>
          <NumberOfCategories>5</NumberOfCategories>
          <NumberOfClasses>2</NumberOfClasses>
          <NumberOfCoinFlips>0</NumberOfCoinFlips>
          <Classes>[YES][NO]</Classes>
      </DecisionTreeModelGeneral>
    + <DecisionTreeModelEvaluation>
    - <DecisionTreeModelDisplay>
        - <![CDATA[

              .[UID=0 Val=HIGH PUID=-1 #=12 Humidity UNKNOWN]
              .+--[UID=1 Val=SUNNY PUID=0 #=6 Outlook LEFT]
              .   +--[LEAF=NO UID=3 PUID=1 #=2 LEFT]
              .   +--[UID=4 Val=WEAK PUID=1 #=4 Wind RIGHT]
              .      +--[LEAF=YES UID=5 PUID=4 #=2 LEFT]
              .      +--[UID=6 Val=OVERCAST PUID=4 #=2 Outlook RIGHT]
              .         +--[LEAF=YES UID=7 PUID=6 #=1 LEFT]
              .         +--[LEAF=NO UID=8 PUID=6 #=1 RIGHT]
              .+--[LEAF=YES UID=2 PUID=0 #=6 RIGHT]
          ]]>
      </DecisionTreeModelDisplay>
```

9.5.1.4. Java

```
for(int i=0;i<work.subbranches.length;i++)
{
       DecisionTreeCategoryBranch sub = work.subbranches[i];
       double[] dvals = new double[ nrows ];
       double[] dres = new double[ nrows ];
       int[] ilines = new int[ nrows ];
       int k=0;
       for( int j=0;j<  parent.subbranches[i].detail.length ;j++)
       {
             if( orientation==cmcMachineLearningEnums.BranchOrientation.LEFT ) {
// LEFT - keep what is less or equal than splitvalue
              if( parent.dto.FlippedACoin ) {
                    if( j > parent.dto.splitValueIndex ) continue;
              }
              else
              if(
parent.subbranches[parent.dto.splitCategoryIndex].detail[j].value > splitvalue )
continue; // split op de splitcategory
             }
             else
             if( orientation == cmcMachineLearningEnums.BranchOrientation.RIGHT )
{  // RIGHT - keep what is higher than splitvalue
              if( parent.dto.FlippedACoin ) {
                    if( j <= parent.dto.splitValueIndex ) continue;
                    }
```

```
                    else
                    if(
parent.subbranches[parent.dto.splitCategoryIndex].detail[j].value <= splitvalue )
continue; // split op de splitcategory
                }
                else {
                  do_error("Unsupported orientation [" + orientation + "]");
                  return null;
                 }
                }
                //
                dvals[ k] = parent.subbranches[i].detail[j].value;
                dres[ k ] = parent.subbranches[i].detail[j].result;
                ilines[ k ] = parent.subbranches[i].detail[j].lineno;
                k++;
        }
        if( sub.setValues( ilines , dvals , dres ) == false ) return null;
}
```

Chapter 10. K Nearest Neighbors

10.1. Summary

K Nearest Neighbors, also known as Collaborative Filtering or Instance-based Learning. KNN is a straightforward and yet powerful ML multi-label (or multi-category) classifier.

The crux with KNN is to determine the optimal value for K.

NOTE – KNN is a classifier whereas K-Means is a clustering algorithm.

10.2. Algorithm

- Step 1. Initialize the data samples. It is important to normalize all features. (Sometimes the sample data is fuzzified, i.e. replaced with a nominal or mean value thus implementing a low-pass filter.)
- Step 2. Initialize the label or target set, i.e. the label related to each sample.
- Step 3. Pick a single sample and determine the distance to all other samples (using Euclidian distance)
- Step 4. Determine the K samples which have the smallest distance to the sample (these are the K Nearest Neighbors).
- Step 5. Perform a vote, i.e. determine the class which occurs most frequently in the K Nearest Neighbors. This is the predicted class for the sample.
- Step 6. Add the predicted class to the confusion matrix and go to step 2.

When you train the model, you need to determine the optimal K value. In most cases this is achieved by iterating (cycling) through a range of K values (for each k between 1 and N) and store the confusion matrix' accuracy or F-Measure for each K value. The optimal K value is the lowest value related to the highest accuracy or F-Measure.

The KNN model is sensitive to small changes in the sample data. In order to address this various optimization strategies have been developed to make the model robust. For example, repartitioning of the sample data into different train and test data sets a couple of times and re-cycling through the entire training.

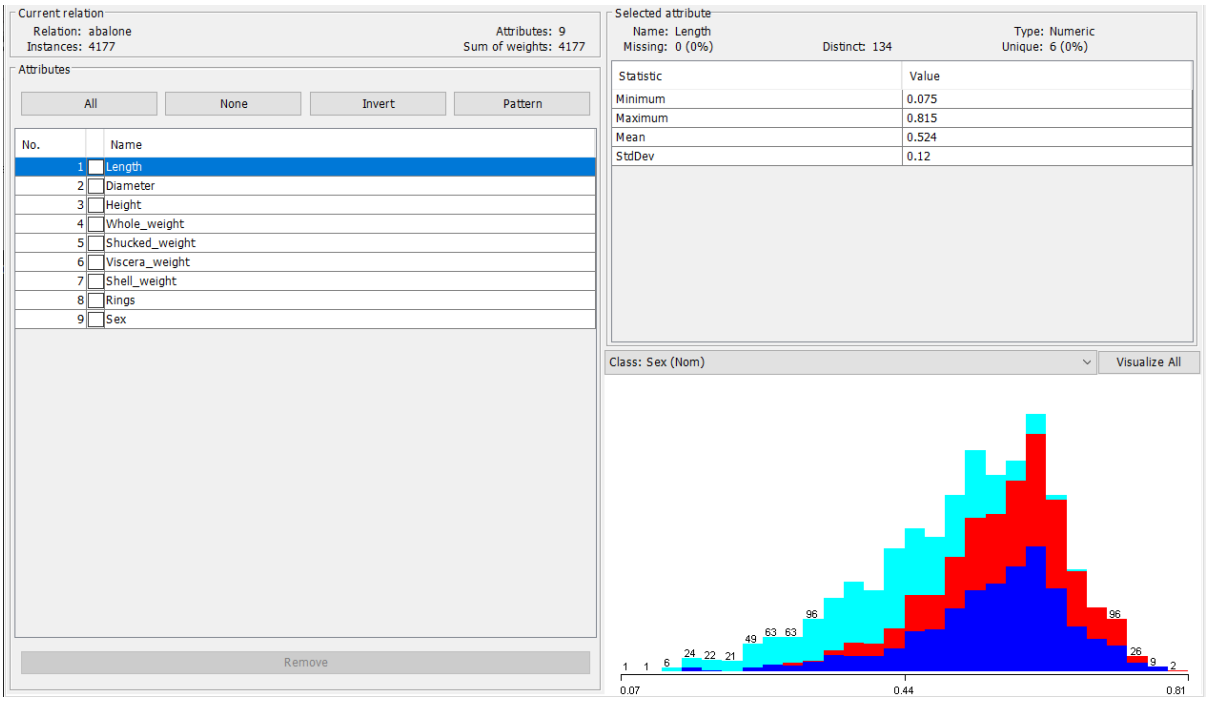
This algorithm is inherently slow during the training phase; but is can be parallelized.

You run the model by simply performing the above logic but only for the optimal value of K. The advantage is that the algorithm scales very well for large sample data sets.

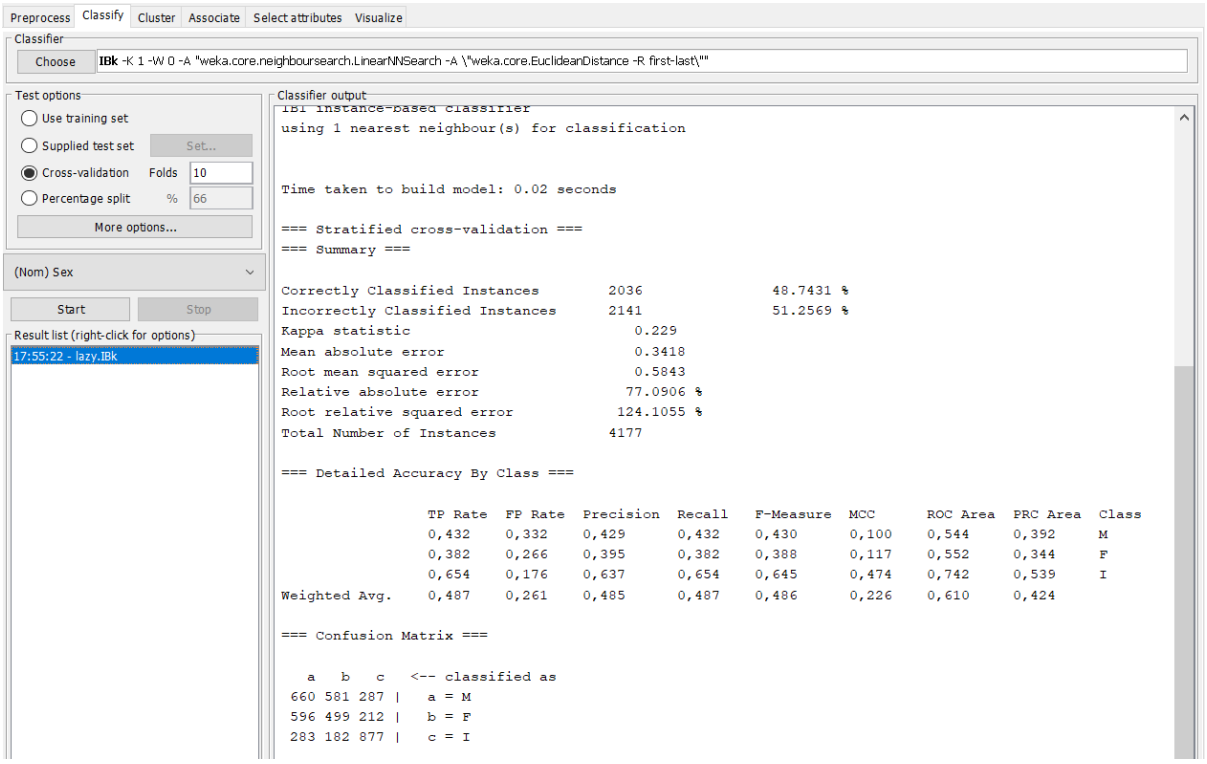
10.3. Implementation

10.3.1. WEKA

In WEKA the algorithm is called IB-K (IB stands for Instance-Based). Below are a few screenshots for predicting the gender of a shell from the Abalone data set.



The accuracy of KNN is low (49%) for predicting the gender of in the Abalone data set. In other words, it does not work at all or the Abalone data might not permit to predict the gender, or the Abalone data set is not linearly separable (see kernel tricks in the SVM section).



10.3.2. Custom developed application

The optimal value for K on the Abalone sample data is 21. It took 2 minutes to complete a training run of 5 repartitions on my laptop for a rather disappointing accuracy of 54%.

```
- <KNNModelGeneral>
    <ModelApplicationDomain>General</ModelApplicationDomain>
    <ARFFFileName>c:\temp\cbrTekStraktor\Tutorial\MachineLearning\SandBox\Abalone.arff</ARFFFileName>
    <OptimalKValue>21</OptimalKValue>
    <NbrOfSampleRows>3633</NbrOfSampleRows>
    <NbrOfFeatures>9</NbrOfFeatures>
    <NbrOfSampleColumns>8</NbrOfSampleColumns>
  </KNNModelGeneral>
```

