

Advanced Machine Learning

Zusammenfassung

Stephan Stofer

1. Mai 2021

Inhaltsverzeichnis

1 Data Classification	11
1.1 Data Quality Assessment	11
1.1.1 Data Cleaning	11
1.1.2 Approaches to DQA	11
1.1.3 Statistische Kennzahlen	12
1.2 Replacement Strategies für NULL Values	13
1.2.1 Feature Engineering	13
1.2.2 Vector Space Model	13
1.3 Pandas Profiling	14
1.4 Fazit	14
2 History & Development	15
2.1 ML in the Context of AI	15
2.2 Development of AI	15
2.3 Der Ursprung von AI	15
2.4 The Connectionists: ANN's, ML and NN	16
2.5 The Human Brain	16
2.5.1 Networks that Learn	16
2.6 Deep Learning vs. Machine Learning	16
2.7 Three Pillars of Machine Learning	16
2.8 Various Options for Hardware Acceleration	16
2.9 Performance Measure	16
2.10 Remaining Challenges	17
3 Machine Learning Fundamentals	18
3.1 Vector Space Model	18
3.2 Data Records	18
3.3 Numerical Encoding of Text	19
3.4 Distance and Similarity	19
3.5 Euclidean Distance or L^2 -Norm	19
3.6 Semantik of Similarity	19
3.7 Cosine Similarity Intuition	19
3.8 Cosine Similarity	20
3.9 Euklid vs. Kosinus	20
3.10 Manhattan Distance	20
3.11 Levenshtein or Edit Distance for Strings	20
3.12 Jaccard Similarity for Sets	21
3.13 Haversine Distance for GEO Data	21
3.14 From Points to Distributions	21
3.15 Mahalanobis Distance between Point and Distribution	21
3.16 Distance may be sensitive to Scale of Axes	21
3.17 Min-Max Normalization	21
3.18 Z-Score Normalisierung	22
3.19 Normalization Parameters	22

3.20	K-Nearest Neighbors Classification (k-NN)	22
3.21	K-Nearest Neighbors Regression	22
3.22	Hyperparameter	23
3.23	Facts on K-Nearest Neighbors	23
4	Data Preparation for Recommender Systems	24
4.1	Convert Strings to lower-case format	24
4.2	Tokenizing	24
4.3	Lemmatization	24
4.4	Stemming	25
4.5	Data	25
5	Feature Engineering	26
5.1	Data	26
5.1.1	Tabular Data	26
5.1.2	Time Series Data	27
5.1.3	Image Data	27
5.1.4	Text Data	27
5.1.5	Feature Engineering vs. Feature Learning	27
5.2	Feature Engineering for Tabular and Time-Series Data	27
5.2.1	Data Quality Assessment & Data Cleaning	28
5.2.2	Data Imputation	28
5.3	Engineering New Features	28
5.3.1	Grouping	29
5.3.2	Binning	29
5.3.3	De-skewed Data	29
5.3.4	Kernel Trick	29
5.3.5	Expert Knowledge	29
5.3.6	Transform Features	31
5.3.7	Expert Features in Time-Series	31
5.4	Image Data & Computer Vision Applications	31
5.4.1	Edges are an Important Feature	32
5.4.2	Image Segmentation	32
5.4.3	Image Denoising	33
5.4.4	Histogram Equalization	33
5.4.5	Scale Invariant Feature Transform - SIFT	33
5.4.6	Feature Learning in Computer Vision	34
5.5	Text Data & Language Applications	34
5.5.1	NLP poses really difficult problems	34
5.5.2	Linguistic Feature Engineering	34
5.5.3	Text Vectorization	35
5.5.4	Modern ML Methods	36
6	Supervised Learning - Regression	37
6.1	Variants of Regression Models	37
6.2	Measuring Regression Quality	37
6.2.1	Regression Errors for Linear Hypothesis	37
6.2.2	Regression Errors for any Hypothesis	37
6.2.3	How to Measure Regression Errors	37
6.2.4	Comparison with Mean Approximation	38
6.3	Machine Learning Quality Assessment	38
6.3.1	Generalization Error	38

6.3.2	Hyperparamters	40
6.3.3	Evaluation Workflow for Hyperparameter Optimization	40
6.3.4	K-Fold Cross-Validation	40
7	Gradient Descent	42
7.1	Fining local minima	42
7.1.1	Multidimensional Functions	42
7.1.2	Contour Lines	42
7.2	(Batch) Gradient Descent	42
7.2.1	The Idea Behind Gradient Descent	42
7.2.2	The Math Behind	43
7.2.3	Batch Gradient Descent	43
7.3	Stochastic Gradient Descent	43
7.3.1	Convergence of Gradient Descent	44
7.3.2	Batch vs. Stochastic Gradient Descent	44
7.3.3	The Effect of Learning Rate α	44
7.3.4	Stopping Criteria	45
8	Applications of Gradient Descent in Machine Learning	46
8.1	Linear Regression by Gradient Descent	46
8.2	Logistic Regression by Gradient Descent	46
8.3	Neural Net Training by Gradient Descent	47
8.4	Deep Learning by Gradient Descent	47
8.5	Introduction	47
8.5.1	Regression can be used for Prediction	47
8.5.2	Terminology	49
8.5.3	Selecting the Regression Parameters	49
8.6	Ordinary Least Squares (OLS)	49
8.6.1	Defining a Cost Function	49
8.6.2	Ordinary Least Squares	50
8.6.3	Linear Regression by Gradient Descent	50
8.7	Regression Performance (R^2)	50
8.7.1	R^2 - The Coefficient of Determination	52
8.7.2	Visualize your Residuals	52
8.7.3	Correlation and R^2	52
8.8	Multiple Linear Regression	52
8.8.1	Matrix with M-Regressors	53
8.8.2	Nonlinear Regression	53
8.8.3	Regularization	53
9	Classification	54
9.1	Introduction	54
9.1.1	Classification Requires a Decision Boundary	54
9.1.2	Simple Classification: k -Nearest Neighbours	54
9.1.3	Classification vs. Regression	55
9.2	Logistic Regression	55
9.2.1	Binary Classification using 1 Feature	55
9.2.2	Binary Classification using 2 Feature	55
9.2.3	Logistic Function	56
9.2.4	Logistic Regression	56
9.2.5	Logistic as a Probability Measure	56
9.2.6	Logistic: Probability of Belonging to a Class	56

9.2.7	Cost Function for Logistic Regression	57
9.2.8	Cross Entropy	57
9.2.9	Logistic Regression by Gradient Descent	57
9.2.10	Non-Linear Decision Boundaries	57
9.2.11	One Versus the Rest	58
9.3	Performance Analysis	58
9.3.1	Confusion Matrix	59
9.4	Performance Optimization	60
9.4.1	Underfitting & Overfitting in Classification	60
9.4.2	How to Diehl with it	61
10	Support Vector Machine	62
10.1	Motivation	62
10.1.1	Uncertainty in Data	62
10.1.2	Large Margin Classifier	62
10.1.3	Support Vectors	63
10.2	The Scalar Product	63
10.2.1	Scalar Product and Vector Length	63
10.2.2	Vector Triangles	63
10.3	Hyperplanes	64
10.3.1	Hessian Normal Form	64
10.3.2	Shifting the Coordinate System	64
10.3.3	How to interpret signed Distances	66
10.4	Large Margin Classifier	66
10.4.1	Controlling the Margin	68
10.4.2	Elegant Problem Formulation	68
10.4.3	How Big is the Margin	68
10.4.4	Primal Optimization Problem	68
10.5	Soft Margin Classifier	69
10.5.1	Outlier Sensitivity	69
10.5.2	Soft Margin Classifier	69
10.5.3	Slack Variables	69
10.5.4	Regularization Parameter	70
10.6	Kernel	70
10.6.1	Not Linearly Separable	70
10.6.2	What is a Kernel	71
10.7	SVM with Kernel	72
10.7.1	Lagrange Methode	72
10.7.2	Lagrange Transformation	72
10.7.3	Kernel Hard-Margin SVM	74
10.7.4	API Check	74
11	Tree Models	75
11.1	Decision Trees	75
11.1.1	Which Features and in which order?	75
11.1.2	Adding Decisions	75
11.1.3	ID3 - Tree Construction Rules	75
11.1.4	Splitting Criterion	76
11.2	Gini Impurity	76
11.2.1	Gini Impurity gets efficient	77
11.2.2	Gini Impurity of continuous Variables	77
11.2.3	Splitting Feature Selection and Stop Criterion	77

11.3 Regression Trees	78
11.4 Discussion	78
11.4.1 Advantages of Decision and Regression Trees	78
11.4.2 Scikit Learn API	78
11.5 Random Forests	78
11.5.1 Building a Random Forest	79
11.5.2 Scikit Learn API	79
12 Model Diagnostics	80
12.1 Model Training	80
12.1.1 Training Curves	80
12.1.2 Mini-Batch Gradient Descent	80
12.1.3 Step-Size Tuning	81
12.2 Model Validation: Learning Curve Diagnostics	81
12.2.1 Learning Curves	81
12.2.2 Diagnosing Under-Fitting (High Bias)	82
12.2.3 Diagnosing Over-Fitting (High Variance)	82
12.3 Model Tuning: Regularization and Other Tricks	82
12.4 Ethical Considerations	83
12.4.1 Ethical Questions	84
12.4.2 Machine Learning SWOT	84
12.4.3 Bias in the Data	84
12.4.4 Ethics Guide for Business	84
12.4.5 ML Workflow	85
13 Neural Networks	86
13.1 The Perceptron - An Artificial Neuron	86
13.1.1 How do we learn?	86
13.1.2 The Perceptron	86
13.2 Feed Forward Neural Networks	88
13.2.1 Add a second Neuron	88
13.2.2 Add a third Neuron	88
13.2.3 Single Layer Neural Network	88
13.2.4 Introducing a Hidden Layer	89
13.2.5 Deep Learning	89
13.3 Neural Network Training by Gradient Descent and Back-Propagation	89
13.3.1 Feed Forward Error Calculation	89
13.3.2 Choosing a Cost Function	89
13.3.3 Recap - Feed Forward Neural Net	90
13.3.4 Back Propagation	90
13.3.5 Training a Feed Forward Neural Network	92
13.4 Activation Functions and the Soft-Max Classifier	92
13.4.1 Choosing an Activation Function	92
14 Back Propagation	94
14.1 Motivation	94
14.2 The Chain Rule	94
14.2.1 The Logistic Function	94
14.2.2 The Multivariate Chain Rule	94
14.3 Gradient Descent again	94
14.4 Back Propagation	96
14.4.1 Familiar Cost & Activation Functions	96

14.4.2	The Multi-Variate Chain Rule	96
15	Convolutional Neural Networks	97
15.1	ImageNet Challenge	97
15.2	What an Image really is	97
15.3	The Naive Approach	97
15.3.1	Tackled the Problem in the Old Times	97
15.3.2	Invariance to Position, Scaling, Rotation	97
15.4	Convolutions & Pooling	97
15.4.1	Filter Matrices	98
15.4.2	Convolutional Layers	98
15.4.3	Pooling	98
15.5	Model Architectures	99
15.5.1	Convolutions on RGB Images	99
15.5.2	Computer Vision Disciplines	100
16	Transfer Learning	101
16.1	Transfer Learning	101
16.2	Transfer Learning Approaches	101
16.2.1	Libraries of pre-trained Models	101
16.2.2	Model Repurposing	101
16.2.3	Fine-Tuning	102
16.2.4	Transfer Learning in Industry	102
16.3	Issues with Supervised Learning	102
16.4	Unsupervised Pre-Training	102
16.4.1	Masked Language Modelling	102
16.4.2	Contrastive Learning	103
17	Recurrent Neural Networks	104
17.1	Unsupervised NLP	104
17.1.1	Form Syntax to Semantics	104
17.1.2	Word Relatedness	104
17.1.3	Word Similarity	104
17.1.4	Continuous Bag of Words applied to 3-Grams	104
17.1.5	Mathematics with Text - Word embeddings	105
17.2	Supervised NLP	105
17.2.1	Recurrent Neural Networks	105
17.2.2	Deep Inside a Sequence-to-Sequence RNN	106
17.2.3	Gated Recurrent Units	106
17.2.4	Long Short-Term Memory Model	106
17.2.5	Bidirectional RNNs	106
17.2.6	Encoder-Decoder Architecture	106
17.3	Attention Models	107
17.3.1	RNNs do not parallelize	107
17.3.2	Transformers	107
18	Code CheatSheet	109
18.1	Python	109
18.2	Pandas	109
18.3	Numpy	109
18.4	Sklearn	109

