

Advanced Machine Learning

Zusammenfassung

Stephan Stofer

26. März 2021

Inhaltsverzeichnis

1 Data Classification	7
1.1 Data Quality Assessment	7
1.1.1 Data Cleaning	7
1.1.2 Approaches to DQA	7
1.1.3 Statistische Kennzahlen	8
1.2 Replacement Strategies für NULL Values	9
1.2.1 Feature Engineering	9
1.2.2 Vector Space Model	9
1.3 Pandas Profiling	10
1.4 Fazit	10
2 History & Development	11
2.1 ML in the Context of AI	11
2.2 Development of AI	11
2.3 Der Ursprung von AI	11
2.4 The Connectionists: ANN's, ML and NN	12
2.5 The Human Brain	12
2.5.1 Networks that Learn	12
2.6 Deep Learning vs. Machine Learning	12
2.7 Three Pillars of Machine Learning	12
2.8 Various Options for Hardware Acceleration	12
2.9 Performance Measure	12
2.10 Remaining Challenges	13
3 Machine Learning Fundamentals	14
3.1 Vector Space Model	14
3.2 Data Records	14
3.3 Numerical Encoding of Text	15
3.4 Distance and Similarity	15
3.5 Euclidean Distance or L^2 -Norm	15
3.6 Semantik of Similarity	15
3.7 Cosine Similarity Intuition	15
3.8 Cosine Similarity	16
3.9 Euklid vs. Kosinus	16
3.10 Manhattan Distance	16
3.11 Levenshtein or Edit Distance for Strings	16
3.12 Jaccard Similarity for Sets	17
3.13 Haversine Distance for GEO Data	17
3.14 From Points to Distributions	17
3.15 Mahalanobis Distance between Point and Distribution	17
3.16 Distance may be sensitive to Scale of Axes	17
3.17 Min-Max Normalization	17
3.18 Z-Score Normalisierung	18
3.19 Normalization Parameters	18

3.20	K-Nearest Neighbors Classification (k-NN)	18
3.21	K-Nearest Neighbors Regression	18
3.22	Hyperparameter	19
3.23	Facts on K-Nearest Neighbors	19
4	Data Preparation for Recommender Systems	20
4.1	Convert Strings to lower-case format	20
4.2	Tokenizing	20
4.3	Lemmatization	20
4.4	Stemming	21
4.5	Data	21
5	Feature Engineering	22
5.1	Data	22
5.1.1	Tabular Data	22
5.1.2	Time Series Data	23
5.1.3	Image Data	23
5.1.4	Text Data	23
5.1.5	Feature Engineering vs. Feature Learning	23
5.2	Feature Engineering for Tabular and Time-Series Data	23
5.2.1	Data Quality Assessment & Data Cleaning	24
5.2.2	Data Imputation	24
5.3	Engineering New Features	24
5.3.1	Grouping	25
5.3.2	Binning	25
5.3.3	De-skewed Data	25
5.3.4	Kernel Trick	25
5.3.5	Expert Knowledge	25
5.3.6	Transform Features	27
5.3.7	Expert Features in Time-Series	27
5.4	Image Data & Computer Vision Applications	27
5.4.1	Edges are an Important Feature	28
5.4.2	Image Segmentation	28
5.4.3	Image Denoising	29
5.4.4	Histogram Equalization	29
5.4.5	Scale Invariant Feature Transform - SIFT	29
5.4.6	Feature Learning in Computer Vision	30
5.5	Text Data & Language Applications	30
5.5.1	NLP poses really difficult problems	30
5.5.2	Linguistic Feature Engineering	30
5.5.3	Text Vectorization	31
5.5.4	Modern ML Methods	32
6	Supervised Learning - Regression	33
6.1	Variants of Regression Models	33
6.2	Measuring Regression Quality	33
6.2.1	Regression Errors for Linear Hypothesis	33
6.2.2	Regression Errors for any Hypothesis	33
6.2.3	How to Measure Regression Errors	33
6.2.4	Comparison with Mean Approximation	34
6.3	Machine Learning Quality Assessment	34
6.3.1	Generalization Error	34

6.3.2	Hyperparamters	35
6.3.3	Evaluation Workflow for Hyperparameter Optimization	36
6.3.4	K-Fold Cross-Validation	36
7	Gradient Descent	37
7.1	Fining local minima	37
7.1.1	Multidimensional Functions	37
7.1.2	Contour Lines	37
7.2	(Batch) Gradient Descent	37
7.2.1	The Idea Behind Gradient Descent	37
7.2.2	The Math Behind	38
7.2.3	Batch Gradient Descent	38
7.3	Stochastic Gradient Descent	38
7.3.1	Convergence of Gradient Descent	39
7.3.2	Batch vs. Stochastic Gradient Descent	39
7.3.3	The Effect of Learning Rate α	39
7.3.4	Stopping Criteria	40
8	Applications of Gradient Descent in Machine Learning	41
8.1	Linear Regression by Gradient Descent	41
8.2	Logistic Regression by Gradient Descent	41
8.3	Neural Net Training by Gradient Descent	42
8.4	Deep Learning by Gradient Descent	42
9	Linear Regression	44
9.1	Introduction	44
9.1.1	Regression can be used for Prediction	44
9.1.2	Terminology	44
9.1.3	Selecting the Regression Parameters	44
9.2	Ordinary Least Squares (OLS)	45
9.2.1	Defining a Cost Function	45
9.2.2	Ordinary Least Squares	46
9.2.3	Linear Regression by Gradient Descent	46
9.3	Regression Performance (R^2)	47
9.3.1	R^2 - The Coefficient of Determination	47
9.3.2	Visualize your Residuals	47
9.3.3	Correlation and R^2	48
9.4	Multiple Linear Regression	48
9.4.1	Matrix with M-Regressors	48
9.4.2	Nonlinear Regression	48
9.4.3	Regularization	48
10	Code CheatSheet	50
10.1	Python	50
10.2	Pandas	50
10.3	Numpy	50
10.4	Sklearn	50

Abbildungsverzeichnis

2.1	Entwicklung von Artificial Intelligence	11
2.2	Mehrere Optionen zur Hardwarebeschleunigung	13
3.1	Geometrische Interpretation von Daten	14
3.2	Manhattan Distanz bei Schachbrett-Muster	16
3.3	K-Nearest Neighbors Classification	18
3.4	K-Nearest Neighbors Regression	19
5.1	Feature Engineering	22
5.2	Image Data	23
5.3	Text Feature Extraction	24
5.4	De-skewed Data	26
5.5	Kernel Trick	26
5.6	Spectrogram	27
5.7	K-Means Cluster mit $k = 3$	28
5.8	Histogram Equalization	29
5.9	Features of Language and Text	30
5.10	One-Hot Encoding	31
5.11	TF-IDF	32
6.1	Vergleich Overfit vs. weiche Approximation	35
6.2	Einfacher Machine Learning Workflow	35
6.3	Evaluation Workflow Hyperparameter Optimization	36
6.4	10-Fold Cross-Validation	36
7.1	Batch Gradient Descent Example	38
7.2	Stochastic Gradient Descent	38
7.3	Convergence of Gradient Descent	39
7.4	Batch vs. Stochastic Gradient Descent	39
7.5	Stopcriterias	40
8.1	Lineare Regression by Gradient Descent	41
8.2	Logistic Regression by Gradient Descent	42
8.3	Neural Net Training by Gradient Descent	43
8.4	Deep Learning by Gradient Descent	43
9.1	Linear Regression	44
9.2	Terminology in linear Regression	45
9.3	The Cost Function in Matrix Form	45
9.4	Konvexe Funktion	46
9.5	Partial Derivative OLS	46
9.6	OLS Example	47
9.7	When fit Linear Regression	47
9.8	Visualize the Residuals	48
9.9	OLS - M Regressors	48

1 Data Classification

Daten werden in zwei Klassen unterteilt. *Numerische* und *Kategorische* Daten. Bei numerische Daten gibt es *stetige* oder *diskrete* Zahlen. Bei Kategorischen sind entweder *ordinal* oder *nominal*. Ordinale haben eine Hierarchie.

1.1 Data Quality Assessment

Daten sind sehr wichtig, der beste ml-Algo nützt nicht, wenn Daten rubbish sind. Mögliche Fehlerquellen:

- Technische Fehler
- Qualität
- schlecht Design
- menschliche Fehler
- Input in Web-Apps (ungeprüfte Eingabefelder)
- Exporte der Daten, falsche Formate - oder Pre-Processing
- Falschangaben durch Benutzer
- Daten haben immer ein Ablaufdaten! (z.B. emailadressen, Adressen)

Ein DQA kommt immer zuerst! Schützt auch die Reputation gegenüber Kunden.

1.1.1 Data Cleaning

Prozess um Fehler in Daten zu beheben (automatisch)/bereinigen. Duplikate entfernen, null-values entfernen, Datenformate ml-friendly aufbereiten (data wrangling). Die Änderungen müssen dokumentiert und versioniert werden, den data provider darüber informieren und die Ursache für die data quality issues untersuchen.

1.1.2 Approaches to DQA

Dies ist detektiv-Arbeit. Wenn etwas verdächtig erscheint, weitergraben! Die Daten werden überprüft, ob sie vertrauenwürdig sind (plausibilisieren).

- Datenquellen und vertrauenwürdigkeit prüfen
- statistische Kennzahlen interpretieren
- daten visualisieren
- Datenranges prüfen (Alter sollte unter 200 sein, Salär > 0, usw.)
- Korrelation zwischen Attributen prüfen (Tachostand und Preis eines Autos)
- Redundanz -> je weniger umso bessere Daten
- Anomalieprüfung in Syntax und Semantik
- NULL Werte und Duplikate erforschen

1.1.3 Statistische Kennzahlen

Geben uns einen Fingerabdruck und erste Plausibilisierung der Daten. Die wichtigsten Kennzahlen sind:

- Mittelwert - $mean$ - $O(n)$
- Modus - $mode$ die Zahl die am meisten vorkommt
- Median - $median$ - $O(n * \log n)$, ist aussagekräftiger

1.1.3.1 Schiefe

Der Mean, Modus und Median geben Auskunft über die Schiefe der Daten. Wir haben eine negative, Linksschiefe $skewness$ wenn $mean - mode < 0$, wenn positiv, Rechts-Schiefe $mean - mode > 0$

1.1.3.2 Median

Sortiere Datenreihe. Der Median enthält 50% der Daten. Die Quantile entsprechen je 25%. Die Interquartils Differenz (IQR) entspricht $Q3 - Q1$.

1.1.3.3 Boxplots

Sehr nützlich zur grafischen Darstellung. *Outliers* sind die Werte die grösser sind als $Q3 + 1.5 * IQR$ respektive $Q1 - 1.5 * IQR$. Minimum bzw. Maximum sind die Werte, die gerade noch ind diese Grenze $1.5 * IQR$ reinpassen.

Wenn viele Outliers müssen Daten genau angeschaut werden, ob sie trotzdem plausibel sind.

1.1.3.4 Five Number Summary of a Data Distribution

In mit Python kann sehr einfach die $Q1, Q2, Q3$, min und max einer Datenreihe ausgegeben werden:

```
import numpy as np
import pandas as pd

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

Auch Boxplots sind sehr einfach:

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

1.1.3.5 Datenverteilung

Die Verteilung wird mit der Varianz betrachtet, wobei diese *sample variance* die Besselkorrektur $(n - 1)$ nutzt. Die Standardabweichung entspricht aus der $\sqrt{Var(x)}$

1.1.3.6 Kovarianz

Die Kovarianz zeigt die Variabilität von zwei Datensätzen auf. Ist der Wert positiv, verhalten sich die beiden Daten ähnlich. Ist sie negativ, entsprechend nicht. Ist aber schwierig zu interpretieren, weil sie nicht normiert ist.

1.1.3.7 Covarianzmatrix

Die covarianzmatrix ist sehr wichtig in ML. Sie enthält alle Covarianzen aller Varianzpaare. Die Diagonale kann durch die Varianz von X ersetzt werden.

1.1.3.8 Pearson Korrelation

Covarianz wird durch die Standardabweichung dividiert. Deshalb ergeben sich Werte zwischen 1 (perfekte Korrelation) und -1 (perfect anti-correlation). Damit kann die Datenreihe verglichen werden. Die Korrelationsmatrix kann als Heatmap gut dargestellt werden.

1.2 Replacement Strategies für NULL Values

Kommen immer wieder vor. ML-Algos können selten damit umgehen und müssen bereinigt werden. Je nach Datenumfang sind versch. Verfahren denkbar:

- Zeilen mit NULL Werten löschen
- Fehlende Daten manuell einsetzen
- Globale Konstanten einsetzen (UNKNOWN, *infty*)
- Tendenzen verwenden (Mittelwert für symmetrische Daten, Medien für Schiefedaten)
- Tendenzen auch pro “Klasse” (Eigenschaften) berechnen (z.B Krebskranke und gesunde Patienten)
- Regressionsmodell (sehr aufwändig und ungewohnt in Praxis)

1.2.1 Feature Engineering

Features entsprechen Spalten. Null-Values können also mit ML erzeugen. Information verfügbar für ML-Algo machen.

1.2.2 Vector Space Model

Entspricht einem Datenset welches ausser dem Key nur numerische Werte enthält. Kategorische Daten können sehr einfach in nummersiche Daten transformiert werden. Zum Beispiel werden die Farben alle zu Spalten und entsprechende Zugehörigkeit mit 1 bzw. 0 gekennzeichnet. Diese werden als Dummy-Variable bezeichnet.

```
import pandas as pd
data = pd.read_csv('cars.csv')
data = pd.get_dummies(data)
```

Python code um Daten entsprechend aufzubereiten.

1.2.2.1 Dummy Variable Trap

Mit dem einfügen von Dummy-Variablen muss die *Multikollinearität* im Auge behalten werden. Wenn n -Dummy Variablen erzeugt werden und $n - 1$ Spalten alle 0 sind, wissen wir zu 100, dass die nte Spalte 1 sein muss. Dies führt zu unterterminierten Matrizen. Die Matrix kann nicht invertiert werden. Um das zu verhindern, muss eine Spalte gelöscht werden! Es gibt aber Verfahren, die immun dagegen sind (z.B. Entscheidungsbäume).

1.3 Pandas Profiling

Effizient in drei Zeilen Code!

