Machine Learning

`I.BA_ML` — Zusammenfassung

Author(s) Dominic, Elias, Hannah, Laura
Date 30. May. 2026
Pages 125

Inhaltsverzeichnis

1. Disciplines in Machine Learning 7
   1.1. Definitions 7
      1.1.1. Artificial Intelligence 7
      1.1.2. Machine Learning 7
      1.1.3. Deep Learning 7
      1.1.4. Generative AI 7
   1.2. Diciplines in ML 7
      1.2.1. Supervised Learning 7
      1.2.2. Unsupervised Learning 8
      1.2.3. Semi-Supervised Learning 8
      1.2.4. Reinforcement Learning 8
2. Data Quality Assessment 9
   2.1. Data Quality 9
      2.1.1. Reasons for Low Data Quality 9
      2.1.2. How to conduct a Data Quality Assessment 9
   2.2. Data Cleaning 9
      2.2.1. Replacement Strategies for Missing Values - NULL Values 10
      2.2.2. Feature engineering 10
      2.2.3. Vector Space Model 10
   2.3. Numerical Encoding of Text - TF-IDF Scores 10
   2.4. Advises on Data Quality Assessment (DQA) 11
   2.5. Data Quality Assessment with YData-Profiling 11
   2.6. Avoid the Dummy Variable Trap 11
   2.7. Central Tendency 11
   2.8. Data Skewness 12
   2.9. Quartiles & Interquartile Range (IQR) 12
   2.10. Five Number Summary 12
   2.11. Boxplots 13
   2.12. Measuring Data Spread 13
      2.12.1. Variance 13
      2.12.2. Standard deviation 13
   2.13. Bessel correction 13
   2.14. Covariance 13
   2.15. Covariance Matrix 14
   2.16. Pearson Correlation 14
   2.17. Correlation Analysis 14
   2.18. Control Questions 14
3. Similarity Measures 16
   3.1. Similarity in ML 16
      3.1.1. Data Records are geometric points 16
      3.1.2. A First Glimpse of Machine Learning 16
      3.1.3. Distance vs. Similarity 16
      3.1.4. Similarity in Context 17
   3.2. Similarity measures 17
      3.2.1. Euclidean Distance 17
      3.2.2. Cosine Similarity Intuition 17
      3.2.3. Manhattan Distance 18
      3.2.4. Levenshtein 19
   3.3. Jaccard Similarity for Sets 19
   3.4. Appendix 20
      3.4.1. Euclid vs. Cosine Similarity 20
      3.4.2. Haversine Distance for GEO Data 20

3.4.3. Mahalanobis Distance between Points und Disitributions 20
3.4.4. Mahalanobis vs. Euclidean Distance 20
4. K-Nearest Neighbors and Data Normalization 21
4.1. K-Nearest Neighbors 21
4.1.1. K-Nearest Neighbors Classification (k-NN) 21
4.1.2. Classification 21
4.1.3. K-Nearest Neighbors Regression 21
4.1.4. Hyperparameter 22
4.1.5. More Facts about K-Nearest Neighbors 22
4.1.6. Python manual 22
4.2. Importance of Normalization 23
4.2.1. Distance may be sensitive to Scale of Axes 23
4.2.2. Unity of Features 24
4.2.3. Min-Max Normalization 24
4.2.4. Z-Score Normalization 24
4.2.5. Normalization Parameters 25
4.2.6. Language Normalization 26
5. Supervised Learning 27
5.1. Regression Hypothesis and Evaluation Metrics 27
5.1.1. Regression Models 27
5.1.2. Regression Hypothesis 27
5.1.3. Regression Errors and Model Selection 28
5.1.4. Scikit-Learn Library 30
5.2. Evaluation Workflows 31
5.2.1. Motivation 31
5.2.2. Perfect Performance vs. Generalization 31
5.2.3. Simplistic Machine Learning Workflow 32
5.2.4. Hyperparameters 32
5.2.5. The Human Brain as Hidden Channel 33
5.2.6. Evaluation Workflow for Model Selection 33
5.2.7. K-Fold Cross-Validation 33
5.2.8. Model Selection with Pipelines 34
5.2.9. How to get Fired as a Data Scientist 34
6. Linear Regression 35
6.1. Overview 35
6.2. Terminology 35
6.3. Ordinary Least Squares (OLS) 35
6.3.1. Cost Function 36
6.4. Gradient Descent 36
6.5. Regression Performance R^2 37
6.6. Beyond Linear Regression 38
6.6.1. Multiple Linear Regression 38
6.6.2. Non Lineare Regression 38
6.6.3. Regularization 38
7. Classification 40
7.1. Decision Boundary & Logistic Regression 40
7.1.1. Find Linear Decision Boundary 40
7.1.2. Probability vs. Odds 40
7.1.3. Logits 41
7.1.4. From Linear to Logistic Regression 41
7.1.5. Sigmoid Function 41
7.1.6. Logistic Regression 42
7.1.7. Multiclass Classification with Logistic Regression 43
7.2. Classification Metrics 43

7.2.1. Linearly Separable Data ... 43
7.2.2. Confusion Matrix ... 44
7.2.3. Classification Threshold ... 47
8. Tree Models ... 51
8.1. Tree Construction Rules ... 51
8.2. Splitting criterion ... 51
8.2.1. Gini impurity ... 52
8.3. Regression Trees ... 55
8.4. Advantages of Tree Models ... 55
8.5. Disadvantages ... 55
8.6. Tree Models in Scikit Learn ... 55
8.7. Random Forest ... 56
8.8. Random Forests in Scikit Learn API ... 56
8.9. Ensembles ... 57
8.9.1. Bagging ... 57
8.9.2. Boosting ... 57
9. Neural Networks ... 58
9.1. Perceptron ... 58
9.1.1. Neurons and the Brain ... 58
9.1.2. Neurons Fire when Activated ... 58
9.1.3. MP-Neuron ... 59
9.1.4. Rosenblatt's Perceptron ... 60
9.1.5. Early Neural Networks and Deep Learning Timeline ... 60
9.1.6. Perceptron as a Logic Unit ... 60
9.1.7. A Simple Analogy ... 61
9.1.8. The Activation Function ... 61
9.1.9. Perceptron with Sigmoid ... 62
9.1.10. Linear vs. Non-Linear Models ... 63
9.1.11. Summary ... 63
9.2. Multilayer Perceptron (MLP) ... 64
9.2.1. Single Layer Neural Network ... 64
9.2.2. Hidden Layers ... 65
9.2.3. Implementation ... 67
9.2.4. XOR Function ... 68
9.2.5. Universal Approximators ... 68
9.2.6. Summary ... 68
9.3. Activation Functions ... 69
9.3.1. Choosing an Activation Function ... 69
9.3.2. Rectified Linear Unit (ReLu) ... 69
9.3.3. Leaky ReLU ... 70
9.3.4. Sigmoid σ ... 70
9.3.5. Tanh ... 70
9.3.6. Softmax for Multi-Label Classification ... 71
9.3.7. Recommendation ... 71
9.4. Cost Functions ... 71
9.4.1. Recommendation ... 71
10. Convolutional Neutral Nets (CNN) ... 72
10.1. Motivation ... 72
10.1.1. Building a Vanilla Neural Network ... 72
10.2. The Old Times: Manual Feature Engineering ... 72
10.2.1. Convolutional Neural Networks ... 72
10.3. Architecture ... 74
10.3.1. Convolution & Pooling ... 74
10.3.2. CNN Architecture ... 79

11. Recurrent Neural Nets (RNN) 83
11.1. Unsupervised Learning NLP 83
11.1.1. Syntax 83
11.1.2. Similarity 83
11.1.3. Word Relatedness 83
11.1.4. Term Frequency - Inverse Document Frequency (TD-IDF) 83
11.1.5. Continuous Bag of Words 84
11.1.6. Mathematics with Text 84
11.1.7. Implementation & Tools 84
11.2. Supervised NLP 84
11.2.1. Types 84
11.2.2. Recurrent Neural Networks - RNN 85
11.2.3. Gated Recurrent Units - GRU 85
11.2.4. Long Short-Term Memory - LSTM 86
11.2.5. Bidirectional RNNs 86
11.2.6. Encoder Decoder 86
11.2.7. Attention 87
12. Generative Models 1 - Transformers 89
12.1. Language Models 89
12.1.1. Markov Chains 89
12.1.2. Text Generation with RNNs 89
12.2. The Rise of the Transformers 89
12.2.1. Transformer Architecture 89
12.3. GPT: Decoder-Only Transformers 90
12.3.1. Encoding Input Tokens 91
12.3.2. Positional Encoding 91
12.3.3. Masked Self-Attention 91
12.3.4. Multi-Head Self-Attention 94
12.4. The Big Picture 94
12.5. Reinforcement Learning with Human Feedback 95
12.6. Training Resources 95
13. Transfer Learning 97
13.1. Models 97
13.2. Strategies 97
13.2.1. Strategy 1: Model Repurposing 97
13.2.2. Strategy 2: Fine Tuning 97
13.3. Impact on the industry 98
13.4. Issues with supervised learning 98
13.5. Unsupervised pre-training 98
13.5.1. Masked Language Modelling 98
13.5.2. Contrastive Learning Example 99
14. Clustering 100
14.1. k-Means Clustering 100
14.1.1. Preparation 100
14.1.2. Steps 100
14.1.3. Calculate Center of Cluster 100
14.1.4. Algorithm in Math 101
14.1.5. Clustering Distortion 101
14.1.6. Convergence and Optimality 101
14.1.7. Elbow Method — Choose Number of Clusters 102
14.2. Agglomerative Clustering 103
14.2.1. Motivation 103
14.2.2. Steps 103
14.2.3. Configurations / Hyperparameters 104

14.2.4. Example ... 104
14.2.5. Dendrogram ... 105
14.2.6. Elbow Method — Choose Number of Clusters ... 105
14.3. k-Means vs. Agglomerative Clustering ... 106
15. Association Rules 1 - Market Basket Analysis ... 107
15.1. Transaction Data ... 107
15.2. Association Rules ... 107
15.3. Support ... 108
15.3.1. Support of a Set of Items ... 108
15.3.2. Support of an Association Rule ... 108
15.3.3. Interpretation of Support ... 108
15.4. Confidence of an Association Rule ... 109
15.4.1. Interpretation of Confidence ... 109
15.5. Trustworthy but Uninteresting Rules ... 110
15.5.1. Issue ... 110
15.6. Lift of an Association Rule ... 110
15.6.1. Interpretation of Lift ... 111
15.7. Business ... 111
16. Association Rules 2 - Algorithms ... 112
16.1. Standard Association Analysis Procedure ... 112
16.2. Apriori Algorithm ... 113
16.2.1. Step 1: Frequent Item Set Generation ... 113
16.2.2. Step 2: Rule Generation from Frequent Item Sets ... 113
16.2.3. Example ... 113
16.3. Implementation ... 114
16.4. Appendix ... 115
16.4.1. Limitations of Apriori and Alternatives ... 115
16.4.2. FP-Tree Example ... 115
17. Recommender Systems ... 116
17.1. Non-Personalized Recommenders ... 116
17.1.1. Applications ... 116
17.1.2. Challenges ... 116
17.1.3. Scoring & Rankings ... 116
17.1.4. Associations ... 116
17.1.5. Shop-Specific Associations ... 116
17.1.6. Pros ... 116
17.1.7. Cons ... 117
17.1.8. Homogeneous vs. Heterogeneous Catalogs ... 117
17.1.9. Content-Based Recommenders ... 118
17.2. Personalized Recommender Systems ... 119
17.2.1. Mining User Preferences ... 119
17.2.2. Structured vs. Unstructured Product Data ... 119
17.2.3. Personalized Content-Based Recommendations ... 120
17.2.4. Collaborative Filtering ... 120
17.2.5. User-to-User Collaborative Filtering ... 122
17.2.6. Item-to-Item Collaborative Filtering ... 122
17.2.7. Matrix Factorization ... 124
17.2.8. The Netflix Challenge 2006 ... 124
17.2.9. Hybridization ... 124
17.2.10. Evaluation of Recommenders ... 124
17.2.11. API: Microsoft Recommender Collection ... 125

1. Disciplines in Machine Learning

1.1. Definitions

1.1.1. Artificial Intelligence

- Technical system that imitate human intelligence

1.1.2. Machine Learning

- AI that learns from experience (= data)
- more Data, better Model

1.1.3. Deep Learning

- Machine Learning models based on deep neural networks

1.1.4. Generative AI

- Generation of data like text, images, audio, video, tables, etc.
- deep learning models
- learns from data and generates data from it

1.2. Diciplines in ML

1.2.1. Supervised Learning

- The algorithm is given **labeled training data**
- The algorithm learns to **predict the label of yet unseen examples**

1.2.1.1. Regression

- Task: Learn to predict car prices from **labeled** training data
- The attribute to be **predicted is continuous** -> Regression Problem

Name	Mileage	Doors	Horsepower	Seats	Year	Price
CHEVROLET CRUZE 1.8 LTZ	5	5	141	5	2013	23410
LANCIA Thesis 2.4 JTD Emblema	180000	4	185	5	2007	8900
MERCEDES-BENZ C 250CDI Avantgarde 4M.	50	4	204	5	2013	54800
ALFA ROMEO Giulietta 2.0 JTDM Dist.	3000	5	140	5	2012	32900
TOYOTA Prius+1.8 VVT-i HSD Sol	18	5	99	7	2012	41730
PEUGEOT 807 2.0 16V Familiy+	62500	5	140	7	2007	18900
MERCEDES-BENZ A 190 Avantgarde	82000	5	125	5	2000	7900
RENAULT Clio 1.2 16V T Night&Day	50	5	103	5	2012	22800
VW Golf Plus 1.4 TSI Comfort	69200	5	140	5	2007	15500
MITSUBISHI Colt 1.3 16V Diamond	29000	5	95	5	2011	11990
PEUGEOT 207 1.6 16V Sport Pack	16900	3	120	5	2008	12700
TOYOTA Avensis 2.0 VVT-i Luna Automat	21000	5	152	5	2010	29800
VW Passat 2.0TDI BMT Comf.4M	20	5	177	5	2013	51870
VW Touran 2.0 TDI High	74500	5	140	7	2003	17900
NISSAN Juke 1.6 tekna	15	5	117	5	2013	24000
RENAULT Grand Espace 3.0dCi Initiale	134040	5	177	6	2004	13950
AUDI A8 2.8	180000	4	193	5	2002	6900
AUDI A6 allroad 2.5TDI quattro	229000	5	180	5	2001	11900
RENAULT Laguna 2.0 16V Expression	67500	5	133	5	2006	9900
NISSAN Primera 2.0 acenta Automat	89000	5	140	5	2005	9900

Labels

1.2.1.2. Classification

- Transaction data of a single customer
- Task: Learn to predict shopping behavior based on **labeled** training data
- The attribute to be predicted is **categorical** (binary) -> Classification Problem

Weather	Store Size	Saturday	Price Reduction	Season	Days since last Purchase	Product purchased
Rain	M	Yes	50%	Spring	10	Yes
Sun	MMM	Yes	20%	Summer	30	No
Cloudy	MM	No	50%	Autumn	10	Yes
Cloudy	M	Yes	20%	Summer	20	Yes
Rain	M	No	20%	Spring	60	No
Rain	M	Yes	50%	Winter	10	Yes
Sun	MMM	No	0%	Autumn	10	No
Sun	M	Yes	50%	Summer	10	Yes
Rain	MM	No	20%	Autumn	60	No
Rain	MM	Yes	20%	Winter	30	No
Rain	M	No	0%	Summer	10	No
Rain	M	Yes	20%	Autumn	60	Yes

Labels

1.2.2. Unsupervised Learning

- The algorithm is given **unlabeled data**
- The algorithm **detects and exploits the inherent structure** of the data

1.2.2.1. Dimensionality Reduction

- reduces the number of input variables in a dataset
- transforming high-dimensional data into a lower-dimensional form while retaining essential information

1.2.2.2. Recommender Systems

- predicts which content (products, movies, music) a user is most likely to want to consume
- goal is to filter out the most relevant hits from a huge amount of data.

1.2.3. Semi-Supervised Learning

- A **mixture** of supervised and unsupervised machine learning techniques
- Usually there is only very limited labeled data available
- Generative AI models are trained in a semi-supervised manner (e.g., Chat-GPT)
  - Probability of the next word

1.2.4. Reinforcement Learning

- The algorithm is guided by a **reward function**
- It searches the ideal behavior that maximizes the agent's reward

2. Data Quality Assessment

2.1. Data Quality

2.1.1. Reasons for Low Data Quality

- Ill-designed, inadequate or inconsistent data formats
- Programming errors (e.g., erroneous error handling)
- Technical issues (e.g., sensor outage)
- Data decay (e.g., outdated e-mail addresses)
- Poorly designed data entry forms (e.g., no input verification)
- Human errors in data export or data pre-processing
- Conversion issues (e.g., decimal format, time zones)
- Deliberate errors and false information (e.g., privacy concerns)

2.1.2. How to conduct a Data Quality Assessment

- Identify data sources and their trustworthiness
- Interpret statistical key figures (see stats refresher)
- Visualize selected portions of the data (e.g., pair plots, correlation)
- Manually check data ranges (e.g., negative salaries or people more than 125 years old)
- Validate plausibility of variable correlations (e.g., correlation between mileage and seats)
- Measure data redundancy (Principal Component Analysis will be explained in later parts)
- Check for anomalies in syntax and semantics
- Explore NULL values and duplicate data records

2.2. Data Cleaning

sometimes data quality can be improved by data cleaning, e.g.,

- duplicate data records can be identified and removed
- null values can be replaced
- Standardize date formats; time zones, currencies, …
- …

never forget to …

- document all changes to the original data
- use a data repository with versioning
- let the data provider know about data quality issues
- investigate the origins of data quality issues

2.2.1. Replacement Strategies for Missing Values - NULL Values

most algorithms cant cope with NULL values

strategy:

- delete rows containing NULL values
  - (e.g., in case you have a lot of data)
- fill in the missing values manually
  - (e.g., take missing weather data from other sources)
- use a measure for central tendency
  - (e.g., mean for symmetric data, median for skewed data)
  - symmetric -> mean
  - skewed -> median
- use a measure for central tendency per class
  - (e.g., different value for healthy and sick people)

2.2.2. Feature engineering

- Makes information accessible for machine learning algorithms
- transforms categorical data into numerical data
- eg. Date -> Hour, Minute, Year, Month, Day

2.2.3. Vector Space Model

- A data set consisting of **numerical data only** is called vector space model
- **Most** machine learning algorithm **expect numerical data** as input
- Feature Engineering
- Delete one dummy variable to avoid the dummy variable trap

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

2.2.3.1. Delete Dummy Variables

for example „weiss“ can be deleted, because when its not „braun“, „gelb“, etc. then it has to be „weiss“

2.3. Numerical Encoding of Text - TF-IDF Scores

- turn human language into numbers
- Term Frequency-Inverse Document Frequency

Goal: Convert text into a format (vectors) that machines can understand and analyze **Vectorization:** Every word is represented as a list of numbers rather than letters.

2.4. Advises on Data Quality Assessment (DQA)

- Never start a project without a DQA
- DQA is the first bullet point in every data science offer
- Reject projects if customers do not want to pay for DQA
- Depending on the data set, pre-define how much time to invest
- DQA is detective work, if something look suspicious, dig deeper
- Repeat DQA with every data set version issued
- Provide a data quality report to your customer

2.5. Data Quality Assessment with YData-Profiling

https://github.com/ydataai/ydata-profiling

2.6. Avoid the Dummy Variable Trap

- Transforming categorical data into dummy variables introduces multicollinearity
- We can predict the nth dummy from the other n-1 dummy variables
  - Solution: simply delete one of the dummy variables
- Multicollinearity among features can be an issue for some learning algorithms
- Linear regression is such an example:
  - Solving linear regression problems analytically requires to invert a matrix
    - Multicollinearity leads to low rank matrices with no inverse
- Solving linear regression problems numerically is done with gradient descent
  - Gradient descent convergence may be very slow on such data
- Other algorithms are immune to multicollinearity, e.g., decision trees
- Principal component analysis can be used to remove multicollinearity from data

2.7. Central Tendency

The **mean** is the average of a set of numeric observations

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

The **mode** is the value that occurs the most

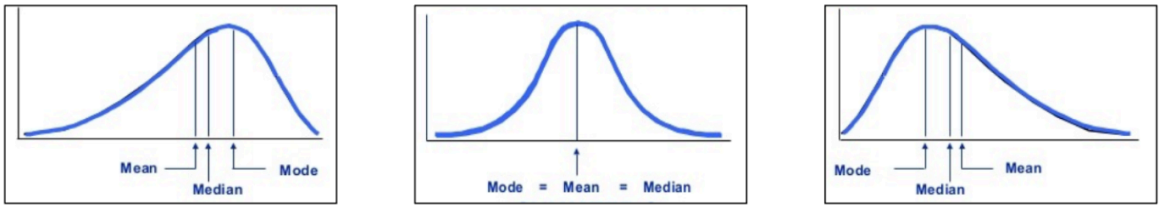
The **median** is the middlemost value of a sorted set of numeric observations

Keep in mind:

- *Mean is efficient to compute, the median is rather expensive*
- *Mean is very sensitive to outliers, median is not*

2.8. Data Skewness

Mean, median and mode give valuable information on data skewness

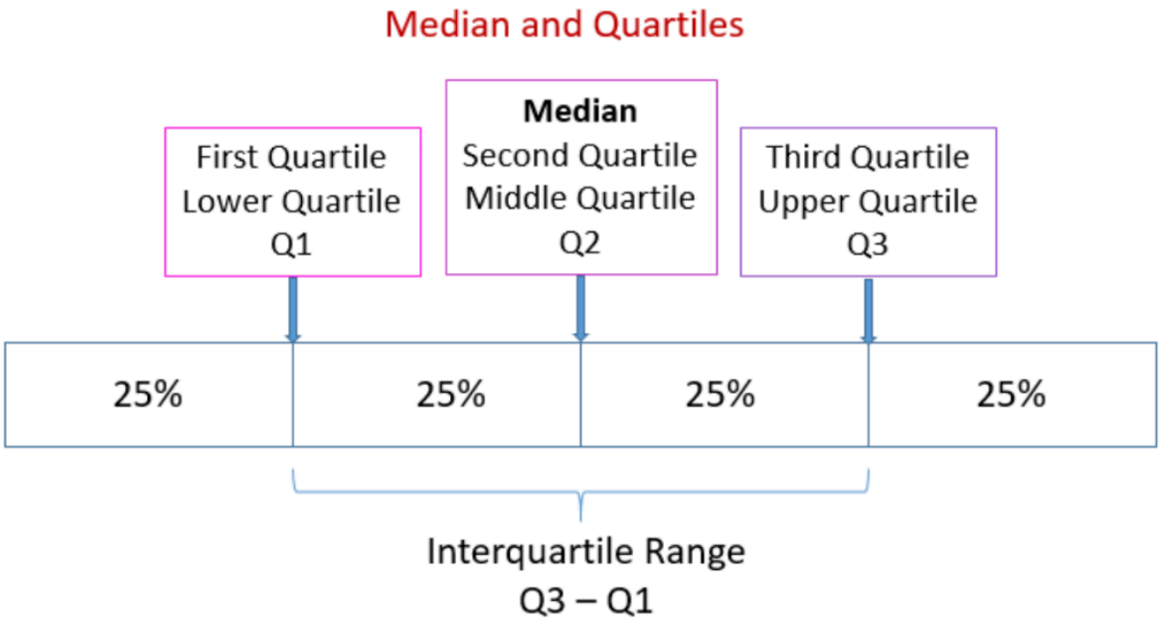


Left-Skewed Data | Symmetric Data | Right-Skewed Data

- left-skewed data = mean − mode < 0
- right-skewed data = mean − mode > 0

important: in ML skewness has to be counteracted

2.9. Quartiles & Interquartile Range (IQR)



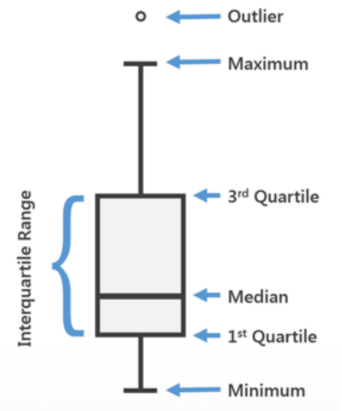
$$\text{IQR} = \text{Q3} - \text{Q1}$$

2.10. Five Number Summary

- median Q2
- quartiles Q1 & Q3
- smallest and largest value

```python
import numpy as np
import pandas as pd

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

2.11. Boxplots



Upper Outlier: values at least $1.5 * \text{IQR}$ above the 3rd quartile

Lower Outlier: values at least $1.5 * \text{IQR}$ above the 1rd quartile

- Outliers are displayed as small circles above or below the boxplot

Minimum is the lowest value still within $1.5 * \text{IQR}$

Maximum is the highest value still within $1.5 * \text{IQR}$

```python
import matplotlib.pyplot as plt

plt.boxplot(x = [data.Mileage, data.Price], labels= ['Mileage', 'Price'])
```

2.12. Measuring Data Spread

2.12.1. Variance

- how much the values spread out in average

$$\text{Var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)^2$$

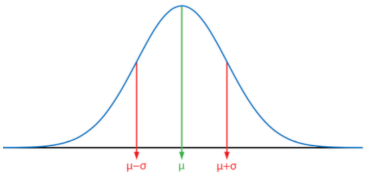
- Division by $n-1$ (instead of n) is due to **Bessel correction**
- easier for futher calculations

2.12.2. Standard deviation

- preserves the units of the data (cm^2 -> cm)

$$\sigma = \sqrt{\text{Var}(X)}$$

2.13. Bessel correction



- sample under-estimates the true spread of the data
- To compensate for this effect, **sample variance divides by $n-1$ instead of n**

2.14. Covariance

Sample **covariance** is the joint variability of two equally sized distributions

$$\text{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y)$$

- lower, lower -> negative x negative -> positive
- bigger, bigger -> positiv x positiv -> positive
- lower, bigger -> negative x positiv -> negative

2.15. Covariance Matrix

A covariance matrix displays the pair-wise covariance among all variables

Since $\text{Cov}(X,X) = \text{Var}(X)$ the matrix looks like this:

$$\begin{bmatrix} Var(X_1) & Cov(X_1,X_2) & Cov(X_1,X_3) & \dots & Cov(X_1,X_n) \\ Cov(X_2,X_1) & Var(X_2) & Cov(X_2,X_3) & \dots & Cov(X_2,X_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Cov(X_n,X_1) & Cov(X_n,X_2) & Cov(X_n,X_3) & \dots & Var(X_n) \end{bmatrix}$$

2.16. Pearson Correlation

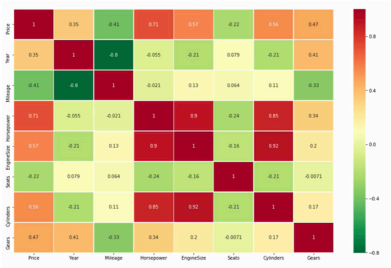
- The difficulty with covariance is that its scale depends on the data scale

$$\rho(X,Y) = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

- normalized range `[-1, 1]`
  - `-1` = perfect anti-correlation
  - `+1` = means perfect correlation
- This enables comparison of correlation between pairs of variables

2.17. Correlation Analysis

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('cars.csv')
data = data.loc[:,['Price', 'Year', … 'Cylinders', 'Gears']]

plt.subplots(figsize=(10, 8))
sns.heatmap(data.corr(), annot=**True**, cmap='RdYlGn_r', linewidths=0.5)

# Replace data.corr() by data.cov() to display the covariance matrix
```



2.18. Control Questions

>What are the respective advantages and disadvantages of mean and median?

>When is a data point considered an outlier?

>What is the advantage of standard deviation over variance?

- standard deviation is easier to interpret
- variance is easier to calculate

>What is the difference between population and sample variance?

- population is everything (N)
- sample is $n - 1$

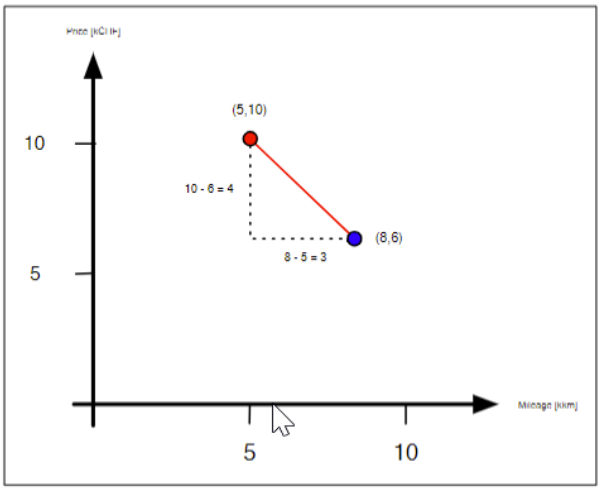
>What is the advantage of Pearson correlation over co-variance?

- Person is normalised (-1,1)
  - easier to compare
- co-variance is depending on data

3. Similarity Measures

3.1. Similarity in ML

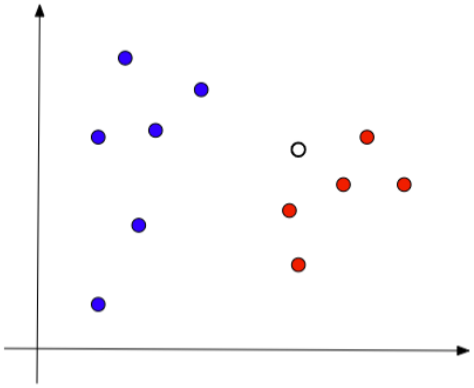
3.1.1. Data Records are geometric points



two cars (red and blue dots)

- the closer the cars are, the more similar they are
- the further apart the cars are, the more different they are

3.1.2. A First Glimpse of Machine Learning



- Training data: colored data points with categories red and blue
- Test data: single (uncolored) point without category
  - assign to category red (because it is nearer to the red category)

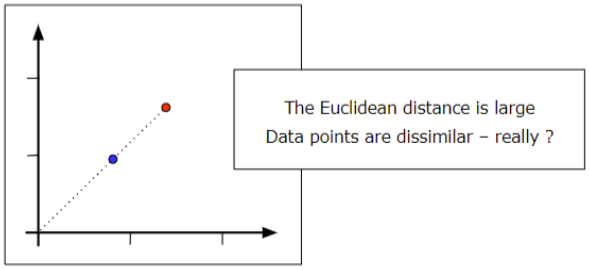
3.1.3. Distance vs. Similarity

Many machine learning algorithms are based on the following assumption:

The closer two data points in the geometric space, the more similar they are

Example:
- online dating
- car reselling platform
- Amazone recommend system

3.1.4. Similarity in Context



- dissimilar for example for a reselling platform
  - A car with twice the horsepower and price is indeed very different
- unreasonable in natural language processing
  - An article with twice as many occurrences of the same words should not be different (copy-paste)

Conclusion: select a similarity measure meaningful in your context.

3.2. Similarity measures

3.2.1. Euclidean Distance

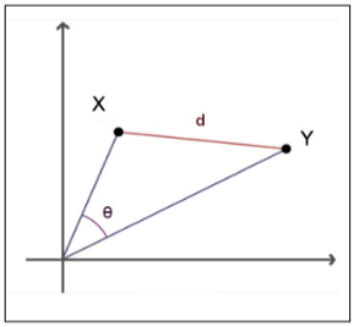
- generalization of Pythagoras formula to an arbitrary number of dimensions
  - calculates the direct length
- also called L^2 norm
- range `[0, ∞ [`
  - shortest distance = 0
- find the most similar data point by minimizing distance

$$\begin{aligned}\text{euclid}(X, Y) &= \|X - Y\|_2 \\ &= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \\ &= \sqrt{\sum_{i=1}^{n} x_i^2 - 2x_i y_i + y_i^2}\end{aligned}$$

```python
def euclidean_distance(v1, v2):
    v1, v2 = np.array(v1), np.array(v2)
    return np.linalg.norm(v1-v2)
```

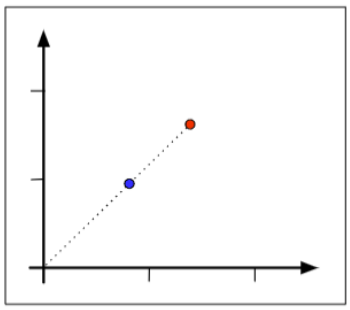
3.2.2. Cosine Similarity Intuition

- angle θ between two vectors
- angel = 0 -> perfect similar
- angel = 1 -> not similar



- **Euclidean distance**: length d

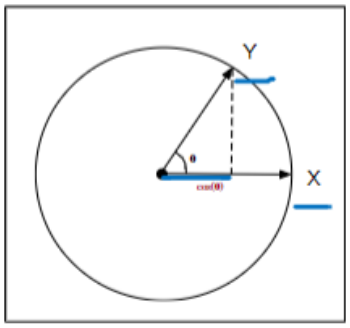
- **cosine similarity**: angle between two vectors



- blue and red have cosine similarity 1
- but low Euclidean similarity

$$\cos(X,Y) = \frac{\langle X,Y\rangle}{\|X\|_2 \cdot \|Y\|_2} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

- range is `[-1,1]`
- but takes absolute value when negative coordinates exist
  - angel = 0°: vectors point in exactly the same direction. The similarity is **1**.
  - angel = 90°: vectors are orthogonal (independent). The similarity is **0**.
  - angel = 180°: vectors are exactly opposite. The similarity is **–1** → **1**.




```python
def cosine_distance(v1, v2):
    v1, v2 = np.array(v1), np.array(v2)
    cos = np.sum(v1 * v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
    return cos
```

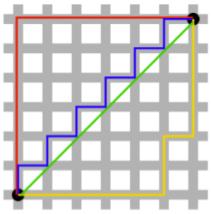
3.2.2.1. cosine distance

$$1 - \text{cosine similarity}$$

```python
def cosine_distance(v1, v2):
    v1, v2 = np.array(v1), np.array(v2)
    cos = np.sum(v1 * v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
    cos_dis = 1 - cos
    return cos_dis
```

3.2.3. Manhattan Distance

$$\text{manhattan}(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$$



- L1 norm
- red, blue and yellow are equivalent
- take the route into account
- range is $[0, \infty[$

3.2.4. 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 $[c]$
    - (sometimes only +1)

Note: implementing the Levenshtein distance efficiently is VERY difficult!