Abbildungsverzeichnis

2.1	Entwicklung von Artificial Intelligence	15
2.2	Mehrere Optionen zur Hardwarebeschleunigung	17
3.1	Geometrische Interpretation von Daten	18
3.2	Manhattan Distanz bei Schachbrett-Muster	20
3.3	K-Nearest Neighbors Classification	22
3.4	K-Nearest Neighbors Regression	23
5.1	Feature Engineering	26
5.2	Image Data	27
5.3	Text Feature Extraction	28
5.4	De-skewed Data	30
5.5	Kernel Trick	30
5.6	Spectrogram	31
5.7	K-Means Cluster mit $k = 3$	32
5.8	Histogram Equalization	33
5.9	Features of Language and Text	34
5.10	One-Hot Encoding	35
5.11	TF-IDF	36
6.1	Vergleich Overfit vs. weiche Approximation	39
6.2	Einfacher Machine Learning Workflow	39
6.3	Standard Workflow Supervised Learning	39
6.4	Evaluation Workflow Hyperparameter Optimization	40
6.5	10-Fold Cross-Validation	41
6.6	ValidationWorkflowSupervisedLearning	41
7.1	Batch Gradient Descent Example	43
7.2	Stochastic Gradient Descent	43
7.3	Convergence of Gradient Descent	44
7.4	Batch vs. Stochastic Gradient Descent	44
7.5	Stopcriterias	45
8.1	Lineare Regression by Gradient Descent	46
8.2	Logistic Regression by Gradient Descent	47
8.3	Neural Net Training by Gradient Descent	48
8.4	Deep Learning by Gradient Descent	48
8.5	Linear Regression	49
8.6	Terminology in linear Regression	49
8.7	The Cost Function in Matrix Form	50
8.8	Konvexe Funktion	50
8.9	Partial Derivative OLS	51
8.10	OLS Example	51
8.11	When fit Linear Regression	51
8.12	Visualize the Residuals	52

8.13 OLS - M Regressors	53
8.14 Nonlinear Regression	53
9.1 Decision Boundary	54
9.2 Two Dimensional Example	56
9.3 Defining a Costfunction for Logistic Regression	57
9.4 Cross Entropy Function $Y = 1$	57
9.5 Cross Entropy Function $Y = 0$	58
9.6 Non-Linear Decision Boundaries	58
9.7 One vs. All	58
9.8 Confusion Matrix	59
9.9 Summary of Classification Metrics	60
9.10 Bias and variance	61
10.1 Uncertainty in Data	62
10.2 Vector Triangles	63
10.3 Scalar Product as Projection	64
10.4 Hessian Normal Form	64
10.5 Example Hessian Normal Form	65
10.6 Distanz Hessian Normal Form	65
10.7 Bias and Offset	65
10.8 Shifting the Coordinate System	66
10.9 Distance to Hyperplane Point Q	66
10.10 Signed Distances	66
10.11 Example Signed Distances	67
10.12 Label Encoding	67
10.13 Introduction the Margin	67
10.14 Controlling the Margin	68
10.15 Elegant Problem Formulation	68
10.16 Hard Margin Classifier	69
10.17 Outlier Sensitivity	69
10.18 Soft Margin Classifier	69
10.19 Slack Variables	70
10.20 Unconstrained Optimization Problem	70
10.21 Example Polynomial Kernel	71
10.22 Example Polynomial Kernel	71
10.23 Example Polynomial Kernel	71
10.24 Example RBF Kernel	72
10.25 Lagrange Method	72
10.26 Lagrange Transformation	73
10.27 Partial Derivatives of Lagrange Formulation	73
10.28 Towards the Dual Problem Formulation	73
10.29 Dual Problem Formulation	73
10.30 Kernel Hard-Margin SVM	74
10.31 Kernel Hard-Margin SVM at Classification Time	74
11.1 Adding Decisions	76
11.2 Gini Impurity	77
11.3 Gini efficient	77
12.1 Different Epochs	80
12.2 Mini-Batch Gradient Descent	82

12.3 Step-Size Tuning Gradient Descent	82
12.4 Diagnosing Under-Fitting	82
12.5 Diagnosing Under-Fitting	83
12.6 Cross-Validation for Regularization Tuning	83
12.7 Machine Learning SWOT	84
13.1 The Biological Neuron	86
13.2 The Perceptron as a Logic Unit	87
13.3 Logic using a Single Layer Perceptron	87
13.4 Linear vs. Non-Linear Models	87
13.5 Add a second Neuron	88
13.6 Add a third Neuron	88
13.7 Add a third Neuron	88
13.8 Hidden Layers Create new Features	89
13.9 Feed Forward Neural Net	90
13.10 Labeled Data to drive Gradient Descent	90
13.11 Back Propagation	90
13.12 NN all together	91
13.13 NN all together in Words	91
13.14 Training a Feed Forward Neural Network	91
13.15 The Rectified Linear Unit (ReLU)	92
13.16 The Leaky ReLU	93
13.17 Soft-Max for Multi-Label Classification	93
13.18 Summary of Activation Functions	93
14.1 Three Layer Neural Network	95
14.2 <i>L</i> -Layer Neural Network	95
14.3 Gradient Descent Training	95
14.4 Back Propagation Derivation	96
14.5 Revisiting Gradient Descent	96
14.6 The Multi-Variate Chain Rule	96
15.1 Transformation to Convolutional Layers	98
15.2 Pooling	99
15.3 Pooling Layers	99
15.4 The Big Picture of CNN	99
15.5 The Big Picture of CNN	100
15.6 The Big Picture of CNN	100
16.1 Masked Language Modelling	103
16.2 Masked Language Modelling	103
17.1 CBoW	105
17.2 Recurrent Neural Networks	106
17.3 Unidirectional Encoder-Decoder Architecture	107
17.4 Bidirectional Encoder-Decoder Architecture	107
17.5 Attention Layer	108
17.6 Math in Attention Layer	108

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 Covarianz

Die Covarianz 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.

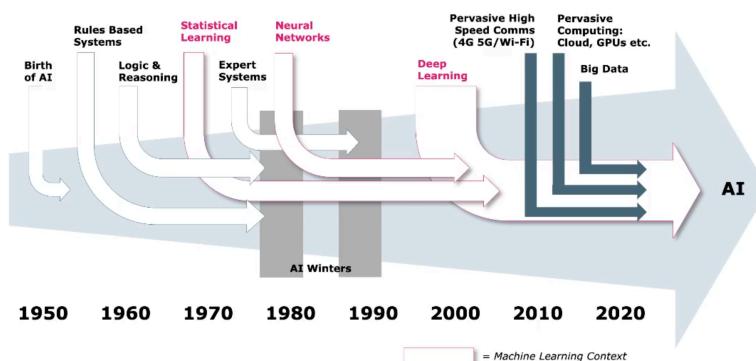


Abbildung 2.1: Entwicklung von Artificial Intelligence

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.

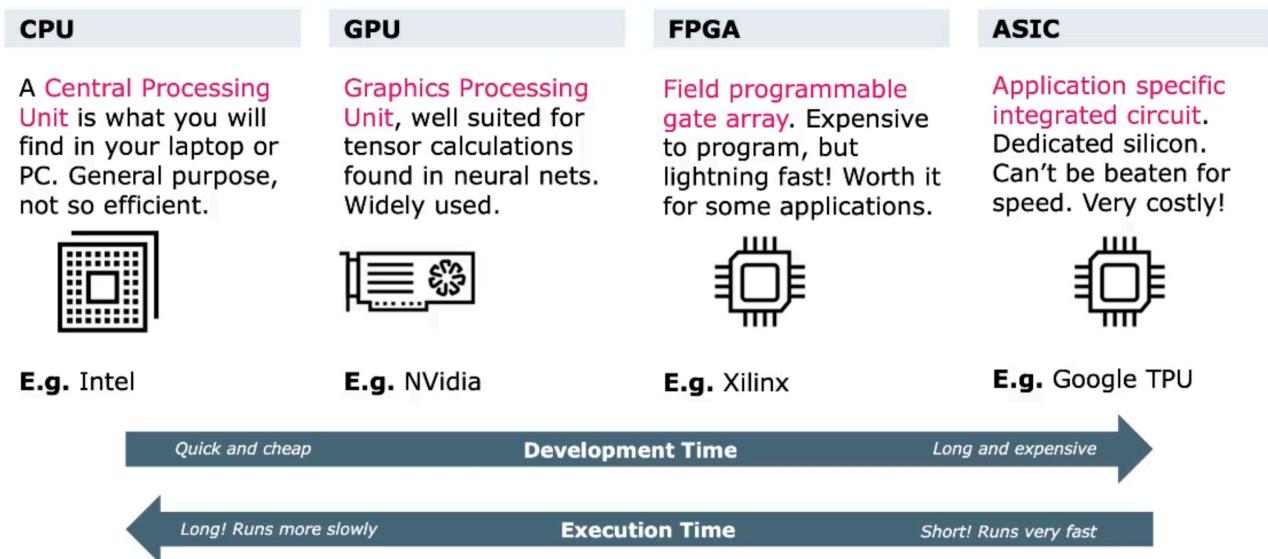


Abbildung 2.2: Mehrere Optionen zur Hardwarebeschleunigung

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.

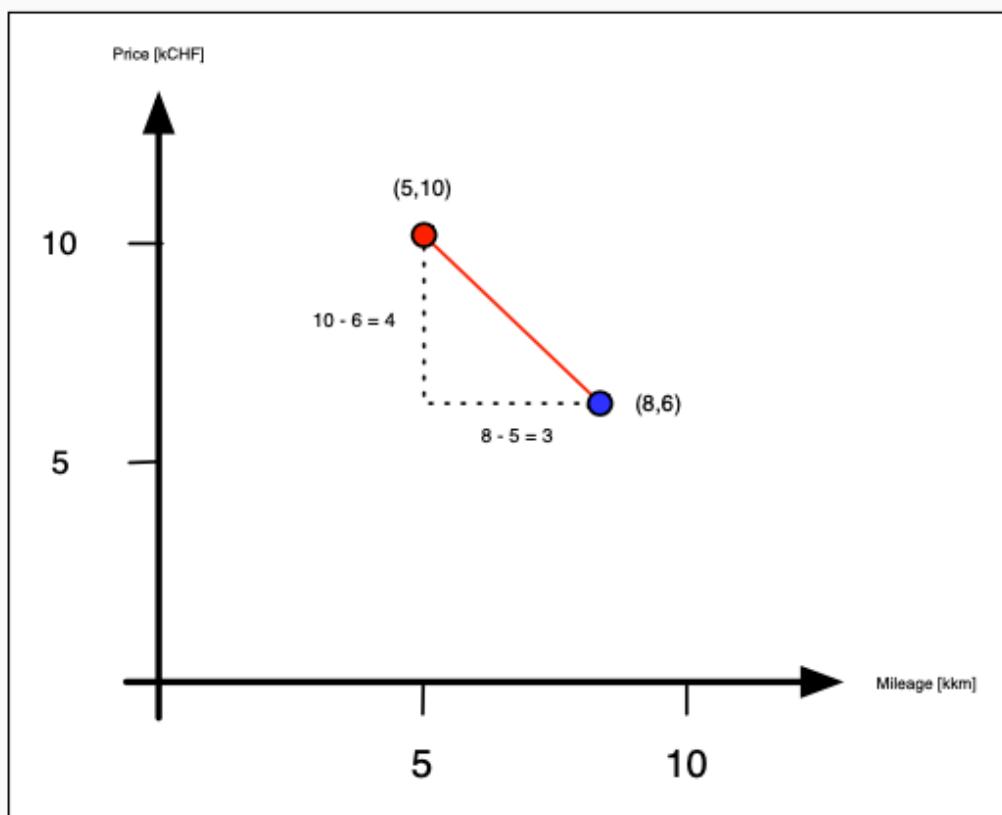


Abbildung 3.1: Geometrische Interpretation von Daten

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 Bildern können mit Hilfe von Filtern 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 Euclidian 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

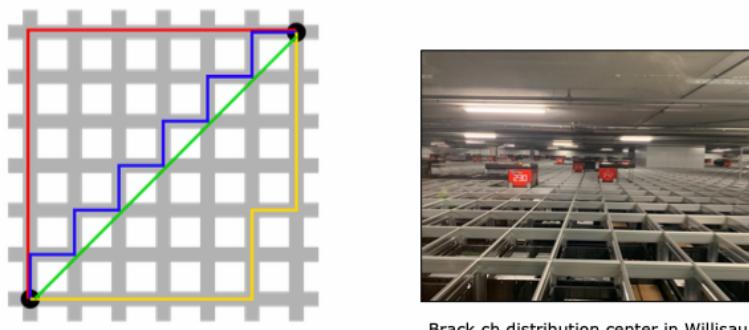
Die beiden Varianten können ganz unterschiedliche Resultate liefern. Werden die Punkte auf den Einheitskreis normiert, ergeben beide Verfahren dasselbe Resultat.