```
import pandas_profiling  
data = pd.read_csv('cars.csv')  
data.profile_report()
```

1.4 Fazit

Bei jedem ML-Projekt ist in Data Quality Assessment Pflicht

2 History & Development

Einführung in die Geschichte und Entwicklung von Machine Learning

2.1 ML in the Context of AI

Machine Learning ist ein Teil von AI, welche normalerweise Menschliche Intelligenz benötigt. Um dies zu erreichen wird ML betrieben. Es lernt ohne explizit programmiert zu werden.

2.2 Development of AI

Mit dieser Disziplin wurde ungefähr 1950 erste Versuche gestartet. Es folgten *Rules Based Systems* und *Logic & Reasoning* zwischen den 1960 bis Anfang 1980.

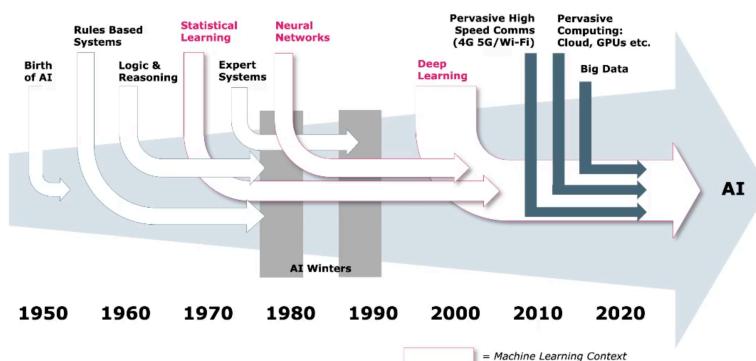


Abbildung 2.1: Entwicklung von Artificial Intelligence

2.3 Der Ursprung von AI

In 1945 erschien erster programmierbarer Computer *ENIAC*. Es entstand eine erste Welle mit viel Forschung und Entwicklung bis Anfang 70er. Die Meinung war, dass Computer alles lösen können. Man merkte aber, dass sie zwar komplexe Aufgaben effizient lösen konnten, jedoch nicht greifen oder etwas erkennen können. Funding wurde immer weniger.

Es ist einfach einem Computer Intelligenz eines Erwachsenen Menschen beizubringen, jedoch sehr schwierig oder unmöglich ihm die Fähigkeiten eines Einjährigen Kindes, wie Wahrnehmung oder Mobilität, beizubringen.

– Moravec Paradox

Während der zweiten Welle *Expert Systems* wurden vor allem mit bestehenden Wissen gearbeitet und mit den verfügbaren Wissen etwas zu machen. Auch diese Welle wurde durch den zweiten *AI Winter* ausgebremst. > Die Hauptlektion aus 35 Jahre AI-Forschung ist, dass komplexe Probleme sind einfach und die Einfachen sehr schwierig. > > – Steven Pinker

1988 kam die Idee auf von Daten und Erfahrungen zu lernen. Mit dem vermehrten Einsatz von Wahrscheinlichkeiten *Bayesian Networks* kam der Erfolg zurück.

2.4 The Connectionists: ANN's, ML and NN

Artificial Neural Networks (ANN's), Machine Learning und Deep Learning (NN). Wie lernt man überhaupt? Das Gehirn ist sehr flexibel und kann aus verschiedenen Inputs lernen und diese Sinne weiterentwickeln.

2.5 The Human Brain

Das menschliche Gehirn hat verschiedene Areale die für unterschiedliche Funktionen verantwortlich sind. Es ist massiv vernetzt und verändert sich ständig (durch lernen/Stimulation). Alle Regionen sind aus dem gleichen Neuralen Netz. Jedes Neuron ist mit 1000 bis 10000 anderen verbunden.

2.5.1 Networks that Learn

1887 wurden die Neuronen als Fundamentals Nervensystem entdeckt. 1943 wurde das Neuronenmodell als logische Einheit modelliert. *If* meine Synapsen etwas wahrnehmen (Threshold überschreiten), *then* sende (abfeuern) ich ein Signal durch das Axon, *else* sende ich nichts. Wenn mehrere Neuronen das gleichzeitig feuern, verbinden sie sich - und so entsteht Lernen. 1958 Frank Rosenblatt übertrug die Idee in Computer Vision und erfand das *Perceptron*, ein einzelner Neuralen Network Layer.

2.6 Deep Learning vs. Machine Learning

Machine Learning benötigt menschliches Feature extraction um zu klassifizieren. Deep Learning nutzt grosse neuronale Netzwerke und lernt mit Daten, Algorithmen und Processing Power.

2.7 Three Pillars of Machine Learning

Data, Algorithmen und Porcessing Power. Was natürlich nicht fehlen darf ist Highspeed Kommunikation, welches all die anderen Punkte begünstigt (Concrete).

2.8 Various Options for Hardware Acceleration

2.9 Performance Measure

F1-Score Performance bewertet AI Models auf Basis ihrer Erkennungsrate in Prozent.

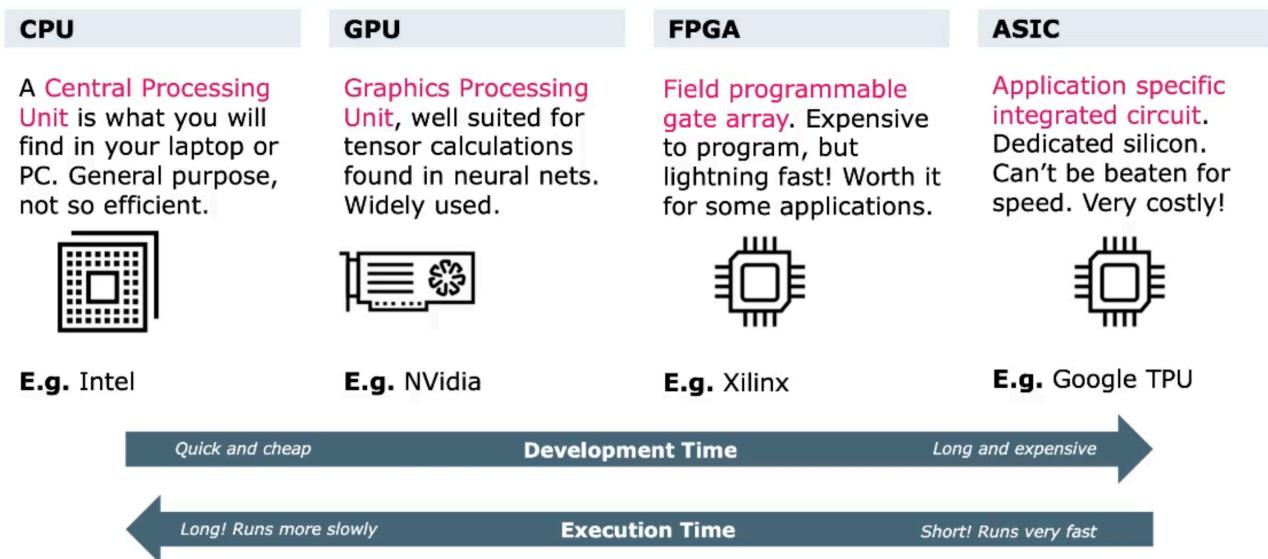


Abbildung 2.2: Mehrere Optionen zur Hardwarebeschleunigung

2.10 Remaining Challenges

- Datenprobleme sind die Größten Fallstricke in ML-Projekten. 80% des Aufwandes geht auf Datenaufbereitung und lediglich 20% fürs Modelling
- Daten haben Tendenzen, möglichst ausgeglichene Daten verwenden
- ML/DL Training ist sehr Energieintensiv

3 Machine Learning Fundamentals

Grundlagen des ML

3.1 Vector Space Model

Ein Vektor Space Model enthält nur numerische Daten (ausser dem Key). Zum Überführen von kategorischen Daten gibt es mehrere Techniken für die Transformation. Von nun an gehen wir jeweils davon aus, dass die Daten numerisch vorhanden sind. Der Key wird normalerweise nicht zu den Daten gezählt.

3.2 Data Records

Beinahe alle ML-Verfahren setzen ein *VSM* voraus. Wenn alle Daten als nummersiche Werte voriegen, kann eine jede Zelle als Spalte interpretiert werden.

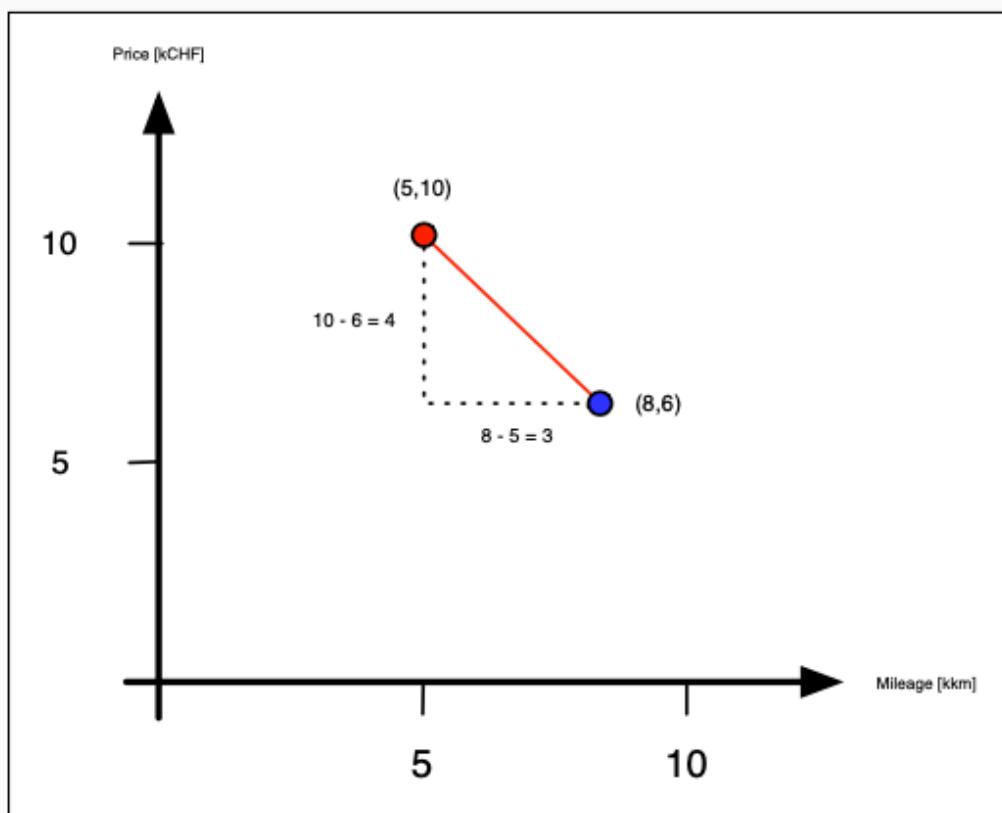


Abbildung 3.1: Geometrische Interpretation von Daten

Die Länge der roten Linie kann mit Hilfe von $\sqrt{a^2 + b^2}$ berechnet werden und als Distanz interpretieret werden.

3.3 Numerical Encoding of Text

Kann man auch ganze Texte oder Bilder in ein Vector Space Model überführen? Bei Bilder können mit Hilfe von Filter transformiert werden. Bei Text funktioniert es mit Hilfe des **TF-IDF Scores** - *term frequency-inverse document frequency*. Diese verbinden Wörter zu numerischen *Vektoren*. Als Basis dient dazu einen *Referenz-Korpus*. Dazu nehmen man alle Wikipediaartikel und stellt diese in Kolonnen dar. Jedes Wort wird nun gemessen anhand der Häufigkeit im ersten Artikel. Danach dividiert durch das Reziproke vom Gesamtwert dieses Wortes über alle Artikel.

3.4 Distance and Similarity

Das Inverse der Distanz ist die Similarität. Je weiter voneinander entfernt, desto unterschiedlicher, respk. je näher umso ähnlicher (Similarität). Auf diesem Konzept basieren alle ML-Algorithmen ausserhalb von Neuronalen Netzen und evtl. Entscheidungsbäume. Beispiele für Distanz und Similarität sind:

- Dating-Site empfiehlt anderes Profil mit den ähnlichsten Vorstellungen
- Auto-Verkaufseite schlägt Preis für Auto anhand der 20 letzten Verkäufe dieses Modell vor
- Webshop empfiehlt Produkte anhand ähnlicher Warenkörbe anderer Benutzer

Wir brauchen also einen Weg die Distanz/Similarität aussagekräftig zu berechnen.

3.5 Euclidiean Distance or L^2 -Norm

Ist eine generalisierte Form der Pythagoras Formel welche für eine beliebig grosse Anzahl an Dimensionen verwendet werden kann. Ist eine Zahl $[0, \infty[$ und findet den ähnlichsten Punkt durch die *Minimierung der Distanz* zwischen zwei Punkten. Die Euklidische Distanz wird auch L^2 -Norm genannt.

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

3.6 Semantik of Similarity

Je nach Daten ergibt die Distanz nicht exakt wider, wie wir das erwarten würden. Bei unterschiedlichen Anwendungszwecke genügt die Similarität nicht den Ansprüchen. So würde die Distanz bei einem Text der Copy-Pasted würde weit auseinander liegen (obwohl gleicher Inhalt). Wir müssen also semantisch sinnvoll die richtige Methode auswählen. Die Distanzfunktion muss auf die Domäne angepasst werden.

3.7 Cosine Similarity Intuition

Die Kosinus Similarität misst den Winkel θ zwischen zwei Vektoren X und Y . Zwei Punkte auf einer Geraden haben den Winkel 0, aber hätten einen grossen *Euklidische Similarität*.

Die Similarität kann bei vielen Algorithmen per Parameter übergeben werden.

