Machine Learning FS2019

Alex Neher

June 6, 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 **labeled training data** and learns to **predict** the **labels** of yet unseen examples.

Unsupervised Learning: The algorithm is given **unlabeled 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 labeled test data**

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

These classifications can be subdivided even more:

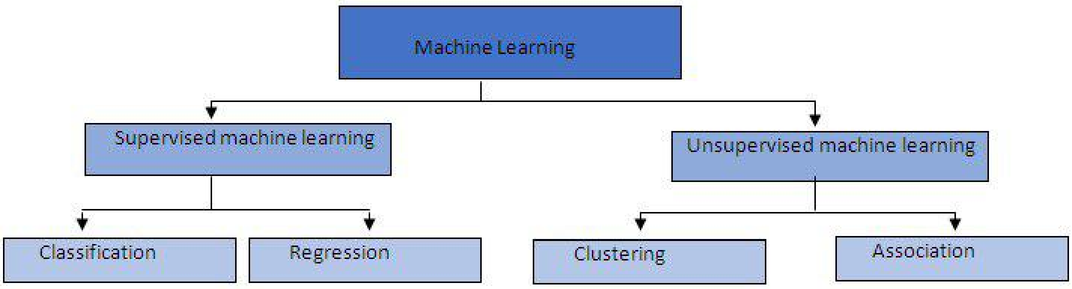


Figure 1.1: Distinction between supervised and unsupervised learning

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 classificiation. However, while classification algorithms sort the given data into given groups, clustering algorithms determine these groups **by themself**. 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.

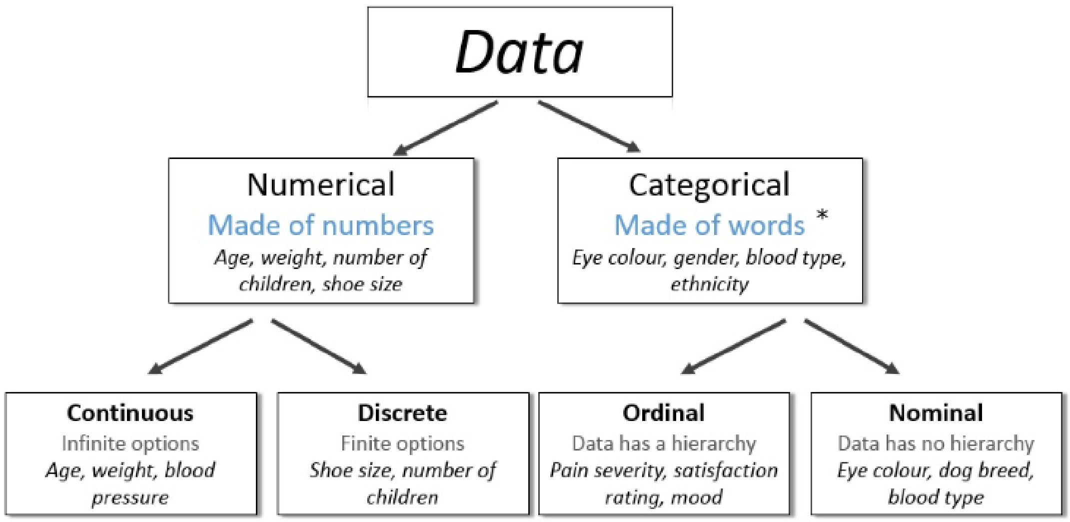


Figure 2.1: Classification of Data

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 seldomly 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 SVN (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 Abb. 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

