Machine Learning FS2019

https://github.com/taneher/HSLU/tree/master/FS19/ML

Alex Neher, Pascal Baumann

August 5, 2019

Contents

1 Introduction

There are two popular definitions of Machine Learning:

> "Field of study that gives computers the ability to earn without being explicitly programmed" (Arthur Samuel, IBM, 1959)

> "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E" (Tom Mitchell, 1998)

So summarizing these two quotes, it can be said, that machine learning is defined as **the process in which machines learn something (mostly) on their own**.

1.1 Disciplines

There are different disciplines in machine learning:

Supervised Learning: The algorithm is given **labelled training data** and learns to **predict** the **labels** of yet unseen examples.

Unsupervised Learning: The algorithm is given **unlabelled data** and **creates labels by itself** based on the structure of the given data

Semi-Supervised Learning: A **mixture** of supervised and unsupervised learning. This approach is usually chosen if there is only **very little labelled test data**

Reinforcement Learning: No data is available, but the algorithm is **being rewarded**. The algorithm searches the ideal behaviour that maximises its reward (Not subject of this lecture)

These classifications can be subdivided even more:

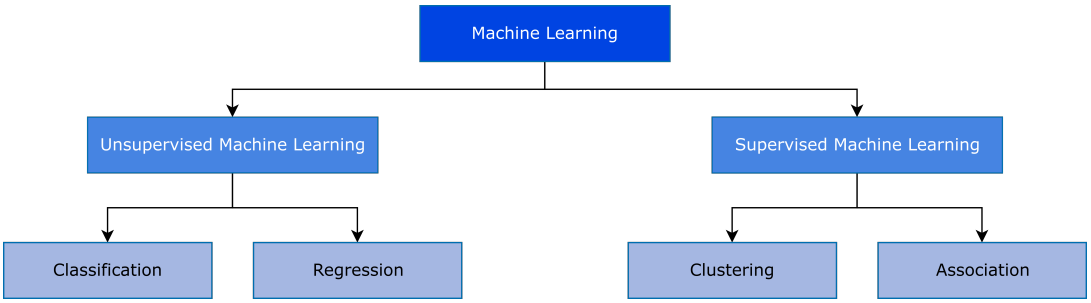


Figure 1.1: Distinction between supervised and unsupervised learning

The main difference between **classification** and **regression** is that when using classification, the result is **categorical**, whereas regression returns **numerical** results.

Clustering is similar to classification. However, while classification algorithms sort the given data into given groups, clustering algorithms determine these groups **by themselves**. This means, you can give a clustering algorithm a seemingly random dataset and the algorithm finds some kind of structure in it.

2 Data Quality

Data is categorized into **numerical** and **categorical** data.

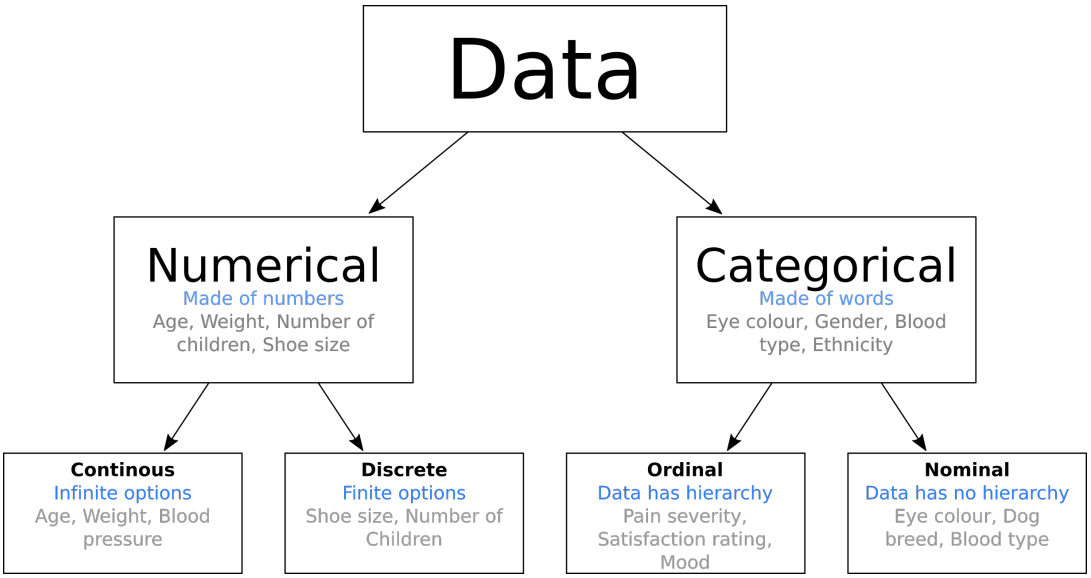


Figure 2.1: Classification of Data

Before any machine learning can take place, the quality of the given data has to be assessed and in some cases improved. Because every prediction made by machine learning algorithms is shit if the data quality is shit.

There are many reasons why the data quality could be poor:

- Ill-designed, inadequate or inconsistent data formats
- Programming errors or technical issues (e.g. sensor outage)
- Data decay (e.g. outdated e-mail addresses)
- Poorly designed data entry forms (e.g. data fields without verification)
- Human errors in data export or data pre-processing
- Deliberate errors and false information (e.g. due to privacy concerns everybody is called Hans Muster and lives at Musterstrasse 123)

2.1 Data Quality Assessment

Before even starting to assess the data-quality, it is seldom a bad idea to **clean** the data first.

1. Identify and remove duplicates
2. Replace null-values (do not delete them because that might falsify the mean and median of the data)
3. Make data formats more machine-friendly (so-called *data-wrangling* e.g. store the gender as boolean)

If you change anything from the original data set, you should always

- Document all the changes
- Use a VCS[1] (e.g. git)
- Let the data provider know that his data quality is shit (maybe they'll improve in the future)
- Investigate the origins of the poor data quality

2.2 Approaches to Data Quality Assessment

Identify data sources and their trustworthiness

Interpret statical key figures: See following sections

Visualize selected portions of the data: e.g. with Pair Plots (see figure 2.2)

Manually check data ranges Negative Salaries, People more than 200 years old...

Validate plausibility of attribute correlation: e.g. are mileage and number of seats in a core correlated? Can one of the columns be removed for redundancy?

Measure data redundancy: Can certain columns be removed due to not adding any real value to the data

Check for anomalies in syntax and semantics: Outliers can really distort a dataset and render the whole algorithm useless. Can be prevented by e.g. normalization of the data or removal of the outlier

Replace NULL Values and remove duplicate values

There are different ways to cope with NULL variables, but they have to be addressed, as most machine learning algorithms do not play well with them.

- Delete all rows with NULL values
  Might be the easiest way if you have loads of data
- Fill in the missing values manually (e.g. from other sources)
  Might be the hardest way if you have loads of data
- Fill in a global constant like N/A, UNKNOWN
- Use a measure for central tendency
  e.g. take the mean if your data is symmetric or take the median if its skewed
- Use a measure for central tendency per class
  e.g. take different values for healthy and sick people
- Use e.g Regression to 'guess' the missing values

[1] Version Control System

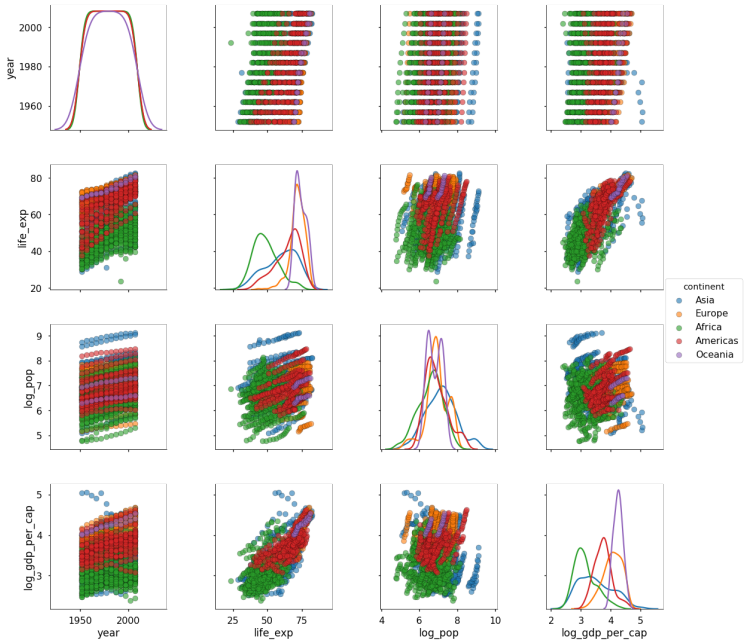


Figure 2.2: Visualisation of Data with Pair Plots

2.3 Statistical Key Figures

These figures can give you a rough overview about the whereabouts of your data-magnitude.

2.3.1 Central Tendency

Mean

This is the average in a set of numeric data. You add all data and divide it by the number of data points

$$\mu_x = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1}$$

Mode

This is the value that occurs the most in a given set of data

Median

This is the middle most value of a sorted set of data. In contrast to the Mean, the Median can give information concerning the distribution of the data.

Given a dataset of $1, 2, 3, 4, 5$, the median and mean are both 3. However, if we have $1, 2, 3, 1000, 10000$, the mean is 2201.2 whereas the median is still 3

2.3.2 Skewness

All of these values can give information concerning the data's **skewness**

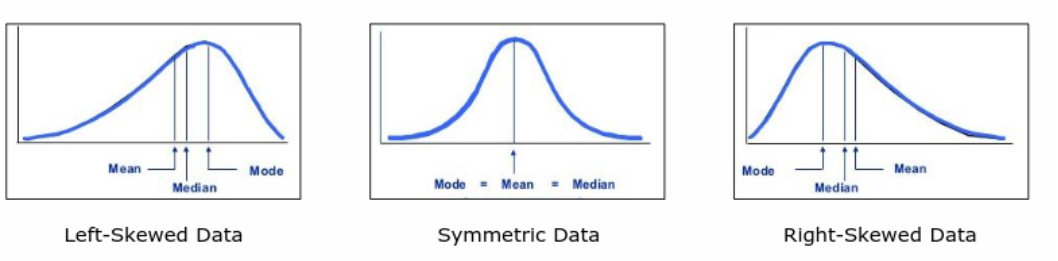


Figure 2.3: Skewness of data

$Mean - Mode > 0 \rightarrow$ Negative skewness / Left-skewed data
$Mean - Mode = 0 \rightarrow$ Symmetric Data
$Mean - Mode > 0 \rightarrow$ Positive skewness / Right-skewed data

2.3.3 Quartile & Interquartile Range (IQR)

The three quartiles divide your data into four equal-sized, consecutive subsets.

To calculate $Q1$, take the median of your data and then again the median of the left half of the data.

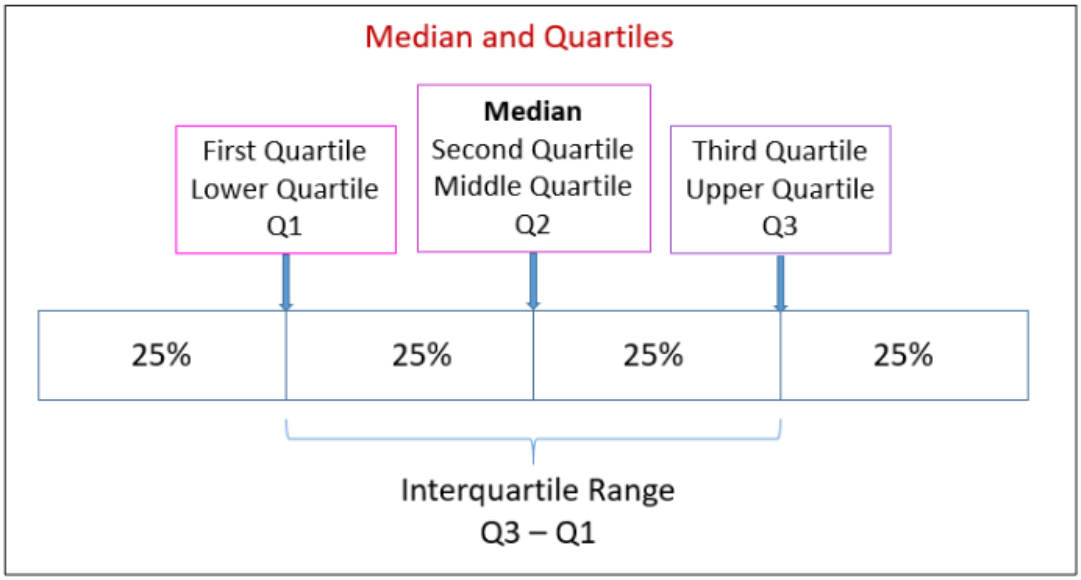


Figure 2.4: Quartiles of a dataset

2.3.4 Five Number Summary

With this method, you can get a pretty good overview of your data. The **Five Number Summary** of a dataset consists of:

- Median Q2
- Quartiles Q1 and Q2
- Smallest individual Value

- Largest individual Value

```
import numpy as np
import panas as pd

s = pd.Series(np.random.rand(100))
s.describe()
```

Listing 2.1: Five Number Summary in Python

```
mean      0.524559
std       0.285565
min       0.003933
25%       0.298367
50%       0.530632
75%       0.765907
max       0.993293
dtype: float64
```

Listing 2.2: Output

2.3.5 Boxplot

This plot is a **visual representation of the five number summary** and can also give information on potential outliers.

Values $1.5 \cdot IQR$ above the 3rd or below the 1st Quartile can be considered outliers and are displayed with small circles.

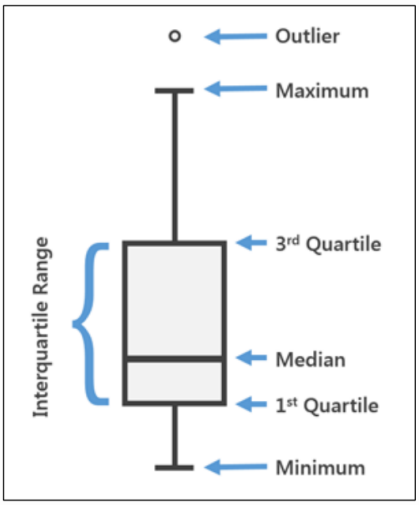


Figure 2.5: Boxplot

2.3.6 Variance

The variance shows **how much the values are spread on average**. This is measured by squaring the sum of all deviations from the mean

$$Var(x) = \frac{1}{1-n} \sum_{i=1}^{n} (x_i - \mu_x)^2 \qquad (2)$$

The standard deviation σ is the square root of the variance $\sqrt{Var(x)}$

2.3.7 Covariance

The covariance is used to determine whether two variables are **connected** to each other.

If both variables are on the same side of the mean, the variance is **positive**, the variables are probably connected. Meaning if the value of one variable is rising, the other one will most likely rise as well.

If one is above and one is below the mean, the variance is **negative**, the variables are most likely **inversely connected** to each other. Meaning if the value of one variable is rising, the other is most likely falling.

If the variables are **independent** of each other, the covariance is zero, as they both cancel each other out.

$$Cov(x, y) = \frac{1}{1-n} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) \tag{3}$$

The **covariance matrix** shows the covariance from all X with all Y. As $Cov(x, x) = Var(x)$, the covariance matrix has the variance of X in its diagonal

2.3.8 Pearson Correlation

Both the covariance and the variance are connected to the scale of the dataset, so the covariance of $X = [1, 2, 3, 4, 5], Y = [6, 7, 8, 9, 10]$ is 2.5, whereas the covariance of $X = [1000, 2000, 3000, 4000, 5000], Y = [6000, 7000, 8000, 9000, 10000]$ is 2'500'000'000. However, the Pearson Correlation is 1 in both examples.

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sigma_x \sigma_y} = \frac{\frac{1}{1-n} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\frac{1}{1-n} \sum_{i=1}^{n} (x_i - \mu_x)^2} \cdot \sqrt{\frac{1}{1-n} \sum_{i=1}^{n} (y_i - \mu_y)^2}} \tag{4}$$

The Pearson Correlation is always between 1 and -1.
1 means the data is perfectly correlated, whereas -1 means that the data is perfectly uncorrelated

2.4 Normalization

It is immensely important that all data is normalized before we run a machine learning algorithm over it. Considering the data in figure 3.2, 'Mileage' and 'Price' are in a completely different scale. If the mileage of the first shown car goes up 500 miles, it's not really a big deal. However, a price increase by 500 would double the car's price.

Such differently scaled data can (and will) falsify the result of every machine learning algorithm you could find. Therefore, **normalization is really important**.

There are two popular normalization approaches: The Min-Max and the Z-Score normalization.

Min-Max normalization
All data is condensed to a value between 0 and 1. The smallest value becomes 0 and the largest one becomes 1.

$$x \to \frac{x - min_x}{max_x - min_x} \tag{5}$$

Z-Score Normalization
The dataset is transformed in such a way, that the mean becomes 0 (so-called *mean-centring*) and the standard deviation is 1

$$x \to \frac{x - \mu_x}{\sigma_x} \tag{6}$$

3 Geometry of Data

3.1 Feature Engineering

Sometimes, data has to be modified to be better accessible/processable for machine learning algorithms. These algorithms can work the best with simple numbers, so that's the data we should be striving for:

Free	Date	Time
283	2015-09-27 00:00:00	06:26:46
282	2015-09-11 00:00:00	05:18:55
280	2015-09-20 00:00:00	21:14:49
283	2015-09-25 00:00:00	01:22:47
0	2015-10-15 00:00:00	08:12:35
0	2015-10-27 00:00:00	10:02:28
281	2015-09-13 00:00:00	12:20:54
168	2015-10-14 00:00:00	08:07:35
283	2015-09-25 00:00:00	05:42:47
283	2015-09-18 00:00:00	22:57:50
279	2015-09-10 00:00:00	20:26:55
279	2015-10-04 00:00:00	18:37:40
84	2015-09-17 00:00:00	17:17:51
86	2015-09-11 00:00:00	08:28:55
3	2015-10-26 00:00:00	13:51:28
281	2015-09-30 00:00:00	00:44:44
252	2015-10-15 00:00:00	07:19:35
280	2015-09-15 00:00:00	00:41:52
282	2015-09-09 00:00:00	06:05:56
0	2015-10-29 00:00:00	12:16:27

Free	Hour	Minute	Year	Month	Day
283	6	26	2015	9	27
282	5	18	2015	9	11
280	21	14	2015	9	20
283	1	22	2015	9	25
0	8	12	2015	10	15
0	10	2	2015	10	27
281	12	20	2015	9	13
168	8	7	2015	10	14
283	5	42	2015	9	25
283	22	57	2015	9	18
279	20	26	2015	9	10
279	18	37	2015	10	4
84	17	17	2015	9	17
86	8	28	2015	9	11
3	13	51	2015	10	26
281	0	44	2015	9	30
252	7	19	2015	10	15
280	0	41	2015	9	15
282	6	5	2015	9	9
0	12	16	2015	10	29

Figure 3.1: Turn 'complicated' data into easier data for better results

3.2 Vector Space Model

As described before, machine learning algorithms work best with **numeric** data. However, the real world isn't that easy and mostly throws categorical data at you. Therefore, you have to convert categorical data to numerical data.

This transformed data can also be visualized in a coordinate system, and we can do math with it.

Name	Price	Mileage	Color
ALFA ROMEO 145 1.4 TS 16V L	500	187000	schwarz
ALFA ROMEO 145 1.8 TS 16V L	2600	182510	rot
ALFA ROMEO 145 1.9 JTD	3500	116000	grau
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	4900	181000	rot
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	800	121000	rot
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	3200	156000	schwarz
ALFA ROMEO 146 2.0 Ti 16V	770	158000	grau
ALFA ROMEO 146 2.0 Ti 16V	1200	119000	rot
ALFA ROMEO 146 2.0 Ti 16V	4900	166000	schwarz
ALFA ROMEO 146 2.0 Ti 16V	4900	102000	silber
ALFA ROMEO 146 2.0 Ti 16V Kit Sport	5800	165000	schwarz
ALFA ROMEO 147 1.6 16V Blackline	11500	46230	braun

Name	Price	Mileage	braun	gelb	grau	grün	rot	schwarz	silber	weiss
ALFA ROMEO 145 1.4 TS 16V L	500	187000	0	0	0	0	0	1	0	0
ALFA ROMEO 145 1.8 TS 16V L	2600	182510	0	0	0	0	1	0	0	0
ALFA ROMEO 145 1.9 JTD	3500	116000	0	0	1	0	0	0	0	0
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	4900	181000	0	0	0	0	1	0	0	0
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	800	121000	0	0	0	0	1	0	0	0
ALFA ROMEO 145 2.0 TS 16V Quadrifog.	3200	156000	0	0	0	0	0	1	0	0
ALFA ROMEO 146 2.0 Ti 16V	770	158000	0	0	1	0	0	0	0	0
ALFA ROMEO 146 2.0 Ti 16V	1200	119000	0	0	0	0	1	0	0	0
ALFA ROMEO 146 2.0 Ti 16V	4900	166000	0	0	0	0	0	1	0	0
ALFA ROMEO 146 2.0 Ti 16V	4900	102000	0	0	0	0	0	0	1	0
ALFA ROMEO 146 2.0 Ti 16V Kit Sport	5800	165000	0	0	0	0	0	1	0	0
ALFA ROMEO 147 1.6 16V Blackline	11500	46230	1	0	0	0	0	0	0	0

Figure 3.2: Turn categorical data into numerical data with the vector space model

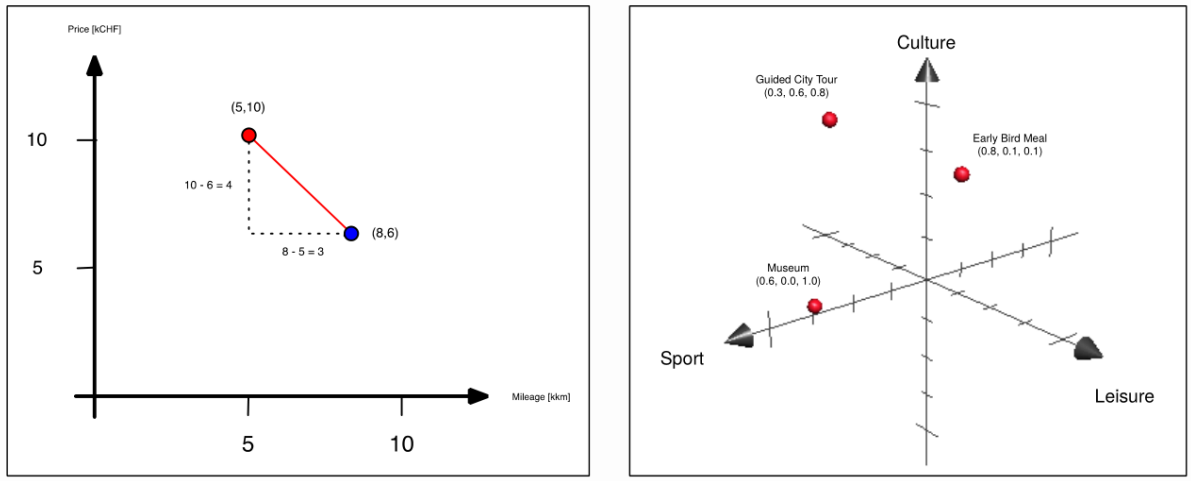


Figure 3.3: Transformed into vector space, data points can be interpreted as geometric points

3.3 Similarity of Data

The maths we want to do is not even overly complicated: We just want to measure the distance between different points. Because **the smaller the distance between two points, the more similar they are**.

3.3.1 Euclidean Distance

The distance between two points is most easily calculated using the **euclidean distance**:

$$dist(X, Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{7}$$

So the distance between the points (5/10) and (8/6) can be calculated as

$$\sqrt{(5-8)^2 + (10-6)^2} \tag{8}$$

$$\sqrt{-3^2 + 4^2} \tag{9}$$

$$\sqrt{9 + 16} \tag{10}$$

$$\sqrt{25} = 5 \tag{11}$$

3.3.2 Cosine Similarity

If you want to to compare two points that appear to be on a line (Pearson Correlation close to 1), but the euclidean distance is high, then the cosine similarity is probably pretty low.

The cosine similarity looks at the **angle** between point A and point B. However, it does also take the euclidean distance into consideration.