```
Completed OK
Results training
<TrainedModelAcurracy>0.5408202587393339</TrainedModelAcurracy>
<TrainedModelEntries>18165</TrainedModelEntries>
                        M           F           I ---> Actual
                  -----------  ----------  ----------
M             [      3481][      3004][       873]
F             [      2010][      1923][       487]
I             [      1204][       763][      4420]
                   Precision  Sensitivity    FMeasure
                  -----------  -----------  ----------
M             [  0,473091][   0,519940][  0,495410]
F             [  0,435068][   0,337961][  0,380415]
I             [  0,692031][   0,764706][  0,726555]
Results testing
<TestedModelAcurracy>0.5569852941176471</TestedModelAcurracy>
<TestedModelEntries>544</TestedModelEntries>
<TestSetPercentage>2</TestSetPercentage>
                        M           F           I ---> Actual
                  -----------  ----------  ----------
M             [        94][        90][        20]
F             [        55][        55][        12]
I             [        40][        24][       154]
                   Precision  Sensitivity    FMeasure
                  -----------  -----------  ----------
M             [  0,460784][   0,497354][  0,478372]
F             [  0,450820][   0,325444][  0,378007]
I             [  0,706422][   0,827957][  0,762376]
```

Chapter 11. Naïve Bayes classifiers

11.1. Source

The following section has been copied without any modifications from an article by Tim Jones https://developer.ibm.com/articles/cc-supervised-learning-models/ It comprises a worked example of the Naïve Bayes algorithm.

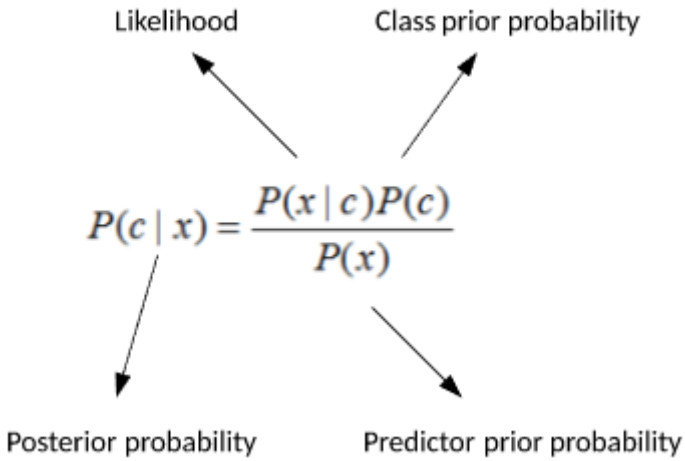
11.2. Summary

Naïve Bayes is a method that permits the construction of classifiers in a simple and straightforward way. One interesting feature of naïve Bayes is that it works well with very small data sets. Naïve Bayes classifiers exploit certain assumptions about the data—namely, that all attributes are independent—but even with this simplification, you can successfully apply the algorithm to complex problems. Let's first look at Bayes theorem, and then we'll work through a simple classification problem.

11.3. Bayes theorem

Bayes theorem, illustrated in Figure 4, provides a way to determine the probability of an event based on prior knowledge of conditions that might be related to the event. It says that the probability of a target (*c*) given a predictor (*x*, called the *posterior probability*) can be calculated from the probability of the predictor given the class (called *likelihood*) multiplied by the prior probability of the class that is divided by the prior probability of the predictor (sometimes called the *evidence*).

Figure 4. Bayes theorem



Put more simply, Bayes theorem permits the calculation of conditional probabilities of some event given prior evidence (target *c* occurs given that a prior event *x* occurred). For example, given a collected data set, you could use Bayes to identify the probability that it will rain given other attributes (such as whether it's overcast or sunny).

11.4. Bayes through example

Let's dig into this theorem through an example. Here in Colorado, one of our biggest winter pastimes is snow sports. However, not every day is a great day to ski. In Figure 5, I've listed 15 observations of weather and temperature and whether it was a ski day. For *weather,* it might be sunny, windy, or snowing; for *temperature,* it's either cold or freezing. My dependent variable is the ski class, which is represented as yes or no. Given a set of conditions, I want to identify whether I should ski. At the right side of this figure is the class probability, represented by the counts of each class (how many instances of yes or no occur in the data set).

Figure 5. Simple table of weather conditions for skiing

Weather	Temperature	Ski?
Sunny	Cold	Yes
Snowing	Cold	Yes
Sunny	Freezing	Yes
Windy	Cold	No
Snowing	Cold	No
Sunny	Cold	Yes
Windy	Cold	No
Snowing	Cold	Yes
Snowing	Cold	Yes
Windy	Freezing	No
Sunny	Cold	No
Snowing	Cold	Yes
Windy	Freezing	No
Windy	Cold	Yes
Snowing	Cold	Yes

	Yes	No
P(c)	9	6

P(YES) = 9/15
P(NO) = 6/15

Next, I build frequency tables from the data set (that is, the individual probabilities for weather and temperature). For each feature attribute, I count and sum the occurrences of each feature for the given class (`P(c)` or Yes/No). This table is shown in Figure 6.

Figure 6. Frequency tables for the data set

Likelihood table		Ski?		
		Yes	No	
Weather	Sunny	3/9	1/6	=4/15
	Windy	1/9	4/6	=5/15
	Snowing	5/9	1/6	=6/15
		=9/15	=6/15	

Likelihood table		Ski?		
		Yes	No	
Temperature	Cold	8/9	4/6	=12/15
	Freezing	1/9	2/6	=3/15
		=9/15	=6/15	

From this table, I can see the frequencies for each event given the target.

Next, I translate my frequency tables into likelihood tables (`P(x|c)`). From the table in Figure 7, I can see that, independent of all other variables, if it's sunny, I'll ski with a probability `P (3/9)` and not ski with a probability of `P (1/6)`. I could also write this as `P (Weather=Sunny | Ski=Yes) = 3/9` and `P (Weather=Sunny | Ski=No) = 1/6`.

Figure 7. Likelihood tables

Frequency table		Ski?	
		Yes	No
Weather	Sunny	3	1
	Windy	1	4
	Snowing	5	1
		9	6

Frequency table		Ski?	
		Yes	No
Temperature	Cold	8	4
	Freezing	1	2
		9	6

Now, I have everything that I need to apply Bayes and predict the probability of a class (whether I'll go skiing) given a set of weather conditions. So, say that I want to know if I'll ski if it's snowing and freezing. I write this equation as `P (Yes | Weather=Snowing & Temperature=Freezing).`

I start by collecting the data for the likelihood (`P (x | c`) and the class probability (`P(c)`). These values came directly out of the likelihood table. So, for the `P(Yes)` side, I multiply `P(Snowing|Yes)` with `P(Freezing|Yes)` and `P(Ski=Yes)`, which gives me the numerator of my equation.

The final step is to calculate the denominator (`P(x)`, or the evidence). This calculation is the same for both classes, so I calculate it once. This step normalizes the result. So, `P(Snowing)` is 5+1/15 (6/15) and `P(Freezing)` is 1+2/15 (3/15). Figure 8 shows this calculation.

[KB ? P(snow|ski) is 5/9 and not 3/9]

P(Yes | Snowing and Freezing)

Probability that we can ski.
P(Weather=Snowing | Ski=Yes) = 3/9
P(Temperature=Freezing | Ski=Yes) = 1/9
P(Ski=Yes) = 9/15

Probability that we dont ski.
P(Weather=Snowing | Ski=No) = 1/6
P(Temperature=Freezing | Ski=No) = 2/6
P(Ski=No) = 6/15

P(x | Ski=Yes)P(Ski=Yes) =
(3/9)*(1/9)*(9/15) = 0.22

P(x | Ski=No)P(Ski=No) =
(1/6)*(2/6)*(6/15) = 0.02

P(x) = P(Weather=Snowing) + P(Temperature=Freezing)
P(x) = (6/15) + (3/15) = 0.6

P(Ski=Yes | x) = 0.22 / 0.6 = **0.37**
P(Ski=No | x) = 0.02 / 0.6 = 0.03

The final step is to calculate the probabilities by using the numerator and denominators I just computed. Given my sample data set, I take the largest of the probabilities for conditions *x,* which leads to my answer (I most likely would ski).

Naïve Bayes makes it easy to predict a class for a set of conditions from a data set and can perform better than some models. The approach does assume that predictors are independent, which doesn't always hold for real-world problems.

11.5. Java implementation

11.5.1. ARRF File

```
@RELATION  Howest-Bayes-Voorbeeld

@ATTRIBUTE weather {Sunny,Snowing,Windy}
@ATTRIBUTE Temperature {Cold,Freezing}
@ATTRIBUTE Ski  {Yes,No}

@DATA
Sunny,Cold,Yes
Snowing,Cold,Yes
Sunny,Freezing,Yes
Windy,Cold,No
Snowing,Cold,No
Sunny,Cold,Yes
Windy,Cold,No
Snowing,Cold,Yes
Snowing,Cold,Yes
Windy,Freezing,No
Sunny,Cold,No
Snowing,Cold,Yes
Windy,Freezing,No
Windy,Cold,Yes
Snowing,Cold,Yes
```

11.5.2. Model evaluation

The size of the data set is a bit too small to make meaningful conclusions, trained accuracy 84% and tested accuracy is 50%.

```
- <BayesModelEvaluation>
  - <TrainedModel>
      <TrainedModelAcurracy>0.8461538461538461</TrainedModelAcurracy>
      <TrainedModelEntries>13</TrainedModelEntries>
    - <![CDATA[

                        YES        NO ---> Actual
                    ---------- ---------- ----------
            YES      [       8][       1]
            NO       [       1][       3]

                     Precision  Sensitivity    FMeasure
                    ---------- ---------- ----------
            YES      [ 0,888889][  0,888889][  0,888889]
            NO       [ 0,750000][  0,750000][  0,750000]
        ]]>
    </TrainedModel>
  - <TestedModel>
      <TestedModelAcurracy>0.5</TestedModelAcurracy>
      <TestedModelEntries>2</TestedModelEntries>
      <TestSetPercentage>13</TestSetPercentage>
    - <![CDATA[

                        YES        NO ---> Actual
                    ---------- ---------- ----------
            YES      [       0][       1]
            NO       [       0][       1]

                     Precision  Sensitivity    FMeasure
                    ---------- ---------- ----------
            YES      [ 0,000000][      NaN][      NaN]
            NO       [ 1,000000][  0,500000][  0,666667]
        ]]>
    </TestedModel>
</BayesModelEvaluation>
```

11.5.3. Model

The trained Bayes model simply stores an overview of the probabilities per feature (attribute) for each class (label) and the overall probability for each class. This is nothing more than a runtime performance improvement short-cut.

In the proposed Java implementation of the model these probabilities are called “ClassProbabilities” and “ClassProbabilitiesPerBin” and stored in an XML file.

ClassProbabilitiesPerBin are calculated and stored for each feature. Picture below shows the ClassProbabilities and ClassProbabilities per bin for the nominal “weather” feature.


```
- <BayesModelGeneral>
    <ModelApplicationDomain>General</ModelApplicationDomain>
    <NumberOfEntries>13</NumberOfEntries>
    <NumberOfBins>16</NumberOfBins>
    <NumberOfModels>3</NumberOfModels>
    <NumberOfClasses>2</NumberOfClasses>
    <Classes>[YES][NO]</Classes>
    <ClassProbabilities>[0.7142857142857143][0.35714285714285715]</ClassProbabilities>
  </BayesModelGeneral>
- <BayesModelEvaluation>
  + <TrainedModel>
    + <![CDATA[]]>
  + <TestedModel>
    + <![CDATA[]]>
  </BayesModelEvaluation>
- <Model>
    <ModelIndex>0</ModelIndex>
    <CategoryName>weather</CategoryName>
    <CategoryType>NOMINAL</CategoryType>
    <NominalValues>[SUNNY][SNOWING][WINDY]</NominalValues>
    <BinSegmentBegins>[0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0][0.0]</BinSegmentBegins>
    <ClassCounts>[9][4]</ClassCounts>
  - <BABinStats>
    + <BACountsPerBin>
    - <BAProbabilitiesPerBin>
        <BAProbabilityPerBin>0.23076923076923078</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.46153846153846156</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.3076923076923077</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
        <BAProbabilityPerBin>0.0</BAProbabilityPerBin>
      </BAProbabilitiesPerBin>
    </BABinStats>
```


11.5.4. Source code

The following code fragment processes the “ClassProbabilitiesPerBin” to predict the result.

```
double feebleProbability = 1 / (double)(HistogramEntries + 1);
double[] predicted = new double[ nclasses ];
for(int i=0;i<nclasses;i++)
{
        double teller=1;
        double noemer=1;
        for(int j=0;j<nattributes;j++)
        {
                if( pset[j][i] == null ) continue;
                // teller : indien de kans 0 is zet dan een zeer kleine kans
```

```
                double telProb = pset[j][i].BinClassProbability;
                if( telProb == 0 ) telProb =  feebleProbability;
                teller = teller * telProb;

                // noemer : indien kans is 0 dan is er geen stat beschikbaar - negeer
de entry dan
                double noeProb = pset[j][i].BinProbability;
                if( noeProb == 0 ) {
                   noeProb = 1;
                    do_log( 1 , "Could not find a probability for any of the classes
in the model for model [" + j + "] for value [" + attributevalues[j] + "] for
class [" + i + "] - Guessing");
                }
                noemer = noemer * noeProb;
        }
        if( noemer == 0 ) predicted[i] = Double.MIN_VALUE;
        else predicted[i] = teller * ClassProbabilities [i] / noemer;