3.2.4.1. Example

- change first word into second word

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

3.3. Jaccard Similarity for Sets

oben: X und Y -> intersection (Schnittmenge)

unten: X mit Y -> everything (X and Y together)

$$\text{jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- The range of the Jaccard similarity is `[0, 1]`
  - 0: no similarity
  - 1: perfect similarity
- Possible applications:
  - Two shopping charts are similar if they contain the same items
  - Text paragraphs are similar if the use the same jargon / stems

```python
def jaccard_similarity(list1, list2):
    a = set(list1)
    b = set(list2)
    intersection = a.intersection(b)
    union = a.union(b)
```

```
    jacsim = len(intersection) / len(union)
    return jacsim
```

3.4. Appendix

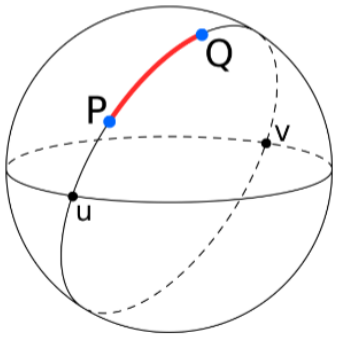
3.4.1. Euclid vs. Cosine Similarity

- On arbitrary data the Euclidean and cosine similarity are unrelated
- Hence, ranking data points with respect to similarity produces different results
- On L2 normalized data cosine similarity is proportional to the squared Euclid distance
- A ranking of L2 normalized data with respect to similarity produces the same result

$$\begin{aligned}\|X - Y\|_2^2 &= \sum_{i=1}^{n} (x_i - y_i)^2 \\ &= \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} y_i^2 = 1 - 2\sum_{i=1}^{n} x_i y_i + 1 \\ &= 2 - 2 \cdot \langle X, Y \rangle = 2 - 2\cos(X, Y)\end{aligned}$$

3.4.2. Haversine Distance for GEO Data

Haversine distance measures geographic distance on a sphere



3.4.3. Mahalanobis Distance between Points und Disitributions

1. Transform the columns of the distribution into uncorrelated variables
2. Scale the columns of the distribution to make their variance equal to 1
3. Calculate the Euclidean distance between the point X and the distribution center

- Measures how many standard deviations X is away from the mean
- Describing the distribution by its mean and covariance matrix Σ

$$\text{mahalanobis}(X, [\mu, \Sigma]) = \sqrt{(X - \mu)^T \Sigma^{-1} (X - \mu)}$$

3.4.4. Mahalanobis vs. Euclidean Distance

If the covariance matrix equals the identity matrix

$$\begin{aligned}\text{mahalanobis}(X, [\mu, \Sigma]) &= \sqrt{(X - \mu)^T \Sigma^{-1} (X - \mu)} \\ &= \sqrt{(X - \mu)^T \boldsymbol{I} (X - \mu)} \\ &= \sqrt{(X - \mu)^T (X - \mu)} \\ &= \|X - \mu\|_2 = \text{euclid}(X, \mu)\end{aligned}$$

- Mahalanobis thus reduces to the Euclidean distance between x and the mean
- The covariance matrix equals the identity when features are perfectly independent

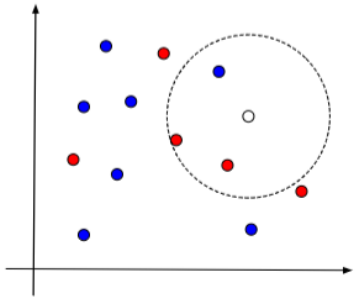
4. K-Nearest Neighbors and Data Normalization

4.1. K-Nearest Neighbors

4.1.1. K-Nearest Neighbors Classification (k-NN)

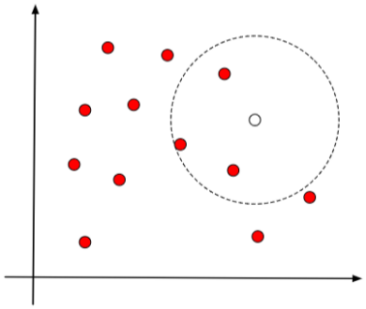
- simplest ML Algorithm
- Data in Vector Space Model with suitable distance measure
- k = Number of Neighbors
- vote can optionally be weighted by 1/d on its distance
  - nearer -> higher value
- sensitive to the scale of the data -> normalize

4.1.2. Classification



- k = 3
- majority voting
- two red, one blue -> new point is red

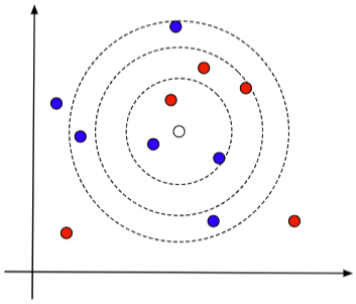
4.1.3. K-Nearest Neighbors Regression



- average of the neighbors

4.1.4. Hyperparameter

- The performance of k-NN depends greatly on the choice of k
- Such manually selected parameters of ML models are called **hyperparameters**



k-NN classification with k = 3, k = 5, k = 8 gives different results

4.1.5. More Facts about K-Nearest Neighbors

- k-NN is very **slow** because all computation is deferred to classification time
- It is the **least data hungry** algorithm; a good choice especially for small data sets
- Also used with very modern ML schemes such as few-shot learning
- Average distance to k-nearest neighbors can be seen as a **local density estimate**

4.1.6. Python manual

4.1.6.1. distance measure

PYTHON

```python
def get_nearest_neighbor(source_car, cars, distance_fn):
    min_distance = float('inf')
    min_idx = -1

    for idx, car in enumerate(cars):
        # Calculate the distance using the provided distance function
        dist = distance_fn(source_car, car)

        # Update the minimum distance and index if a closer car is found
        if dist < min_distance:
            min_distance = dist
            min_idx = idx

    return min_distance, np.int64(min_idx)
```

4.1.6.2. Scikit-Learn

PYTHON

```python
#train
knn = NearestNeighbors(n_neighbors=1, metric="euclidean")
knn.fit(X_train_transform)

#caluclate
distance, idx = knn.kneighbors(X_car0_transform)
```

4.1.6.3. similarity measure

PYTHON

```python
def nearest_neighbor(source_wine, wines):
    idx = -1
    similarity = -1

    for i, wine in enumerate(wines):
```

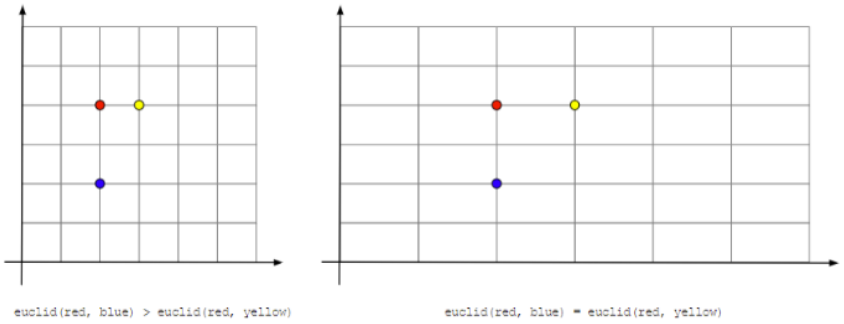
```
        sim = jaccard_similarity(source_wine, wine)

        if sim > similarity:
            similarity = sim
            idx = i
    return idx, similarity
```

4.1.6.3.1. easier solution

```python
def nearest_neighbor(wine, wines):
    similarities = np.array([jaccard_similarity(wine, w) for w in wines])
    similarity = np.max(similarities)
    idx = np.argmax(similarities)
    return idx, similarity
```

4.2. Importance of Normalization

4.2.1. Distance may be sensitive to Scale of Axes



- x-axis is multiplied by a factor of 2

4.2.2. Unity of Features

Each car is displayed with its most similar neighbor using the Euclidean distance

	Price	Seats
0	44800	2
1	22800	5
2	183710	5
3	19900	7

- Car 0 is most similar to car 1
- Car 1 is most similar to car 3
- Car 2 is most similar to car 0
- Car 3 is most similar to car 1

- price dominates the Seats
- Price can go up to 2 Mio. but Seats only to 20 or so

The next table contains the same data normalized to $[0, 1]$

	Price	Seats
0	0.15	0
1	0.02	0.6
2	1	0.6
3	0	1

- Car 0 is most similar to car 1
- Car 1 is most similar to car 3
- Car 2 is most similar to car 1
- Car 3 is most similar to car 1

Car 2 has different matches because price dominates seats in the first table.

4.2.2.1. Dominance among Features

- The scale in each dimension depends on the data:
  - the largest price in the Autoscout24 dataset is 698'000 CHF
  - the largest number of seats is 20
- Solution
  - Preprocess your data and bring each attribute to approx. the same scale
  - We introduce two strategies for supervised and unsupervised learning

4.2.3. Min-Max Normalization

- Transforms your data to the unit interval `[0,1]`
- largest data value becomes 1
- smallest data value becomes 0

$$x \mapsto \frac{x - \min_X}{\max_X - \min_X}$$

Advantage:
- allows interpretation in %
- no negative values -> e.g., cosine similarity >= 0

Disadvantage
- cannot be used for supervised learning (min / max unknown)

4.2.3.1. sklearn.preprocessing

```python
scaler = MinMaxScaler()

X_train_transform = scaler.fit_transform(X_train)
X_car0_transform = scaler.transform([X_car0])
```

4.2.4. Z-Score Normalization

- Transforms your data to mean = 0 and standard deviation = 1

$$x \mapsto \frac{x - \mu_X}{\sigma_X}$$

Advantage:
- works regardless of supervised / unsupervised learning

Disadvantage
- no straight-forward interpretation
- adopts negative values

4.2.5. Normalization Parameters

- Min-Max Normalization: (min / max)
- Z-Score Normalization: (mean, std)

Keep in Mind: Assess normalization parameters on training set for supervised learning
- The model may have unknown information about test data.

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('data/autoscout24/cars.csv')
df = df[['Price', 'Seats', 'Mileage']]
print(df.head())

# Min-Max Normlization
scaler = MinMaxScaler()
df[df.columns] = scaler.fit_transform(df)
print(df.head())

# Undo Normalization
df[df.columns] = scaler.inverse_transform(df)
print(df.head())

# Z-score Normalization
scaler = StandardScaler()
df[df.columns] = scaler.fit_transform(df)

print(df.head())