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, ∞ [



Brack.ch distribution center in Willisau

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.

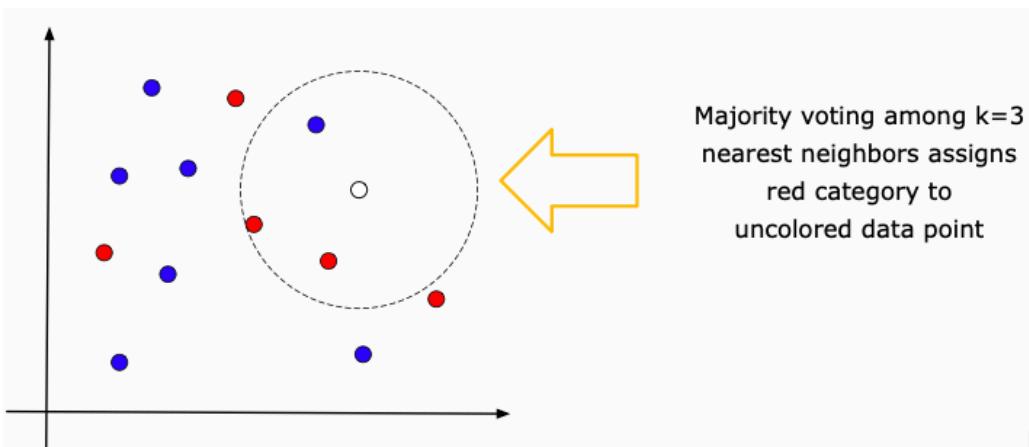


Abbildung 3.3: K-Nearest Neighbors Classification

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.

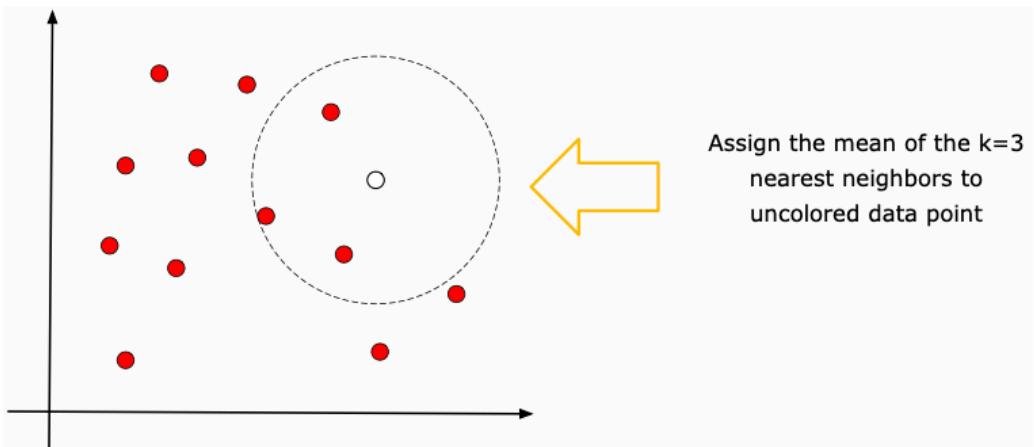


Abbildung 3.4: K-Nearest Neighbors Regression

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.

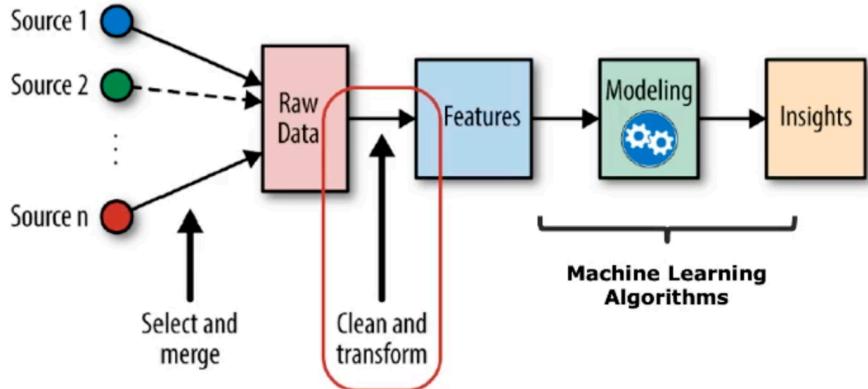


Abbildung 5.1: Feature Engineering

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.

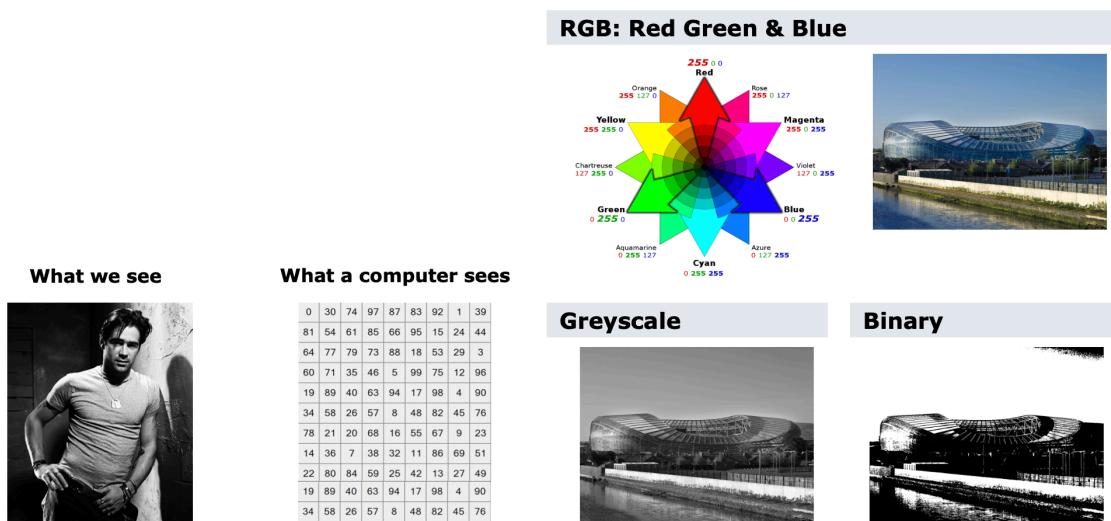


Abbildung 5.2: Image Data

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

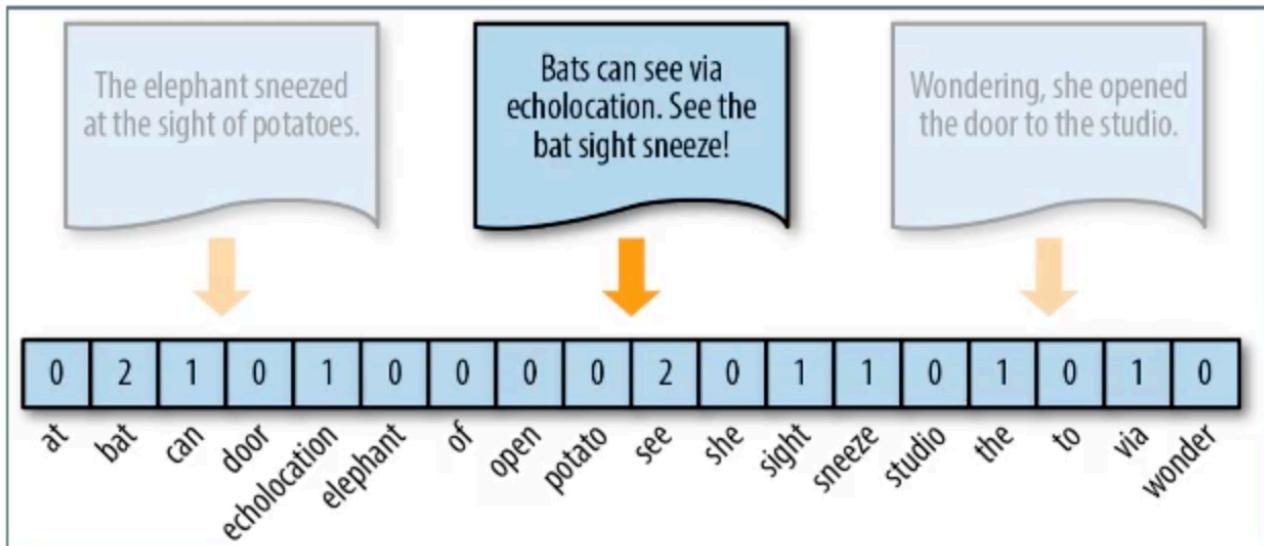


Abbildung 5.3: Text Feature Extraction

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.