The cosine similarity is essentially just the scalar product of the two points.

$$sim(X,Y) = \frac{\sum_{i=i}^{n} x_i y_i}{\sqrt{\sum_{i=i}^{n} {x_i}^2}\sqrt{\sum_{i=i}^{n} {y_i}^2}} \quad (12)$$

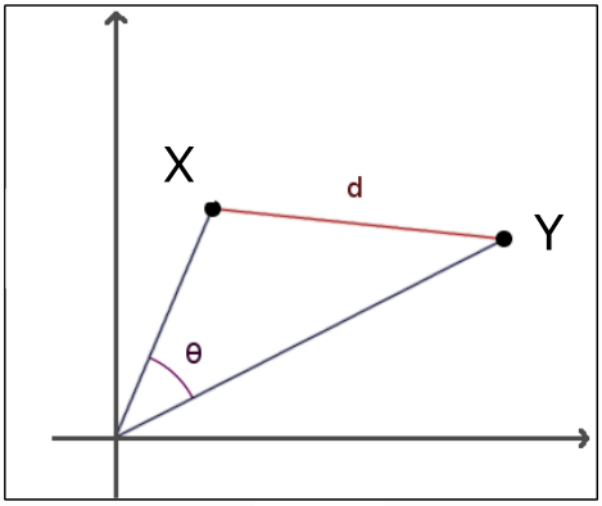


Figure 3.4: Cosine Similarity

$$dist(X,Y) = 1 - sim(X,Y) \quad (13)$$

3.3.3 Levenshtein / Edit Distance for Strings

Count the minimal number of changes necessary to turn one string into another:

- count +1 when deleting a character [d]
- count +1 when adding a character [a]
- count +2 when changing a character [c]

1. Word	2. Word	Levenshtein Distance
Hello	Yellow	1 [c] + 1 [a] = 3
MacDonald	McDonalds	1 [d] + 1 [a] = 2
banana	ananas	? d+a=2

Figure 3.5: Examples for Levenshtein Distance

4 Supervised Machine Learning

4.1 Regression and classification algorithms

Regression

- Linear Regression
- Polynomial Regression
- k-NN Regression
- Support Vector Regression
- Neural Networks
- Regression Trees

Classification

- Logistic Regression
- Naïve Bayes
- k-NN
- Support Vector Machines
- Neural Networks
- Decision Trees

4.2 Decision Boundaries

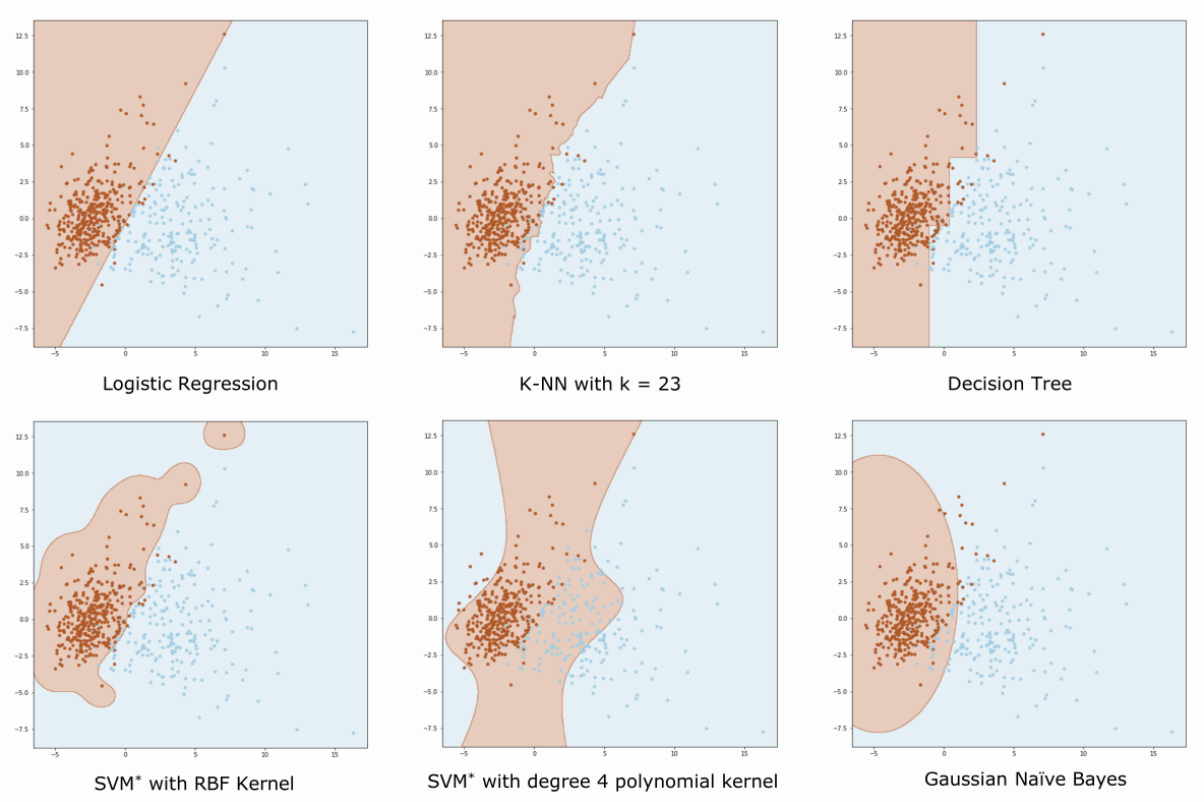


Figure 4.1: Decision Boundaries for different classification approaches

Classifications usually end in something like figure 4.1. The example shows a classification whether a tumour is benign or cancerous. Brown means cancerous and blue means benign. Even though the data points are the same in all pictures, different approaches yield different results.

The goal of a 'good' classification-algorithm is to produce as few false-positives (algorithm says is cancer, but is actually not) and false-negatives (algorithm says its benign but is actually cancerous) as possible.

4.2.1 Kernel-Trick

The data in Fig. 4.1 is still theoretically linearly separable. But in case it is not, you could use the so-called 'kernel-trick', where you simply add a dimension and change your point of view (see Fig. 4.2)

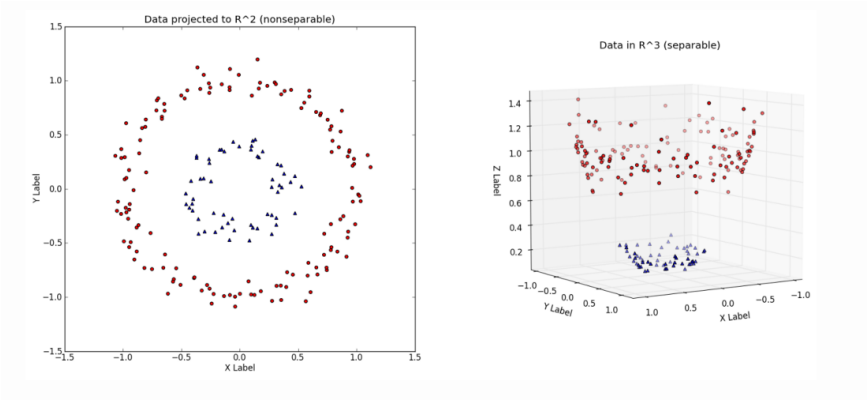
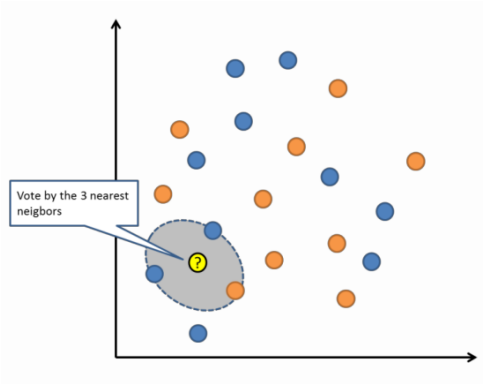


Figure 4.2: Kernel Trick to linearly separate data that is not linearly separable

4.3 k-Nearest-Neighbour

```
from sklearn.neighbors import
    KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print("Test Set Accuracy for k=3" + ":
    (:.2f)".format(acc))
```



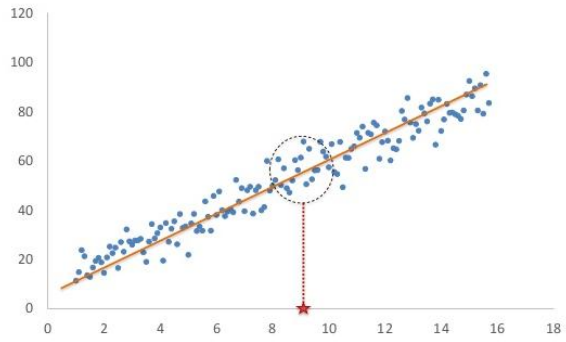


Figure 4.3: Regression with k-NN

The k-nearest-Neighbour algorithm assigns the label of its nearest data point to the sample data point. If `n_neighbours` is > 1, it assigns the label of the most neighbours (via majority voting).

Even though the K-NN algorithm is pretty slow compared to other algorithms, it is often a good choice if you only have a limited amount of data, because other, more performant algorithms usually require a lot more data to get decent predictions.

Usually, k-NN treats all neighbours as equals. However, you could assign a weight to each data point. This weight depends on the distance d from the sample point. This is especially useful if you want to use k-NN to e.g. form a regression line.

You can use k-NN for regression by simply assigning the mean of all k neighbours as label to the sample data.

4.4 Training- and Test data

If you use the same data to train and test your algorithm, it might occur that the algorithm is 'memorizing' the data and gives you brilliant results. However, if you release it into the wild, where it encounters different data, it will perform really poorly. This is called **overfitting**

To counter overfitting, you usually split your data into **training data** and **test data**. You train the algorithm with the training data and test it with the test data (who would've thought...).

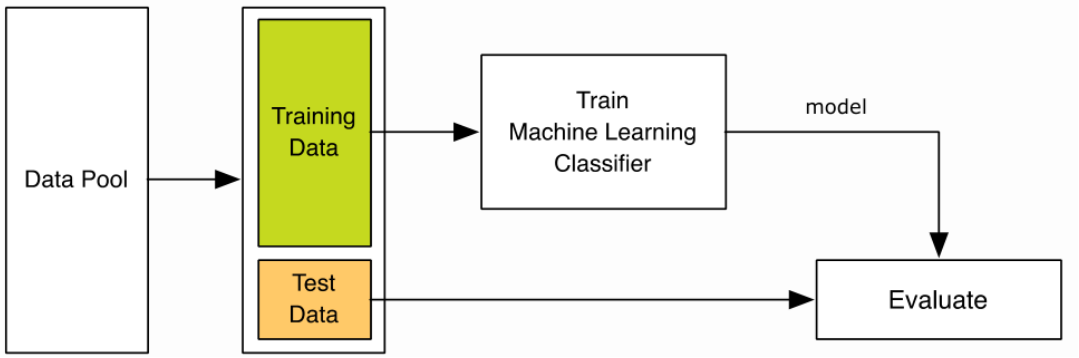


Figure 4.4: Split your data into training- and test data

training means that you tweak the parameters of the algorithm to minimise the cost function.

testing means that you test the performance of the algorithm with those tweaked parameters on *yet unseen data*. If the algorithm has already seen the data, you might run into overfitting problems.

If you need to tweak your hyperparameters a lot (e.g. the 'k' in k-NN), you should probably use a more complex evaluation workflow. Because if you keep tweaking the hyperparameters and then testing them with the same data, you'll end up with the very same overfitting problem that I explained earlier (and will therefore probably get fired and have to live on the street).

Therefore, it is recommended that you add **validation data** to your workflow. You train your model with the training data, validate the results with the validation data, and if the result is satisfactory, you can test it on entirely different test data.

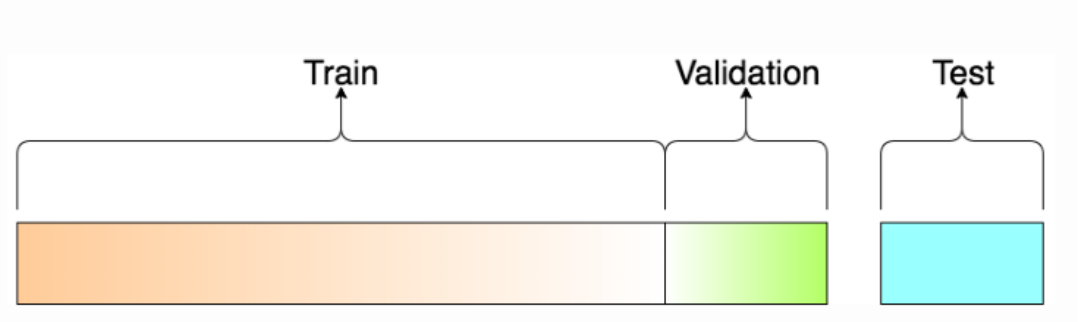


Figure 4.5: Add validation data to the mix

This method requires quite a lot of data. If you do not have the required amount of data, you could for example use **cross validation**. You still split your data into training- and testdata and then use a different 'slice' of your training data to validate the hyperparameter.

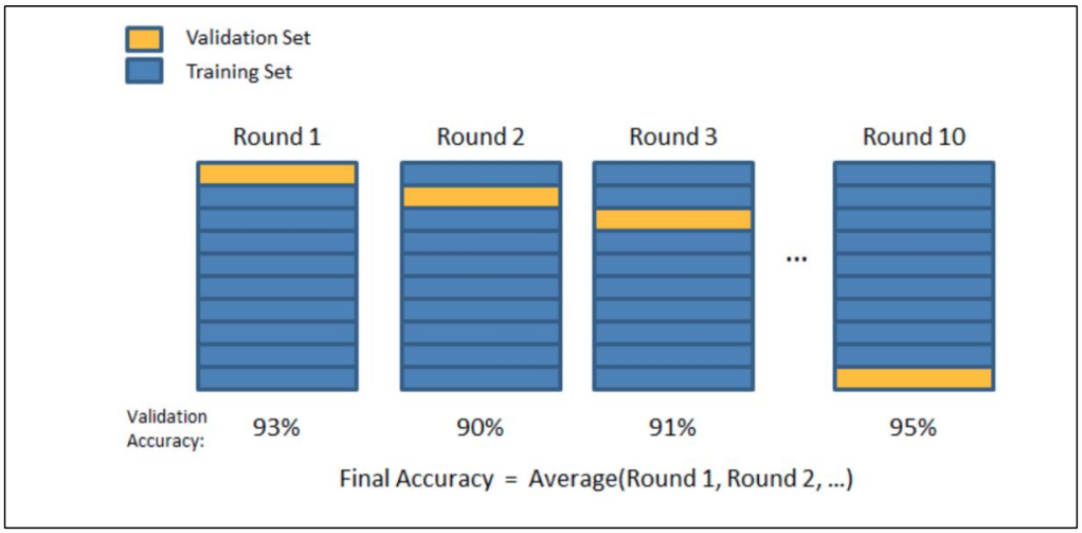


Figure 4.6: 10-fold cross validation

4.5 Measuring the performance of classification

To verify that your tweaked parameters are indeed within the margin that is acceptable, you need to do some quality assurance first.

4.5.1 Confusion Matrix

n=165	Predicted YES	Predicted NO
Actual No	50	10
Actual YES	5	100

True Positive: Predicted Yes, Actual Yes

True Negative: Predicted No, Actual No

False Positive: Predicted Yes, Actual No

False Negative: Predicted No, Actual Yes

The confusion matrix shows, how many true/false positives and true/false negatives the algorithm produced.

With these values, one can calculate the algorithms **Accuracy** and **Error Rate**

4.5.2 Accuracy and Error Rate

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total} \tag{14}$$

$$Error\ Rate = \frac{False\ Positive + False\ Negative}{Total} = 1 - Accuracy \tag{15}$$

In our case the Accuracy would be $\frac{50+100}{50+100+5+10} = \frac{150}{165} = 0.91 = 91\%$ and the error rate therefore 9%

4.5.3 Sensitivity

Accuracy works great on balanced data, but it's not reliable on imbalanced data because Accuracy only checks how many times the classifier was right.

If there were 5000 NO instances and 20 YES instances, a classifier that only returns NO would have an accuracy of over 99%.

The **Sensitivity** (also called 'Recall') counts how many true positives there are.

$$Sensitivity = \frac{True\ Positive}{Actual\ YES} = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{16}$$

In our confusion matrix from earlier, the Sensitivity would be $\frac{100}{100+5} = \frac{100}{105} = 0.95 = 95\%$

4.5.4 Specificity

This is the inverse of the Sensitivity. It counts how many NOs the algorithm correctly predicted.

$$Specificity = \frac{True\ Negative}{Actual\ NO} = \frac{True\ Negative}{True\ Negative + False\ Positive} \tag{17}$$

In our confusion matrix from earlier, the Specificity would be $\frac{50}{50+10} = \frac{50}{60} = 0.83 = 83\%$

4.5.5 Precision

Both of the preceding measures relied on the true negative. However, what would you do if you could not count the True Negatives? You use **Precision**. Precision shows how many times the algorithm is correct if it predicts YES.

$$Precision = \frac{True\ Positive}{Predicted\ YES} = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{18}$$

In our confusion matrix from earlier, the Precision would be $\frac{100}{100+10} = \frac{100}{110} = 0.91 = 91\%$

4.5.6 F1 Score

The F1 score is the harmonic mean between precision and recall/sensitivity

$$F1 = \frac{2 \cdot Precision \cdot Sensitivity}{Precision + Sensitivity} \tag{19}$$

In our confusion matrix from earlier, the Precision would be $\frac{2 \cdot 0.91 \cdot 0.95}{0.91+0.95} = \frac{1.73}{1.86} = 0.93 = 93\%$

Due to the fact that the F1 score does not take True Negatives into account, it tends to be strongly biased towards the worse score.

However, it is still one of the best methods to solve a classification problem with skewed data.

4.6 Measuring the performance of regression

4.6.1 Coefficient of Determination

$$R^2 = 1 - \frac{\frac{1}{m}\sum_{i=1}^{m}(y_i - f_i)^2}{\frac{1}{m}\sum_{i=1}^{m}(y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^{m}(y_i - f_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2} \tag{20}$$

- The sum of errors does not make sense (positive and negative errors cancel out)

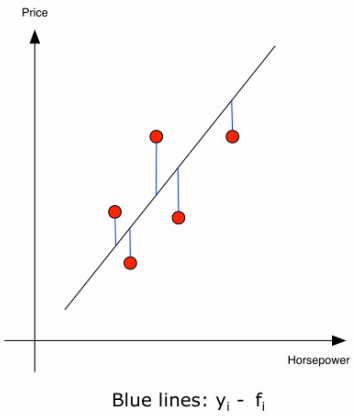
$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f_i)$$

- Mean Absolute Error

$$\frac{1}{m}\sum_{i=1}^{m}|y_i - f_i|$$

- Mean Squared Error

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f_i)^2$$



R^2 is a statistical measure of how well the predictions approximate the real data points. The top is the sum of squared errors (how much does the prediction deviate from the actual value) and the botom is the deviation of the mean.

$R^2 = 1$ is a perfect prediction-line

$R^2 = 0.53$ means that 53% of the predictions are correct and can be explained by the model.

5 Gradient Descent

The problem with Linear Regression is, that it does not scale well. For problems like these we can use Gradient Descent. The gradient (denoted with the Nabla Operator ∇) uses the properties of partial derivatives, which compute the slope with regard to an axis in a point a, b.

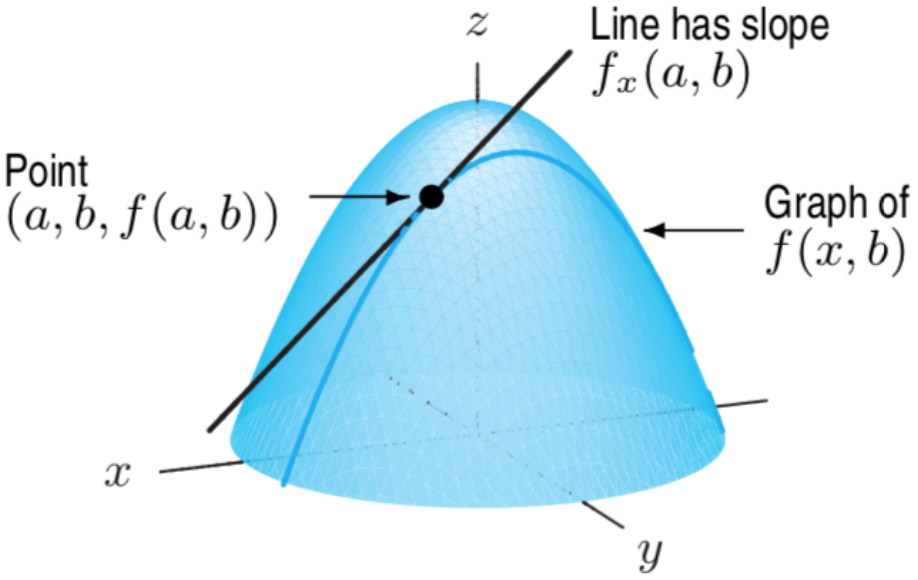


Figure 5.1: Partial derivative with regard to the x-axis in the point a,b (©J. Bürgler, 2019)

The gradient ∇f of the scalar function $f : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto f(x, y) = z$ is a vector, whose components are partial derivatives of f.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\delta f}{\delta x}(x,y) \\ \frac{\delta f}{\delta z}(x,y) \end{bmatrix} = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix} \tag{21}$$

5.0.1 Example of a gradient

Calculate the gradient of the function $f(x, y) = (3x + 2y)^2$ in general and in the point $(1, 2)$.

$$\nabla f(x,y) = \begin{bmatrix} 6(3x+2y) \\ 4(3x+2y) \end{bmatrix}$$
$$\nabla f(1,2) = \begin{bmatrix} 42 \\ 28 \end{bmatrix}$$

5.1 Properties of the gradient

- The Gradient of a function f is a vector consisting of the partial derivatives of f with respect to all of its variables
- It points to the direction of maximal increase
- The negative Gradient points to the direction of maximal decrease
- The Gradient being a null-vector indicates a local extrema
- The Gradient is perpendicular to the contour lines of that function

5.2 Gradient descent

To find a local minima of a function f we follow the direction of the steepest descent, and thus the negative Gradient. As the Gradient is constantly changing, we have to take small steps.

5.2.1 Batch Gradient Descent

Elevating the strengths of vectors we can Gradient Descent along all columns of our data. For this we want to minimise the cost function

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - y^{(i)})^2 \tag{22}$$

where $h(\boldsymbol{\theta}, \mathbf{x}^{(i)})$ is

$$h(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}) = \boldsymbol{x}^{(i)}\boldsymbol{\theta} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_m x_m^{(i)} \tag{23}$$

And calculate the next iteration with the learning rate α (usually $0 \leq \alpha \leq 0.1$)

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla J(\boldsymbol{\theta}_k) \tag{24}$$

It can be shown that this yields

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{1}{n}(h(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}) - y^{(i)}) \cdot \boldsymbol{x}^{(i)} \tag{25}$$

5.3 Stochastic Gradient Descent

In Batch Gradient Descent the update to the parameter vector $\boldsymbol{\theta}$ is done all at once using the whole set of n training samples. If n is in the order of several 10^6, this update process can become slow. One possible alternative is to update $\boldsymbol{\theta}$ for each training sample separately. This is done by randomly shuffling the training examples first and then updating $\boldsymbol{\theta}$ for each training sample.

5.3.1 Instructions

- Choose an initial parameter vector $\boldsymbol{\theta}$ and a learning rate α
- Repeat until an approximate minimum is obtained
  - Randomly shuffle the samples in the training set
  - For each sample $i \in 1, 2, ..., n$ do

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla J_i(\boldsymbol{\theta}_k) = \boldsymbol{\theta}_k - \alpha \frac{1}{n}(h(\boldsymbol{\theta}_k, \boldsymbol{x}^{(i)}) - y^{(i)}) \cdot \boldsymbol{x}^{(i)}$$

    where $h(\boldsymbol{\theta}_k, \boldsymbol{x}^{(i)}) = \boldsymbol{\theta}_k^T \boldsymbol{x}^{(i)}$ and

$$J_i(\boldsymbol{\theta}) = \frac{1}{2n}(h(\boldsymbol{\theta}_k, \boldsymbol{x}^{(i)}) - y^{(i)})^2$$

    is the cost for one data point $(x^{(i)}, y^{(i)})$

The difference to Batch Gradient Descent is that only one piece of data from the dataset is used to calculate the updated parameter and this piece of data is chosen randomly.