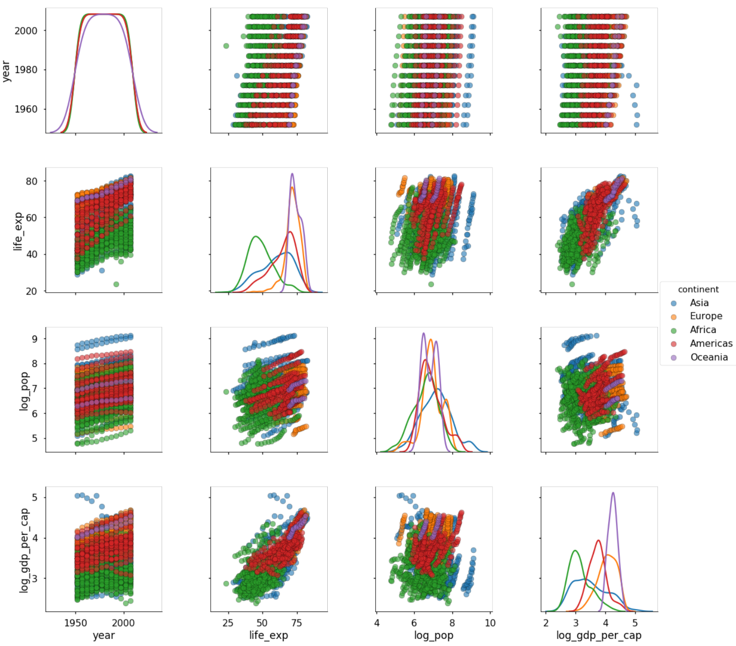


Figure 2.2: Visualisation of Data with Pair Plots

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

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

Mode

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

Median

This is the middlemost 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 mean is still 3

2.3.2 Skewdness

All of these values can give information concerning the datas **skewness**

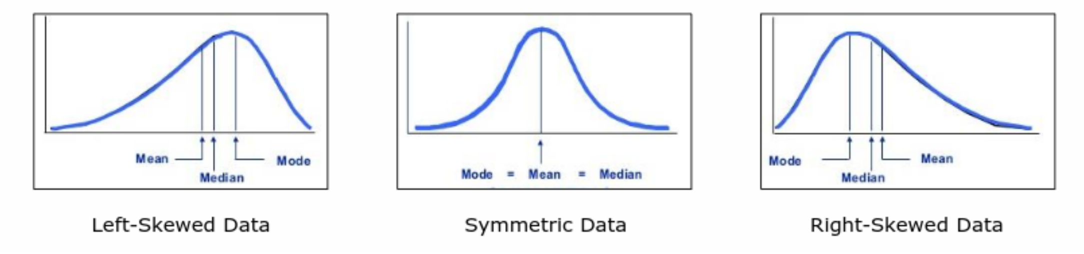


Figure 2.3: Skewness of data

$Mean - Mode > 0 \rightarrow$ 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, cnsecutive subsets.