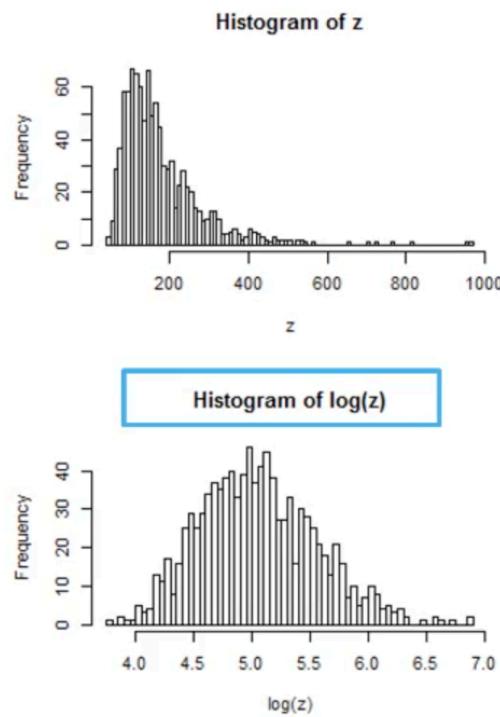


Abbildung 5.4: De-skewed Data

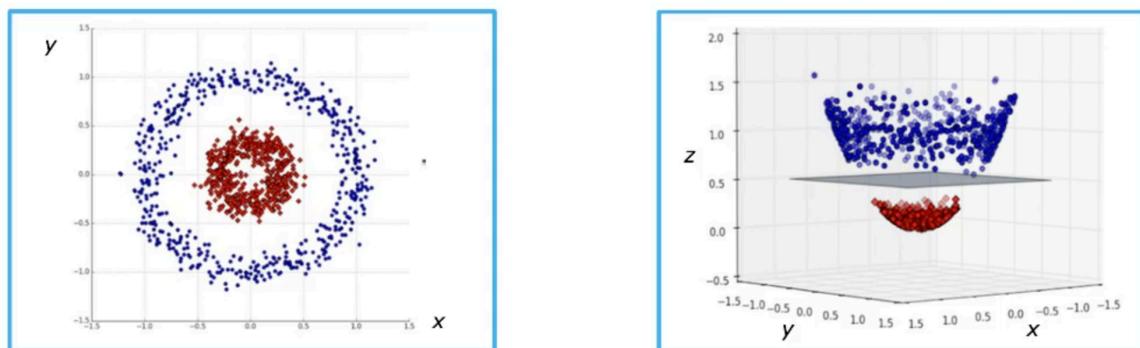


Abbildung 5.5: Kernel Trick

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()
```

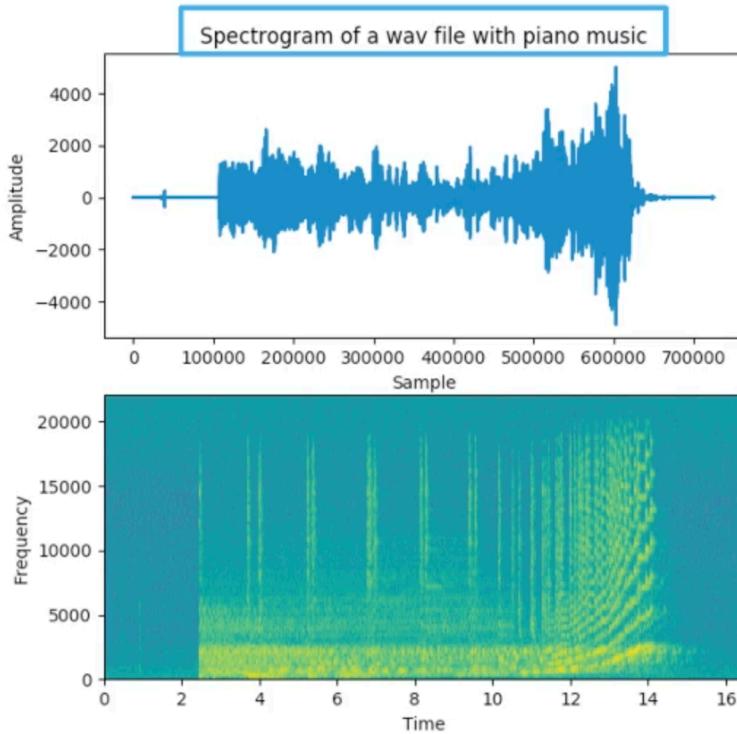


Abbildung 5.6: Spectrogram

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.

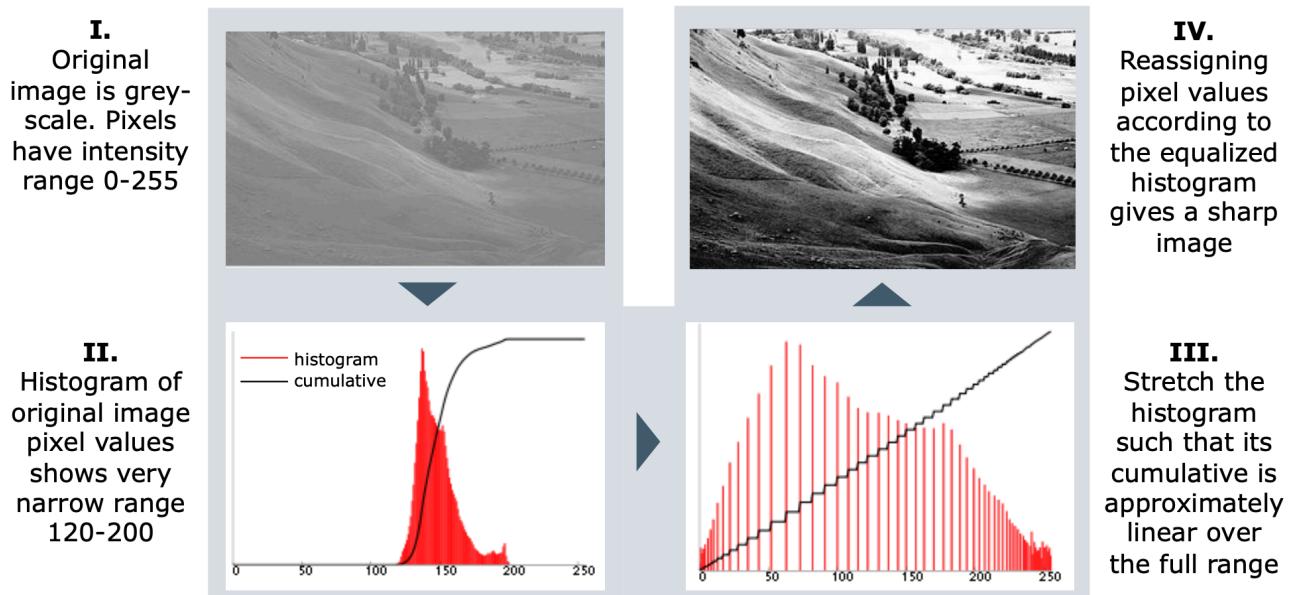


Abbildung 5.8: Histogram Equalization

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.

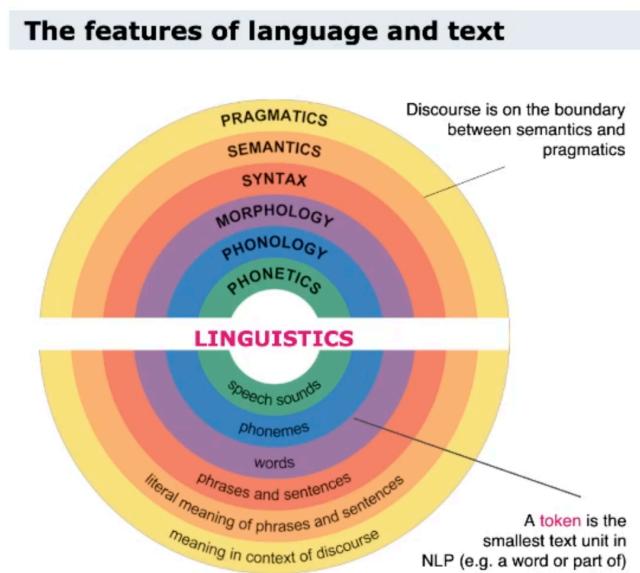


Abbildung 5.9: Features of Language and Text

- 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	1	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.

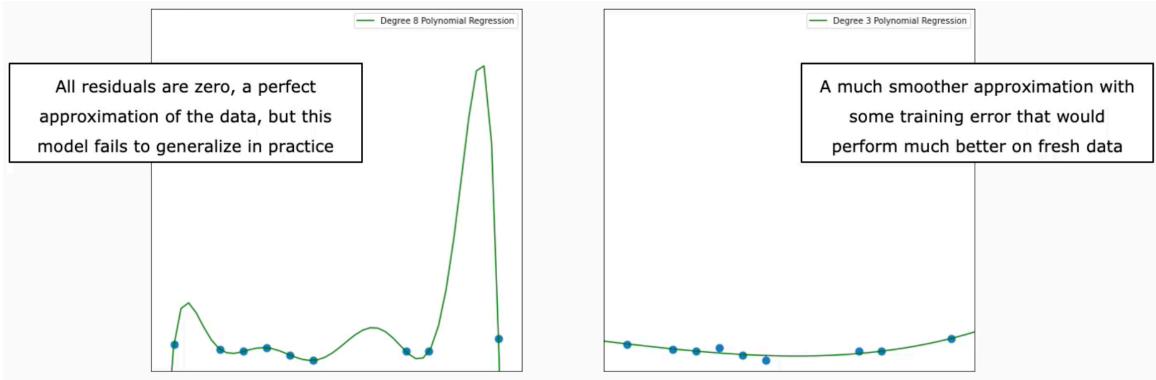


Abbildung 6.1: Vergleich Overfit vs. weiche Approximation

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

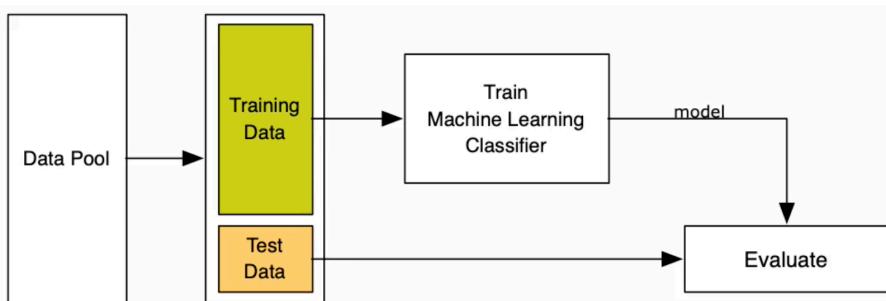


Abbildung 6.2: Einfacher Machine Learning Workflow

weiteres Beispiel

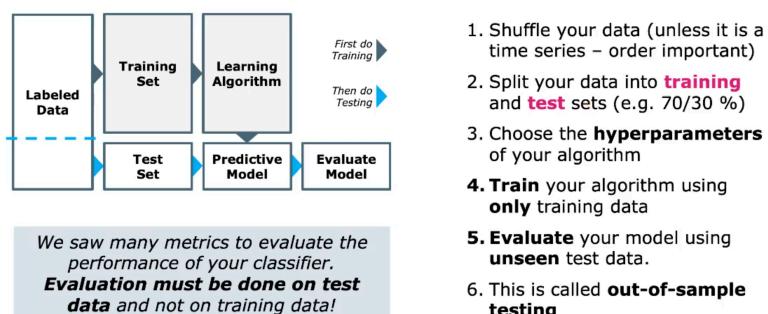


Abbildung 6.3: Standard Workflow Supervised Learning

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 Polynomials für Regressions Modelle. Als Engineer müssen wir die besten Hyperparameter aussuchen, welche auf ungesesehenen Daten am besten performt. Ein simpler Train-Test split funktioniert da leider nicht.

Parameter hingegen sind optimierbare Gewichte im System die automatisch optimiert werden können (zb. mit [Gradient Descent](#)).

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 Hyperparametern 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 6.4: 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 6.5 ausgeführt.

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.

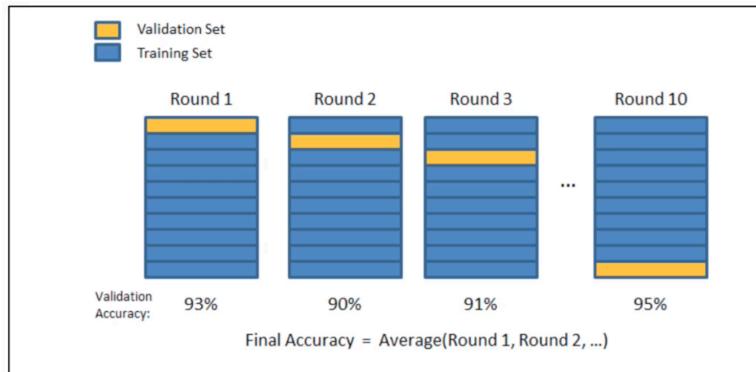


Abbildung 6.5: 10-Fold Cross-Validation

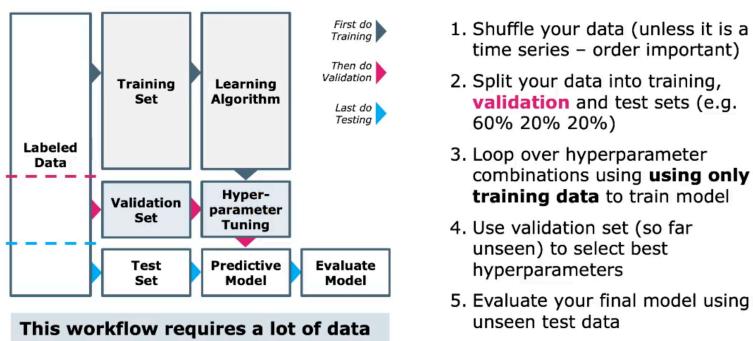


Abbildung 6.6: ValidationWorkflowSupervisedLearning

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.

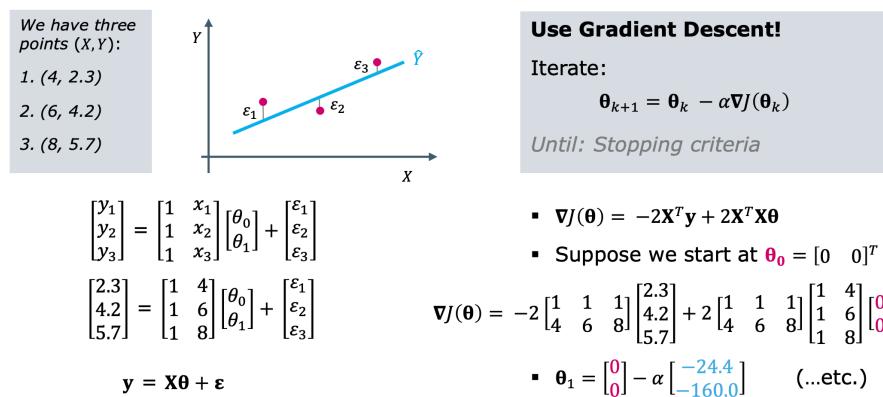


Abbildung 7.1: Batch Gradient Descent Example

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.

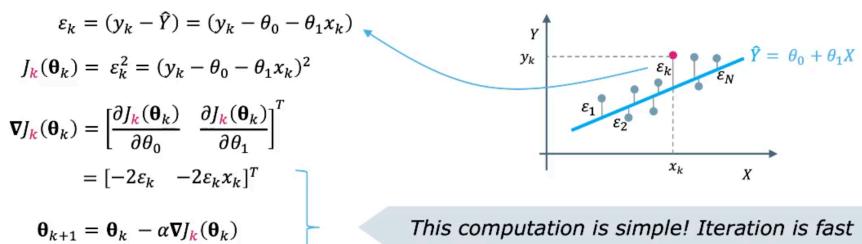


Abbildung 7.2: Stochastic Gradient Descent

7.3.1 Convergence of Gradient Descent

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

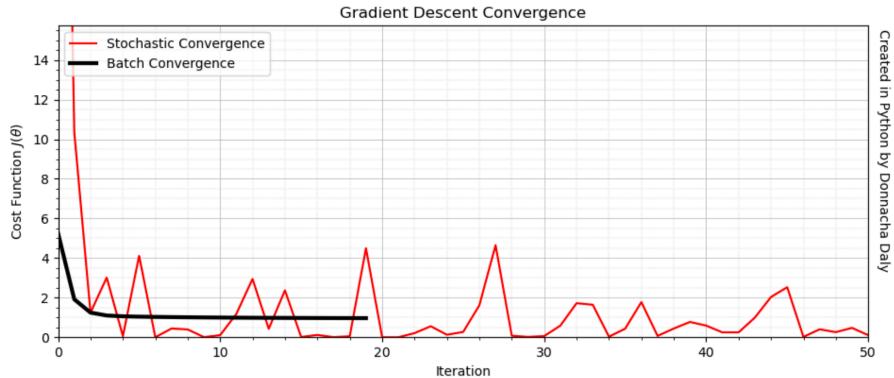


Abbildung 7.3: Convergence of Gradient Descent

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

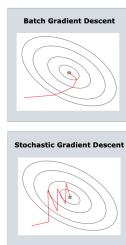


Abbildung 7.4: Batch vs. Stochastic Gradient Descent

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.

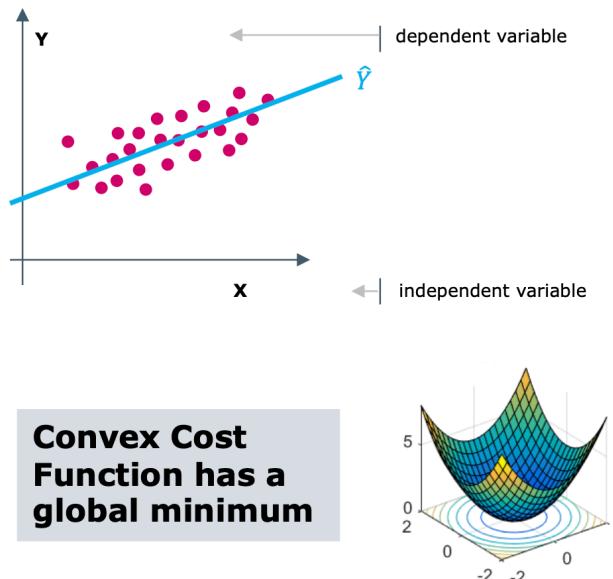
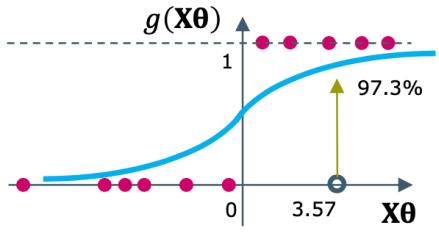


Abbildung 8.1: Lineare Regression by Gradient Descent

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.

99# Linear Regression

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

8.5 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:

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

8.5.1 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.

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

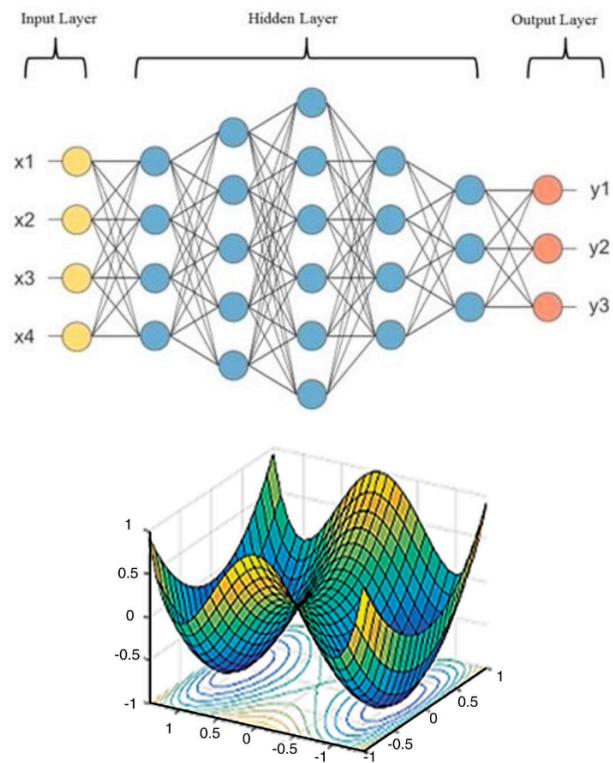


Abbildung 8.3: Neural Net Training by Gradient Descent

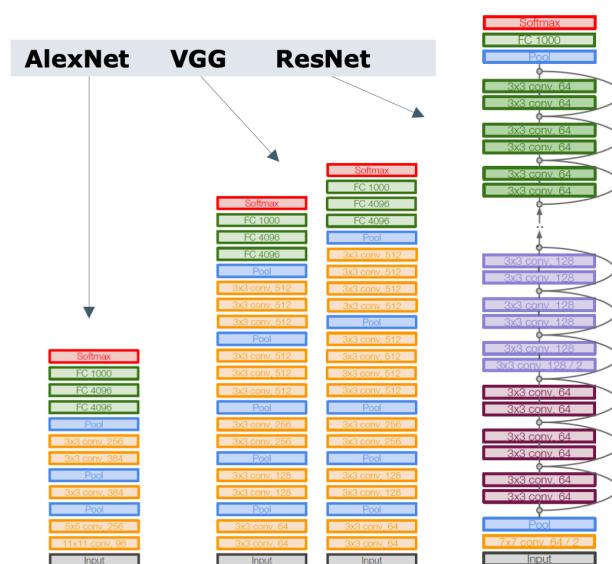


Abbildung 8.4: Deep Learning by Gradient Descent

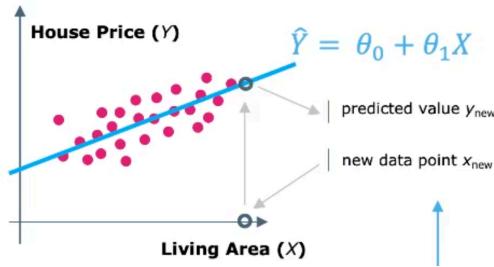


Abbildung 8.5: Linear Regression

8.5.2 Terminology

- 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)
- $y_j = \theta_0 + \theta_1 x_j + \epsilon_j$

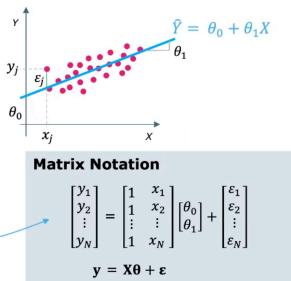


Abbildung 8.6: Terminology in linear Regression

8.5.3 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.

8.6 Ordinary Least Squares (OLS)

Wie können wir den Regressionsfehler messen?

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

8.6.1 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* $J(\theta_0, \theta_1)$.

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

8.6.1.1 Writing the Cost Function in Matrix Form

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

Abbildung 8.7: The Cost Function in Matrix Form

8.6.1.2 Minimizing the Cost Function

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

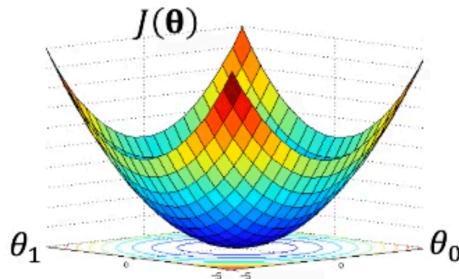


Abbildung 8.8: Konvexe Funktion

Das Minimum finden wir mit *Calculus*.

8.6.2 Ordinary Least Squares

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

8.6.2.1 Example in Python

8.6.3 Linear Regression by Gradient Descent

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

8.7 Regression Performance (R^2)

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

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.

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

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

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

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

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

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

OLS!

Abbildung 8.9: Partial Derivative OLS

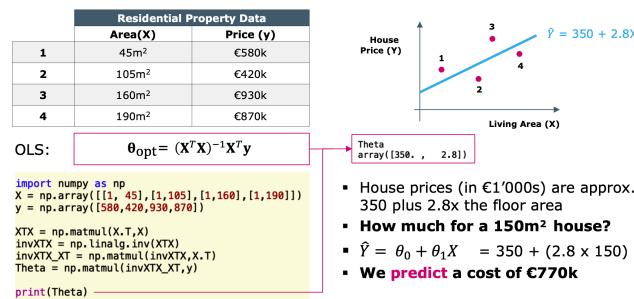


Abbildung 8.10: OLS Example

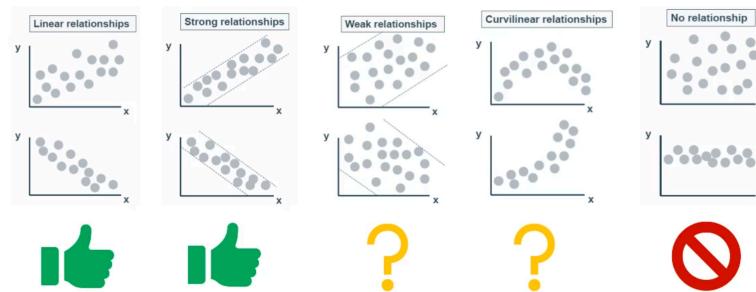


Abbildung 8.11: When fit Linear Regression

8.7.1 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!

8.7.2 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.

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

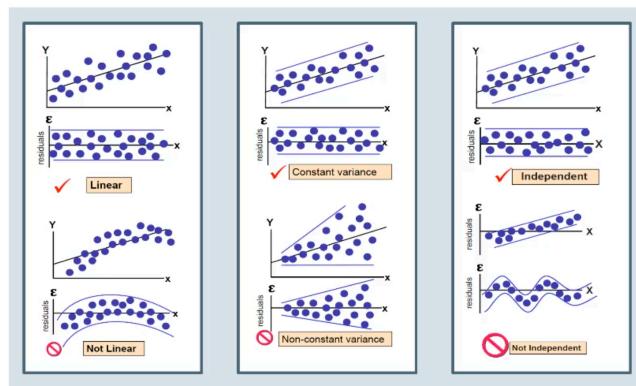


Abbildung 8.12: Visualize the Residuals

8.7.3 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 Abhängigkeit* messen erwarten wir einen Zusammenhang so dass gilt:

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

Durch Korrelation gilt nicht automatisch ein kausaler Zusammenhang.

8.8 Multiple Linear Regression

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

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

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

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

Abbildung 8.13: OLS - M Regressors

- 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**

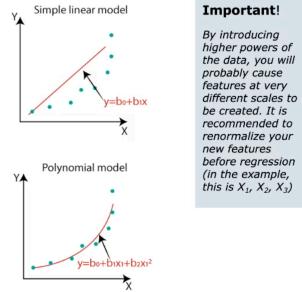


Abbildung 8.14: Nonlinear Regression

8.8.1 Matrix with M-Regressors

8.8.2 Nonlinear Regression

8.8.3 Regularization

Wenn wir mehr extra Feature haben, um unsere Trainingsdaten fitten, kann es sein dass das Model overfittet. Overfitting führt zu Generalisierungsfehler wenn neue Daten kommen. Wir müssen uns auf die Feature fokussieren welche die grösste erklärende Macht haben. Dies ist Regularisierung.

8.8.3.1 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

8.8.3.2 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*

9 Classification

Wie können Daten in versch. Klassen separiert werden.

9.1 Introduction

Binary Classification ist die Klassifizierung in zwei Klassen (Features). Wenn wir nun einen neuen Datenpunkt erhalten, ist die Frage in welche Klasse wir den neuen Punkt zuweisen können.

9.1.1 Classification Requires a Decision Boundary