5.3.2 Stochastic Gradient Descent - Example

```
def parallel_cost(X,Y,x_data,y_data):
m = X.shape[0]; n = X.shape[1]
tot = np.zeros((m,n))
for i in range(1,len(x_data)):
tot += (X + Y * x_data[i] - y_data[i]) ** 2
return tot/(2 * len(x_data))

alpha = 0.03; epochs = 2000

for epoch in np.arange(0,epochs):
arr = np.arange(X.shape[0])
np.random.shuffle(arr)
for sample in np.arange(0,X.shape[0]):
i = arr[sample]
preds_i = X[i].dot(theta)
error_i = preds_i - y[i]
grad_i = X[i].dot(error_i)/X.shape[0]
theta = theta - alpha * grad_i
theta0_path.append(theta[0])
theta1_path.append(theta[1])
```

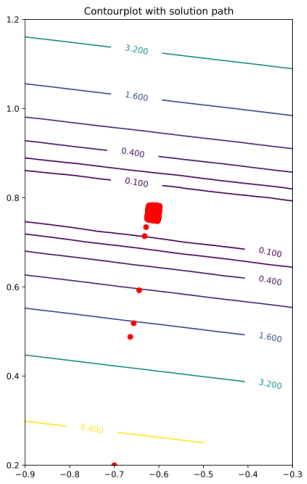


Figure 5.2: Stochastic Gradient Descent

5.4 Polynomial Regression, Feature Scaling and Checking Convergence

If we do not have a linear relationship, but a polynomial one between the features and estimated outcome, our equation is in the general form of

$$h(\boldsymbol{\theta}, \boldsymbol{x}) = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_n x^n \tag{26}$$

We could then use the features $x_1 = x$, $x_2 = x^2$ and $x_3 = x^3$, but if the features are not in the same range, we have a problem. To solve this we must use feature scaling:

1. Make sure the features are on a similar scale and get every feature into the range of $-1 \leq x_i \leq 1$
   (a) Replace x_i with $x_i - \bar{x}_i$ to have features with an approximately zero mean. Does not apply to $x_0 = 1$
   (b) Divide $x_i - \bar{x}_i$ by the range of $x_i = max(x_i) - min(x_i)$ or the standard deviation of x_i
2. Test if Gradient Descent is working correctly by plotting the cost function $J(\boldsymbol{\theta})$ versus the number of iterations
3. Declare convergence if $J(\boldsymbol{\theta})$ decreases by less than a given threshold (for example 10^-3) in one iteration

For sufficiently small α, $J(\boldsymbol{\theta})$ should decrease in every iteration, but a too small α leads to a slow convergence. Start with $\alpha = 0.001$, proceed with $0.003, 0.01, 0.03, 0.1, \ldots$.

6 Linear Regression

Linear regression is a **supervised machine learning algorithm** to predict values based on previous variables.

It can also be used to verify whether two variables X and Y are **dependent** on each other.

Linear Regression produces a **straight line** (who would have thought) and each data point has a certain distance from that line. That distance is called **error** or **residual**.

These errors should cancel each other out, as the regression-line should be placed in the middle of all the points.

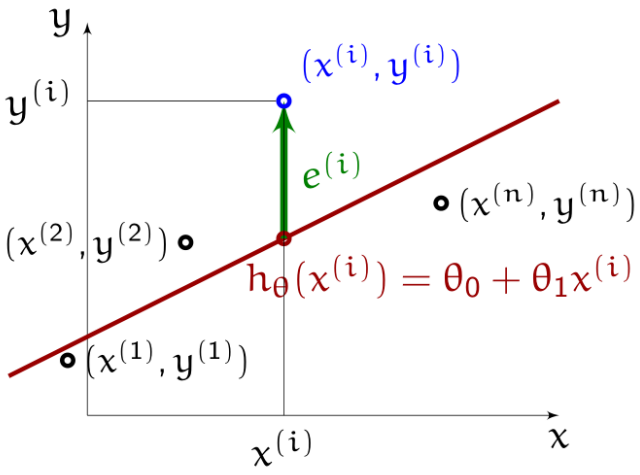


Figure 6.1: Linear Regression

The equation for a 'normal' line is $g(x) = m \cdot x + q$, where m symbolizes the slope of the line and q tells you where the line crosses the y-axis.

The equation for the regression line is very much like the 'normal' line-equation, but we use Greek symbols, because why not...

$$h_\theta(x) = \theta_0 + \theta_1 \cdot x \tag{27}$$

As mentioned before, deviation from that line are called errors. Therefore, the error of a given point i can be calculated from the difference of the actual y_i from the $h_\theta(x)$ (the y on the line)

$$e_i = y_i - (\theta_0 + \theta_1 \cdot x_i) \tag{28}$$

The goal of linear regression is to find the **optimal line**, ergo the line that **produces the smallest errors**.

However, as mentioned earlier, the errors usually cancel each other out, as sometimes the line lies above the data point (negative error) and sometimes it lies below it (positive error). To solve that problem, the parameter we will use to determine the quality of the produced line will be **the sum of squared error**. By squaring the errors, we can't have negative errors (because squared numbers cannot be negative). Therefore, the smaller the sum of squared errors, the better the line.

The sum of squared errors can be calculated as follows:

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (e_i)^2 = \frac{1}{2n} \sum_{i=1}^{n} [y_i - (\theta_0 + \theta_1 \cdot x_i)]^2 \tag{29}$$

To minimise $J(\theta_0, \theta_1)$, the **gradient** of J must be **minimised** or, ideally, zeroed.
This leads to the following formulas for θ_0 and θ_1

$$\theta_1 = \frac{S_{xy}}{S_{xx}} \tag{30}$$

$$\theta_0 = \bar{y} - \theta_1 \cdot \bar{x} \tag{31}$$

Where $\bar{x}$ and $\bar{y}$ are the **mean** of the x and y values respectively

S_{xx} and S_{xy} are called **regression coefficients** and are calculated as follows:

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \qquad (32) \qquad\qquad S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad (33)$$

Example

$X = [4, 6, 8, 10]$
$Y = [2.3, 4.1, 5.7, 6.9]$
$\bar{X} = \frac{1}{4}(4 \cdot 6 \cdot 8 \cdot 10) = \underline{\underline{7}}$
$\bar{Y} = \frac{1}{4}(2.3 \cdot 4.1 \cdot 5.7 \cdot 6.9) = \underline{\underline{7.25}}$
$S_{xx} = (4-7)^2 + (6-7)^2 + (8-7)^2 + (10-7)^2 = \underline{\underline{20}}$
$S_{xy} = (4-7)(2.3-7.25) + (6-7)(4.1-7.25) + (8-7)(5.7-7.25) + (10-7)(6.9-7.25) = \underline{\underline{15.4}}$
$\theta_1 = \frac{20}{15.4} = \underline{\underline{0.77}}$
$\theta_0 = 4.75 - (0.77 - 7) = \underline{\underline{-0.64}}$

From this follows that the regression line is:

$$h_\theta(x) = -0.64 + 0.77 \cdot x$$

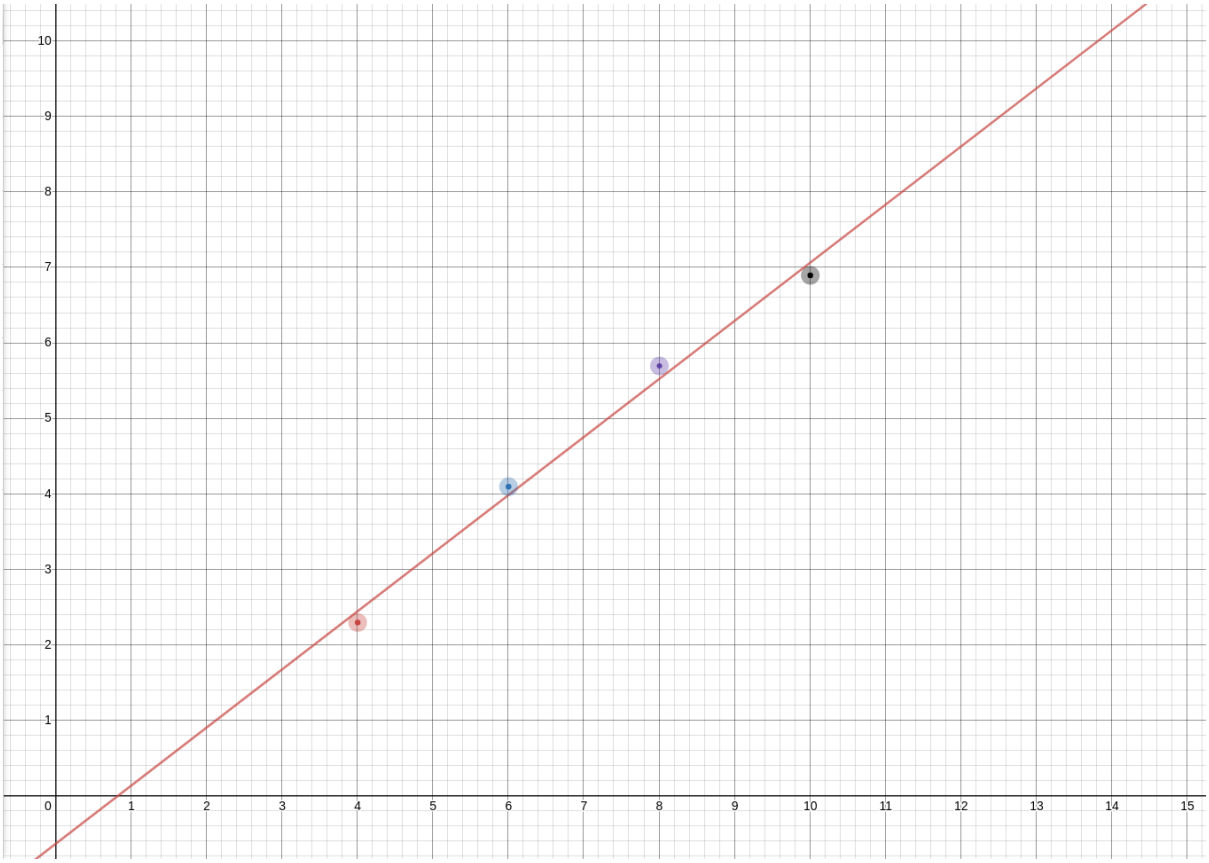


Figure 6.2: Regression line of the example

From figure 6.2, we can now see that for $x_i = 12$, y would most likely be at 8.6.

6.1 Coefficient of Determination

These coefficients were already mentioned in section 4.6.1 as 'way to measure how good a regression is'. But how does it even do that?

As mentioned earlier, the goal of a good regression is to **minimise the sum of squared errors**. However, there are three kinds of sum of squared Errors:

$\bar{y}$**:** Mean of all Y

$\bar{x}$**:** Mean of all X

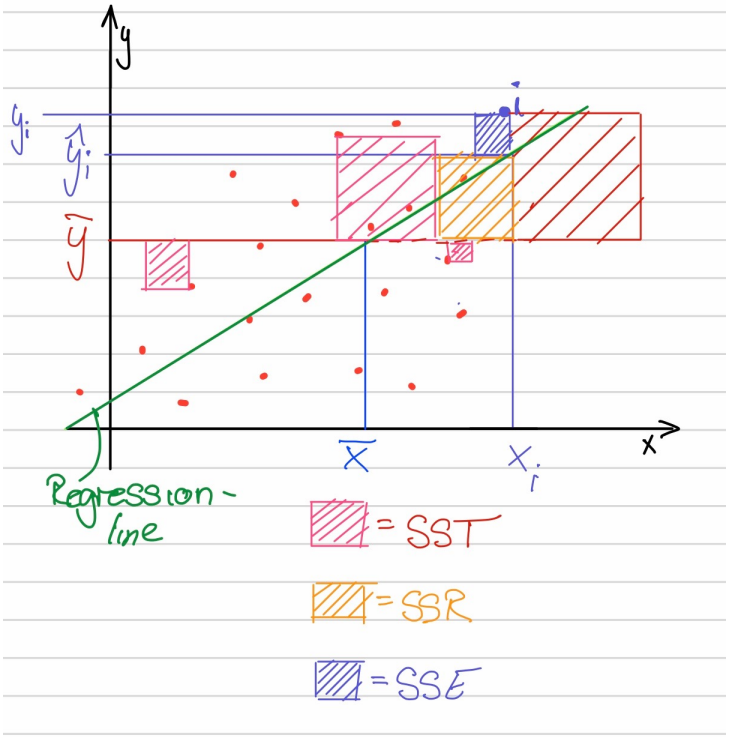
$\hat{y}_i$**:** Expected value of y based on the regression line

y_i**:** Actual value of y

SST : *Total sum of squares*
Sum of SSE and SSR

SSE : *Sum of squared Errors*
The sum of the deviations from the predicted points to the regression line

SSR : *Sum of squares explained by regression*
The sum of all deviations from the mean ($\bar{y}$) to the regression line



As illustrated in the figure above, $SST = SSR + SSE$.

$$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{34}$$

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{35}$$

$$SSR = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 \tag{36}$$

But how does the aforementioned Coefficient of Determination relate to all of this?

R^2 is the **fraction of the total error that can be explained by the regression**. If all errors can be explained by the regression ($R^2 = 1$), then the regression is perfect.

$$R^2 = \frac{SSR}{SST} = \frac{SST - SSE}{SST} = 1 - \frac{SSE}{SST} \tag{37}$$

There's yet another error, the *Mean Squared Error* (MSE). MSE is basically the mean of SSE. It is calculated as

$$MSE = \frac{SSE}{n-2} = \frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{38}$$

6.2 Correlation Analysis

So now that you have the regression parameters θ_0 and θ_1. We can calculate and graph the regression line. However, that regression line is only as good as your parameters. So how do you test how well you calculated the parameters?

Easy, you calculate the **standard deviation of these parameters**.

$$s_{\theta_0} = \sqrt{MSE} \cdot \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \tag{39}$$

$$s_{\theta_1} = \sqrt{MSE} \cdot \sqrt{\frac{1}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \tag{40}$$

Based on these standard deviations, you can calculate a **confidence interval**. This interval is a **degree of uncertainty**. Say we conducted a study, and we publish our results with a **confidence level** of 95%.

This means that if we used the same sampling method to select different samples and computed an interval estimate for each sample, we would expect the true population parameter to fall within the interval estimates 95% of the time.

These intervals can be calculated with the aforementioned standard deviation of θ_0 and θ_1 and some *critical values*. These critical values depend on how high your confidence level and degrees of freedom are.

Kritische Grenzen der t-Verteilung

Kritische Grenzen $t_{1-\frac{\alpha}{2}}(\nu)$ der t-Verteilung mit ν Freiheitsgraden.

ν	1	2	3	4	5	6	7	8	9	10	11	12
$\alpha = 0.05$	12.71	4.30	3.18	2.78	2.57	2.45	2.36	2.31	2.26	2.23	2.20	2.18
$\alpha = 0.01$	63.66	9.92	5.84	4.60	4.03	3.71	3.50	3.36	3.25	3.17	3.11	3.05
ν	13	14	15	16	18	20	25	30	40	60	100	200
$\alpha = 0.05$	2.16	2.14	2.13	2.12	2.10	2.09	2.06	2.04	2.02	2.00	1.98	1.97
$\alpha = 0.01$	3.01	2.98	2.95	2.92	2.88	2.85	2.79	2.75	2.70	2.66	2.63	2.60

Für $\nu \to \infty$ konvergieren die Grenzen gegen die kritischen Grenzen 1.96 bzw. 2.58 der Normalverteilung.

Figure 6.3: Critical Values for Confidence Intervals

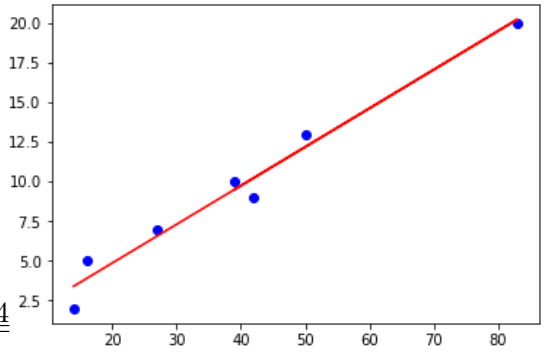
Degrees of freedom tell you how many regression parameters you already had to calculate. Take for example SSE,where both θ_0 and θ_1 are needed. Therefore, SSE has a degree of freedom of $n - 2$, given that you had to calculate 2 parameters.

Confidence Intervals can be calculated with $[\theta_0 - (critical\ value \cdot \theta_0); \theta_0 + (critical\ value \cdot \theta_0)]$

6.3 Linear Regression Example

$X = [14, 16, 27, 42, 83, 50, 39]$

$Y = [2, 5, 7, 9, 20, 13, 10]$



$$\bar{X} = \frac{1}{7}\sum_{i=1}^{7}(x_i) = \frac{14+16+27+42+83+50+39}{7}$$

$$= \frac{271}{7} = \underline{\underline{38.7}}$$

$$\bar{Y} = \frac{1}{7}\sum_{i=1}^{7}(y_i) = \frac{2+5+7+9+28+13+10}{7} = \frac{66}{7} = \underline{\underline{9.4}}$$

$$S_{xy} = \sum_{i=1}^{7}(x_i - \bar{x})(y_i - \bar{y}) = (14-38.7)(2-9.4) + (16-38.7)(5-9.4) + (27-38.7)(7-9.4)$$
$$+ (42-38.7)(9-9.4) + (83-38.7)(20-9.4) + (50-38.7)(13-9.4)$$
$$+ (39-38.7)(10-9.4) = \underline{\underline{819}}$$

$$S_{xx} = \sum_{i=1}^{7}(x_i - \bar{x})^2 = (14-38.7)^2 + (16-38.7)^2 + (27-38.7)^2 + (42-38.7)^2 + (83-38.7)^2$$
$$+ (50-38.7)^2 + (39-38.7)^2 = \underline{\underline{3363.4}}$$

$$\theta_1 = \frac{S_{xy}}{S_{xx}} = \frac{819}{3363.4} = \underline{\underline{0.24}}$$

$$\theta_0 = \bar{y} - \theta_1 \cdot \bar{x} = 9.4 - 0.24 \cdot 38.7 = \underline{\underline{-0.008}}$$

$$SSE = \sum_{i=1}^{7}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{7}(y_i - (\theta_0 + \theta_1 \cdot x))^2 = \sum_{i=1}^{7}(y_i - (-0.008 + 0.24 \cdot x))^2$$

$$= \sum_{i=1}^{7}(y_i - (-0.232 \cdot x))^2 = \underline{\underline{5.87}}$$

$$SSR = \sum_{i}^{7}(\hat{y}_i - \bar{y})^2 = \sum_{i}^{7}((\theta_0 + \theta_1 \cdot x) - 38.7)^2 = \sum_{i}^{7}((-0.008 + 0.24 \cdot x) - 38.7)^2$$

$$= \sum_{i}^{7}((-0.232 \cdot x) - 38.7)^2 = \underline{\underline{199.84}}$$

$$SST = SSE + SSR = 5.87 + 199.84 = \underline{\underline{205.71}}$$

$$R^2 = \frac{SSR}{SST} = \frac{199.84}{205.71} = \underline{\underline{0.97}}$$

$$MSE = \frac{SSE}{n-2} = \frac{5.87}{7-2} = \underline{\underline{1.17}}$$

$$s_{\theta_0} = \sqrt{MSE} \cdot \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} = \sqrt{1.17} \cdot \sqrt{\frac{1}{7} + \frac{38.7^2}{\sum_{i=1}^{n}(x_i - 38.7)^2}} = \underline{\underline{-0.008}}$$

$$s_{\theta_1} = \sqrt{MSE} \cdot \sqrt{\frac{1}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} = \sqrt{1.17} \cdot \sqrt{\frac{1}{\sum_{i=1}^{n}(x_i - 38.7)^2}} = \underline{\underline{0.019}}$$

$$Interval\ \theta_0 = [\theta_0 - 2.57 \cdot \theta_0; \theta_0 + 2.57 \cdot \theta_0] = \underline{\underline{[-2.14; 2.12]}}$$

$$Interval\ \theta_1 [\theta_1 - 2.57 \cdot \theta_1; \theta_1 + 2.57 \cdot \theta_1] = \underline{\underline{[0.20; 0.29]}}$$

The results of the last page can also be achieved with the following python-code:

```
import matplotlib.pyplot as plt
import scipy.stats as st
import seaborn as sns
import pandas as pd
import numpy as np

       %precision 10
       %matplotlib inline

x=np.array([14, 16, 27, 42, 83, 50, 39])
y=np.array([2, 5, 7, 9, 20, 13, 10])

mean_x = np.mean(x)
mean_y = np.mean(y)

Sxx = np.sum((x-mean_x)**2)
Syy = np.sum((y-mean_y)**2)
Sxy = np.sum((x-mean_x)*(y-mean_y))

thet1 = Sxy/Sxx
thet0 = mean_y-thet1*mean_x

# To show the regression line
plt.plot(x,y, 'bo')
plt.plot(x,thet0+thet1*x, 'r')
plt.show()

hat_y = thet0+thet1*x

SSE=np.sum(((y-hat_y))**2)

SSR=np.sum((hat_y-mean_y)**2)

SST = SSE + SSR

R_sq = SSR/SST

MSE = SSE/(len(x)-2)

Sthet0 = np.sqrt(MSE)*np.sqrt((1/len(x))+(mean_x**2)/(np.sum((x-mean_x)**2)))

Sthet1 = (np.sqrt(MSE))*np.sqrt(1/(np.sum((x-mean_x)**2)))

print("theta_1: " + str(thet1))

print("Interval theta0: [" + str(thet0 - (2.57*Sthet0)) + " ; " + str(thet0 +
    (2.57*Sthet0)) + "]")

print("Interval theta1: [" + str(thet1 - (2.57*Sthet1)) + " ; " + str(thet1 +
    (2.57*Sthet1)) + "]")
```

6.4 Multiple Linear Regression

Linear Regression is great and all, but what if you wanted to predict an independent value based on **multiple** other values? In that case we would need **multi-linear regression**. In

Multiple Linear Regression we try to fit a plane instead of a line. This plane is defined by

$$y = h_\theta(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{41}$$

And mapped to the sample with index i

$$y_i = \theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i} + e_i$$

This can be transformed to matrices

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} \\ 1 & x_{1,2} & x_{2,2} \\ & \vdots & \\ 1 & x_{1,n} & x_{2,n} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

This can be solved as follows

$$\boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

In more than three dimensions our plane becomes a hyperplane, and the model is

$$y_i = \theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i} + \cdots + \theta_m x_{m,i} + e_i = \sum_{k=0}^{n} \theta_k x_{k,i} + e_i$$

6.4.1 Example multi-linear regression

Suppose we want to predict the weight of a weightlifter based on the training hours per week and the delivery of protein.

i	**y**	x_1	x_2
1	93	2	1.1
2	106	2	1.9
3	146	4	2
4	140	5	1.5
5	151	6	1.3
6	158	7	2.1
7	130	4	1.8
8	159	5	2.5

i**:** No. of observation

y**:** Weight in kg

x_1**:** Training h/week

x_2**:** Protein intake g/kg/day

We assume that the function to calculate the weight from training-hours and protein intake is as follows:

$$y = h_\theta(x_1, x_2) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$$

The following solution minimises the sum of squared errors in this model (and therefore get a hella good regression).

$$\theta = (X^T X)^{-1} X^T y = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \rightarrow \begin{bmatrix} 55.7 \\ 11.1 \\ 17.5 \end{bmatrix}$$

According to this, the regression plane would be:

$$y = h_\theta(x_1, x_2) = 55.7 + 11.1 \cdot x_1 + 17.5 \cdot x_2$$

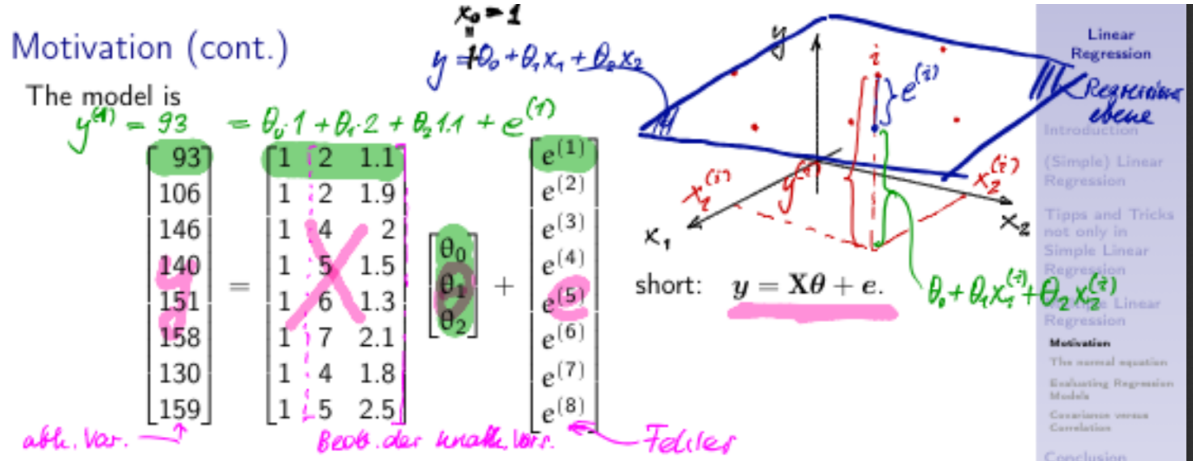


Figure 6.4: Model of the multilinear regression (©J. Bürgler, 2019)

Of course, this can also be done rather easily in Python:

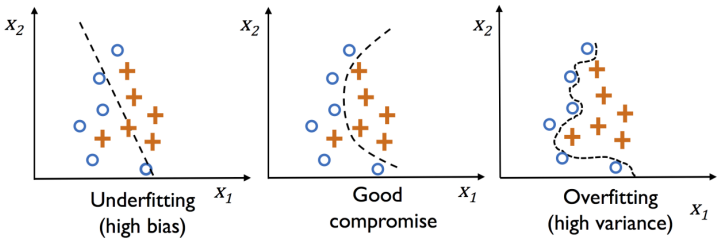
```
import numpy as np

X = np.matrix([[2,1.1],[2,1.9],[4,2.0],[5,1.5],[6,1.3],[7,2.1],[4,1.8],[5,2.5]])

# Anzahl Reihen von X anhand der ersten Zeile (= Anzahl Beobachtungen)
n = X.shape[0]

y = np.matrix([93,106,146,140,151,158,130,159])

# Append a column of '1' infront of the X-matrix (to get the 'X'-matrix from the
    figure)
X_ext = np.c_[np.ones((n,1)),X]

theta = np.linalg.inv(X_ext.T.dot(X_est)).dot(X_ext.T).dot(y)

print(theta)
```

7 Regularisation

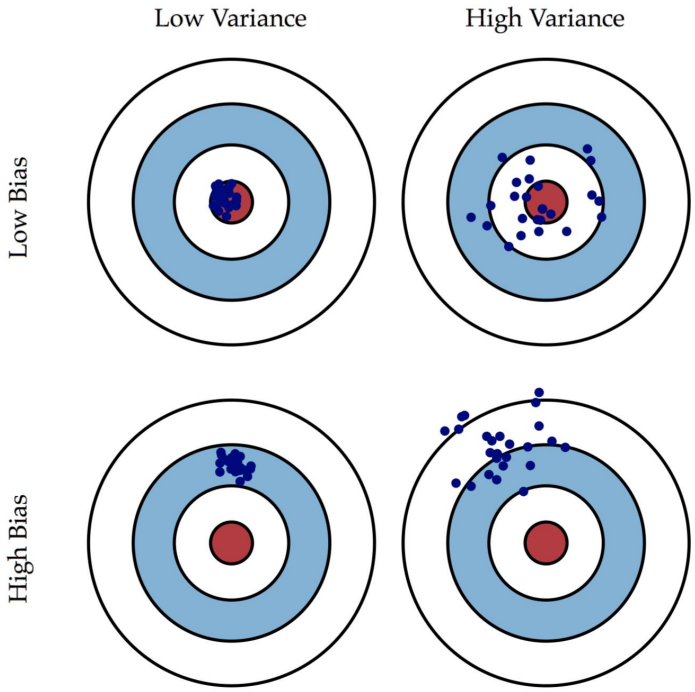
Under- and overfitting are common problems in machine learning. Underfitting occurs when the model is too general and can't aptly represent the data. Overfitting occurs, when the model is too precisely fitted to the training data.