# Undo Normalization
df[df.columns] = scaler.inverse_transform(df)

print(df.head())
```

4.2.6. Language Normalization

```python
import nltk
```

- Natural Language Toolkit

4.2.6.1. lemmatization

- Endungen werden nach festen Regeln abgeschnitten
- weiss nichts über die Grammatik
- nutz Heuristiken

4.2.6.1.1. Beispiel

- *Computing* → *Comput*
- *Computer* → *Comput*
- *Computed* → *Comput*

4.2.6.2. stemming

- nutzt Wörterbücher und den Kontext

4.2.6.2.1. Beispiel

- *ist, war, bin* → *sein*
- *besser* → *gut*
- *Häuser* → *Haus*

5. Supervised Learning

DEFINITION: Supervised learning is the process where a machine learning model learns from labeled training data to predict specific target values for new, unseen data.

5.1. Regression Hypothesis and Evaluation Metrics

5.1.1. Regression Models

DEFINITION: As a specific type of supervised learning , regression models use labeled examples to learn a mathematical curve that predicts a continuous number like a car price.

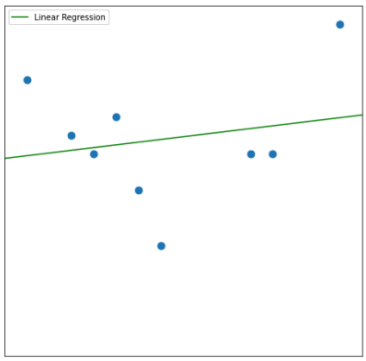
- **Common Types**: Linear Regression, Polynomial Regression, k-Nearest Neighbors, Support Vector Regression, Regression Trees, Random Forests, Neural Networks, XGBoost, and Gaussian Process Regression
- **Key Difference**: Models are distinguished by their hypothesis, which is the specific mathematical function they try to fit to the data

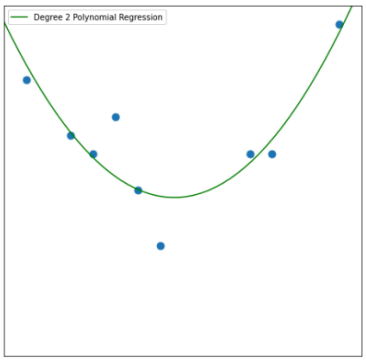
5.1.2. Regression Hypothesis

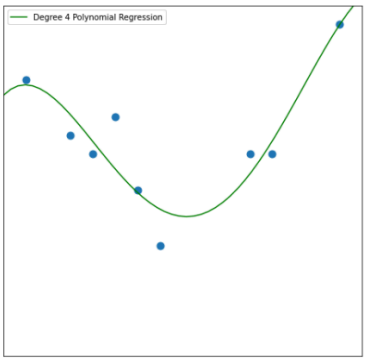
DEFINITION: Models differ in their hypothesis. A hypothesis is the mathematical function fitted to the data.

5.1.2.1. Linear and Polynomial Regression Hypothesis

- Creates a data-dependent approximation of the given data points
- Linear regression fits a simple straight line through the data
- Polynomial regression fits a curve of a specific degree to capture more complex continuous patterns

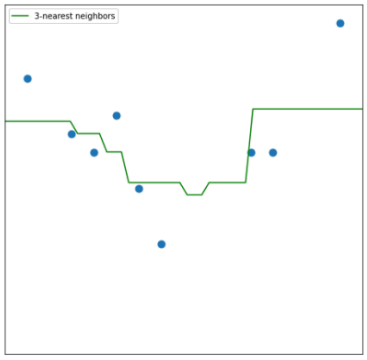


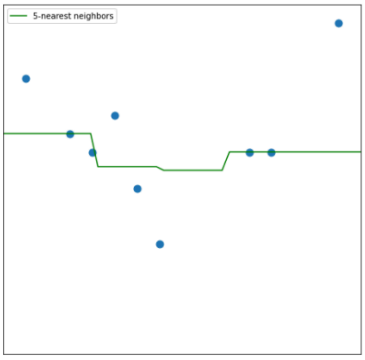


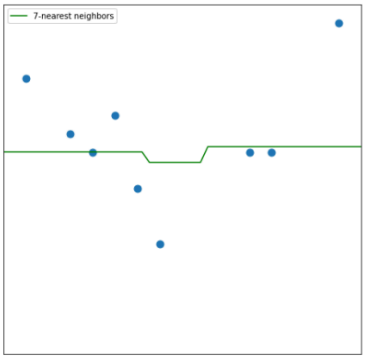


5.1.2.2. k-Nearest Neighbors Hypotheses

- Relies on calculating the average (AVG) of the closest neighboring points
- The number of neighbors is a crucial hyperparameter that defines the model shape
- Choosing more neighbors results in a more stable average and a straighter prediction curve

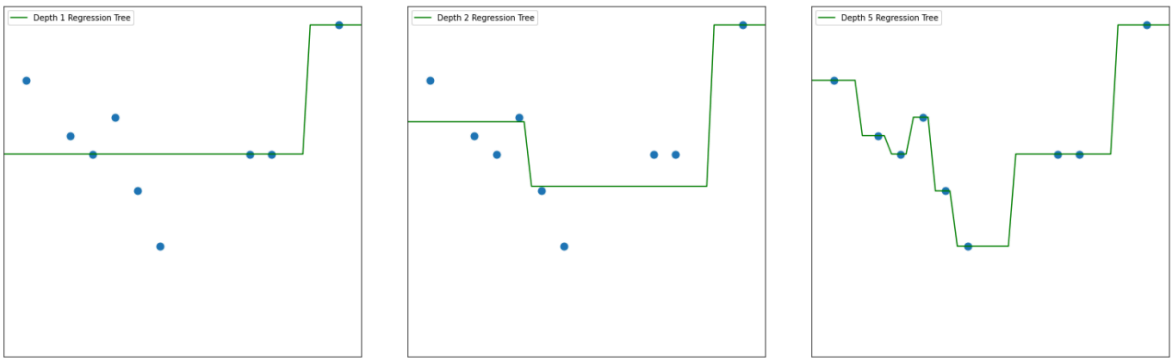






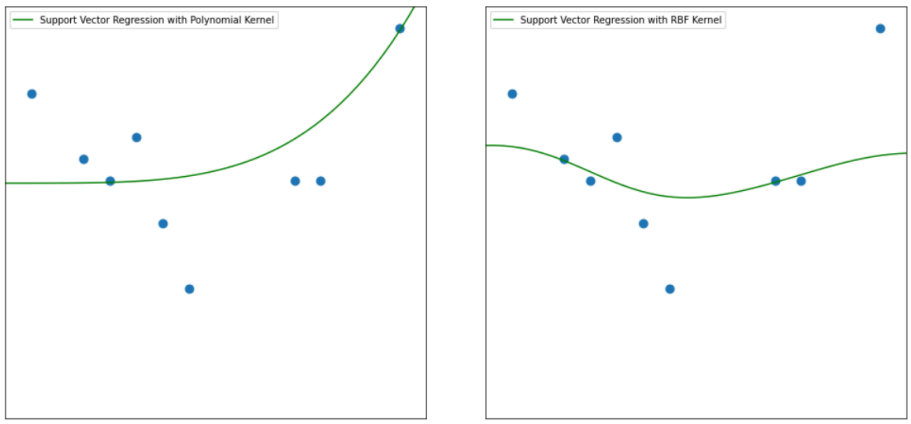
5.1.2.3. Regression Tree Hypotheses

- Creates only a step function rather than a smooth continuous curve
- The complexity depends directly on the chosen tree depth
- Deeper trees create more detailed steps to fit the training data closer



5.1.2.4. Support Vector Kernel Hypotheses

- Uses specific kernel functions to transform and approximate the data
- Kernel functions are mathematical tools that transform input data into a higher-dimensional space
- The Polynomial Kernel creates a curve similar to standard polynomial regression
- The „RBF Kernel“ utilizes Gaussian functions to model the data distribution

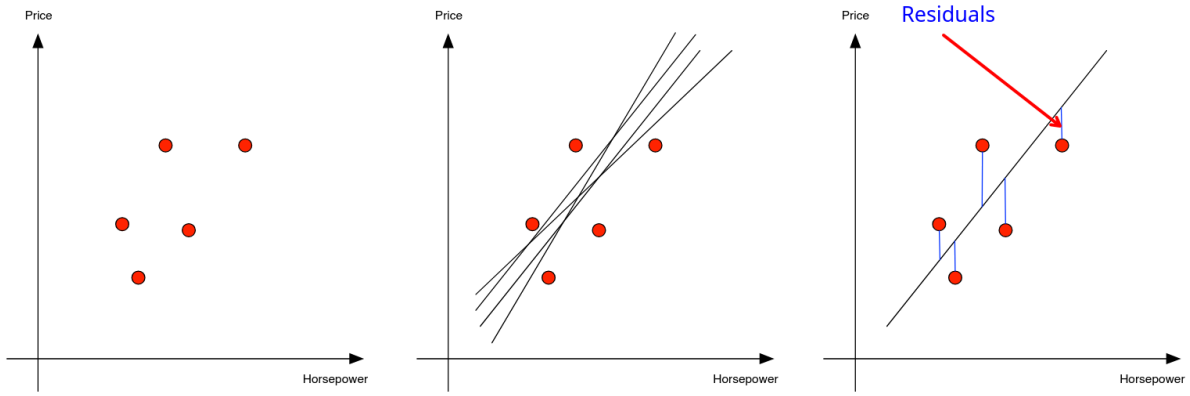


5.1.3. Regression Errors and Model Selection

DEFINITION: To identify the best model we need a way to measure predictive performance

5.1.3.1. Concept of Residuals

- Residuals are the differences between actual data points and model predictions (**the error**)
- Large residuals indicate poor data approximation
- This concept applies to any hypothesis



5.1.3.2. Minimizing the Error

- The best overall model minimizes the cumulated residuals
- Total error can be computed in different mathematical ways
- There is no single universally best model

5.1.3.3. Measure Regression Errors

- Assume model predictions f_i for actual data values y_i where $i = 1...m$
- $y_i - f_i$ = residuals

Summing errors

- Summing errors is not allowed because positive and negative values cancel out

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

MAE: Mean Absolute Error

- Easier for human interpretation
- Shares the exact same scale as the original data

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

MAPE: Mean Absolute Percentage Error

- Easier for human interpretation
- Useful for making percentage-based statements

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

MSE: Mean Squared Error

- Ideal for computers to optimize model performance
- More difficult to interpret manually due to the squared scale

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

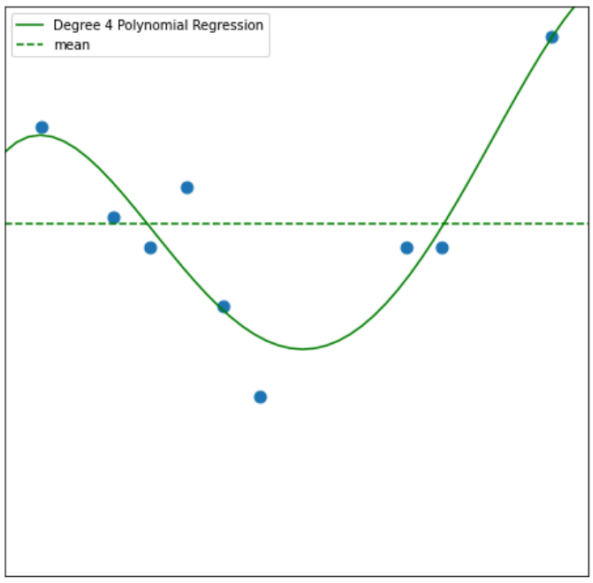
R-Squared R^2

- Human-readable percentage of explained data variance (how far data points spread out from the average)
- **Must be maximized (unlike MAE, MAPE and MSE)**
- $R^2 = 1$: Perfect fit
- $R^2 = 0$: Explains 0% variance, no better than simple mean
- $R^2 < 0$: Performs worse than simple mean approximation
- $R^2 \leq 1$: Always true because we subtract a positive fraction from 1

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

Comparison with Mean Approximation

- Evaluates how much better the regression is compared to simply taking the mean
- Compares the model's error (MSE) against the mean's error



5.1.4. Scikit-Learn Library

DEFINITION: Open-source Python library for machine learning and predictive data analysis.

Regression	
explained_variance	metrics.explained_variance_score
max_error	metrics.max_error
neg_mean_absolute_error	metrics.mean_absolute_error
neg_mean_squared_error	metrics.mean_squared_error
neg_root_mean_squared_error	metrics.mean_squared_error
neg_mean_squared_log_error	metrics.mean_squared_log_error
neg_median_absolute_error	metrics.median_absolute_error
r2	metrics.r2_score
neg_mean_poisson_deviance	metrics.mean_poisson_deviance
neg_mean_gamma_deviance	metrics.mean_gamma_deviance
neg_mean_absolute_percentage_error	metrics.mean_absolute_percentage_error
d2_absolute_error_score	metrics.d2_absolute_error_score
d2_pinball_score	metrics.d2_pinball_score
d2_tweedie_score	metrics.d2_tweedie_score

5.2. Evaluation Workflows

5.2.1. Motivation

- **Goal**: Identify the best model among $m_1 ... m_{\{10\}}$

5.2.1.1. Don'ts

- Never evaluate models on their own training data
- **Bad practice**:
  1. Train on training data
  2. Predict on the exact same data
  3. Calculate performance
  4. Select the best model

5.2.1.2. Key Concepts

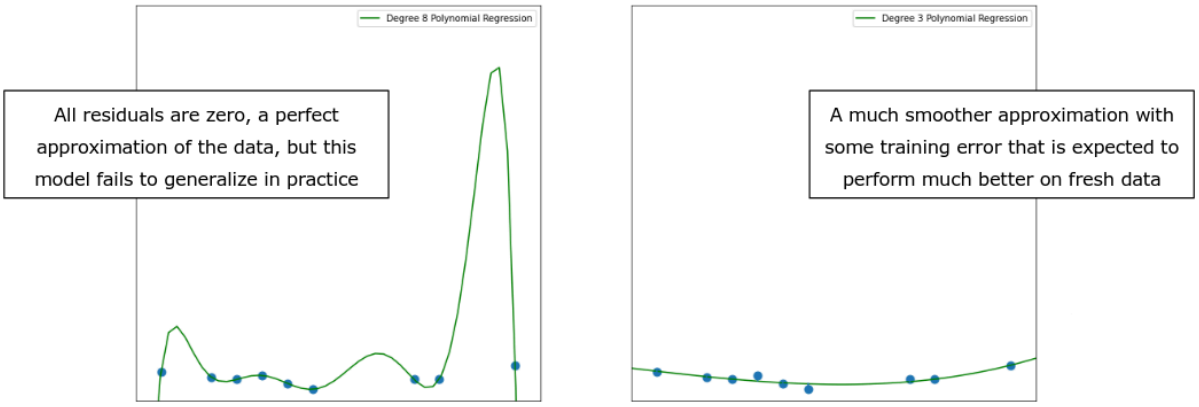
- **Training Error:** A low training error does not guarantee good performance on fresh data
- **Overfitting:** The model memorizes the exact data points instead of learning the underlying rules
- **Generalization Error:** Expected error on unseen data (must be low for a good model)

5.2.2. Perfect Performance vs. Generalization

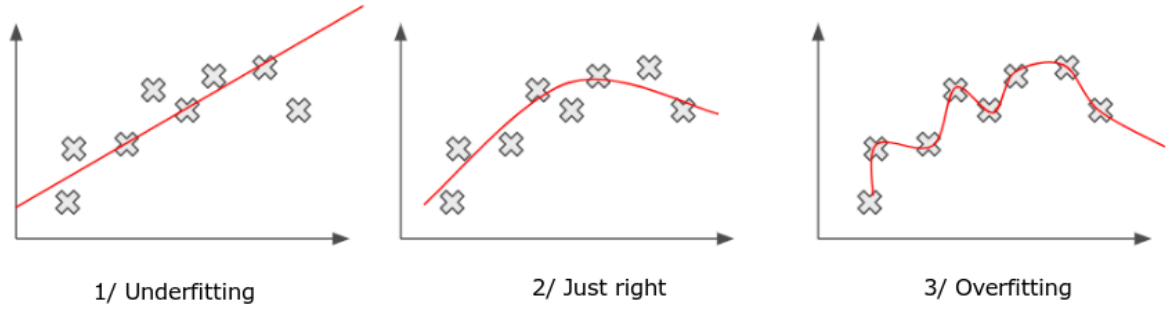
DEFINITION: **Training Error**: Error on training data, no guarantee for unseen data

DEFINITION: **Generalization Error**: Expected error on fresh data, shows true distribution learning

- **Max Training performance (Interpolation)**: Complex models (polynomials, trees, NNs) perfectly fit data (zero residuals, zero training error)
- **The Trap**: This perfect approximation often fails to generalize (performs poorly on new data)
- **Intuition**:
  - **Overfitted (e.g., Degree 8)**: Zero training error, captures noise, poor on fresh data
  - **Smoother (e.g., Degree 3)**: Non-zero training error, reflects true distribution, better on fresh data

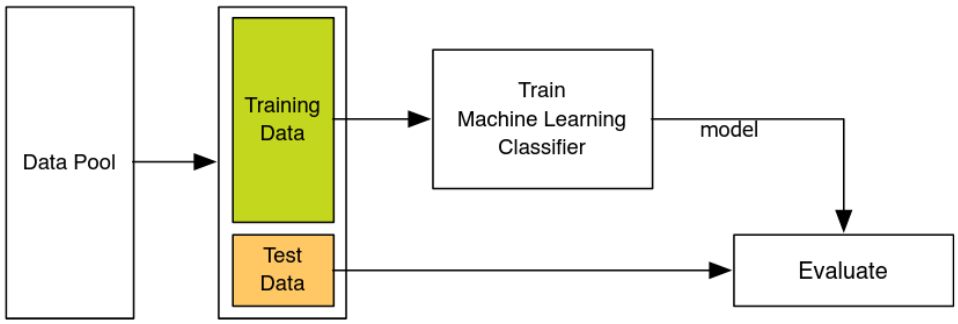


5.2.2.1. Underfitting vs. Overfitting



- **Underfitting**:
  - Too simple
  - High training error
  - High generalization error
- **Just Right (Goal)**:
  - Learns true distribution
  - Low training error
  - Low generalization error
  - Best for new data.
- **Overfitting**:
  - Too complex
  - Zero training error
  - High generalization error
- **Key Rule**: Never aim for perfect training performance.

5.2.3. Simplistic Machine Learning Workflow



- **Workflow**: Shuffle data -> Split (e.g., 80% Train, 20% Test) -> Train -> Evaluate (on Test set)
- **Purpose**: Estimate performance on unseen data
- **Limitations**: Requires massive amounts of data, only evaluates a single fixed hyperparameter config

5.2.4. Hyperparameters

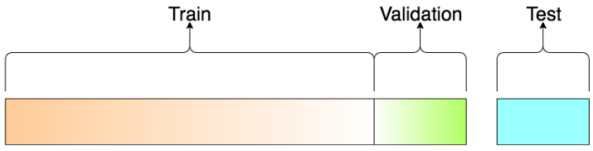
DEFINITION: The manual configuration of machine learning models

- **Examples**:
  - Number of neighbors (k)
  - Degree of polynomial
  - Regularization parameters
  - SVM Kernel
  - Tree depth
  - Neural Network settings (layers, neurons, activation functions)
- **Crucial Rule**: Never use simplistic workflow for multiple models/configs

5.2.5. The Human Brain as Hidden Channel

- **Problem**: Changing model settings after checking test results
- **Why it fails**: You accidentally inject knowledge about the test data into the model
- **Result**: The test data is not unseen anymore, so the performance score is artificially high
- **Core Message**: Use your test data only once

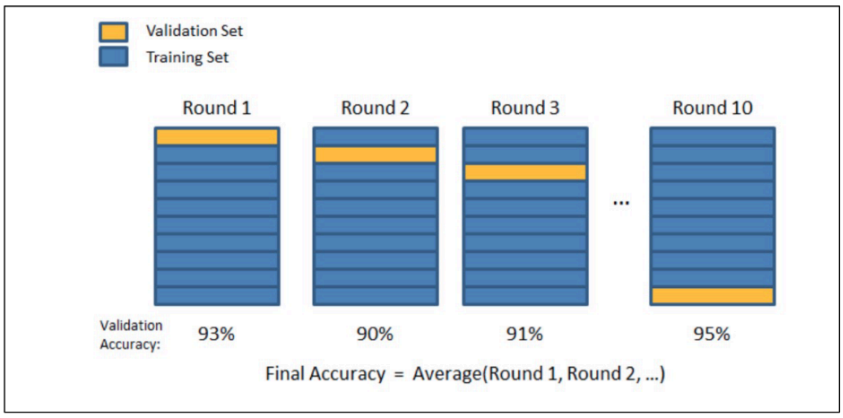
5.2.6. Evaluation Workflow for Model Selection



- **Recommended Data Split**: 60% Train, 20% Validation, 20% Test
- **Proper Steps**:
  1. Train models with different configurations on the training set
  2. Evaluate all variations on the validation set
  3. Select the best performing model based on validation results
  4. Evaluate this final model exactly once on the test set
- **Limitation**: This robust workflow requires a large amount of data

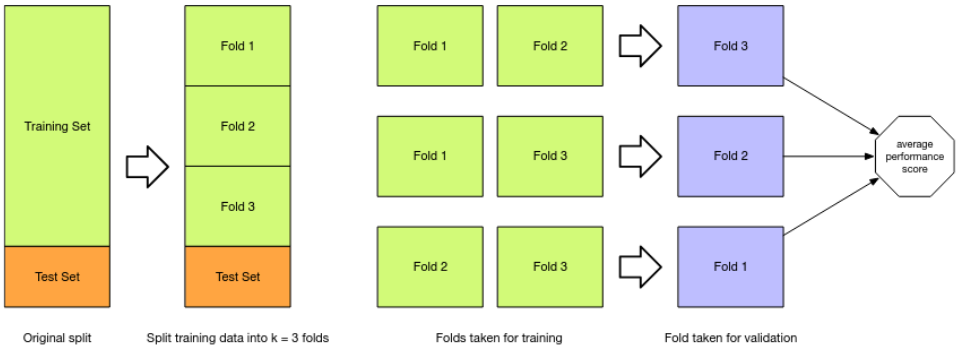
5.2.7. K-Fold Cross-Validation

- **Problem**: Standard 60/20/20 split requires large datasets
- **Solution**: Split 80% Train / 20% Test (lock test set away)
- **Process**: Perform k-fold cross-validation on the 80% training pool
  - Divide training data into k parts (e.g., 10 rounds)
  - Train on $k - 1$ parts, validate on the remaining part
  - **Final Score**: Average accuracy of all k rounds
- **Advantage**: Maximizes training data (e.g., 72% train, 8% validation per round)



5.2.7.1. Illustration: 3-Fold Cross-Validation

- **Process ($k = 3$)**:
  - Split training data into 3 equal folds
  - Train on 2 folds, validate on the remaining 1 fold
  - Repeat 3 times (each fold acts as validation exactly once)
- **Result**: Calculate the average performance score from all 3 rounds



5.2.8. Model Selection with Pipelines

1. **Setup**: Prepare data and set a `random_state` for consistent, reproducible results.
2. **Split**: Separate your data into 80% for training and 20% for the final „secret“ test.
3. **Pipeline**: Bundle scaling and model training together to prevent any data leakage.
4. **Tuning**: Use GridSearchCV to automatically find the best settings via cross-validation.
5. **Selection**: Identify the winning model configuration with the highest R^2 score.
6. **Final Test**: Evaluate the winner exactly once on the unseen 20% test data.

5.2.9. How to get Fired as a Data Scientist

- **Case 1: Tuning on Test Set**
  - **Problem**: Picking model settings ($k = 6$) because they perform best on the test set
  - **Result**: The test set is no longer „unseen“, making the final accuracy score fake and over-optimistic
- **Case 2: Scaling Leakage**
  - **Problem**: Calculating mean/standard deviation on the **entire** dataset before splitting it
  - **Why it fails**: Information from the test set „leaks“ into the training process through the scaler
  - **Result**: The model already knows the distribution of the test data before it is even tested
- **Core Message**: Keep your test data locked away and use it only once at the very end

6. Linear Regression

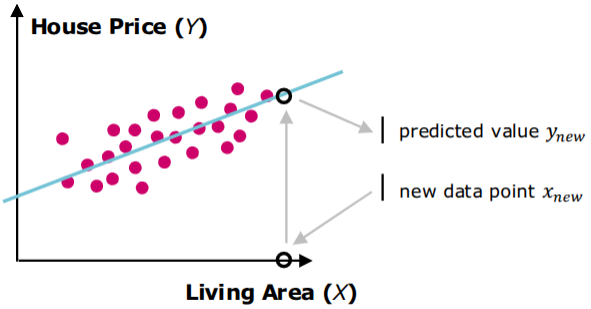
6.1. Overview

DEFINITION: The goal of regression is to find a relationship between features if one exists.

With linear regression the assumption is that the relationship between is approximately linear.

An estimate for Y is definde as:
$\hat{Y} = \theta_0 + \theta_1 X$

New data can now be used to make predictions:
$y_{\text{new}} = \theta_0 + \theta_1 x_{\text{new}}$



To determine the parameters θ_0 and θ_1 supervised learning can be used.

6.2. Terminology

Term	Explanation
X	Independent Variable (Regressor)
Y	Dependet Variable
(x_j, y_j)	the j-th data point
$h_{\theta(x_j)} = \theta_0 + \theta_1 x_j$	Hypotheses
θ_0, θ_1	regression parameters
θ_0	Y-intecept of the fitted line
θ_1	slope of the fitted line
$\hat{Y} = \theta_0 + \theta_1 X$	line of regression
ε_j	j-th residual (error term)
$y_j = \theta_0 + \theta_1 x_j + \varepsilon_j$	Function of y_j

Matrix Notation of Linear Regressions

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & X_N \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

Short form:

$$y = X\theta + \varepsilon$$

6.3. Ordinary Least Squares (OLS)

The regression parameter of the hypothesis are selected by minimizing a measure of the **total fitting error**

The following metrics can be used the measure the regression error over N data points:

MAE: Mean Absolute Error
$\frac{1}{N}\sum_{j=1}^{N}|e_j| = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$

MAPE: Mean Absolute Percentage Error
$\frac{1}{N}\sum_{j=1}^{N}\left|\frac{e_j}{y_j}\right| = \frac{1}{N}\sum_{j=1}^{N}\left|\frac{y_j - \hat{y}_j}{y_j}\right|$

MSE: Mean Squared Error $\frac{1}{N}\sum_{j=1}^{N} e_j^2 = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y}_j\right)^2$

MSE is a very common regression measuere because its a convex, continous and differentiable function of θ_0 , θ_1

6.3.1. Cost Function

The cost function $J(\theta_0, \theta_1)$ is used to minimze θ_1 and θ_2 using MSE.

$$J(\theta_0, \theta_1) = \frac{1}{N}\sum_{j=1}^{N}\varepsilon_j^2 = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y_j}^2\right) = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \theta_0 + \theta_1 x_j\right)^2$$

In Matrix Form the cost function looks like this:

$$J(\theta) = \varepsilon^T\varepsilon = (y - X\theta)^T(y - X\theta) = y^Ty - 2\theta^TX^Ty + \theta^TX^TX\theta$$

Because the cost function is **convex** it has a single global minimum.

- The minimum can be found by calculating the slope of the function and requiring it to be 0

How to get to the slope:

$$\frac{\partial(\theta^TX^TX\theta)}{\partial(\theta)} = 2X^TX\theta$$

$$\frac{\partial(\theta^TX^TX\theta)}{\partial(\theta)} = X^Ty$$

$$\frac{\partial J(\theta)}{\partial(\theta)} = -2X^Ty + 2X^TX\theta = 0$$

$$X^TX\theta = X^Ty$$

$$\theta_{\text{opt}} = \left(X^TX\right)^{-1}X^Ty$$

TL;DR: OLS $= \theta_{\text{opt}} = \left(X^TX\right)^{-1}X^Ty$

6.4. Gradient Descent

While finding θ_0 and θ_1 by OLS can be done in one pass it can also be down iteratively by gradient Descent.

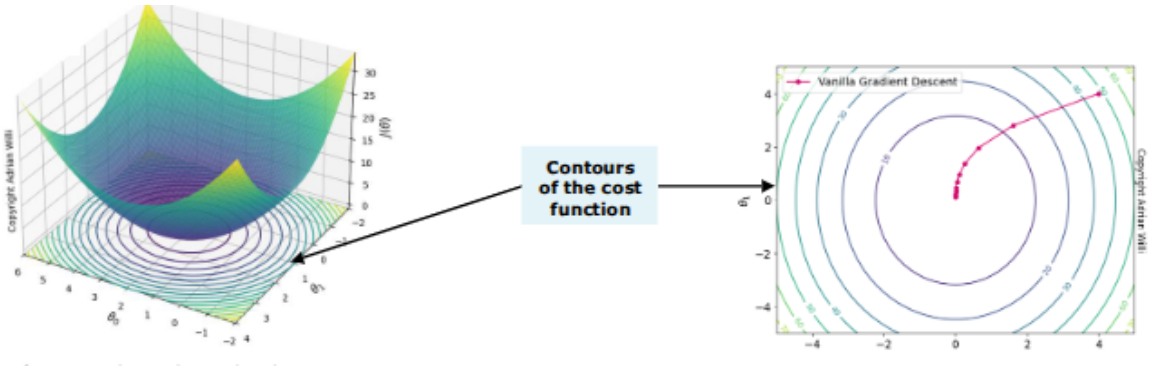
$\rightarrow$ this is done when calculating an inverse is costly becuase the scale of the data is very large.

1. Start with a random guess for θ
2. Take small steps down the cost surface
3. Step direction = negative of gradient

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha\frac{\partial(J(\theta))}{\partial(\theta)}$$

Alpha:

- learning rate
- decided how big the steps are



6.5. Regression Performance R^2

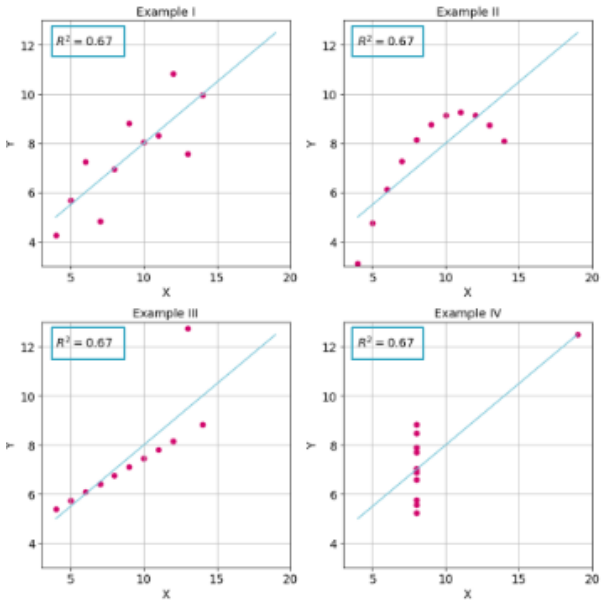
To measure the regression performance the prediction is compared to a prediction done by taking the simple mean of the data.

- Variability of Y around regressor line $\longrightarrow$ MSE
- Variability of Y around its mea $\overline{Y}$ $\longrightarrow$ Variance
- The performance metric R^2 is the ratio $1 - \frac{\text{MSE}}{\text{Variance}}$

$$R^2 = 1 - \frac{\sum_j \varepsilon_j^2}{\sum_j \delta_j^2}$$

$R^2 = 100\% \longrightarrow$ perfect fit of the regression line to the data

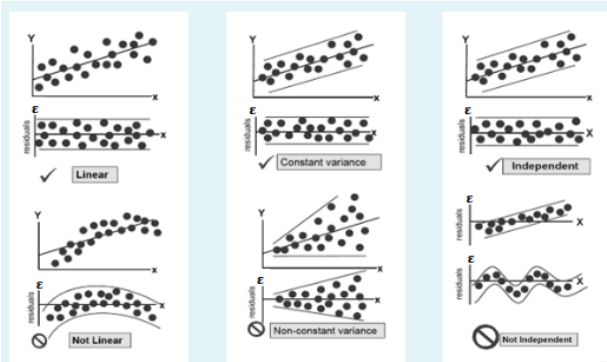
R^2 is not always a good metrix as illustrated in these examples:



To escape this trap it is important to visualize the residuals:

$$\varepsilon = y - X\theta$$

$\longrightarrow$ for a good fit the residuals should look like zero-mean, stationary, white noise



WICHTIG: Correlation does not imply causation)

6.6. Beyond Linear Regression

6.6.1. Multiple Linear Regression

- regression with more than one Variable
- regression with two features is visualized as a plane in 3-Dimensions
- regression with more features is a hyperplan in N-dimensional feature space
- visualization higher dimensions is not trivial

The matrix form for m-Regressors looks like this:

$$\begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} = \begin{pmatrix} 1 & x_1(1) & x_2(1) & \dots & x_{M(1)} \\ 1 & x_1(2) & x_2(2) & \dots & x_{M(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_1(N) & x_2(N) & \dots & x_{M(N)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{pmatrix}) + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

The formula for the Ordinary Least Square stays the same:

$$\begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_M \end{pmatrix} = (X^T X)^{-1} X^T y$$

6.6.2. Non Lineare Regression

If the relationship between the relationships is not linear polynomial regression can be used

$$h(\theta, X) = \theta_0 + \theta_1 X + \theta_2 X^2 + \theta_3 X^3$$

With a little trick this be converted into a linear hypotheses: $X_1 = X, X_2 = X^2, X_3 = X^3$

This results in a linear hypothesis

$$h(\theta, X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3$$

WICHTIG: By introducing higher powers of the data the feature will be created at a different scale. Renormalize the features before the regression

6.6.3. Regularization

By using more features we can get a better fot to our training data ⟶ this can result in overfitting

This way the model will fit worse with new data

To avoid this the focus has to be on the features with the most explanatory powers.

This is called **Regularization**

Methods for Regularization:
- Ridge
- LASSO

6.6.3.1. Ridge Regression

For ridge regression an extra term is added to the cost function

$$J(\theta) = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y_j}\right)^2 + \lambda\sum_{k=1}^{M}\theta_k^2$$

In ridge regression coefficients can not become 0

6.6.3.2. LASSO Regression

For the Lasso Regression the term added is the absolute value of the parameters

$$J(\theta) = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y_j}\right)^2 + \lambda\sum_{k=1}^{M}|\theta_k|$$

In LASSO regression coefficients can become 0

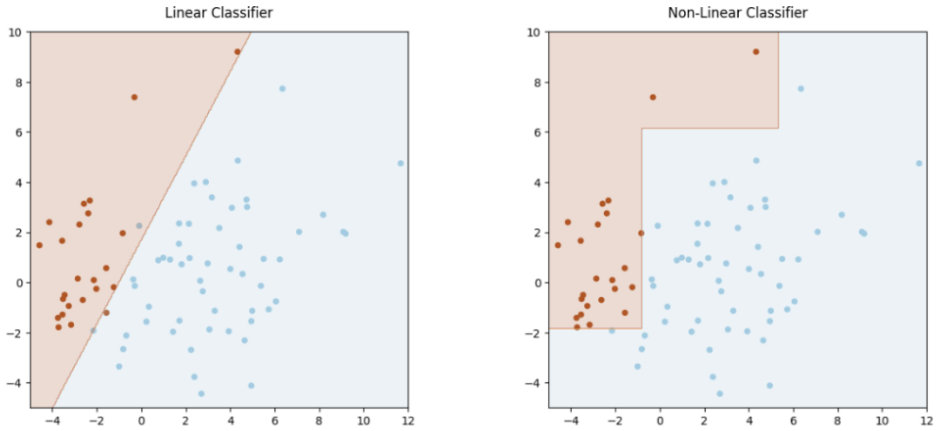
6.6.3.3. Regularization Parameter λ

- Larger Parameters get penalized
- Only important features will drive the regression
- less relevant features get supressed
- control by Regularization parameter λ

7. Classification

7.1. Decision Boundary & Logistic Regression

A **classifier** draws a **decision boundary** to separate data points



- **Linear Classifier**: Draws a straight line
  - **Example**: Logistic Regression
- **Non-Linear Classifier**: Creates complex, jagged, or curved boundaries
  - **Example**: Decision Tree

7.1.1. Find Linear Decision Boundary

- **Range Linear Regression**: $]-\infty, +\infty[$
- **Range Probability**: $[0, 1]$
- Constraint range of Linear Regression to Probability
- **Trick**: Use linear regression to model the logit (log-odds) instead of probability.

7.1.2. Probability vs. Odds

Probabilities

- Ratios of something happening, to everything that could happen
- Values between 0 and 1

Odds:

- Ratios of something happening, to something not happening
- Can have any non-negative real (reelle Zahlen), only lower bounded

$$\text{odds} = \frac{p}{1-p}$$

7.1.2.1. Example

Horse range with 6 equally fast horses:

- Probability to win: $p = \frac{1}{6}$ (1 among the 6 horses will win)
- Odd to win: 1 to 5 (1 horse will win and 5 will lose)

7.1.3. Logits

DEFINITION: The logit is a mathematical trick that turns a probability into an infinite number, which allows us to use a straight line to predict classification outcomes.

- **Logit**: logarithm of the odds
- Positvely and negatively unbounded
  - **Goal**: $-\infty, +\infty$
- To be able to do mathematics regression

$$\text{logit} = \log\left(\frac{p}{1-p}\right)$$

Symmetry:

$$\log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p) = -(\log(1-p) - \log(p)) = -\log\left(\frac{1-p}{p}\right)$$

- Probability of winning = Difference between the probability of winning and the probability of losing
- Logits are infinite in both directions, which matches the unlimited range of linear regression.

7.1.4. From Linear to Logistic Regression

- **Goal**: 0 / 1 to make a decision

Step 1: Linear Regression

$$h_{\theta(\boldsymbol{x})} = \Theta \boldsymbol{x}^T$$

- **Problem**: Can have any value

Step 2: Use logit to have $-\infty, +\infty$

$$\log\left(\frac{p}{1-p}\right) = \Theta \boldsymbol{x}^T$$

Step 3: Solve the equation for p: **Sigmoid function**

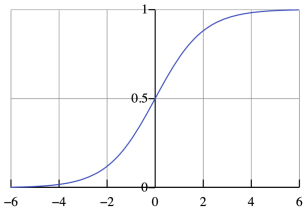
$$p = \frac{1}{1 + e^{-\Theta \boldsymbol{x}^T}}$$

- **Result**: p is always between 0 and 1

7.1.5. Sigmoid Function

DEFINITION: Squeeze any positiv / negativ number into Probability from $[0, 1]$

$$g(z) = \frac{1}{1 + e^{-z}}$$



7.1.6. Logistic Regression

DEFINITION: A classification model that uses linear regression to estimate logits, then applies the Sigmoid function to map them into a $[0, 1]$ probability range

$$h_{\Theta(\boldsymbol{x})} = g(\Theta \boldsymbol{x}^T) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n) \quad \text{with} \quad g(z) = \frac{1}{1 + e^{-z}}$$

1. Sigmoid Function is applied to linear regression
2. Logits are estimated by linear regression
3. Sigmoid transforms the estimated logit into a probability value

7.1.6.1. Training

DEFINITION: Finding parameters values θ that minimize some **loss function**

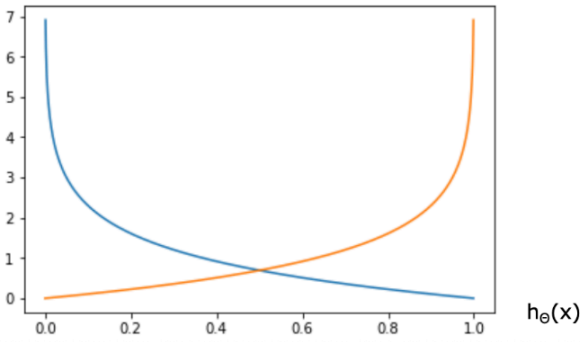
- For linear regression, the loss function is proportional to the mean squared error (MSE)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(y^i - h_\theta(x^i)\right)^2$$

- MSE is **not** a suitable
  - MSE is convex (one local minimum), sigmoid is not → learning process can get stuck in a local minimum (before finding the best path)
- MSE take care of distances (penalty's), we need penalty's if the classification is wrong

7.1.6.2. Loss Function - Cross Entropy

$$\text{cost}\left(h_{\Theta(\boldsymbol{x})}, y\right) = \begin{cases} -\log\left(h_{\Theta(\boldsymbol{x})}\right) & \text{if } y = 1 \\ -\log\left(1 - h_{\Theta(\boldsymbol{x})}\right) & \text{if } y = 0 \end{cases}$$



- The if/else definition is not convenient for optimization
- **Transform**:
  - Combine both cases into a single differentiable loss function

$$\text{cost}\left(h_{\Theta(\boldsymbol{x})}, y\right) = y \cdot \left(-\log\left(h_{\Theta(\boldsymbol{x})}\right)\right) + (1 - y) \cdot \left(-\log\left(1 - h_{\Theta(\boldsymbol{x})}\right)\right)$$

- Is convex
- Can be **minimized** efficiently with gradient-based methods

7.1.6.3. Summary

Training data:

- Labeled examples used for learning

$$\{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), ..., (\boldsymbol{x}^{(m)}, y^{(m)})\}$$

$$\text{with } \boldsymbol{x}^{(i)} \in \mathbb{R}^n \text{ and } y^{(i)} \in \{0, 1\} \text{ and } x_1^{(i)} = 1$$

Hypothesis:

- Predicts the probability of class 1

$$h_{\Theta(\boldsymbol{x})} = g(\Theta \boldsymbol{x}^T) = \frac{1}{1 + e^{-\Theta \boldsymbol{x}^T}}$$

Loss function:

- Cross-entropy loss for binary classification

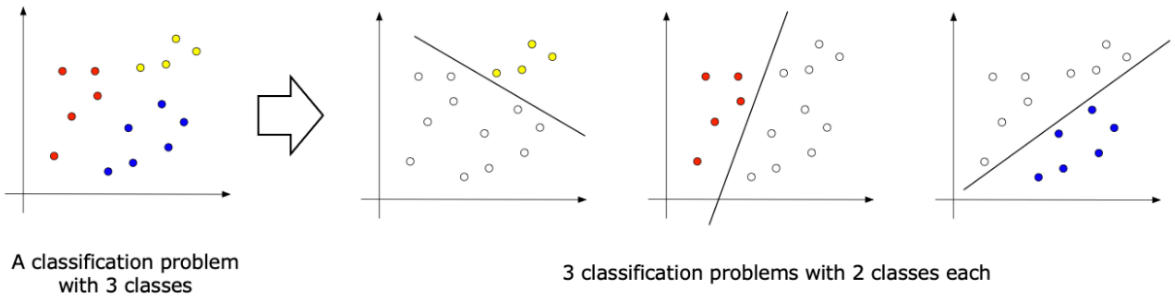
$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \Big(y^{(i)} \cdot \log\Big(h_{\Theta(\boldsymbol{x}^{(i)})}\Big) + \big(1 - y^{(i)}\big) \cdot \log\Big(1 - h_{\Theta(\boldsymbol{x}^{(i)})}\Big) \Big)$$

Gradient descent finds parameter values that minimize $J(\Theta)$

7.1.7. Multiclass Classification with Logistic Regression

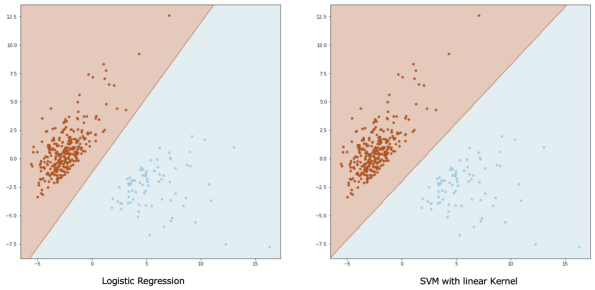
DEFINITION: Adapting binary logistic regression for problems with multiple categories

- **One-vs-Rest (OvR) Approach**: Training k separate binary models for a problem with k classes.
- **Model Specificity**: Each binary model distinguishes one specific class from all other classes combined.
- **Prediction Strategy**: Assigning data points to the class associated with the highest confidence score among all models.
- **Visual Representation**: A 3-class problem (red, yellow, blue dots) decomposed into three individual linear decision boundaries.



7.2. Classification Metrics

7.2.1. Linearly Separable Data

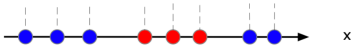


Logistic Regression:

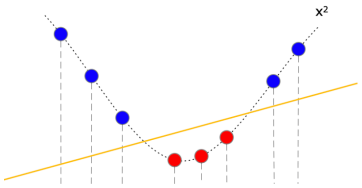
- Tend to orange, because there are more

SVM:
- Maximizes the margin (distance) between different classes
- **Margin Maximization**: Focuses on the optimal hyperplane
- **Tolerance**: Allows for misclassified points (soft margin) to ensure a robust boundary
- see CVAI

7.2.1.1. SVMKernel Trick (SVM)
- Can't be solved with Linear Equation:

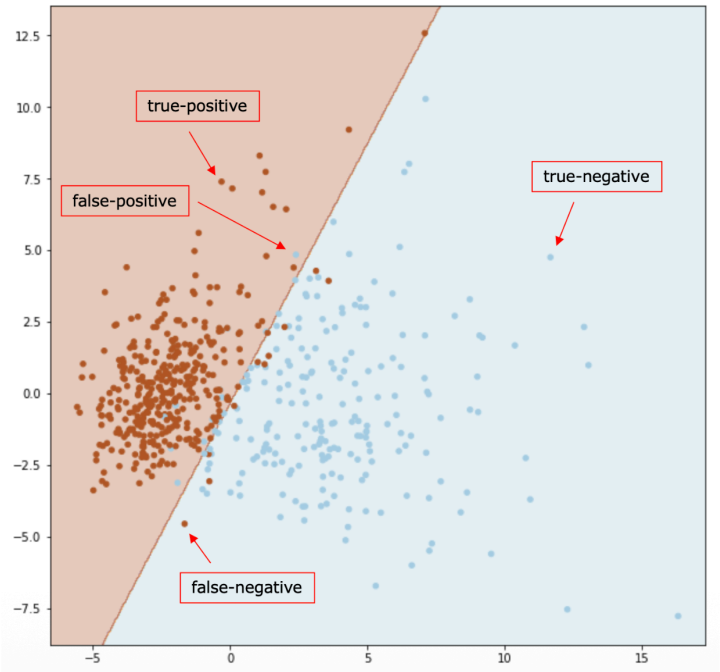


- **Trick:** Project Data into higher dimensional space:



- Can now be solved

7.2.2. Confusion Matrix



- Orange = positive class
- Blue = negativ class

Example:

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

true-negative (TN)
- Predicted NO, actual NO
- 50

false-positive (FP)
- Predicted YES, actual NO
- 10

false-negative (FN)
- Predicted NO, actual YES
- 5

true-positive (TP)
- Predicted YES, actual YES
- 100

7.2.2.1. Accuracy & Error-Rate

Accuracy: How often the **classifier was right**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

Error-Rate: How often the **classifier was wrong**

$$\text{Error Rate} = \frac{\text{FP} + \text{FN}}{\text{Total}} = 1 - \text{Accuracy}$$

7.2.2.1.1. Problem with Accuracy

Accuracy Limitations:

- Unreliable for **imbalanced datasets** (e.g., 5000 „No“ vs. 20 „Yes“ cases).
- High accuracy (99.6%) can be achieved by a „dumb“ classifier that predicts „No“ every time.
- Fails to identify any actual positive cases in rare disease scenarios.

Misleading Weighting:

- Treats **False Positives (FP)** and **False Negatives (FN)** as having equal importance.
- It ignores that a False Negative (e.g., missing cancer) is often much worse than a False Positive (e.g., false alarm)
- **Avoid Accuracy**: Do not use it for imbalanced data or when error costs differ.
- **Use Alternatives**: Switch to Precision, Recall, or F1-Score for a more reliable performance overview

7.2.2.2. Precision

When the model predicts YES, how often is it correct (Quality of positive predictions)

- **P*recision: TP** / P*redicted YES

$$\text{Precision} = \frac{\text{TP}}{\text{Predicted YES}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 1	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

7.2.2.3. Recall - True-Positive Rate - Sensitivity

When the true value is YES, how often does the model predict YES (Quantity of positive cases found)

- **R*ecall: TP** / R*eal YES

$$\text{Sensitivity} = \frac{\text{TP}}{\text{Actual YES}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

7.2.2.4. F1 Score

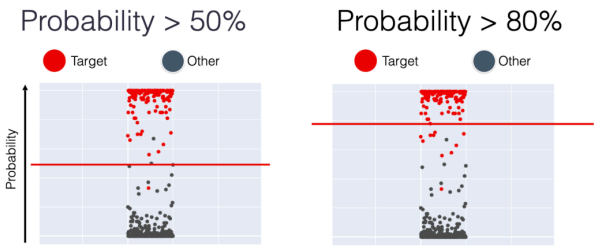
Harmonic mean between precision and recall

- **Why**: Normal average doesn't work for ratios with different bases

$$\mathrm{F1} = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

- **Property**: Strongly biased towards the worse score
- **Usage**: Best for imbalanced / skewed data
- **90% Score** Accuracy = Often a bad model | F1 = A good model

7.2.3. Classification Threshold



- Classifiers output a **probability** rather than a simple „Yes" or „No."
- The **threshold** (e.g., 0.5, 0.8) determines the point at which a probability is classified as the positive class.
- **Higher thresholds** increase confidence in positive predictions but increase the risk of missing cases (more FN).
- **Lower thresholds** capture more positive cases but increase the risk of false alarms (more FP).

7.2.3.1. Confusion Matrix per Threshold

Patients	A	B	C	D	E	F	G	H	I
Cancer	0	0	0	0	0	1	1	1	1
Model Prediction	0.10	0.20	0.60	0.45	0.30	0.55	0.75	0.90	0.85

Probability ≥ 0.5	0	0	1	0	0	1	1	1	1
Probability ≥ 0.6	0	0	1	0	0	0	1	1	1
Probability ≥ 0.7	0	0	0	0	0	0	1	1	1
Probability ≥ 0.8	0	0	0	0	0	0	0	1	1

	Prediction = 0	Prediction = 1
Cancer = 0	4	1
Cancer = 1	0	4

Confusion Matrix for Threshold 0.5

	Prediction = 0	Prediction = 1
Cancer = 0	4	1
Cancer = 1	1	3

Confusion Matrix for Threshold 0.6

	Prediction = 0	Prediction = 1
Cancer = 0	5	0
Cancer = 1	1	3

Confusion Matrix for Threshold 0.7

	Prediction = 0	Prediction = 1
Cancer = 0	5	0
Cancer = 1	2	2

Confusion Matrix for Threshold 0.8

- Calculate for each Threshold:

$$\text{True-Positive Rate (Recall)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Same as Recall

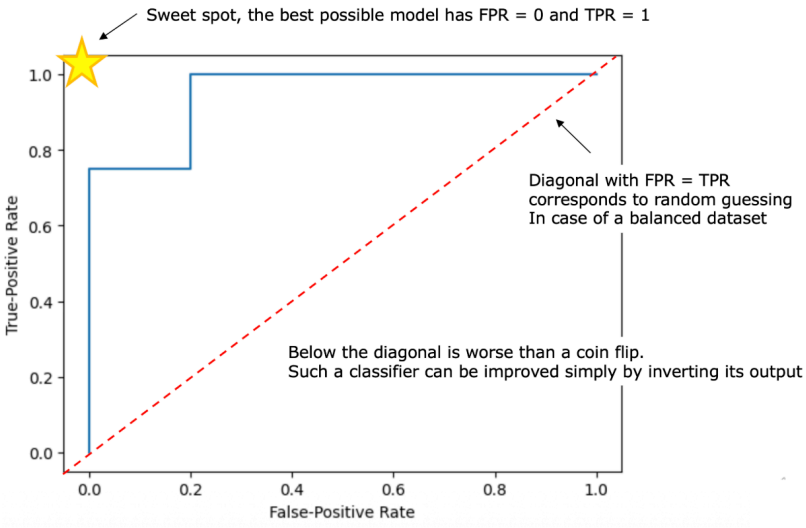
$$\text{False-Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

	True-Positive Rate	False-Positive Rate
Threshold 0.5	1	0.2
Threshold 0.6	0.75	0.2
Threshold 0.7	0.75	0
Threshold 0.8	0.5	0

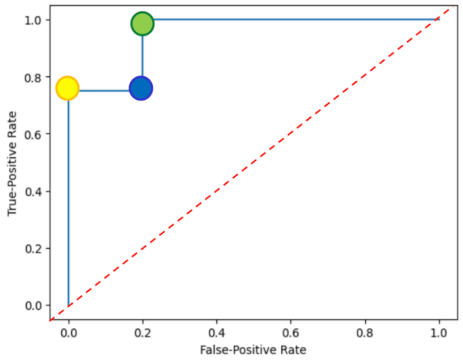
Trade-Off

- **Threshold 0.5:** High TPR (1.0) but also a higher FPR (0.2). Good for not missing anyone, but includes false alarms.
- **Threshold 0.8:** Lower FPR (0.0) but a much lower TPR (0.5). High certainty in its predictions, but misses half of the actual cancer cases.

7.2.3.2. ROC Curves



- Plots the **True-Positive Rate (TPR)** against the **False-Positive Rate (FPR)** at various threshold levels.
- Each point on the blue line represents a different classification threshold.



- **Yellow Point:** Preferred for minimizing false alarms (low FPR) while maintaining decent detection.
- **Green Point:** Preferred for maximizing detections (high TPR) while accepting some false alarms.
- **Blue Point:** Suboptimal; for the same TPR, yellow has a lower FPR, and for the same FPR, green has a higher TPR.

7.2.3.3. Area under the ROC Curve - AUROC

- Used to compare the overall performance of different classifiers.
- Unlike Accuracy or F1-score, it does not require choosing a specific threshold.
- It measures performance across **all possible thresholds** simultaneously.

Interpretation

- Value ranges from **0 to 1**.
- **AUROC = 0.5:** Represents a model that is no better than random guessing.
- **Higher AUROC (Bigger Area):** Indicates a model that performs better overall, regardless of where the threshold is set.
- **Example**: AUC=0.95 means a 95% chance of correctly ranking a positive example higher than a negative one.

7.2.3.4. Precision-Recall Curve

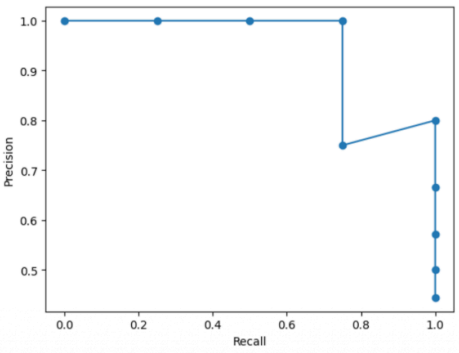
ROC/AUROC Limitations:

- Should be avoided if the dataset is **strongly imbalanced**.
- False-Positive Rate (FPR) becomes deceptively small when the number of actual negatives is very large, making a model look better than it is.

Precision-Recall Curve (PRC):

- Replaces FPR with **Precision** to focus on the quality of positive predictions.
- Effectively highlights model performance in finding the minority class.

Threshold	Precision	Recall
0.10	0.444444	1.00
0.20	0.500000	1.00
0.30	0.571429	1.00
0.45	0.666667	1.00
0.55	0.800000	1.00
0.60	0.750000	0.75
0.75	1.000000	0.75
0.85	1.000000	0.50
0.90	1.000000	0.25
NaN	1.000000	0.00

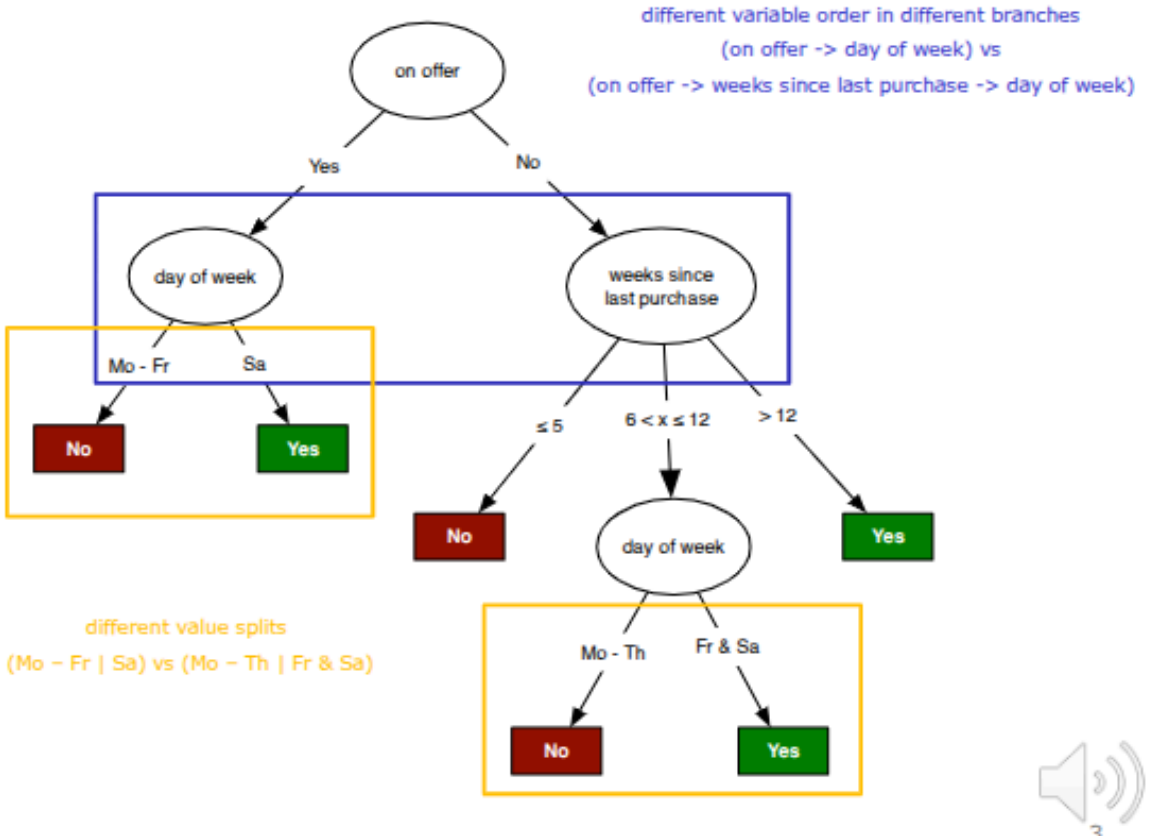


Metric Relationship:

- As thresholds change, **Precision** and **Recall** typically move in opposite directions.
- Higher thresholds lead to higher Precision (fewer false alarms) but lower Recall (more missed cases).

8. Tree Models

DEFINITION: Tree models try to predict values by splitting the problem up into multiple decisions



Trees can be different based on:
- the variables order in the different branches
- the value splits

The learning algorithm of a decision tree must filter out non-informative features
- it is most beneficial if the ratio between the different values in a branch is very different or even pure

8.1. Tree Construction Rules

1. if only positive or negative instances are left (pure leaf)
   - stop branching and assign the corresponding decision as leaf node
2. if some positive and some negative examples remain:
   - choose another feature for branching or decide according to label frequency
3. if no instances are left such a combination of features does not occur in the training set.
   - 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 trianing data is contradictory, noisy, non-deteministic or contains hidden features
   - decide according to the more frequent label

Point 3 and 4 are infrequent cases used only for error handling

8.2. Splitting criterion

Two approaches:
- select features with the lowest Gini impurity
- select feature with highest information gain

8.2.1. Gini impurity

DEFINITION: Gini impurity measures the likelihood of an incorrect classification of a random data record, if that record was randomly labelled according to the distribution of labels from the training data

The lower the Gini impurity score the better

Formula:

$$1 - \sum_{i=1}^{J} pi^2$$

- p_i: Probability of an item belonging to class i

Scale

- **0 (Pure):** All records in a node belong to a single class (e.g., 100% „Yes“ or 100% „No“).
- **0.5 (Max Impurity for binary):** Equal distribution of classes (e.g., 50/50 split).
- The lower the better

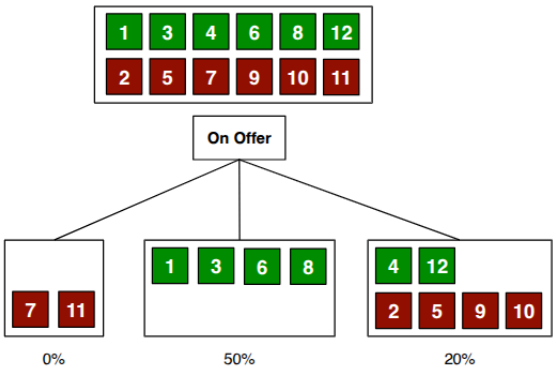
Gini for entire feature / Weighted Gini

$$\text{Weight} = \frac{\text{Items in Branch}}{\text{Total Items before split}}$$

$$\text{Gini for Feature} = \sum(\text{Weight} \times \text{Gini of Node})$$

8.2.1.1. Example

The example is based on the following tree



Label	p(No)	p(Yes)	1 - p(No)² - p(Yes)²
0%	1	0	0
20%	2/3	1/3	4/9
50%	0	1	0

Label Distribution: p(0%) = 2/12, p(20%) = 6/12, p(50%) = 4/12

Gini Impurity: 2/12 * 0 + 6/12 * 4/9 + 4/12 * 0 = 2/9

8.2.1.2. Giny Impurity for continous Variables

1. Sort table according to feature variables
2. calculate average of adjacent values
3. take averages as splitting criteria and proceed in the usual way

8.2.1.2.1. Example

Data Record	Features						Target
	Weather	Store Size	Saturday	Price Reduction	Season	Days since last Purchase	Product purchased
1	Rain	*left out for simplicity*				10	Yes
2	Sun					30	No
3	Cloudy					10	Yes
4	Cloudy					20	Yes
5	Rain					60	No
6	Rain					10	Yes
7	Sun					10	No
8	Sun					10	Yes
9	Rain					60	No
10	Rain					30	No
11	Rain					10	No
12	Rain					60	Yes

Data Record	Features						Target
	Weather	Store Size	Saturday	Price Reduction	Season	Days since last Purchase	Product purchased
1	Rain	*left out for simplicity*				10	Yes
3	Cloudy					10	Yes
6	Rain					10	Yes
7	Sun					10	No
8	Sun					10	Yes
11	Rain					10	No
4	Cloudy					20	Yes
2	Sun					30	No
10	Rain					30	No
5	Rain					60	No
9	Rain					60	No
12	Rain					60	Yes

Data Record	Features						Target
	Weather	Store Size	Saturday	Price Reduction	Season	Days since last Purchase	Product purchased
1	Rain	*left out for simplicity*				10	Yes
3	Cloudy						Yes
6	Rain						Yes
7	Sun						No
8	Sun						Yes
11	Rain						No
4	Cloudy					20	Yes
2	Sun					30	No
10	Rain						No
5	Rain					60	No
9	Rain						No
12	Rain						Yes

averages between adjacent values

15

25

45

Label	p(No)	p(Yes)	1 - p(No)² - p(Yes)²
< 15	1/3	2/3	4/9
>= 15 < 25	0	1	0
>= 25 < 45	1	0	0
>= 45	2/3	1/3	4/9

Label Distribution: p(< 15) = 6/12, p(>= 15 < 25) = 1/12, p(>= 25 < 45) = 2/12, p(>= 45) = 3/12

Gini impurity: 6/12 * 4/9 + 1/12 * 0 + 2/12 * 0 + 3/12 * 4/9 = 1/3

8.3. Regression Trees

- are grown like decision trees for classification but with **different splitting criteria**
- instead of Gini impurity, minimize **variance** of values in the same subset
- predict **average value** of all instances in a leaf node

8.4. Advantages of Tree Models

1. Simple to understand and interpret for humans
2. can handle both numeric and categorical data
3. almost no data preparation
4. perform well on large datasets

8.5. Disadvantages

1. often not as accurate as other approaches
2. Learner tend to create over-complex trees that do not generalize well (overfitting)
3. Trees are very sensitive to data quality

8.6. Tree Models in Scikit Learn

Important Hyperparameter:

- max_depth -> controls pruning and therefore counteracts overfitting

8.7. Random Forest

DEFINITION: A random forest is a collection of decision trees

Counteracts:
- overfitting of data trees
- the sensitivity to data quality

The decision trees are built from a subset of the training data:
1. Random row sampling with replacement
2. Random column feature selection without replacment

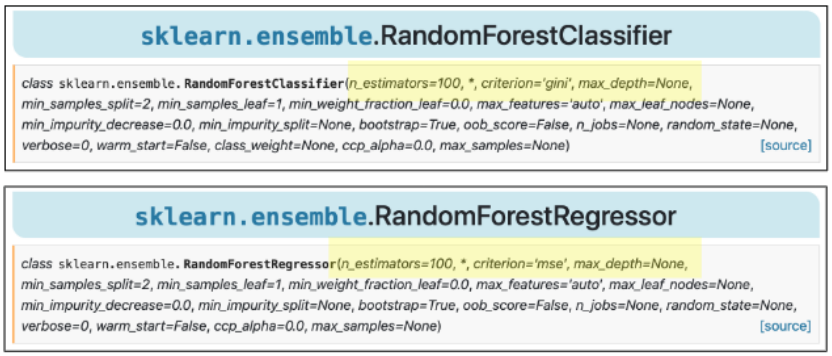
Training Phase:
1. Choose random subset D^* of your training set Disadvantages
2. Choose random subset A^* of attributes
3. Build a decision of regression tree from A^* and D^*
4. Repeat step 1-3 for the desired number of trees

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
   - Average for regression
   - Majority vote for classification

Advantages:
- Only a fraction of the sampled data creates irrelevant structures that would lead to overfitting
- An anomaly is only contained in a tiny fraction of the sampled data structures

8.8. Random Forests in Scikit Learn API

Important Hyperparameters:
- max_depth -> controls pruning to counteract overfitting
- n_estimators -> refers to the number of trees



8.9. Ensembles

- Complex machine learning models built from simpler models
- Two techniques are bagging and boosting

WICHTIG: Do not mix different model in production, it makes the ensemble to complex

8.9.1. Bagging

Computes a weighted average of votes from simple models

- Example: random forest

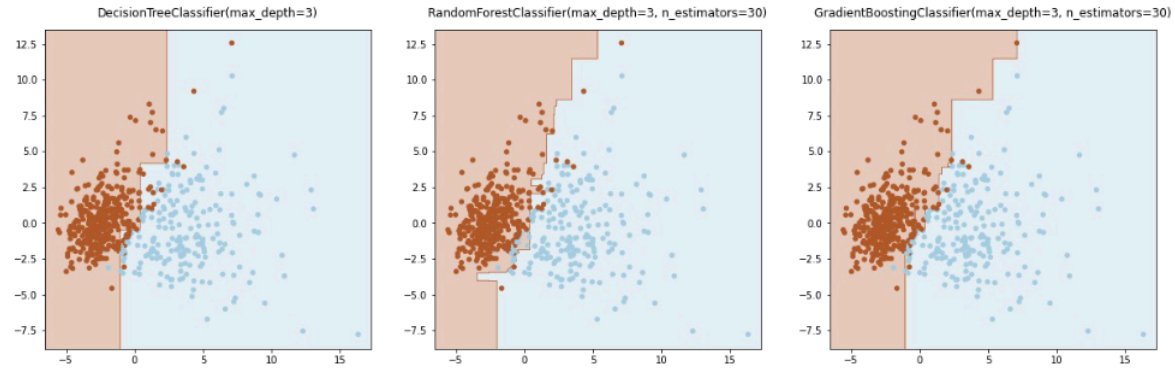
8.9.2. Boosting

Gradually improves a simple model by correcting its errors

Gradient Boosting

- At training step m = 1,...,M model Version F_{m+1} is an imporvement of F_m
- A term h_m is added to F_m that is trained to correct the erros of F_m
- $F_{m+1}(x) = F_m(x) + h_m(x)$

Decision Boundarys for Regular Decisions Trees, Random Forests and Gradient Boosting:



9. Neural Networks

9.1. Perceptron

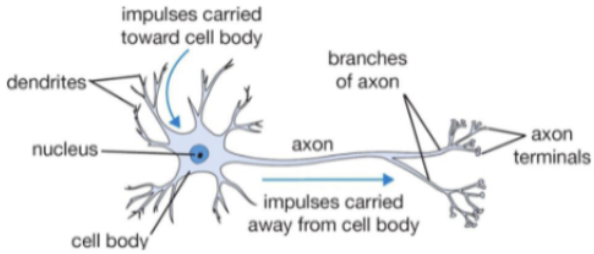
9.1.1. Neurons and the Brain

Neuron:

- electrically excitable cell
- communicates with other cells
- communication via synapses

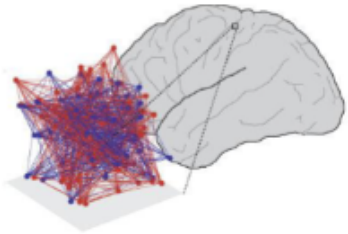
consists of:

- **Dendrite**: receives signals from other neurons
- **Soma** (cell body): processes the information
- **Axon**: Transmits the output of this neuron
- **Synapse**: Point of connection to other neurons



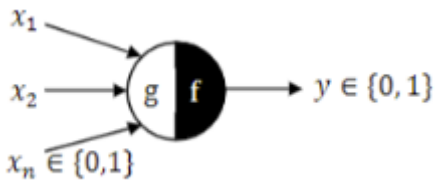
9.1.2. Neurons Fire when Activated

- human brain: 86 billion neurons form 100 trillion connections to each other
- Neurons receive input **signals** via their dendrites and send output signals down the axon.
- At axon terminals the neuron can **transmit a signal** across the synapse to another cell.
- The signaling process is partly **electrical** and partly **chemical**.
- **action potential:**
  - if voltage chance have a large enough amount over a short interval
  - neuron generates an all-or-nothing electrochemical pulse (= action potential)
- action potential travels rapidly along the axes and actives more synaptic connections



9.1.3. MP-Neuron

- McCulloch & Pitts Neuron
- first computational model of a neuron



- Two Parts g & f
- g: Takes an input and performs an aggregation
- f: makes a decision on the value

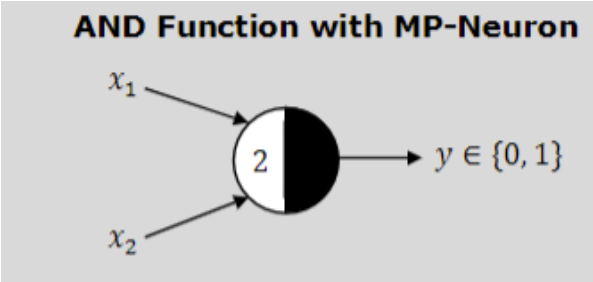
$$g(x_1, x_2, ..., x_n) = g(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$$

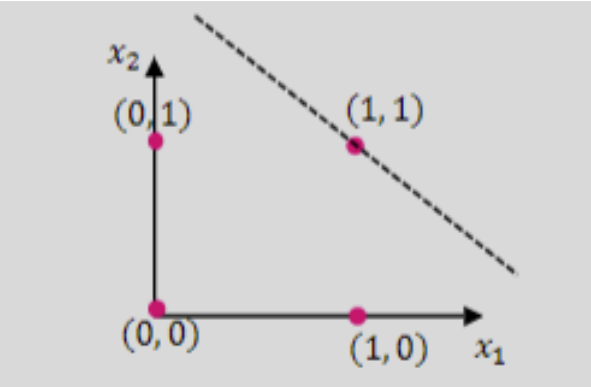
$$\begin{aligned} y = f(g(\boldsymbol{x})) &= 1 \quad \text{if } g(\boldsymbol{x}) \geq \theta \\ &= 0 \quad \text{if } g(\boldsymbol{x}) < \theta \end{aligned}$$

- θ: thresholding parameter

9.1.3.1. AND Function

MP-Neuron can be used to model a AND Function:





$$x_1 + x_2 = \sum_{\{i=1\}}^{\{2\}} x_i = 2$$

9.1.4. Rosenblatt's Perceptron

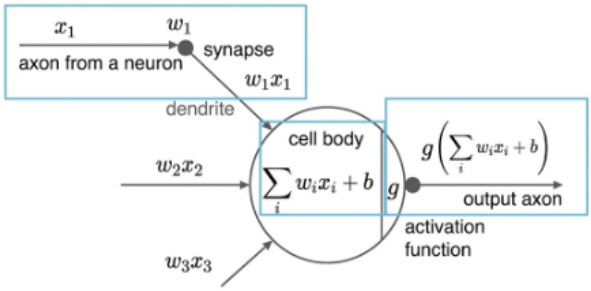
- learning algorithm that adjust **weights automatically** (unlike MP neuron)
- weights are based on **classification errors**
- perceptron are enabled to learn from data **iteratively**
- limited to linearly separable data (MP neuron as well)

9.1.5. Early Neural Networks and Deep Learning Timeline

Year	Milestone / Key Figures	Description & Significance
1960	*Widrow & Hoff: ADALINE*	Introduction of a „nicely differentiable" neuron model (*Adaptive Linear Neuron*).
1969	*Minsky & Papert: „Perceptrons"*	Publication of the influential book highlighting the computational limitations of simple perceptrons.
1970s+	*The first „AI Winter"*	A period where neural research was seen as a „dead end." The eventual solution was the widespread adoption of backpropagation.
Multiple	*Backpropagation Concept*	The mathematical foundation of backpropagation was independently formulated many times by different researchers.
1986	*Rumelhart & Hinton*	Independently formulated backpropagation and published a landmark paper showing that it actually works in practice.

9.1.6. Perceptron as a Logic Unit

- **Inputs** x_i are scaled by **weights** w_i
- **Bias** b used for extra tuning of activation layer
- **Activation** is Boolean TRUE if threshold is exceeded



$g(z)$: **Activation Function**

- Produces **activation** a

9.1.7. A Simple Analogy

Question: Should I go to the beach festival?

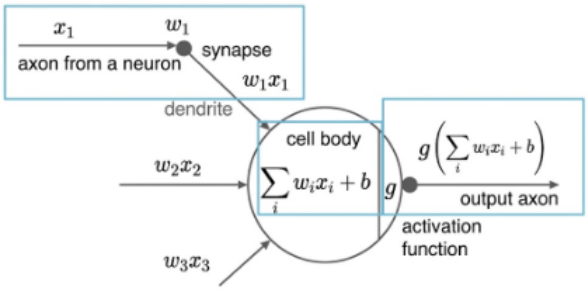
1. Is the weather good? x_1
2. Does my friend want to come? x_2
3. Is there public transport? x_3

The three factors must be weighted: w_1, w_2, w_3

Answer: $a \in \{\text{Yes}, \text{No}\}$

- Yes: $x_1 w_{11} + x_2 w_{12} + x_3 w_{13} \geq (-b)$
- No: $x_1 w_{11} + x_2 w_{12} + x_3 w_{13} < (-b)$

$a = \begin{cases} 1 \text{ if } b+x_1 w_1+x_2 w_2+x_3 w_3 \geq 0 \\ 0 \text{ if } b+x_1 w_1+x_2 w_2+x_3 w_3 < 0 \end{cases}$



9.1.8. The Activation Function

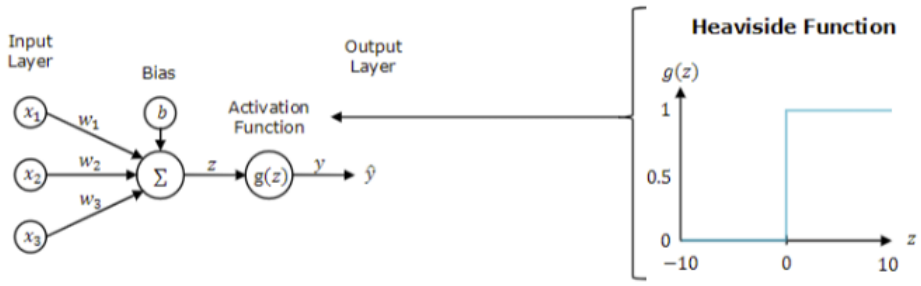
Different Activiation Functions can be choosen

- Logistic Function (Sigmoid)
- Softmax Function

9.1.8.1. Heaviside

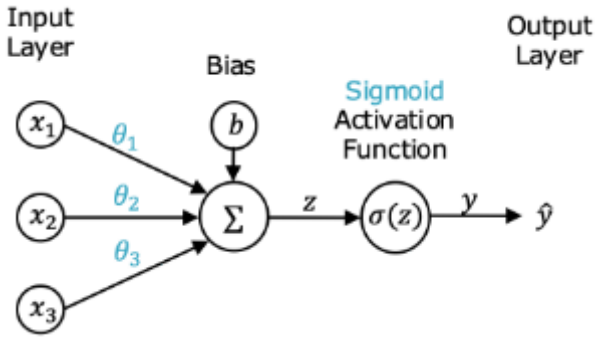
- used by Rosenblatt & previous analogy

$g(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ if } z < 0 \end{cases}$



9.1.8.2. Sigmoid

- rename weight w_i to θ_i
- replace threshold with sigmoid activation $\sigma(z)$
- can be trained with gradient decent



Logistic Regression

$$\boldsymbol{X\theta} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$\hat{Y} = g(\boldsymbol{X\theta}) = \frac{1}{1 + e^{-\boldsymbol{X\theta}}}$$

The Preceptron

$$z = b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

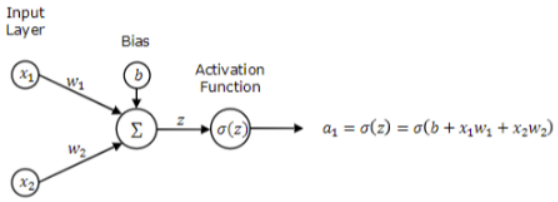
$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

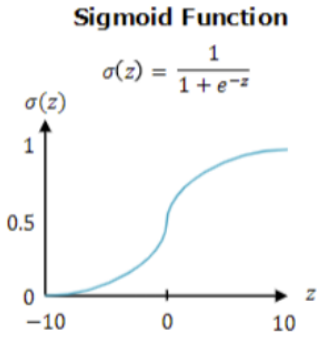
9.1.9. Perceptron with Sigmoid

DEFINITION: AND function

9.1.9.1. Example

- single-layer perceptron: 2-input x 1-output
- with sigmoid activation



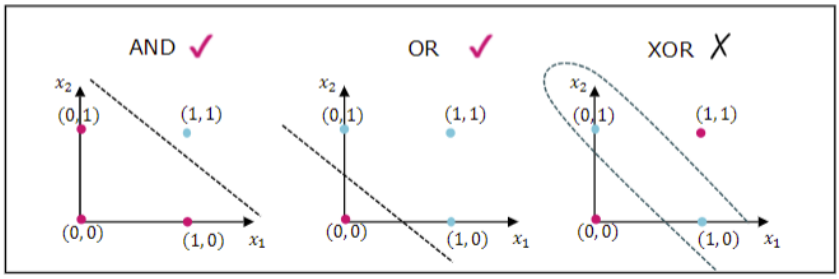


Suppose: b= $-30, w_1 = 20, w_2 = 20$

x_1	x_2	z	$a_1 = \sigma(z)$
0	0	$-30 + 0 + 0 = -30$	~ 0
0	1	$-30 + 0 + 20 = -10$	~ 0
1	0	$-30 + 20 + 0 = -10$	~ 0
1	1	$-30 + 20 + 20 = +10$	~ 1

9.1.10. Linear vs. Non-Linear Models

- perceptron can implement the AND function
- exclusive OR (XOR) function cannot be implemented
- Generally: a **single layer network cannot implement non-lineare models**

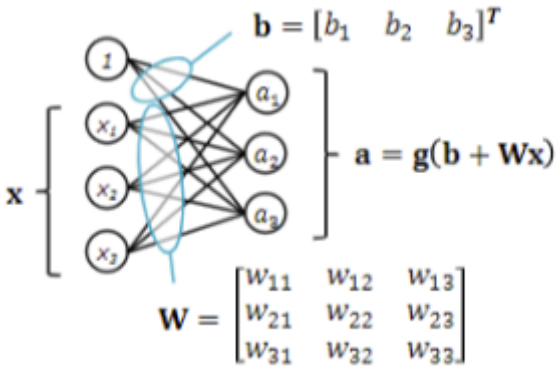


9.1.11. Summary

- Modelled on real behaviour and physiology of neurons in the brain.
- fundamental unit of learning
- very simple to implement
- training of date to find b and w_i
  - same way as logistic regression: gradient descent
- connection of perceptrons → artificial neural networks (ANNs)

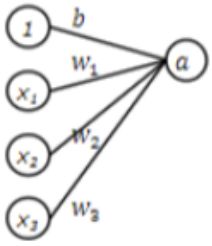
9.2. Multilayer Perceptron (MLP)

9.2.1. Single Layer Neural Network



- can generalize to any M x N size
- can implement any linear separable classification problem
- if $g(z)$ is the logistic function
  - each output a_j can be interpreted as one-vs-all classification

9.2.1.1. Single Neuron (Perceptron)



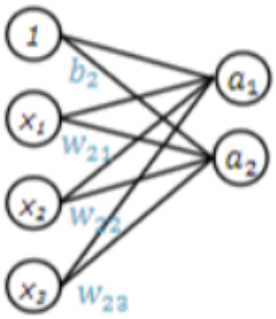
$a = g(b + x_1 w_1 + x_2 w_2 + x_3 w_3)$

vectorize:

- $\boldsymbol{w} = [w_1 \quad w_2 \quad w_3]^T$
- $\boldsymbol{x} = [x_1 \quad x_2 \quad x_3]^T$

$\rightarrow$ **feed-forward network** (data flows left-to-right)

9.2.1.2. Add a Second Neuron

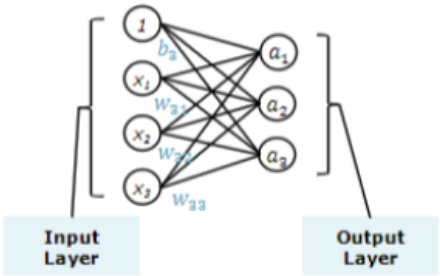


$$a_1 = g(b_1 + x_1 w_{11} + x_2 w_{12} + x_3 w_{13})$$
$$a_3 = g(b_1 + x_1 w_{21} + x_2 w_{22} + x_3 w_{23})$$

Vectorize again:
- $\boldsymbol{a} = [a_1 \quad a_2]^T$ (output)
- $\boldsymbol{b} = [b_1 \quad b_2]^T$ (bias)
- $\boldsymbol{W}|_{ij} = W_{ij}$ (weights)
- $\boldsymbol{g}$ is element-wise function g

9.2.1.3. Add a Third Neuron

artificial neural network (ANN), 3-input x 3-output

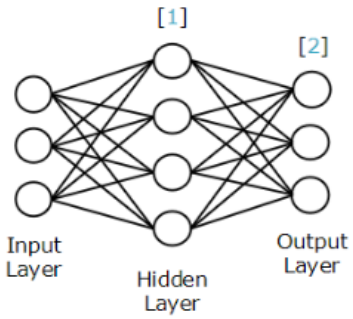


$$a_1 = g(b_1 + x_1 w_{11} + x_2 w_{12} + x_3 w_{13})$$
$$a_2 = g(b_2 + x_1 w_{21} + x_2 w_{22} + x_3 w_{23})$$
$$a_3 = g(b_3 + x_1 w_{31} + x_2 w_{32} + x_3 w_{33})$$

Vectorize:
$\boldsymbol{a} = \boldsymbol{g}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x})$

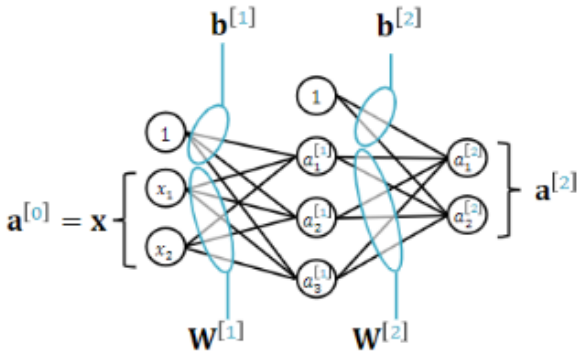
9.2.2. Hidden Layers

Layer of neurons (or nodes) situated between the input layer and the output layer



Naming Convention: Picture above is a **two-layer network**
- Layer `[1]` & Layer `[2]`
- Input Layers are not counted

9.2.2.1. Creating New Features



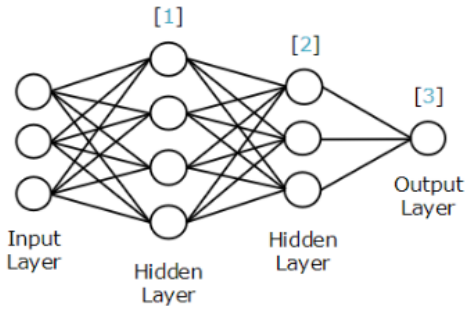
- By convention we rename input x to $\boldsymbol{a}^{[1]}$

$$\boldsymbol{a}^{[2]} = \boldsymbol{g}\big(\boldsymbol{b}^{[2]} + \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]}\big)$$
$$\boldsymbol{a}^{[1]} = \boldsymbol{g}\big(\boldsymbol{b}^{[1]} + \boldsymbol{W}^{[1]}\boldsymbol{a}^{[0]}\big)$$

$\rightarrow$ created **new feature** $\boldsymbol{a}^{[1]}$

9.2.2.2. Multiple Hidden Layers

It is also possible to have multiple hidden layers.



- 3-layer Network with 2 hidden layers
- Hidden Layer: 1: 4 units / neurons
- Hidden Layer 2: 3 units / neurons

9.2.2.3. Feed-Forward Network

DEFINITION: A Network where Signals go in one direction only, left to right.

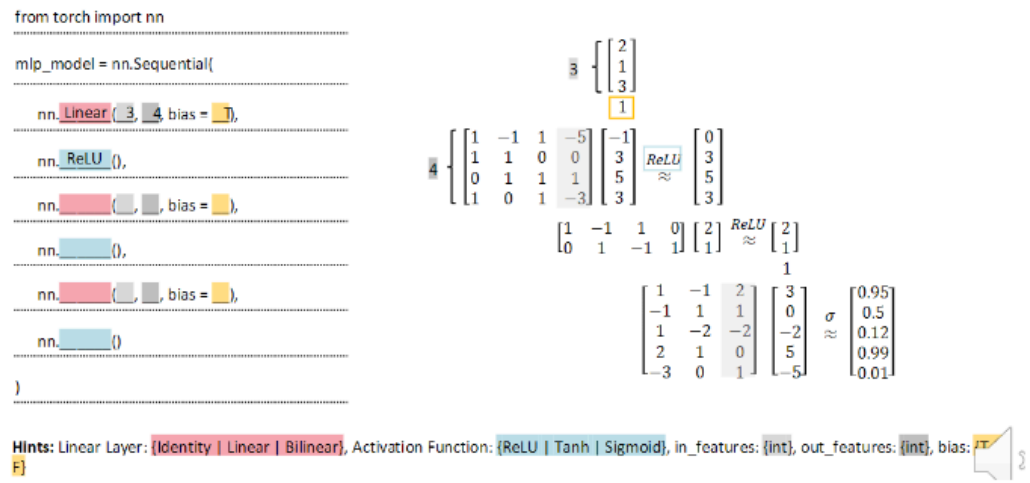
9.2.3. Implementation

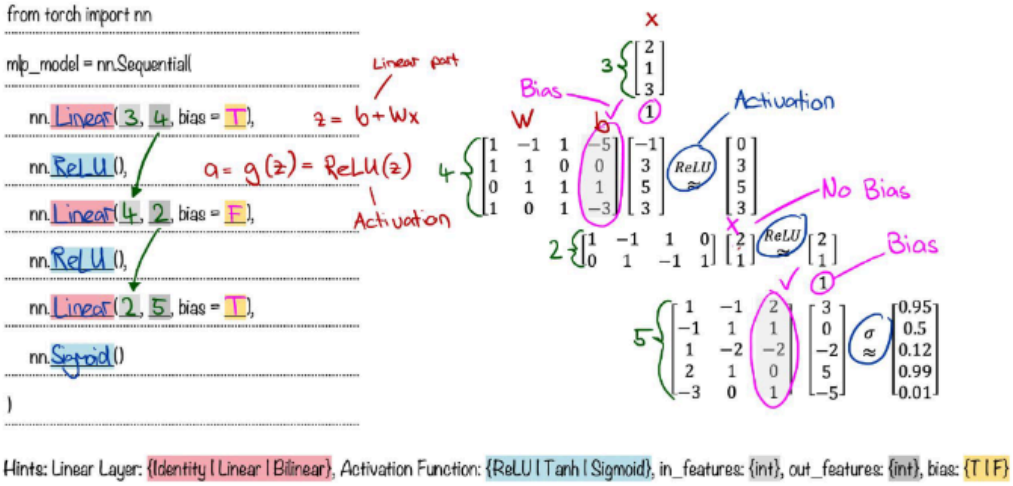
- Feed-forward multiple layer networks are very easy to implement