}
```

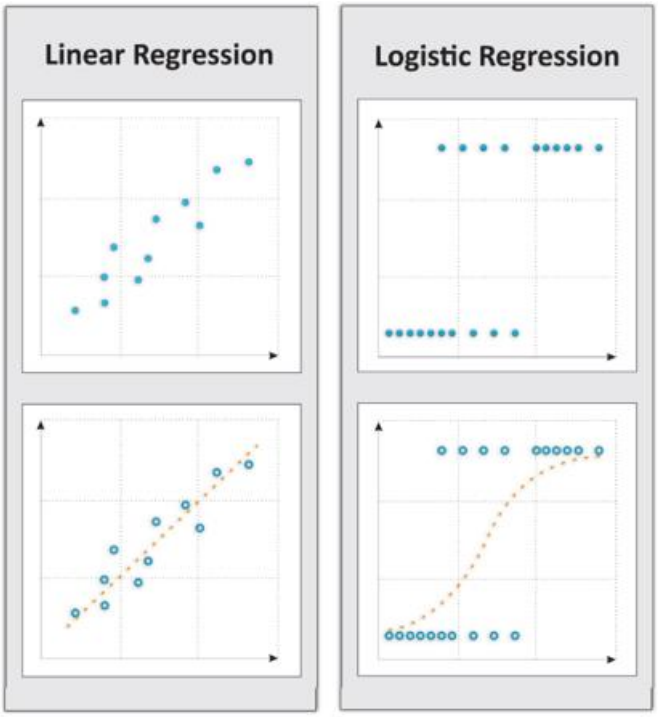
Chapter 12. Logistic Regression

12.1. Definition

The idea is to determine a regression equation (which is a linear function) capable for predicting a categorical result, i.e. a category, class or label (and not a curve). Logistic regression is even if called regression, a classification method. In addition, logistic regression is based on the probability of a sample belonging to a class and maximizing that odd.

The input variables (x) are the factors that drive the prediction of the target classes. The input variables can be continuous (numeric) or categorical (nominal).

The target variables are either binary (a.k.a. dichotomous) or proportional between zero and one.



Source of the diagram: reference document [01]

The results in the right-most diagram are categorical, even binary (0,1).

The lower right-most diagram shows a what the possible shape of a regression curve might look like for predicting a 0,1 categorical target. A curve that fits that shape is called a sigmoid curve. Its inclination (slope) and narrowness are defined by weights on a linear equation, e.g. sigmoid (ax + b).

Logistic regression is about finding the weights that shape the sigmoid that fits best to the way the categories are distributed across the sample set.

NOTE – We will limit the discussion to dichotomous results and not multi-categorical results (which is the purpose of SoftMax algorithm).

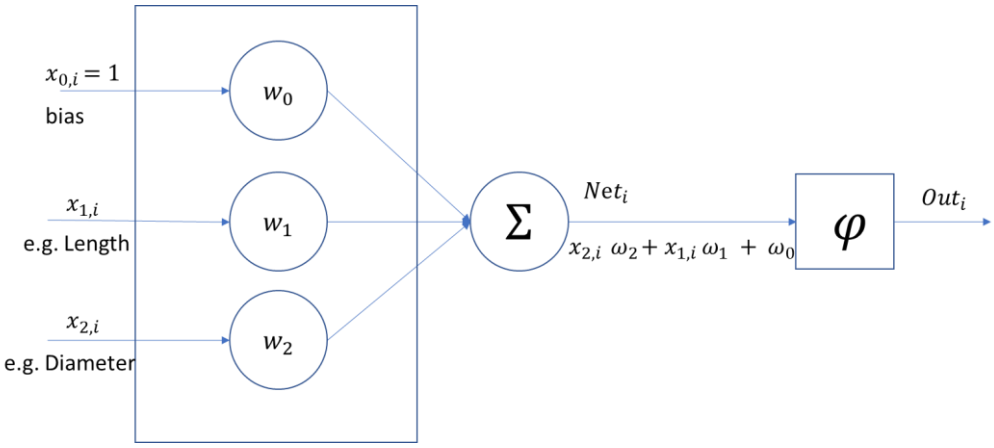
12.2. Approach

12.2.1. Premise

We want to

- create a linear equation or function ($w_n x_n + .. + w_1 x_1 + w_0$)
- which is capable of calculating categorical values 0 or 1

The issue is that a linear function is not able to do that. So, we will need to introduce an additional function which implements such threshold (or step) behavior. This is the “activation” function (phi).



Possible notations

Let us have a look again at our linear regression equation

$$y_i = w_n x_n + w_{n-1} x_{n-1} + .. + w_1 x_1 + w_0$$

Remember that y_i is a single numeric value which is the result of the dot product of vector W and the augmented vector x_i (a multi-variable vector onto which a constant 1 has been added).

To be mathematically more precise the single value y_i is the product of the transposed vector W (with dimensions N,1) and the augmented vector x_i (with dimension 1,N). Assuming the linear equation had N-1 dimensions.

$$y_i = \begin{bmatrix} w_0 \\ .. \\ w_n \end{bmatrix} [\ 1\ , x_1\ ,\quad x_2\ ,\ ..\ ,\quad x_n\ ]$$

If you want to define this for all samples in the set, you can express the above as a matrix of all results Y and the various input variables X.

$$\bar{\bar{Y}} = \bar{W}^T \bar{X}$$

12.2.2. Log-odd function or Logit

To reiterate, we want y_i to be either 0 or 1. Yet there is no linear function that shows this behavior. Therefore, we have no alternative than to move out of the linear domain to another, in this case the probability domain, more specifically the logarithmic probability domain.

Logistic regression is based on the idea of modeling the odds of belonging to class 1 (using the natural logarithmic function).

Now suppose we introduce the probability P(C=1|x) or p ($\overline{x_i}$), i.e. the probability that an element belongs to class 1. Clearly, the same element belongs to class 0 with a probability 1 - p($\overline{x_i}$).

This is the odd or the conditional probability.

$$odd = \frac{\mathrm{p}(\overline{x_i})}{1 - \mathrm{p}(\overline{x_i})}$$

NOTE - Probability of throwing 2 in a single dice throw is 1/6 (0.166). The odd for this throw is 1/6 against 5/6, so 1 against 5 (or 0.2).

The result of the odd range between 0 and positive infinite, so not between 0 and 1. The logarithm of an odd is called a logit or log-odd function. The results of the natural logarithm range from 0 to infinite. So, we need something more. The "logi" function will enable us to partially achieve this.

The logit function is defined as follows:

$$z_i = \mathrm{LOGIT} = ln\left(\frac{\mathrm{p}(\overline{x_i})}{1 - \mathrm{p}(\overline{x_i})}\right)$$

The issue is that z_i is still not either 0 or 1, but continuous and might also be greater than 1.

Remember: If $a = \ln(b)\ then\ b = e^a$

$$e^{z_i} = \frac{\mathrm{p}(\overline{x_i})}{1 - \mathrm{p}(\overline{x_i})}$$

Let's rewrite for p (and keep in mind z_i is a single value)

$$e^{z_i}(1 - \mathrm{p}(\overline{x_i})) = \mathrm{p}(\overline{x_i})$$

$$e^{z_i} - e^{z_i}\mathrm{p}(\overline{x_i}) = \mathrm{p}(\overline{x_i})$$

$$e^{z_i} = \mathrm{p}(\overline{x_i}) + e^{z_i}\mathrm{p}(\overline{x_i})$$

$$e^{z_i} = \mathrm{p}(\overline{x_i})(1 + e^{z_i})$$

$$e^{z_i} = \mathrm{p}(\overline{x_i})(1 + e^{z_i})$$

$$\mathrm{p}(\overline{x_i}) = \frac{e^{z_i}}{1 + e^{z_i}}$$

$$\mathrm{p}(\overline{x_i}) = \frac{1}{1 + \frac{1}{e^{z_i}}}$$

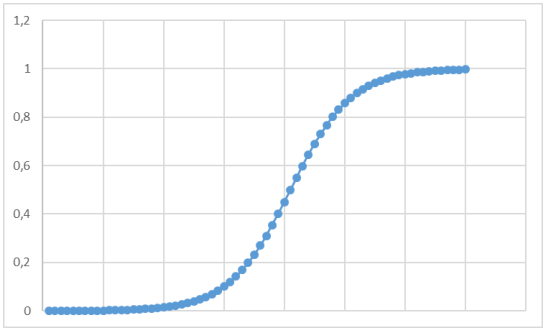
$$\text{Inverse LOGIT} = LOGIT^{-1} = \mathrm{p}(\overline{x_i}) = \frac{1}{1 + e^{-z_i}}$$

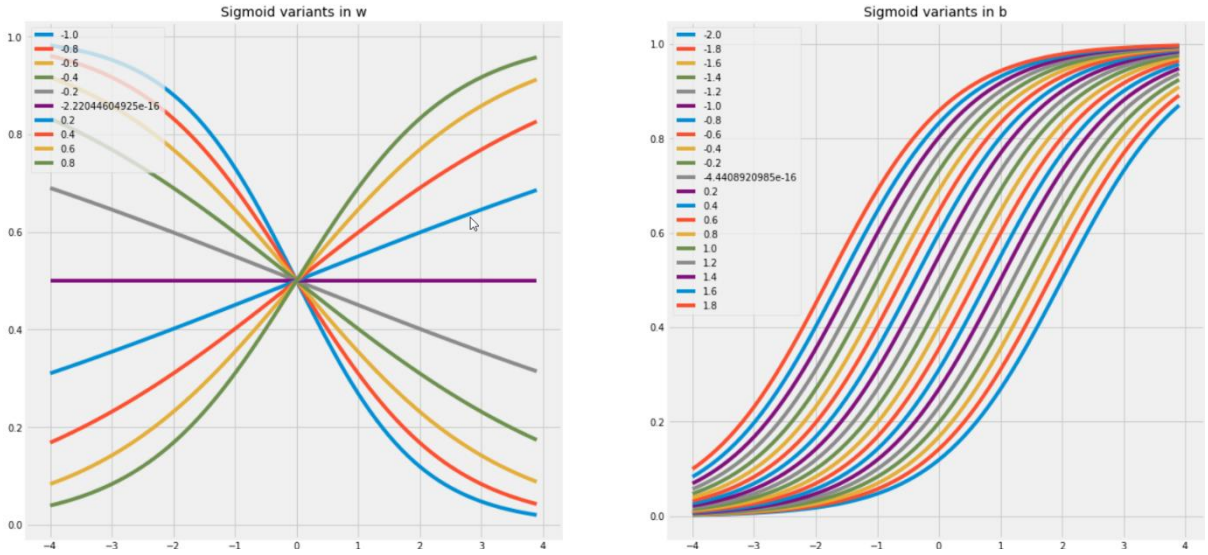
So we now have the probability expressed as a function dependent of z_i which is actually a continuous value between 0 and 1. So we already satisfy the requirement of between 0 and 1. It is not dichotomous yet but this can easily be fixed by applying a filter condition, e.g. zi < 0.5 is 0 else 1.

12.2.3. Logistic curve or sigmoid

The formula $\frac{1}{1+e^{-z_i}}$ is actually the formula of a well-known curve by mathematicians: the logistic curve of sigmoid, which is defined by the "sigma" character.

$$\mathrm{p}(\overline{x_i}) = \frac{1}{1 + e^{-z_i}} = \sigma(\overline{x_i}) = sigmoid(\overline{x_i})$$





Source of the diagram: reference document [01].

The above diagram depicts the changes in the shape of the sigmoid curve sigmoid (wx + b). Intuitively you can see that by optimizing the weights (w) and bias (b or w0) the distribution of the categories 0 and 1 can be predicted based on the input variables.

IMPORTANT – The sigmoid of x=0 is always 0.5 (the intercept of the sigmoid curve is 0,5) so we also have our potential filter function.

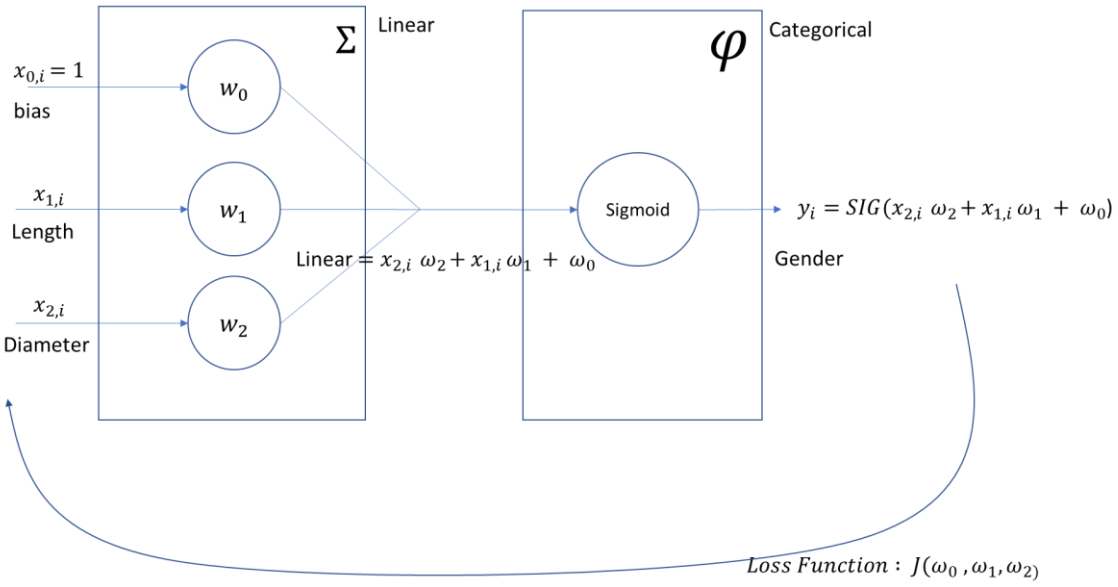
12.2.4. Maximum likelihood and Loss Entropy

To re-iterate, we have

- A so called "activation function", in this case $sigmoid\ (\overline{x_i})$ or $\sigma\ (\overline{x_i})$ that enables us to model a result that continuously ranges between 0 and 1
- We also want to this activation function to work on the a regression equation $\overline{W}^T\overline{X}$ (which is called Net function in neural networks), which is a linear function dependent of a weight vector W and which will be fed by the x variables in our sample set.

Let us now try to find an approach to find the maximum likelihood (or maximum odd) which can be defined by the weights of our linear regression equation.

Mathematicians have proven that the L2 (Mean Squared Error) Loss function cannot be used when the activation function is a sigmoid. We will use another approach called Loss Entropy or Cross Entropy Loss function. It works like this.



The probability for a class to be 1 is defined as P(C=1|x)

The probability for a class to be 0 is then defined as P(C=0|x) which is simply 1 –P(C=1|x)

- P(C=1|x) (which will hence onwards be referred to as is by H(x))
- P(C=0|x) = 1 – P(C=0|x) or simply: 1 – H(x)

The above 2 statements can be combined into one formula using the following trick

$$\text{P(C=y|x)} = H(x)^t(1 - H(x))^{1-t}$$

Where

- If y=0 then $H(x)^0(1 - H(x))^1$ or $1 - H(x)$

- If y=1 then $H(x)^1(1-H(x))^0$ or $H(x)$

The maximum of P(C=y|x) is the maximum of the multiplication of $H(x)^t$ and $(1-H(x))^{1-t}$. The formula looks as follows

$$Likelihood\ of\ \ L(w) = \prod_{m=1}^{M} \ \ (H_m^{y_m}(1-\ H_m)^{1-\ y_m}$$

- M is the number rows or vectors in our sample. (KB – in Linear regression you used N)
- y is category (or label), so y is 0 or 1.

NOTE – Most books express this as L(w,b), however I assume b to be w0, so L(w) works fine as a definition.

Determining the maximum of a multiplicative or quadratic function is cumbersome. The trick to solve this is to transform it to switch to the natural logarithm, because then we just need to determine the maximum of a sum (remember that ln(ab) = ln(a) + ln(b) and also that ln(a)^n is n ln(a))

Furthermore, by negating the equation (multiply by -1) we now need to find the minimum instead of the maximum; and finding a minimum of a sum can be achieved taking the partial derivative or gradient and performing the gradient descent.

$$-\ln\left(L(w)\right) = -\ln\left(\prod_{m=1}^{M} \ \ (H_m^{y_m}(1-\ H_m)^{1-\ y_m}\right)$$

Can be rewritten to the sum of

$$E(w) = -\sum_{m=1}^{m}\{\ y_m \ln\left(H_m\right)\ + (1-y_m)\ln(1-H_m)\ \}$$

NOTE - I have limited the notation to a linear equation of first degree ($w_1x_{1m}+w_0$). A bit clumsy.

$$E(w) = -\sum_{m=1}^{M} y_m \ln\left(SIGMOID(w_1x_{1m}+w_0)\ + (1-y_m)\ln\left(1-(SIGMOID(w_1x_{1m}+w_0)\right)\right.$$

$$E(w) = -\sum_{m=1}^{m} y_m \ln\left(\sigma(w_1x_{1m}+w_0)\ + (1-y_m)\ln\left(1-(\sigma(w_1x_{1m}+w_0)\right)\right.$$

12.2.5. Gradient descent

The minimal can be found using the gradient descent approach (see linear regression). Gradient descent is an iterative technique for converging to a minimum by moving into the direction defined by the gradient or partial derivative of the loss function.

There are a couple of convenient mathematical tricks when determining the partial derivative for w0 and w1.

- The derivative of ln(x) is 1/x
- The derivative of the sigmoid is $\sigma\ (1-\sigma)$
- The derivative of a sum of functions is the sum of the derivatives of each function
- The chain rule when determining the partial derivatives

I did not try to solve the above partial derivatives for w0 and w1 (I did but I lost my patience), so let us acknowledge that the partial derivatives will be a vector with 2 elements as follows. Source is https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html

$$\begin{cases} \frac{\delta E(W)}{\delta w_1} = \sum_{m=1}^{M} (\sigma(x_m) - y_m)\, x_m^1 \\ \frac{\delta E(W)}{\delta w_0} = \sum_{m=1}^{M} (\sigma(x_m) - y_m)\, x_m^0 \end{cases}$$

We now apply the same convenience transformations as for the linear regression to express the iterative nature of the gradient descent w_k is the result of w_{k-1} minus the gradient descent).

We will again introduce the step size (alpha) to define the magnitude of our step. Alpha can be positive or negative, the weights ill adapt accordingly.

$$\begin{cases} w_{i_k} = w_{i_{k-1}} - \propto \sum_{m=1}^{M} (\sigma(x_m) - y_m)\, x_m^1 \\ w_{0_k} = w_{0_{k-1}} - \propto \sum_{m=1}^{M} (\sigma(x_m) - y_m) \end{cases}$$

12.2.6. Linear regression formula

So, we finally get to the logistic regression formula.

$$\bar{W} \dashleftarrow \bar{W} - \alpha \nabla\ \sigma(\bar{W}^T \bar{X})$$

- W is the weight vector (we might need to transpose it so that the matrix multiplications work out alight)
- X are vectors of with the input variables (of N dimensionality or columns and there are M samples or rows)
- $\bar{W}^T \bar{X}$ is our linear equation, our actual model.
- $\sigma(\bar{W}^T \bar{X})$ is the sigmoid of our model
- $\nabla\sigma(\bar{W}^T \bar{X})$ is the gradient, i.e. the partial derivative of the difference between the actual categories (Y) and the result of the sigmoid function.
  $\alpha\ \nabla\sigma(\bar{W}^T \bar{X})$ is the stepsize multiplied to the gradient
- $\bar{W} + \alpha\ \nabla\sigma(\bar{W}^T \bar{X})$ is the current W vector minus the stepsize times gradient, so the updated weight vector.

12.2.6.1. Intuitive notation

IMPORTANT – The gradient comprises a multiplication with X (input variables). In the final formula right above, X is not explicitly mentioned, so you might accidentally overlook it.

I therefore tend to rewrite the formula in matrix notation also to make the coding easier and to check the alignment of the dimensions when multiplying.

$$W - \alpha \left(\left(Y - \sigma(W\, X^T) \right) X \right)$$

Where

- M is the number of samples or rows in the data set
- N the dimension or number or columns in the data set
- X are the input variables of the sample data, so (M,N)
- Y are the categories or classes or results of the sample (M,1)
- W are the weight factors (N,1) or (1,N) according to the way you like to code this vector.

12.2.7. Pseudo code

- Read data M rows of N columns and create the augmented matrix. This is the DataMatrix of dimension (M,N)
- Read M rows of labeled categories. This is the LabelMatrix of dimension (M,1)
- Initialize the WeightMatrix of dimension (N,1), for example set all weights to 0
- Iterate (for a predefined number of cycles or until the gradient does not change any more)
    - Calculate the sigmoid of the multiplication of WeightMatrix and DataMatrix (TempMatrix)
    - LabelMatrix minus TempMatrix is Temp2Matix
    - Temp2Matrix multiply with DataMatrix is Temp3Matrix (DO NOT FORGET THIS)
    - multiply Temp3Matrix with the stepsize (GradientMatrix)
    - Previous WeightMatrix minus GradientMatrix

12.2.8. Implementation

12.2.8.1. Python