To address overfitting we can try these approaches

1. Reduce the number of features
   - Manually select which features to keep
   - Model selection algorithm
2. Regularisation
   - Keep all features, but reduce magnitude of parameters θ_i
   - Works well with lots of features which each contribute just a bit to predict y



(a) Example of different fittings



(b) Examples of different biases and variances

Figure 7.1: Under- and Overfitting, Biases and Variances

7.1 Ridge Regularisation

$$J(\theta) = \frac{1}{2n}\left[\sum_{i=1}^{n}(h(\theta, x_i) - y_i)^2 + \lambda \sum_{j=1}^{m} \theta_j^2\right] \tag{42}$$

The second sum goes from 1 to the number of features m and it does not contain θ_0. The regularisation hyperparameter λ controls the amount of regularisation. If $\lambda = 0$ there will be no regularization, and if $\lambda = \infty$ there will be underfitting because only θ_0 will be different from zero. In Ridge regularised regression, we choose θ to minimise $J(\theta)$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} J(\theta) \tag{43}$$

large λ	High Bias / Underfit
intermediate λ	Correct
small λ	High Variance / Overfit

7.2 How to choose the right model

7.2.1 Hold-out / Simple Cross Validation

1. Randomly split the Sample S into S_{train} and S_{cv} called the **hold-out cross validation set**. A standard split is 70% S_{train} and 30% S_{cv}.

2. For each degree of the polynomial k, train the model M_k on S_{train} to get the parameter vector $\boldsymbol{\theta}_k$

3. Select $\boldsymbol{\theta}_k$ with the smallest error $\hat{\varepsilon}_{S_{cv}}(\boldsymbol{\theta}_k)$

$$k = \underset{k \in 1,2,...,10}{\operatorname{argmin}} \hat{\varepsilon}_{S_{cv}}(\boldsymbol{\theta}_k)$$

   where $\hat{\varepsilon}_{S_{cv}}(\boldsymbol{\theta}_k)$ is the empirical error, if we use $\boldsymbol{\theta}_k$ as the parameter vector on the cross validation set S_{cv}.

The issue with **hold-out cross validation** is, that it wastes the amount of data used for the cross validation.

7.2.2 k-fold Cross Validation

To reduce the waste of data, we could hold out less data used per run: we use k-fold cross validation **k-fold cross validation**.

1. Randomly split S into l disjoint subsets $S_1, S_2, ..., S_l$ of $\frac{n}{l}$ training examples each

2. Evaluate each model $M_k, k \in 1, 2, ..., 10$ as follows

   - For $j = 1, 2, ..., l$ train M_k on all data, except the subset S_j to get $\boldsymbol{\theta}_{kj}$ and test $\boldsymbol{\theta}_{kj}$ on S_j to get the empirical error $\hat{\varepsilon}_{S_j}(\boldsymbol{\theta}_{kj})$
   - The estimated generalisation of the model M_k is then calculated as the average of the $\hat{\varepsilon}_{S_j}(\boldsymbol{\theta}_{kj})$

$$\frac{1}{l}\sum_{j=1}^{l} \hat{\varepsilon}_{S_j}(\boldsymbol{\theta}_{kj}) \text{ where } \hat{\varepsilon}_{S_j}(\boldsymbol{\theta}_{kj}) = \frac{1}{2n}\sum_{i=1}^{n}(h(\boldsymbol{\theta}_k, \boldsymbol{x}^{(i)}) - y^{(i)})^2$$

3. Pick the model $M - K$ with the lowest estimated generalisation error and retrain that model on the entirety of the training set S.

8 Support Vector Machines

8.1 Scalar Product

The scalar product between two vectors $\vec{a}, \vec{b}$ is defined as

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{44}$$

The following rules hold true for the scalar product

symmetry $\quad \vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$

distributivity $\quad \vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$

multiplication by scalars $\lambda(\vec{a} \cdot \vec{b}) = (\lambda\vec{a}) \cdot \vec{b} = \vec{a} \cdot (\lambda\vec{b})$

Using the scalar product we can define the length of a vector, also called the L^2-Norm

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} = \sqrt{\vec{a} \cdot \vec{a}} \tag{45}$$

Two vectors are orthogonal if and only if their scalar product is zero:

$$\vec{a} \cdot \vec{b} = 0 \Leftrightarrow \vec{a} \perp \vec{b} \tag{46}$$

Example of scalar product

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = 1 \cdot 3 + 2 \cdot 4 = 11$$

8.1.1 The Hessian normal form of a straight line

The straight line g is defined by some point x_0 and the normal vector $\vec{n}$ with length 1 (refer to figure 8.1).

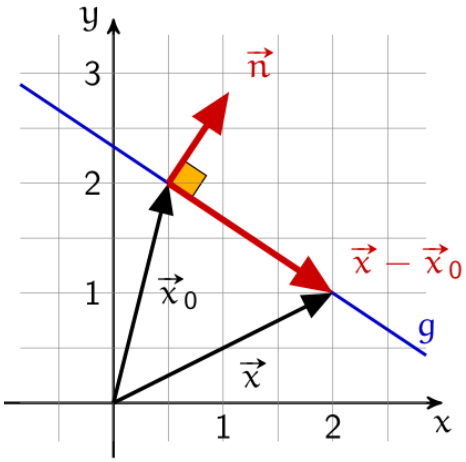


Figure 8.1: Plot of g and the relevant vectors $\vec{x_0}$ and $\vec{n}$

For any point x on the line we have

$$\vec{n} \cdot (\vec{x} - \vec{x_0}) = 0$$

And find

$$\Rightarrow n_x x + n_y y - (n_x x_0 n_y y_0) = n_x x + n_y y + d = 0$$

With $d = -(n_x x_0 + n_y y_0)$ the signed distance $-\vec{n} \cdot \vec{x_0}$ from the origin

The generalisation of the Hessian normal form to a plane in 3D-space is straight forward:

$$\vec{n} \cdot (\vec{x} - \vec{x_0}) = 0 \textbf{ or } \vec{n} \cdot \vec{x} + d = 0 \text{ with } d = -\vec{n} \cdot \vec{x_0} \tag{47}$$

Example of the Hessian Normal Form The line g is taken from figure 8.2

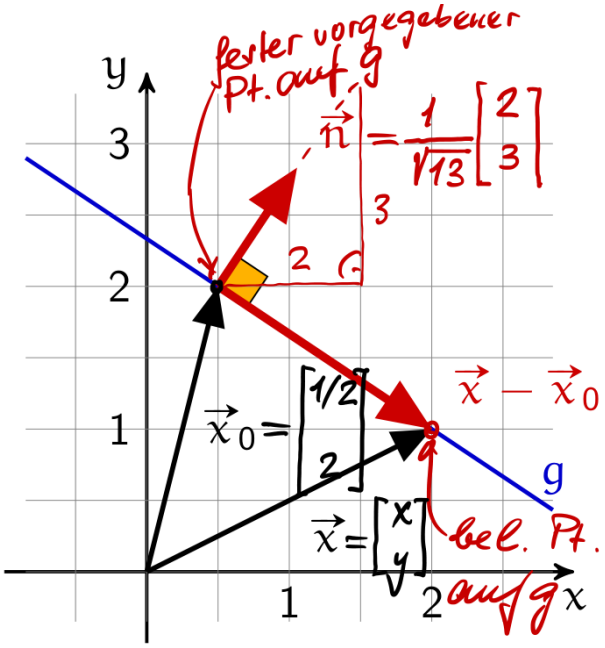


Figure 8.2: Annotated figure 8.1

$$\vec{n} = \frac{1}{\sqrt{13}} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \left(\|\vec{n}\| = 13 \rightarrow \frac{1}{\sqrt{13}} \cdot \vec{n} \rightarrow \|\vec{n}\| = 1 \right)$$

$$\begin{aligned} 0 &= \vec{n} \cdot (\vec{x} - \vec{x_0}) \\ &= \frac{1}{\sqrt{13}} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} 0.5 \\ 2 \end{bmatrix} \right) \\ &= \frac{1}{\sqrt{13}} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} x - 0.5 \\ y - 2 \end{bmatrix} \end{aligned}$$

8.1.2 Motivation for Support Vector Machines

- Classifying data means splitting it up
- There are many ways to split the data, even using only a linear function
- We want the best possible split

The line in the middle of figure 8.3 is called the hyperplane (a line in two dimensions, a plane in three). We know that this is the best possible split, because it is the furthest from both clusters. The data points (samples) closest to the hyperplane are called support vectors.

8.2 Basic ideas and features

- Finds optimal hyperplane for linearly separable patterns
- Extended patterns which are not linearly separable are transformed into another (possibly higher dimensional) feature space
- Not affected by the local minima problem
- No dimensionality problem

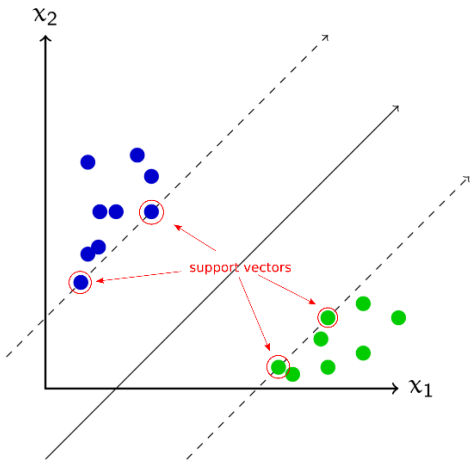


Figure 8.3: Support Vector Machine

- Modular design allows separate implementation of various components
- Support vectors are the data points or samples closest to the decision hyperplane
- Support vectors are the most difficult to classify
- Support vectors directly influence the position of the decision hyperplane and are the critical elements in the training set
- Problem of finding an optimal separating hyperplane can be solved using standard optimisation techniques

8.2.1 Linear Classifier

A linear classifier has the form

$$f(\vec{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m = b + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m = b + \vec{w}^T \vec{x}_b \tag{48}$$

where $\vec{w} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ is the normal vector on the hyperplane and $b = \theta_0$ is the bias.

8.2.2 How to determine optimality

In figure 8.4 $\vec{w}$ is the normal vector on the decision hyperplane, its length depends on the margin which is maximised through the Support Vector Machine. Maximising the margin $\frac{2}{\|\vec{w}\|}$ is equivalent to minimising $\frac{\vec{w}^T\vec{w}}{2}$, therefore we

$$\underset{\vec{w},b}{\text{minimise}} \ \frac{1}{2}\vec{w}^T\vec{w} \tag{49}$$

subject to $y^{(i)}(\vec{w}^T\vec{x}^{(i)} + b) \geq 1$ for $i = 1, 2, ..., n$

8.3 From the hard margin to the soft margin problem

So far we have discussed the hard margin problem (see equation 49), which fails if the problem is not separable. To address this we could introduce a **slack variable** ζ_i for each data point $x^{(i)}$ and solve the **soft margin problem**

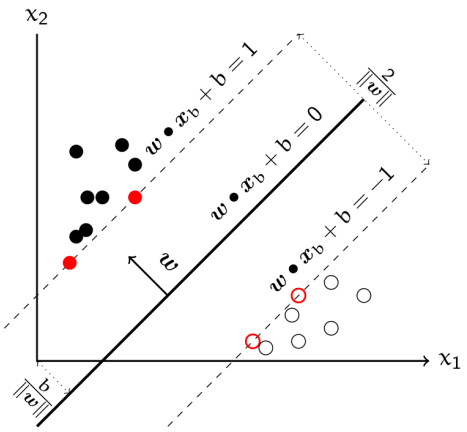


Figure 8.4: Hyperplane with support vectors in red

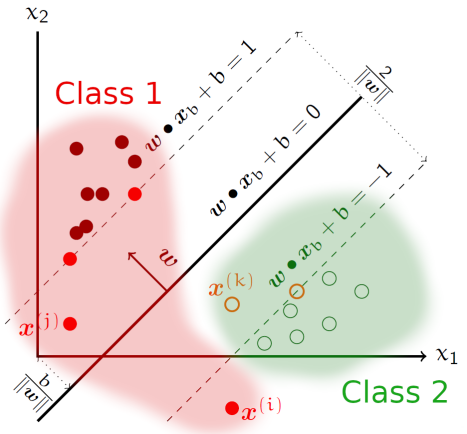


Figure 8.5: Support Vector Machine with soft margins

$$\underset{\vec{w},b}{\text{minimise}}\ \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{n}\zeta_i \text{ subject to } y^{(i)}(\vec{w}^T\vec{x}^{(i)} + b) \geq 1 - \zeta_i \tag{50}$$

with $\zeta_i \geq 0$ for $i = 1, 2, ..., n$. The slack variable ζ_i measures how much the i-th instance $\vec{x}^{(i)}$ is allowed to violate the margin.

8.3.1 The slack variable explained

The margin can be less than 1, by setting $\zeta_i > 0$, but then this gets factored in as a penalty $C\zeta_i$ in the minimisation. Thus the sum $\sum_{i=1}^{n}\zeta_i$ gives an upper bound on the number of training errors. With this soft margin Support Vector Machines minimise training error traded off against better margins. The parameter C is the *regularisation parameter*, which provides a way to control overfitting:

small C $(C \to 0)$	Constraints are easily ignored, we obtain a large margin
large C $(C \to \infty)$	Constraints are hard to ignore, we obtain a narrow margin
$C = \infty)$	All constraints enforced, we have a hard margin

Therefore, we still have a quadratic optimisation problem with a unique minimum, and we only have to deal with the parameter C.

8.4 The Kernel Trick

The kernel transforms a problem to another coordinate system, where the problem may be linearly separable.

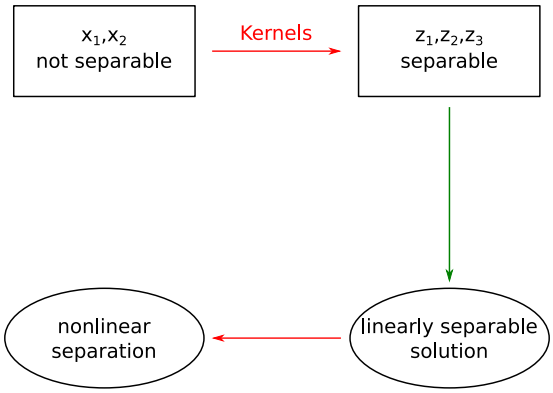


Figure 8.6: Course of action for the kernel method

8.4.1 Kernel Functions Examples

polar coordinates $\Phi : \mathbb{R}^2 \to \mathbb{R}^2, \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} r \\ \theta \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ \arctan(\frac{x_2}{x_2}) \pm \pi \end{bmatrix}$ (refer to figure 8.7)

polynomial mapping $\Phi : \mathbb{R}^2 \to \mathbb{R}^3, (x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2} \cdot x_1 \cdot x_2, x_2^2) = \Phi(x_1, x_2)$
(refer to figure 8.8)

radial $\Phi(x_1, x_2) = e^{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}}$

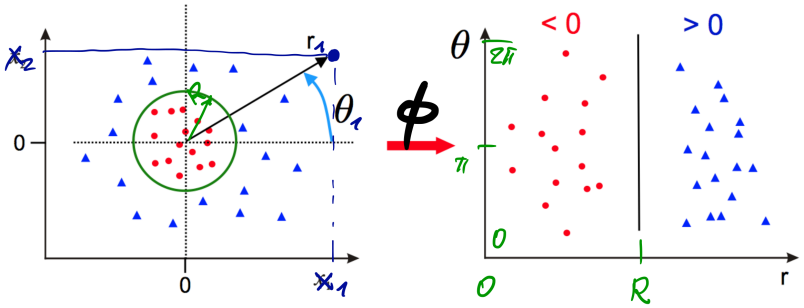


Figure 8.7: Transformation from cartesian to polar coordinate system

Kernel function Φ satisfy:

positive semidefinite $\forall x_1, x_2 (\Phi(x_1, x_2) \geq 0)$

symmetric $\Phi(x_1, x_2) = \Phi(x_2, x_1)$

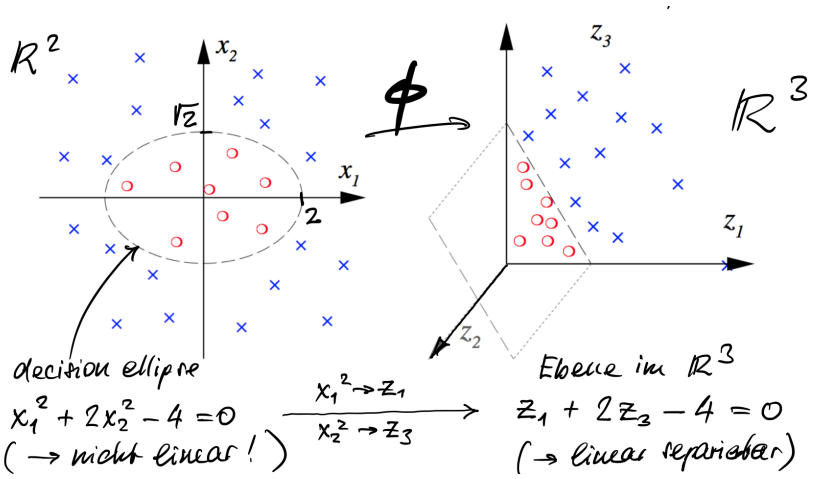


Figure 8.8: Transformation to higher dimension

9 Clustering and Association Rules

9.1 k-Means Algorithm

The k-Means algorithm is an algorithm that automatically identifies cluster centres. One has to specify the amount of clusters one wants before it can start though. The algorithm functions as follows:

1. Input: Number of clusters $k > 0$ and data points $x_1, ..., x_n \in \mathbb{R}^m$
2. Randomly choose k cluster centres $\mu_1, ..., \mu_k \in \mathbb{R}^m$
3. Repeat until convergence
   (a) Assign each data point x_i to its nearest cluster centre μ_j
   $$c_i = \underset{j \in \{1,...,k\}}{\operatorname{argmin}} \|x_i - \mu_j\|^2$$
   (b) Update each cluster centre to the mean of all assigned data points
   $$\mu_j := \frac{\sum_{i:c_i=j} x_i}{|\{i : c_i = j\}|}$$

9.1.1 A note on the Euclidean Distance between two points

$$\begin{aligned} x &= (x_1, x_2) \\ \|x\| &= \sqrt{x_1^2 + x_2^2} \\ \|x\|^2 &= x_1^2 + x_2^2 \\ y &= (y_1, y_2) \\ \|x - y\| &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \end{aligned}$$

9.1.2 Clustering Distortion

The total distortion can be measured by summing up the squared distance between each point and its cluster centre

$$\sum_{i=1}^{n} \|x_i - \mu_{c_i}\|^2 \tag{51}$$

The average distortion per data point is

$$\frac{1}{n}\sum_{i=1}^{n}\|x_i - \mu_{c_i}\|^2 \tag{52}$$

Average distortion allows comparing clusters over different data sets

9.1.3 Convergence and Optimality

Optimal clustering minimises the total distortion, which is a NP-hard problem for $x > 1$. Therefore, k-Means only approximates the optimal solution, but always converges, albeit not necessarily to a global minimum. In practice k-Means is executed several times and the clustering with minimum distortion is chosen.

9.1.4 Choose the Number of Clusters

The choice of clusters balances between low distortion and fewest number of clusters. The elbow method finds a good compromise in general (see figure 9.1).

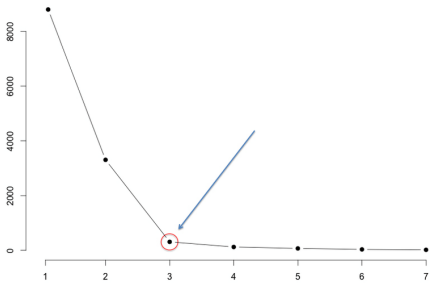


Figure 9.1: The elbow method to find the best number of clusters

9.2 Association

An **association rule** is an implication $X \to Y$, where X and Y are disjoint item sets (refer to 9.2).