```python
import numpy as np

# Forward-pass of a 3-layer neural network
# Initialize random weights and biases
W1, b1 = np.random.randn(4, 3), np.random.randn(4, 1)
W2, b2 = np.random.randn(3, 4), np.random.randn(3, 1)
W3, b3 = np.random.randn(1, 3), np.random.randn(1, 1)

f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (3x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- possible to solve arbitrary classification problems (with the right training data)
- single hidden layer: possible to approximate any continuous function
- Note: training is harder

9.2.3.1. PyTorch



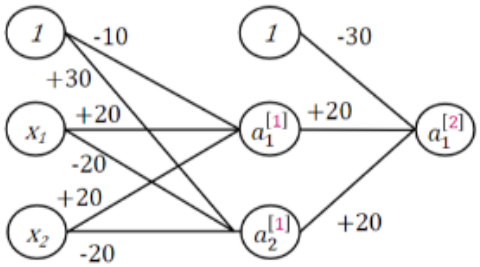


9.2.4. XOR Function

- XOR (Exclusive OR) cannot be implemented with a single layer ANN
- can be implemented with a **multi-layer feed forward neural network**

Example:

- 2-layer network
- 3 hidden units



x_1	x_2	$z_1^{[1]}$	$z_2^{[1]}$	$a_1^{[1]}$	$a_2^{[1]}$	$z_1^{[2]}$	$a_1^{[1]}$
0	0	-10	+30	0	1	-10	0
0	1	+10	+10	1	1	+10	1
1	0	+10	+10	1	1	+10	1
1	1	-10	-10	1	0	-10	0

Non-linear classification boundaries can be implemented with hidden layers.