Klassifizierung ist eine Form von *Supervised Learning*. Dem Algorithmus werden gelabelte Trainingsdaten übergeben. Die Daten müssen vorgängig von einem Experten klassifiziert werden. Um eine Voraussage zu machen benötigen wir einen Klassifizierungsalgorithmus, welcher das Label von *unseen Samples* erkennen kann.

Das Klassifizierungsproblem reduziert sich auf das Finden einer passenden *Decision Boundary*, basierend auf den gelabelten Trainingsdaten. Das Beispiel in Abbildung 9.1 zeigt eine *lineare* Boundary, non-lineare sind aber auch möglich.

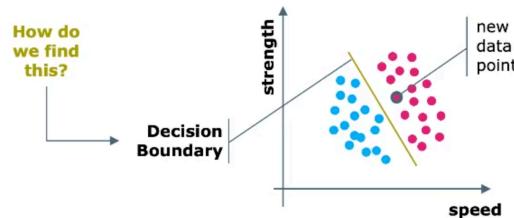


Abbildung 9.1: Decision Boundary

Es gibt mehrere Methoden für Klassifizierung:

- Logistic Regression
- k-NN
- Decision Tree
- SVM mit RBF-Kernel
- SVM mit n -polynomialem Kernel
- Gaussian Naive Bayes

9.1.2 Simple Classification: *k*-Nearest Neighbours

k-NN ist eine Mehrheitswahl der k -nahesten Punkte. Dazu wird ein Distanzmaß benötigt z.B. [Euclidean Distance or \$L^2\$ -Norm](#). Ist am wenigsten Datenhungrig.

9.1.2.1 Changing Hyperparameter k

Verschiedene k ergeben verschiedene Klassifizierungsresultate. Mit der *Confusion Matrix* und entsprechenden Metriken können wir die Performance messen. Die optimale k finden wir mittels *Hyperparameter tuning*. Wichtig, damit die Skalenwerte (oder Dimensionen) der Variablen zu einander passen müssen die Daten **normalisiert** werden!! Ansonsten dominiert eine der beiden.

9.1.2.2 Pros and Cons of k -NN

pros

- Einfach zu implementieren z.B. mit `sklearn` Package für Python.
- ideal für kleine Datenset
- als *Baseline* nützlich um andere Klassifikatoren zu vergleichen

cons

- langsam weil die ganze Berechnung zu Klassifizierungszeit gemacht wird (keine Learningphase)

9.1.3 Classification vs. Regression

Klassifizierung sagt *Kategorien* voraus. Die gelabelten Daten sind vom Typ *categorical*. Der Klassifikator lernt eine **Decision Boundary**

Beispiel Klassifizierungsproblem: Sagt Kaufverhalten voraus (ein Label), basierend auf categorical labeled trainingsdaten. Das Attribut welches vorausgesagt werden soll ist kategorisch

Regression sagt *Werte/Zahlen* voraus. Die gelabelten Daten sind numerisch und lernt die Beziehung/Zusammenhang in den Daten.

Beispiel Regressionsproblem: Sagt Preis (numerisch) eines Autos voraus, welcher anhand gelabelten Trainingsdaten erlernt wurde. Das Attribut welches vorausgesagt werden soll ist continuierlich (numerisch).

9.2 Logistic Regression

Ist ein binäres, lineares *Klassifizierungs*-Problem.

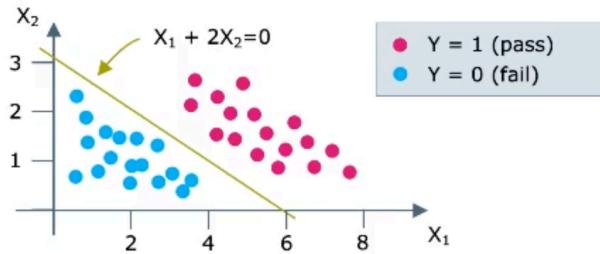
9.2.1 Binary Classification using 1 Feature

Bei der Klassifizierung separieren wir die Daten in *Kategorien*. Wenn ein kategorisches *Label Y* nur zwei Werte (z.B besucht/nicht besucht) annehmen kann, sprechen wir von *Binary Classification*. Wir labelen diese Werte mit 1 bzw. 0 (true/false). Beispiel, Wenn *X* bei Wert 10 das Label separiert (1/0), ist diese Linie die *Decision Boundary*.

9.2.2 Binary Classification using 2 Feature

Mit mehr als einem Features welches die Klassen unterscheidet, gehen wir für die Klassifizierung in eine höhere Dimension. Bei zwei Features entspricht unsere Hypothese einer Geraden. Dafür benötigen wir einen *linearen binären Klassifikator*. Dazu können wir *Logistic Regression* nutzen.

Two Dimensional Example



In the example shown the line " $X_1 + 2X_2 = 0$ " separates the classes well. It is a good **decision boundary**.

Abbildung 9.2: Two Dimensional Example

9.2.3 Logistic Function

Eine *Logistic Function*, siehe in Abbildung 9.2.3 links, ist eine symmetrische Sigmoid (S-) geformte Funktion $g(z) = \frac{1}{1+e^{-z}}$ mit vielen angenehmen Eigenschaften wie $0 < g(z) < 1$ oder dass sie stetig differenzierbar ist: $g'(z) = g(z)(1 - g(z)) \geq 0$

Logistic Function and Regression {width=60%}

9.2.4 Logistic Regression

In der logistic Regression fitten wir **sigmoid** $\hat{\mathbf{Y}} = g(\mathbf{X}\Theta)$ ¹. Die Decision Boundary, siehe in Abbildung 9.2.3 rechts, ist als $\mathbf{X}\Theta = 0$ definiert. Wir erhalten eine W'keit ob eine Klasse zu 1 oder 0 gehört.

9.2.5 Logistic as a Probability Measure

Weil die Werte zwischen 0 und 1 liegen ist es eine ideales Wahrscheinlichkeitsmass. Ist z gross und positiv die Wahrscheinlichkeit ist *hoch*, dass die Daten zu der 1er-Klass gehören. Ist z negativ, die ist Wahrscheinlichkeit *klein*, der 1er-Klass anzugehören. Ist $z = 0$ sind wir genau auf der Decision Boundary. Die W'keit ist 50%, dass die Daten zur 1er-Klass gehören.

9.2.6 Logistic: Probability of Belonging to a Class

Wir berechnen die W'keit aus einer gewichteten linearen Kombi der Features $\mathbf{X}\Theta$. Generell, haben wir für n -Datenpunkte und m Features:

$$P(Y = 1|\mathbf{X}) = \hat{\mathbf{Y}} = g(\mathbf{X}\Theta), \text{ wobei } g(z) = \frac{1}{1 + e^{-z}}$$

¹Remember; in linearer Regression fitten wir eine Gerade $\hat{\mathbf{Y}} = \mathbf{X}\Theta$. Θ ist immer noch eine lineare Kombi aus den Parameter.

mit $\mathbf{X}\Theta = 3.57$ gibt das:

$$P(Y = 1) = \frac{1}{1 + e^{-3.57}}$$

Die sigmoid-Funktion sollte genau zwischen den beiden Punkten sein.

9.2.7 Cost Function for Logistic Regression

Wenn $Y = 1$, möchten wir dass $g(\mathbf{X}\Theta)$ so nahe wie möglich an 1 ist. Dies ist die W'keit, dass $Y = 1$. Zur Herleitung der Kostenfunktion suchen wir nach einer Funktion, die tief ist, falls $g(\mathbf{X}\Theta)$ nahe bei 1 und hoch wenn $g(\mathbf{X}\Theta)$ nahe bei 0 (dann ist es eine mis-classification).

In ähnlicher Weise machen wir das für $Y = 0$. Wir definieren eine Kostenfunktion, die tief ist, falls $g(\mathbf{X}\Theta)$ nahe 0 und hoch, wenn $g(\mathbf{X}\Theta)$ nahe 1 (mis-classification).

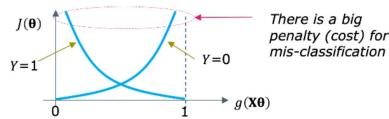


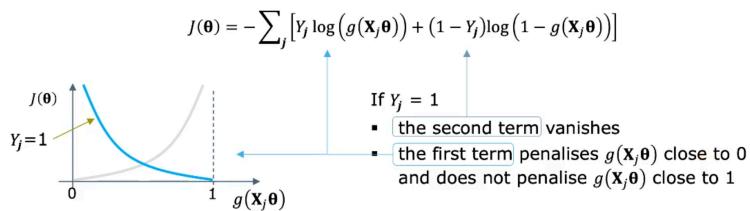
Abbildung 9.3: Defining a Costfunction for Logistic Regression

Die *cross-entropy*-Funktion hat genau diese Eigenschaften gemäss Abbildung 9.3.

9.2.8 Cross Entropy

Für $Y = 1$ gilt

- If we have N labelled data points (\mathbf{x}_j, Y_j) with $j = 0, 1, 2, \dots, N$
then the **cross entropy** cost function is defined as follows:



This is advanced material but you don't have to fully understand the details of cross entropy to appreciate that we have a cost function which penalizes mis-classification

Abbildung 9.4: Cross Entropy Function $Y = 1$

Für $Y = 0$ gilt

9.2.9 Logistic Regression by Gradient Descent

In Logistic Regression können wir die optimalen Parameter nicht mit der OLS finden. Deshalb nutzen mit Gradient Descent dazu.

9.2.10 Non-Linear Decision Boundaries

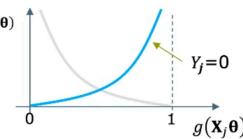
Mit Feature Engineering können auch Decision Boundaries finden, wenn keine lineare möglich ist. Im Beispiel in Abbildung 9.6 fügen wir zwei Features $V_1 = X_1^2$ und $V_2 = X_2^2$ (abgeleitet von V_1 und V_2 hinzu. Damit finden wir die Decision Boundary $g(\mathbf{X}\Theta) = 50$

- If we have N labelled data points (\mathbf{x}_j, Y_j) with $j = 0, 1, 2, \dots, N$ then the **cross entropy** cost function is defined as follows:

$$J(\boldsymbol{\theta}) = - \sum_j [Y_j \log(g(\mathbf{x}_j, \boldsymbol{\theta})) + (1 - Y_j) \log(1 - g(\mathbf{x}_j, \boldsymbol{\theta}))]$$

If $Y_j = 0$

- the first term vanishes
- the second term penalises $g(\mathbf{x}_j, \boldsymbol{\theta})$ close to 1 and does not penalise $g(\mathbf{x}_j, \boldsymbol{\theta})$ close to 0



This is advanced material but you don't have to fully understand the details of cross entropy to appreciate that we have a cost function which penalizes mis-classification

Abbildung 9.5: Cross Entropy Function $Y = 0$

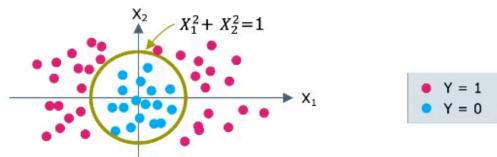


Abbildung 9.6: Non-Linear Decision Boundaries

9.2.11 One Versus the Rest

Falls wir mehrere Klassifizierungen machen müssen, machen wir eine «One vs. All» Klassifizierung. Danach iterieren wir in die nächste Gruppe und klassifizieren weiter.

Iterated binary classification

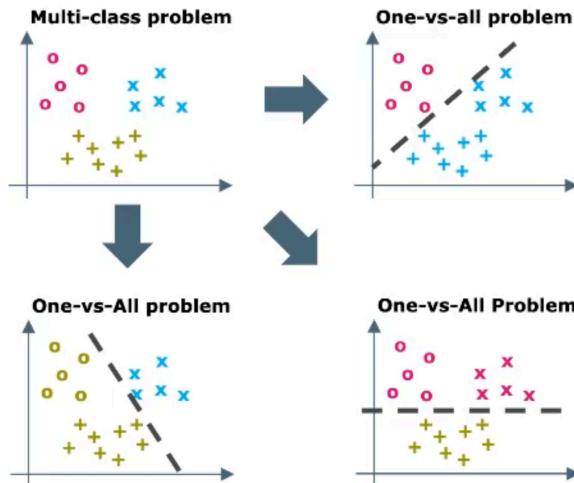


Abbildung 9.7: One vs. All

9.3 Performance Analysis

True positive/negatives bedeutet, dass der Klassifier richtig bewertet hat. **False positive/negatives** bedeutet, dass der Klassifier falsch bewertet hat.

Wir messen die Statistik dieser Fehler um die Performance zu evaluieren.

9.3.1 Confusion Matrix

		Predicted	
		No	Yes
Actual	No	True Negatives (TN)	False Positives (FP)
	Yes	False Negatives (FN)	True Positives (TP)

		Predicted	
		No	Yes
Actual	No	50	10
	Yes	5	100

Accuracy

how frequently are we correct?

Error-Rate

how frequently are we wrong?

$$\text{Accuracy} = (\text{TP} + \text{TN}) / \text{Total}$$

$$= 150 / 165 = 91\%$$

$$\text{Error Rate} = 1 - \text{Accuracy} = 9\%$$

Abbildung 9.8: Confusion Matrix

9.3.1.1 Imbalanced Data

Wenn die Daten sehr unausgeglichen sind (5000 No/ 20 Yes), erreicht man zwar eine hohe Accuracy, welche aber keine Aussage hat. Immer die Daten prüfen!

9.3.1.2 Sensitivity - Performance on YES Instances

Die Sensitivität misst wie oft das Modell YES voraussagt, im Verhältnis zum wahren YES. Es wird auch *True-Positive Rate* oder *Recall* genannt.

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

9.3.1.3 Specificity - Performance on NO Instances

Die Genauigkeit misst wie oft das Modell NO voraussagt, im Verhältnis zum wahren NO.

$$\text{Specificity} = \frac{\text{TP}}{\text{TN} + \text{FP}}$$

9.3.1.4 Precision - If True Negatives not Available

Manchmal sind keine TN vorhanden, dann nutzen wir Precision. Das Modell sagt YES voraus, wie oft ist das korrekt.

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

9.3.1.5 F1 Score

Ist der harmonische Mittelwert zwischen Precision und Recall (Sensitivity) in Prozent. By Design nimmt F1 keine TN zum bewerten

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

F1 ist stark vom schlechteren Score hingezogen, also dann nutzen, wenn wir ein Klassifizierungsproblem mit schiefen Daten haben!

9.3.1.6 Summary of Classification Metrics

		Predicted	
		No	Yes
Actual	No	50	10
	Yes	5	100

Sensitivity = $\frac{\text{TP}}{\text{Actual YES}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
Sensitivity is also called Recall

Specificity = $\frac{\text{TN}}{\text{Actual NO}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$

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

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

- Recall = $100 / 105 = 95\%$
- Specificity = $50 / 60 = 83\%$
- Precision = $100 / 110 = 91\%$
- F1-Score = $200 / 215 = 93\%$

F1-Score is the harmonic mean of precision & recall. It can be the best single metric to use if you have to choose.

Abbildung 9.9: Summary of Classification Metrics

9.4 Performance Optimization

Mit welchen Mittel können wir die Performance optimieren.

9.4.1 Underfitting & Overfitting in Classification

Ist ein Modell underefitted, können die Klassen im Training und Test ungenügend separiert werden. Dies bedeutet es hat einen hohe *Bias*. Ist ein Modell overfitted, ist es überoptimiert auf den Trainingsdaten. Gute Performance auf TRainingsdaten aber schlechte beim Testdaten. Dies bedeutet, es hat eine hohe *Varianz*.

9.4.1.1 Bias and Variance

Mit **Bias** ist gemeint, dass man systematisch einen bestimmten Wert vom Ziel entfernt ist. Das Modell ist konsistent aber nicht akurat im Durchschnitt. Die **Varianz** meint die systematische *Verstreuung* der Daten. Das Modell ist akurat im Durchschnitt ist aber inkonsistent und generalisiert nicht.

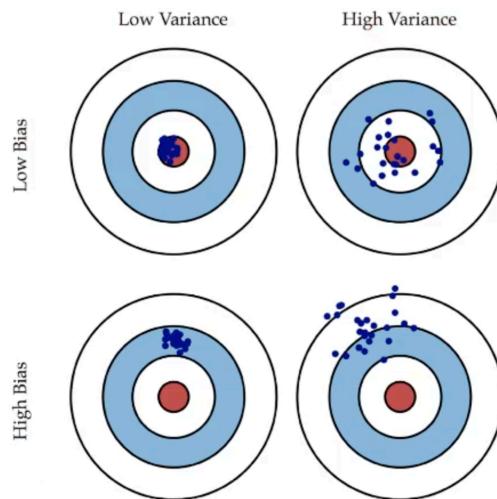


Abbildung 9.10: Bias and variance

9.4.2 How to Diehl with it

Dealing with bias versuche

- grösseres Set von Features verwenden
- verschiedene Sets von Features
- Feature Engineering
- komplexere Algorithmen verwenden
- mehr Daten muss nicht die Lösung sein

Dealing with variance versuche

- Anzahl Features reduzieren
- mehr Trainingsdaten erhalten
- **Data Cleaning** (Outliers eliminieren)
- Training früh stoppen um overfitting zu vermeiden
- Advanced:
 - Reduktion von Features automatisieren (Feature Selection)
 - Regularisierung
 - Ensemble methods

10 Support Vector Machine

Skalierbar, komplexe Entscheidungsgrenzen (Decision Boundaries) mit Kernels, insensitive auf Outliers. Es geht um Distanzen - welche zu den nahesten Punkten maximiert werden soll. Ist aber linear!

10.1 Motivation

Klassifizierungsprobleme können je nach Random Seed eine unterschiedliche Geradee als Deecision Boundary ergeben.

10.1.1 Uncertainty in Data

Unsicherheit in Daten. Z.B. ein Messgerät ist nicht immer gleich genau. Sie haben eine gewissee Unschärfe. Wenn nun eine Entscheidungsgrenze sehr nahe an Punkten anliegt, könnte es sein dass die Unschärfe dazu führt, dass ein Punkte die Grenze überschreiten würde.

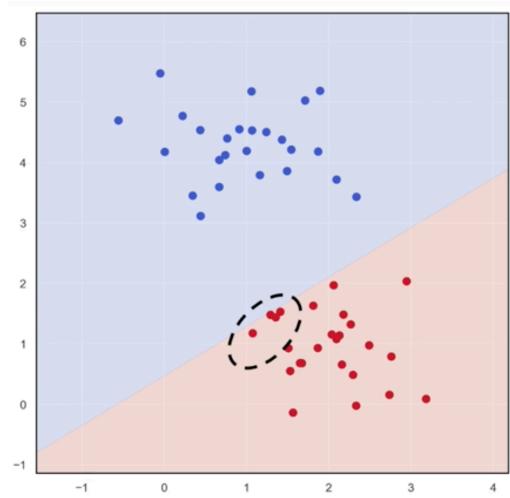


Abbildung 10.1: Uncertainty in Data

10.1.2 Large Margin Classifier

Die Support Vector State Machine ist anders. Die Optimierung der Gerade wird so gemacht, dass die Gerade so liegt, dass sie möglichst grossen Abstand zu beiden Seiten hat. Der Abstand zeigt an wie sicher der Punkt nicht auf die andere Seite springt.

Lineare Regressionsverfahren setzen die Gerade in Abhängigkeit der Geraden beliebig.

10.1.3 Support Vectors

Support Vektoren sind die Punkte im Diagramm, welche am nahesten zur Gerade liegen. Die *Margin* ist eine symbolische Gerade die parallel zur Decision Boundary verläuft und durch die Support Vektoren geht.

Weil die Descision Boundary von den Support Vektoren abhängt, ist die SVM insensitiv gegenüber Outliers. Nur wenn Support Vektoren ändern, ändert die Decision Boundary.

10.2 The Scalar Product

Das Skalar oder Dot-Produkt ist eine Nummer. Dazu werden die Vektoren elementweise miteinander multipliziert und summiert.

$$\vec{p} \cdot \vec{q} = x_1x_2 + y_1y_2 + z_1z_2$$

10.2.1 Scalar Product and Vector Length

Die L^2 -Norm (siehe [Euclidiean Distance or \$L^2\$ -Norm](#)) ist die Wurzel der Quadrate der Komponenten, aufsummieren und Wurzel ziehen.

$$\|\mathbf{x}\| = \sqrt{x_1^2 + \dots + x_n^2}$$