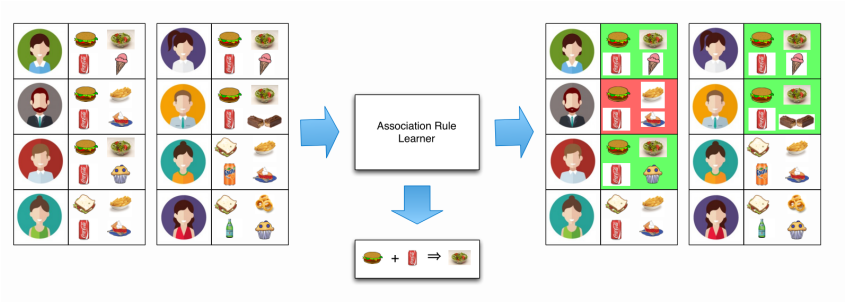


Figure 9.2: A graphical representation of an Association Rule Learner

9.2.1 Support of a Set of Items

Support is the proportion of transactions which contains a specific item set, and thus measures how frequently items are bought together.

$$\text{support}(\{i_1, ..., i_n\}) = \frac{\#\text{purchases of } \{\text{i`1,...,i`n}\}}{\#\text{transactions}}$$

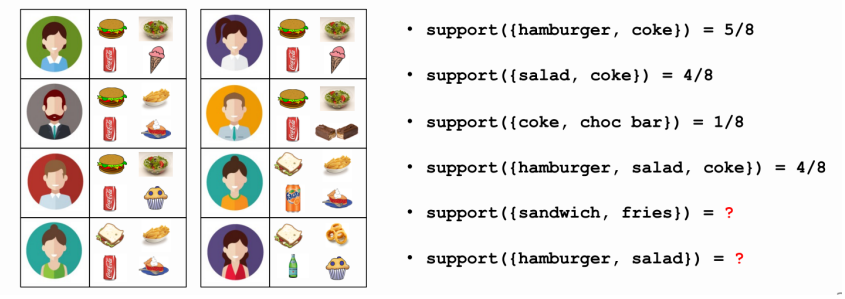


Figure 9.3: Examples of support

9.2.2 Support of an Association Rule

The support of an association rule is defined as

$$\text{support}(X \to Y) = \text{support}(X \cup Y)$$

The support is *direction invariant* and therefore cannot measure the quality of a directed rule

$$\text{support}(X \to Y) = \text{support}(Y \to X)$$

Figure 9.4: Support of association rules

The support measures how frequently an item set occurs in the data. Rules with low support may occur simply by chance, while **good association rules have a high support**.

Theorem 1 *Support = Interestingness*

9.2.3 Confidence of an Association Rule

The **confidence** of an association rule is defined as

$$\text{confidence}(X \to Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$$

For a given rule $X \to Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence therefore measures reliability or trustworthiness of a rule, and a **good association rule has high confidence**.

Theorem 2 *Confidence = Trustworthiness*

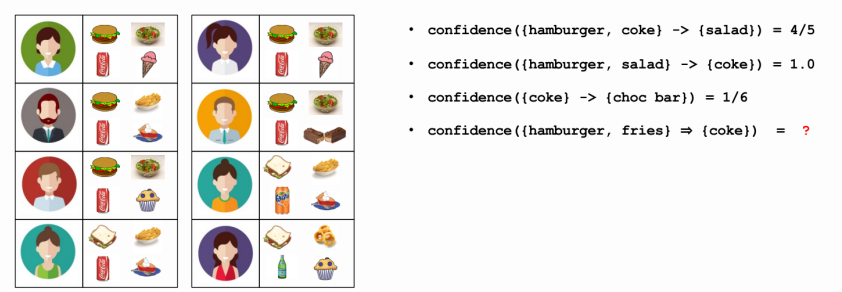


Figure 9.5: Examples of confidence

9.2.4 Apriori Algorithm

Given a set of transactions we want to find all rules having a minimum support $\geq \min_s$ and a minimum confidence $\geq \min_c$. The Apriori Algorithm finds such association rules in two steps:

1. Generate frequent item sets satisfying the support threshold
2. Extract rules from frequent item sets satisfying the confidence threshold

The Apriori Algorithm has the following two properties, the second one allows for efficient pruning (refer to figure 9.6):

1. If an item set is **frequent** (has high support), all of its subsets must be frequent too
2. If an item set is **infrequent** (has low support), all of its super-sets must be infrequent too

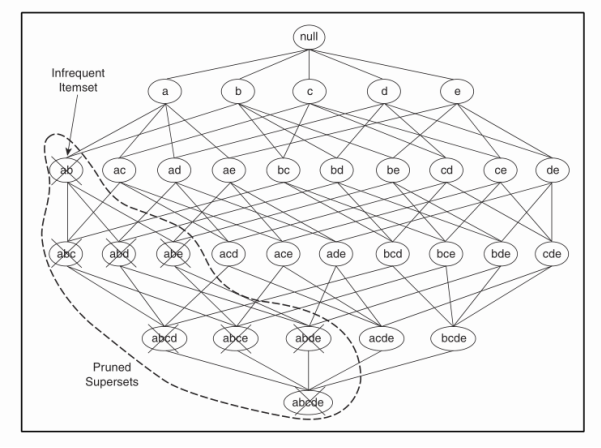


Figure 9.6: Pruning of supersets of sets with low support

After the pruning we generate Rules from the Frequent Item Sets and prune this tree as well. The method for pruning is based on the fact that if a rule $[X \rightarrow Y - X]$ violates the confidence threshold, then any rule $[X' \rightarrow Y - X']$ with X' being a subset of X violates the confidence threshold as well (refer to figure 9.7)

9.2.5 Lift of an Association Rule

The **lift** of an association rule is defined as and measures how many times more often X and Y show up together than expected if they were statistically independent.

$$\text{lift}(X \rightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X) \cdot \text{support}(Y)}$$

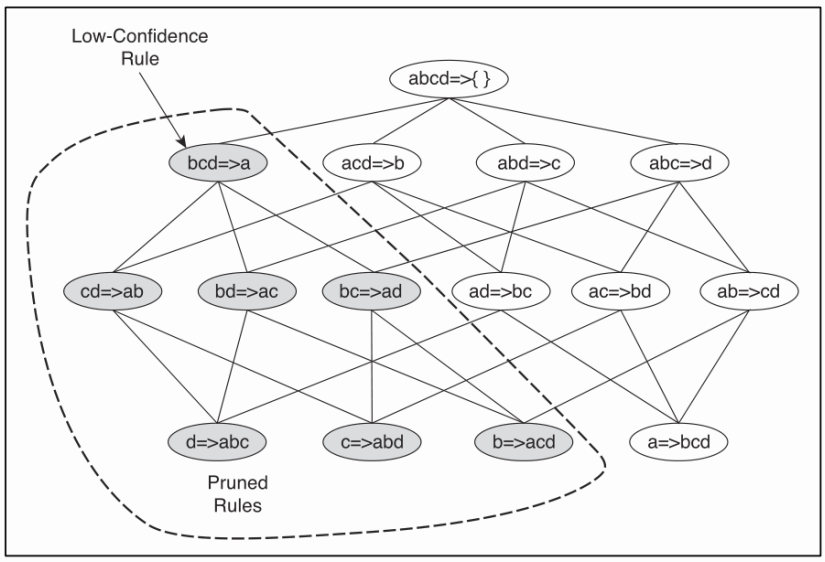


Figure 9.7: Pruning of subset of a low confidence set

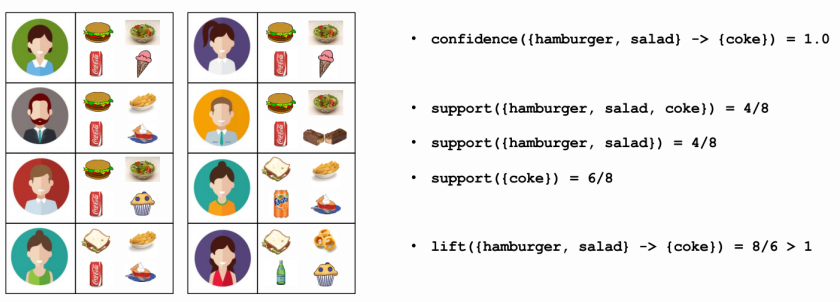


Figure 9.8: Example of lift

lift $= 1$	X and Y are statistically independent
lift < 1	indicates that X and Y appear less often together than expected, the occurrence of X has a negative effect on the occurrence of Y and vice-versa, X and Y are **anti-correlated**
lift > 1	indicates that X and Y appear more often together than expected, the occurrence of X has a positive effect on the occurrence of Y and vice-versa, X and Y are **correlated**

In general, the larger the lift value, the stronger the association between X and Y. A good rule has high lift.

Theorem 3 *Lift = Association Strength*

10 Anomaly or Outlier Detection

Anomaly or Outlier Detection is an important topic in unsupervised learning. Anomaly and outlier are used as synonyms. Anomalies are patterns in data that do not conform to a well-defined notion of normal behaviour.

We differentiate between three types of outliers

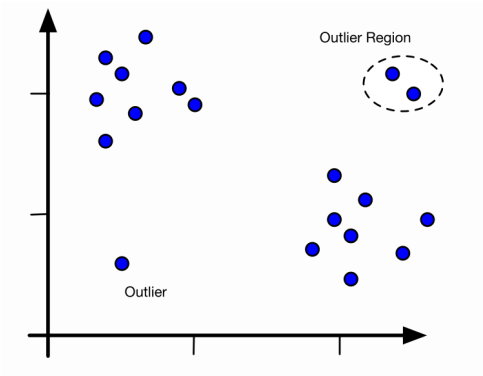


Figure 10.1: Example of an outlier and outlier region

1. **Global outliers** deviate significantly from the rest of the dataset
2. **Contextual outliers** deviate significantly with respect to a specific context
3. **Collective outliers** deviate as a group from the entire dataset but not necessarily as individuals

We can solve the problem of outlier detection with machine learning:

Supervised Learning Outlier detection can be considered a classification problem with categories *normal* and *outlier*.
When outliers are rare events, collecting enough training data becomes a severe issue, and special measures must be taken because the two classes are extremely unbalanced

Unsupervised Learning These methods assume that outliers deviate from the structural pattern of normal data objects.

1. Statistical Methods
2. Proximity-based Methods
3. Clustering Methods

10.1 Statistical Methods

Statistical methods assume that normal data objects are generated by a stochastic model, and that data not following that model are outliers.

1. Assumption: The data follows a normal distribution with parameters μ and σ^2
2. Method: Find the best parameters with Maximum Likelihood method
3. Conclusion: Find threshold where most of the data is located, for example $\mu \pm 3\sigma$ and classify all data outside of this range as outliers

Observe that this techniques for anomaly detection consider only single variables.

10.2 Proximity-based Methods

In proximity-based methods, records that are far from others are considered outliers. There are two types of proximity-based methods:

1. **Distance based methods** find outliers with respect to the global dataset
2. **Density-based methods** find outliers with respect to local neighbourhood

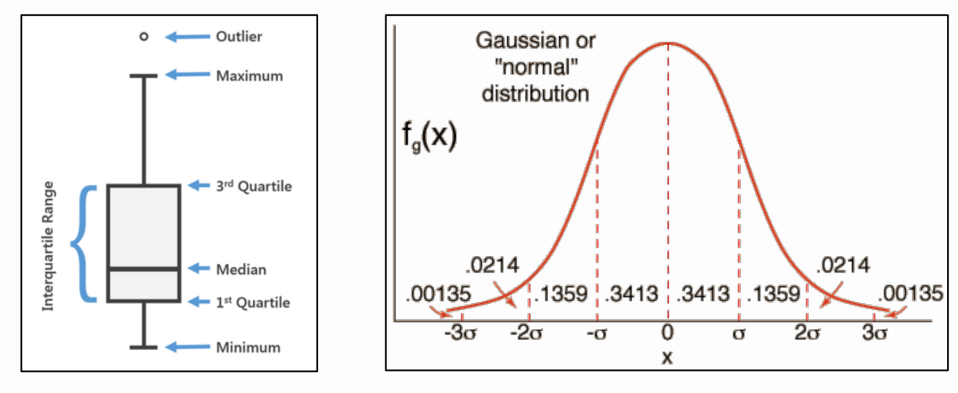


Figure 10.2: Comparison of BoxPlots to Normal Distribution

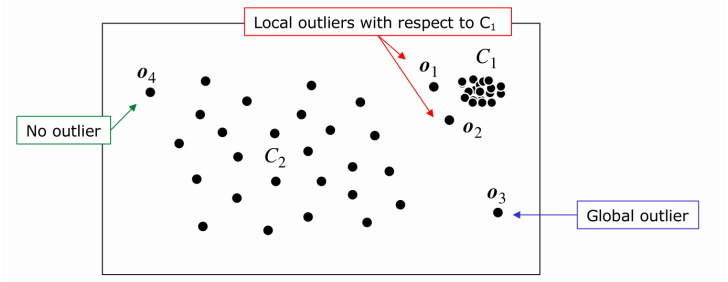


Figure 10.3: Outliers classified by proximity-based methods

10.2.1 Distance-Based Outliers

- Specify a distance threshold $r > 0$ and a fraction threshold $0 < \pi \leq 1$
- For each record count the numbers of other records in the r-neighbourhood
- Consider a record as outlier if $\frac{|\{o' \in D : o \neq o' \text{ and } dist(o,o') \leq r\}|}{|D|} \leq \pi$

10.2.2 Density-Based Outliers

The density around a non-outlier is similar to the density around its neighbours, while the density around an outlier is significantly different from its neighbours (refer to figure 10.4).

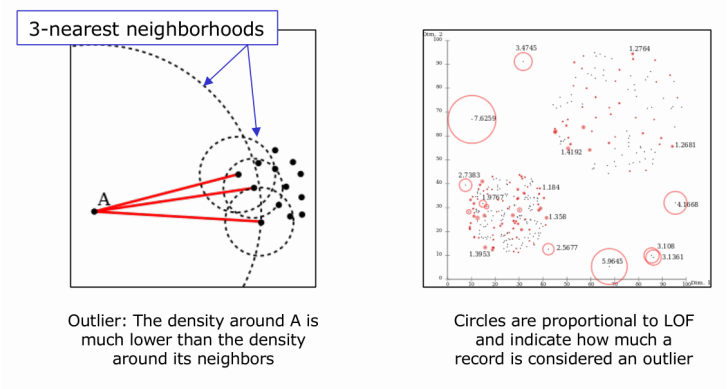


Figure 10.4: Graphical representation of density-based outlier detection

The local outlier factor (LOF) is a score that indicates how likely it is that a certain data point is an outlier. A LOF ≈ 1 means no outlier, while LOF $\gg 1$ means outlier. The k-Distance

is the distance of a point to its k-th neighbour.

Reachability Distance The reachability distance is the maximum of the distance between two points and the k-distance of the second point. Consider it as a smoothing term, as a lower distance bound that depends on the neighbourhood density.

$$\text{reachability distance}_k(A, B) = \max\{\text{k-distance}(B), d(A, B)\} \tag{53}$$

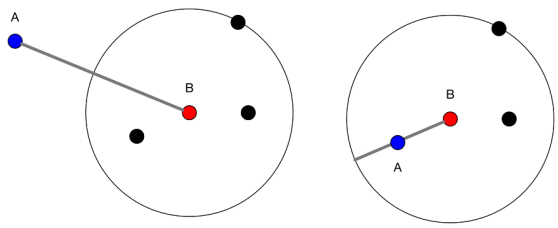


Figure 10.5: Reachability distance visualised

Local Reachability Density (lrd) Calculate the reachability distance of A to all its k-nearest neighbours and take the average of that number, then get the density by taking the inverse. Intuitively, the local reachability density tells how far we have to travel to reach the next point or cluster of points, the lower the local reachability density of a point, the less dense its region, the longer we have to travel.

$$\text{lrd}_k(A) := \frac{1}{\left(\frac{\sum_{B \in N_k(A)} \text{reachability distance}_k(A,B)}{|N_k(A)|}\right)} \tag{54}$$

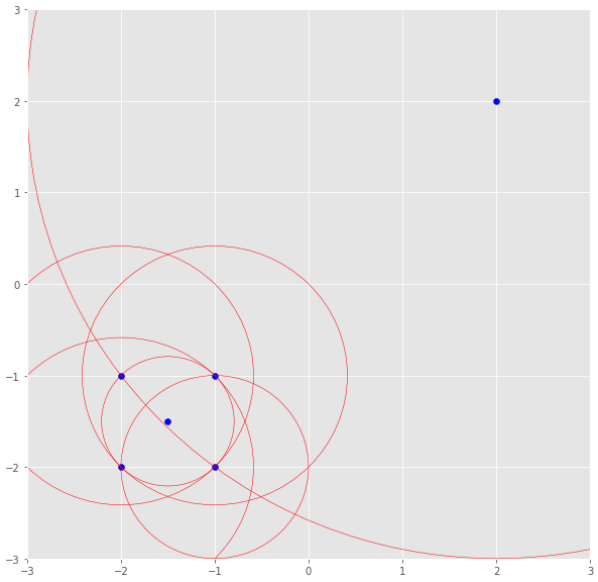


Figure 10.6: The local reachability density is not symmetric

Local Outlier Factor The local outlier factor is the local reachability density of a point in comparison to its k neighbours, more specifically, k ratios of the local reachability distance of each point to its neighbouring points are calculated and averaged.

$$\mathrm{LOF}_k(A) := \frac{\sum_{B \in N_k(A)} \frac{\mathrm{lrd}(B)}{\mathrm{lrd}(A)}}{|N_k(A)|} = \frac{\frac{\sum_{B \in N_k(A)} \mathrm{lrd}(B)}{|N_k(A)|}}{\mathrm{lrd}(A)} \tag{55}$$

$\mathrm{LOF}(k) \approx 1$	similar density as neighbours : **Normal**
$\mathrm{LOF}(k) < 1$	higher density than neighbours : **Inlier**
$\mathrm{LOF}(k) > 1$	lower density than neighbours : **Outlier**

10.3 Clustering Methods

Clustering-based approaches detect outliers by examining the relationship between records and clusters.

1. Records not belonging to any cluster are outliers
2. Records with a large distance to their closest cluster centre are outliers
3. All records in a small and sparse cluster are considered outliers

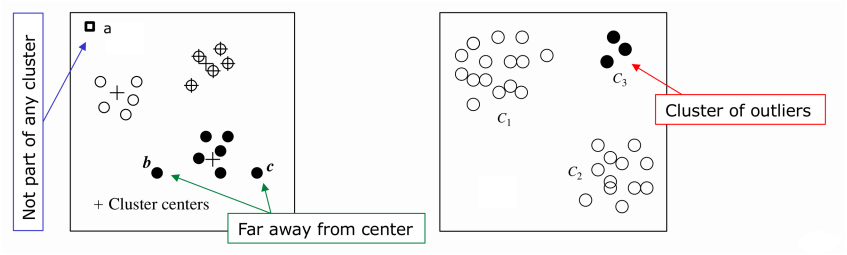


Figure 10.7: Different kind of cluster outliers

Classification-based Approaches Train a classifier, a Support Vector Machine for example, to distinguish normal data from anomalies and generate the decision boundary of the normal class. New objects outside the decision boundary are considered outliers.

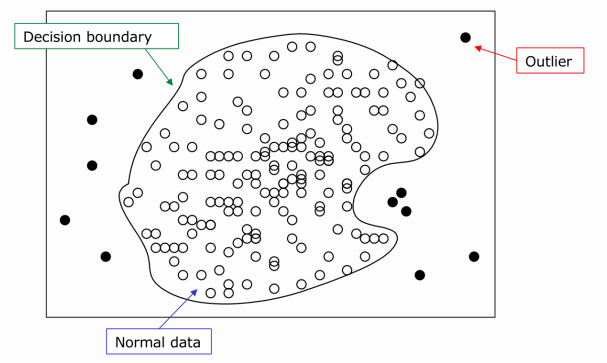


Figure 10.8: Example of a classifier for outlier detection

11 Recommender Systems

11.1 Definition

> "A recommender system is any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options" (Burke, 2002)

Or easier said:

> "A recommender system is a software application capable of suggesting interesting things to its users after learning their preferences over time" (Jennach et. al., 2010)

So basically recommender systems are the algorithms that show up under your Amazon searches as 'customers who bought [xxx] also bought [yyy]'.

As opposed to Support or Lift, recommender systems are **individualized**.

The goal of recommender systems is to give customers a better overview, to help them cope with the information overload problem (Amazon currently has over 20'000 results of 'Laptop Stand', but I only need one).

As a nice side effect, companies make more sales, because their customers can actually find what they're looking for.

11.2 History

The history of recommender systems can be split into three waves:

1st wave (1991 - 2000)
5-star ratings

2nd wave (2001 - 2010)
Integration of personal web pages/facebook/myspace etc.

3rd wave (2011 -)
Smart technologies, big data, voice activated assistants

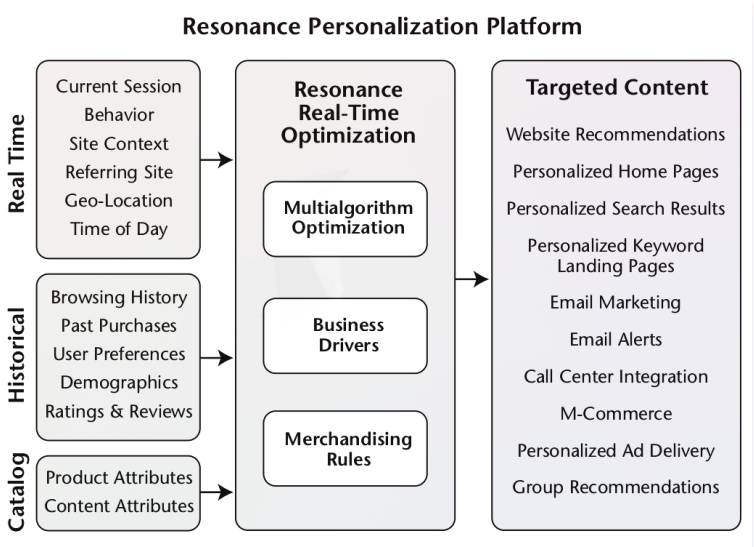


Figure 11.1: Generic Picture of a Commercial Recommender System

11.3 Business Model

- Recommender Systems are only provided as SaaS (Software as a Service), you only rent the system
- All data is managed by the vendors. The clients only get the recommendations produced by this data and can display them at their will
- Subscription prices for these systems are related to their success, usually clients pay a percentage of their revenue
- Due to the recent privacy-boom, some customers and clients do not want to provide all shopping data to these system-vendors. They'd rather have an in-house system.
- Having loads of data is hardly bad, the *real* success in these systems lies in the configuration of the system and the evaluation of the data. This requires a deep understanding of the recommender algorithm.

11.4 Recommendations by Associations

Most people think alike. People who buy cereal mostly also buy milk.

To calculate the percentage of buyers who bought Y after buying X:

$$\frac{X \text{ and } Y}{X} \tag{56}$$

This method is simple enough, but does not really work. A good example of why this won't work is Anchovies Paste and Banana. Basically everybody who buys Anchovies Paste also buys Banana. Does that mean they should be sold together as a package? No, because these two products do not have anything in common. You could pair up basically every 'exotic' product with Banana according to this calculation. This is the case because *Everybody needs bananas.* The fact that they also bought Anchovies Paste does not have anything to do with the fact that they needed bananas.

A better way to calculate basically the exact same thing, without having the Anchovies/Banana problem would be to check the percentage of buyers who bought the Anchovies Paste and also bought bananas and compare that number to the percentage of buyers who has not bought Anchovies Paste but bought bananas. That way, the popularity of bananas is of no consequences.

$$\frac{\frac{X \text{ and } Y}{X}}{\frac{\neg X \text{ and } Y}{\neg X}} \tag{57}$$

Another important thing to check before making a recommendation: What does the buyer already have in their cart? Let's take the example of a fast food restaurant: I am ordering ice cream and get the recommendation that I might also like ketchup. While I *do* like ketchup (although Mayonnaise, especially garlic one is still superior), I do not necessarily want it as a side to my ice cream.

Another example: If I buy ski boots, you could assume that I already own a pair of skis, so why would you recommend me to buy yet another pair of skis? Most people who bought skis and ski boots probably bought them at the same time.

To summarize:

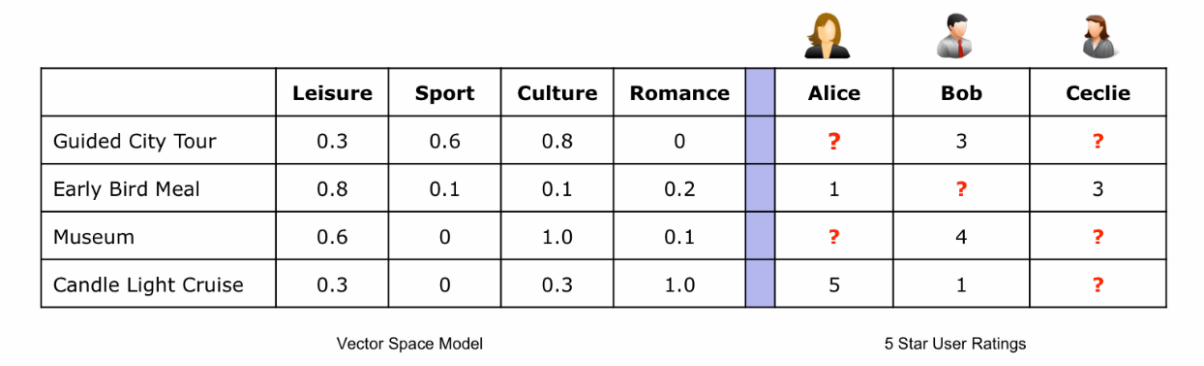
- Recommendations by Associations are *fast.* No data collection and evaluation necessary

- Associations are *context aware*. Only pairs of products that match together will be recommended together
- Associations can be specifically *tailored* to certain businesses
- Associations look at *shopping carts* instead of buyers and do not consider the customers personal preferences

11.5 Context-Based Recommendations

Instead of checking what customers actually bought, why not check and leverage what customers *want* to buy?