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

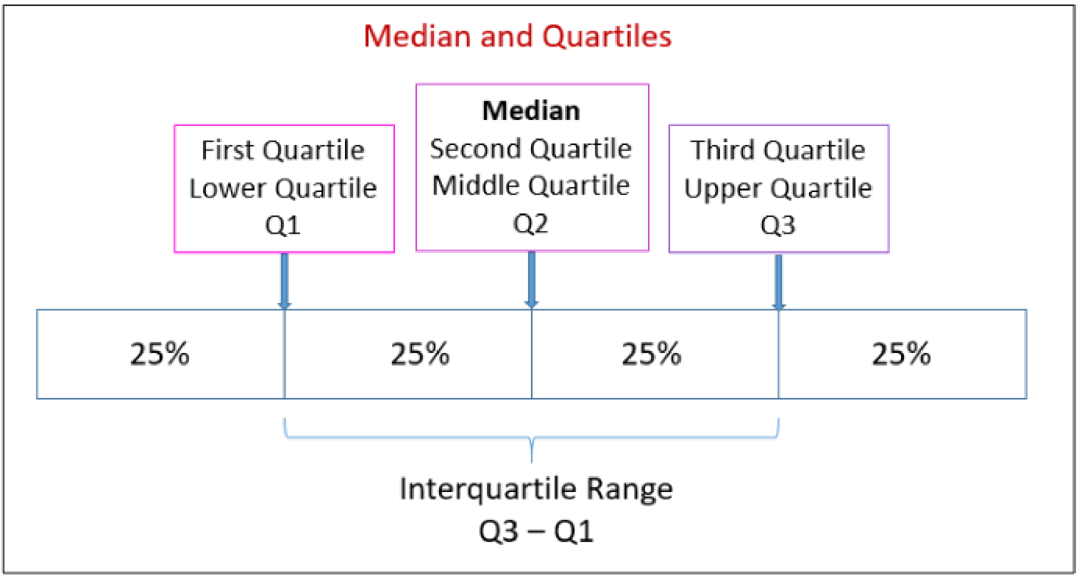


Figure 2.4: Quartiles of a dataset

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

```python
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.

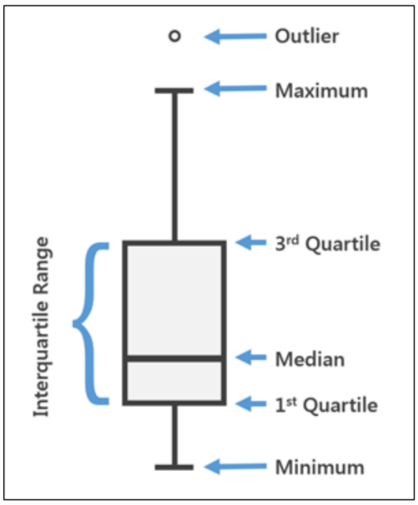


Figure 2.5: Boxplot

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

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

The standard deviation is calculated as $\sqrt{variance}$

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** from each other, the covariance is zero, as they both cancel each other out.

$$Cov(x, y)\frac{1}{1-n}\textstyle\sum_{i=1}^{n}(x_i - \mu x)(y_i - \mu y)$$

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 Perason Correlation is 1 in both examples.

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sigma_x \sigma_y}$$

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