9.2.5. Universal Approximators

Any continuous function $f : [0, 1]^n \rightarrow [0, 1]$ can be approximated by a neural network with at least 1 hidden layer

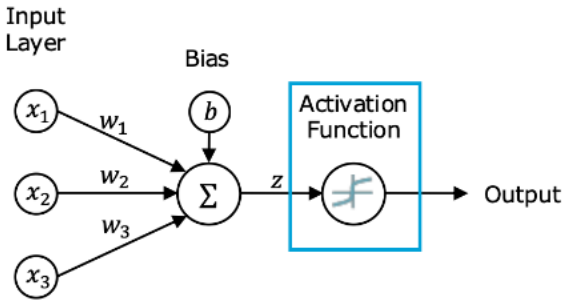
Question to think about: Does this theorem increase or decrease the risk that our model will overfit?

9.2.6. Summary

- in neural networks **artificial neurons are connected**
- learn from **training data**
- easy to implement
- massive **parallelism** makes them very efficient (matrix operations)
- if large enough → implement any non-linear classifier
- highly **fault**- and **noise-tolerant**
- **feed forward neural networks can solve very complex problems**

9.3. Activation Functions

non-linear activation function to implement useful logic

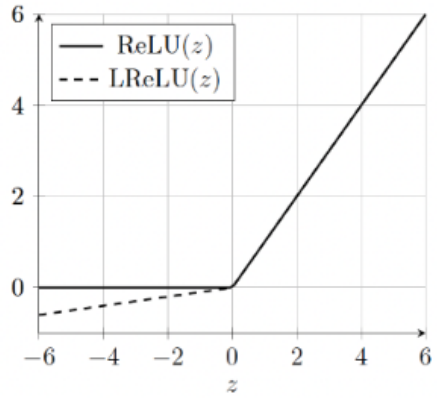


9.3.1. Choosing an Activation Function

- activations between $0l = al = 1$ for probability-based classification.
- preference for **continuous and differentiable functions** to **support gradient descent**.
- **avoidance of flat functions to prevent „vanishing gradients“** and training delays.
- monotonic functions guarantee convex error surfaces in single-layer models.

9.3.2. Rectified Linear Unit (ReLu)

- piecewise linear function
- **output is positive** or zero if output would be negative
- alleviates the vanishing gradient problem
- default activation function for many neural networks
- but, **dying units problem**, because the gradient is zero for negative values
- **good choice for hidden layer activation**

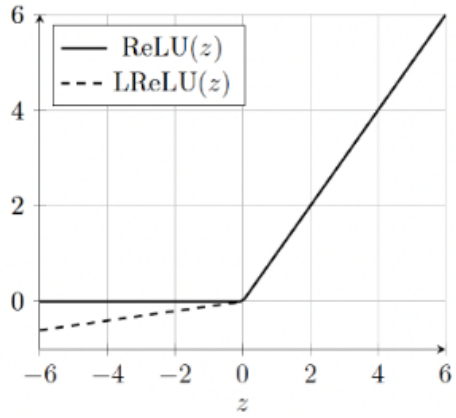


$$\text{ReLU}(z) = \max(0, z)$$

$$\text{ReLU}'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$

9.3.3. Leaky ReLU

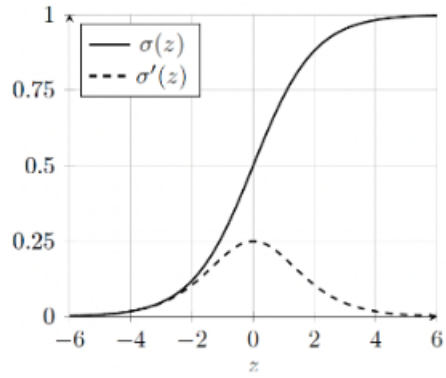
- solves dying units problem
- and vanishing gradient problem
- **non-zero activation** for negative values
- good performance
- but results **are not always consistent**
- **good choice for hidden layer activation**



$$\text{Leaky ReLU}(z) = \max(0.01z, z)$$

$$\text{Leaky ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0.01 & \text{otherwise} \end{cases}$$

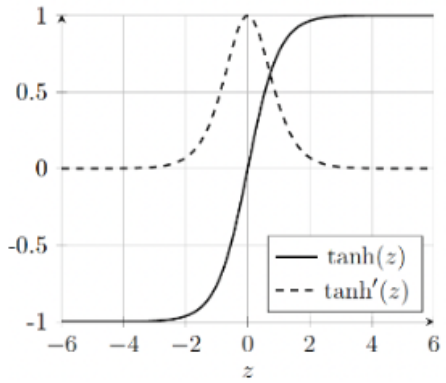
9.3.4. Sigmoid σ



$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

9.3.5. Tanh



$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = 2\sigma(2z) - 1$$

$$\tanh'(z) = 1 - \tanh^2(z)$$

9.3.6. Softmax for Multi-Label Classification

- softmax activation at the output layer
- to get a **probability of belonging to each class**
  - It **normalizes** the output of a network to a **probability distribution**.
  - **sum** of all probabilities across classes should be **1** (100%)
- softmax is a generalization of the logistic function to multiple dimensions
- used in multinomial logistic regression
- often used as the **last activation function** of neural network
- inputs called **logits**

$$\text{softmax}_j(z) = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

- Takes a vector of K real values known as **logits** (z_k).
- **Inputs**: positive, negative, zero, or greater than one.
- **output**: probabilities

9.3.7. Recommendation

- **ReLU** for hidden layers
- **Sigmoid** for binary classification in last layer
- **softmax** for multi-class classification in last layer

9.4. Cost Functions

DEFINITION: measures how poorly our model performs (depends on the actual problem)

Regression

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Mean Abs. % Error (MAPE)

Binary Classification

- Binary Cross Entropy (BCE)
- Hinge Loss
- Squared Hinge Loss

Multi Classification

- Multilabel Cross Entropy
- Kullback-Leibler Divergence

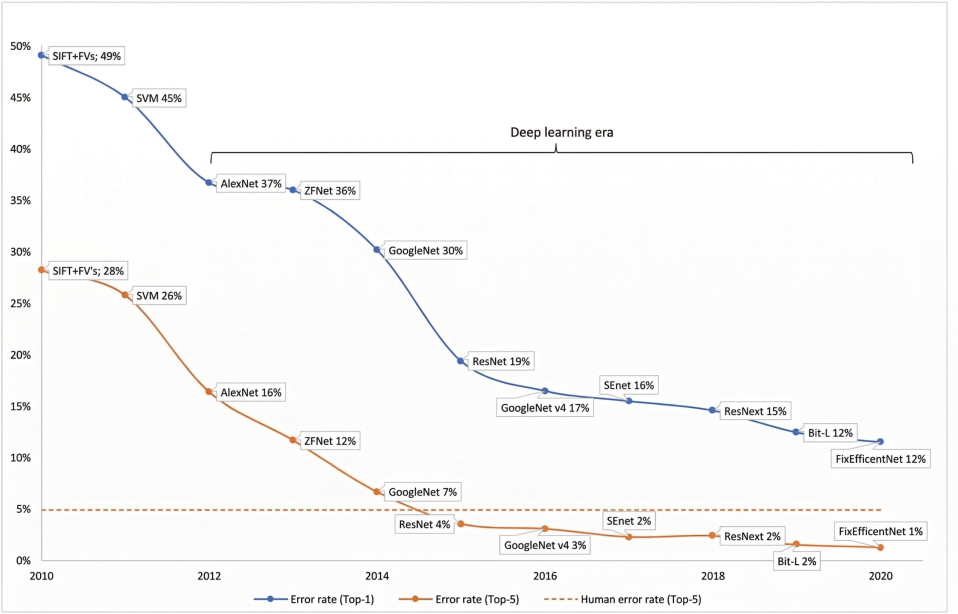
9.4.1. Recommendation

- **MSE** for linear regression
- **cross entropy** for binary classification
- **multilabel cross entropy** for multi-class classification

10. Convolutional Neutral Nets (CNN)

10.1. Motivation

- Distinguishing specific categories in images is a challenge even for humans
- Since 2010, the ImageNet challenge error rates have plummeted, with deep learning eventually surpassing human accuracy
- An image really consists of RGB color channels or grayscale values



10.1.1. Building a Vanilla Neural Network

Processing images with standard Neural Networks leads to a parameter explosion:

- A 224x224 image creates > 150k input values
- Connecting these to a typical 1024-neuron hidden layer yields > 150 million weights in just the first layer
- **Deep networks with such massive parameter counts are nearly impossible to train**

10.2. The Old Times: Manual Feature Engineering

Before CNNs, problems were tackled using manual feature engineering:

- It transforms unstructured data into structured data
- Performance is only as good as the human-selected filters (error rates never got better than 25%)
- It needs a complicated workflow to process many pixels at once

t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18	t19
0.000625	0.0175	0.00125	0.04375	0.00375	0.001875	0.013125	0.055625	0.00125	0.003125	0.008125	0.00625	0	0.03125	0.00125	0.001875	0.01875	0.06375	0.040625	0.00625
0	0.051875	0.024375	0.0225	0.008125	0.001875	0.038125	0.124375	0.010625	0.003125	0.035	0.0275	0.0075	0.06	0.015625	0.014375	0.014375	0.085625	0.108125	0.0075
0.0175	0.04	0.03125	0.005625	0.00875	0.001875	0.009375	0.12375	0.008125	0.005	0.024375	0.01	0.001875	0.05125	0.001875	0.021875	0.01625	0.050625	0.106875	0.02
0.11625	0.011875	0.016875	0.009375	0.001875	0.00125	0.024375	0.000625	0.0175	0.009375	0.010625	0.016875	0.01375	0.00875	0.014375	0.018125	0.0225	0.035625	0.004375	0.0125
0.066875	0.034375	0.035625	0.00375	0.006875	0.000625	0.025	0.06875	0.01				0.0125	0.02875	0.014375	0.026875	0.02125	0.049375	0.0675	0.01125
0.205625	0	0.011875	0.025	0.005	0.000625	0	0	0				0.010625	0	0.004375	0.046875	0.016875	0.01375	0.0025	0.03125
0.005	0.003125	0.10375	0.03375	0.031875	0	0.0075	0.009375	0.001				0.00125	0	0.013125	0.021875	0.003125	0.1025	0.08	0.029375
0.056875	0	0.0025	0	0.001875	0	0	0					0	0	0	0.06625	0	0	0	0.060625
0	0.0475	0.02125	0.09875	0.058125	0	0.0125	0.219375	0.003				0.01375	0.0375	0.036875	0.00625	0.015	0.016875	0.085625	0.001875
0.000625	0.0475	0.045	0.064375	0.02625	0	0.01	0.16875	0.00				0.0025	0.045625	0.009375	0.01	0.003125	0.05375	0.098125	0.0025
0	0.145	0.00375	0	0.065625	0	0.0025	0.25125					0	0.020625	0.0025	0	0.02625	0.0525	0.230625	0
0.023125	0.020625	0.0075	0.00125	0.0175	0.0025	0.014375	0.041875	0.0				0.0025	0.018125	0.01875	0.023125	0.0275	0.00125	0.06125	0.000625
0	0.15	0	0	0.0775	0	0	0.39					0	0.05	0.000625	0	0	0.008125	0.1475	0
0.01125	0.009375	0.004375	0.00125	0.025625	0	0.01	0.040625	0.016				0	0.015625	0.009375	0.021875	0.03375	0.00125	0.03625	0
0.001875	0.104375	0.014375	0	0.084375	0	0.00125	0.28125					0	0.060625	0.008125	0.02375	0	0.01625	0.1075	0
0.006875	0.0075	0.015625	0.006875	0.015	0	0.005	0.07	0.006				0	0.00375	0.009375	0.021875	0.0125	0.01125	0.06625	0.004375
0.016875	0.028125	0.01125	0.006875	0.02375	0	0.005	0.139375	0.005625	0	0.006875	0.005	0.003125	0.050625	0.010625	0.020625	0.00625	0.01125	0.04	0.004375
0.020625	0.045	0.036875	0	0.005625	0.003125	0	0.13625	0.006875	0.00875	0.001875	0.004375	0	0.056875	0	0.025	0.039375	0.0425	0.115625	0.019375
0.021875	0.04375	0.010625	0	0.05	0	0.000625	0.190625	0.00125	0	0.0025	0.005	0.00625	0.0975	0.008125	0.045	0	0.005	0.08625	0
0.016875	0.03	0.026875	0	0.1075	0	0.000625	0.128125	0.000625	0	0.010625	0.006875	0.003125	0.08375	0.0175	0.048125	0	0.01625	0.10375	0.00375
0	0	0.114375	0	0.00625	0	0	0.001875	0	0	0	0	0	0	0	0.2475	0.003125	0.21625	0.191875	0
0	0	0.039375	0	0.054375	0	0	0.075	0	0	0	0	0	0.074375	0	0.209375	0.003125	0.0375	0.1575	0.001875

Abbildung 1: Transforming an unstructured image into structured data arrays for manual feature engineering

10.2.1. Convolutional Neural Networks

CNNs automatically extract hierarchical features from raw pixels.

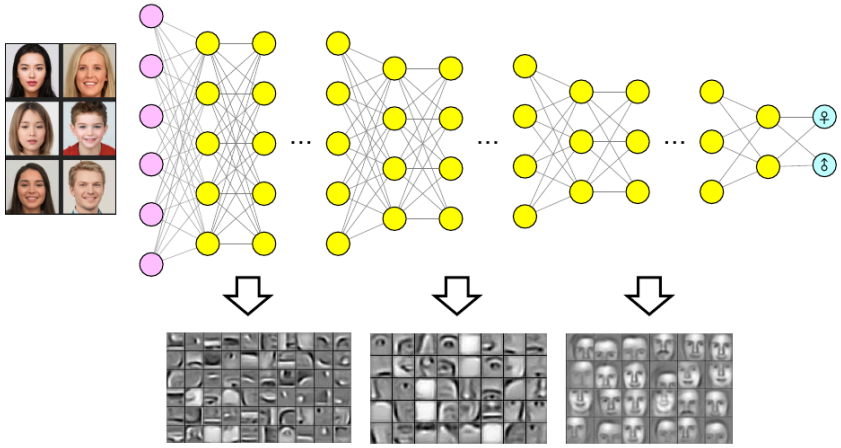


Abbildung 2: CNNs as feature extractors

10.2.1.1. Computer Vision Disciplines

- **Classification + Localization:** Object and bounding box
- **Object Detection:** Multiple objects and bounding boxes
- **Semantic Segmentation:** Pixel-level class assignment
- **Instance Segmentation:** Pixel-level individual object separation

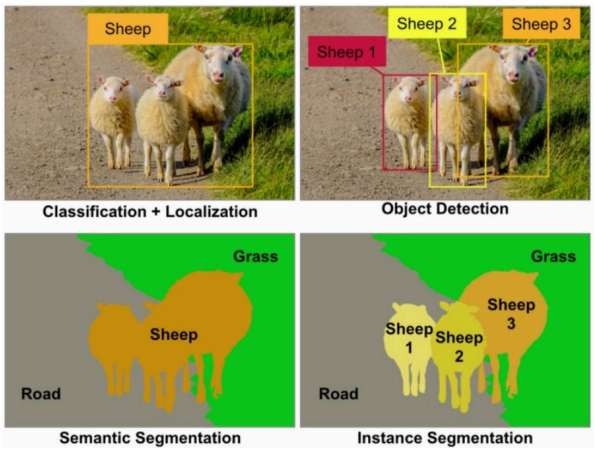


Abbildung 3: Computer Vision Disciplines

10.3. Architecture

10.3.1. Convolution & Pooling

DEFINITION: Convolution extracts local features using filters, while pooling reduces the image dimensions to compress information and retain only the most prominent features

10.3.1.1. Filter Matrices

- Pixels are processed together with their neighbors to capture the local context
- The mathematical operation multiplies pixel values and filter values element-wise and sums them up
- An activation function is sometimes applied to the final result (VGG16)

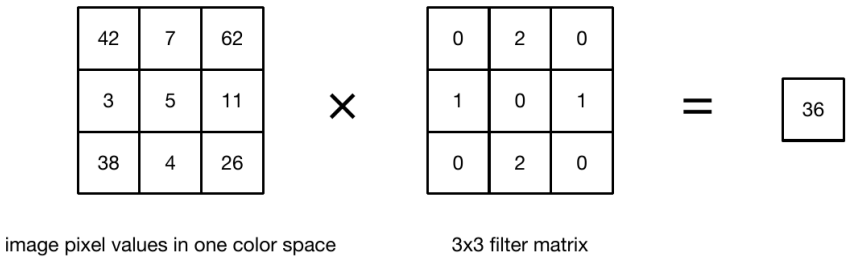


Abbildung 4: $7 * 2 + 3 * 1 + 11 * 1 + 4 * 2 = 36$

10.3.1.2. Stride

DEFINITION: Stride defines how many pixels the filter moves across the image at each step

- A stride of 1 means shifting the filter by exactly one pixel
- The convolution operation naturally shrinks the target image dimensions

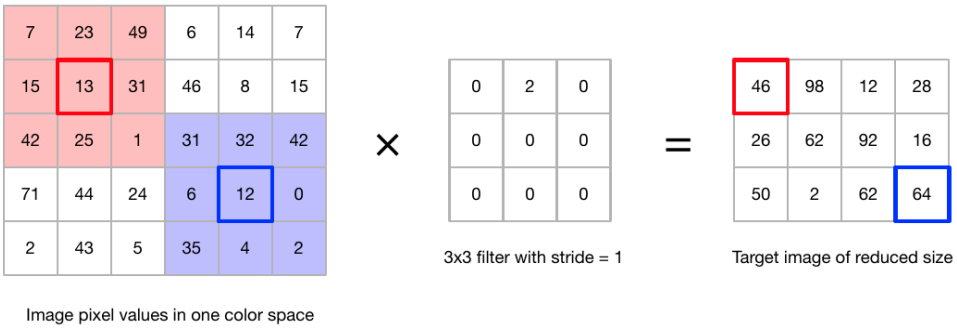


Abbildung 5: Convolution with a 3x3 filter and stride of 1, showing the reduction in image size

10.3.1.3. Padding

- Padding adds a **border of zeros** around the input image
- Image retain its original dimensions after the convolution
- The final size of the output depends on the input image size, padding, and stride

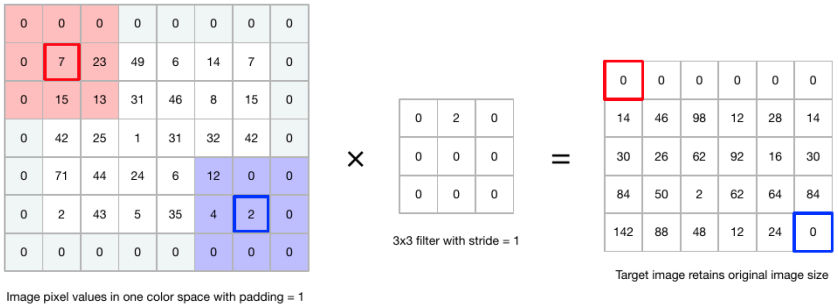
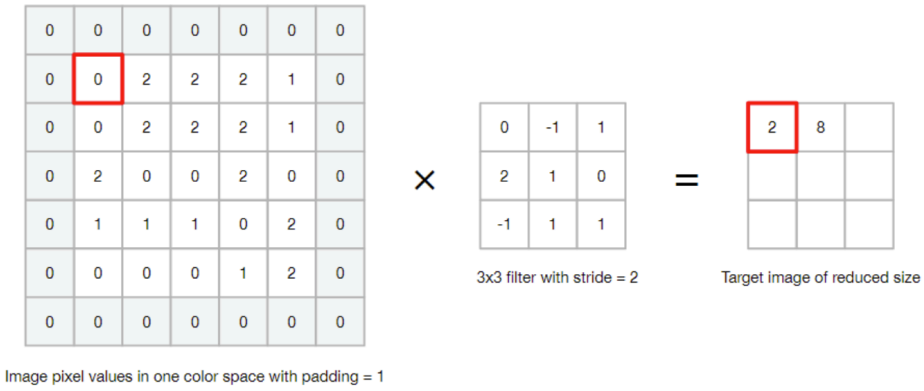


Abbildung 6: Convolution with padding of 1, allowing the target image to retain the original size

10.3.1.4. Convolution with Padding and Stride

- Combining padding and a stride greater than 1 will still cause the target image to shrink
- A stride of 2 requires even more padding to maintain the original image dimensions

Example:



$$\text{Output Size} = \frac{\text{InputSize} - \text{FilterSize} + 2 * \text{Padding}}{\text{Stride}} + 1$$
$$= \frac{5 - 3 + 2 * 1}{2} + 1 = 3$$

- To ensure the target image retrains the original input size:

$$\text{Padding} = \frac{\text{Stride} * (\text{InputSize} - 1) - \text{InputSize} + \text{FilterSize}}{2}$$
$$= \frac{2 * (5 - 1) - 5 + 3}{2} = \frac{6}{2} = 3$$

- You would need a padding of 3 on all sides to maintain the 5x5 resolution.

10.3.1.5. Convolutions as Feature Detectors

- Convolutions act as image filters to extract specific characteristics
- Different filter matrices are designed to detect various patterns, such as horizontal or vertical lines
- A classic example is the Sobel filter, which is specifically used for edge detection

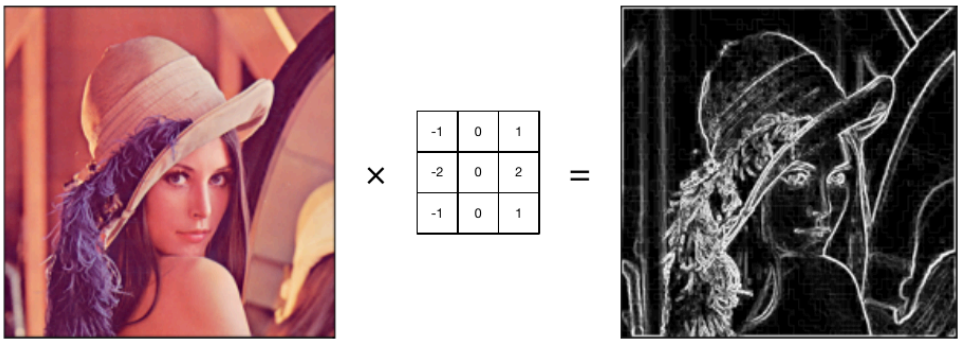


Abbildung 7: Applying a Sobel filter to an image to perform edge detection

10.3.1.6. 1D CNN

Instead of manually defining filter matrices, a cnn learns the optimal filter weights (e.g., w_1, w_2, w_3) during the training process.

Abbildung 8: A 1D convolution where the filter values are represented as learnable weights

10.3.1.6.1. Weight Sharing Concept

- A convolutional layer acts as a neural network where neurons share the exact same weights and bias
- This concept significantly reduces the total amount of learnable parameters
- **Example**: A 1D filter of size 3 only requires 4 parameters (3 weights and 1 bias)

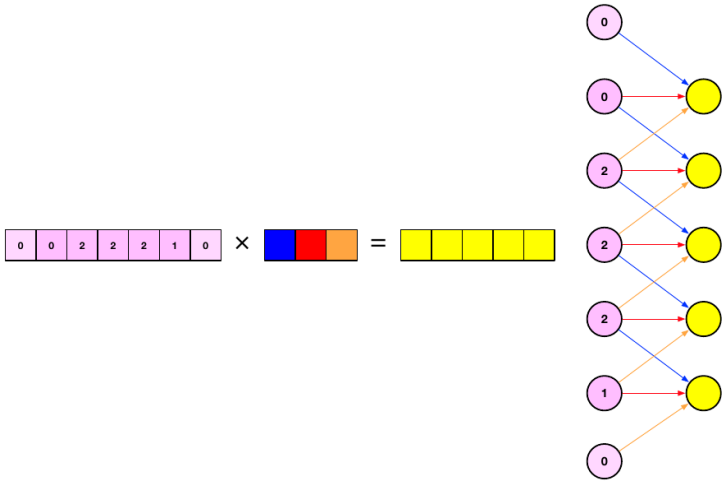


Abbildung 9: 1D Convolution represented as a neural network illustrating weight sharing

10.3.1.6.2. Stacked 1D Convolutions

- Layers are stacked to learn complex features from simpler ones
- Example below
  - All layers have filter size = 3 and stride = 1
  - Input layer and first convolutional layer use padding
  - Afterwards, image size reduces to 3 pixels

Parameter Calculation:

- Weight sharing limits each layer to 4 parameters (3 weights + 1 bias)
- Total parameters for 3 layers: layers $\times$ parameters $= 3 \times 4 = 12$

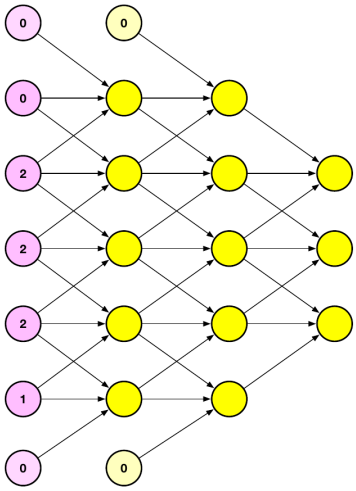


Abbildung 10: Stacked 1D convolutional layers with parameter sharing

10.3.1.6.3. Parameter Calculation for a Full Network

The total parameters result from the sum of weights and biases across different layer types:

- **Convolutional Layers (yellow):** Use weight sharing (3 weights $+ 1$ bias per filter)
- **Fully Connected Layer (green):** Unique weights for all connections (3×3 weights $+ 3$ biases)
- **Final Output Layer (blue):** Connects features to prediction (3×1 weight $+ 1$ bias)

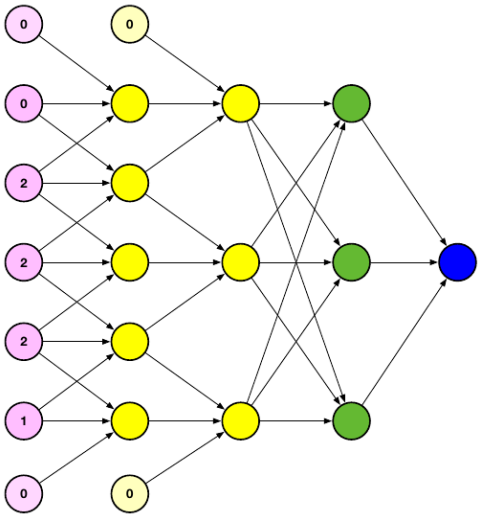


Abbildung 11: Full network combining convolutional, fully connected and output layers

$$\text{Total} = \underbrace{2 \times (3+1)}_{\text{Conv Layers}} + \underbrace{(3 \times 3 + 3)}_{\text{FC Layer}} + \underbrace{(3 \times 1 + 1)}_{\text{Output Layer}} = 24$$

10.3.1.6.4. Multiple Filters in Parallel

A convolutional layer can apply multiple filters simultaneously to extract different features in one pass.

Example:

- **Depth:** Each filter creates its own channel in the output. 8 filters result in an output depth of 8.
- **Calculation:** Applying 8 filters (3×3) to a 28×28 image (no padding) results in a $26 \times 26 \times 8$ tensor.

Parameters:

- Each filter has its own weights and bias: $(9 + 1) = 10$ parameters.
- Total for 8 filters: $8 \times 10 = 80$ parameters.

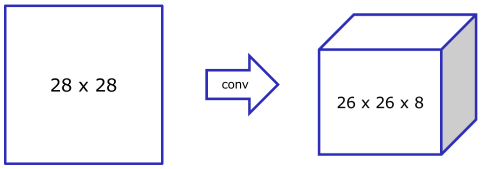


Abbildung 12: Applying 8 filters increases the depth of the output tensor

10.3.1.7. Tensors and Depth Consistency

- **Definitions:** 1D = vector, 2D = matrix, 3D+ = tensor.
- **Consistency:** A filter's depth must match the input's depth. If the input has 8 channels, the filter must be $3 \times 3 \times 8$.
- **Reduction:** One 3D filter slides across the input to produce a single **2D output map**.