Ein Einheitsvektor ist es dann, wenn die Länge 1 ist. Zum normieren wird, ist Vektor durch die Länge zu dividieren.

10.2.2 Vector Triangles

Vektoren haben eine Länge und Richtung. Aber keine Position. Deshalb ergibt sich aus \vec{x} , \vec{y} und $\vec{x} - \vec{y}$ ein Dreieck (wenn $0 < \theta < 90^\circ$).

10.2.2.1 The Cosine Formula

Sofern \vec{x} und \vec{y} nicht trivial (nicht null) sind, gilt

$$\vec{x} \cdot \vec{y} = \|\vec{x}\| * \|\vec{y}\| * \cos(\theta)$$

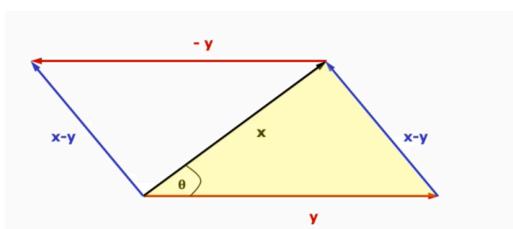


Abbildung 10.2: Vector Triangles

10.2.2.2 Scalar Product as Projection

Wenn \vec{y} ein Einheitsvektor ist, entspricht das Skalarprodukt $\vec{x} \cdot \vec{y}$ der *Projektion* von \vec{x} auf \vec{y}

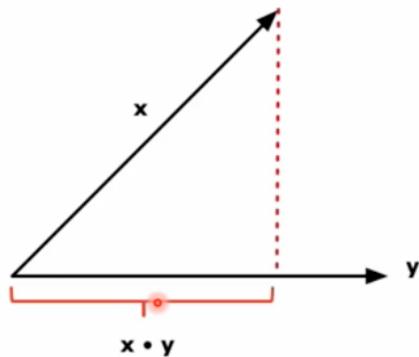


Abbildung 10.3: Scalar Product as Projection

10.2.2.3 Perpendicularity Test

Das Skalarprodukt von senkrechten Vektoren ist null. Aufgabe; finde Vektor y damit er senkrecht auf x steht. Dann Funktion $x_1 * y_1 + x_2 * y_2 + x_3 * y_3 = 0$ aufstellen und lösen.

10.3 Hyperplanes

Eine Hyperebene ist definiert durch einen Auffahrtpunkt P und einem Richtungsvektor w . Wir fordern von w , dass er ein Einheitsvektor entspricht und rechtwinklig zur Ebene steht.

10.3.1 Hessian Normal Form

- Derivation of hyperplane equation
- Let X be any other point on the hyperplane
- The vector $\mathbf{x} - \mathbf{p}$ lies on the hyperplane and points from P to X
- Because w is perpendicular to all vectors on the hyperplane we have $w \cdot (\mathbf{x} - \mathbf{p}) = 0$
- It follows that $w \cdot (\mathbf{x} - \mathbf{p}) = w \cdot \mathbf{x} - w \cdot \mathbf{p} = w \cdot \mathbf{x} + b = 0$ with $b = -w \cdot \mathbf{p}$
- The hyperplane consists of all points X such that $w \cdot \mathbf{x} + b = 0$ with $b = -w \cdot \mathbf{p}$ and $\|w\| = 1$
- This representation of hyperplanes is called **Hessian normal form**

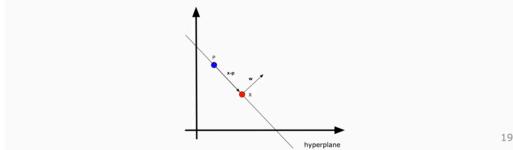


Abbildung 10.4: Hessian Normal Form

10.3.2 Shifting the Coordinate System

Vektoren müssen nicht verschoben werden, weil die keine Position haben

Example

- Anchor point $P = (0.5, 2)$ and direction vector $[2, 3]^T$
- The normal vector must have unit length, we define

$$\mathbf{w} = \frac{1}{\sqrt{13}} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{ such that } \|\mathbf{w}\| = 1$$

- We carry out the calculations from the previous slide

$$\begin{aligned} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \bullet \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right) &= \frac{1}{\sqrt{13}} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} \bullet \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 2 \end{bmatrix} \right) \\ &= \frac{2}{\sqrt{13}} \cdot x_1 + \frac{3}{\sqrt{13}} \cdot x_2 - \left(\frac{2 \cdot 0.5}{\sqrt{13}} + \frac{3 \cdot 2}{\sqrt{13}} \right) \\ &= \frac{2}{\sqrt{13}} \cdot x_1 + \frac{3}{\sqrt{13}} \cdot x_2 - \frac{7}{\sqrt{13}} = 0 \end{aligned}$$

- The hyperplane equation is

$$\frac{2}{\sqrt{13}} \cdot x_1 + \frac{3}{\sqrt{13}} \cdot x_2 + b = 0 \text{ with } b = -\frac{7}{\sqrt{13}}$$

Abbildung 10.5: Example Hessian Normal Form

Distance from Hessian Hyperplane to Origin

- The line segment for the shortest distance is perpendicular to the hyperplane
- Here this corresponds to the length of the (dashed) position vector \mathbf{x}
- The position vector \mathbf{x} must therefore have the same direction as the normal vector
- It follows that $\mathbf{x} / \|\mathbf{x}\| = \mathbf{w}$
- For any point P on the hyperplane we have $\mathbf{p} \bullet \mathbf{w} + b = 0$ and therefore $\mathbf{p} \bullet \mathbf{w} = -b$
- The scalar product $\mathbf{p} \bullet (\mathbf{x} / \|\mathbf{x}\|) = \mathbf{p} \bullet \mathbf{w}$ equals the projection, which is exactly $\|\mathbf{x}\|$
- Therefore $\|\mathbf{x}\| = \mathbf{p} \bullet \mathbf{w} = -b$ corresponds to the distance from the origin to the hyperplane



Abbildung 10.6: Distanz Hessian Normal Form

Bias and Offset

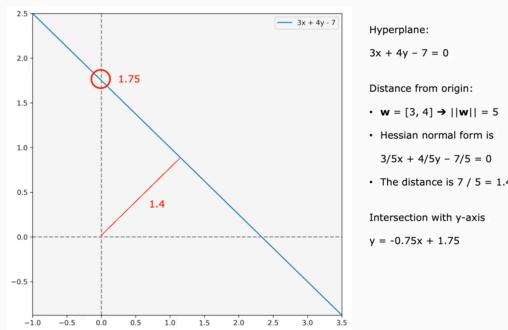


Abbildung 10.7: Bias and Offset

Shifting the Coordinate System

- We know that the distance from the origin is $\|\mathbf{w}\|$
- But how to calculate the distance from any other point Q ...
- The trick is to shift the coordinate system so that Q becomes the new origin
- Shifting the coordinate system changes all points X into $X_{\text{new}} = X - Q$

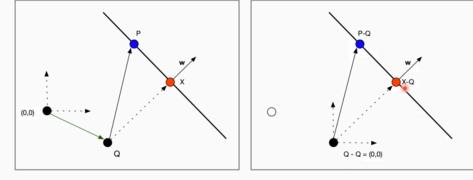


Abbildung 10.8: Shifting the Coordinate System

Distance from Hessian Hyperplane to Point Q

- Shifting the coordinate system changes the hyperplane equation to $\mathbf{w} \cdot \mathbf{x} + b_{\text{new}} = 0$
 - The distance from the new origin Q is $-b_{\text{new}}$
 - Because $P_{\text{new}} = P - Q$ is a point on the hyperplane we have $\mathbf{w} \cdot (P - Q) + b_{\text{new}} = 0$
 - Therefore
- $$\mathbf{w} \cdot P_{\text{new}} + b_{\text{new}} = \mathbf{w} \cdot (P - Q) + b_{\text{new}} = \mathbf{w} \cdot P - \mathbf{w} \cdot Q + b_{\text{new}} = -b - \mathbf{w} \cdot Q + b_{\text{new}} = 0$$
- The distance from Q to the hyperplane is $-b_{\text{new}} = -(\mathbf{w} \cdot Q + b)$

Insert the coordinates of a point into the Hessian normal form and directly obtain the distance of the hyperplane from this point

Abbildung 10.9: Distance to Hyperplane Point Q

10.3.3 How to interpret signed Distances

Eine Hyperebene teilt einen Raum in einen positiven und negativen Halbraum (+/-HP - Halfplane). By Convention zeigt der Vektor \vec{w} immer in die Richtung von -HP! Die Punkte in +HP haben eine positivee Distanz und die Punkte in -HP eine negative Distanz zur Hyperebene.

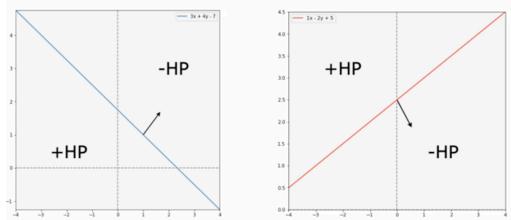


Abbildung 10.10: Signed Distances

10.4 Large Margin Classifier

In der binären Klassifizierung wählen wir für die Labels die Werte 0 und 1. Bei Support Vector Machines codieren die *labels* mit -1 und +1. Dies ist nur eine Namengebung, abere dadurch ergibt sich eine elegante Formulierung.

Die Hyperebene teilt lediglich die beiden Punktfamilien. Weil wir aber eine Distanz wollen, führen wir ein $M = 1$ ein und kontrollieren lediglich die Skalierung.

Example

- Calculate the distances from $(3, 2)$ to the hyperplane $3x_1 + 4x_2 - 7 = 0$

- Transform into Hessian normal form:

$$\mathbf{w} = [3, 4] \rightarrow \|\mathbf{w}\| = 5$$

$$3/5x_1 + 4/5x_2 - 7/5 = 0$$

- Insert the position vector $\mathbf{p} = [3, 2]$

$$3/5 * 3 + 4/5 * 2 - 7/5 = 16/5$$

$$\text{Distance is } -16/5$$

- The distance is negative because $(3, 2)$ is in $-\text{HP}$

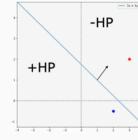
- Calculate the distances from $(2, -0.5)$ to the hyperplane $3x_1 + 4x_2 - 7 = 0$

- Insert the position vector $\mathbf{p} = [2, -0.5]$

$$3/5 * 2 - 4/5 * 1/2 - 7/5 = -6/10$$

$$\text{Distance is } +6/10$$

- The distance is positive because $(2, -0.5)$ is in $+\text{HP}$



27

Abbildung 10.11: Example Signed Distances

Label Encoding