3.8 Cosine Similarity

Durch das Normieren $\frac{X}{\|X\|_2}$ der Vektoren werden die Punkte auf dem Einheitskreis abgebildet. Das Skalarprodukt entspricht der Projektion eines Vektors Y auf den Vektor X. Der Kosinus liegt zwischen [1, 1], für die Distanz verwenden wir aber ein positiven Wert, weshalb wir den Betrag verwenden, wird Kosinus Similarität. 0 bedeutet kleinste Distanz - maximale Similarität und 1 grösste Distanz - Dissimilarität.

$$\text{Kosinusdistanz} = 1 - \text{Kosinus Similarität}$$

3.9 Euklid vs. Kosinus

Die beiden Varianten können ganz unterschiedliche Resultate liefern. Werden die Punkte auf den Einheitskreis normiert, ergeben beide Verfahren dasselbe Resultat.

3.10 Manhattan Distance

Auf einem Schachbrett oder auf der Strasse von Manhattan kann nicht die Euklidische Distanz zur Bestimmung der Entfernung verwendet werden, da man keiner Geraden folgen kann. Die Manhattan Distanz ist definiert durch

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

Der Wertebereich der Distanz liegt zwischen [0, ∞ [

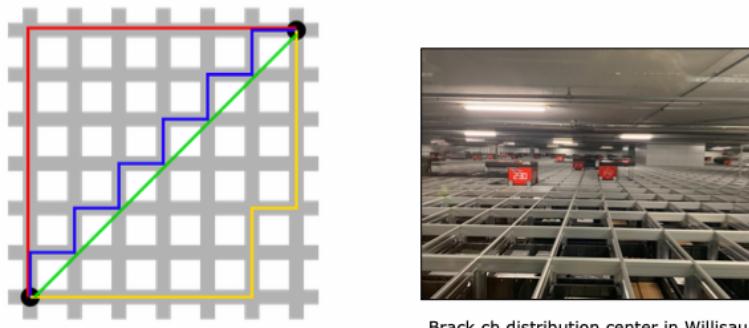


Abbildung 3.2: Manhattan Distanz bei Schachbrett-Muster

3.11 Levenshtein or Edit Distance for Strings

Die Levenshtein Methode zählt die minimale Anzahl an nötigen Operationen an einem Wort um dieses in ein anderes zu überführen. Jede Operation gibt Strafpunkte und werden addiert. Die Summe entspricht der *Levenshtein Distance*. Je grösser umso weniger Similarität zwischen den beiden Wörtern. Die Operationen werden wie folgt gewertet:

- +1, wenn Buchstabe gelöscht wird
- +1, wenn Buchstabe hinzugefügt wird
- +2, wenn Buchstabe ausgetauscht wird (löschen und hinzufügen) - manchmal wird nur +1 gewertet

Die effiziente Implementierung ist sehr Herausfordernd! Nichts desto trotz, sehr wichtiger und oft verwendeter Algorithmus, zum Beispiel als Wörterbuch. Betrachtet bei Fehler alle Wörter mit kleinster Distanz (Abweichung) und schlägt diese als Korrektur vor.

3.12 Jaccard Similarity for Sets

Betrachtet die Menge einzelnen Wörter und dividiert sie durch die Gesamte Anzahl an Wörter. Der Wertebereich liegt zwischen [0, 1] und ist definiert durch

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

Sie kann bei Recommender Systemen angewendet werden. Zum Beispiel wenn zwei Warenkörbe gleiche Produkte enthalten oder wenn zwei Texte gleiches Jargon verwenden.

3.13 Haversine Distance for GEO Data

Flugzeuge reisen nach dieser Distanz, da sie die Krümmung der Erde berücksichtigt. Sie misst in der Atmosphäre die Distanz, ergo ist die Similarität klein, wenn die Distanz gross und vice-versa.

3.14 From Points to Distributions

Manchmal müssen auch Verteilungen beurteilt werden und nicht nur die Punkt zu Punkt Distanz. Es gibt auch die distribution-to-distribution, welche bei der Bildanalyse verwendet wird.

3.15 Mahalanobis Distance between Point and Distribution

Misst die Entfernung des Punkts als wie viele Standardabweichungen der Punkt vom Mittelwert entfernt liegt (Art Euklidische Distanz).

- Erst wird die Verteilung normalisiert indem korrelierende Variablen in unkorrelierende transformiert werden
- dann wird skaliert, dass die Varianz gleich 1 wird
- zum Schluss wird die euklidsche Distanz zwischen Punkt X und dem Mittelwert berechnet

3.16 Distance may be sensitive to Scale of Axes

Die Einheit von Features können Daten verfälschen (zB. km -> m). Sie dominieren dann das Ergebnis einer anderen Einheit die viel kleiner ausfällt. Deshalb müssen vor dem betreiben von Machine Learning Daten normalisiert werden. Idealerweise haben alle Feature den gleichen Wertebereich.

3.17 Min-Max Normalization

Variante um Daten zu normalisieren und transformiert die Werte ins Intervall [0, 1]. Der Grösste Werte in der Spalte erhält den Wert 1, der kleinste der Wert 0. Die Werte dazwischen werden skaliert.

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

Dies erlaubt die Interpretation in Prozent und man hat keine negativen Werte. Kann aber nicht für *supervised learning* verwendet werden, weil min/max nicht bekannt sind (nach dem Training).

3.18 Z-Score Normalisierung

Die Daten werden so transformiert, dass der Mittelwert 0 ist und die Standardabweichung 1.

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

Kann für supervised und unsupervised learning verwendet werden. Der Nachteil ist die fehlende Prozentinterpretation. Kann zu negativen Werten führen (negative Preise oder Anzahl von etwas). Zur Interpretation müsste zurücktransformiert werden.

3.19 Normalization Parameters

Bei Min/Max Normalisation werden (min/max) benötigt, beim Z-Score (mean, std). Wichtig, Parameter aus Trainings-Daten bestimmen (nicht Testdaten), Min/Max nur für supervised learning verwenden (ausser Daten enthalten globale Min/Max). Normalisierten Parameter speichern um später zu Skalieren.

3.20 K-Nearest Neighbors Classification (k-NN)

Ist wahrscheinlich der einfachste ML-A. Ist ein Verfahren für Regression und Klassifizierung. Zeigt auf, wie wichtig Distanz bzw. Similarität für ML ist. K-Nearest Neighbors muss ein von Hand definierten Parameter k mitgegeben werden. $k = 3$ bedeutet, dass K-Nearest die drei nächsten Punkte suchen und die Klassifikation eines neuen Punkts anhand deren Eigenschaften klassifiziert wird. k-NN mit $k = 1$ wird einfach das Labels des nahesten Punktes übernommen. Bei $k > 1$ wird ein Mehrheitsvoting gegenüber den k -nahesten Punkten gemacht.

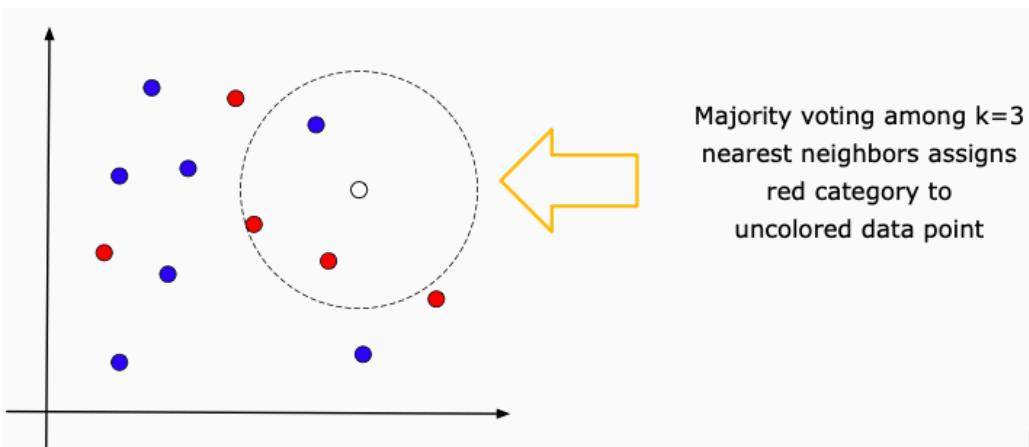


Abbildung 3.3: K-Nearest Neighbors Classification

3.21 K-Nearest Neighbors Regression

Löst Regressionsproblem zum Beispiel Preisvorhersage aus einem Regressionsmodell der k -nahesten Punkte. Funktionsweise mit Parameter k analog k-NN-Klassifizierung.

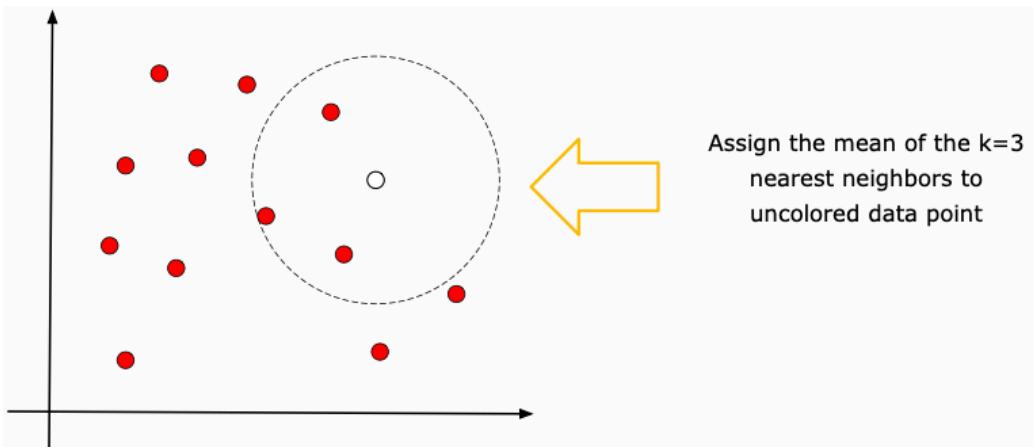


Abbildung 3.4: K-Nearest Neighbors Regression

3.22 Hyperparameter

Der Wert von k wird Hyperparameter genannt und entspricht der Anzahl Nachbarn. Das Resultat kann stark vom gewählten k differieren. Es ist also wichtig dieses möglichst optimal zu wählen (es gibt keine Optimierungsmöglichkeit durch einen Compi). Als weiteren Parameter kann die Distanz/Similaritätswert übergeben werden.

3.23 Facts on K-Nearest Neighbors

- Sehr langsam, weil jedesmal Similarität zu allen Punkten berechnet und dann die Distanz berechnet und nahesten ausgewählt. Dies wird bei jedem Datenpunkt gemacht
- Für kleine Dataset gute Baseline, legt Massstäbe fest für andere Algos
- Mehrheitswahl bedeutet alle Nachbarn sind gleich stimmberechtigt, egal wie weit sie entfernt liegen
- Alternativ kann per Parameter die Distanz berücksichtigt werden $\frac{1}{d}$ mit $d = \text{distance} > 0$
- benötigt am wenigsten Daten, reichen Daten nicht aus, gibt es keine Möglichkeit für ML
- k ist sogenannter Hyperparameter

4 Data Preparation for Recommender Systems

Um die Daten (strings) verarbeiten zu können müssen diese erst Vorverarbeitet werden

4.1 Convert Strings to lower-case format

Text in lower-case Buchstaben umwandeln damit gleiche Worte immer gleich geschrieben werden.