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.

$$x \rightarrow \frac{x - min_x}{max_x - min_x}$$

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

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

3 Geometry of Data

3.1 Feature Engineering

Sometimes, data has to be modified to be better accessible/processable for machine learning algorithms. These algorithmus 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.

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

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

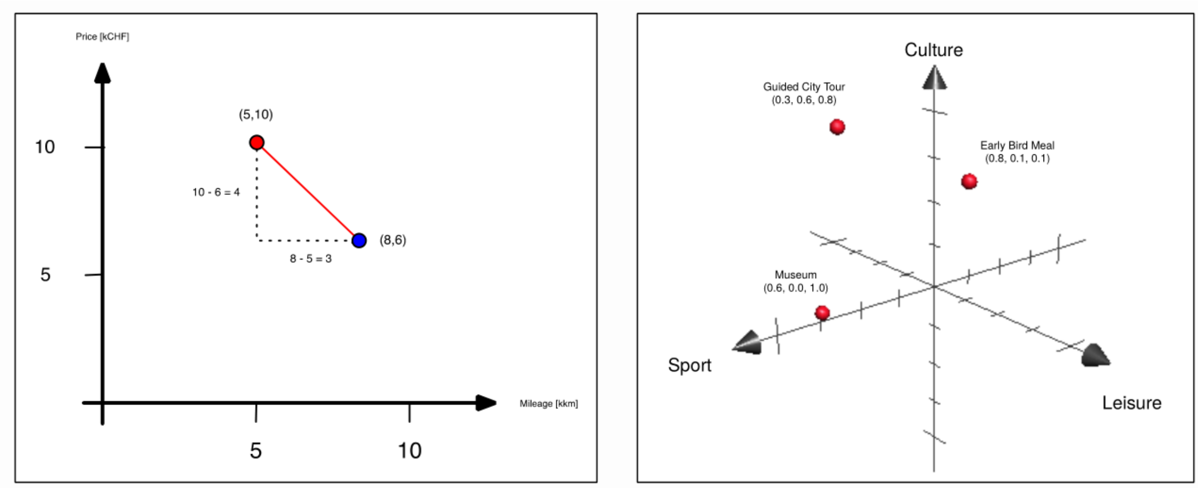


Figure 3.3: Transformed into vector space, data points can be interpreted as geometric points

3.3 Similarity of Data

The math 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**:

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

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