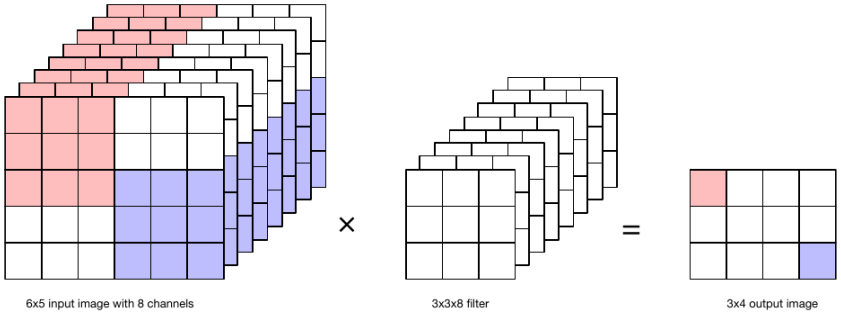


Abbildung 13: A 3x3x8 filter processing an 8-channel input

10.3.1.7.1. Stacked Convolutions with Multiple Filters

- **Layer 1:** Uses 2 parallel filters (size 3), producing 2 output channels.
  - **Parameters**: $(3 \text{ weights} + 1 \text{ bias}) \times 2 \text{ filters} = 8$
- **Layer 2:** Matches input depth with 2-channel filters. 1 filter merges channels back to 1 image.
  - **Parameters**: $(3 \times 2 \text{ weights} + 1 \text{ bias}) \times 1 \text{ filter} = 7$

Total Parameters: $8 + 7 = 15$

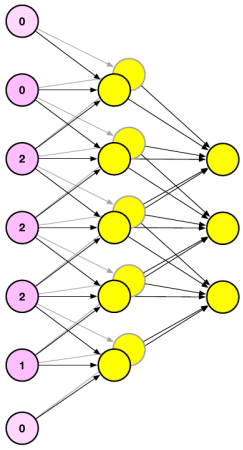


Abbildung 14: Channel expansion in Layer 1 and merging in Layer 2

10.3.1.8. Pooling

DEFINITION: Pooling reduces width and height to decrease redundancy and save computation, allowing the network to „zoom out“ for better feature detection.

- **Operation:** Summarizes a pixel neighborhood using a function like max, min, or average.
- **Max-Pooling:** Selects the maximum value from a window (e.g., 3×3). This preserves dominant features and reduces noise.
- **Size Reduction:** Width and height are divided by the „Pool Size“.
  - **Example:** A $26 \times 26 \times 8$ tensor with pool size 2 becomes $13 \times 13 \times 8$.
- **Channel Consistency:** Pooling does **not** change the number of channels (depth).

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} - \text{Pool Size}}{\text{Stride}} \right\rfloor + 1 = \left\lfloor \frac{6-3}{3} \right\rfloor + 1 = 2$$

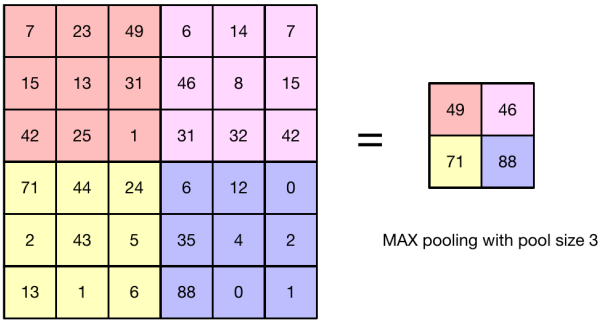


Abbildung 15: Max-pooling reduces dimensions while preserving depth

10.3.2. CNN Architecture

10.3.2.1. Softmax

A Convolutional Neural Network (CNN) typically ends with a classification head to map extracted features to specific classes.

- **CNN Sequence:** Interleaved convolutional layers (feature extraction) and pooling layers (feature abstraction/reduction).
- **Classification Head:** After the final pooling, a fully connected (dense) layer is attached to map the 3D features to a 1D output vector.
- **Softmax Function:** Converts the raw output values into a probability distribution.
  - Each value is between 0 and 1.
  - The sum of all outputs equals 1 (100%).
  - It quantifies the network's confidence for each class (e.g., digits 0-9).

Example:

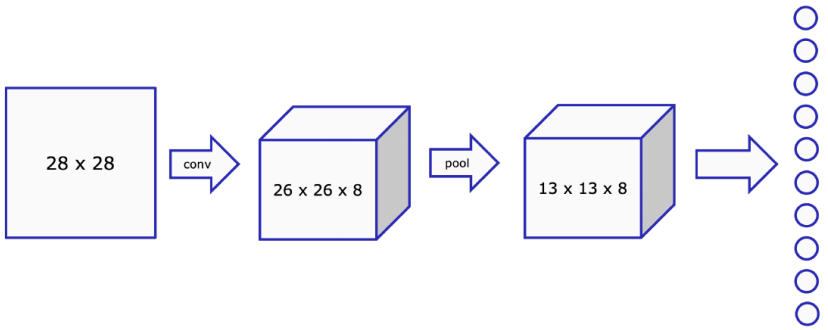


Abbildung 16: CNN architecture ending in a fully connected layer with Softmax activation

$$\text{Output} = \left\lfloor \frac{\text{Input} - \text{Kernel} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

- **Convolutional Layer:**
  - „Kernel" represents the **Filter Size**

$$\left\lfloor \frac{28 - 3 + 2 \times 0}{1} \right\rfloor + 1 = \mathbf{26}$$

  (Depth is determined by the number of filters, here **8**)

- **Pooling Layer:**
  - „Kernel" represents the **Pool Size**

$$\left\lfloor \frac{26 - 2 + 2 \times 0}{2} \right\rfloor + 1 = \mathbf{13}$$

  (Depth remains unchanged at **8**)

- **Flattening:**

$$\text{Neurons} = \text{Width} \times \text{Height} \times \text{Depth} = 13 \times 13 \times 8 = \mathbf{1'352}$$

10.3.2.2. Big Picture

A Convolutional Neural Network is divided into two main functional parts: Feature Learning and Classification.

- **Feature Learning:** A repeating sequence of Convolution (with ReLU) and Pooling layers.
  - Convolution extracts patterns.
  - Pooling abstracts these patterns and reduces the data size.
- **Classification:**
  - **Flattening:** The 3D output of the last pooling layer is „unrolled“ into a 1D vector to make it compatible with standard dense layers.
  - **Fully Connected Layers:** Combine all extracted features to make a decision.
  - **Softmax:** The final layer that outputs a probability distribution across the target classes (e.g., Car, Truck, Van).

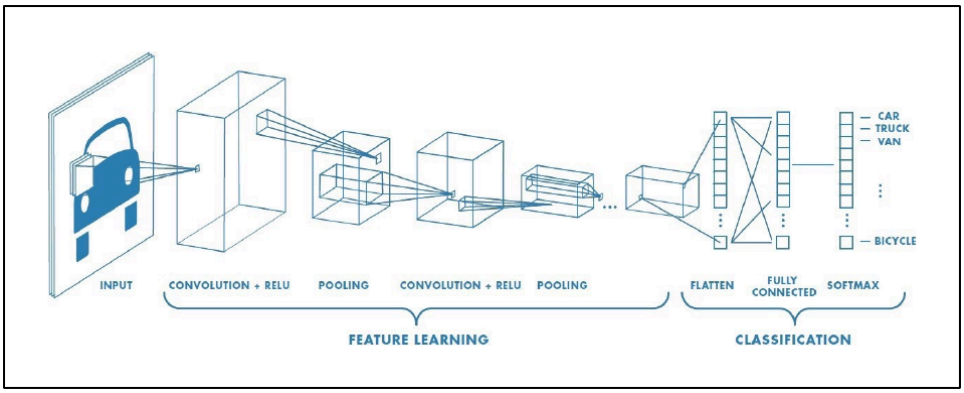


Abbildung 17: Complete CNN pipeline from RGB input to class probability

10.3.2.3. VGG 16 Architecture (2015)

Achieved 92.7% accuracy on ImageNet.

- Parameter sharing keeps model size tractable
- Progressively extracts information while decreasing resolution
- Uses Convolution, ReLU, Max Pooling, and Fully Connected layers

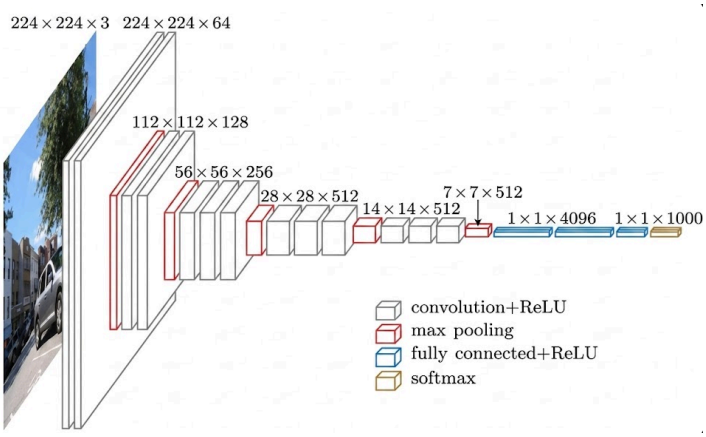


Abbildung 18: VGG 16 Architecture

10.3.2.3.1. CNN Parameter Calculation

The total number of learnable parameters is the sum of weights and biases across all layers.

- **Convolutional Layer Formula:**

$$\text{Params} = ((\text{Width} \times \text{Height} \times \text{Input Depth}) + 1) \times \text{Number of Filters}$$

- **Dense (Fully Connected) Formula:**

$$\text{Params} = (\text{Input Neurons} \times \text{Output Neurons}) + \text{Output Neurons (Biases)}$$

- **Flatten Formula:**

$$\text{Neurons} = \text{Width} \times \text{Height} \times \text{Depth}$$

Example

1. **Layer 1 (Conv2D):** 2 filters of 3×3, input depth 3 (RGB).

$$((3 \times 3 \times 3) + 1) \times 2 = \mathbf{56}$$

2. **Layer 2 (Conv2D):** 3 filters of 3×3, input depth 2 = anzahl Filter von Layer 1

$$((3 \times 3 \times 2) + 1) \times 3 = \mathbf{57}$$

3. **Layer 3 (Flatten):** Converts $16 \times 16 \times 3$ tensor to 768 neurons.

$$\text{Parameters} = \mathbf{0}$$

   (Data reorganization only)

4. **Layer 4 (Dense):** 768 inputs to 2 output classes.

$$(768 \times 2) + 2 = \mathbf{1'358}$$

Total Parameters: $56 + 57 + 0 + 1'538 = \mathbf{1'651}$

Code:

```py
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense

model = tf.keras.Sequential(name='Simple CNN Model')

model.add(Conv2D(2, (3,3), input_shape=(20,20,3)))
model.add(Conv2D(3, (3,3)))
model.add(Flatten())
model.add(Dense(2))

model.summary()
```

11. Recurrent Neural Nets (RNN)

11.1. Unsupervised Learning NLP

11.1.1. Syntax

DEFINITION: compare the spelling of words

11.1.2. Similarity

DEFINITION: how similar words are in respect to keystrokes

Levenstein Distance
- +1 when adding `[a]`
- +1 deleting `[d]`
- +1 substituting `[s]`

1. Word	2. Word	Levenshtein Distance
Hello	Yellow	1 [s] + 1 [a] = 2
Fahrrad	Velo	7

11.1.3. Word Relatedness

Idea 1: words are related when in **same document**
- eg: DOG & LEASH
- **Issue:** Relatedness of synonyms is low

Idea 2: words are related when in **same topic**
- importance from **word with each article**
- $\rightarrow$ **TF-IDF**

Idea 3 similar when words appear in **same context**
- $\rightarrow$ **Continuous Bag of Words**
- eg
  - in context pet: `PARROT` & `HAMSTER` more similar then `PARROT` & `COBRA`
  - `FAHRRAD` & `VELO` very similar

11.1.4. Term Frequency - Inverse Document Frequency (TD-IDF)

DEFINITION: relative importance of a word with respect to document

$$\text{TF-IDF}(x, y) = \text{TF}(x, y) \cdot \log\left(\frac{N}{\text{DF}(x)}\right)$$

- x: word
- y: document
- N: number of documents / articles in corpus
- TF: Term Frequency - counts number of occurrences of x in y
- DF: Document Frequency - number of documents that contain x

Examples
- `QUETZALCOATLUS` high TF-IDF: important to dinosaurs but rare elsewhere
- `AND` low TF-IDF: used everywhere

11.1.4.1. Numerical Encoding of Text - TF-IDF at Scale

Every word becomes a vector where the length = number of documents
- (e.g., **6.5M dimensions** for Wikipedia).

Similarity: Word relationships are calculated using **row-wise cosine similarity**.

	Wikipedia #1	Wikipedia #2	Wikipedia #3	Wikipedia #4	...
1. Word	0.05	0.001	0	0	...
2. Word				0.65	...
3. Word				0.002	...
4. Word				0.0031	...
...	...	...	...	...	...

TF-IDF Scores measures importance of word i with respect to document j

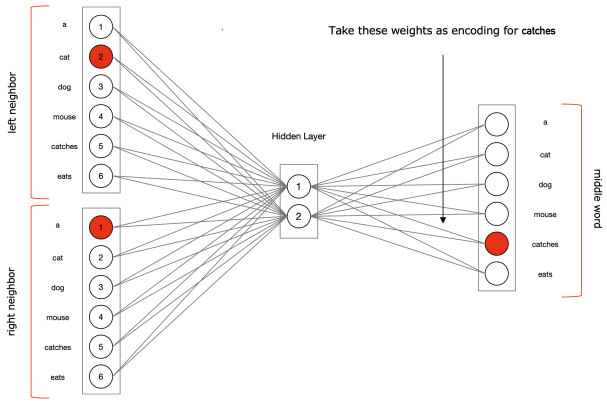
The Bottlenecks:

- **Sparsity:** Most vector values are zero.
- **Efficiency:** High memory usage and slow calculations make it impractical for massive datasets.

11.1.5. Continuous Bag of Words

DEFINITION: Predict a **middle word** using its surrounding neighbors (context).

- **hidden layer weights** become the numerical encoding for the word
- **Efficiency:** Unlike TF-IDF, vector size can be chosen (e.g. 200 dimensions)
  - $\rightarrow$ dense and fast



- **Left and right** neighbors share the **same weight matrix**.

11.1.6. Mathematics with Text

Embeddings enable semantic math (e.g., $\text{King} - \text{Man} + \text{Woman} = \text{Queen}$)

WICHTIG: **Bias:** Neural models inherit and amplify data prejudices.

11.1.7. Implementation & Tools

TF-IDF: Use Scikit-Learn's `TfidfVectorizer`.

FastText: Recommended for **Word Embeddings**.

- https://fasttext.cc/

11.2. Supervised NLP

11.2.1. Types

Sequence-to-Vector Models

- Sentiment analysis
- Input: sequence
- Output: vector

Vector-to-Sequence Models

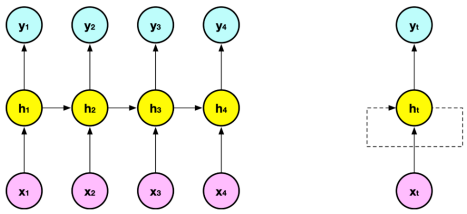
- Image Captioning
- Input: vector

- Output: sequence

Sequence-to-Sequence Models
- Text Summarization, Translation
- Input: sequence
- Output: sequence

11.2.2. Recurrent Neural Networks - RNN

DEFINITION: RNN are made for the supervised analysis of arbitrary time-series data



- **next hidden state h_t**: calculated using previous $h_{\{t-1\}}$ and next input x_t
- **next output y_t**: calculated using h_t

WICHTIG: input tokens, output tokens & hidden state are **fixe-sized vectors**

for variable: use Encoder / Decoder

RNNs use **same parameters** at every time step to process sequences:
- **Weight Sharing:**
  - $W_{\{xh\}}$: Input to hidden.
    - $W_{\{hh\}}$: Hidden to hidden (recurrent link).
    - $W_{\{hy\}}$: Hidden to output.
    - **Biases:** b_h and b_y remain constant across all steps.

Hidden State

$$h_t = \varphi\left(W_{\{xh\}}x_t + W_{\{hh\}}h_{\{t-1\}} + b_h\right)$$

Output

$$y_t = \varphi\left(W_{\{hy\}}h_t + b_y\right)$$

- φ represents an activation function like tanh or ReLU.

11.2.2.1. Vanishing Gradient

Gradients are **multiplied** backward through time
- small values shrink toward zero.

Problem: Prevents learning **long-term dependencies** (earlier info is lost).
- Forgetting „son“ (singular) when choosing „was/were“ much later

Fix: **GRU** & **LSTM**

11.2.3. Gated Recurrent Units - GRU

Candidate ($\tilde{h}_t$): potential new state based on current input x_t & past memory h_{t-1}.

Update Gate (Γ_{update}): A sigmoid (σ) value (0 to 1) deciding how much to update vs. retain.

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$\Gamma_{\text{update}} = \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u)$$
$$h_t = \Gamma_{\text{update}} \cdot \tilde{h}_t + (1 - \Gamma_{\text{update}}) \cdot h_{t-1}$$

11.2.4. Long Short-Term Memory - LSTM

LSTMs add a dedicated **cell memory state**

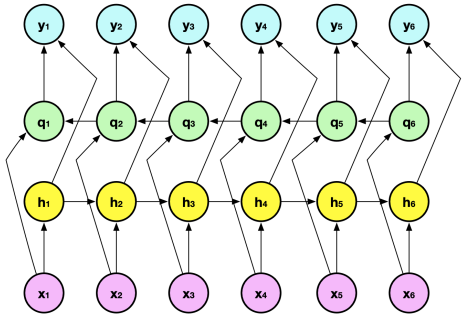
- GRUs are faster/simpler
- LSTMs often perform better on complex tasks

11.2.5. Bidirectional RNNs

Problem: Standard RNNs only „see“ the past.

- eg: Teddy can be: Roosevelt or bear

> **DEFINITION**: Uses two hidden layers.
>
> **Forward Layer (Yellow):** Processes the sequence from start to finish.
>
> **Backward Layer (Green):** Processes the sequence from end to start.



Output y_i: Combines information from „past“ & „future“ $\rightarrow$ full context of word

11.2.6. Encoder Decoder

Problem: Standard RNNs require equal input/output lengths.

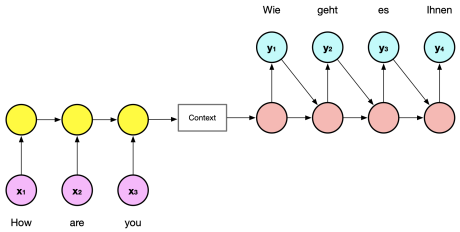
- Fails for **Translation** or **Sentiment Analysis**.

Solution: Encoder-Decoder architecture.

11.2.6.1. Unidirectional

Encoder: Compresses the full input into a single **context vector**.

Decoder: Uses that vector + a `START` token to generate the output word-by-word.

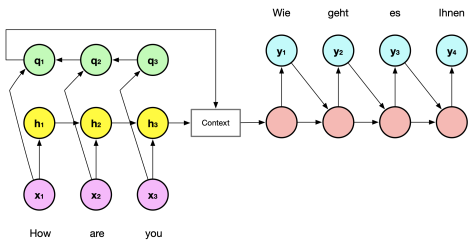


Each output (y_i) is fed back in as the input for the next step ($y_{\{i+1\}}$).

11.2.6.2. Bidirectional

Encoder: Uses both **forward** (yellow) & **backward** (green) layers to encode the input

Decoder: Generates the translation word-by-word.



Context: Combines the final states of both directions into one rich **context vector**.

11.2.7. Attention

DEFINITION: Mimics possibility to **look up information in another place**

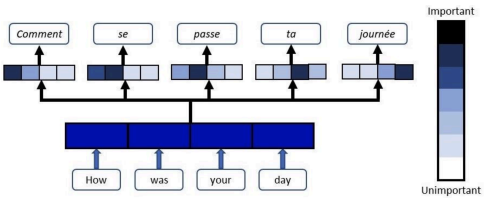
Decoder have access to **all encoder states**.

How important is each input for the next output

Attention values
- determine how **much information** to extract from each encoder state
- $a^{\{<7,2>\}} \rightarrow$ how much attention 7th output to 2nd input

Benefit: Global receptive field

Intuition behind Attention Prioritizing input words based on output relevance:

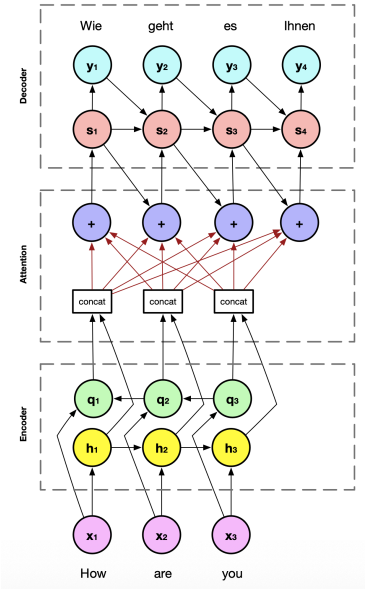


11.2.7.1. Attention Layer

Encoder
- bi-directional RNN (LSTM/GRU)
- combines forward (h_t) & backward (q_t) states into one representation

Decoder
- Generates output (y_t) using its own state (s_t) combined with a „weighted sum“ of encoder states.

Attention Values
- weights between **0 and 1** that sum to **1**.
- Input weighting (%) per output



11.2.7.2. Math

- **Score Calculation ($e^{\langle i,j \rangle}$):** Measures relevance between the previous decoder state and current input states.

$$e^{\langle i,j \rangle} = \varphi\big(s_{i-1}, [h_j, q_j]\big)$$

- **Weight Normalization ($\alpha^{\langle i,j \rangle}$):** Converts scores into a probability distribution using Softmax.

$$\alpha^{\langle i,j \rangle} = \text{softmax}\big(e^{\langle i,j \rangle}\big)$$

- **Context Vector (a_i):** Computes the weighted sum of encoder states to be used for the next output.

$$a_i = \sum_{j=1}^{3} \alpha^{\langle i,j \rangle} \cdot [h_j, q_j]$$

11.2.7.3. Parallelize

WICHTIG: RNNs process steps one-by-one → **hard to parallelize**.

- **Model Parallelism:** Different GPUs per layer; allows overlapping execution.
- **Data Parallelism:** Partitioned data across multiple models with shared weights.

→ Transformers: works without recurrencies

12. Generative Models 1 - Transformers

12.1. Language Models

- Shows probability of word sequences: $p(w_1, ..., w_m)$
- Predicts next word using mathematical conditional probability

$$p(w_3 \mid w_1, w_2) = \frac{p(w_1, w_2, w_3)}{p(w_1, w_2)}$$

- Creates text by picking words based on these probabilities

12.1.1. Markov Chains

- Calculates probabilities just by counting how often words appear together
- **2nd order Markov chain:** remembers only last two words → cannot write good long sentences
- Adding more memory requires too much computing power
- **Example**: For 1000 words, remembering just 3 words needs 1000^3 (one billion) values
  - Remembering 30 words would need more memory than atoms in the universe 🤯

12.1.2. Text Generation with RNNs

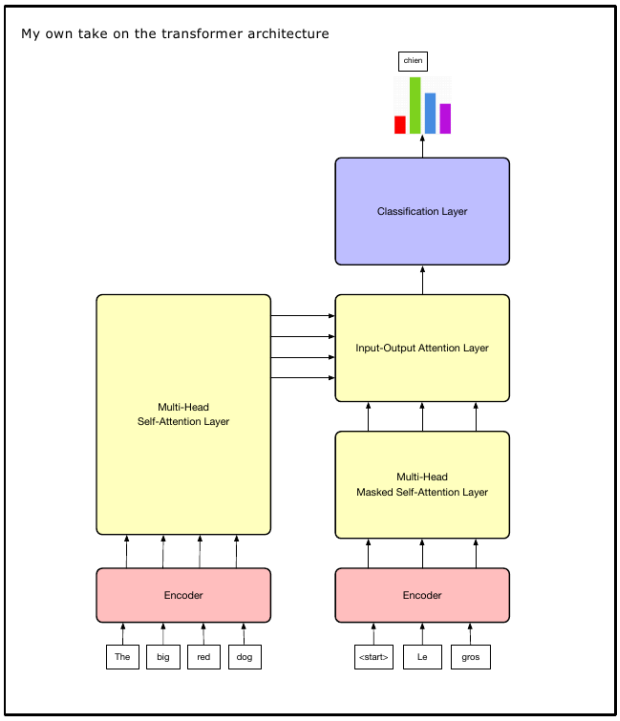
- Can remember longer word sequences without needing too much memory
- Uses a continuous state space to store information
- Works well for specific topics and short texts (like cooking recipes)
- The text quality depends heavily on how much data is used for training:
  - **30 KB:** No structure, repeats words often
  - **10 MB:** Better structure, but makes no logical sense
  - **350 MB:** Clear and logical text

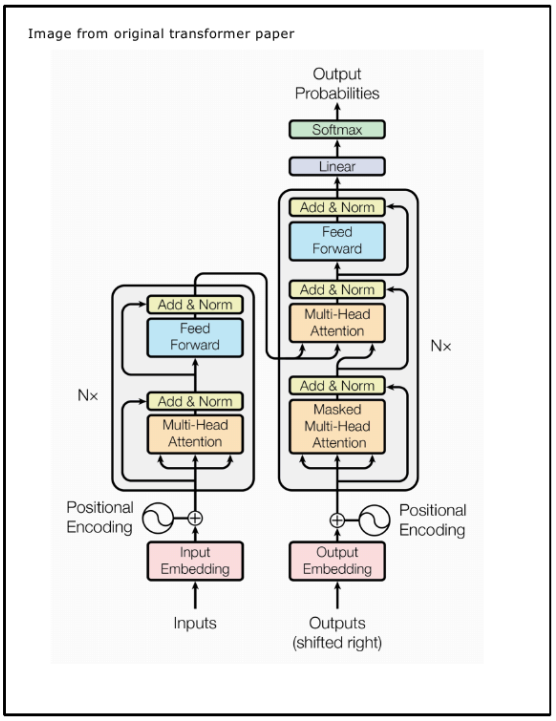
12.2. The Rise of the Transformers

- Introduced in 2017, Paper: „Attention Is All You Need“
- Processes sequential data without using recurrences (unlike RNNs)
- Allows processing whole sentences at once
  - Faster
  - Better understanding

12.2.1. Transformer Architecture

- Original architecture consists of a separate encoder and decoder
- **Encoder:** Takes input text & processes its meaning
- **Decoder:** Uses processed information to generate the final output text
- Perfectly designed for language translation
- Doesn’t require both parts
- Encoder alone is sufficient:
  - Text embeddings
  - Understanding tasks like sentiment analysis

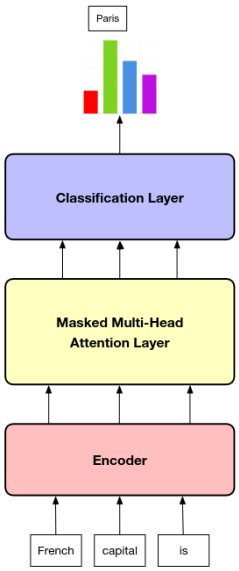




12.3. GPT: Decoder-Only Transformers

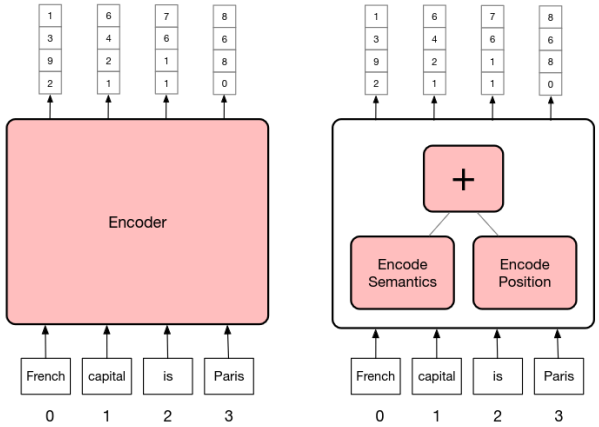
DEFINITION:
- Decoder-only architecture
- Generates text one word at a time
- Good at performing tasks it was not specifically trained on

- Encoders need too much task-specific data for finetuning
- Large decoder-only models show superior zero-shot generalization, meaning they perform well on untrained tasks
- This architecture is known as GPT (Generative Pre-trained Transformers)
- Examples include ChatGPT, LLaMA, Falcon, BARD, Cohere, PaLM and Claude
- GPT models generate text sequentially, one word at a time



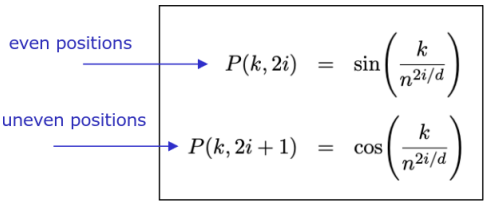
12.3.1. Encoding Input Tokens

- Words are processed in two simple steps:
  - **Step 1 (Meaning):** Words are transformed into vectors (embeddings) to capture what they mean
  - **Step 2 (Position):** Because transformers look at all words at once, they do not naturally know the word order
- **Position information is required** for every word vector
  - Order changes meaning (for example, „A woman ate a fish“ vs. „A fish ate a woman“)
- Meaning and position are calculated simultaneously and then added together



12.3.2. Positional Encoding

- Uses **sine and cosine functions to represent where a word is in a sentence**
- **Even positions:** Sine function
- **Odd positions:** Cosine function
- Calculation uses variables like word position (k) and embedding index (i)
- The embedding space dimension (d) is often 512
- These position numbers can be calculated for all words at the same time (in parallel)
- Finally, these position vectors are added directly to the word meaning vectors



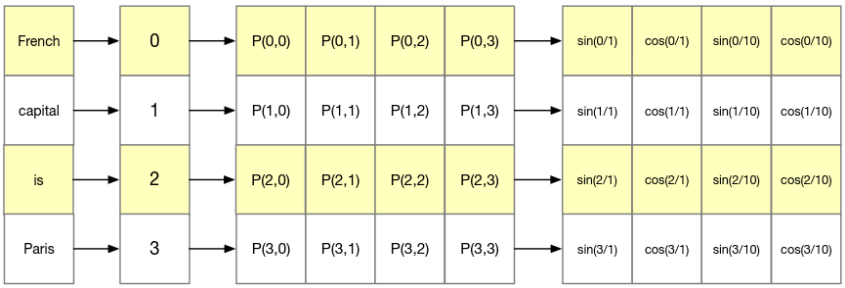


Abbildung 19: dimensions d = 4, frequencies of the sine and cosine waves n = 100

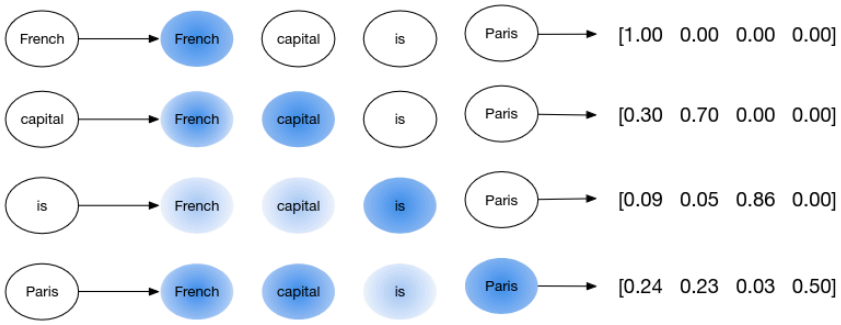
12.3.3. Masked Self-Attention

DEFINITION: **Contextual Relevance:** Weighs a word against all previous words in the sequence.

Look-ahead Mask: Blocks access to future data during the process.
- Prevents „cheating“ to ensure the model learns to predict realistically.