```
df['description'] = df['description'].str.lower()
```

4.2 Tokenizing

Tokenizing ist ein Verfahren um einen Text zu bereinigen. Das Resultat der Tokenisierung ist eine Liste von Tokens, die als Liste im technischen Sinn, oder als Abfolge von durch Zeilenumbrüche getrennte Tokens. De- ren eine Tokenklasse angehängt wird. Während diesem Vorgang werden einige Einzelaufgaben bewältigt. Die Tokens werden auch als *Stop Words* bezeichnet.

- Abkürzungen erkennen und isolieren (es gibt auch gleiche Abkürzungen für unterschiedliche Worte)
- Interpunktionen und Sonderzeichen erkennen (Problem diverse Sonderzeichen wie /, @, #, \$, usw. gehören oft einem Token an und dürfen nicht isoliert werden)
- kontrahierte Formen expandieren; l'auto → la auto; gilt das nachher als Artikel oder Pronomen? Führt zu Ambiguität.
- komplexe Tokens erkennen und isolieren; allgemeine Zahlen wie 10 000, Telefonnummer, Datum und Zeit, URLs, Vor- und Nachnamen (was ist mit Titel?)
- ggf. Tokens normalisieren
 - Abkürzungen vereinheitlichen
 - Datums, Zeit- und Massangaben vereinheitlichen
 - Zahlen
- ggf. Tokens klassifizieren (d.h. Tokenklassen bilden); Klassen wie number, date, time, abbr, currency, temp, length usw. bilden
 - Diese Schuhe haben sFr. 147.- gekostet. → [diese,schuhe,haben,currency(147,sfr,Rp),gekostet,,]
 - Wir treffen uns am 24. April um 15 Uhr. → [wir,treffen,uns,am,date(24,4,Jr),um,time(15,Min,Sec),,]

4.3 Lemmatization

Die Lemmatisierung ist das Rückführen von Wörtern in ihre Grundform. So wie sie im Wörterbuch stehen, diese werden als *Lemma* bezeichnet. Das ursprüngliche Wort ist die *Vollform*.

4.4 Stemming

Als Stemming wird die Stammformreduktion oder Normalformenreduktion bezeichnet. Es ist ein Verfahren um verschiedene morphologische Varianten eines Wortes auf ihr gemeinsamen Wortstamm zurückzuführen. Zum Beispiel der **Deklination** von *Wortes* oder *Wörter* zu *Wort* und **Konjugation** von *geseheen* oder *sah* zu *sehen*. Bekanntes Framework ist das Snowball von Martin Porter.

4.5 Data

Daraus ergeben sich normalisierte Daten die für das Training und den Test verwendet werden. Das Set ist immer in die zwei Gruppen Test und Training aufzuteilen.

5 Feature Engineering

Ist eine Manipulation an unseren Rohdaten um höhere Performance und bessere Ergebnisse im Kontext von Machine Learning zu erreichen. Kurz, wir möchten es für die Algorithmen einfacher machen.

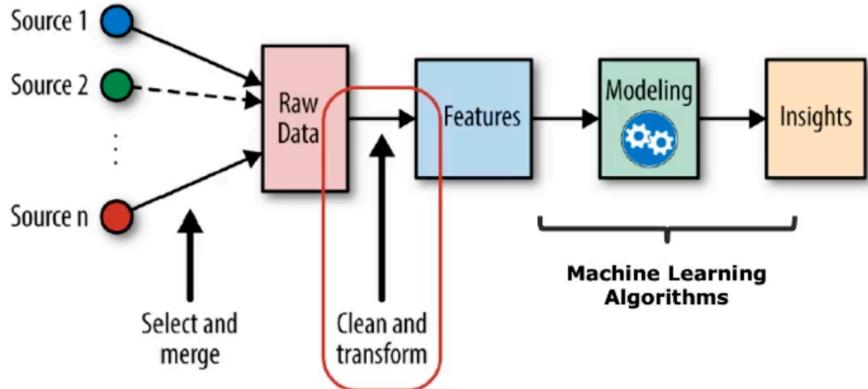


Abbildung 5.1: Feature Engineering

Kann die Performance der ML-Algos verbessern. Domänenexperten können mit Fachwissen helfen, nützliche Features zu extrahieren. In modernen DL-Ansätzen werden Feature *gelernt* und nicht engineered.

5.1 Data

Als Menschen können wir sehr schnell aus einem Datenset eine Klassifizierung machen, indem wir ihre Eigenschaften miteinander vergleichen. Diese Eigenschaften werden *attributes* oder *feature* genannt. Eigenschaften können sein:

- Farbe
- Form
- usw.

5.1.1 Tabular Data

Meist sind Daten in tabellarisch verfügbar. Jede Zeile entspricht einem Dateneintrag und wird als ein *Datenpunkt* (*data point*) interpretiert. Die Zeilen enthalten *features* der verschiedenen Datenpunkte. Sofern die Daten nummerisch sind, können wir sie als *multi-dimensionalen Vektoren* in einem *feature space* interpretieren. Die Distanz zwischen zwei Punkten entsprechen einer Similarität (je näher umso ähnlicher).

5.1.2 Time Series Data

Zeitfolgen sind eine Spezialform von tabellarischen Daten welche in einer *chronologischen Sequenz* vorliegen. Eines der Feature ist dabei ein Set aus Zeitstempel. Die Abfolge muss vollständig sein um die Daten korrekt interpretieren zu können. Audiodaten oder Börsenentwicklungen sind gute Beispiele für Zeitfolgen. Diese Sequenzen dürfen nicht gemixt werden, da sie sonst nicht mehr dem Original entsprechen.

5.1.3 Image Data

Auch Bilder sind tabellarische Daten. Ein Schwarweiss-Foto enthält Nummern, welche die *Pixelintensität* repräsentieren im Range von [0, 255] (bei 8bit, 2^8). SW-Bilder haben ein Layer an Pixel, farbige (RGB) haben drei Ebenen. Ein Bild kann auch binär sein, 0 entspricht Schwarz, 1 entspricht weiss.

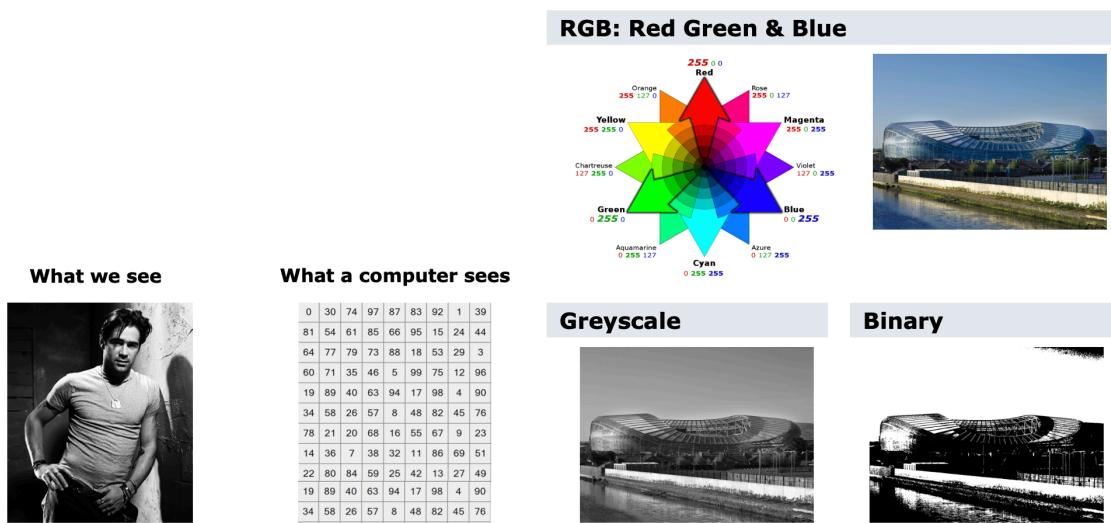


Abbildung 5.2: Image Data

5.1.4 Text Data

Die menschliche Sprache ist eine hohe kognitive Kunst und macht es für Maschinen sehr schwierig von Text zu lernen. Aber Text ist überall und häufig die Kerndaten in einem ML-Projekt. *Rohe Textdaten* ist oft nicht direkt für ML brauchbar und deshalb idealer Kandidat für *Feature Engineering*.

5.1.5 Feature Engineering vs. Feature Learning

Moderne *Deep Learning* Algorithmen können Features direkt aus den Daten lernen. Aber häufig stehen nicht die umfangreichen Daten zur Verfügung die nötig wären um DL zu betreiben, welch auch gelabelt sein müssten. Außerdem bietet die Genialität und Expertise des Menschen mehr als DL kann.

5.2 Feature Engineering for Tabular and Time-Series Data

Tabellarische Daten werden auch *Panel Data* genannt. Daher hat auch das Python Package *Pandas* seinen Namen (**Panel Data**).

An example of text feature extraction

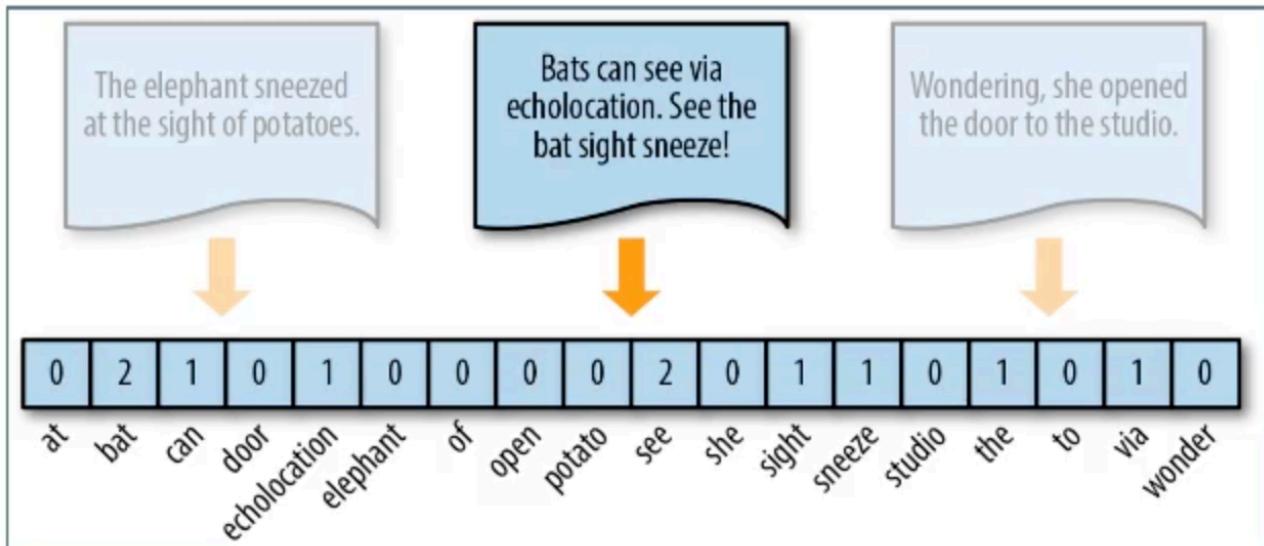


Abbildung 5.3: Text Feature Extraction

5.2.1 Data Quality Assessment & Data Cleaning

Jedes Projekt startet mit einem DQA und Data Cleaning. *Daten-Normalisierung* ist auch entscheidend um einen dynamischen Wertebereich zwischen den Features zu vereinheitlichen. Die wichtigsten Dinge nochmals erwähnt:

- Originaldaten sicher aufbewahren
- Änderungen an den Daten dokumentieren
- Duplikate entfernen
- Irrelevante Daten entfernen
- Falsche bzw. unterschiedliche Bezeichnungen vereinheitlichen
- Outliers (Ausreisser) auf kausalität prüfen
- Fehlende Daten mit Mittelwert, Median füllen

5.2.2 Data Imputation

Der Prozess um fehlende Daten mit Werten die wir «raten» zu ersetzen wird *Imputation* genannt. Es ist ein Risiko, weil die Schlussfolgerung (Konklusion) verfälscht werden kann. Mit Vorsicht ausführen und nur, falls wenige fehlen. Sonst Feature komplett entfernen.

Nullwerte sind häufig mit *NaN* oder *NA* vermerkt. Mit der Pandas-Methode `df.fillna()` können nulls mit 0, dem *Median*, *Mean*, o.ä. ersetzt werden.

5.3 Engineering New Features

Mit den Techniken von *Grouping* und *Binning*.

5.3.1 Grouping

Mit Datengruppierung können wir *kategorische* Daten mit ähnlichen Typen zusammenfügen.

	Original Feature	New Feature
Bezeichnung	Title of Person	Gender of Person
Values	Mr., Sir, Miss, Lady, Mrs., Fr., Prof., Mx.	Male, Female, Unspecified

5.3.2 Binning

Mit Daten-Binning können *nummerische* Werte in Bereiche gliedern (in einem Bin).

Beispiel von Zeiten am Boston Marathon

	Original Feature	New Feature
Bezeichnung	Finishing Time	Finishing Group
Values	beliebige Zeit	2-3h, 3-4h, >4h

Zweck kann sein, die Leute zu finden, die nächstes Jahr ein anderen Startblock erhalten.

5.3.3 De-skewed Data

Wenn wir Daten binnen, werden einige Bin's mehr Einträge haben als andere. Solche Daten werden schief genannt (skewed). Um das zu beheben können wir auf den Originaldaten den Logarithmus anwenden und die Schiefe wird oft entfernt. Dies muss aber **vor** dem Binning gemacht werden. Hint: Den Logarithmus kann man nur von positiven Zahlen ziehen. Ergo müssen negative Zahlen erst gemappt werden.

5.3.4 Kernel Trick

Manchmal sind die Daten die ein Feature beschreiben schwierig zu analysieren. Dann kann man ein *Kernel*-Feature hinzufügen. Im Beispiel auf 5.5 fügen wir das Feature $z = x^2 + y^2$ hinzu. Dies wird *Kernel Trick* genannt. Die Kernel-Funktion transformiert die Daten in einen höher dimensionalen Featureraum. Die Daten können einfacher analysiert werden.

5.3.5 Expert Knowledge

Expertise ist eine der grösste Schlüssel um Daten zu verbessern. Als Beispiel der *Buffett Indicator*, welcher die Entwicklung von Börsenwerte mit einer eigens entwickelten Formel aus Indikatoren voraussagen kann. Dazu brauchte er Expertenwissen um das neue Feature, seinen Indikator, zu generieren. Dies wäre nicht offensichtlich nur aus den Rohdaten.

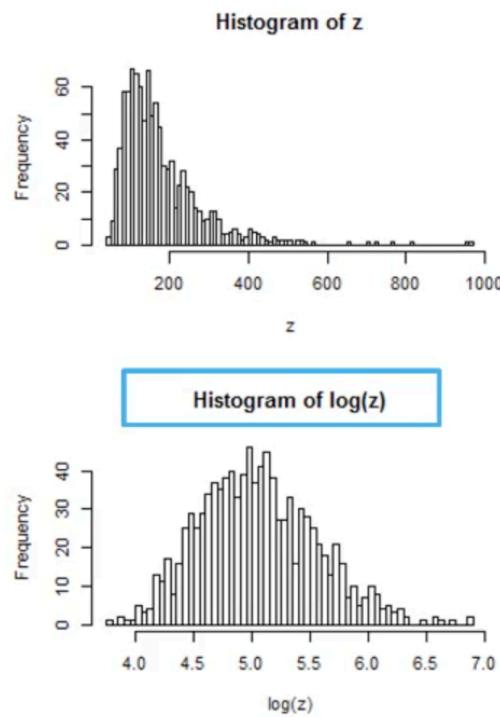


Abbildung 5.4: De-skewed Data

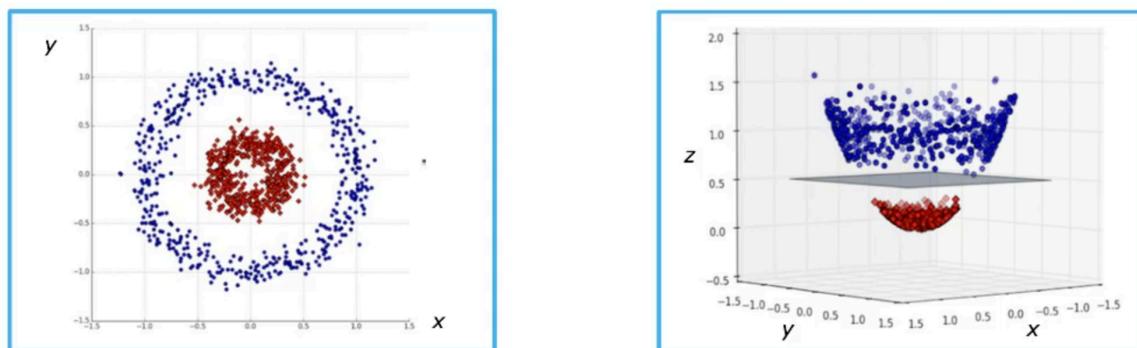


Abbildung 5.5: Kernel Trick

5.3.6 Transform Features

Daten in Zeitfolgen müssen in ihrer Originalreihenfolge bleiben. *Time-Frequency Transformationen* sind lineare Transformationen, welche die Zeitfolge beibehalten. Solche *Spectrogramme* zeigen wie die Frequenz mit der Zeit ändert und können Dinge aufzeigen, die im Rohmaterial verborgen bleiben.