Thanks to user ratings, wish lists, watch lists, viewed items etc., online vendors can create a very detailed shopping profile of their customers.



	Leisure	Sport	Culture	Romance		Alice	Bob	Ceclie
Guided City Tour	0.3	0.6	0.8	0		?	3	?
Early Bird Meal	0.8	0.1	0.1	0.2		1	?	3
Museum	0.6	0	1.0	0.1		?	4	?
Candle Light Cruise	0.3	0	0.3	1.0		5	1	?

Figure 11.2: Context Based Recommendations

	Leisure	Sport	Culture	Romance
Guided City Tour	0.3	0.6	0.8	0
Early Bird Meal	0.8	0.1	0.1	0.2
Museum	0.6	0	1.0	0.1
Candle Light Cruise	0.3	0	0.3	1.0

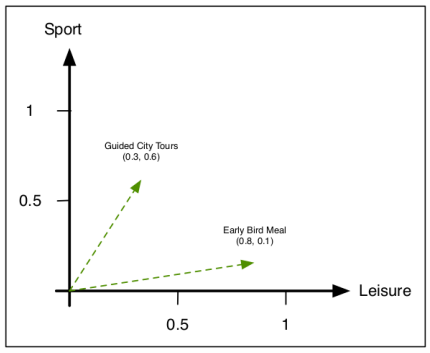


Figure 11.3: Item Similarity

The idea of Context-Based Recommendations is to recommend the user similar products they have not rated/bought yet.

1. Manually or automatically attribute your catalogue data
2. Collect as much user preferences as you can (e.g. by ratings, wish lists, item viewed etc.)
3. Compute predictions for products labelled with ? based on item similarity (See Equation 12 on page 14)
4. Recommend products with high prediction value

Another approach of Context-Based Recommendations is to analyse similar products:

Neighbors N_j refer to the (at most) N rated items most similar to item j, N being a hyperparameter that can be tweaked for improved algorithm-performance Select only rated items with similarity > 0 (choose less than N if you cannot find enough candidates)

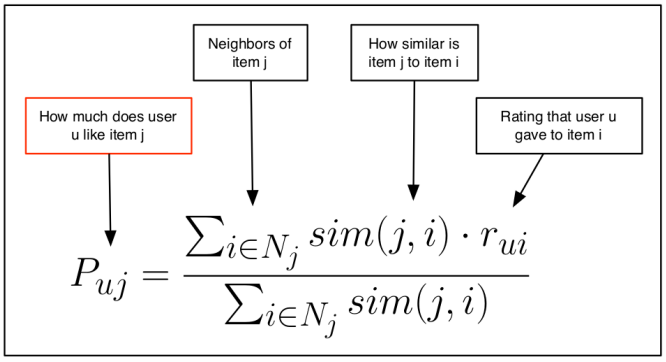


Figure 11.4: Context Based Recommendations with Neighbours

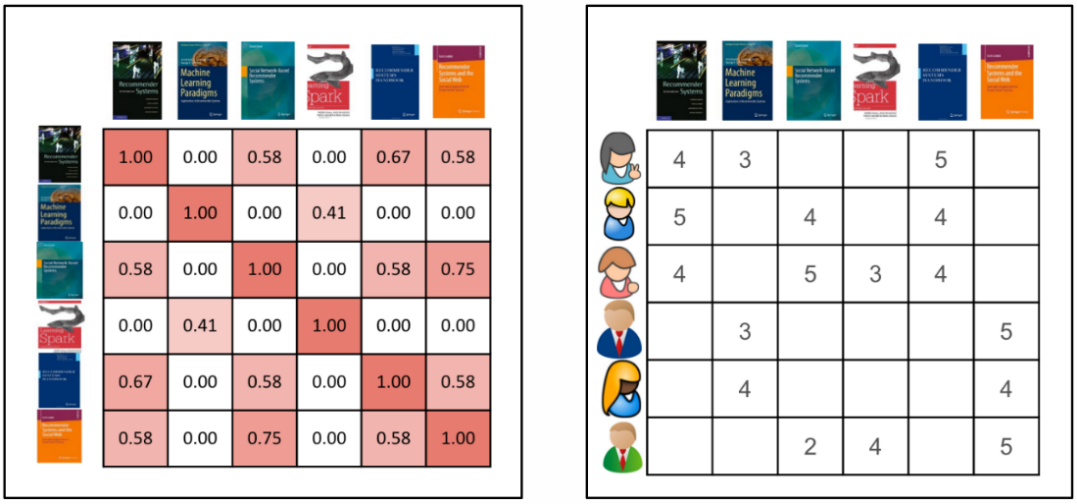


Figure 11.5: Similarity between books

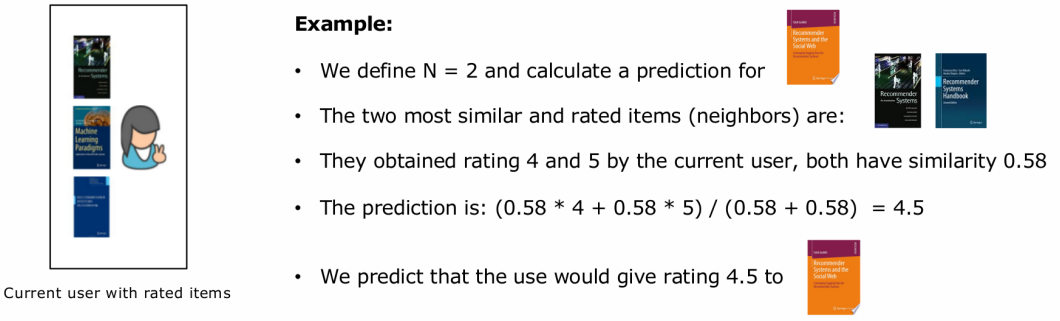


Figure 11.6: Example of N-Based Context-Based Recommendations

Advantages of Contex-Based Recommendations

- A users profile is built only from their own ratings and recommendations can therefore be better explained
- New items can be recommended without having to be rated first

Disadvantages

- The user will never find something unexpected they might like in their recommendations
- Do product attributes really reflect how users decide?
- What do you recommend new users?

11.6 User-to-User Collaborative Filtering

This approach is similar to the similar products' context based recommendations, but instead of similar products, it compares the customer to *other similar customers*.

A set of users who liked the same items in the past, probably share the same preferences. Therefore, the user gets recommendations of products that many 'Neighbours' (similar users) liked, but the user has not purchased (yet).

There's a similar approach as to the Context-Based Recommendations, but there's no need for product attributes any more. The formula to calculate how much user u will wile product j is also fairly similar.

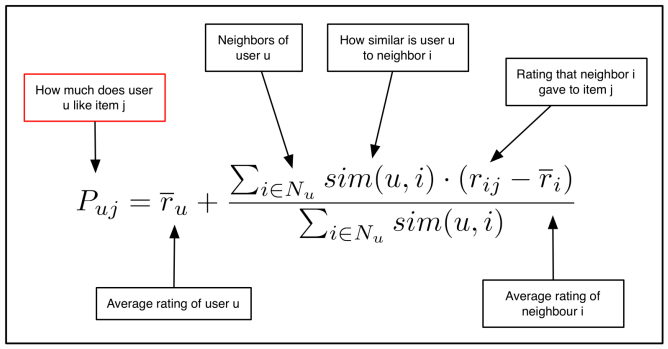


Figure 11.7: Context Based Recommendations with Neighbours

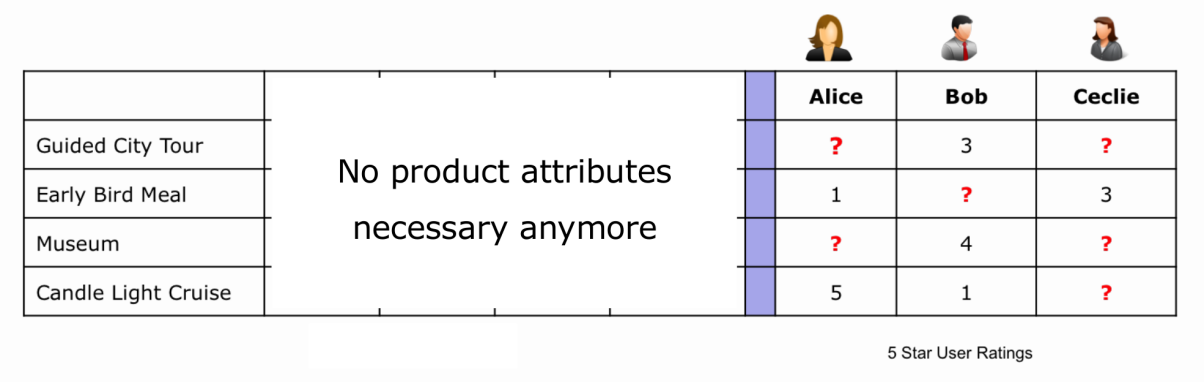


Figure 11.8: User to User Collaborative Filtering

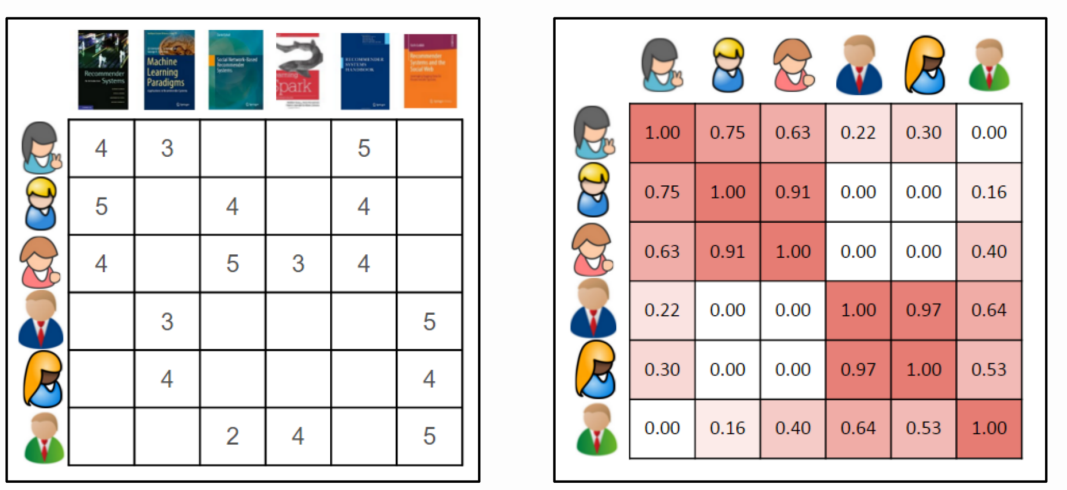


Figure 11.9: Similarity between users

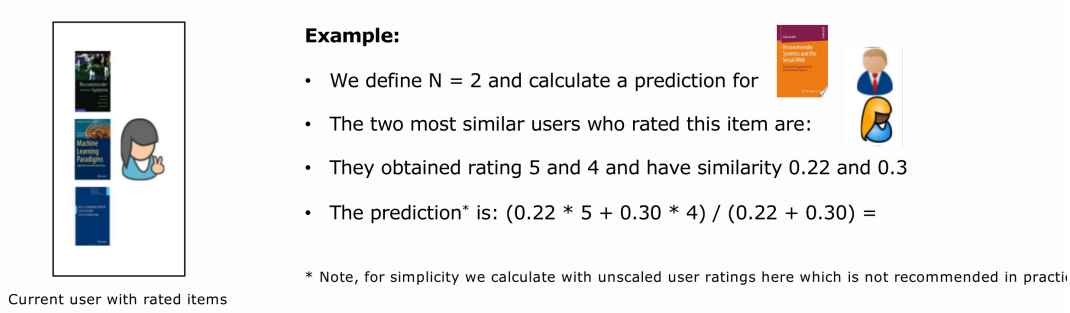


Figure 11.10: Example of N-Based User to User Filtering Recommendations

11.7 Item-to-Item Collaborative Filtering

Because user provides constantly change, pre-computation does not necessarily work. In contrast, for pairs of items, similarity is stable and can be pre-computed.

The main difference is that we compare row vectors for neighbour search instead of column vectors.

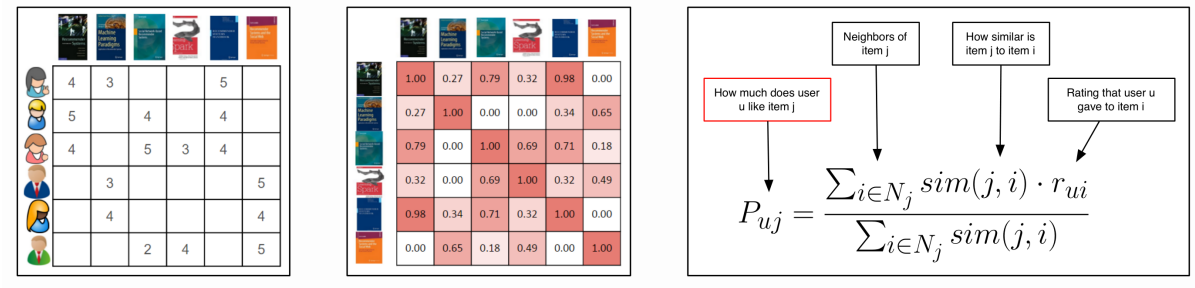


Figure 11.11: Item to Item Collaborative Filtering

Advantages of Collaborative Filtering Recommendations

- Products from different categories can be recommended

Disadvantages

- If the matrix is too sparsely filled, no neighbours will be found
- It needs loads of people to participate to make decent recommendations
- The system must first learn new users preferences
- Users with non-mainstream taste will not get decent recommendations

11.8 Low Rank Matrix Factorisation

To improve the recommendations for sparsely filled matrices, there's something called **Low Rank Matrix Factorisation**. The matrix we saw before (Fig. 11.11, 11.9, 11.5) can be split up in two matrices, which are more densely filled.

One matrix holds all users and their ratings for the products they rated and the second matrix holds all products.

The first matrix (U) holds all users and their ratings. Each row is a user and all the ratings on all products they have made.

The second matrix (V) holds all the products and the rating they've got. This matrix is **transposed**.

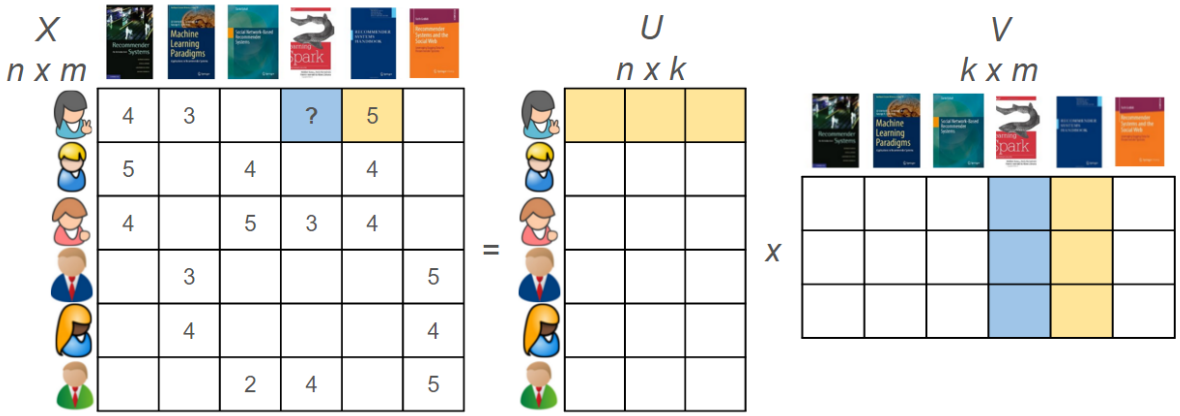


Figure 11.12: Low Rank Matrix Factorization

From this, we can see that the original matrix can be written as $U \times V^T$

$$U = \begin{bmatrix} U_{1,1} & U_{1,2} & \dots & U_{1,k} \\ U_{2,1} & U_{2,2} & \dots & U_{2,k} \\ U_{3,1} & U_{1,2} & \dots & U_{3,k} \\ U_{4,1} & U_{4,2} & \dots & U_{4,k} \\ \dots & & & \\ U_{n,1} & U_{n,2} & \dots & U_{n,k} \end{bmatrix} \qquad V = \begin{bmatrix} V_{1,1} & V_{1,2} & V_{1,3} & V_{1,4} \dots & V_{1,m} \\ V_{2,1} & V_{2,2} & V_{2,3} & V_{2,4} \dots & V_{2,m} \\ \vdots & & & & \\ V_{k,1} & V_{k,2} & V_{k,3} & V_{k,4} \dots & V_{k,m} \end{bmatrix}$$

$U_{1,1}$ is the rating User 1 gave the book 1, $U_{2,1}$ is the rating User 2 gave the book 1 etc.

v is the product matrix. The V_1 column stores all ratings book 1 has got, V_2 all ratings of book 2 etc.

So if we want to check what rating user 3 has given book 5, we just need to calculate $U \times V^T$ and get the entry $(3, 5)$.

That way, we can also 'guess' what rating a user would give a book they have not rated or bought yet. The rating of book 2 by user 2 is sorted in $U \times V, (2, 2)$.

11.9 Hybrid Recommender Systems

The best result can usually be achieved if you mash multiple recommender-algorithms together. That way, their different advantages and disadvantages might cancel each other out. If you mix a content-based and a collaborative algorithm, you could for example eliminate the cold start problem.

But how could you interpret the different results from the different algorithms?

Weighted: Score an item as a weighted sum of scores from different algorithms

Mixed: Results of different algorithms presented to the user

Cascade: One algorithm refines the result of another one

Switching: Different methods/algorithms can be used for different use cases

11.10 Evaluation

How would you evaluate recommender systems? Typical train/test/validate splits of your data will not work.

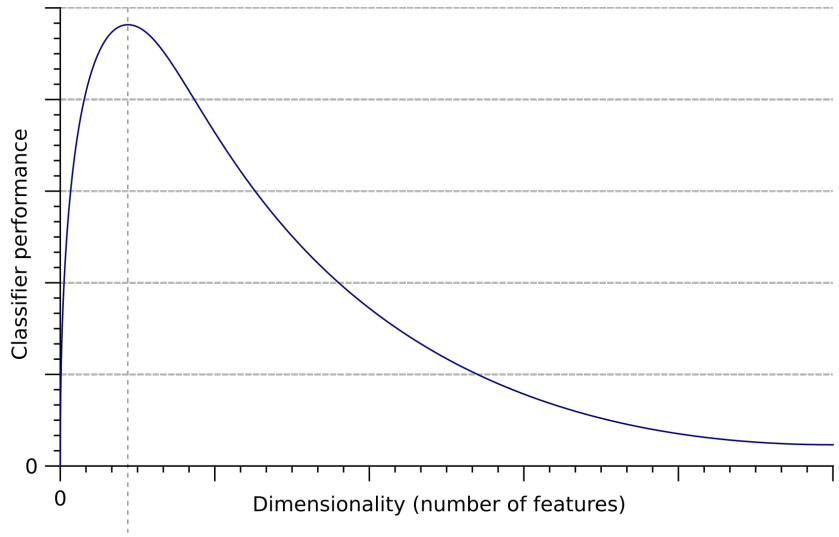
Instead the data must be split over several users, because to assess how good the recommendations are, we need the users whole shopping history.

An often used performance measurement is $P@k$ (Precision @ k). How many of the k recommended items were actually relevant. In case many users have less than k recommended products, you could also go with $minP@k$.

12 Dimensionality Reduction

In Big Data, the provided data usually has *a lot* of features, many of which do not add any significant information and could be deleted without losing anything important.

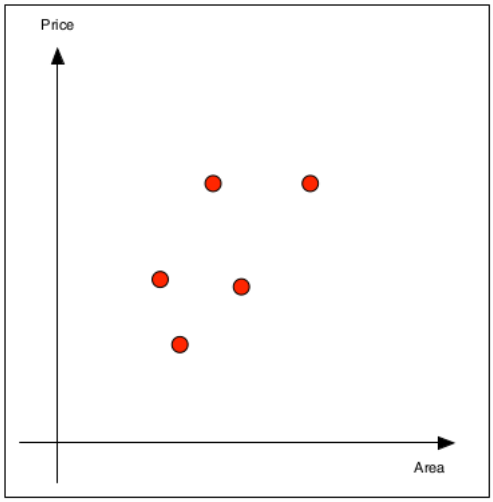
The following figure shows that more features don't necessarily produce better results



Optimal number of features

12.1 Reduction by Projection

Let's say we want to reduce the dimensions on the right to one dimension instead of two. Which dimension should be removed to keep the most information? Price or Area?



If you look at figure 12.1, you can see that for the correct solution, you need to think a bit outside the box. Instead of simply cutting away one or another dimension, the dimensions are *rotated* and are a linear combination of the original dimensions.

$$\text{Area} = \alpha_1 \cdot \text{Area} + \beta_1 \cdot \text{Price}$$

$$\text{Price} = \alpha_2 \cdot \text{Area} + \beta_2 \cdot \text{Price}$$

or more precisely:

$$x' = x \cdot cos\theta + y \cdot sin\theta \qquad y' = -x \cdot sin\theta + y \cdot cos\theta$$

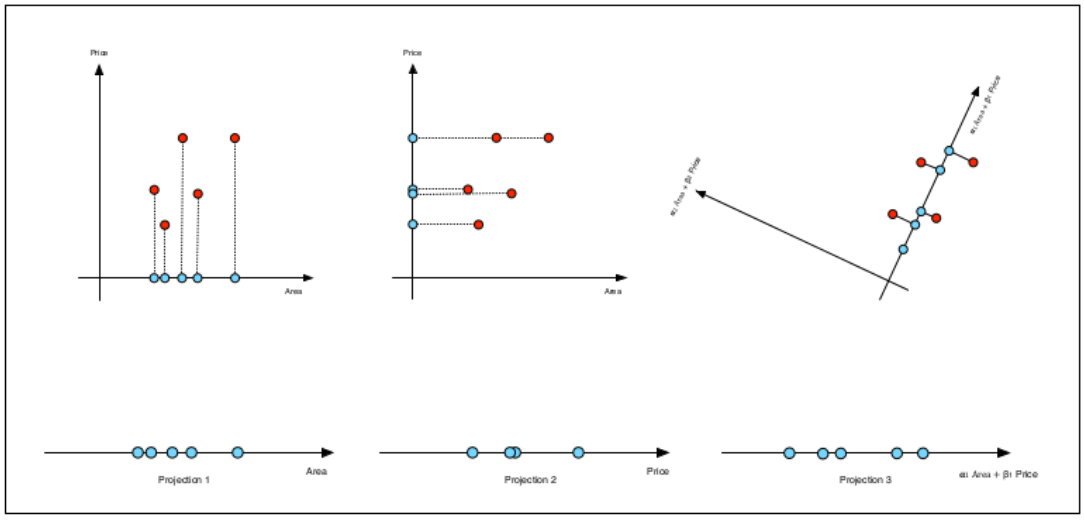


Figure 12.1: Different ways to reduce dimensions of data points

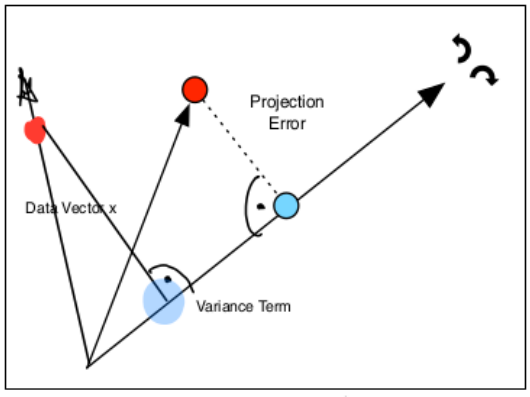


Figure 12.2: How to measure retained information

With almost any dimension reduction, some information will get lost. The question is *how much*?
Generally, the following holds true:

> "The smaller the error, the larger the variance, the more informative the projection"

As can be seen in fig. 12.2, the further away the vector strays from the variance term, the larger the projection error grows. This in turn falsifies the information.
Therefore: The best projection is the one that minimises information loss or equivalently that maximises the variance.