```
def loadDataSet():
    dataMat = []; labelMat = []
    fr = open('testSet.txt')
    for line in fr.readlines():
        lineArr = line.strip().split()
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
        labelMat.append(int(lineArr[2]))
    return dataMat,labelMat

def sigmoid(inX):
    return 1.0/(1+exp(-inX))

def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)
    labelMat = mat(classLabels).transpose()
    m,n = shape(dataMatrix)
    alpha = 0.001
    maxCycles = 500
    weights = ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix*weights)
        error = (labelMat - h)
        weights = weights + alpha * dataMatrix.transpose()* error
    return weights
```

1 Convert to NumPy matrix data type

2 Matrix multiplication

12.2.8.2. Java

Java implementation. Should you ever compare the code with the version on Github. All lines that perform error checking have been left out of the following bits of code.

```
//  FORMULE = W - alfa ( (y - sigmoid( w XT ))X )
double minLoss = -1;
double prevloss = 0;
double  maxAccuracy = Double.NaN;
int minLossCycle=-1;
for(int cycle=0;cycle<NbrOfCycles;cycle++)
{
 cmcMatrix h1 = vrout.multiplyMatrix( weightMatrix , dataMatrixTransposed ); //
(1,n) * (n,m) =>  (1,m)

 cmcMatrix h2b = vrout.sigmoidMatrix( h1 ); // (1,m)

 cmcMatrix errorMatrix = vrout.substractMatrix( targetMatrixTransposed , h2b );
// 1,m - 1,m = 1,m

  // (y - sigmoid( w XT )) X     (1,m) (m,n) => 1,n    => number of columns
  cmcMatrix gradientMatrix = vrout.multiplyMatrix( errorMatrix , dataMatrix );

  //  apply stepsize
  cmcMatrix t2 = vrout.scalarMultiplyMatrix( StepSize , gradientMatrix);  // alfa
* 1,n
```

```
  //
  prevWeight = weightMatrix;  // 1,n
  weightMatrix = vrout.addMatrix( weightMatrix , t2 );   // (n,1)

  // Max Accurary is not necessarily the best trained model, but hey ..
  double dac = determineCurrentAccuracy( dataMatrixTransposed , weightMatrix ,
targetMatrixTransposed);
  if( Double.isNaN( dac ) ) return false;
  if( cycle == 0 ) { maxAccuracy = dac; OptimalNbrOfCycles = NbrOfCycles; }
  if( dac > maxAccuracy ) { maxAccuracy = dac; OptimalNbrOfCycles = cycle; }

  // alternatief = zoek naar de cycle waar de weights niet meer wijzigen - maxaccu
is overfitted
  double wdist=0;
  for(int k=0;k<weightMatrix.getNbrOfColumns();k++)
  {
       wdist +=  Math.abs(weightMatrix.getValues()[0][k] -
prevWeight.getValues()[0][k]);
  }
  if( cycle == 0 ) { minLoss = wdist; }
  if( minLoss > wdist ) { minLoss = wdist; minLossCycle = cycle; }
 }
```

12.2.9. Results and model

The following screenshot shows the result after training a linear regression model on Iris data set (after having modified it towards a one against all version. “A one against all” approach is basically ensuring that there are only two target categories, of which one is the one you want to assess).

The accuracy on the test data set is 100%, which is comforting should you ever feel the urge to detect Iris-Setosa flowers.

Optimal number of cycles is nine and you can observe the weight vector values for the various features on the picture below.

The model also stores the normalization parameters (mean and standard deviation per features). You need to use the values of the training set to preprocess any new set of data as opposed to recalculating the mean and standard deviation of the new set.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Application : cbrTekStraktor V0.4 (18-Apr-2019) -->
<!-- Start : 25-APR.-2019 16:47:21 -->
<LogisticRegressionModel>
  - <LRModelGeneral>
        <ModelApplicationDomain>General</ModelApplicationDomain>
        <ARFFFileName>c:\temp\cbrTekStraktor\Tutorial\MachineLearning\SandBox\Iris - Tweaked.txt</ARFFFileName>
        <MaxCycles>2500</MaxCycles>
        <OptimalCycles>9</OptimalCycles>
        <StepSize>0.001</StepSize>
        <NumberOfFeatures>5</NumberOfFeatures>
        <Weights>[0.9284436250965923][0.2908568403179856][-0.2651055827249416][0.3882196042221724][0.37344737984393694]</Weights>
    </LRModelGeneral>
  + <LogisticRegressionModelEvaluation>
  + <Categories>
  - <Normalisation>
        <NormalisationMean>[5.819230769230772][3.079230769230766][3.704615384615386][1.188461538461539]</NormalisationMean>
        <NormalisationStdDev>[0.8120312509591537][0.4469875314921896][1.7774147910086608][0.7728568381710111]</NormalisationStdDev>
    </Normalisation>
</LogisticRegressionModel>
```

Chapter 13. Stochastic gradient descent

As you could see in the section on logistic regression, we can easily transform the formulas for Logistic Regression into procedural code. However, the algorithm is computationally intensive. Each iteration recalculates the outcome of the then current regression equation, the sigmoid, the gradient, etc.

Therefore, a lightweight method called Stochastic Gradient Descent (SGD) was developed.

In SGD, one simply performs the same algorithm as the Logistic Regression, however on a subset of the training data. It is a stochastic approach because the sample subsets are randomly selected.

One also switches to another randomly created subset in between cycles. The latter approach is called “mini-batch”.

NOTE - Mathematicians have proved that SGD will converge to the same weights as the full Logistic Regression. If you think about it: this makes sense. Whether you define a loss function surface by a few points or by multiple points, the slope will be identifiable, and a minimum will be reached albeit in a coarser manner.

Chapter 14. Support vector machines

14.1. Summary

Support Vector Machine (SVM) is a supervised learning classification algorithm.

Sequential Minimal Optimization (SMO) is an optimized (computational simpler) version of

Data scientists consider the SMO to be the best out of the box classifier, i.e. it delivers good accuracy, low bias and has good training performance.

14.2. Origin

Both SVM and SMO are recent algorithms. SVM was conceived by Vladimir Vapnik in the 1970's and finalized in the 1990's when he moved to the US and started working for AT&T. SMO was created in 1998 by John Platt.

SVM and SMO were quite popular during the turn of the century and were then used extensively in OCR for handwritten texts and similar applications.

14.3. Quick mathematical recap

In order to understand the Support Vector machine algorithm some specific (even arcane) mathematical concepts might need to be refreshed, in particular the concept of a hyperplane and Lagrange functions.

14.3.1. The idea of a separating hyperplane

A hyperplane is a "subspace of one dimension less than the ambient space". Wow!

For example, in the 3D space, a hyperplane is a 2D plane (ax +by +c). In the 2D space, it is just a line (ax +b or ax + by – c = 0 or the notation used in these handouts is $w_1x_1 + w_2x_2 + W_0$)

A hyperplane equation is mathematically the same as a regression equation.

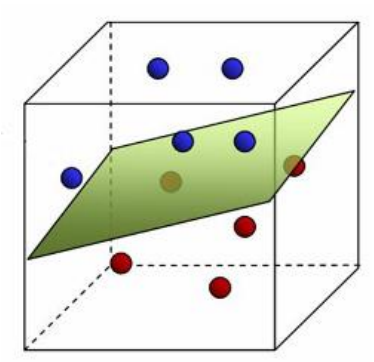
The basic idea of a separating hyperplane is straightforward. Hyperplanes are used in ML to define a plane that splits the categories of sample data in an N-dimensional space into neatly segregated areas. The objective is obviously to find the hyperplane that separates the categories in the best manner (in popular terms find the weight vector – see notation below). So again, an optimization problem.

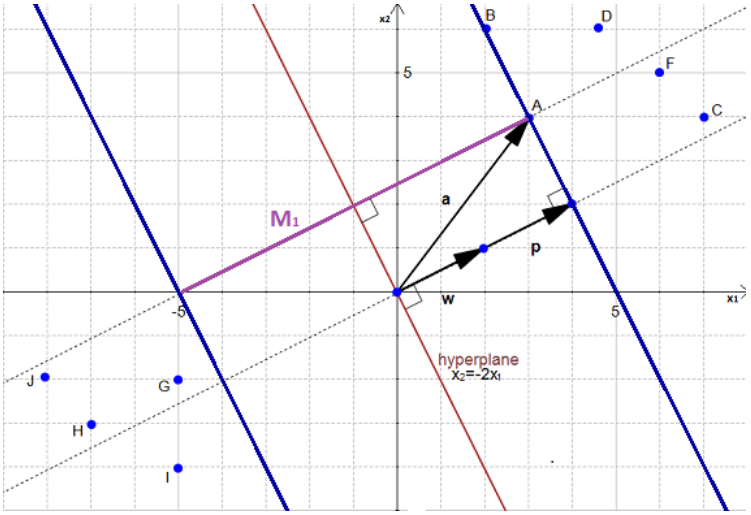
The equation of a hyperplane can be written as $\overline{w} \,.\, \bar{x}$ (see appendix – Math Recap)

In 2D, this is easy to understand. You try to find the "hyper-line" (there are an infinite number of regression lines imaginable in a sample) which is as far away as possible from the points closest to the hyper-line.

The hyperplane weight vector is always orthogonal (perpendicular in 2D) to the hyperplane, hence the name support vector $\overline{w}$. Image a steel support beam running from the hyperplane to one of the closest data points.

The following diagrams demonstrate the concepts in 3D and 2D.