```
import matplotlib.pyplot as plot
from scipy.io import wavfile

signalData = wavfile.read('y.wav')
plot.specgram(signalData)
plot.xlabel('Time')
plot.ylabel('Frequency')
plot.show()
```

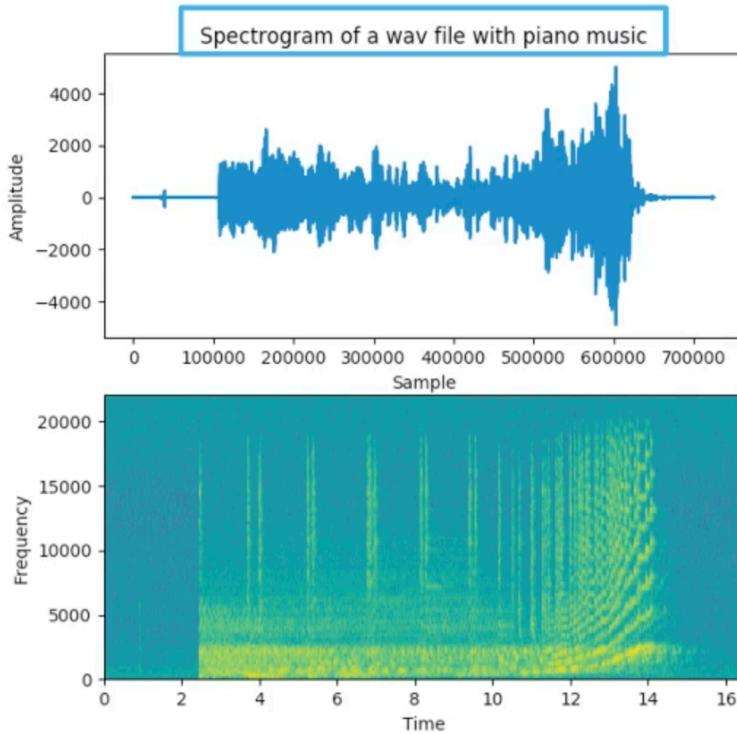


Abbildung 5.6: Spectrogram

5.3.7 Expert Features in Time-Series

Experten sind oft nötig, um richtige Schlüsse zu ziehen, oder auf Features (Formeln oder Betrachtungsweisen) hinzuweisen die nützlich sein können. Damit kann zum Beispiel die *Volatilität* abschätzen zu können und somit Risiken besser abzuschätzen.

5.4 Image Data & Computer Vision Applications

Digitale Bilddaten sind numerisch.

5.4.1 Edges are an Important Feature

In einem Bild sind *Features* besonders wichtig. Wir können aus einer kleiner Skizze schon viel entnehmen. *Edges* sind eines der wichtigsten Image-Features. Weitere sind:

- gerade Linien
- Ecken
- Objekte
- Segmente (gleiches Material, gleiche Farbe)
- usw.

Es gibt viele Algorithmen um Feature zu detektieren.

5.4.1.1 Edge Detection

Ein einfache Möglichkeit um Ecken zu finden ist der *Gradient* eines Bilders zu analysieren. Der Gradient $\nabla(I)$ eines Bildes I ist die erste Ableitung in eine Richtung. Eine weitere Möglichkeit ist das übereinanderlegen des Bildes und verschieben um einen Pixel in eine Richtung. Danach Pixelwerte vom Original subtrahieren. Je nach Richtung erhält man die Ableitung nach x bzw. y . Euklidischer Betrag, damit nicht nur horizontale und vertikale Kanten gefunden werden:

$$I_{\text{edges}} = \sqrt{\nabla_x^2(I) + \nabla_y^2(I)}$$

Daraus erhalten wir ein neues Bild, welches die Kanten hervor hebt.

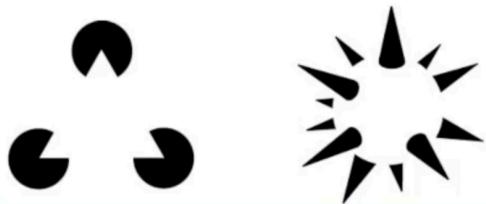
5.4.2 Image Segmentation

Kanten sind oft die Grenzen von Bildsegmenten. Bei der *Image Segmentation* werden Bilder in Segmente aufgeteilt. *K-means clustering* gruppier Pixel in k Segmente (unsupervised).



Abbildung 5.7: K-Means Cluster mit $k = 3$

Gestalt Principles sind menschliche Warnehmungen, wie ähnliche Elemente gruppiert werden und Patterns erkannt werden. Auch können komplexe Bilder vereinfacht werden, wenn wir Objekte wahrnehmen



5.4.3 Image Denoising

Um ein Bild zu glätten, können wir jeden Pixel mit dem Mittelwert der Nachbarn (3x3) dem jetzigen Pixel zuweisen. *Convolution* sind lineare Filter. Mit Image Filtering wird ein neues Bild erzeugt, dessen Pixel eine lineare Kombination der Originalen Pixel sind. Dies ist mittels *convolution* implementiert.

5.4.4 Histogram Equalization

Histogrammausgleich kann helfen, ein Bild zu schärfen.

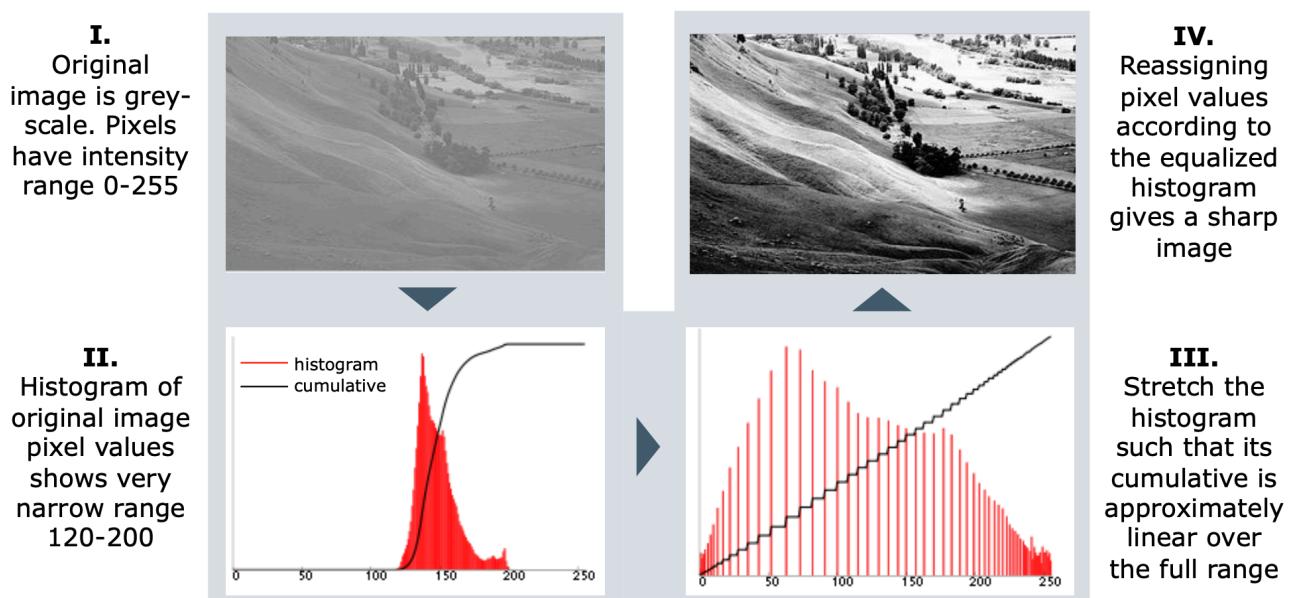


Abbildung 5.8: Histogram Equalization

5.4.5 Scale Invariant Feature Transform - SIFT

Ist ein Feature-Erkennungs Algorithmus um *lokale Features* in einem Bild zu finden. Dazu werden *key-points* aus einem Set von Referenzbildern in eine Datenbank extrahiert. Die Key-Points sind besondere Merkmale in einem Bild bzw. dessen abgebildeten Objekt. Diese werden dann im neuen Bild wieder gesucht, indem die Features verglichen werden. Auch wenn das Bild skaliert wurde, Noise- und Belichtungsresistenz. Der Bildinhalt wird transformiert in lokale Feature-Koordinaten welche invariant gegen Translation, Rotation, Skalierung und weitere Bildparameter.

5.4.6 Feature Learning in Computer Vision

Feature Learning mit Bildern verschiebt sich mehr und mehr in Richtung Deep Neural Networks.

5.5 Text Data & Language Applications

Natural Language Processing (NLP) ist die Schnittmenge aus Computer Science und Linguistik. Diese beschäftigt sich mit der Form der Sprache, Bedeutung und Kontext. Mit NLP versuchen wir *Representations* zu erreichen, welche uns nützliche Sachen mit Text ermöglichen wie Übersetzung und Konversation.

5.5.1 NLP poses really difficult problems

Es gibt etwa 6000 Sprachen und enthält Mehrdeutigkeiten und Redundanzen. Außerdem ist sie Kontextrelevant und viele Wörter sind sehr rar.

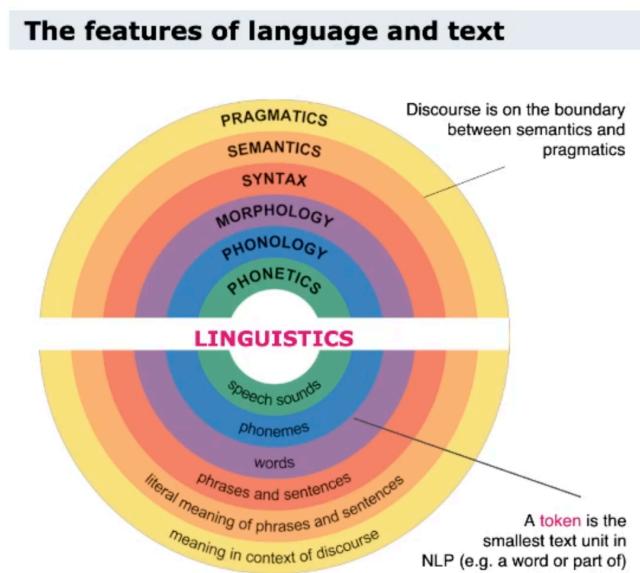


Abbildung 5.9: Features of Language and Text

- Morphology = Stammworte

5.5.2 Linguistic Feature Engineering

Text für ML vorbereiten um *Semantic Language Task* (Sprechen) erreichen zu können.

1. **Tokenization** Erzeugt *Tokens* aus dem Text, einzelne Worte und Punctuation
2. **Stop-Word Removal** Füllwörter oder weniger nützliche Tokens werden entfernt (Sie, du, auf, am, ..)
3. **Stemming&Lemmatization** Tokens werden in ihre Grundform überführt (Wörter → Wort)
4. **Part of Speech Tagging** Labels wie Verb, Nomen, .. werden den Tokens angefügt
5. **Syntax** Text Parsing, Grammatik, Zusammenhang zwischen Tokens

5.5.3 Text Vectorization

Text wird mit statistischen Features untersucht und nummersich dargestellt. Dies hat einige Vorteile.

- Vektorisiert kann der Text algorithmisch verarbeitet werden
- besitzt so ein Distanzmaß (z.B. Euklidische Distanz)
- und kann so Similarität messen

5.5.3.1 One-Hot Encoding

Bei der *One-Hot Encoding*-Methode wird aus dem Korpus (Referenzmenge) eine Liste von einmaligen Worten erzeugt. Jedes Wort des neuen Textes wird dann hinzugefügt und überall dort, wo ein Wort vorhanden ist, mit 1 markiert. Diese Repräsentation ist sehr spärlich, aber trotzdem hoch-dimensional, sie wächst extrem schnell.

Corpus (Reference Text Data)						
<i>There was no hope for him this time: it was the third stroke. Night after night I had passed the house (it was vacation time) and studied the lighted square of window: and night after night I had found it lighted in the same way, faintly and evenly.</i>						
↓						New text
after	0	hope	for	him	I	passed
and	0	0	0	0	0	0
evenly	0	0	0	0	0	0
faintly	0	0	0	0	0	0
for	0	0	1	0	0	0
found	0	0	0	0	0	0
had	0	0	0	0	0	0
him	0	0	0	1	0	0
hope	0	1	0	0	0	0
house	0	0	0	0	0	0
I	1	0	0	0	1	0
in	0	0	0	0	0	0
it	0	0	0	0	0	0
lighted	0	0	0	0	0	0
night	0	0	0	0	0	0
no	0	0	0	0	0	0
of	0	0	0	0	0	0
passed	0	0	0	0	0	1
...	0	0	0	0	0	0
was	0	0	0	0	0	0
way	0	0	0	0	0	0
window	0	0	0	0	0	0

The encoding of this new text based on the vocabulary is a matrix of **binary** values with 31 rows and 6 columns
186 entries

Abbildung 5.10: One-Hot Encoding

5.5.3.2 Bag-of-Words

Ähnlich wie One-Hot Encoding, zusätzlich wird die Frequenz der Häufigkeit repräsentiert. Jeder Text der analysiert werden soll wird als Vektor hinzugefügt. Die Vektorspalten entsprechen den Tokens die im Text vorkommen und die Values entsprechen der Häufigkeit. Diese Methode ist effizienter aber immer noch spärliche Repräsentation. Ausserdem geht die Ordnungen der Worte verloren!

Two texts input →		I hope for him I passed	And it was hope and... no	The encoding of each new text is a vector of integer values with 31 rows Can easily measure distance between texts!
Same Corpus →		after	0	
		and	0	
		evenly	0	
		faintly	0	
		for	1	
		found	0	
		had	0	
		him	1	
		hope	1	
		house	0	
		I	2	
		in	0	
		it	0	
		lighted	0	
		night	1	
		no	0	
		of	0	
		passed	1	
		...	0	
		was	0	
		way	0	
		window	0	

5.5.3.3 Term Frequency-Inverse Document Frequency (TF-IDF)

Verbreitet genutzter Textvektorisierung Algorithmus. Setzt sich aus zwei Berechnungen zusammen. Die *Term Frequency* gibt an wie oft ein Wort im Vergleich aller Worte im gleichen Dokument vorkommt $TF = \frac{w}{d}$. Mit k Dokumenten gibt es Worte die sehr häufig vorkommen zum Beispiel «und». Diese sollten bestraft werden.

Dazu wird der *Inverse Document Frequency* (IDF) verwendet. Die Document Frequency ist das Verhältnis zwischen k Dokumenten welche das Wort w enthalten. Davon das Inverse $IDF = \frac{1}{\frac{k}{w}}$. Um nun den TF-IDF zu erhalten müssen beide Werte multipliziert werden. Um *Schiefe* auszugleichen sollte erst der $\log(IDF)$ gezogen werden.

Document #1 I like melted cheese sandwiches

Document #2 I love melted cheese toasties

Vocabulary	Term Frequency		IDF	TF-IDF	
	Doc #1	Doc #2		Doc #1	Doc #2
cheese	1/5	1/5	2/2	0	0
I	1/5	1/5	2/2	0	0
like	1/5	0	2/1	log(2/1)/5	0
love	0	1/5	2/1	0	log(2/1)/5
melted	1/5	1/5	2/2	0	0
sandwiches	1/5	0	2/1	log(2/1)/5	0
toasties	0	1/5	2/1	0	log(2/1)/5

Abbildung 5.11: TF-IDF

5.5.4 Modern ML Methods

Gehen einen Schritt weiter und lernen die wichtigen Features von Text aus grossen Datensets.

6 Supervised Learning - Regression

Überwachtes Lernen mit Regressionsmodellen. Ziel ist es die Zielvariable (*Attribut*) vorauszusagen anhand *gelabelled* Daten, wobei die Zielvariable stetig ist. Das Voraussagen einer Kategorie wäre ein Klassifizierungsproblem.

6.1 Variants of Regression Models

Berühmte Regressionsmodelle sind * Linear Regression * Polynomial Regression * k-NN Regression * Support Vector Regression * Regression Trees and Random Forests * Neural Networks * XGBoost Regression * ...