12.2 Data Redundancy

As said in the previous page: With almost any dimension reduction, some information will get lost.

However, sometimes this does not hold true. This is the case if some data is *redundant*

Take for example a **mean-centred feature** that always takes the same value (so, the mean). This feature does not give you any valuable information and can therefore safely be dropped.

Another example are **several features that essentially say the same**, say the height of a test person in centimetres and in inches. One of the two can be dropped. In the unlikely case that both measurements need to be used, one can easily be calculated from the other.

12.3 Strategies for Dimensionality Reduction

To summarize: There are two strategies to reduce the number of dimensions:

1. Eliminate redundancy
   Fully redundant features can be dropped without any information loss
   Combine features in order to remove all covariance in the data

2. Destroy as little information as possible
   Mimize the information loss during the projection by maximizing the retained variance in the projected data.

12.4 Base Transformation on Data Matrices

As already seen in section 11.8, a $m \times n$ matrix Y can be split up in a $m \times n$ matrix X and a $m \times m$ matrix P.

$$P \cdot X = Y$$

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \cdot \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

P essentially just rotates and stretches X to transform it into Y. Therefore, P performs **base transformation** on X.
The rows of P are the new base vectors of Y, also called **principal components**

12.5 Eigenvectors and Eigenvalues

Eigenvector: A vector that is a only a multiple of itself (i.e. stretches) when multiplied with the matrix A, no rotation!

Eigenvalue: The scalar value by which the vector is stretched when multiplied with the matrix
The equation looks as follows: $Av = \lambda v$

A vector $v \neq 0$ is called an eigenvector if it satisfies $Av = \lambda v$. λ is called the eigenvalue of the eigenvector v.

Eigenvectors (and therefore eigenvalues) only exist for **squared matrices**.

Another neat fact is that every square matrix A can be written as

$$A = EDE^T \tag{58}$$

where D is a diagonal matrix of eigenvalues and E is a matrix of eigenvectors of A arranged as columns.

12.6 Principal Component Analysis (PCA)

As already discussed in section 2.3.7, the **covariance matrix** shows the covariance from all X with all Y. As $Cov(x, x) = Var(x)$, the covariance matrix has the variance of X in its diagonal

The covariance matrix of a data matrix Y, S_Y shows covariance between features in the off-diagonal terms. As we have learned in section 12.3, we want to eliminate covariance between features to reduce redundancy. Therefore, the covariance matrix S_Y should be a **diagonal matrix** (meaning all values except the diagonal values are zero).

Additionally, the diagonal values of S_Y, which show the variance of each dimension can be used to eliminate low-variance dimension, as they are nearly mean-centred and therefore useless.

Since $Y = PX$, we can say

$$S_Y = \frac{1}{n-1} YY^T = \frac{1}{n-1}(PX)(PX)^T \qquad = \frac{1}{n-1} PXX^T P^T = \frac{1}{n-1} P(XX^T)P^T$$

Therefore, the goal of PCA is to find a square matrix P to a data matrix X where the following holds true:

$$P(XX^T)P^T \text{ is a diagonal matrix} \tag{59}$$

The rows of this matrix P are the **principal components** of X.

12.6.1 Calculation

1. Compute all eigenvectors of XX^T
2. Build a matrix E whose columns are the eigenvectors of XX^T
3. Define $P = E^T$

12.6.2 Dimensionality Reduction with PCA

- The diagonal covariance matrix S_Y is made of the eigenvalues of XX^T
- Every eigenvalue corresponds to the variance of one feature in our data matrix Y
- Features with zero variance do not contain any information → Drop them
- The sum of eigenvalues corresponds to the total variance in the data
- Calculate the percentage of how much each feature contributes to the total variance
- Drop the features that don't contribute enough variance to be relevant

In practice:

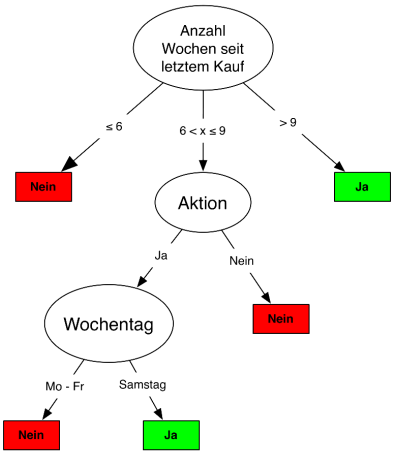
1. Each eigenvector (row) in $P = E^T$ matches to one eigenvalue in S_Y
2. sort the eigenvalues in S_Y in decreasing order and rearrange the rows in P
3. Eliminate features (rows) from the lowest to the highest eigenvalue

13 Decision Trees and Random Forests

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

13.1 Historical Survey of Decision Tree Algorithms

- ID3 (Iterative Dichotomiser 3) invented in 1986
  Uses **information gain** for feature selection
- C4.5 added a number of improvements and extensions to ID3
  Uses **information gain** for feature selection, integrates these numerical features though automated interval partitioning, has more sophisticated tree pruning and features with costs
- CART (Classification and Regression Trees) is very similar to C4.5
  can be used for classification and regression, uses **Gini index** for feature selection in classification, then uses variance reduction (mean squared error) for feature selection in regression, constructs binary trees with feature selection for every possible split and is used in scikit-learn in an optimised version




MIGROS

Menge	Rabatt		
1.000	0.00	ANDROS ORANGENSAFT 1L	4.95
1.000	0.00	CLEVERBAG 17L ROLLE	2.25
1.000	0.00	GURKEN TP ST S.	1.30
1.000	1.50	TR.WEISS KERNLOS AB 500G	2.10
1.000	0.75	ERDBEEREN AB 250G	1.95
1.000	0.00	SE OSTEREIER CH FL 50+4ER	3.25
1.000	0.00	DATTELTOMATEN AB 250G	2.50
1.000	0.80	THAIK KOKOS-MILCH 500ML	3.20
0.321	2.25	OP ADR P-SCHNITZEL 2STK	8.35
1.000	0.00	SOFT TORTILLAS 326G	4.80
2.000	0.00	HEI M-DRINK HOCHPAST 1L	3.30
0.179	0.00	BANANEN I LOSE	0.50
0.162	0.00	TS BRATSPECK MIDI	3.40
1.000	0.00	BUNDZWIEBEL BD S.	1.40
1.000	0.00	AB ENDIVIENSALAT 200G	2.40
1.000	0.00	BIO SULTANINEN 400G	2.80
0.156	0.00	WILDBACHKAESE FP 150G	3.75
1.000	0.00	TOMME LA FLEUR	5.50
0.214	0.00	APF.JAZZ I LOSE	1.05
1.000	0.00	EIER CH FL 53G+ 6ER	3.60
1.000	0.00	CUM +5x Pkt. ges Sort	0.00
2.000	0.00	GOUDA IN SCHEIBEN 150G	5.20
1.000	0.40	BIO JOGHURT HIMBEER 500G	1.50
0.096	0.00	SE VALENTINS-PLATTE MAXI	4.75
1.000	0.00	TS BUTTERZOPF 700G	4.50
1.000	0.50	MANGO ST S.	1.70
1.000	0.00	VALFLORA BUTTER 100G	1.40

Figure 13.1: An example of a decision tree based on a purchase

13.2 Tree Construction Rules

1. If only positive or negative instances are left, stop branching and assign the corresponding decision as leaf node
   Example: On Offer = 0% or 50%
2. If some positive and some negative examples remain, choose another feature and refine the tree one step further
   Example: On Offer = 20% → next feature: Saturday
3. If no instances are left, such a combination of feature values does not occur in the training set. In this case, look at the parent node and decide according to the more frequent label
4. If there are still instances but no features any more, the training data is contradictory, noisy, non-deterministic or contains hidden features. In this case, decide according to the more frequent label.

13.2.1 Splitting Criterion

We need a way to measure how informative a feature is. Then, we always select the most informative feature. There are two main variants called information gain and Gini index.

13.3 Gini Impurity

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset. Gini impurity is non-negative and reaches zero when all cases in the node fall into a single target category. Choose the feature with lowest Gini impurity value for splitting.

$$\sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i(1-p_i) = \sum_{i=1}^{J} (p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2 \tag{60}$$

13.3.1 Example for a Gini calculation

$$\begin{aligned} p(\text{Weather} = \text{Rain}) &\quad \frac{7}{12} \\ p(\text{Weather} = \text{Sun}) &\quad \frac{3}{12} \\ p(\text{Weather} = \text{Cloudy}) &\quad \frac{2}{12} \end{aligned}$$

$$\text{Impurity}(\text{Weather}) = 1 - (\frac{7}{12})^2 - (\frac{3}{12})^2 - (\frac{2}{12})^2 = \frac{41}{72} \approx 0.569$$

13.4 Regression Trees in CART

The regression trees predict the average value of all instances in a leaf node and are grown like classification trees but with a different splitting criterion. As splitting criterion, they minimise the variance of the values in the same subset.

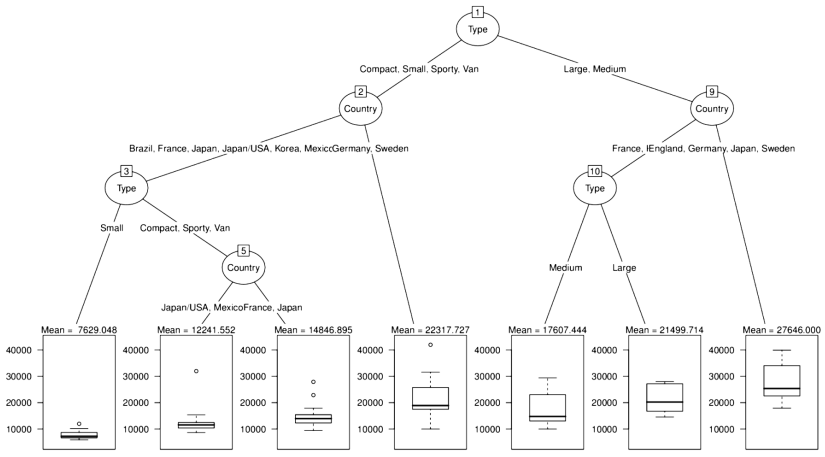


Figure 13.2: A CART regression tree

13.5 Advantages and Disadvantages of Decision Trees

Advantages

- Simple to understand and interpret for humans

- Can handle both numeric and categorical data
- Require almost no data preparation
- Perform well even on larger datasets

Disadvantages

- Trees are often not as accurate as other approaches
- Learners tend to create over-complex trees that do not generalize well - they overfit the data
- A small change in the training data can result in a big change in the tree - highly sensitive to the training data

13.6 Association Rules versus Decision Trees

Mining association rules is unsupervised learning, whereas building decision trees is supervised learning. Whereas association rules process pure transactions, decision trees incorporate other information. Therefore, both techniques can be used in a personalized and unpersonalised manner.

13.7 Random Forest

Random forests try to address the problems of decision trees. They are simply a collection of decision trees built from the same training set using **random sampling with replacement** and a **random attribute selection**, the prediction from a forest is a combination of the individual tree's results.

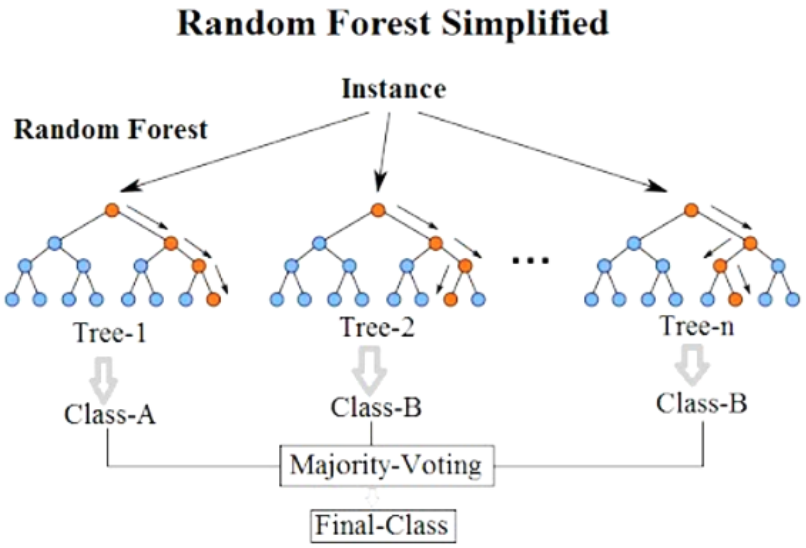


Figure 13.3: An illustration of a random forest

13.7.1 Building a Random Forest

Training Phase

1. Choose a random subset D* of your training set D
2. Choose a random subset A* of attributes
3. Build a decision tree from A* and D*
4. Repeat step 1 – 3 for the number of decision trees you want

Decision Phase

1. Get a separate decision from each tree in the random forest
2. Combine the result of each tree to obtain the final decision by taking the average for regression or majority vote for classification

14 Debugging

Motivation

1. You carefully split your data into training, validation and test set
2. You put aside your test set
3. You train your favourite learning algorithm on the training set
4. You get 20% error on the validation set which is unacceptably high

14.1 Bias and Variance

Different sources of error can prevent supervised learning algorithms from generalizing beyond their training set. The **bias** results from false assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant structure between features and target. While the **variance** results from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data rather than the intended outputs.

- A model with high bias and low variance **underfits** the data
- A model with low bias and high variance **overfits** the data

Also refer to the figure 7.1a in section 7.

14.1.1 Under- and Overfitting in Classification

Underfitting

- A straight line is just not complex enough to separate the classes
- This model performs badly on both training and test data

Overfitting

- This model optimizes to much with respect to the training data
- It shows high performance on training but low performance on test data

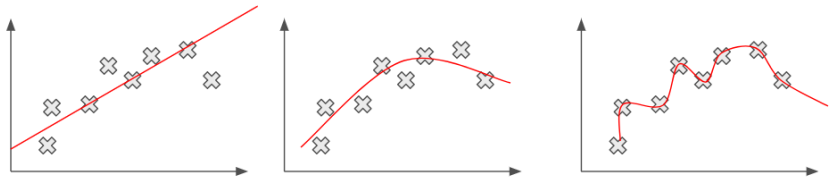


Figure 14.1: Underfitting and Overfitting in Regression

14.1.2 Under- and Overfitting in Regression

Underfitting

- A straight line is just not complex enough to approximate the data
- This model performs badly on both training and test data

Overfitting

- This model optimizes too much with respect to the training data
- It shows high performance on training but low performance on test data

14.1.3 Learning Curve for Underfitting

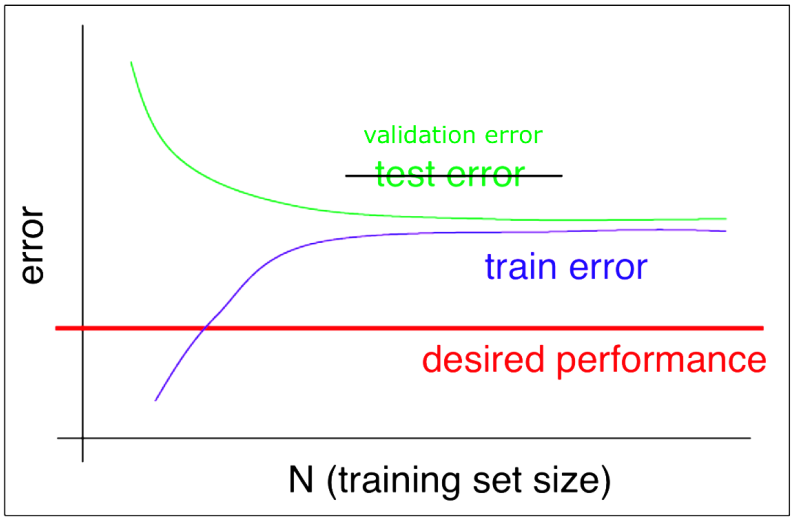


Figure 14.2: Underfitting has bad performance on training and test data

14.1.4 Counteracting Underfitting

Underfitting is often easier to repair, as it means that your model is misses relevant structures, so you can try to make it more complex through:

- A larger set of features
- A different set of features
- More complex machine learning algorithms

14.1.5 Learning Curve for Overfitting

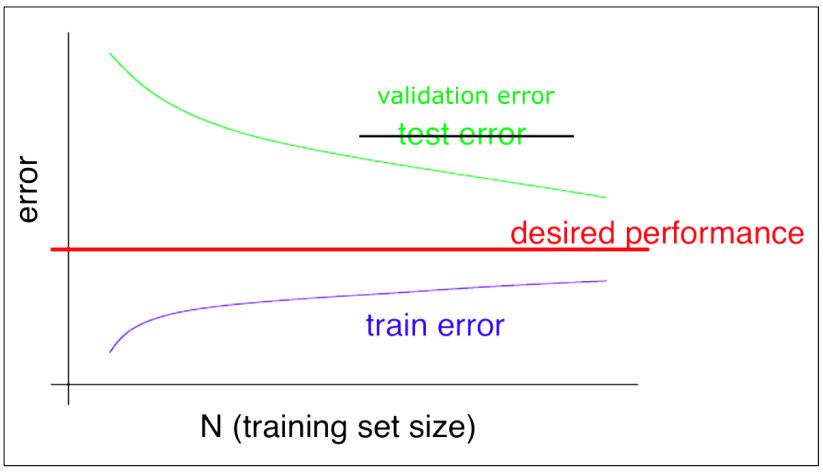


Figure 14.3: Overfitting has good performance on training but bad performance on test data

14.1.6 Counteracting Overfitting

Overfitting happens more frequently and is much harder to repair. It occurs because your model is too sensitive to the training data, you could try to:

- Getting more training data
- Better data cleaning and eliminate outliers
- A smaller set of features through feature selection
- Early stopping during iterative training
- Tuning the regularization parameter
- Ensemble methods

14.1.7 Early Stopping

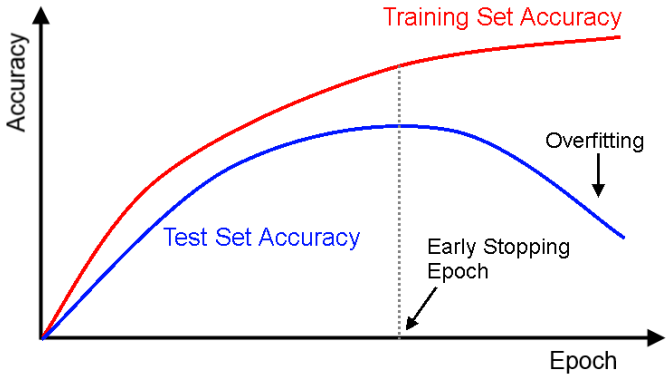


Figure 14.4: Stopping the training early to avoid overfitting

14.2 How to Approach a Machine Learning Project

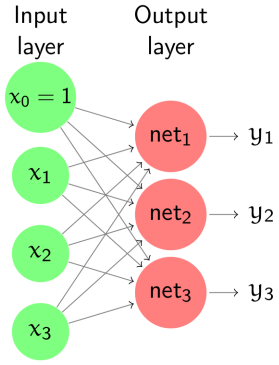
1. Data quality assessment is always worth the time
2. Visualize your data
3. Specify and implement a water-proof quality assessment workflow
4. Implement a quick-and-dirty baseline classifier or regressor
5. Run error analysis and diagnostics
6. Learn and improve

15 Artificial Neural Networks

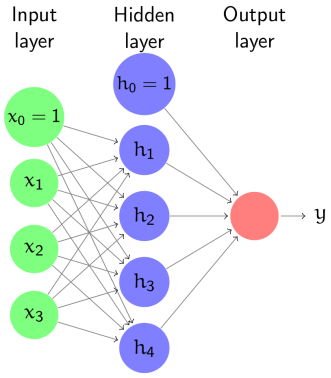
Artificial neural networks try to imitate a biological neural system. The smallest subunit of it is called a neuron. A neuron combines one or multiple input signals from preceding neurons and based on these signals computes an output signal which is propagated to selected neighbouring neurons. Many neurons in multiple layers are able to solve complex problems.

Most artificial neural network (ANN) are layered. We distinguish between the following three types

1. Single-layer feedforward networks
2. Multilayer feedforward networks
3. Recurrent networks (includes feedback into previous layers).



(a) A single-layer feedforward network



(b) Multi-layer feedforward network with one hidden layer

Figure 15.1: Feedforward networks

15.1 The Artificial Neuron

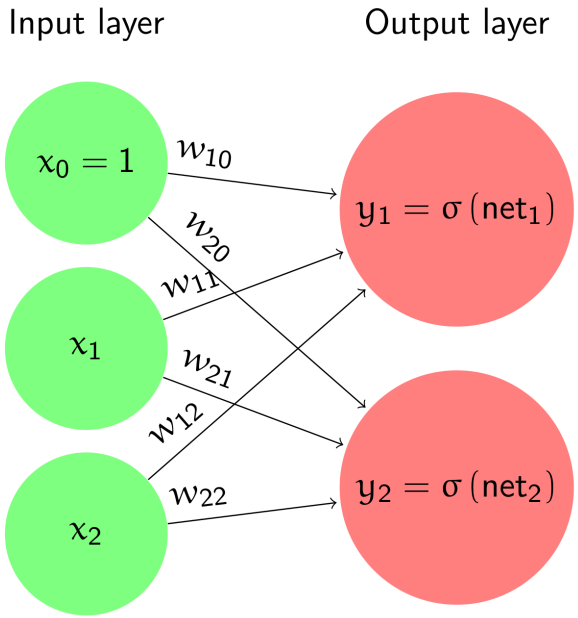


Figure 15.2: Slice of an input and output layer in an artificial neural network

Each neuron fires only if the calculated signal from all the inputs is over a threshold θ. For each neuron j, the weighted sum of the n input signal net_j is calculated as

$$\text{net}_j = \sum_{i=0}^{n} w_{ji}x_i = w_{j0}x_0 + w_{j1}x_1 + \cdots + w_{jn}x_n = w_{j0} \cdot 1 + w_{j1}x_1 + \cdots + w_{jn}x_n \tag{61}$$

Each weight w_{ji} follows the pattern $w_{\text{ReceivingNode InputNode}}$ so w_{12} means the weight for the node 1 and signal node 2. The node x_0 always has a value of 1 and is used as a threshold $\theta_j = -w_{j0}$. Thus, if a neuron fires can be written as

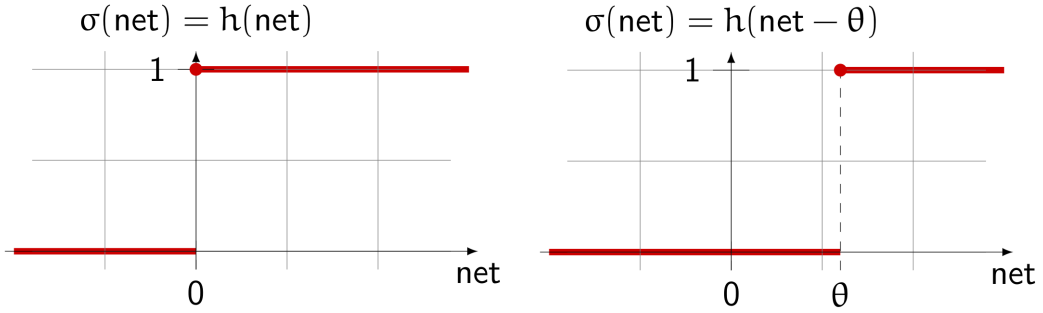
$$\begin{aligned}
\text{net}_j &> 0 \\
\sum_{i=0}^{n} w_{ji}x_i &> 0 \\
w_{j0} + \sum_{i=1}^{n} w_{ji}x_i &> 0 \\
-\theta_j + \sum_{i=1}^{n} w_{ji}x_i &> 0 \\
\sum_{i=1}^{n} w_{ji}x_i &> \theta_j \\
\sum_{i=1}^{n} w_{ji}x_i &> -w_{j0}
\end{aligned}$$

To model the firing of the j-th output neurons, we have to apply the **activation function** σ to the net input

$$y_j = \sigma(\text{net}_j) = \sigma(\sum_{i=0}^{n} w_{ji}x_i) = \sigma(w_{j0}x_0 + w_{j1}x_1 + \cdots + w_{jn}x_n) \tag{62}$$

15.2 The Activation Function

The binary threshold function with threshold zero ($\theta = 0$) is the usual Heaviside function. Shifting this function, we obtain the binary threshold function with threshold θ. The threshold is also called bias.