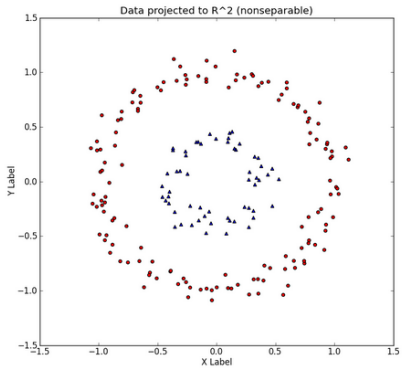


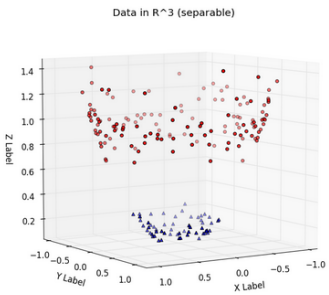


14.3.2. Kernel trick

It gets problematic when a hyperplane, which should separate the data, cannot be found. In the figure below there are data points belonging to the same category (the red triangles) which are enveloped by the other type of classifiers (the blue stars). You simply cannot define a line to separate these two categories. Unfortunately, the majority of data is not linearly separable. Such is life.

In this case we will uplift the sample data of N dimensionality into the N+1 dimension, via the so called “kernel trick”. Mathematicians have proved that a separating hyperplane can (always?) be found in the N+1 of even higher dimensional space. In the example diagram depicted below the data in 3D is separable whereas it was impossible to separate the same data in 2D. The ‘trick’ here is to consider the (nominal) result class as a the 3rd dimension (Z-Label on the picture).

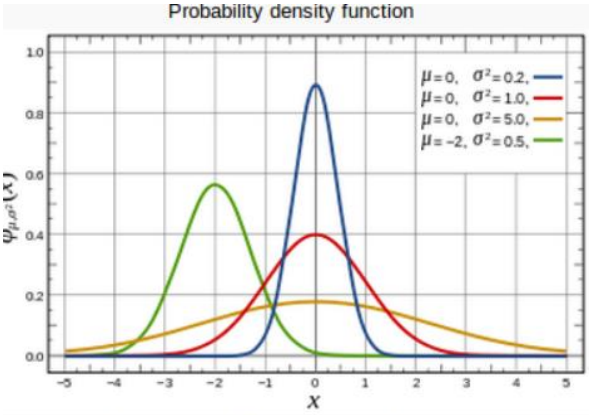




Source : https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d

NOTE – The SVM algorithm – by design or per chance – had the advantage that all operations performed on Kernels result in "inner products", i.e. matrix multiplications of dimensionality [1, n]. [N, 1] resulting in a single scalar value (a matrix of dimension [1, 1]). So, you are never confronted directly with the consequences of a higher dimensionality only with the results of operations performed on higher dimension elements. To put it more bluntly you only need to address formulas like K ($x_i.x_j$) (sometime written as K (x_i, x_j))

There is a choice of kernel trick transformations.

Formula	Comment
K ($x_i.x_j$) = $x_i.x_j$	Actually, this is the absence of a kernel function, i.e. just the dot product of 2 vectors.
K ($x_i.x_j$) = $(x_i.x_j + 1)^p$	Polynomial kernel
K ($x_i.x_j$) = $e^{-\frac{1}{2\sigma^2}\|\|x_i-x_j\|\|^2}$	Gaussian kernel; special case of the RBF Sigma is a free parameter. If you set it to 1, there are an unlimited number of dimensions in the kernel space. $\|\|x_i - x_j\|\|^2$ is the squared Euclidian distance between 2 vectors.  The red curve is the *standard normal distribution* Rule of thumb: Smaller sigma tends to make a local classifier; larger sigma tends to make a much more general classifier. http://haohanw.blogspot.com/2014/03/ml-how-sigma-matters-in-svm-rbf-kernel.html
K ($x_i.x_j$) = $e^{-\gamma\|\|x_i-x_j\|\|^2}$	Radial Basis Function (RBF). See Gaussian.
K ($x_i.x_j$) = $\tanh(\eta x_i.x_j + v)$	Sigmoid kernel (activation function in neural networks)

14.3.3. Lagrange multipliers

In mathematical optimization, the method of Lagrange multipliers is an approach for finding the local maxima and minima of a function subject to equality constraints (i.e. subject to the condition that

one or more equations have to be satisfied exactly by the chosen values of the variables) or by their opposites, i.e. inequality constraints.

An optimization problem is written or defined as follows. There is the objective function (minimize x) and the equality constraint to which the objective function must adhere (g(x) = 0)

$$minimize_x \;\; f(x)$$

$$subject\ to \;\; g(x) = 0$$

Lagrange (1736-1813) discovered that to find the minimum x (depending on g(x)=0) it appears that when that minimum of f is found the partial derivative of f and g are pointing in the same direction (have the same slope or inclination).

$$\nabla f(x) = \propto \nabla g(x)$$

Where $\propto$ is called the Lagrange multiplier.

To simplify the notation mathematician introduced the Lagrange function (curly L).

$$\mathcal{L}(x, \propto)$$

The Lagrange gradient formula can be put down as follows.

$$\nabla\mathcal{L}(x, \propto) = \nabla f(x) - \propto \nabla g(x)$$

So, if you want to find a minimum you 'just' solve the following, i.e. you go and find alpha

$$\nabla\mathcal{L}(x, \propto) \;=\; \nabla f(x) - \propto \nabla g(x) = 0$$

14.4. SVM algorithm

The SVM algorithm and mathematical background will not documented in this syllabus.

The major rationale for this decision is that to solve an SVM you need to rely on quadratic programming libraries or tools. These tools are most often to be obtained from third party software providers. Quadratic programming is a complicated topic and way beyond the scope of these introduction handouts, also given that the handouts attempt to provide functioning and stand-alone source code and given the complexity of quadratic programming, I just stepped away from describing the SVM.

14.5. Reader's guide

If you are interested in the SVM topic, you are advised to download reference document [06]. This document is an excellent introduction on hyperplanes, Lagrange multipliers, Kernel tricks, Karush-Kuhn-Tucker constraints, SVM and eventually the optimized SMO algorithm. It is a good idea to grasp the basics of SVM In order to understand SMO algorithm described in the next section.

Chapter 15. Sequential Minimal Optimization (SMO)

15.1. Introduction

SMO is a simplified and optimized SVM algorithm. It aims to identify relevant and optimized Lagrange multipliers (alphas and not weights or omegas).

SMO is quite recent. Platt created the algorithm in 1998.

15.2. Approach

Support Vector Machines and SMO in essence seek to solve an optimization problem for finding a best separating hyperplane (expressed by the vector W) on linearly separable sample data.
Remember: We want to find a hyperplane that is furthest away of all of the data points closest to the hyperplane.

Mathematicians define the above optimization problem more precise as follows: "Given a linearly separable set of training data find the optimal separating hyperplane by defining a normal vector W and bias b for which the geometric margin is the largest".

$$maximimize_{w,b}\ (minimum_{i=1..m}\ \gamma_i)$$

$$subject\ to\ \gamma_i \geq M \quad For\ each\ i = 1..m$$

Where

- M is the geometric margin $M = minimum_{i=1..m}\ \gamma_i$ So we want to maximize the hyperplane which is closest, so the minimum)
- $\gamma_i = y_i(\frac{w}{\|W\|}.x_i + \frac{b}{\|W\|})$ (this is a hyperplane equation normalized over the amplitude)
- Observe 1..m because we are using b
- this is an inequality constraint ($\gamma_i \geq M$)

The remainder of this section is an abridged version of how to solve the optimization problem for a hyperplane.

Assume that the above optimization problem can be re-expressed towards the following minimization problem.

$$minimize\ (w, b) \quad \frac{1}{2}\|W\|^2$$

Subject to

$$y_i(w.x_i) + b - 1 \geq 0\ \ for\ any\ i = 1..m$$

Where

- W is the weight vector or the hyperplane vector
- b is the bias.
- y are the labels and must be either -1 or 1

NOTE– Platt defined the SMO algorithm using the $\bar{W}\bar{X}$ + b (or $\bar{W}\bar{X}^T + b$ should you prefer to use matrices instead of vectors) convention for defining the hyperplane equation. Platt explicitly defined bias b (and not implicitly the w_0 notation which throughout these handouts is used).

As 'it happens', the above optimization can be solved using Lagrange multipliers. A bit insolent type of statement, considering that in reality the SVM and SMO problem were only nailed down in the 1980's and eventually only in 1998. Yet, without going into too much detail, 'it happens' that the SMO algorithm solves the following optimization problem.

The objective function is

$$minimize_{\propto} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \propto_i \propto_j y_i y_j K(x_i . x_j) - \sum_{i=1}^{m} \propto_i$$

Subject to the following inequality and equality constraints

$$\begin{cases} C \geq \propto_i \geq 0 \quad for\ any\ i = 1..m \\ \sum_{i=1}^{m} \propto_i y_i = 0 \end{cases}$$

Where

- $\propto$ (alpha) is the Lagrange multiplier. There is one Lagrange multiplier for each row. Rows with a Lagrange multiplier different from zero are in fact the data points out of which the support vectors start.
- C is the Soft Margin constant. A small C gives a wider margin at the cost of misclassification, a very large C does not accommodate for any tolerance at all. C is set via trial and error but remember that C is proper to a certain type of data set.
- K is the Kernel trick function. It is used to move from N dimension to N+1 dimension in order to solve non-linearly separable distributions.
- M is the number of rows in the sample data set
- Y is the labeled result (the category defined to be correct in the training data).
- NOTE – The Kernel trick is sometimes noted as follows (x_1, x_1) ; given that the Kernel trick is not really a dot product (except when there is no Kernel trick because then you need to use a dot product).

The trick is to find the optimal $\propto_1 and \propto_2$ (sometimes also referred to by $\propto_i and \propto_j$) and then backwards derive W and b. We already know from the introduction on SVM that we should try to find the Lagrange multiplier where the partial derivative or gradient is zero.

Again, without going into too much detail. Once you have the Lagrange multipliers, you can backtrack to W (the weight vector) and the bias b.

$$W = \sum_{i=1}^{m} \propto_i y_i x_i$$

The weight vector has the dimensionality of the sample data set, i.e. stated rather more bluntly N is the number of variables (features, columns, etc.) in the training data set and M is the number of rows in the training data.

Bias b can be computed once the weight vector is knows as follows (math proof can be found in reference document [06]).

$$b = y_i - w \,.\, x_i$$

NOTE – the algorithm, which is described in the next section, uses another method, which was set forward by authors enhancing Platt's original SMO but also before by Vapnik. They state that b is the average of the nearest positive support vector and the furthest negative support vector. Just in case you forgot, the labels must be in the range between -1 and 1.

$$b = \frac{maximum_{y_i} = -1K(w.x_i) + \; minimum_{y_i} = 1K(w.x_i)}{2}$$

15.3. Algorithm

The SLO algorithm breaks down the quadratic optimization problem in small and sizeable problems to solve. We assume a vector (or a list or an array) of Lagrange multipliers, we initialize all multipliers to zero and we take two multipliers ($\propto_1, \propto_2$) which we will now start tweaking (actually we apply gradients changes to $\propto_1^{new}$ and $\propto_2^{new}$ – see next section). We keep on changing the two chosen Lagrange multipliers until the objective function reaches a minimum.

We then take two other multipliers (they must be different from the first two) and keep on changing those multipliers until the objective function again reaches a minimum. We continue doing this until there are no more changes occurring on the multipliers.

Hence the name sequential: SMO solves the problem via series of smaller problems in a stepwise or sequential manner.

The advantage of SMO is that you do not need a quadratic solver tool.

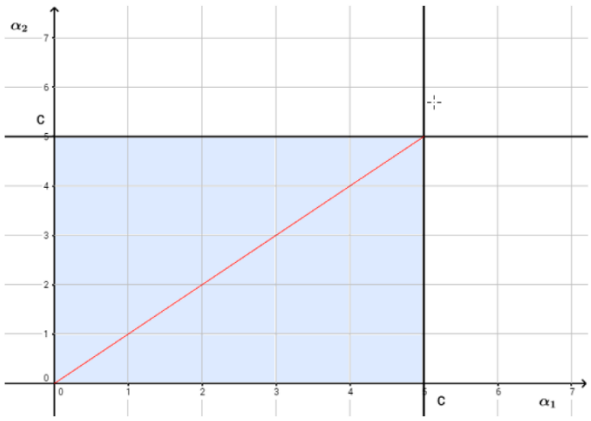
15.4. Non-optimized version of SMO

The trick to remember is that when you choose the two multipliers the (Karush-Kuhn-Tucker) conditions must still be met

$$\begin{cases} C \geq \propto_i \geq 0 \quad for\ any\ i = 1\,..\,m \\ \sum_{i=1}^{m} \propto_i y_i = 0 \end{cases}$$

The first constraint is called the 'box constraint'. Whichever multiplier is chosen, you know that it must be within the boundaries of a box of width and height C.

The second condition boils down to a line constraint, the multipliers must be on the pink diagonal in the box.



Calculate the second Lagrange multiplier van as follows. Remember that there is one Lagrange multiplier for each row in the sample data set. (I left out the proof)

$$\propto_2^{new} = \propto_2^{previous} + \frac{y_2(E_1 - E_2)}{K(x_1.x_1) + K(x_2.x_2) - 2K(x_1.x_2)}$$

Where

- K is the Kernel trick function
- y is the actual label of the vector related to the 2nd Lagrange multiplier
- x_1 is the variable vector attached to the first multiplier
- x_2 is the variable vector attached to the second multiplier

The first Lagrange multiplier can be calculated as follows (proof left out)

$$\propto_1^{new} = \propto_1^{previous} + y_1 y_2(\propto_2^{previous} - \propto_2^{previous})$$

NOTE – this algorithm also uses a tolerance value. This is the Epsilon value reappearing. We discussed it at the beginning of Linear Regression algorithm.