12.3.3.1. Intuition

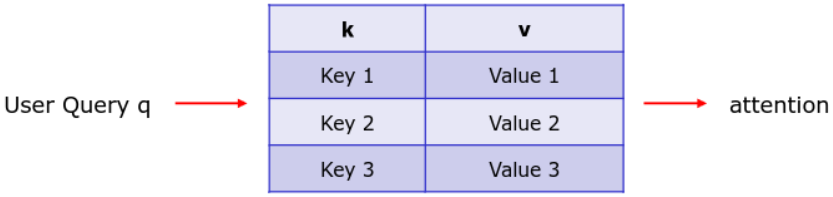
- Evaluates the relevance (connection) of a specific word compared to all previous words
- The masking ensures the model cannot look into the future to prevent cheating
- This process generates one attention vector per word
- **Issue:** A word usually has the strongest connection to itself, causing the network to ignore other words
- **Solution:** Compute multiple attention vectors per word and return a weighted sum
  - → Multi-Head Self-Attention



12.3.3.2. Information retrieval

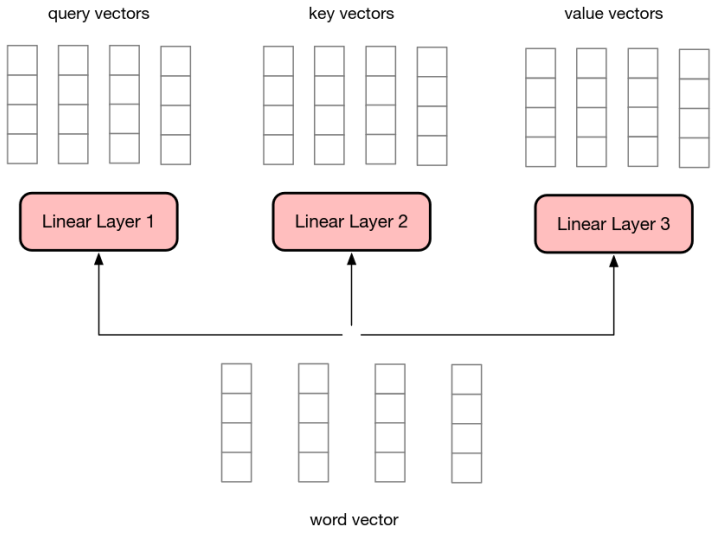
- Concept from information retrieval systems
- User query (q) is compared against a set of keys (k)
- Process finds the best match and returns the corresponding value (v)
- Similarity function is used to compare queries and keys
- Dot product acts as an unscaled cosine similarity for this comparison

$$\text{attention}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \sum_i \text{sim}(q, k_i) \cdot v_i$$



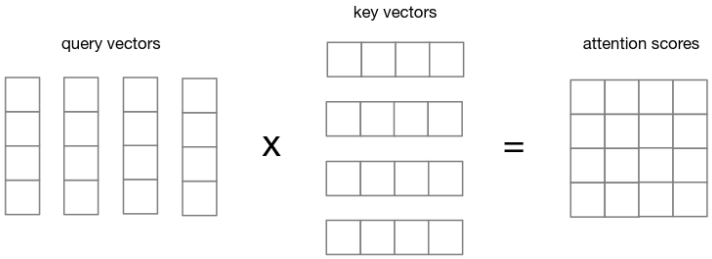
12.3.3.3. Query, Key, Value

- For the neural network implementation, every single word vector is fed into 3 distinct neural networks (linear layers)
- Process generates: **query vector**, **key vector**, and **value vector** for each word
- Weights used to calculate these vectors are identical for every word

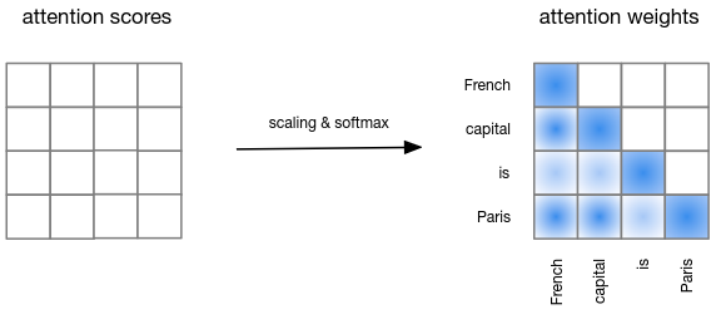


12.3.3.4. Implementation

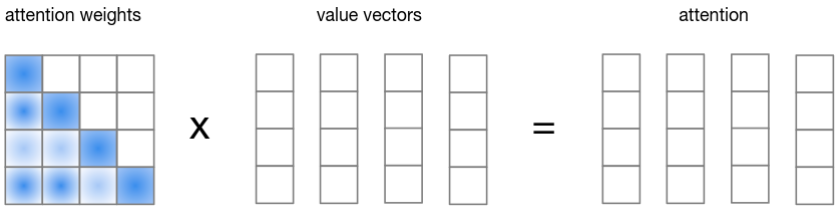
- Concept structured like a search engine
  - **Query** vectors: search term
  - **Key** vectors: keywords attached to various websites
  - **Value** vectors: actual content of websites
- **Step 1 (Attention Scores): Query vectors multiplied with transposed key vectors** (matrix multiplication)
- Resulting attention score: Raw mathematical similarity between the current word and previous words



- **Step 2 (Attention Weights):** Scores are **scaled for numerical stability** and passed through a **softmax function**
- Attention weights convert the raw scores into probabilities
  - Determines how much focus the model should put on each specific word



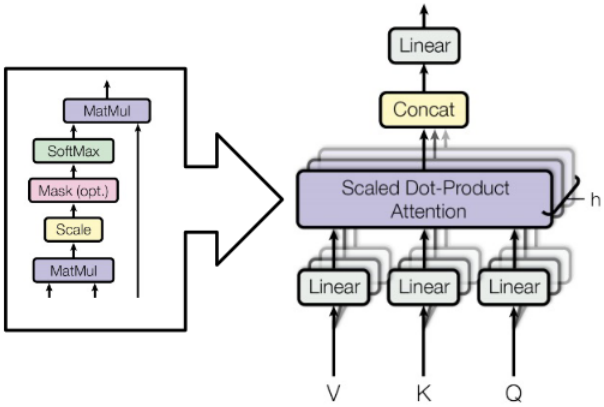
- **Step 3 (Final Attention): Attention weights multiplied with value vectors** to produce the final output
- Output: Combined information from all relevant sources
- Provides a new word vector that includes the context and meaning of all surrounding words



12.3.4. Multi-Head Self-Attention

DEFINITION:
- Computes multiple attention vectors at once.
- Captures varied relationships across the sequence.
- Prevents the model from over-weighting the target word.

- Single word vectors tend to attend mostly to themselves
- To solve this we calculate multiple attention vectors per word and average them
- Q, K, V are projected into different subspaces using learnable matrices
- This process runs in parallel across multiple heads (8 originally, 96 in GPT-3)
- All head outputs are concatenated and passed through a linear layer to restore the original dimensions



12.3.4.1. Math

- **Single head:** Scaled dot product for similarity
- Softmax scales weights between 0 & 1 before multiplying with the value vector

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Every head applies attention to a different projection of Q, K and V

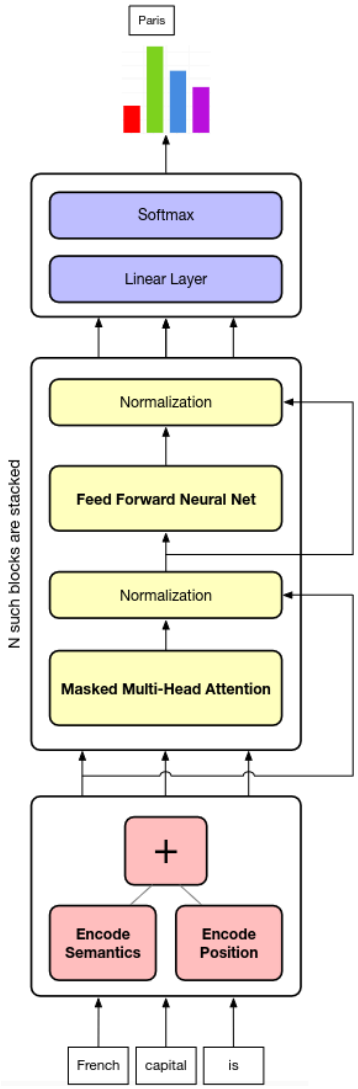
$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

- Finally all heads are concatenated and multiplied by a linear matrix to restore the dimensionality

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

12.4. The Big Picture

- **Multi-Head Attention** works as a linear operation
  - Feed-forward networks with two layers and ReLU activation help learn complex patterns
- **Normalization layers** keep the training process stable and efficient
- Skip connections prevent gradients from disappearing during training
- Original architecture: Stacks 6 blocks with 8 attention heads each



- **Final linear layer**: Matches data to the total vocabulary size
- **Softmax function** turns the output into a probability distribution for all possible words
- The model generates the next word by sampling from these probabilities

12.5. Reinforcement Learning with Human Feedback

- Human guidance is used to align the model behavior with human preferences

Step 1 Collect demonstration data

- Prompt is selected from a dataset
- Human expert writes the ideal answer to demonstrate the desired behavior
- Model is fine-tuned using these high-quality examples

Step 2 Train a reward model

- Model generates several different answers for a single prompt
- Human labeler ranks outputs from best to worst
- Ranking data is used to train a reward model

Step 3 Optimize with reinforcement learning

- Model generates a new output for a prompt from the dataset
- Reward model calculates a score for this generated output
- The PPO algorithm uses this reward score to update the model policy

12.6. Training Resources

- **LLaMA (Feb. 2023)** 65B parameters and 4.75 TB text data

- **Hardware** 1,022,362 GPU hours on A100-80GB
- **Consumption** 449 MWh power used
- **Footprint** 172,000 kg CO_2
- **Comparison** One SpaceX Falcon launch emits 336,552 kg CO_2

13. Transfer Learning

DEFINITION: pe-traines model is used for new similar task

Transfer Learning is based on the following observations:

Neural Nets are Feature Extractors:
- Early layers learn basic features (edge filter, color blobs)
- the more layers the more complex the structure that the net learns
- final layer learns prototype objects
- Feature Extraction of Neural Network is called **Backbone**
- Classification, localization, etc. are attached to the Backbone

WICHTIG: Learning basic features is task agnostic

DEFINITION: In transfer learning we first train a base network on a base dataset and task, the new repurpose the learned features or transferm them, to a second target datest and task. This process will tend to work, if the features are general instead of specific to the base task

WICHTIG: Works best when features are general across both tasks rather than task-specific.

13.1. Models
- Public Datasets for Pre Training exist
- Libraries of pre-trained models exist

13.2. Strategies

13.2.1. Strategy 1: Model Repurposing
1. Choose a pre-trained model for object detection
2. Chop off the last layer (object localization)
3. Add a new object localization layer with two categories
4. Train only the last layer with the domain-specific dataset

Features:
- Super Efficient
  - only train a single layer
- The least data hungry strategy
- backbone of pre-trained model is inherited
- backbone layers are frozen
- backpropagation of gradients is stopped at frozen layers

13.2.2. Strategy 2: Fine Tuning
- Early backbone layer are frozen
- Retrain subsequent backbone layers for complex strucutres
  - layers receive new gradients during backpropagation
- unfrozen layers start with the pre-trained weight values
- extreme case: Unfreeze all backbone layers

13.3. Impact on the industry

- Less Training Data necessary
- much cheaper
- smaller carbon footpring

⟶ Small Businesses can also train AI models thanks to transfer learning

13.4. Issues with supervised learning

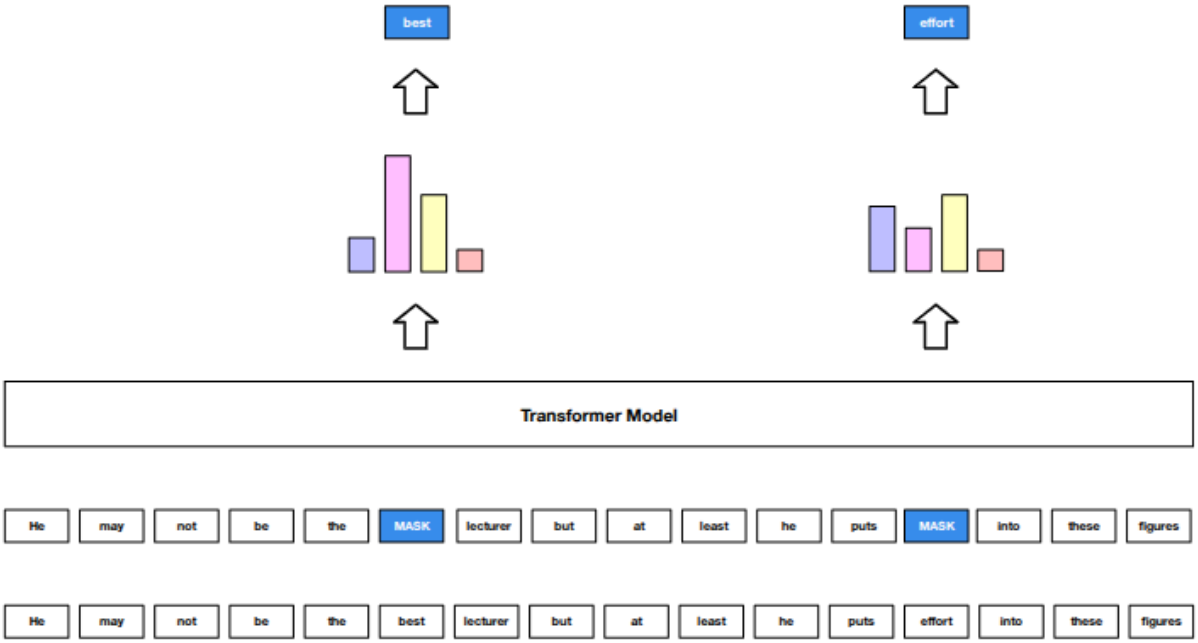
- Labelled image datasets
  - a huge dataset today maybe too small for tomorrows application
- Labelled video Datasets
  - too time intensive
- Labelled text datasets
  - easy for english, difficult for infrequent languages
- Labelling quality
  - image segmentation: annotations on pixel level
  - different opinions from experts

13.5. Unsupervised pre-training

- Solves the bottleneck because raw data is abundant.
- Allows networks to continue growing without relying on manual labels.
- **Key Techniques:**
  - Masked Language Modelling
  - Contrastive Learning

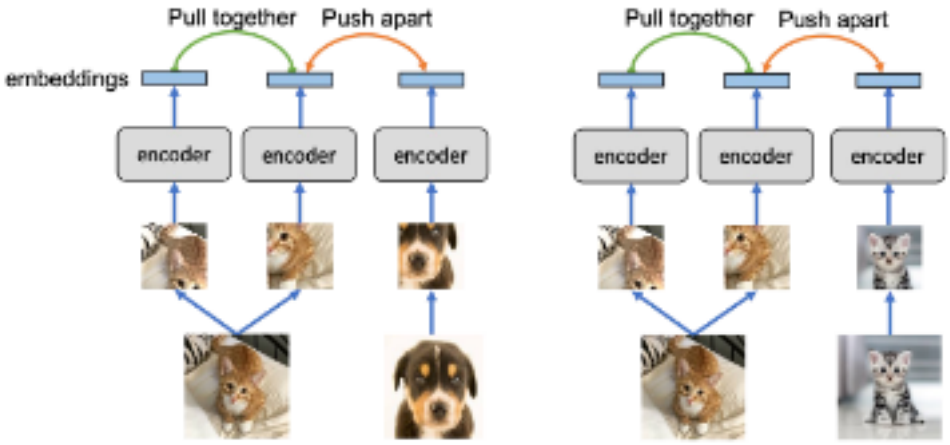
13.5.1. Masked Language Modelling

- Randomly hide (mask) words in massive amounts of unlabeled text.
- Train the model to predict those missing words based on the surrounding context.



13.5.2. Contrastive Learning Example

- Take a huge number of unlabeled images
- Randomly sample two tiles A and B from the same image
- Add tile B along with many tiles from other images to batch
- Neural net must output hight similarity between A and B and low similarity otherwise



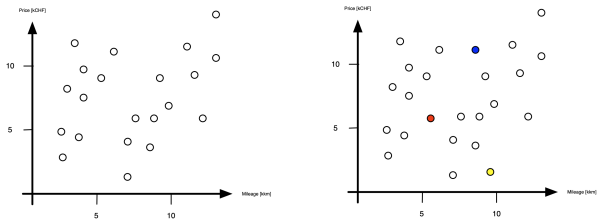
14. Clustering

14.1. k-Means Clustering

DEFINITION: **Unsupervised** clustering algorithm. Goal: find the **center of each cluster**.

14.1.1. Preparation

1. Choose number of clusters k in advance (e.g. $k = 3$)
2. Normalize data because distances are sensitive to scale
3. (Randomly) place **cluster centers** — not real data points



14.1.2. Steps

1. **Assign:** For each data point, find the **nearest cluster center** (by distance).
2. **Update:** Recalculate each center as the **mean** of all assigned points → move center to new coordinate.
3. **Repeat** until cluster centers **no longer change** (usually stops after 2 iterations without change).

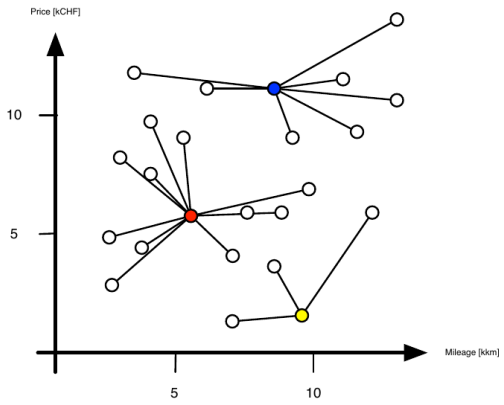


Abbildung 20: Assign

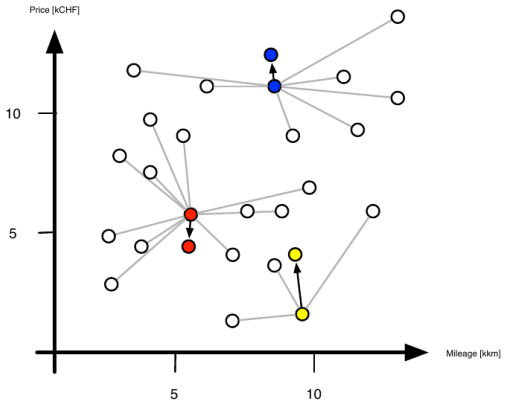


Abbildung 21: Update

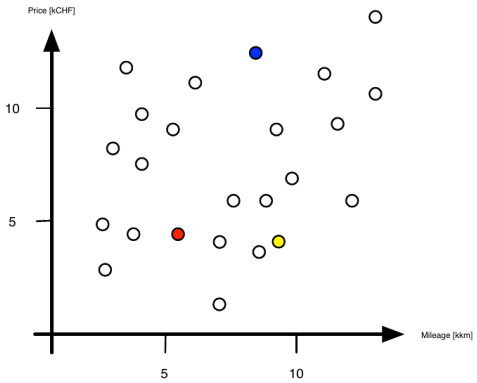


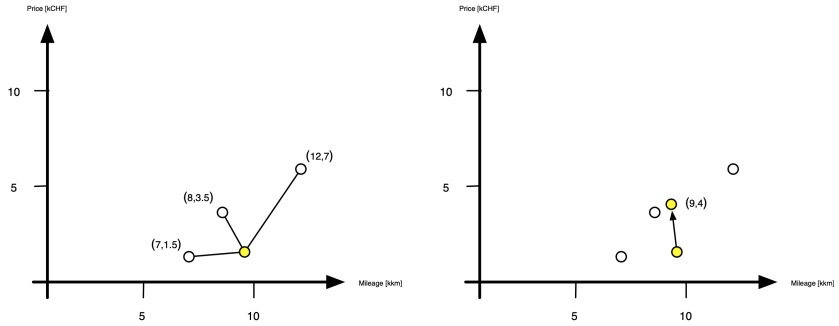
Abbildung 22: Repeat

14.1.3. Calculate Center of Cluster

New center = **mean of all assigned points** per dimension:

$$\mu_k = \left(\frac{x_1 + x_2 + ... + x_n}{n}, \frac{y_1 + y_2 + ... + y_n}{n} \right)$$

Example (cluster with 3 points):



$$\left(\frac{7 + 8 + 12}{3}, \frac{1.5 + 3.5 + 7}{3} \right) = (9, 4)$$

$\rightarrow$ Center moves to new coordinate $(9, 4)$.

14.1.4. Algorithm in Math

Input: number of clusters $k > 0$ and data points $\boldsymbol{x}_1, ..., \boldsymbol{x}_n \in \mathbb{R}^m$

1. Randomly choose k cluster centers $\mu_1, ..., \mu_k \in \mathbb{R}^m$
2. Repeat until convergence:

(a) Assign each data point $\boldsymbol{x}_i$ to its nearest cluster center μ_j:

$$c_i = \underset{j \in \{1,...,k\}}{\operatorname{argmin}} \|\boldsymbol{x}_i - \mu_j\|^2$$

- c_i: index of the nearest cluster center to $\boldsymbol{x}_i$
- $\|\boldsymbol{x}_i - \mu_j\|^2$: squared distance, proportional to **variance**

(b) Update each cluster center to the mean of all assigned points:

$$\mu_j := \frac{\sum_{i:c_i=j} \boldsymbol{x}_i}{|\{i : c_i = j\}|}$$

- Numerator: sum of all data points assigned to cluster j
- Denominator: count of data points assigned to cluster j

14.1.5. Clustering Distortion

Total Distortion: sum of squared distances between each point and its cluster center:

$$\sum_{i=1}^{n} \|\boldsymbol{x}_i - \mu_{c_i}\|^2$$

Average Distortion per data point:

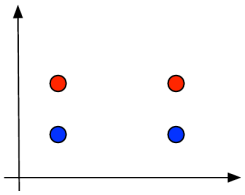
$$\frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mu_{c_i}\|^2$$

HINWEIS: Average distortion allows comparison of clusterings **across different datasets**.

14.1.6. Convergence and Optimality

- Optimal clustering = minimizes total distortion $\rightarrow$ **NP-hard** (even $k = 2$)
- k-Means only **approximates** the optimal solution
- **Always converges**, but **not** necessarily to a **global minimum**

- Risk: stuck in a **local minimum**



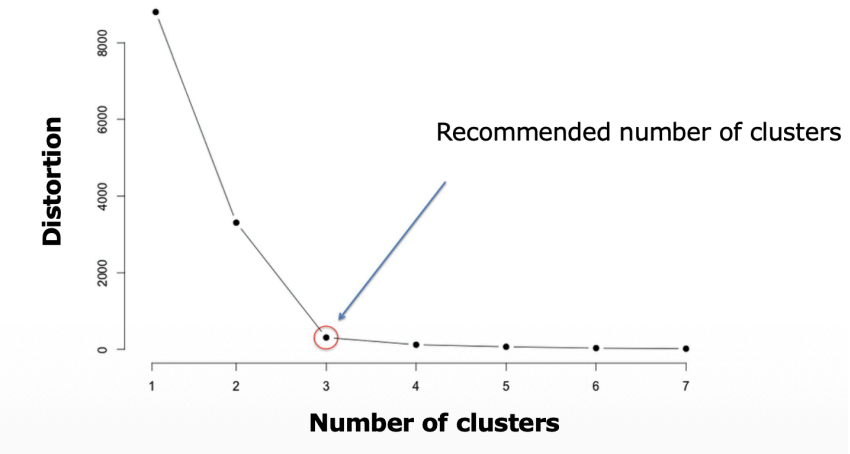
Practice: run **many times** → keep result with **lowest distortion** (Scikit-learn default: **10 runs**)

14.1.7. Elbow Method — Choose Number of Clusters

HINWEIS: If the number of clusters is known from the problem, use that k (e.g., k=10 for digits 0–9).

- $k = 1 \rightarrow$ distortion **maximal**
- $k = n$ (each point = center) $\rightarrow$ distortion $= 0$
- More clusters always means smaller distortion
- Prefer **few clusters** + **low distortion**

Elbow Method: pick k at the „elbow“ point where distortion stops decreasing significantly.



14.2. Agglomerative Clustering

DEFINITION: **Unsupervised** clustering algorithm.

Goal: iteratively merge the closest clusters into a hierarchy.

Outputs a **dendrogram**.

- No need to choose k in advance
- Number of clusters can be decided after the algorithm runs (by cutting the dendrogram)

WICHTIG: Does not scale to large datasets → only suitable for smaller amounts of data.

14.2.1. Motivation

- Produces a human-readable hierarchy (dendrogram) — unlike the flat output of k-Means
- Popular in **marketing analytics** (e.g. customer segmentation) where data is limited and interpretability matters
- Completely **deterministic** — no random initialization

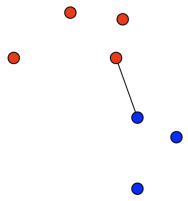
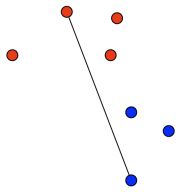
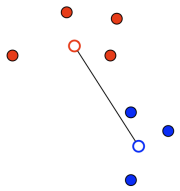
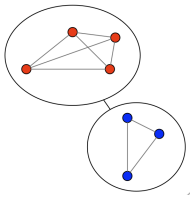
14.2.2. Steps

1. **Initialize:** treat every data point as its own cluster
2. **Repeat** until stop condition is met:
   - Calculate distances between all pairs of clusters
   - Merge the two closest clusters into one new cluster
3. **Stop** when condition is met (e.g. target number of clusters, cluster density, or interactively via dendrogram)

HINWEIS: To merge all n data points into a single cluster, the algorithm always performs exactly $n - 1$ steps.

14.2.3. Configurations / Hyperparameters

- **Distance measure:** any distance/similarity metric (e.g. Euclidean)
- **Stop criterion:** constant number of clusters, cluster density, or interactive
- **Linkage:** defines which points of two clusters are used to measure distance:

Linkage	Distance measured between...	
Simple	Closest members of two clusters (min)	
Complete	Farthest members of two clusters (max)	
Average	Centroids of two clusters	
Ward	Minimizes variance inside clusters, maximizes between clusters	

14.2.4. Example

Start: 6 points A–F, each is its own cluster.

	A	B	C	D	E	F
A						
B	2					
C	5	3				
D	21	17	10			
E	24	18	13	1		
F	77	61	54	24	17	

Step 1: D and E are closest (distance = 1) → merge into $\{D, E\}$

Update distances using simple linkage (take minimum):

$$d(A, \{D, E\}) = \min(d(A, D), d(A, E)) = \min(21, 24) = 21$$

Step 2: A and B are closest (distance = 2) → merge into $\{A, B\}$

→ Continue merging until stop condition is met.

	A	B	C	D	E	F
A						
B	2					
C	5	3				
D	21	17	10			
E	24	18	13	1		
F	77	61	54	24	17	

start

	A	B	C	D,E	F
A					
B	2				
C	5	3			
D, E	21 (21. vs. 24)	17 (17. vs. 18)	10 (10. vs. 13)		
F	77	61	54	17 (17. vs. 24)	

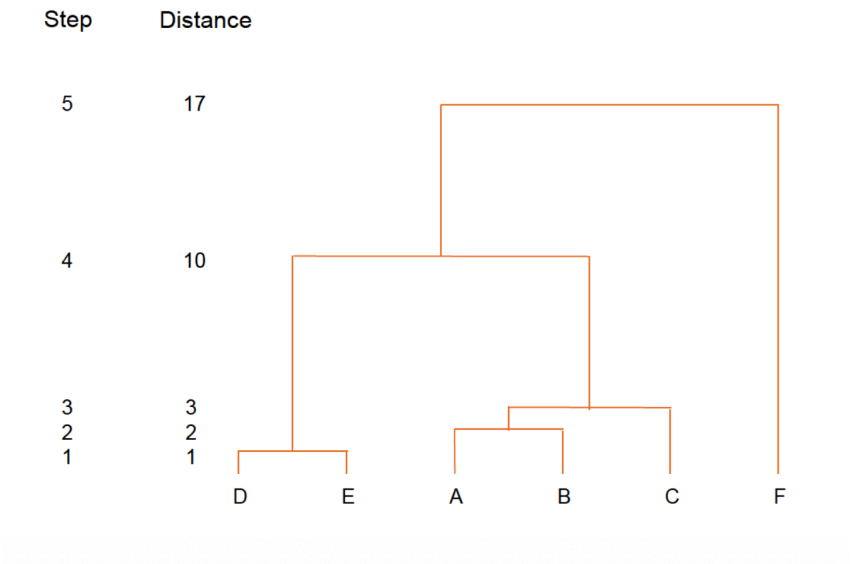
step 1: merge D and E

	A, B	C	D, E	F
A, B				
C	3			
D, E	17	10		
F	61	54	17	

step 2: merge A and B

14.2.5. Dendrogram

Records every merge step with the distance at which it occurred.



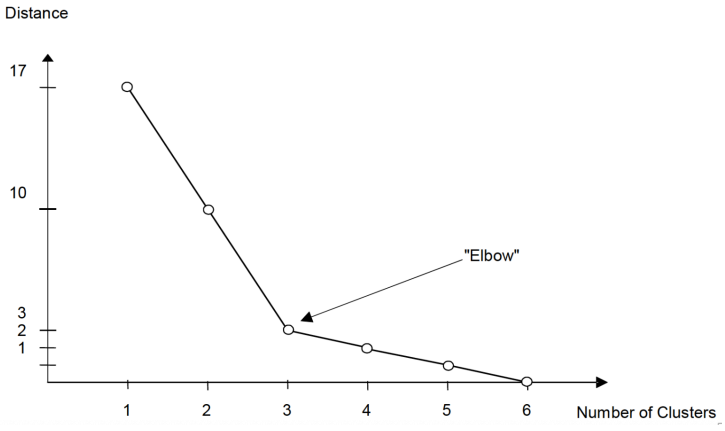
The dendrogram lets you **cut at any level** to obtain different numbers of clusters, without rerunning the algorithm.

14.2.6. Elbow Method — Choose Number of Clusters

Merge distance (y-axis) vs. number of clusters (x-axis):

- Large drop in distance = meaningful merge
- Small drop = clusters are getting forced together

Elbow Method: pick k at the point where distance stops dropping significantly.



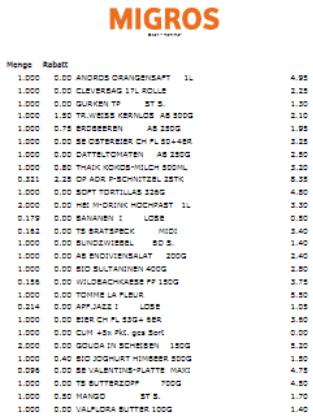
14.3. k-Means vs. Agglomerative Clustering

DEFINITION: Key difference: In k-Means, two points can be in the same cluster one step and split the next. In agglomerative clustering, once merged, points **stay together forever**.

	k-Means	**Agglomerative**
Stop criterion	Only fixed k	k, density, distance, interactive
Scalability	Large datasets	Small datasets only
Deterministic	No (random init)	Yes, always
Interpretability	Low	High (dendrogram)
New data	Difficult (centers shift)	Very difficult (rebuild hierarchy)

15. Association Rules 1 - Market Basket Analysis

15.1. Transaction Data



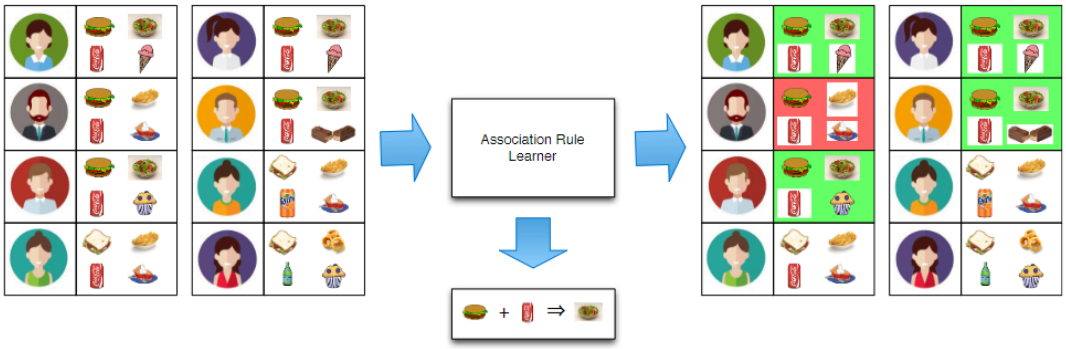
DEFINITION: Data describing a single business transaction (example shopping cart)

- for analysis we need the transactions in binary format

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

15.2. Association Rules

DEFINITION: Implication of $X \rightarrow Y$, where X and Y are disjoint item sets



15.3. Support

15.3.1. Support of a Set of Items

DEFINITION: Support is the proportion of transactions which contains a specific item set

(Measure how frequently items are bought together)

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

- `support({hamburger, coke}) = 5/8`
- `support({salad, coke}) = 4/8`
- `support({coke, choc bar}) = 1/8`
- `support({hamburger, salad, coke}) = 4/8`
- `support({sandwich, fries}) = 2/8`
- `support({hamburger, salad}) = 4/8 = 2/4 = 1/2`

15.3.2. Support of an Association Rule

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

- `support({salad} -> {hamburger, coke}) = 4/8`
- `support({sandwich} -> {fries}) = 2/8`
- `support({hamburger, salad} -> {ice cream}) = 2/8`

Note:
- Support is direction invariant
- cannot measure the quality of a directed rule

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

15.3.3. Interpretation of Support

- measures how frequently an item set occurs in the data
- rules with very low support
  - may occur randomly
  - may be uninteresting from a business perspective
- Important: Do not confuse support with e.g., accuracy of a classifier
  - classifier with accuracy 25% is bad
  - item with support 25% is unrealistically high
- support with 0.1 → in every 10th the item appears, this is very often!