Unterscheiden sich durch ihre *Hypothese*, wie sich die Funktion an ihre Daten *fittet* (anpasst). Je dan Regressionsmodell benötigt die Funktion mehr oder weniger Daten um eine Funktion anzupassen (z.B. Neuronale Netze sind sehr Datenhungrig).

6.2 Measuring Regression Quality

Welches Model approximiert die Daten am besten (Güte).

6.2.1 Regression Errors for Linear Hypothesis

Residuen minimieren den Fehlerterm (Distanz zwischen Gerade und effektivem Datenpunkt).

6.2.2 Regression Errors for any Hypothesis

Residuen werden bei allen Modellen verwendet. Das beste Model ist das, welches die kumulierten Residuen minimiert. Es gibt versch. Methoden um Fehler (Residuen) zu minimieren.

6.2.3 How to Measure Regression Errors

Wir nehmen die Funktion f_i als Voraussage-Funktion an für $i = 1, 2, \dots, m$. Die Differenz (Fehler/Residuen) ergibt sich zwischen Voraussage und effektiven Wert aus $y_i - f_i$. Die Summe der Fehler $\frac{1}{m} \sum_{i=1}^m (y_i - f_i)$ ist unsinnig, weil sich positive und negative Fehler auslöschen würden. Geeigneter sind folgende Summen:

6.2.3.1 MAE - Mean Absolute Error

Betragswerte der einzelnen Residuen.

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

6.2.3.2 MAPE - Mean Absolute Percentage Error

Besser als MAE, weil prozentualer Fehler.

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

6.2.3.3 MSE - Mean Squared Error

Vorteil, grössere Abweichungen werden stärker bestraft als kleinere, weil quadriert. Wird oft zur Optimierung verwendet, aber schwieriger zu Interpretieren als MAE und MAPE.

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

6.2.4 Comparison with Mean Approximation

Man kann auch den Mittelwert mit der Approximation vergleichen um den Fehler zu kumulieren. Wie viel besser/schlechter die Approximation mit dem Vergleich zum Mittelwert.

6.2.4.1 R-Squared (R^2) - Coefficient of Determination

R^2 misst, wie gut das Modell die Abweichungen der Varianz erklärt. Der Wert liegt zwischen [0, 1]. Der Wert kann als Prozent interpretiert werden. $R^2 = 0.53$ bedeutet, dass 53 der Datenvarianz durch das Modell erklärt wird. Wir *minimieren* MAE, MAPE und MSE aber *maximieren* R^2 .

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

Es ist möglich, dass der R^2 negativ wird. Genau dann, wenn das Modell schlechter performt als der Mittelwert (horizontale Gerade) und der Bruch > 1 wird.

6.3 Machine Learning Quality Assessment

Grundsätzlich ist das Modell zu wählen, welches den grössten R^2 -Wert ergibt.

6.3.1 Generalization Error

Das minimieren von Trainingsfehler garantiert nicht, dass das Verfahren auch auf neuen (*fresh/unseen*) Daten auch gut performt. Praktisch relevant ist deshalb die Performance auf neuen Daten, welche *generalization Error* genannt wird. Modelle die gut auf Trainingsdaten performen, aber schlecht auf unseen Daten wird *overfitting* genannt (Trainingsdaten auswendig, kann Wissen aber nicht auf neuen Daten anwenden).

6.3.1.1 Perfect Performance on Training Data

Unwidersprüchliche Daten können perfekt approximiert werden (Interpolationstheorem). Für m verschiedene Datenpunkte ergibt ein Polynom mit Grad $m - 1$.

Modell soll Tendenz der Daten und nicht die Trainingsdaten auswendig lernen.

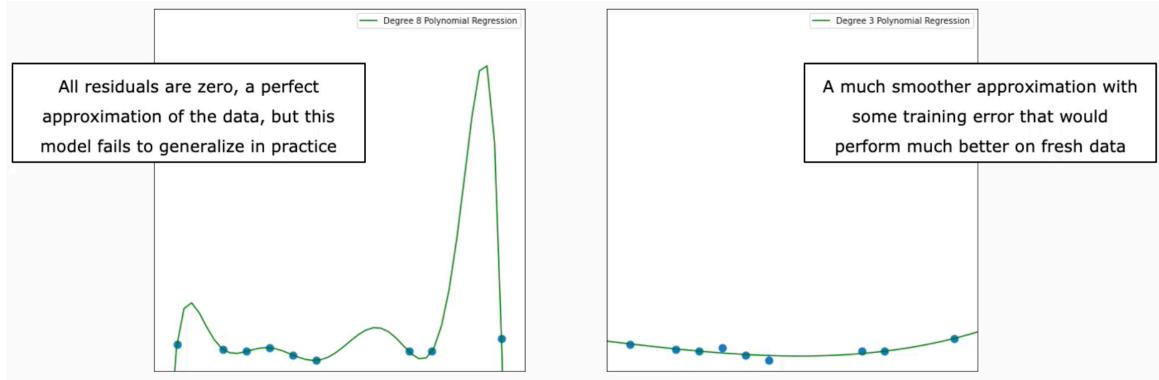


Abbildung 6.1: Vergleich Overfit vs. weiche Approximation

6.3.1.2 Simplistic Machine Learning Workflow

Dieser Workflow funktioniert nur, wenn wir viele Daten haben (wir splitten) und fixe Hyperparameter verwendet werden. Beim Vergleich von Modellen darf dies nicht angewandt werden.

1. Daten mit Zufalls-Seed vermischen (ausser bei Timeseries)
2. Daten in Trainings- und Testdaten aufteilen (80/20)
3. Regressionsmodell auf Trainingsdaten trainieren
4. Evaluation auf dem Testset um Performance schätzen

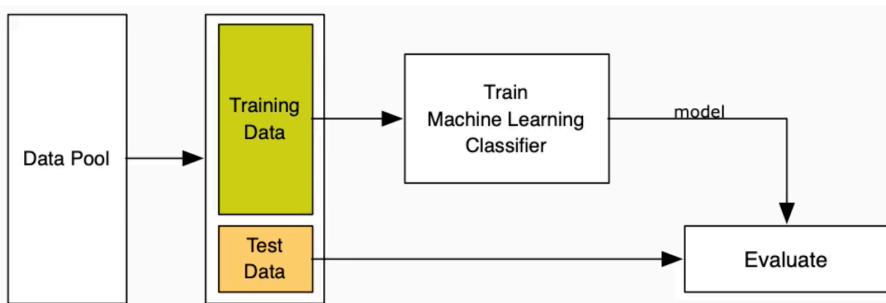


Abbildung 6.2: Einfacher Machine Learning Workflow

6.3.2 Hyperparameters

Hyperparameter bilden die manuelle Konfiguration eines ML-Models und sind je nach Modell sehr verschieden. Als Beispiel Anzahl k Nachbarn bei k-NN oder Grad des Polynoms für Regressions Modelle. Als Engineer müssen wir die besten Hyperparameter aussuchen, welche auf ungesiehenen Daten am besten performt. Ein simpler Train-Test split funktioniert da leider nicht.

6.3.2.1 How to get Fired as Data Scientist

Beim Vergleich von Hyperparameter darf die Aussage, wie Genau das Modell performen wird nur dann tätigen, wenn es *truly unseen* Daten sind (sobald einmal verwendet sind sie nicht mehr unseen).

6.3.3 Evaluation Workflow for Hyperparameter Optimization

Daten sollen in 60/20/20 Teile für Training/Validation/Test aufgeteilt werden. Die Testdaten wegsperren und erst für finalen Performancetest **einmal** verwenden. Arbeiten nur mit Trainings- und Validationsdaten 1. über alle Hyperparameter Kombinationen loopen 1. Modelle mit gewählten Hyperparameter mit *Trainingsdaten* trainieren 1. Performance für Model mit *Validationsdaten* messen 1. Bestes Modell (Performance) auswählen 1. Modell mit *Testdaten* evaluieren, dies ergibt Performanceschätzung für *truly unseen Data*.

Ist man unzufrieden, braucht man neue Daten. Man darf keine Änderung am Modell vornehmen, ohne neue Daten zur Hand zu haben. Dieser Workflow benötigt viele Daten.



Abbildung 6.3: Evaluation Workflow Hyperparameter Optimization

6.3.4 K-Fold Cross-Validation

Wenn wir zu wenig Daten haben um 60/20/20 zu teilen, kann das $K - Fold$ -Prinzip aushelfen. Man teilt die Trainings- und Testdaten in 80/20 auf. Die Testdaten wegsperren. Auf den 80% der Daten wird die *K-Fold Cross Validation* gemäss Abbildung 6.4 ausgeführt.

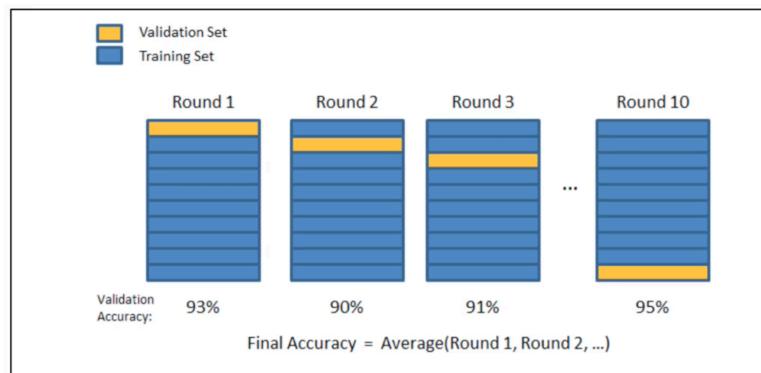


Abbildung 6.4: 10-Fold Cross-Validation

Die 80% Daten werden in k Teil unterteilt. Das Training wird jeweils auf $k - 1$ Teile ausgeführt und validiert. Nach jeder Runde wird ein anderes Validierungs- und Trainingsset verwendet. Die Gesamtgenauigkeit ergibt sich aus dem Durchschnitt der Genauigkeit aller Runden. Wird mittlerweile auch dann verwendet, wenn sehr sehr viele Daten vorhanden wären. Grund ist, dass wir aus diesem Verfahren stabile Aussage erhalten oder wenn Training sehr, sehr rechenintensiv.

7 Gradient Descent

Wie findet man die relevanten Punkte in einer Funktion. Es ist eine Form von *supervised learning*. Es wird von gelabelten Daten gelernt. Dazu wird eine Kostenfunktion definiert und die Parameter so optimiert, dass man zum lokalen Minimum strebt.

7.1 Fining local minima

Wenn wir ein Kostenfunktion minimieren möchten, suchen wir nach dem lokalen bzw. globalen Minimum. Um einen Parameter (z.B Geschwindigkeit) anpassen möchten um einen zweiten (z.B. Gefahr) zu minimieren wird *Parameter Tuning* genannt. Der Parameter θ können wir *tunen* weil er in unserer *direkten* Kontrolle ist. $J(\theta)$ ist die Kostenfunktion und wird verändert, wenn wir θ verändern. J wird als indirekt kontrolliert.

7.1.1 Multidimensional Functions

Auch mit Multidimensionalen Parameterraum gelten die Regeln der Mini-/Maxima. Wobei das Minimum einer Kostenfunktion $J(\theta)$ ein Vektor ist $\theta = [\theta_0, \theta_1]^T$. Um diese zu finden nutzen wir Partielle Ableitungen.

7.1.2 Contour Lines

Kontourlinien zeigen den Wert von θ für welche die Kostenfunktion $J(\theta) = c$, wobei c einer Konstante entspricht.

7.2 (Batch) Gradient Descent

Der Gradient ist ein Vektor, welcher die partiellen Ableitungen von $J(\theta)$ enthält. Mehr dazu in IMATH. Batch heisst es, weil es auf einem grossen Datenbestand angewendet wird.

$\nabla J(\theta)$ wird in Englisch “grad jay theta” ausgesprochen.

Der Gradient von $J(\theta)$

7.2.1 The Idea Behind Gradient Descent

Starte irgendwo und bewege dich *entgegen* des Gradienten. In Pseudocode:

1. Caclulate the Gradient of starting point $\nabla J(\theta_0)$
2. move small step α in the exact opposite direction $-\alpha \nabla J(\theta_0)$
3. go back to first step, iterate until arriving at the local minimum

Die Step-Grösse α bestimmt die Geschwindigkeit der *Konvergenz*. Die Daten müssen vorher **normalisiert** werden.

7.2.2 The Math Behind

Der Algorithmus entspricht folgendem mit definiertem α :

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k)$$

Iterieren bis zum Stopkriterium

7.2.3 Batch Gradient Descent