Initialize the weight vector to all zero values

Loop through all weight vectors (so from 0 to N) until there no more changes on the multipliers

Calculate the predicted value with current weight: w.$x_1 + b$ (by applying the hypothesis function to current set of multipliers and using the formula to derive W from the set of multipliers)

Determine the error $E_1 = predicted - y_1$

We want to find multipliers, which are not bounded to the box, i.e. we need a set of multipliers that must be changed. This can be expressed as follows

- If $y_1 E_1$ *is smaller than negative tolearance and smaller than* C; *or*
- If $y_1 E_1$ *is* larger than the positive tolerance and larger than C

Pick another multiplier αj at random

Calculate the predicted value with current weight: w.$x_2 + b$

Determine the error $E_2 = predicted - y_2$

We must also find bounds L and H:

- If y_1 differs from y_2 then L = maximum(0, $\propto_2 - \propto_1$), H = minimum(C, C + $\propto_2 - \propto_1$)
- If y_1 equals y_2 then L = maximum(0, $\propto_2 + \propto_1 - C$), H = minimum(C, $\propto_2 + \propto_1$)

If (L = H) then continue (i.e. pick another α)

Now calculate the second multiplier using the formula discussed above

Clip the second multiplier (if $\propto_2$is larger than H then H, if smaller then L then L)

Calculate the first multiplier (using the formula above)

Finally, we derive the bias b; we calculate two possible biases

- $b_1 = b - E_1 - y_1(\propto_1^{new} - \propto_1^{previous})K(x_1.x_1) - y_2(\propto_2^{new} - \propto_2^{previous})K(x_1.x_2)$
- $b_2 = b - E_2 - y_2(\propto_1^{new} - \propto_1^{previous})K(x_1.x_2) - y_2(\propto_2^{new} - \propto_2^{previous})K(x_2.x_2)$

Moreover, we select b as follows

- If $0 < \propto_i < C$ *then* $b1$
- If $0 < \propto_2 < C$ *then* $b2$
- Otherwise ($b_1 + b_2$) / 2

Set a flag that multipliers have been adapted; and iterate to the next i

15.4.1. Python implementation

NOTE - You can find a Python based implementation and abbreviated algorithm description at this URL https://www.codeproject.com/Articles/1267445/An-Introduction-to-Support-Vector-Machine-SVM-and

15.4.2. Implementation

15.4.2.1. ARF File

The IRIS sample data set has been selected here to demonstrate the algorithm. The set has three possible results, so the data was previously modified so that a “one against all” approach could be used (Setosa against all others). The same approach as was demonstrated in the Logistic regression example.

15.4.2.2. Result

The result is not too bad, 100% accuracy on the training and test set.

```
- <SMOModelGeneral>
    <ModelApplicationDomain>General</ModelApplicationDomain>
    <ARFFFileName>c:\temp\cbrTekStraktor\Tutorial\MachineLearning\SandBox\Iris - Tweaked.ARFF</ARFFFileName>
    <KernelTrickType>NONE</KernelTrickType>
    <NumberOfFeatures>5</NumberOfFeatures>
    <MaxCycles>40</MaxCycles>
    <Tolerance>0.001</Tolerance>
    <SoftMargin>0.6</SoftMargin>
    <NumberOfSupportVectors>14</NumberOfSupportVectors>
  </SMOModelGeneral>
- <SMOModelEvaluation>
  - <TrainedModel>
      <TrainedModelAcurracy>1.0</TrainedModelAcurracy>
      <TrainedModelEntries>130</TrainedModelEntries>
    - <![CDATA[

                    IRIS-SETOSA      OTHER ---> Actual
                    -----------  ----------  ----------
          IRIS-SETOSA   [      87][       0]
          OTHER         [       0][      43]

                     Precision  Sensitivity    FMeasure
                    -----------  -----------  ----------
          IRIS-SETOSA   [  1,000000][   1,000000][  1,000000]
          OTHER         [  1,000000][   1,000000][  1,000000]
      ]]>
    </TrainedModel>
  - <TestedModel>
      <TestedModelAcurracy>1.0</TestedModelAcurracy>
      <TestedModelEntries>20</TestedModelEntries>
      <TestSetPercentage>13</TestSetPercentage>
    - <![CDATA[

                    IRIS-SETOSA      OTHER ---> Actual
                    -----------  ----------  ----------
          IRIS-SETOSA   [      13][       0]
          OTHER         [       0][       7]

                     Precision  Sensitivity    FMeasure
                    -----------  -----------  ----------
          IRIS-SETOSA   [  1,000000][   1,000000][  1,000000]
          OTHER         [  1,000000][   1,000000][  1,000000]
      ]]>
    </TestedModel>
```

15.4.2.3. Java code

This is a possible implementation in Java. I left out the routines that perform matrix calculation for legibility purposes. Java is quite cumbersome to perform this type of operations.

```
for(int cycle=0;cycle<MaxNbrOfCycles;cycle++)
{
        int alphaPairChanges=0;
        for(int i=0;i<nrows;i++)
```

```
        {
                double fXi =  getPredictedValue( dataMatrix , i , alphaMatrix ,
resultMatrix , b);
                if( Double.isNaN(fXi) ) return false;
                double Ei = fXi - resultMatrix.getValues()[i][0];  // Mind the
correct order

                // see whether Lagrange multipliers can be changed
                if( (((resultMatrix.getValues()[i][0]*Ei) < (0-tolerance)) &&
(alphaMatrix.getValues()[i][0] < C))  ||
                    (((resultMatrix.getValues()[i][0]*Ei) > tolerance) &&
(alphaMatrix.getValues()[i][0] > 0)) )
                {
                      int j = mrout.getRandomNumberExcluding( nrows , i );
                      if( j < 0 ) { do_error( "Fetching random number"); return
false; }
                      double fXj = getPredictedValue( dataMatrix , j , alphaMatrix
, resultMatrix , b);
                      if( Double.isNaN(fXj) ) return false;
                      double Ej = fXj - resultMatrix.getValues()[j][0];
                      //do_log( 1 , "      " + cycle + " fXj=" + fXj + " Ej=" +
Ej);
                      //
                      double alphaIOld = alphaMatrix.getValues()[i][0];
                      double alphaJOld = alphaMatrix.getValues()[j][0];
                      double labelI = resultMatrix.getValues()[i][0];
                      double labelJ = resultMatrix.getValues()[j][0];
                      double L = Double.NaN;
                      double H = Double.NaN;
                      //  alphas must always between 0 and C
                      if( labelI != labelJ ) {
                            L = Math.max(  0 , (alphaJOld - alphaIOld) );
                            H = Math.min(  C , C + (alphaJOld - alphaIOld) );
                      }
                      else {
                            L = Math.max(  0 , (alphaJOld + alphaIOld) - C );
                            H = Math.min(  C , (alphaJOld + alphaIOld) );
                      }
                      if( L==H ) {
                            //do_log( 1 , "H==L");
                            continue;
                      }
                      // 2nd lagrange multiplier - begin met noemer
                      double dxixi = getXiXj( dataMatrix , i , i);
                       double dxjxj = getXiXj( dataMatrix , j , j);
                       double dxixj = getXiXj( dataMatrix , i , j);
                       double eta = dxixi + dxjxj - ((double)2 * dxixj);
                      if( eta <= 0 ) {  //  Why ??
                       continue;
                      }
                      //  2nd Lagrange multiplier - alpha2
                      double alphaJ = alphaJOld + ((labelJ * (Ei - Ej)) / eta);
                      alphaJ = clipAlpha( alphaJ , H , L );
                      alphaMatrix.setValue( j , 0 , alphaJ );
                      //
                      if( Math.abs( alphaJ - alphaJOld ) < 0.00001 ) {
                           continue;
                      }
                      //  1st Lagrange multiplier   alhpa + yiyj(alph2old ) alpha2)
```

```
                    double alphaI = alphaIOld + (labelI * labelJ * ( alphaJOld -
alphaJ));
                    alphaMatrix.setValue( i , 0 , alphaI );
                    //  B
                    double b1 = b - Ei - (labelI * (alphaI - alphaIOld) * dxixi)
- (labelJ * (alphaJ - alphaJOld) * dxixj);
                    double b2 = b - Ej - (labelI * (alphaI - alphaIOld) * dxixj)
- (labelJ * (alphaJ - alphaJOld) * dxjxj);
                    //
                    if( (0 < alphaI) && (C > alphaI) ) b = b1;
                    else
                    if( (0 < alphaJ) && (C > alphaJ) ) b= b2;
                    else {
                          b = ( b1 + b2) / (double)2;
                    }
                    alphaPairChanges++;

              }
              else { // no change
                    //do_log( 1 , "No change possible");
              }

  } // rows
    // no changes so cycle reset
   if( alphaPairChanges != 0 ) cycle=0;
  }
```

15.5. Optimized version of SMO

The above algorithm is very slow in part because we assess every possible combination of the Lagrange multipliers.

Platt defined a smart approach for finding the best performing multipliers. The so-called Full Platt SMO algorithm consists of an outer loop for choosing the first Lagrange multiplier and an inner loop for choosing the second multiplier (or alpha).

The outer loop switches between single passes over the entire dataset and single passes over non-bound multipliers (the alphas which are not bound to the 0 and C limit). The run over the entire dataset is as easy as it sounds, i.e. just loop over every alpha in the set. The pass over the non-bound set starts by creating an exclusion list of multipliers that won't change.

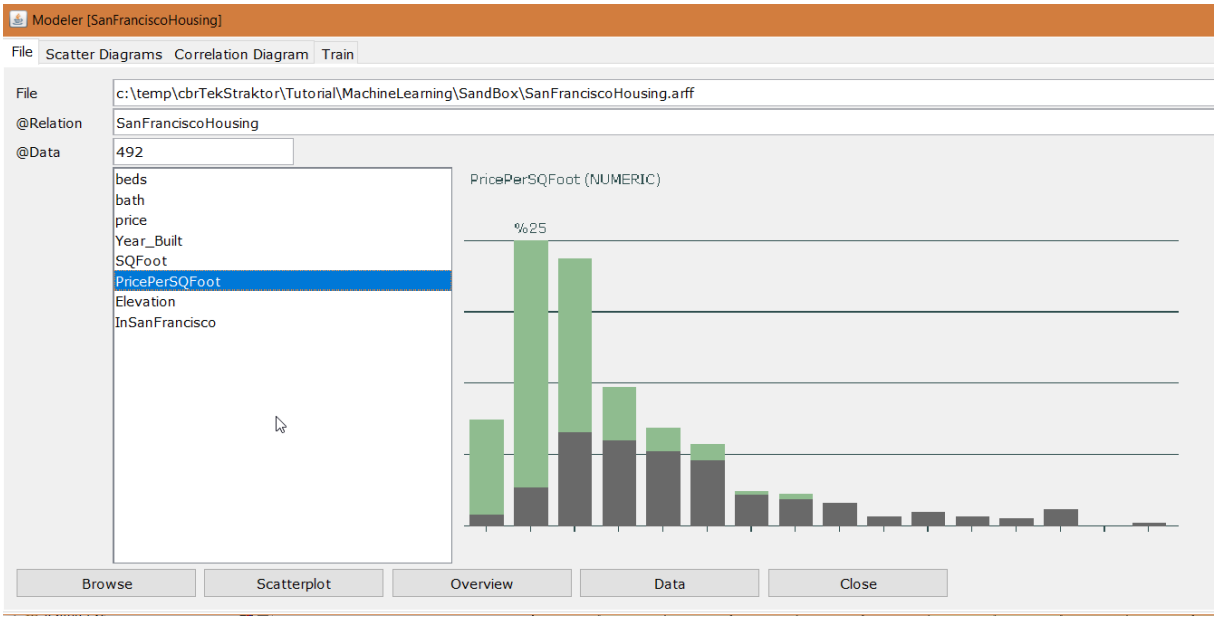
The second Lagrange multiplier is selected once we have the first one and is chosen in such a way that it will maximize the step size during optimization. In the simplified SMO, we calculated the error E2 after choosing j randomly. Now we are going to keep a track of the error values and choose the set of alphas that maximize the step size, stated differently we pick the set of alphas with the largest difference between E1 and E2.

15.5.1. Implementation

This section provides a few code snippets of a possible Java implementation. It will give you the gist of the way to implement the full Platt algorithm.

15.5.1.1. ARF File

In order to demonstrate the SMO, the San Francisco Housing data set was taken. The idea is to predict whether a house is located in SF or not based on the following features.



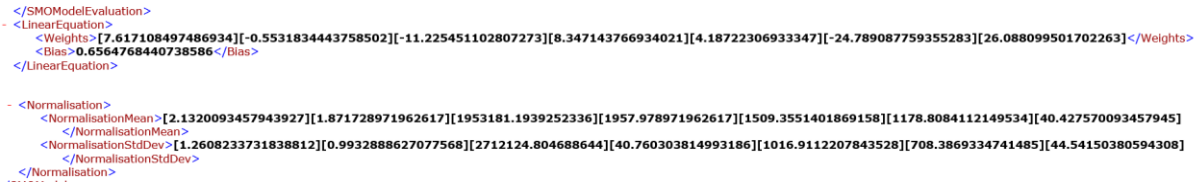
15.5.1.2. Results

Results for a Gaussian kernel function and sigma of four, is around 86%.

```
- <SMOModelGeneral>
    <ModelApplicationDomain>General</ModelApplicationDomain>
    <ARFFFileName>c:\temp\cbrTekStraktor\Tutorial\MachineLearning\SandBox\SanFranciscoHousing.arff</ARFFFileName>
    <KernelTrickType>GAUSSIAN</KernelTrickType>
    <NumberOfFeatures>8</NumberOfFeatures>
    <MaxCycles>40</MaxCycles>
    <Tolerance>0.001</Tolerance>
    <SoftMargin>0.6</SoftMargin>
    <NumberOfSupportVectors>226</NumberOfSupportVectors>
  </SMOModelGeneral>
- <SMOModelEvaluation>
  - <TrainedModel>
      <TrainedModelAcurracy>0.8691588785046729</TrainedModelAcurracy>
      <TrainedModelEntries>428</TrainedModelEntries>
    - <![CDATA[


                        0        1 ---> Actual
                  ---------- ---------- ----------
            0     [      186][        9]
            1     [       47][      186]

                   Precision  Sensitivity   FMeasure
                  ---------- ---------- ----------
            0     [ 0,953846][  0,798283][ 0,869159]
            1     [ 0,798283][  0,953846][ 0,869159]
      ]]>
    </TrainedModel>
  - <TestedModel>
      <TestedModelAcurracy>0.78125</TestedModelAcurracy>
      <TestedModelEntries>64</TestedModelEntries>
      <TestSetPercentage>13</TestSetPercentage>
    - <![CDATA[


                        0        1 ---> Actual
                  ---------- ---------- ----------
            0     [       23][        2]
            1     [       12][       27]

                   Precision  Sensitivity   FMeasure
                  ---------- ---------- ----------
            0     [ 0,920000][  0,657143][ 0,766667]
            1     [ 0,692308][  0,931034][ 0,794118]
      ]]>
    </TestedModel>
```

15.5.1.3. Model

The model just stores the weight vector (and its elements) and the bias and the normalization mean and standard deviation. It is a pre-requisite to process new data using the same normalization parameters (sigma and delta) as the ones of the trained model.