WICHTIG: A good association rule has a high support
Support = Interestingness

15.4. Confidence of an Association Rule

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

- `confidence({hamburger, coke} -> {salad}) = 4/5`
- `confidence({hamburger, salad} -> {coke}) = 1.0`
- `confidence({coke} -> {choc bar}) = 1/6`
- `confidence({hamburger, fries} ⇒ {coke}) = 1.0`

15.4.1. Interpretation of Confidence

DEFINITION: Determines how frequently items in Y appear in transaction that contain X. Measures reliability or trustworthiness of a rule

- for a given rule $X \to Y$
- the higher the confidence, the more likely it is for Y too be present in transactions that contain X
- confidence provides an estimate of the conditional probability $p(Y|X)$
- confidence should be very near to 1

Example:
- {bread, eggs, milk}
  - support = 0.15
- {bread, eggs}
  - support = 0.15
- then {bread, eggs} → {milk}
  - confidence = 1
- mean: the rule is correct in 100% of the transactions that contain bread and eggs

WICHTIG: a good association rule has high confidence **Confidence = Trustworthiness**

15.5. Trustworthy but Uninteresting Rules

WICHTIG: We **must not** just consider one of the two measures

rule: {anchovies paste} → {salad}
- support is very low (because anchovies pasta is rarely bought)
- confidence is approximately 1

rule: {eggs} → {banana}
- high support
- low confidence
- because both items are bought very often
- but together by chance

15.5.1. Issue

- confidence of $X \to Y$ takes only the antecedent X and the cooccurence of $X \cup Y$ into account
  - but not the consequence Y
- support and confidence cannot filter out rules that occur coincidentally (zufällig)
- **Solution: lift**

15.6. Lift of an Association Rule

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

Note:
- lift is direction invariant
- cannot replace confidence

- `confidence({hamburger, salad} -> {coke}) = 1.0`
- `support({hamburger, salad, coke}) = 4/8`
- `support({hamburger, salad}) = 4/8`
- `support({coke}) = 6/8`
- `lift({hamburger, salad} -> {coke}) = 8/6 > 1`

15.6.1. Interpretation of Lift

DEFINITION: Measures how many times more often X and Y show up together than expected if they were statistically independent

- Lift = 1: X and Y are statistically independent
- Lift < 1: 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: 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
- **The larger the lift value, the stronger the association between X and Y**
- lift value are not upper bounded

WICHTIG: a good association rule has a high lift **Lift = Association Strength**

15.7. Business

Assume that the simple rule $\{X\} \rightarrow \{Y\}$ has high support, high confidence and lift > 1. How can a store manager benefit from this information

- Put X and Y closer (or more distant) in the store
- Package X with Y
- Package X and Y with a poorly selling item
- Give discount on only one of X and Y
- Increase the price of X and lower the price of Y (or vice versa)
- Advertise only one of X and Y, i.e. do not advertise X and Y together
- ...

16. Association Rules 2 - Algorithms

16.1. Standard Association Analysis Procedure

Process:

1. Pre-process transactions
2. ML: find rules with high support and high confidence
3. Calculate lift value for each interesting rule
4. sort rules in descending order with respect to lift value
   - consider hight-lift rules first
   - ignore rules with lift value close to 1

General:

- $\min_s$: support threshold
- $\min_c$: confidence thresholds
- **goal**: finding rules having support >= $\min_s$ and confidence >= $\min_c$

Complexity:

- Input can have a lot of transactions
- brute-force search is impossible
  - even for small shops
- for n products there are $3^n - 2^{\{n+1\}} + 1$ possible association rules
  - 602 rules for only 6 products
  - 57002 rules for 10 products
  - ...
- **rules grow exponentially**

16.2. Apriori Algorithm

Two step algorithm:

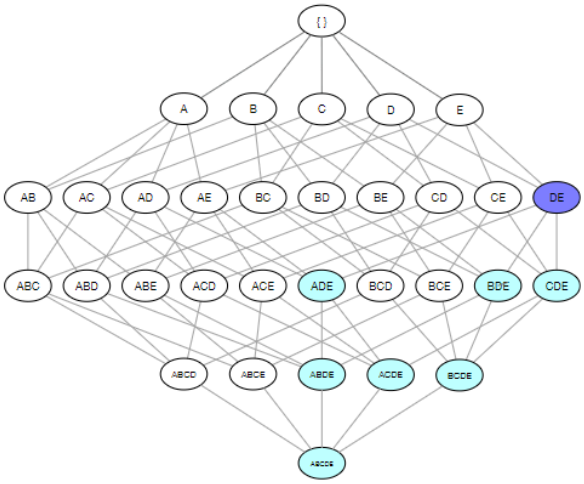
1. generate frequent item sets statisfying the support threshold
2. Extract rules from frequent item sets satisfying the confidence threshold

16.2.1. Step 1: Frequent Item Set Generation

Apriori Property:

- If an item set is frequent (high support) all subsets must be frequent too.
- If an item set is infrequent (low support) all super-sets must be infrequent too
  - makes the pruning more efficient

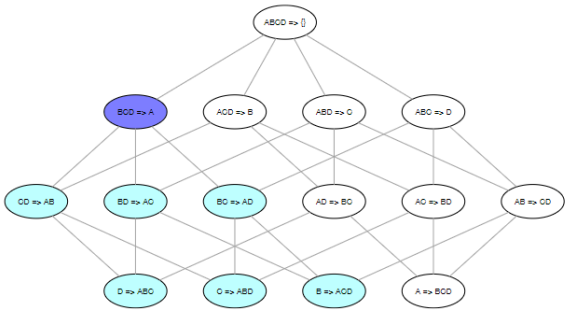
Visualization in a **lattice** of subsets:



- it starts with one product, then two and so on
- blue node DE is infrequent *to* all connected nodes (light-blue) can be eliminated
- the graph connects two nodes if one is a subset of the other
- called **lattice of subsets**

16.2.2. Step 2: Rule Generation from Frequent Item Sets

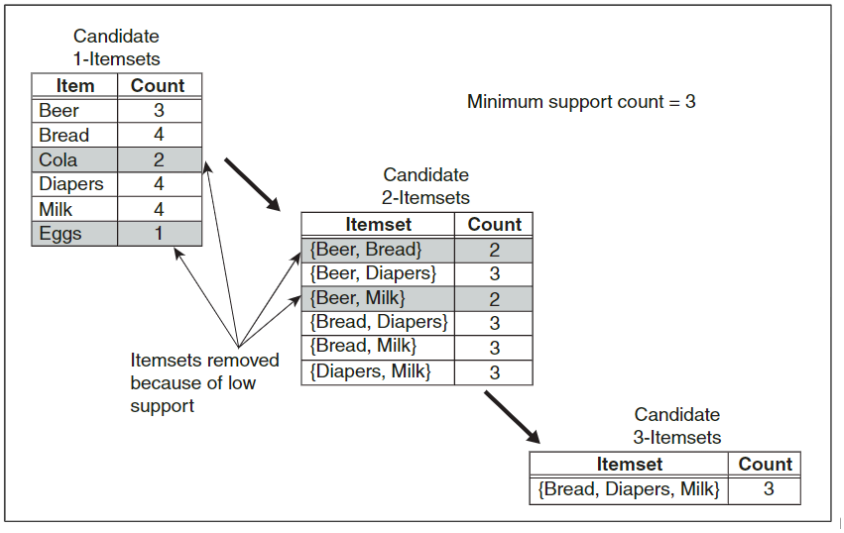
- $X \cup Y$ is a frequent item set found in step 1
- When a rule $[X \rightarrow Y]$ violates the confidence threshold, then any rule $[X - \{c\} \rightarrow Y \cup \{c\}]$ violates the confidence threshold as well
  - c can be any item



- blue node BCD $\Rightarrow A$ violates confidence bound, all connected nodes (light-blue) can be eliminated
- called **lattice of rules**

16.2.3. Example

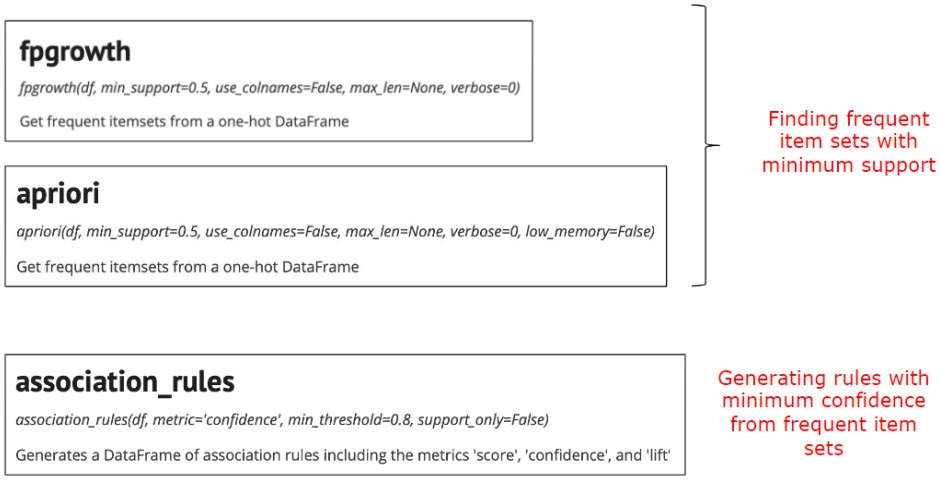
Step 1:



- threshold is 3
- in every step some candidates are eliminated because of the threshold

16.3. Implementation

- not possible in Scikit Learn API
- but in **mlxtend**



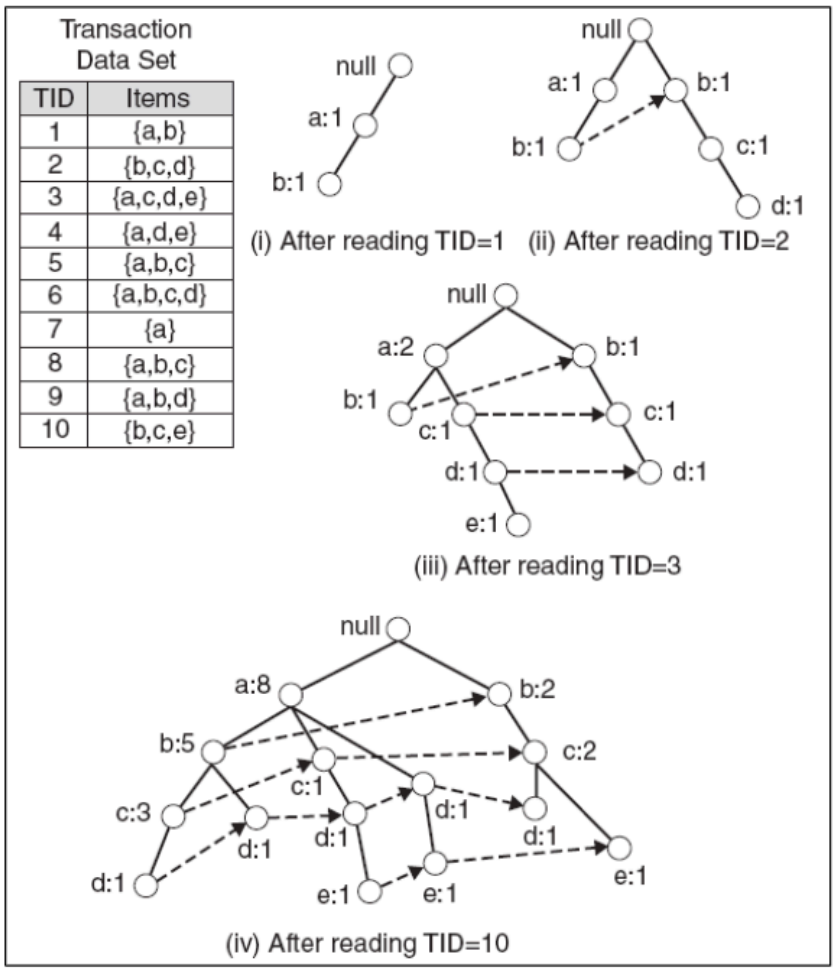
16.4. Appendix

16.4.1. Limitations of Apriori and Alternatives

- Apriori is slow for larger data sets
  - uses breadth-first search (Breitensuche)
- ECLAT uses depth-frist search (Tiefensuche) and set intersection for pruning
- Frequent Pattern Growth (FP-Growth) extracts frequent item sets based on a specialized data structure called FP-Tree
- **Recommendation**:
  - for small data sets: Apriori
  - larger data sets: FP-Growth

16.4.2. FP-Tree Example

- transactions are sorted
- nodes correspond to items and have a counter
- FP-Growth reads one transaction at a time and maps it to a path
- Counters are incremented for each item
- Pointers are maintained between nodes containing the same item
- FP-Growth extracts frequent item sets from the FP-Tree using a bottom-up (from leaves to root) divide-and-conquer algorithm



17. Recommender Systems

17.1. Non-Personalized Recommenders

DEFINITION: Recommender systems suggest items to users. Non-personalized recommenders give the **same recommendation to all users** in the same context.

17.1.1. Applications

- **Online**: Amazon („customers who also bought“), Zalando („complete your look“)
- **Offline**: Targeted mailings, loyalty card campaigns (e.g. STUcard)
- Real-world impact: up to **2.55× sales** when replacing human experts (sommelier vs. algorithm)

17.1.2. Challenges

1. **In-house competition**: IT vs. marketing; human marketers feel threatened
2. **Taste changes over time**: preferences shift year to year
3. **Efficiency**: minimize explicit user interactions needed for training
4. **Cold Start Problem**: how to recommend to new users or new items?

17.1.3. Scoring & Rankings

Steps:

1. Collect ratings (e.g. from {0, ..., 5})
2. Calculate mean → round off
3. Alternatively, use sales figures instead of ratings

Examples: Amazon star ratings, BoxOffice.com sales, iTunes Charts, Zagat Restaurant Guide

17.1.4. Associations

Goal: Recommend items that are frequently bought together (context-aware).

17.1.4.1. Confidence (naïve — does not work well!)

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

Problem: Banana (Y) is popular everywhere → inflated confidence regardless of X.

17.1.4.2. Corrected: Lift

$$\frac{X \text{ and } Y}{X} \div \frac{\neg X \text{ and } Y}{\neg X}$$

Compensates for the overall popularity of Y.

- $\frac{X \text{ and } Y}{X}$ is the percentage of times Y was bought given that X was bought
- $\frac{\neg X \text{ and } Y}{\neg X}$ is the percentage of times Y was bought when X was **not** bought

17.1.5. Shop-Specific Associations

Three strategies to define „bought together“:

1. **Ever bought X and Y** → not good for fast food sauces (too broad)
2. **Bought X and Y in the same transaction** → not good for accessories like skis + ski boots (bought separately)
3. **Bought X and Y within one month** → usually the best trade-off

WICHTIG: Wrong time window → surprising or irrelevant recommendations (e.g. smoking pipe + pirate hat).

17.1.6. Pros

Easy, fast, cheap

- uses existing transaction data

Context-aware
- recommends products that go together

Tailorable
- time constraints, „viewed“ vs. „bought“

17.1.7. Cons

Non-personalized
- looks at baskets, not individual preferences

17.1.8. Homogeneous vs. Heterogeneous Catalogs

Type	Description	Examples
Homogeneous	All products share a common feature set	Car rental, real estate, recipes
Heterogeneous	Mixed product types	Amazon, Galaxus

For homogeneous catalogs → represent products as **vectors** and measure similarity.

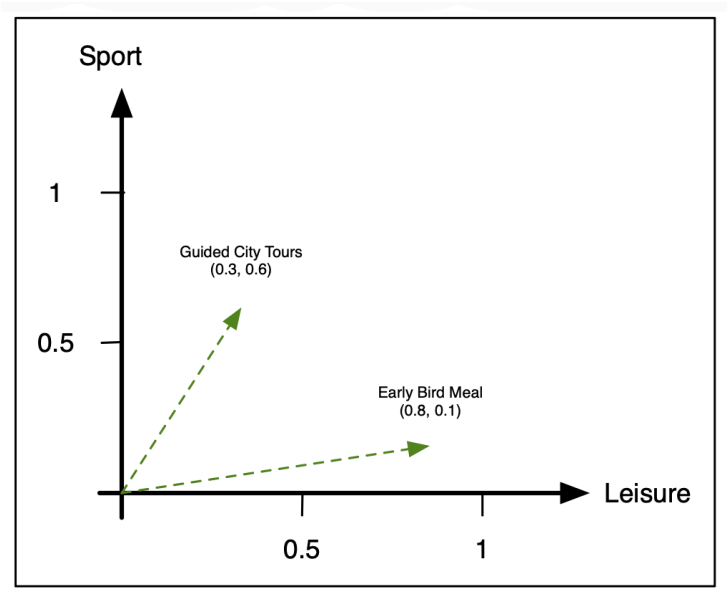
Example feature vectors:

	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
Candlelight Cruise	0.3	0	0.3	1.0

17.1.9. Content-Based Recommenders

DEFINITION: When a user clicks a product, recommend the most similar products using **cosine similarity**.

$$\cos(\theta) = \frac{a \cdot b}{\|a\| \cdot \|b\|}$$



Strategy variants:

- Display the most similar products directly
- recommend the 3 highest-priced items among the top-10 most similar

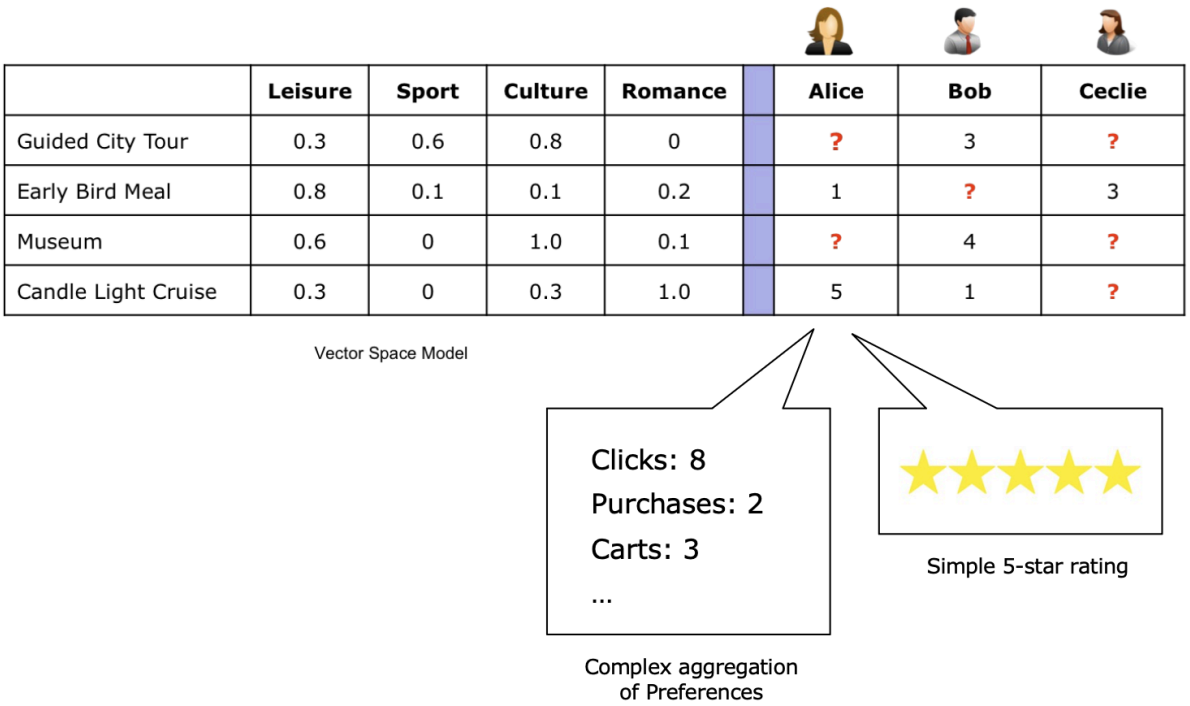
Application (cooking recipes):

- Compare instruction *text* (not ingredient list)
- Neural net encodes text → fixed-length vector
- Cosine similarity finds similar recipes
- Everything pre-processable → computationally efficient

17.2. Personalized Recommender Systems

DEFINITION: Personalized recommenders predict ratings for individual users based on their own preferences.

Goal: compute a prediction for items labeled with **?** and recommend those with the highest predicted score.



	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	?

17.2.1. Mining User Preferences

Two types of preference signals:

- **Explicit**: star ratings, likes/dislikes, wish lists
- **Implicit**: clicks, purchases, reading time, add-to-cart

Implicit signals require complex aggregation (clicks: 8, purchases: 2, carts: 3, ...)

- $\rightarrow$ must be converted into a single preference score.

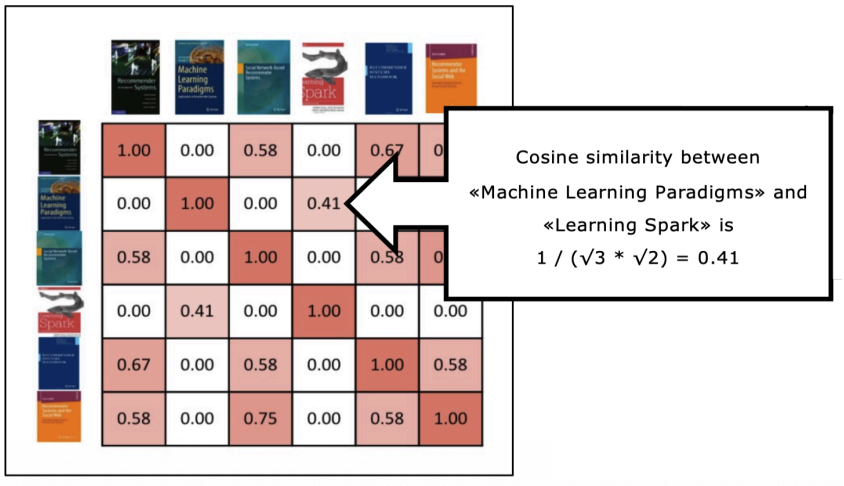
17.2.2. Structured vs. Unstructured Product Data

Structured (homogeneous catalog):

- products described by hand-crafted feature vectors
- use cosine similarity directly on feature matrix.

Unstructured (heterogeneous catalog, e.g. books):

- extract keywords with TF-IDF from titles/descriptions
- build binary term-document matrix
- apply cosine similarity.



Note: this already enables non-personalized recommendations

- just show the most similar item to whatever the user is currently viewing.

17.2.3. Personalized Content-Based Recommendations

DEFINITION: Predict how much user u would like item j by combining similarity to already-rated items with the user's own ratings.

$$P_{uj} = \frac{\sum_{i \in N_j} \text{sim}(j,i) \cdot r_{ui}}{\sum_{i \in N_j} \text{sim}(j,i)}$$

- N_j: the (at most) N rated items most similar to item j $\rightarrow$ hyperparameter
- r_{ui}: rating user u gave to item i
- Only include neighbors with similarity > 0

Example (N = 2):

- Two most similar rated neighbors both have similarity 0.58, rated 4 and 5
- Prediction: $\frac{0.58 \cdot 4 + 0.58 \cdot 5}{0.58 + 0.58} = 4.5$

17.2.3.1. Pros

User independence & transparency

- profile built only from own ratings; explainable

No new item cold start

- item similarity is independent of ratings

17.2.3.2. Cons

Limited content analysis

- product attributes may not reflect real user decisions

Overspecialization

- only recommends items similar to past purchases; no surprises

Severe new user cold start

- no ratings r_{ui} exist for new users

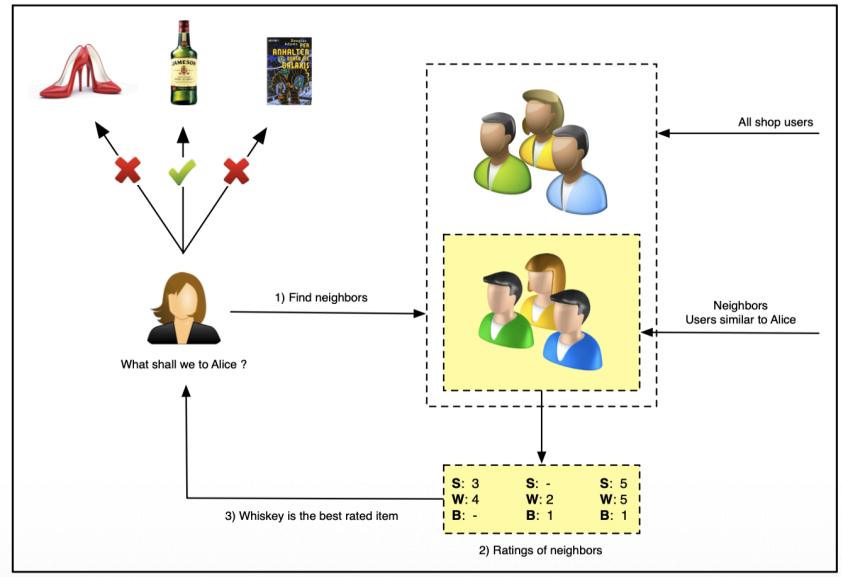
17.2.4. Collaborative Filtering

DEFINITION: **Hypothesis:** users who liked the same items in the past share the same preferences.

This is a computerized word-of-mouth process $\rightarrow$ no product attributes needed.

Steps:

1. Find neighbors (similar users / items)
2. Look at their ratings
3. Recommend the best-rated items the current user hasn't seen yet

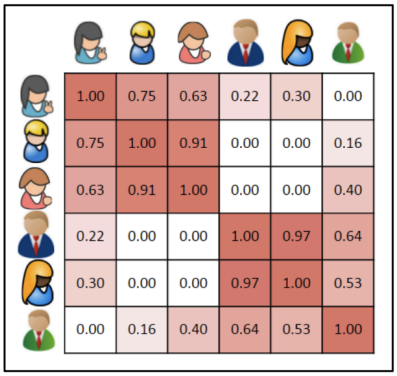


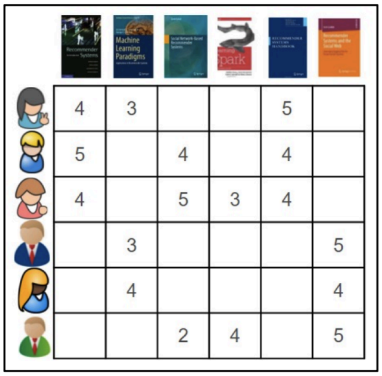
17.2.5. User-to-User Collaborative Filtering

DEFINITION: Find users similar to u (compare column vectors), then predict based on their ratings → adjusted for each user's average rating level.

		Alice	Bob	Ceclie
Guided City Tour	No product attributes necessary anymore	?	3	?
Early Bird Meal		1	?	3
Museum		?	4	?
Candle Light Cruise		5	1	?

5 Star User Ratings





$$P_{uj} = \overline{r}_u + \frac{\sum_{i \in N_u} \text{sim}(u,i) \cdot (r_{ij} - \overline{r}_i)}{\sum_{i \in N_u} \text{sim}(u,i)}$$

- $\overline{r}_u$: average rating of user u
- $\overline{r}_i$: average rating of neighbor i
- $r_{ij} - \overline{r}_i$: deviation from neighbor's average (normalizes rating style)
- $\text{sim}(u_1, u_2)$: computed with cosine similarity on column vectors
- Good neighborhood size: N = 20 (research-backed)

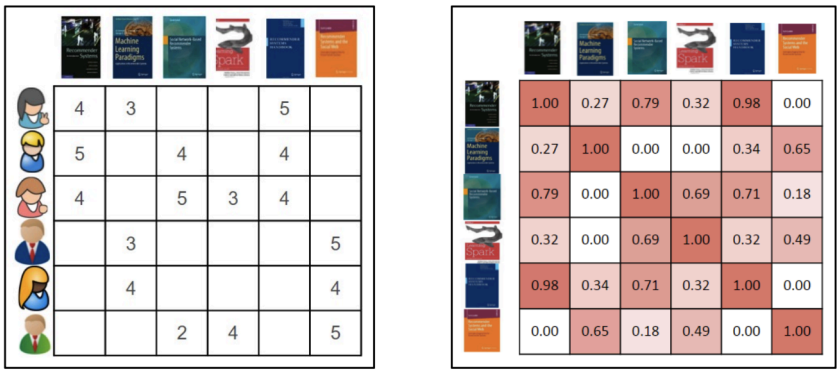
Example ($N = 2$, simplified without scaling):
- Two most similar users rated the target item 5 and 4, similarities 0.22 and 0.30
- Prediction: $\frac{0.22 \cdot 5 + 0.30 \cdot 4}{0.22 + 0.30} = 4.42$

17.2.6. Item-to-Item Collaborative Filtering

DEFINITION:

User profiles change constantly → pre-computation doesn't work.

Item-to-item similarity is stable → can be pre-computed offline.



$$P_{uj} = \frac{\sum_{i \in N_j} \text{sim}(j,i) \cdot r_{ui}}{\sum_{i \in N_j} \text{sim}(j,i)}$$

WICHTIG: The formula is identical to personalized content-based recommendations. The only difference is how similarity is computed:

Method	Similarity based on
Content-Based	Item attributes (feature vectors)
Item-to-Item CF	User ratings (row vectors)

17.2.6.1. Pros

Cross-category recommendations

- items from different categories can be linked

17.2.6.2. Cons

Needs many users

- sparse matrices → no neighbors found

New user cold start

- system must first collect ratings from new users

New item cold start

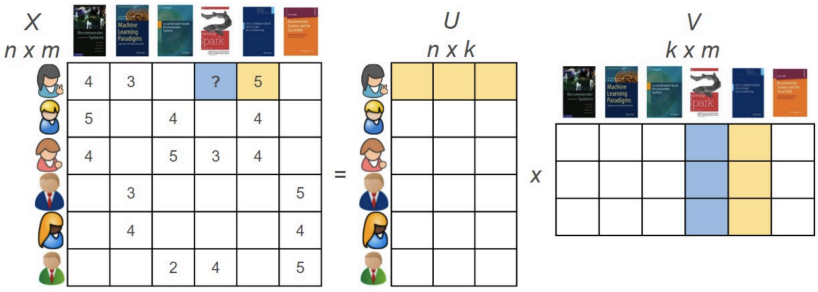
- new items need many ratings before being recommended

Grey sheep problem

- users with unusual taste get poor recommendations

17.2.7. Matrix Factorization

DEFINITION: Decompose the sparse $n \times m$ rating matrix X into two denser matrices



$$X_{n \times m} \approx U_{n \times k} \times V_{k \times m}$$

- k = **latent space** dimension (much smaller than n or m)
- Both U and V are much denser than X
- Factorization is optimized to approximate existing ratings as closely as possible
- Prediction for user u, item j = single scalar product (one matrix entry)

17.2.8. The Netflix Challenge 2006

- Open competition for the best CF algorithm on Netflix data
- Winner beat Netflix' own algorithm by 10.06% → prize of $1M donated to charity
- Netflix never implemented the winning algorithm

> The additional accuracy gains did not seem to justify the engineering effort needed to bring them into a production environment.

Lesson: Real-world engineering constraints matter more than benchmark accuracy.

17.2.9. Hybridization

DEFINITION: No single algorithm is best.

Combine them so the strengths of one compensate for the weaknesses of another.

- Key use case: **mitigate cold start by combining content-based (handles new items) and collaborative filtering (handles known users)**.

Four industry strategies:
1. **Weighted**: final score = weighted sum of scores from multiple recommenders
2. **Mixed**: show results from different recommenders side by side
3. **Cascade**: one recommender refines the output of another
4. **Switching**: use different methods in different parts of the shop

17.2.10. Evaluation of Recommenders

Offline evaluation: precision@k on validation/test set

$$\text{precision@k} = \frac{\text{relevant items among top-}k\text{ recommendations}}{k}$$

WICHTIG: Take offline scores with a grain of salt:
- A product may be irrelevant just because the user never encountered it
- The user may already own it from another shop
- Recommenders generally achieve rather low precision@k

Online evaluation: A-B testing in production

17.2.11. API: Microsoft Recommender Collection

- No recommender algorithms in scikit-learn
- Large open-source library: collaborative filtering + content-based
- GPU-enabled, neural net-based algorithms for complex preference learning
- Includes evaluation metrics and hyperparameter optimization
- Direct Azure integration

github.com/recommenders-team/recommenders