(a) Heaviside Function with threshold zero

(b) Heaviside Function with threshold θ

Figure 15.3: Heaviside Functions

Using the Heaviside-Function h we can write

$$\sigma(\text{net}_j) = h(\text{net}_j - \theta_j) \tag{63}$$

$$\sigma(\text{net}_j) = \begin{cases} 0 & \text{net}_j < \theta \\ 1 & \text{net}_j \geq \theta \end{cases} \tag{64}$$

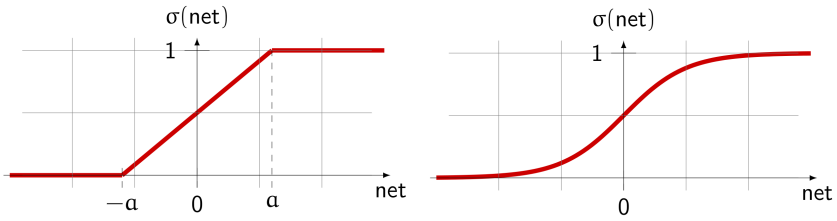


Figure 15.4: The sigmoid activation function

$$\sigma(\text{net}_j) = \begin{cases} 0 & \text{net}_j \leq -a \\ \frac{\text{net}_j}{2a} + \frac{1}{2} & -a < \text{net}_j < a \\ 1 & \text{net}_j \geq a \end{cases} \tag{65}$$

$$\sigma(\text{net}_j) = \frac{1}{1 + e^{-a \cdot \text{net}_j}} \tag{66}$$

$$\sigma'(\text{net}_j) = a \cdot \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) \tag{67}$$

For $a \to \infty$ the sigmoid function converges towards the Heaviside function. Contrary to the threshold function, the sigmoid function has a continuous derivative (see equation 67). This is of paramount importance in multilayer neural networks, where we have to apply back propagation.

15.3 Description of a Neural Network using Matrices

The artificial neural network in figure 15.2 can be described using matrices as follows

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \right) \tag{68}$$

$$\vec{y} = \sigma(\vec{W}\vec{x}) \tag{69}$$

15.4 Single-Layer Feedforward Networks

Input nodes do no computation, they are just used as input. All output nodes compute their output using the same activation function σ (either the Heaviside Step function or the Sigmoid function). These networks are used as binary classifiers if the problem is linearly separable and called Perceptrons.

15.4.1 Learning Rules - Weights Update

The update of the weight w_{ji} from input x_i to output y_j with the learning rate η being a constant typically between 0...0.5

$$\Delta w_{ji} = \eta \cdot y_j \cdot x_i \tag{70}$$

When we factor in the error e_j to account for a target value t_j higher or lower than the actual output y_j, we get what is called the **δ-rule** by Widrow and Hoff or **Hebb's learning rule** by Bürgler.

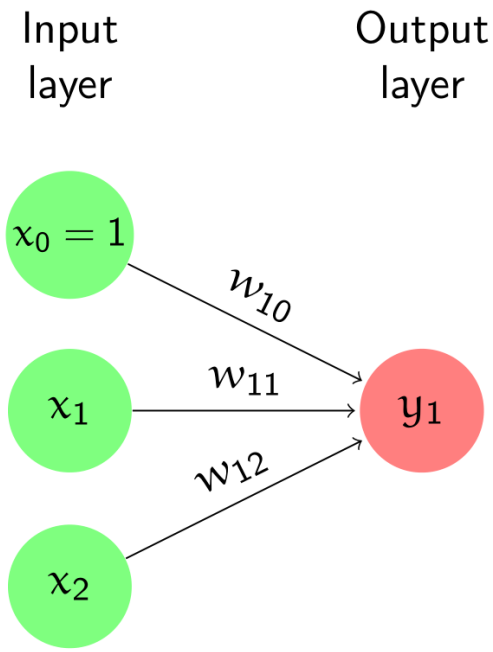
$$e_j = t_j - y_j \tag{71}$$

$$\Delta w_{ji} = \eta \cdot e_j \cdot x_i \tag{72}$$

Example Realise the logic AND function using a single layer feedforward network and the Heaviside function. The starting values are $w_{10} = -1$, $w_{11} = 1$, $w_{12} = 0$ and $\eta = 0.3$.

x_1	x_2	$t = x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0

(a) Function to simulate in the example



(b) Network used in the example

Figure 15.5: Example parameters

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1				

$$\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 = -1 + 1 \cdot 1 + 0 \cdot 1 = 0$$
$$y_1 = h(\text{net}_1) = h(0) = 0$$
$$e_1 = t - y_1 = 1 - 0 = 1$$
$$\eta \cdot e_1 = \eta(t - y_1) = 0.3 \cdot 1 = 0.3$$
$$\Delta w_{10} = \eta(t - y_1) \cdot x_0 = 0.3 \cdot 1 \cdot 1 = 0.3$$
$$\Delta w_{11} = \eta(t - y_1) \cdot x_1 = 0.3 \cdot 1 \cdot 1 = 0.3$$
$$\Delta w_{12} = \eta(t - y_1) \cdot x_2 = 0.3 \cdot 1 \cdot 1 = 0.3$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3

$$w'_{10} = w_{10} + \Delta w_{10} = -1 + 0.3 \cdot 1 \cdot 1 = -1 + 0.3 = -0.7$$
$$w'_{11} = w_{11} + \Delta w_{11} = 1 + 0.3 \cdot 1 \cdot 1 = 1 + 0.3 = 1.3$$
$$w'_{12} = w_{12} + \Delta w_{12} = 0.3 + 0.3 \cdot 1 \cdot 1 = 0 + 0.3 = 0.3$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3
-0.7	1.3	0.3	1	0	0.3	1	0	-0.3	-0.3	-0.3	0

$$\begin{aligned}
\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 &= -0.7 + 1.3 \cdot 1 + 0.3 \cdot 0 = 0.6 \\
y_1 = h(\text{net}_1) &= h(0.6) = 1 \\
e_1 = t - y_1 &= 0 - 1 = -1 \\
\eta \cdot e_1 = \eta(t - y_1) &= 0.3 \cdot -1 = -0.3 \\
w'_{10} = w_{10} + \Delta w_{10} &= -0.7 + 0.3 \cdot (-1) \cdot 1 = -0.7 - 0.3 = -1 \\
w'_{11} = w_{11} + \Delta w_{11} &= 1.3 + 0.3 \cdot (-1) \cdot 1 = 1.3 - 0.3 = 1 \\
w'_{12} = w_{12} + \Delta w_{12} &= 0.3 + 0.3 \cdot (-1) \cdot 0 = 0.3 + 0 = 0.3
\end{aligned}$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3
-0.7	1.3	0.3	1	0	0.3	1	0	-0.3	-0.3	-0.3	0
-1	1	0.3	0	0	0.3	0	0	0	0	0	0

$$\begin{aligned}
\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 &= -1 + 1 \cdot 0 + 0.3 \cdot 0 = -1 \\
y_1 = h(\text{net}_1) &= h(-1) = 0 \\
e_1 = t - y_1 &= 0 - 0 = 0 \\
\eta \cdot e_1 = \eta(t - y_1) &= 0.3 \cdot 0 = 0 \\
w'_{10} = w_{10} + \Delta w_{10} &= -1 + 0.3 \cdot 0 \cdot 1 = -1 + 0 = -1 \\
w'_{11} = w_{11} + \Delta w_{11} &= 1 + 0.3 \cdot 0 \cdot 0 = 1 + 0 = 1 \\
w'_{12} = w_{12} + \Delta w_{12} &= 0.3 + 0.3 \cdot 0 \cdot 0 = 0.3 + 0 = 0.3
\end{aligned}$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3
-0.7	1.3	0.3	1	0	0.3	1	0	-0.3	-0.3	-0.3	0
-1	1	0.3	0	0	0.3	0	0	0	0	0	0
-1	1	0.3	0	1	0.3	0	0	0	0	0	0

$$\begin{aligned}
\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 &= -1 + 1 \cdot 0 + 0.3 \cdot 1 = -0.7 \\
y_1 = h(\text{net}_1) &= h(-0.7) = 0 \\
e_1 = t - y_1 &= 0 - 0 = 0 \\
\eta \cdot e_1 = \eta(t - y_1) &= 0.3 \cdot 0 = 0 \\
w'_{10} = w_{10} + \Delta w_{10} &= -1 + 0.3 \cdot 0 \cdot 1 = -1 + 0 = -1 \\
w'_{11} = w_{11} + \Delta w_{11} &= 1 + 0.3 \cdot 0 \cdot 0 = 1 + 0 = 1 \\
w'_{12} = w_{12} + \Delta w_{12} &= 0.3 + 0.3 \cdot 0 \cdot 1 = 0.3 + 0 = 0.3
\end{aligned}$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t-y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3
-0.7	1.3	0.3	1	0	0.3	1	0	-0.3	-0.3	-0.3	0
-1	1	0.3	0	0	0.3	0	0	0	0	0	0
-1	1	0.3	0	1	0.3	0	0	0	0	0	0
-1	1	0.3	1	0	0.3	0	0	0	0	0	0

$$
\begin{aligned}
\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 &= -1 + 1 \cdot 1 + 0.3 \cdot 0 = 0 \\
y_1 = h(\text{net}_1) &= h(0) = 0 \\
e_1 = t - y_1 &= 0 - 0 = 0 \\
\eta \cdot e_1 = \eta(t - y_1) &= 0.3 \cdot 0 = 0 \\
w'_{10} = w_{10} + \Delta w_{10} &= -1 + 0.3 \cdot 0 \cdot 1 = -1 + 0 = -1 \\
w'_{11} = w_{11} + \Delta w_{11} &= 1 + 0.3 \cdot 0 \cdot 1 = 1 + 0 = 1 \\
w'_{12} = w_{12} + \Delta w_{12} &= 0.3 + 0.3 \cdot 0 \cdot 0 = 0.3 + 0 = 0.3
\end{aligned}
$$

w_{10}	w_{11}	w_{12}	x_1	x_2	η	y_1	t	$\eta(t - y_1)$	Δw_{10}	Δw_{11}	Δw_{12}
-1	1	0	1	1	0.3	0	1	0.3	0.3	0.3	0.3
-0.7	1.3	0.3	1	0	0.3	1	0	-0.3	-0.3	-0.3	0
-1	1	0.3	0	0	0.3	0	0	0	0	0	0
-1	1	0.3	0	1	0.3	0	0	0	0	0	0
-1	1	0.3	1	1	0.3	1	1	0	0	0	0

$$
\begin{aligned}
\text{net}_1 = w_{10} + w_{11}x_1 + w_{12}x_2 &= -1 + 1 \cdot 1 + 0.3 \cdot 1 = 0.3 \\
y_1 = h(\text{net}_1) &= h(0.3) = 1 \\
e_1 = t - y_1 &= 1 - 1 = 0 \\
\eta \cdot e_1 = \eta(t - y_1) &= 0.3 \cdot 0 = 0 \\
w'_{10} = w_{10} + \Delta w_{10} &= -1 + 0.3 \cdot 0 \cdot 1 = -1 + 0 = -1 \\
w'_{11} = w_{11} + \Delta w_{11} &= 1 + 0.3 \cdot 0 \cdot 1 = 1 + 0 = 1 \\
w'_{12} = w_{12} + \Delta w_{12} &= 0.3 + 0.3 \cdot 0 \cdot 1 = 0.3 + 0 = 0.3
\end{aligned}
$$

15.4.2 Updating Weights Using Vectors

Given the input vector $\vec{x}$ and the weight vector w_j for a given node j, we can write the output after the activation function h with n input nodes as follows

$$
y_j = h\left(\sum_{i=0}^{n} w_{ji}x_i\right) = h(\vec{w}_j \cdot \vec{x}) \tag{73}
$$

Therefore we can write the output as

$$
y_j = h(\vec{w}_j \cdot \vec{x}) = \begin{cases} 1 & \vec{w}_j \cdot \vec{x} \geq 0 \\ 0 & \vec{w}_j \cdot \vec{x} < 0 \end{cases} \tag{74}
$$

And the update of the weights $\vec{w}'_j$ is written as follows

$$
\vec{w}'_j = \vec{w}_j + \eta(t - y_j) \cdot \vec{x} \tag{75}
$$

15.5 Multi-Layer Feedforward Networks

A multi-layer network has an input layer, where no computation is done, one or more hidden layers where computation take place, and one output layer. The signals can only travel forward, so there is no feedback or loops. A notable example is the multilayer Perceptron.

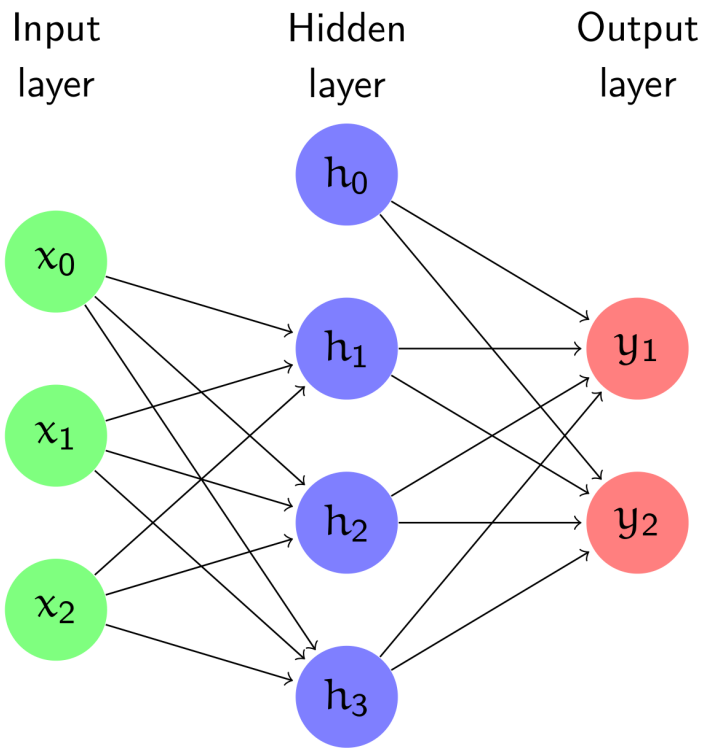


Figure 15.6: A fully connected multilayer feedworward network with one hidden layer

In figure 15.6 the bias neuron always has the index 0 and a set value of 1. We have two input neurons and a hidden layer with three neurons. The weights of the connection between input and hidden layer are denoted by $w_{ji}^{(1)}$ (from input nodes x_i to hidden layer nodes h_j) and the ones between hidden and output layer are denoted by $w_{kj}^{(2)}$ (from hidden nodes h_i to output nodes y_k). In all of this we would like to minimise the (L2-) error E of the training sample $(\vec{x}^{(\nu)}, \vec{t}^{(\nu)})$ with input $\vec{x}^{(\nu)}$ and output $\vec{t}^{(\nu)}$. N_O is the count of output nodes, N_I the count of input nodes and N_H the count of hidden nodes.

$$E\left(\vec{W}^{(1)}, \vec{W}^{(2)}\right) = \frac{1}{2}\sum_{k=1}^{N_O}\left(e_k^{(\nu)}\right)^2 = \frac{1}{2}\sum_{k=1}^{N_O}\left(t_k^{(\nu)} - y_k^{(\nu)}\right)^2 \tag{76}$$

15.5.1 Forward Sweep

1. Read input/target pair $(\vec{x}, \vec{t})$
2. For each hidden neuron $j = 1, 2, ..., N_H$ compute $h_j = \sigma\left(\sum_{i=0}^{N_I} w_{ji}^{(1)} x_i\right)$
3. For each output neuron $k = 1, 2, ..., N_O$ compute $y_k = \sigma\left(\sum_{j=0}^{N_H} w_{kj}^{(2)} h_j\right)$, the error $e_k = t_k - y_k$ and the resulting Delta $\delta_k^{(o)} = y_k(1 - y_k)e_k$

The forward propagation can be written in matrix notation:

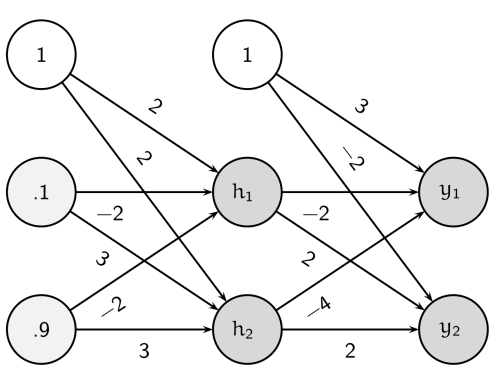
$$h_j = \sigma\left(\sum_{i=0}^{N_I} w_j i^{(1)} x_i\right) \Rightarrow \vec{h} = \sigma\left(\vec{W}^{(1)}\vec{x}\right)$$

$$y_k = \sigma\left(\sum_{j=0}^{N_H} w_k j^{(2)} h_j\right) \Rightarrow \vec{y} = \sigma\left(\vec{W}^{(2)}\vec{h}\right)$$

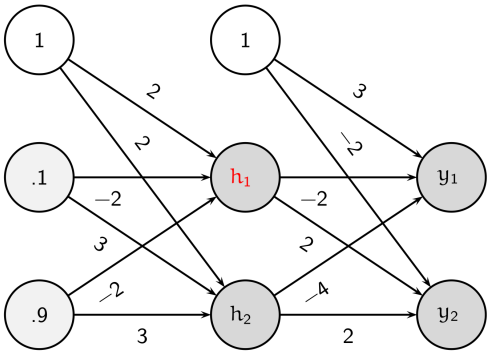
15.5.2 Backpropagation

1. Correct the weights $w_kj^{(2)}$ according to $\Delta w_kj^{(2)} = \eta\delta_k^{(o)}h_i$
2. For each hidden neuron $j = 1, 2..., N_H$ compute $\delta_j^{(h)} = h_i(1-h_i)\sum_{k=1}^{N_O}\delta_k^{(o)}w_{kj}^{(2)}$ where we sum over all successors k of the hidden neuron i
3. Correct the weights $w_{ji}^{(1)} according to \Delta w_{ji}^{(1)} = \eta\delta_j^{(h)}x_i$
4. If not converged or not all input pairs, return to Step 1 of the Forward Sweep

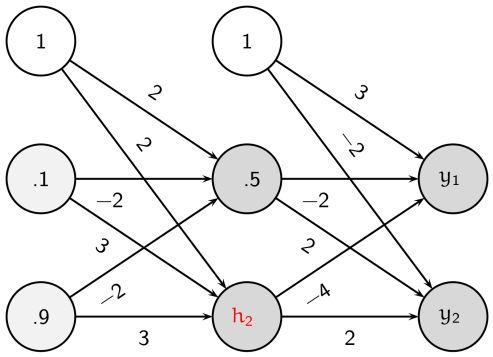
15.5.3 Example for the Calculation



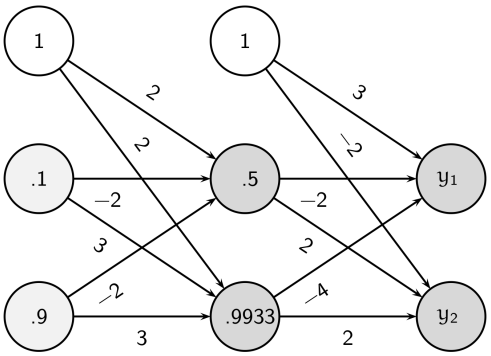
(a) Starting values



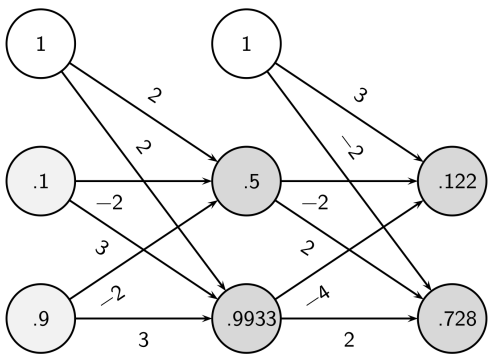
(b) $h_1 = \sigma(1 \cdot 2 + 0.1 \cdot (-2) + 0.9 \cdot (-2)) = \sigma(0) = 0.5$



(c) $h_1 = \sigma(1 \cdot 2 + 0.1 \cdot 3 + 0.9 \cdot 3) = \sigma(5) = 0.9933$



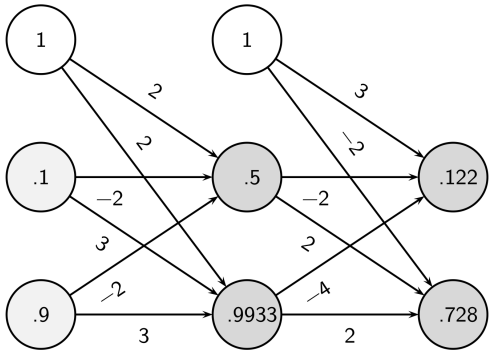
(d) All values for the hidden layer calculated



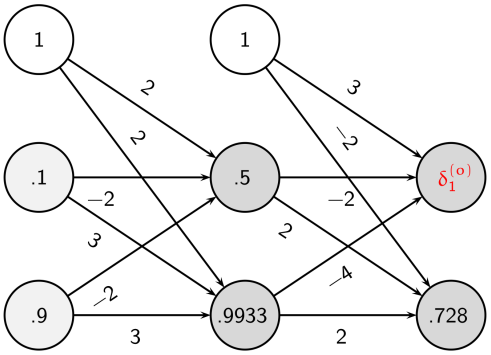
(e) Final weights

Figure 15.7: Calculation of the node values of a multilayer feedforward networks

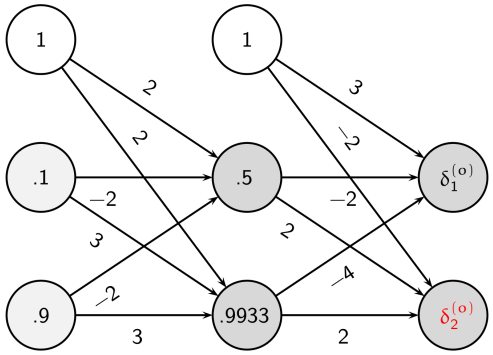
Backpropagation with target values $t_1 = 0$, $t_2 = 1$ and learning rate $\eta = 0.2$.



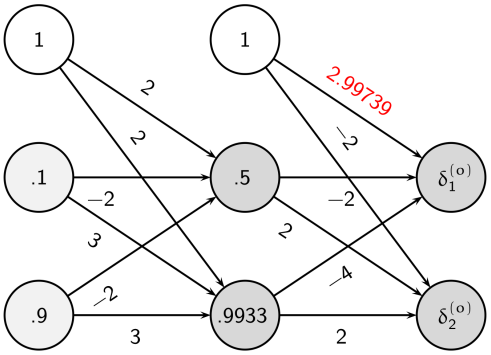
(a) Starting weights



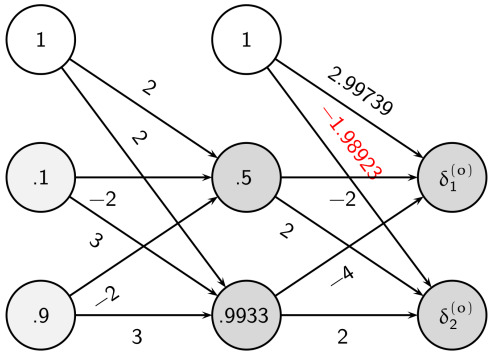
(b) $\delta_1^{(o)} = y_1(1-y_1)(t_1-y_1) = 0.122 \cdot (1 - 0.122) \cdot (0.0 - 0.122) = -0.01307$



(c) $\delta_2^{(o)} = y_2(1-y_2)(t_2-y_2) = 0.728 \cdot (1 - 0.728) \cdot (1.0 - 0.728) = 0.05386$



(d) $w_{10}^{(2)} = w_{10}^{(2)} + \eta\delta_1^{(o)}h_0 = 3 + 0.2 \cdot (-0.01307) \cdot 1 = 2.99739$



(e) $w_{20}^{(2)} = w_{20}^{(2)} + \eta\delta_2^{(o)}h_0 = -2 + 0.2 \cdot (0.05386) \cdot 1 = -1.98923$

Figure 15.8: Calculation of the weights of a multilayer feedforward networks

To calculate the delta for the neurons of the hidden layers, we sum over all successors of neuron j

$$\delta_j^{(h)} = h_j(1-h_j)\sum_{k=1}^{N_O} w_{kj}(2)\delta_k^{(o)} \tag{77}$$

Then we correct the weight of $w_{ji}^{(1)}$ according to the following formula, with η as the learning rate.

$$w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} + \Delta w_{ji}^{(1)} = w_{ji}^{(1)} + \eta\delta_j^{(h)}x_i \tag{78}$$