Wir haben eine Kostenfunktion $J(\theta)$ welche die Summe aller Fehlerterme (Residuen) enthält. Bei jeder Iteration k werden alle Datenpunkt (X, Y) verwendet um ∇J zu berechnen. Dies wird *Batch* Gradient Descent genannt. Es ist eine *iterative* Methode welche auf dem Gradienten basiert. Dabei werden vorhandene Daten genutzt um den Gradienten abzuschätzen. Zum Beispiel bei linearer Regression versuchen wir eine Gerade so anzupassen, dass Sie ideal zwischen die Datenpunkte passt.

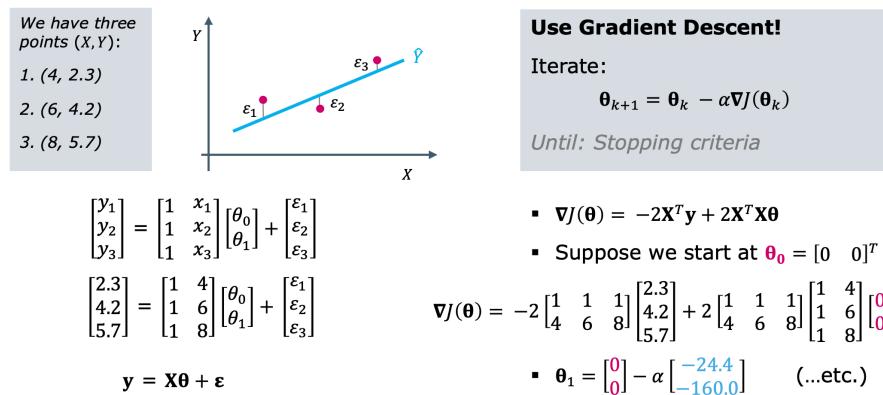


Abbildung 7.1: Batch Gradient Descent Example

7.3 Stochastic Gradient Descent

Weil im Batch gradient descent in jeder Iteration *alle* Datenpunkte verwendet werden, können wir zwar garantieren dass wir auf ein Minimum konvergieren, jedoch wenn wir eine grosse Anzahl N haben wir das Training sehr langsam. Außerdem kann sich das Training in einem lokalen anstatt globalen Minimum reinarbeiten.

Die Alternative ist *Stochastic Gradient Decent*. Diese nutzt pro Iteration nur **einen** Datenpunkt (ein Sample vom Datenset) um den Gradienten abzuschätzen. Der Punkt wird dabei zufällig ausgewählt.

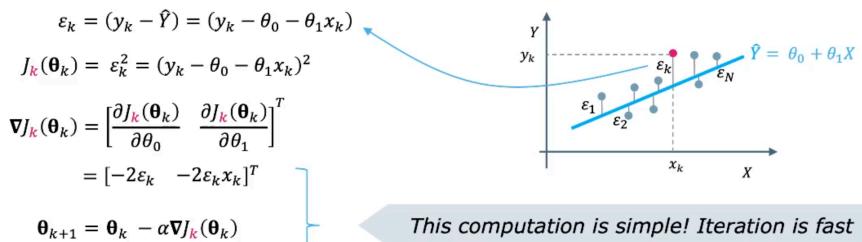


Abbildung 7.2: Stochastic Gradient Descent

7.3.1 Convergence of Gradient Descent

In der Abbildung 7.3 erkennt man klar die Unterschiede der Konvergenz der beiden Verfahren.

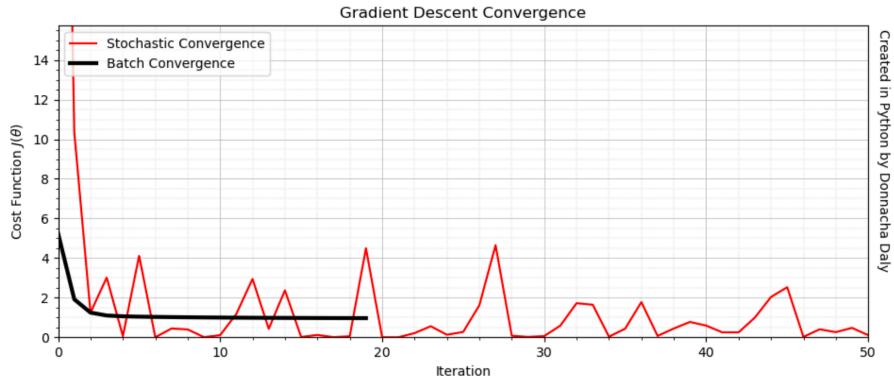


Abbildung 7.3: Convergence of Gradient Descent

7.3.2 Batch vs. Stochastic Gradient Descent

	Batch	Stochastic
pros	<ul style="list-style-type: none"> * Globale Minimum wird für konvexe Kostenfunktionen gefunden * konvergiert zu einem lokalen Minimum für nicht konvexe Funktionen 	<ul style="list-style-type: none"> * sehr schnell * erlaubt online-learning
cons	<ul style="list-style-type: none"> * langsam für grosse Datensets * Kein Online-Learning (z.B. für neue Daten) 	<ul style="list-style-type: none"> * hohe Varianzen zwischen Punkten * Springt umher, auch wenn nahe am Minimum

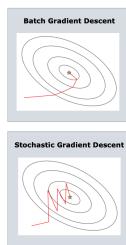


Abbildung 7.4: Batch vs. Stochastic Gradient Descent

7.3.3 The Effect of Learning Rate α

Die Konvergenzrate wird mit der *Schrittgrösse* α definiert. Normalerweise liegt $0.01 \leq \alpha \leq 0.1$ und je grösser der Wert umso grösser ist der Schritt in jeder Iteration. Die Gefahr, dass Minimum zu überspringen steigt damit. Wenn sie aber kleiner sind, umso mehr Iterationen sind nötig, bis das Minimum gefunden wird. Ideal wäre mit grossem α zu starten und kleiner zu werden.

7.3.4 Stopping Criteria

Die Stopkriterien sind sehr ähnlich. Die beste Option ist häufig Datenabhängig.

Batch Gradient Descent	Stochastic Gradient Descent
Iterate: $\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k)$	Iterate: $\theta_{k+1} = \theta_k - \alpha \nabla J_k(\theta_k)$
<i>Until: Stopping criteria</i>	<i>Until: Stopping criteria</i>

Abbildung 7.5: Stopcriterias

Stoppen wenn

- ein Maximum an Iterationen erreicht wurde
- die Kostenfunktion einen gewissen Threshold unterschritten hat
- die Reduktion der Kostenfunktion zwischen Iterationen sehr klein ist
- der Gradient der Kostenfunktion klein ist
- die Schritte pro Iteration kleiner als einen gewissen Threshold sind

8 Applications of Gradient Descent in Machine Learning

Gradient Descent wird überall in ML verwendet. Sei es beim finden von optimalen Parameter bei linearer oder logische Regression oder beim finden der *Weights* im Training von neuronalen Netzwerken oder beim verfeinern von Billionen von Parametern in Deep Learning.

8.1 Linear Regression by Gradient Descent

In 1D-linearer Regression wir versuchen eine Gerade an die Datenpunkte anzupassen. Mit zwei Parameter, versuchen wir eine Ebene (x,y,z in 3D) an die Punkte anzupassen. Sobald mehr als drei Dimensionen im Spiel sind, fitten wir eine *Hyperebene* oder *hyperplane*. In all diesen Fällen ist unsere Kostenfunktion ein Mass wie arm das Fitting ist. Normalerweise wird es mit der Summe der Quadrate der Residuen berechnet. Der Gradient der Kostenfunktion ist dann eine lineare Funktion der Parameter.

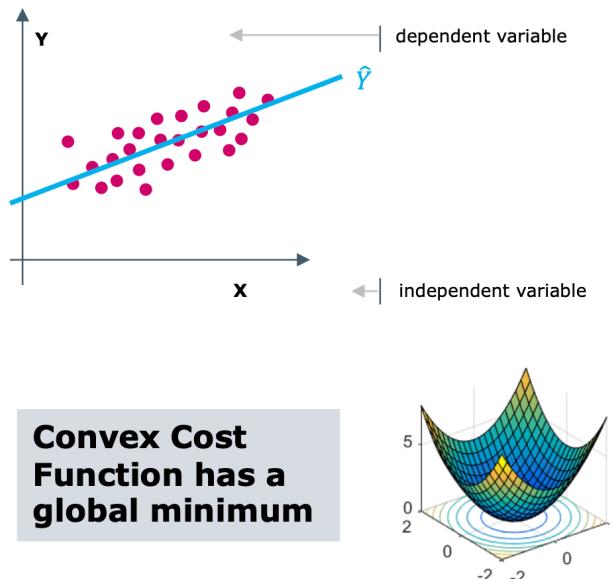
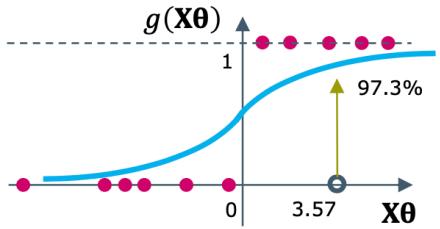


Abbildung 8.1: Lineare Regression by Gradient Descent

8.2 Logistic Regression by Gradient Descent

Hier wird versucht *sigmoide*-Daten so anzupassen, dass sie für *binäre* Klassifikation genutzt werden kann. Die Parameter werden durch Gradient Descent optimiert.



The **logistic function** is a symmetric, sigmoid (S)

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

Abbildung 8.2: Logistic Regression by Gradient Descent

8.3 Neural Net Training by Gradient Descent

Dabei werden die *Gewichte* (*weights*) so optimiert, dass Klassifizierungsfehler reduziert werden können. Es werden *labelled data* genutzt um die Fehler zu messen und eine Kostenfunktion zu entwickeln. Die Kostenfunktion ist *non-convex*. Mit *back propagation* kann die Geschwindigkeit optimiert werden.

8.4 Deep Learning by Gradient Descent

Es werden viele Daten benötigt. Es werden immer wieder die Gradientengleichung verwendet. Es wird eine Kostenfunktion erstellt, welche die Gewichte optimieren (lokales Minimum). Auch hier kann Back-Propagation helfen, allerdings tritt das Problem vom Verschwinden des Gradiententermes auf. Die Konvergenz kann dadurch gleichbleiben. Um dies zu verhindern sind spezielle Tricks nötig.

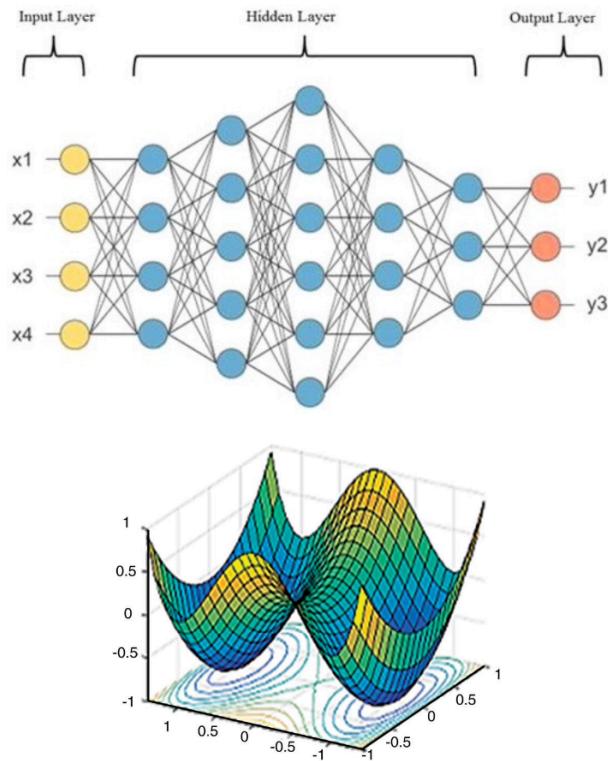


Abbildung 8.3: Neural Net Training by Gradient Descent

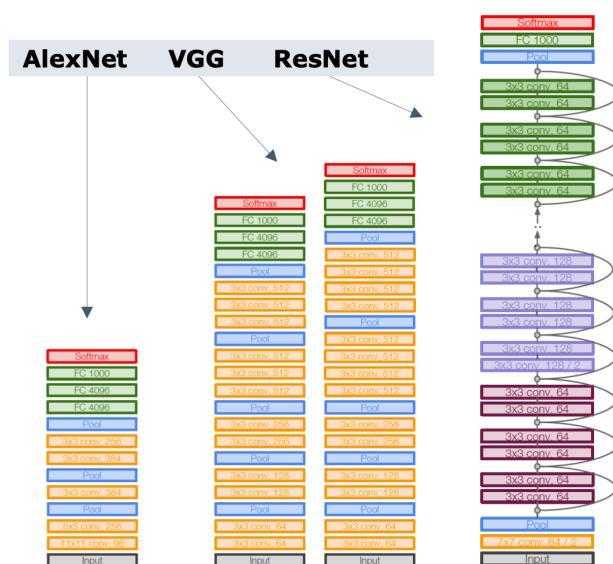


Abbildung 8.4: Deep Learning by Gradient Descent

9 Linear Regression

Gibt es eine Beziehung zwischen zwei Variablen und falls ist sie stetig oder nur in einer gewissen Periode?

9.1 Introduction

Regression wird genutzt um Relationen in Daten zu finden. Das Ziel ist es die Relation zwischen Features zu finden, *wenn es existiert*. Wenn wir voraussetzen, dass die Relation approximativ linear, können wir die **lineare Regression** wie folgt ausdrücken:

$$\hat{Y} = \theta_0 + \theta_1 X$$

9.1.1 Regression can be used for Prediction

Wenn wir eine Gerade finden können, können *neue* Datenpunkt abgeschätzt werden. Wir nutzen lineare Regression um die Voraussage für neue Datenpunkte zu treffen.

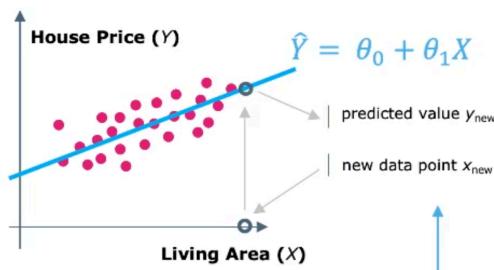


Abbildung 9.1: Linear Regression

Aber wie finden wir die Parameter θ_0 und θ_1 ? **Supervised Learning**

9.1.2 Terminology

9.1.3 Selecting the Regression Parameters

In einem linearen Modell ist unsere Hypothese, dass die Relation zwischen den beiden Variablen eine gerade Linie ist:

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

Um nun die beste Gerade zu finden, müssen wir θ_0 und θ_1 so wählen, dass das Total der *Fitting Error* minimal ist.

- X is the **independent** variable (regressor)
- Y is the **dependent** variable
- The chart Y vs. X is a **scatter plot**
- The j -th data point is (x_j, y_j)
- **Hypotheses:** $h_{\theta}(x_j) = \theta_0 + \theta_1 x_j$
- θ_0 and θ_1 are regression **parameters**
- θ_0 is the Y -intercept of the fitted line
- θ_1 is the slope of the fitted
- $\hat{Y} = \theta_0 + \theta_1 X$ is the **line of regression**
- ϵ_j is the j -th **residual** (error term)
- $y_j = \theta_0 + \theta_1 x_j + \epsilon_j$

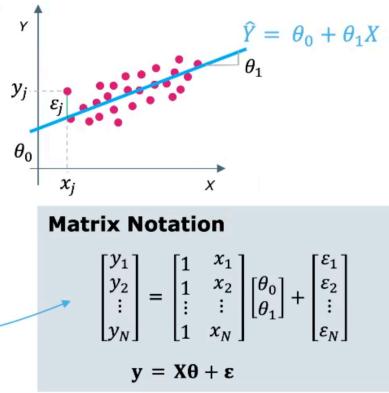


Abbildung 9.2: Terminology in linear Regression

9.2 Ordinary Least Squares (OLS)

Wie können wir den Regressionsfehler messen?

- MAE - Mean Absolute Error
- MAPE - Mean Absolute Percentage Error
- MSE - Mean Squared Error

9.2.1 Defining a Cost Function

Der *MSE - Mean Squared Error* ist der Favorit als Regressionsmaß. Es ist eine konvexe Funktion der Parameter und stetig **differenzierbar** beider Parameter (θ_0, θ_1) . Wir wählen also θ_0, θ_1 um den MSE zu minimieren. Die Funktion nennen wir *Cost Function* $J(\theta_0, \theta_1)$.

$$J(\theta_0, \theta_1) = \frac{1}{N} \sum_{j=1}^N \epsilon_j^2 = \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2 = \frac{1}{N} \sum_{j=1}^N (y_j - \theta_0 + \theta_1 x_j)^2$$

9.2.1.1 Writing the Cost Function in Matrix Form

<p>#1: Recall matrix notation for residuals</p> $y_j = \theta_0 + \theta_1 x_j + \epsilon_j$ $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$ $\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}$ $\Rightarrow \boldsymbol{\epsilon} = \mathbf{y} - \mathbf{X}\boldsymbol{\theta}$	<p>#2: Use residuals in cost function $J(\boldsymbol{\theta})$</p> $J(\boldsymbol{\theta}) = \sum_j \epsilon_j^2 = [\epsilon_1 \quad \epsilon_2 \quad \dots \quad \epsilon_N] \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$ $\Rightarrow J(\boldsymbol{\theta}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$ <p>#3: Matrix notation for cost function</p> $J(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$ $\Rightarrow J(\boldsymbol{\theta}) = \mathbf{y}^T \mathbf{y} - 2\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{y} + \boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X}\boldsymbol{\theta}$