15.5.1.4. Java code

15.5.1.4.1. Outer loop

You can see how the logic switches between scanning the entire set and only the set of bound Lagrange multipliers.

```
// FULL Platt SMO version
int alphaPairChanges=0;
boolean entireSet=true;
int cycle = 0;
while ( (cycle <dto.getMaxCycles()) || (alphaPairChanges>0) || (entireSet==true) )
{
	alphaPairChanges=0;
	if( entireSet ) {
		for(int i=0;i<dto.getNbrOfRows();i++)
		{
		int change = innerLoop( i , dto );
		if( change < 0 ) { do_error( "innerloop error I"); return false; }
		alphaPairChanges += change;
		}
		cycle++;
	}
	else {
		boolean[] nonBoundList = new boolean[ dto.getNbrOfRows() ];
		for(int i=0;i<nonBoundList.length ; i++ ) nonBoundList[i]=false;
			cmcMatrix alphaMatrix = dto.getLagrangeMultiplierMatrix();
			for(int i=0;i<alphaMatrix.getNbrOfRows();i++)
			{
			if( (alphaMatrix.getValues()[i][0] > 0) &&
(alphaMatrix.getValues()[i][0] < dto.getC()) ) nonBoundList[i]=true;
			}
				//
			for(int i=0;i<nonBoundList.length;i++)
			{
			int change = innerLoop( i , dto );
			if( change < 0 ) { do_error( "innerloop error II"); return
false; }
			alphaPairChanges += change;
			}
			cycle++;
	}
	if( entireSet == true ) entireSet = false;
	else
	if( alphaPairChanges == 0 ) entireSet = true;
}
```

15.5.1.4.2. Inner loop

The inner loop is almost identical to the one of the non-optimized SMO. Error and exception checking lines have been removed.

The code relies on the following functions

- calculateError(); which calculates the predicted values and determines the difference with the anticipated result. The predicted result calls the kernel trick.
- getNextIndexForMultiplier(); which gets the optimized (Ei,Ej) combination
- getKernelTrickedXiXj(); which is just a function to make the logic more readable. It performs K(xi,xj)
- recalculateAndStoreError(); which calculates all errors and stores the so that the maximum (Ei,Ej) can be determined

```
double Ei = calculateError( i , dto );
// see whether Lagrange multipliers can be changed
if( (((resultMatrix.getValues()[i][0]*Ei) < (0-dto.getTolerance())) &&
(alphaMatrix.getValues()[i][0] < dto.getC()))  ||
(((resultMatrix.getValues()[i][0]*Ei) > dto.getTolerance()) &&
(alphaMatrix.getValues()[i][0] > 0)) )
{
  int j = getNextIndexForMultiplier( i , dto );
  double Ej = calculateError( j , dto );
  if( Double.isNaN(Ej) ) return -1;
  //
  double alphaIOld = alphaMatrix.getValues()[i][0];
  double alphaJOld = alphaMatrix.getValues()[j][0];
  double labelI = resultMatrix.getValues()[i][0];
  double labelJ = resultMatrix.getValues()[j][0];
  double L = Double.NaN;
  double H = Double.NaN;
  //  alphas must always between 0 and C
  if( labelI != labelJ ) {
   L = Math.max(  0 , (alphaJOld - alphaIOld) );
   H = Math.min(  dto.getC() , dto.getC() + (alphaJOld - alphaIOld) );
  }
  else {
   L = Math.max(  0 , (alphaJOld + alphaIOld) - dto.getC() );
   H = Math.min(  dto.getC() , (alphaJOld + alphaIOld) );
  }
  if( L==H ) return 0;
  // 2nd lagrange multiplier - begin met noemer
  double dKxixi = getKernelTrickedXiXj( dataMatrix , i , i , dto);
  double dKxjxj = getKernelTrickedXiXj( dataMatrix , j , j , dto);
  double dKxixj = getKernelTrickedXiXj( dataMatrix , i , j , dto);
  double eta = dKxixi + dKxjxj - ((double)2 * dKxixj);
  if( eta <= 0 )  return 0;

 //  2nd Lagrange multiplier - alpha2
  double alphaJ = alphaJOld + ((labelJ * (Ei - Ej)) / eta);
  alphaJ = clipAlpha( alphaJ , H , L );
  dto.getAlphaMatrix().setValue( j , 0 , alphaJ );
  if( recalculateAndStoreError( j , dto ) == false ) return -1;
  //
  if( Math.abs( alphaJ - alphaJOld ) < 0.00001 ) return 0;

 //  1st Lagrange multiplier   alhpa + yiyj(alph2old ) alpha2)
```

```
  double alphaI = alphaIOld + (labelI * labelJ * ( alphaJOld - alphaJ));
  dto.getAlphaMatrix().setValue( i , 0 , alphaI );
  if( recalculateAndStoreError( i , dto ) == false ) return -1;
 //  B
  double b1 = dto.getBias() - Ei - (labelI * (alphaI - alphaIOld) * dKxixi) -
(labelJ * (alphaJ - alphaJOld) * dKxixj);
  double b2 = dto.getBias() - Ej - (labelI * (alphaI - alphaIOld) * dKxixj) -
(labelJ * (alphaJ - alphaJOld) * dKxjxj);
  //
  if( (0 < alphaI) && (dto.getC() > alphaI) ) dto.setBias(b1);
  else
  if( (0 < alphaJ) && (dto.getC() > alphaJ) ) dto.setBias(b2);
  else {
    dto.setBias( ( b1 + b2) / (double)2 );
  }
   return 1;
 }
 else { // no change possible
 return 0; }
```

15.5.1.4.3. Kernel trick

Just in case you wondered how the Kernel trick looks like in Java. Error checking and exception handlers have been removed to make the code snippet more readable. In essence a matrix and vector multiplication. You just need to keep track of aligning columns and rows.

```
cmcMatrix kernelMatrix = null;
// Which kernel
switch( dto.getKernelTrickTipe() )
{
case NONE : { kernelMatrix = vrout.multiplyMatrix( XMatrix ,
vrout.transposeMatrix( XVector ) ); break; }
case GAUSSIAN : ;
case RBF  : {
double factor = 0;
if( dto.getKernelTrickTipe() == cmcMachineLearningEnums.KernelType.GAUSSIAN ) {
  factor = 1 / ((double)2 * cmcMachineLearningConstants.SMO_GAUSSIAN_SIGMA *
cmcMachineLearningConstants.SMO_GAUSSIAN_SIGMA);
}
if( dto.getKernelTrickTipe() == cmcMachineLearningEnums.KernelType.RBF ) {
 factor = cmcMachineLearningConstants.SMO_RBF_GAMMA;
}

double[][] rbf = new double[XMatrix.getNbrOfRows()][1];  // m,1
for(int i=0;i<XMatrix.getNbrOfRows();i++)
{
 double deltaDistance = 0;
 for(int j=0;j<XMatrix.getNbrOfColumns();j++)
 {
  double dd = XMatrix.getValues()[i][j] - XVector.getValues()[0][j];    //
euclidian = som vd kwadraten verschil
  deltaDistance += ( dd * dd );
 }
 double term = 0 - (deltaDistance * factor);
```

```
 rbf[i][0] = Math.exp( term );
 }
 kernelMatrix = new cmcMatrix( rbf );
 break;
 }
 default : { do_error( "Unsupported Kernel [" + dto.getKernelTrickTipe() + "]");
break; }
 }
 return kernelMatrix;
```

Chapter 16. Neural Network – Perceptron

16.1. Definition

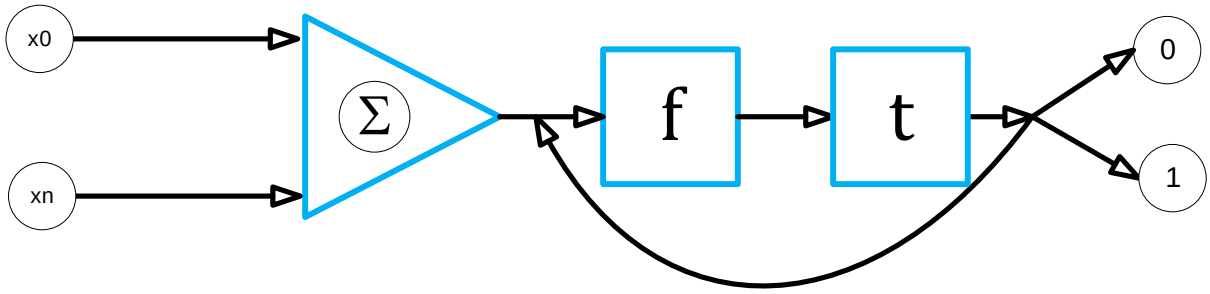
A perceptron is a classifier for 2 classes {C1, C2} or {-1, 1}. It relies on feedback provided by a loss function. An alternative name is Perceptron Learning Algorithm (PLA).

16.2. History

The first neural network algorithm was described in 1943 by Pitts in a paper "A logical calculus of ideas immanent in nervous activities". The first perceptron first implementation was in 1957 by Frank Rosenblatt. There were then even hardware implementations of the algorithm.

16.3. Concept

By now, it should have occurred to you that Machine Learning algorithms share a standard pattern. This very same pattern is also at the core of the perceptron.



The perceptron algorithm contains

- An aggregator (sigma), which aggregates the inputs from various input variables {x0, x1, .. , xn}
- An activation or transfer function f($\sum \bar{w}^T \bar{x}$).
- A perceptron also limits the regression equation to the first degree so limited to (w_0, w_1)
- A step function (t) which transforms the output of f into {-1,1}

NOTE – Although F and T have distinct functionality, F and T are often combined into single building block.

The differences between perceptron and logistic regression are

- The output function. The perceptron uses the step function (-1,1), whereas the logistic regression uses the sigmoid function.

- The loss function. Perceptron uses the Mean Square Error (MSE). The Logistic regression uses the Loss Entropy function.

Therefore, the optimization of the perceptron is similar to the optimization of the Linear Regression.

$$J(w_1, w_0) = \frac{1}{2M}\sum_{i=1}^{M}(y_i - T(w_1 x_{1i} + w_0))^2$$

M is the number of samples

The reasoning is the same; there will be two partial derivatives

$$\begin{cases} w_{0_{k+1}} \leftarrow w_{0_k} - \alpha \frac{1}{M}\sum_{i=1}^{M} T\left(w_1 x_{1i} + w_0 - y_i\right)1 \\ w_{1_{k+1}} \leftarrow w_{1_k} - \alpha \frac{1}{M}\sum_{i=1}^{M} T\left(w_1 x_{1i} + w_0 - y_i\right) x_{1i} \end{cases}$$

The perceptron formalized notation is

$$\overline{w_{k+1}} = \overline{w_k} - \alpha \sum^{N} \nabla J$$

$$\overline{w_{k+1}} = \overline{w_k} + \nabla \sum_{i=1}^{M} w^T x_i t_i$$

Where

- M is the number of samples
- N is the number of dimensions (features, columns)
- M is the dimensionality, in the case of perceptron 2, so M=1
- w_1 is the slope and w_0 is the bias (or intercept)
- w^T is the transposed weight matrix

16.4. Constraints

A perceptron is still a linear classifier and just like any other linear classifier, (see the discussion on linear separability) a perceptron cannot solve non-linear problems. In effect, it is possible that the algorithm does not produce a result.

16.5. Pseudo code

- Step 1. Initiate the weights randomly
- Step 2. Compute y using the current weights (keep in mind that y I either -1 or 1)
- Step 3. If all calculated (predicted) classes are equal to the actual class or category, then stop
- Step 4. For all samples where the calculated y differs from the actual class or category perform step 5
- Step 5. Adjust the weights whereby the adjustment: Δ weight = actual label . x_i. This means that you pick a random sample which misclassified (i), fetch the actual class y_i and substract the calculated value of y to get the updated set of weights and go back to Step 2

16.6. Java

Implementation is straight-forward. The calcHypothesisAndGetMatch() function is bespoke to the package implemented and calculates the y value (w.x) and runs this through the step function to fix the result to -1 or 1. It returns a list of misclassified samples.