$$\begin{aligned} \sqrt{(5-8)^2 + (10-6)^2} \\ \sqrt{-3^2 + 4^2} \\ \sqrt{9+16} \\ \sqrt{25} = 5 \end{aligned}$$

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.

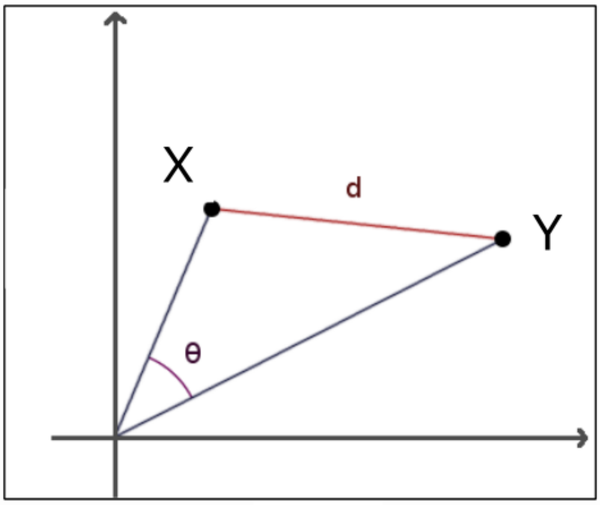


Figure 3.4: Cosine Similarity

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

$$dist(X,Y) = 1 - sim(X,Y)$$

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

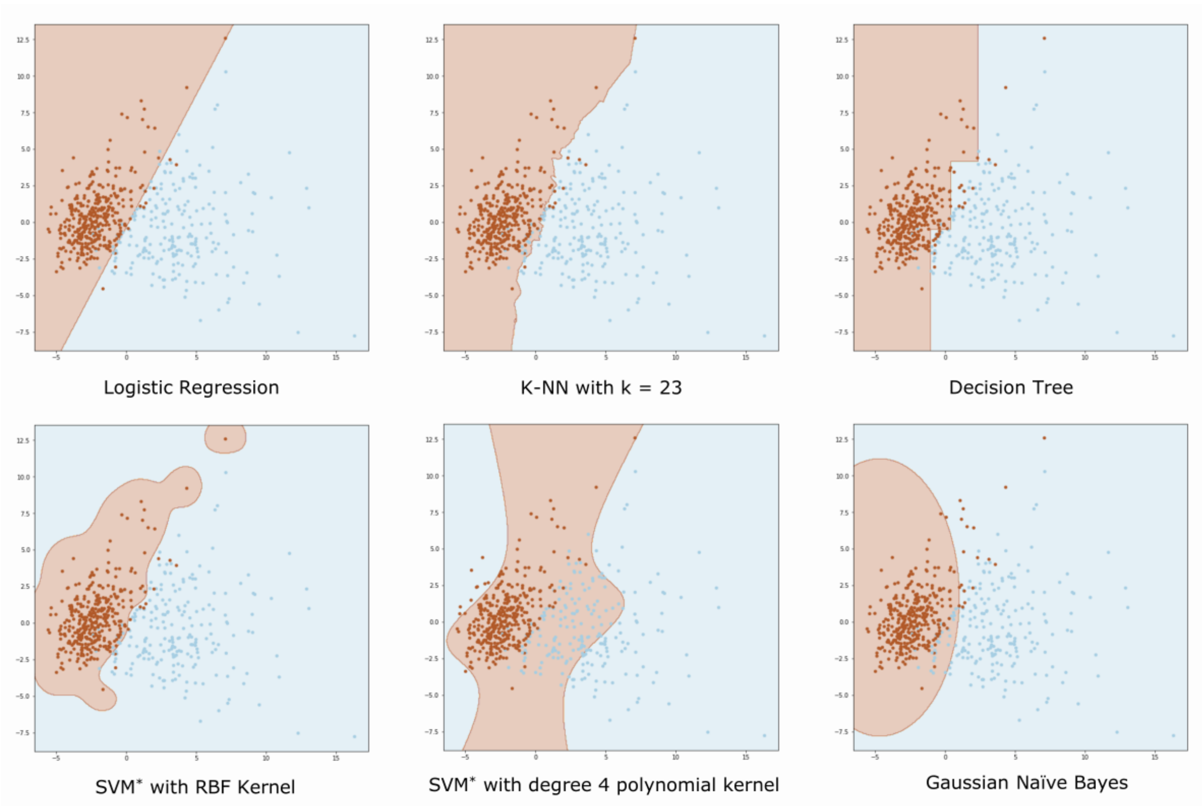


Figure 4.1: Decision Boundaries for different classification approaches

Classifications usually end in somithing like Figure 4.1. The example shows a classification whether a tumor 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-positive (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)

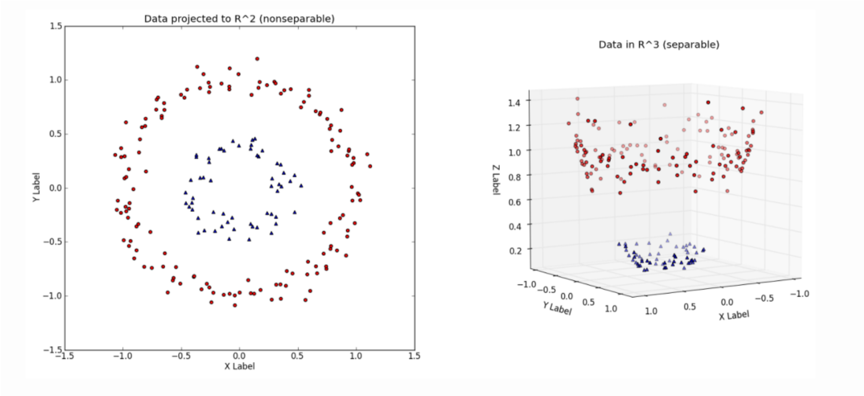


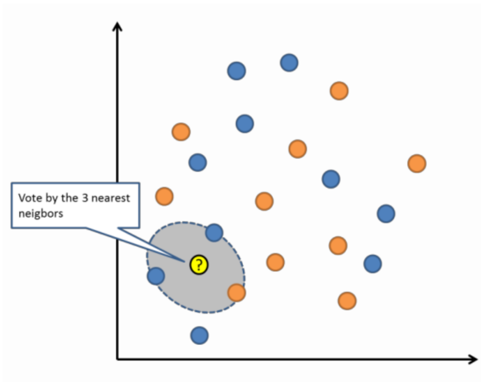
Figure 4.2: Kernel Trick to linearly separate data that is not linearly separable

4.3 k-Nearest-Neighbor