--	--

Abbildung 9.3: The Cost Function in Matrix Form

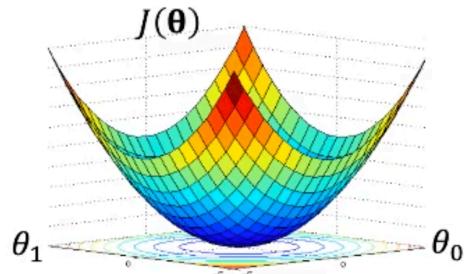


Abbildung 9.4: Konvexe Funktion

9.2.1.2 Minimizing the Cost Function

Die Kostenfunktion ist eine *konvexe* (Schüsselform) mit einem einzigen Globalen Minimum.

Das Minimum finden wir mit *Calculus*.

9.2.2 Ordinary Least Squares

Um das Minimum der Kostenfunktion zu finden muss der Gradient 0 sein.

$\mathbf{X}^T \mathbf{X}$ is a symmetric matrix

$$\frac{\partial(\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 2 \mathbf{X}^T \mathbf{X} \boldsymbol{\theta}$$

$$\frac{\partial(\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{y})}{\partial \boldsymbol{\theta}} = \mathbf{X}^T \mathbf{y}$$

↗ $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -2 \mathbf{X}^T \mathbf{y} + 2 \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} = 0$

$$\Rightarrow \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} = \mathbf{X}^T \mathbf{y}$$

$\Rightarrow \boldsymbol{\theta}_{\text{opt}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

OLS!

Abbildung 9.5: Partial Derivative OLS

9.2.2.1 Example in Python

9.2.3 Linear Regression by Gradient Descent

Eine effizientere Möglichkeit (besonders bei grossen Datensets) das Minimum zu finden ist die Gradientenmethode. Die Lösung wird so iterativ gefunden. Mehr dazu in [Gradient Descent](#).

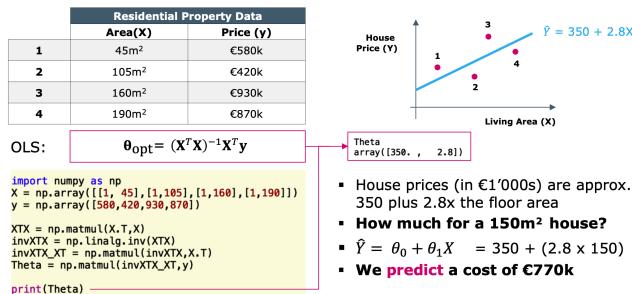


Abbildung 9.6: OLS Example

9.3 Regression Performance (R^2)

Lineare Regression können wir einsetzen, wenn die Datenpunkte linear Zusammenhängen, einen Zusammenhang besteht. Dies müssen wir aber prüfen können.

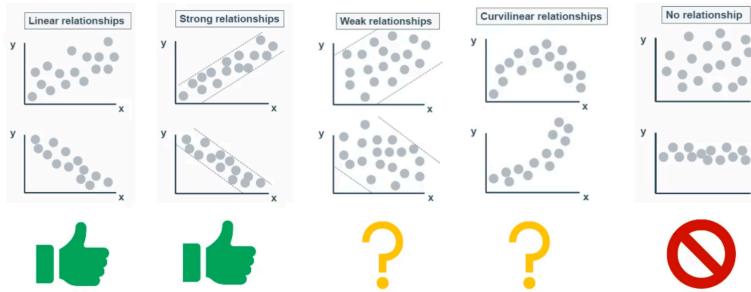


Abbildung 9.7: When fit Linear Regression

Die Variabilität von Y der Gerade ist der [MSE - Mean Square Error]. Die Variabilität von Y vom Mittelwert \hat{Y} ist die *Varianz*. Das Verhältnis dieser beiden Zahlen ist eine gute Performance Metrik.

9.3.1 R^2 - The Coefficient of Determination

R^2 ist ein Mass wie stark die Variabilität von Y durch X erklärt wird. Der Wert erklärt die Variabilität der abhängigen Variable. $1 - R^2$ ist die Variabilität die unerklärt bleibt. Siehe [R-Squared \(\$R^2\$ \) - Coefficient of Determination](#)

Der Wert von R^2 ist nicht 100% verlass. Die Daten sollte *immer* geplottet/visualisiert werden. Die Verteilung kann R^2 beeinflussen!

9.3.2 Visualize your Residuals

Die Residuen verbleiben, wenn wir die beste passende Gerade von den Daten abziehen. Für einen guten *fit* die Residuen sollten wie *zero-mean*, stationär, *white noise* aussehen.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \epsilon \implies \epsilon = \mathbf{y} - \mathbf{X}\boldsymbol{\theta}$$

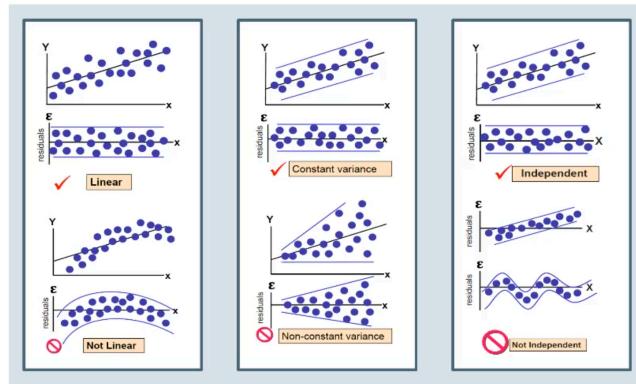


Abbildung 9.8: Visualize the Residuals

9.3.3 Correlation and R^2

Korrelation ist ein generelles Mass wie zwei Variablen sich in Relation zur anderen verhalten. Die *Pearson correlation* $\rho_{X,Y}$ ist ein spezieller Fall, welcher nur die lineare Relation zwischen X und Y betrachtet. Es ist ein normalisiertes Mass der Covarianz und liegt zwischen $-1 \leq \rho_{X,Y} \leq 1$.

Weil wir die *lineare Abhangigkeit* messen erwarten wir einen Zusammenhang so dass gilt:

$$r \rho_{X,Y}^2 = R^2$$

Durch Korrelation gilt nicht automatisch ein kausaler Zusammenhang.

9.4 Multiple Linear Regression

Wenn wir mehr als ein Feature in einem N -Dimensionalen Raum betrachten.

9.4.1 Matrix with M-Regressors

$$y(j) = \theta_0 + \theta_1 x_1(j) + \theta_2 x_2(j) + \dots + \theta_M x_M(j) + \varepsilon(j)$$

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} 1 & x_1(1) & x_2(1) & \dots & x_M(1) \\ 1 & x_1(2) & x_2(2) & \dots & x_M(2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1(N) & x_2(N) & \dots & x_M(N) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix} + \begin{bmatrix} \varepsilon(1) \\ \varepsilon(2) \\ \vdots \\ \varepsilon(N) \end{bmatrix}$$

y Vector of samples of Independent Variable	X Matrix of samples of Dependent Variables (Regressors)	θ Regression Parameter Vector	ε Vector of Regression residuals
---	---	---	--

Ordinary Least Squares

$$\begin{aligned} y &= X\theta + \varepsilon \\ \Rightarrow \varepsilon &= y - X\theta \\ J(\theta) &= \varepsilon^T \varepsilon \\ \Rightarrow J(\theta) &= y^T y - 2\theta^T X^T y + \theta^T X^T X \theta \\ \frac{\partial J(\theta)}{\partial \theta} &= -2X^T y + 2X^T X \theta = 0 \\ \Rightarrow \theta_{\text{opt}} &= (X^T X)^{-1} X^T y \\ \Rightarrow \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix}_{\text{opt}} &= (X^T X)^{-1} X^T y \end{aligned}$$

Abbildung 9.9: OLS - M Regressors

9.4.2 Nonlinear Regression

9.4.3 Regularization

Wenn wir mehr extra Feature haben, um unsere Trainingsdaten fitten, kann es sein dass das Model overfittet. Overfitting fuhrt zu Generalisierungsfehler wenn neue Daten kommen. Wir mussen uns auf die Feature fokusieren welche die grosste erklarende Macht haben. Dies ist Regularisierung.

- So far our hypothesis is a linear relationship between variables X and Y
- What if the relationship is nonlinear?
- We can use **polynomial regression**
- For example suppose our hypothesis is
- $h(\theta, X) = \theta_0 + \theta_1 X + \theta_2 X^2 + \theta_3 X^3$
- **Trick:** let $X_1 = X$, $X_2 = X^2$, $X_3 = X^3$
- Now we have a linear hypothesis!
- $h(\theta, X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3$
- **Multiple linear regression**

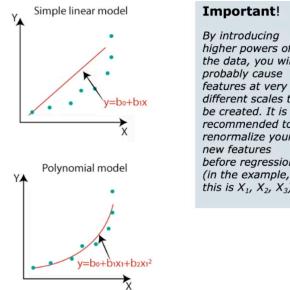


Abbildung 9.10: Nonlinear Regression

9.4.3.1 Ridge and LASSO

Beide Methoden sind sehr ähnliche Methoden der Regularisierung (overfitting verhindern). Beide fügen der **Ordinary Least Squares (OLS)**, also der Kostenfunktion, einen extra Term hinzu, um die unwichtigen Feature weniger zu gewichten.

- Ridge: nutzt die Summe der Quadrate der Parameter, irrelevante Features werden sehr klein
- LASSE: nutzt die Summe der Beträge der Parameter, irrelevante Features werden 0

9.4.3.2 Regularization Parameter λ

Grosse Parameter werden bestraft. Hat Effekt, dass nur die wichtigen Feature die Regression steuern. Weniger relevante Features werden unterdrückt. Kontrolle über den *Regularisierungsparameter λ*

10 Code CheatSheet

Contains several code snippets and useful library functions.

10.1 Python

Code	Parameters	Description
<code>~bool</code>		flip boolean result

10.2 Pandas

Code	Parameters	Description
<code>pd.qcut(data['column-name'], q=4, labels=False)</code>	<code>q</code> =number of Quantiles, <code>labels</code> =binlabelsarray or False	bin numerical data in q buckets
<code>pd.get_dummies(data, drop_first=True)</code>	<code>drop_first</code> = removes one of the columns to get $k - 1$	convert categorical features to numerical ones. decide for each feature to use *

one hot encoding* or *label encoding* `df.duplicated(keep='first')` or `df.duplicated(keep='last')` remains in set|list all duplicate entries. `keep=False` shows all duplicated entries. if param omitted one entry remains in the set
`df.drop_duplicates(inplace=True)` removes duplicates directly, otherwise a copy gets returned (default false)| drop all duplicates '`df.columnname.nunique()`' | returns the number of unique values in total

10.3 Numpy

Code	Parameters	Description
<code>np.asarray(x)</code>		returns a 1d array
<code>np.asarray(x).shape</code> returns dimension of \$n\$-d array.	<code>a</code> =array, <code>shift</code> =number of places elements are shifted	shift elements in direction of axis
(160,3) means 160 data points in 3 dimensions		
<code>np.roll(a, shift, axis=None)</code>		

10.4 Sklearn

Code	Parameters	Description
<code>sklearn.neighbors.KNeighborsRegressor()</code> <code>train_test_split(X,y, test_size=0.2,random_state=42)</code>	<code>weights=Gewichtung der k-nahesten Datenpunkte, metric=DistanceMetric</code> <code>X & y must have the same size,*</code>	K-NN Regression

`test_size*` represent the proportion of the dataset, `random_state` controls the shuffling applied - with an int output is reproducible | Split arrays or matrices into random train and test subsets. Returns for each array **two** lists with a train and test set `DummyRegressor()` | `strategy` with options {mean, median, quantile, constant} | instantiate a `DummyRegressor` to get an idea how good our model performs.