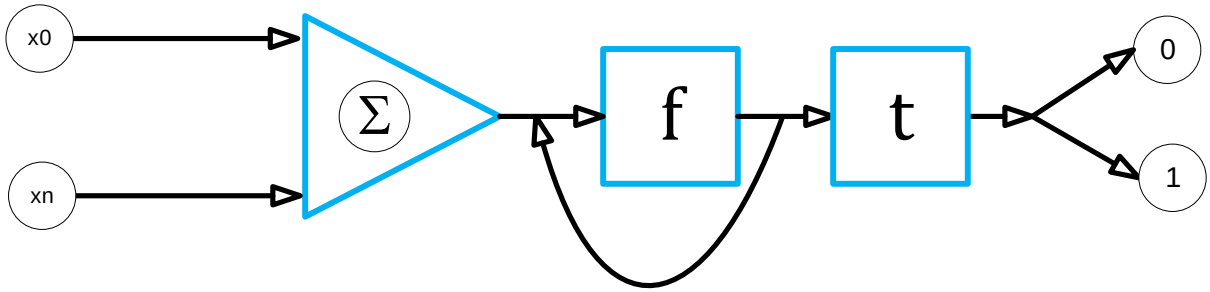
```
for(int i=0;i<MAX_ATTEMPTS;i++)
{
  attempts++;
  classifiedCorrectly = calcHypothesisAndGetMatch( w , augmentedlist ,
anticipatedlist , false );
  int nerrors=0;
  for(int j=0;j<classifiedCorrectly.length;j++)
  {
        if( classifiedCorrectly[j] == false ) nerrors++;
  }
  trend[attempts-1] = nerrors;
  if( nerrors == 0 ) { found = true;  break;  }
  // make a list of all vectors for which the function did not match
  int[] pickList = new int[ nerrors ];
  nerrors=0;
  for(int j=0;j<classifiedCorrectly.length;j++)
  {
  if( classifiedCorrectly[j] == false ) { pickList[nerrors] = j; nerrors++; }
  }
  // now just fetch a random vector for which the function did not match
  int pdx = rn.nextInt((nerrors+1)*97) % nerrors;
  int idx = pickList[pdx];
  // update rule
  // get the random vector for which function failed and multiply with its
(single) anticipated result (1,-1)
  cmcVector zz = vrout.scalarMultiplicationVector( augmentedlist[ idx ] ,
anticipatedlist[idx] );
  // modify all weight by adding the result of the update rule.
  w = vrout.addVectors( w , zz );
}

//-------------------------------------------------------------
private boolean[] calcHypothesisAndGetMatch( cmcVector w , cmcVector[]
augmentedlist , double[] anticipatedlist , boolean verbose)
//-------------------------------------------------------------
{
boolean[] classifiedCorrectly = new boolean[ anticipatedlist.length ];
```

```
for(int i=0 ; i<augmentedlist.length ; i++ )
{
 double dot = vrout.dotproduct( w , augmentedlist[i] );
 int mult = ( dot < 0 ) ? -1 : 1;  // zero is 1 ?? een punt op de hyperplane is
ok?
 classifiedCorrectly[ i ] = ((int)anticipatedlist[i] * mult == 1 ) ? true :
false;  // -1.-1 en 1.1 OK
 return classifiedCorrectly;
}
```

Chapter 17. Neural Network – ADALINE

17.1. Concept

The Adaline (Adaptive Linear Neuron or later Adaptive Linear Element) algorithm is a modification of the Perceptron. The perceptron algorithm adjusts the weight vector by performing a Mean Squared Loss function on the results of the T function (-1,1). Adaline performs the adjustment on the values of the activation function (f), so the values prior to be being submitted to the step function T.



17.2. Origin

It was developed by Bernard Widrow and Ted Hoff at Stanford University in 1960.

17.3. Formula

The Adaline formalized notation is (perceptron without the step function t_n)

$$\overline{w_{k+1}} = \overline{w_k} - \alpha \sum^{N} \nabla J$$

$$\overline{w_{k+1}} = \overline{w_k} + \nabla \sum_{i=1}^{M} w^T x_i$$

Where

- M is the number of samples
- N is the dimensionality, in the case of adaline limited to 2; so M=1 or $w_1 x_1 + w_0$
- w1 is the slope and w0 is the bias.
- W.T is the transposed weight matrix

17.4. Intuitive notation

IMPORTANT – The gradient comprises a multiplication with X (input variables). In the final formula right above, X is not explicitly mentioned, so you might accidentally overlook it.

I therefore tend to rewrite the formula in matrix notation also to make the coding easier and to check the alignment of the dimensions when multiplying.

$$W - \alpha ((Y - (W X^T)) X)$$

17.5. Pseudo code

- Step 1. Initiate the weights randomly
- Step 2. Compute y using the current weights but do not apply the step; assume this to be O
- Step 3. Compute the error using a Mean Squared loss formula (L2)
- Step 4. Adjust all weights using the gradient descent approach.
- Step 5. Continue until adjustments are below a predefined cut-over value or when a number of iterations has been reached.

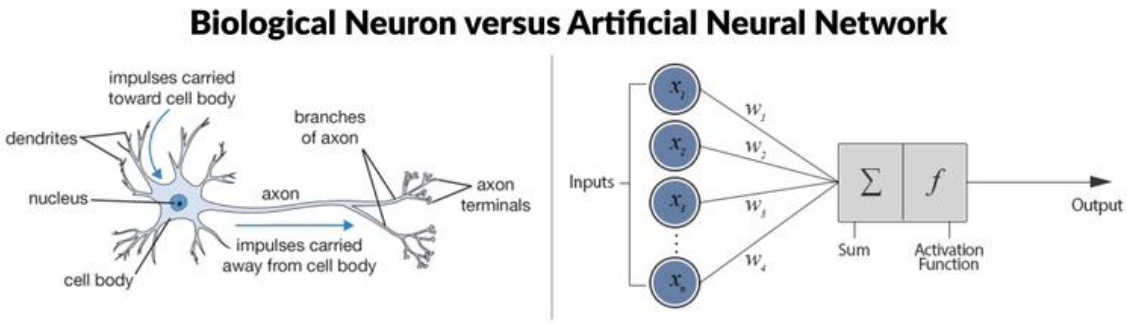
NOTE – Another way of looking at Adaline is : Logistic Regression without applying the sigmoid function.

17.6. Java implementation

The Java code is almost identical to the code for a perceptron. It suffices to leave out the step function (int mult = (dot < 0) ? -1 : 1;) in the procedure calcHypothesisAndGetMatch().

Chapter 18. Neural networks Biological

The following picture shows the correlation of a simple neural network and the interaction of neurons in a brain.



Source image: Keras web site.

Chapter 19. Where to find the source code

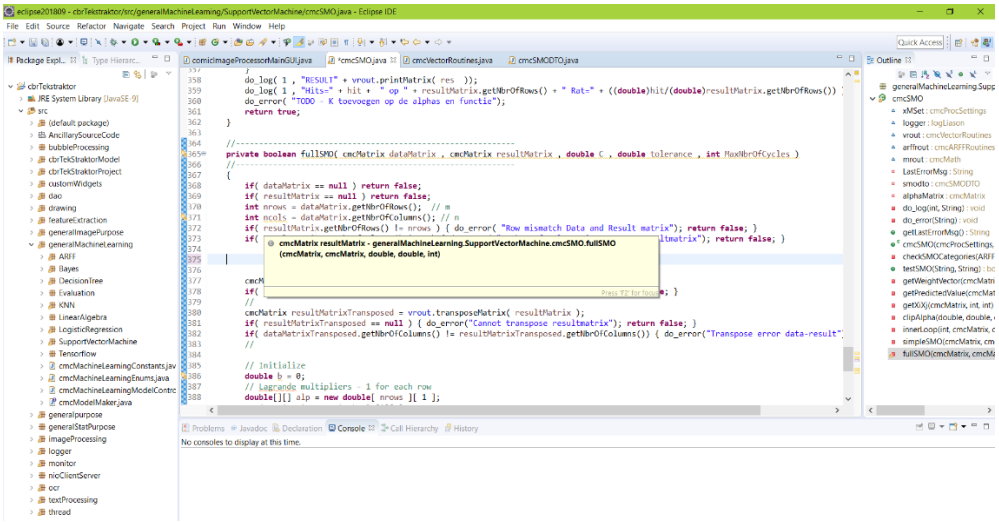
The Java source code used in these handouts can be found on Github.

https://github.com/koenberton/AxiansSummerSchool2021

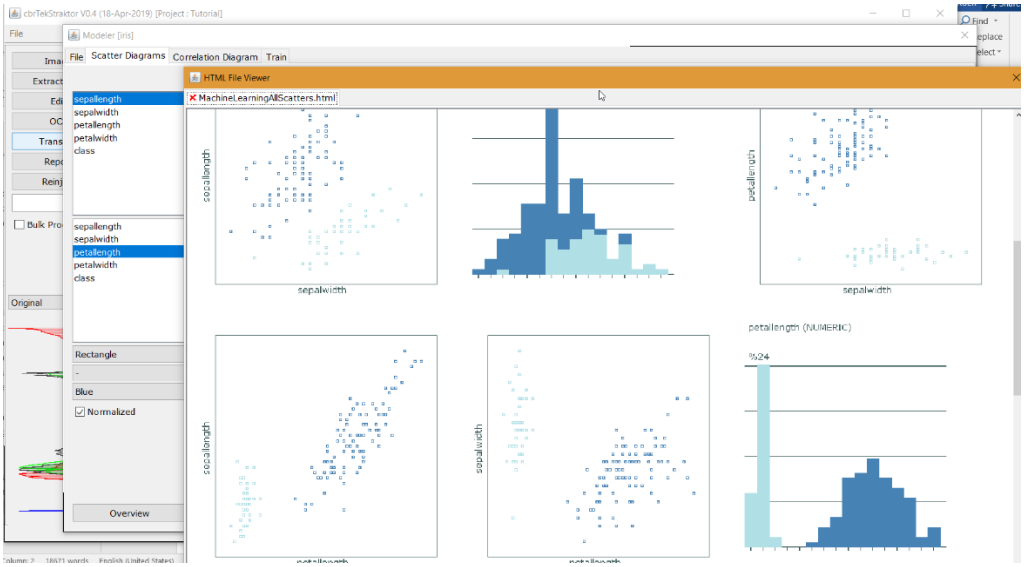
The ML source code is part of a larger project (cbrTekStraktor), which aims to extract plain text from comic book bubbles. This application relies - unsurprisingly - on image processing, statistical and machine learning algorithms.

The ML source code is located in the src/generalMachineLearning package.

In order to access the Machine Learning modules, launch the application with the command line parameter -ML



The ML application can be found under File > Classifiers > ML Modeler. Screenshot of modeler showing the Iris data set scatterplots.



Chapter 20. Appendix – Math recap

A good recap on Linear algebra can be found at https://introcs.cs.princeton.edu/java/95linear/

Vector stuff

A vector $\bar{v} = (x_1, x_2, .., x_n)$

The amplitude of a vector A = $\sqrt{x_n}$

The direction of a vector ($\frac{x_1}{A}, \frac{x_2}{A}, .., \frac{x_n}{A}$)

Dot product of 2 vectors $\bar{a} . \bar{b} = a_1 b_1 + a_2 b_2 + .. + a_n b_n$

If you switch to matrices instead of vectors: $A^T B$: the multiplication of the transposed matrix A and B

Augmented vector, is a vector onto which the constant 1 is added $= (1, x_1, x_2, .., x_n)$

You can use the augmented vector to write an equation $w_0 + x_1 w_1 + x_2 w_2 + .. + x_n w_n$ as the dot product $\bar{w} . \bar{x}$ = (or matrix multiplication $W^T X$)

Throughout this handout the weight vector $\bar{w}$ or weight matrix W is used; only when discussing SVM/SMO the b is used instead of w_0

Chapter 21. Appendix Haralick metrics

The following GLCM (Gray Level co-occurrence matrix) functions characterize the texture of an image by calculating how often pairs of pixel with specific values and in a specified spatial relationship occur in an image, creating a matrix, and then extracting statistical measures from it. This idea comes from the concept of moments in physics. So, you do not analyse the picture directly but its transformation into gradients.

Major metrics are contrast, correlation, homegenity and entropy (energy). There can also be used in information reduction technique.

Developed by Haralick in the 1970's and originally used in breast cancer detection from Röntgen pictures.

Feature	Equation	Ref.
Autocorrelation	$\sum_{i=1}^{N}\sum_{j=1}^{N}(i\cdot j)p(i,j)$	17
Cluster Prominence	$\sum_{i=1}^{N}\sum_{j=1}^{N}(i+j-2\mu)^3 p(i,j)$	1
Cluster shade	$\sum_{i=1}^{N}\sum_{j=1}^{N}(i+j-2\mu)^4 p(i,j)$	1
Contrast	$\sum_{i=1}^{N}\sum_{j=1}^{N}(i-j)^2 p(i,j)$	1
Correlation	$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{(i\cdot j)p(i,j)-\mu_x\mu_y}{\sigma_x\sigma_y}$	1
Difference entropy	$-\sum_{k=0}^{N-1} p_{x-y}(k)\log p_{x-y}(k)$	1
Difference variance	$\sum_{k=0}^{N-1}(k-\mu_{x-y})^2 p_{x-y}(k)$	1
Dissimilarity	$\sum_{i=1}^{N}\sum_{j=1}^{N}\lvert i-j\rvert\cdot p(i,j)$	17
Energy	$\sum_{i=1}^{N}\sum_{j=1}^{N}p(i,j)^2$	1
Entropy	$-\sum_{i=1}^{N}\sum_{j=1}^{N}p(i,j)\log p(i,j)$	1
Homogeneity	$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{p(i,j)}{1+(i-j)^2}$	17
Information measure of correlation 1	$\frac{HXY-HXY1}{\max(HX,HY)}$	1
Information measure of correlation 2	$\sqrt{1-\exp[-2(HXY2-HXY)]}$	1
Inverse difference	$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{p(i,j)}{1+\lvert i-j\rvert}$	16
Maximum probability	$\max_{i,j}p(i,j)$	17
Sum average, μ_{x+y}	$\sum_{k=2}^{2N} kp_{x+y}(k)$	1
Sum entropy	$-\sum_{k=2}^{2N} p_{x+y}(k)\log p_{x+y}(k)$	1
Sum of squares	$\sum_{i=1}^{N}\sum_{j=1}^{N}(i-\mu)^2 p(i,j)$	1
Sum variance	$\sum_{k=2}^{2N}(k-\mu_{x+y})^2 p_{x+y}(k)$	1