```python
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))
```



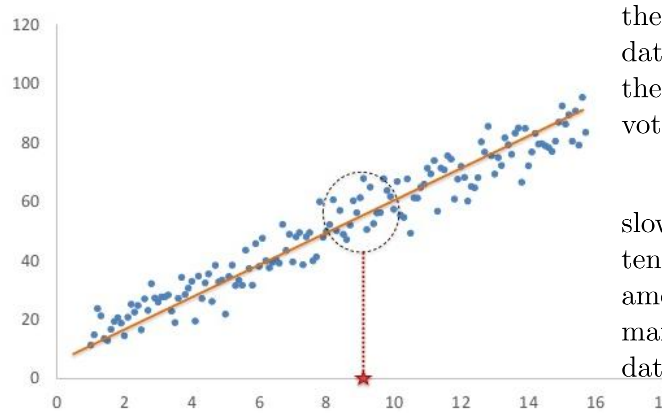


Figure 4.3: Regression with k-NN

The k-neeares-Neighbor algorithm assigns the label of its nearest datapoint to the sample datapoint. If `n_neighbors` 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 neighbors as equals. However, you could assign a weight to each datapoint. 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 neighbors as label to the sample data.

4.4 Training- and Testdata

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 **trainingdata** and **testdata**. You train the algorithm with the training data and test it with the testdata (who would've thought...).

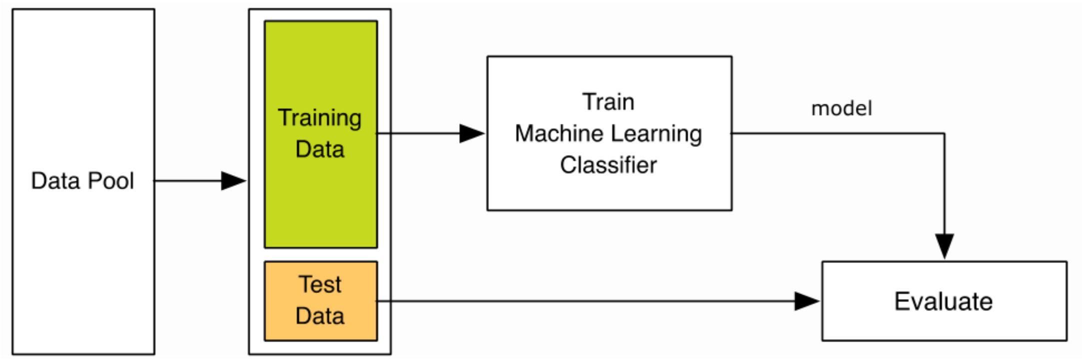


Figure 4.4: Split your data into training- and testdata

training means that you tweak the parameters of the algorithm to minimize 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 validatin data, and if the result is satisfactory, you can test it on entirely different test data.

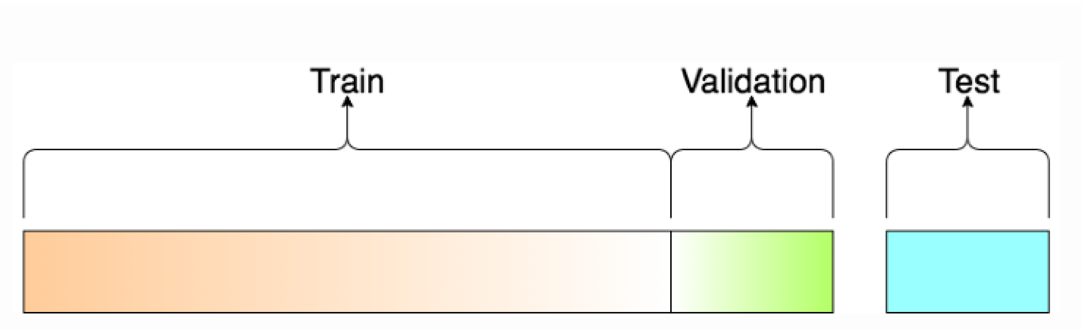


Figure 4.5: Add validation data to the mix

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.

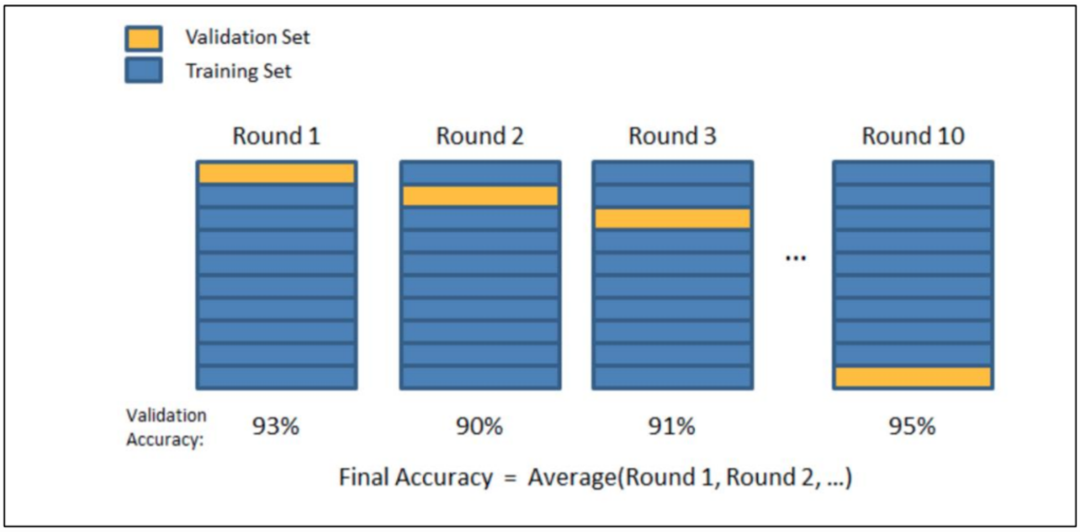


Figure 4.6: 10-fold cross validation

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 assuranc 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

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

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

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 inbalanced 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.

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

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

4.5.4 Sepcificity

This is the inverse of the Sensitivity. It counts how many NO the algorithm correctly predicted.o

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

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.

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

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

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

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

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

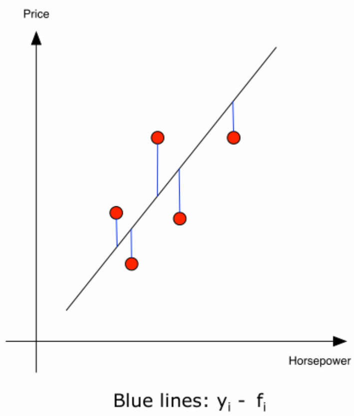
$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f_i)$$

- Mean Absolute Error

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

- Mean Squared Error

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



4.6.1 Coefficient of Determination

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

R^2 is a staticstical 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.

$R^3 = 1$ is a perfect prediction-line

$R^3 = 0.53$ means that 53% of the of the predictions are correct and can be explained by the model.