- In a binary classification problem we traditionally encode labels with 0 and 1
- For support vector machines we encode labels with -1 and +1
- Training set is $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$ with $y^{(i)} \in \{-1, +1\}$
- The classifier shall correctly predict all training samples

$$\mathbf{w} \bullet \mathbf{x}^{(i)} + b \geq 0 \quad \text{if } y^{(i)} = +1$$

$$\mathbf{w} \bullet \mathbf{x}^{(i)} + b \leq 0 \quad \text{if } y^{(i)} = -1$$

- Remember that inserting the point coordinates gives the signed distance
- For one class the distance is positive for the other class the distance is negative
- This means that each class is in a half-plane separated by $\mathbf{w} \bullet \mathbf{x} + b = 0$

Abbildung 10.12: Label Encoding

- We want a large margin classifier with margin $M > 0$

$$\mathbf{w} \bullet \mathbf{x}^{(i)} + b \geq M \quad \text{if } y^{(i)} = +1$$

$$\mathbf{w} \bullet \mathbf{x}^{(i)} + b \leq -M \quad \text{if } y^{(i)} = -1$$

Abbildung 10.13: Introduction the Margin

10.4.1 Controlling the Margin

Die rote Linie auf der Abbildung 10.14 wird mit einer Konstante multipliziert. Damit vergrößert oder verkleinert sich der Margin. Mit dieser Skalierung von \vec{w} kann die Margin kontrolliert werden. Die Bedingung bleibt immer gleich, nämlich $+ - 1$, jedoch verändern wir so die «Einheit» des Abstandes (mm, cm, m, km)

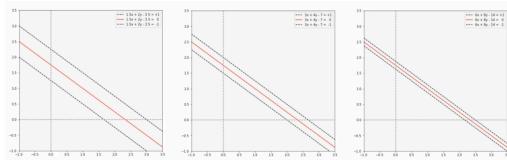


Abbildung 10.14: Controlling the Margin

10.4.2 Elegant Problem Formulation

Weil wir im obigen Fall eine Fallunterscheidung haben (if/else), ergäbe sich eine umständliche Formulierung für die Optimierung. Weil wir die Labels +1 und -1 verwendete, können wir die Labels in die Formel reinmultiplizieren und erreichen damit eine allgemein gültige Bedingung (Constraint).

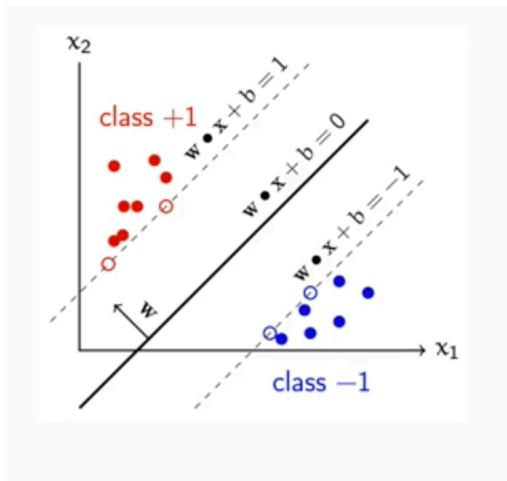


Abbildung 10.15: Elegant Problem Formulation

10.4.3 How Big is the Margin

Die Distanz können wir ja aus $-b$ aus der hessischen Normalform ablesen. Wenn wir die Gleichung aber skalieren entspricht sie nicht mehr der Hessischen Normalenform. Um dies zu umgehen können den Term normieren. Da wir zwei Distanzen in beide Richtungen haben ergibt sich die gesamte Margin aus

$$-(b-1)/\|\vec{w}\| + (b+1)/\|\vec{w}\| = \frac{2}{\|\vec{w}\|}$$

10.4.4 Primal Optimization Problem

Forderungen

1. Datenpunkte müssen auf richter Seite klassifiziert sein
2. Margin muss maximiert sein

Anstatt $\frac{2}{\|w\|}$ zu maximieren, können wir auch einfach $\|w\|$ minimieren, oder noch besser $\frac{1}{2} * \|w\|^2 = \frac{1}{2} w \cdot w$. Es ist eine Minimierung unter Constraints, welches ein quadratisches Optimierungsproblem ist.

Diese Variante ist anfällig auf Overfitting, weil eine Linie gefunden werden muss.

Entspricht einem *Hard Margin Classifier* mit welchem ein Binäres Klassifizierungsproblem gelöst werden kann.

$$\underset{\mathbf{w}, b}{\text{minimize}} \frac{1}{2} \mathbf{w} \cdot \mathbf{w} \quad \text{subject to} \quad y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)} + b) - 1 \geq 0 \quad \text{for } i = 1, \dots, m$$

Does not allow classification errors in the training set

Abbildung 10.16: Hard Margin Classifier

10.5 Soft Margin Classifier

Das Problem der Hard Margin Classifier ist, dass jeder Punkt klar einer Klasse zugeordnet werden kann. Das Modell wird overfitted. Der Soft Margin Classifier erlaubt eine weichere Linie, wo auch «Übertretungen» erlaubt sind.

10.5.1 Outlier Sensitivity

Hard Margin Classifiers sind sehr sensiv gegen Outliers. In der Abbildung 10.17 sehen wir, dass der graue neue Punkt eher zur Mehrheit der roten Punkte passen würde. Aber wegen der harten Grenze zu blau zugeordnet wird. Das Modell ist also overfittet und generalisiert zu wenig auf unseen Data.



Abbildung 10.17: Outlier Sensitivity

10.5.2 Soft Margin Classifier

Ein *Soft Margin Classifier* erlaubt die Missklassifizierung von Trainingsdaten. Es ist ein Trade-off zwischen besserer Generalisierung auf *unseen* Data, entgegen der Klassifizierungsdaten zu Trainingsdaten. Es muss die Überlegung gemacht werden, noch mehr Punkte zu ignorieren. Dazu Cross-Validation nutzen!

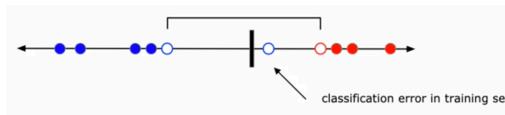


Abbildung 10.18: Soft Margin Classifier

10.5.3 Slack Variables

Jeder Datenpunkt erhält eine Schlafvariable ϵ , welche ein Mass ist, wie viel der Punkt von der Margin abweichen darf. Der Hyper-Parameter C in 10.19 kann ich steuern, wie ob ein eher engerer Margin schlimmer ist oder mehr Verletzungen in den Trainingsdaten erlaubt sind. Evaluation von C via Cross-Validation.

- Hard margin SVM optimization problem:

$$\underset{\mathbf{w}, b}{\text{minimize}} \frac{1}{2} \mathbf{w} \bullet \mathbf{w} \quad \text{subject to } y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b) - 1 \geq 0$$

- Soft margin SVM optimization problem:

$$\underset{\mathbf{w}, b}{\text{minimize}} \frac{1}{2} \mathbf{w} \bullet \mathbf{w} + C \sum_{i=1}^n \xi_i \quad \text{subject to } y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b) \geq 1 - \xi_i$$

minimize the total violation how much each point violates the margin

$i = 1, \dots, m$ with $\xi_i \geq 0$

In German, we use the funny word Schlußvariable for slack variable

8

Abbildung 10.19: Slack Variables

10.5.4 Regularization Parameter

Mit Hilfe vom C , siehe ref{slackvariable} können wir overfitting entgegenwirken.

- je kleiner C , desto weicher, eher dürfen Punkte den Margin überschreiten → führt zu grosser Margin
- je grösser C , desto harter ist die Verletzung → führt zu kleiner Margin
- mit $C = \infty$ werden keine Verletzungen akzeptiert und der Soft wird wieder zu Hard Margin Classifier

Es ist immer noch ein quadratisches Optimierungsproblem mit eindeutigen Minimum. Es ist ein Trade-Off zwischen Trainingsfehler und Margin, kontrollierbar via Hyper-Parameter C . Bester Trade-Off via Cross-Validation für C finden.

Unconstrained Optimization Problem

$$\begin{aligned} y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b) &\geq 1 - \xi_i \\ 1 - y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b) &\leq \xi_i \end{aligned}$$

Because $\xi_i \geq 0$ it follows that

$$\xi_i = \max(0, 1 - y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b))$$

This leads to the following optimization problem for soft margin SVM

$$\underset{\mathbf{w}, b}{\text{minimize}} \frac{1}{2} \mathbf{w} \bullet \mathbf{w} + C \cdot \sum_{i=1}^n \max(0, 1 - y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b))$$

Abbildung 10.20: Unconstrained Optimization Problem

10.6 Kernel

Einer der coolsten Tricks in ML und erlauben komplexere Entscheidungsgrenzen. Transformieren Daten in Höheren Dimensionalen Raum.

10.6.1 Not Linearly Separable

Wenn Daten nicht klar in zwei Gruppen unterteilt werden kann (Klassen sind kreisförmig) können Daten in höheren dimensionalen Raum projiziert werden. Durch quadrieren können Daten projiziert werden. Das ist die Idee des Kernels.

10.6.2 What is a Kernel

Berechnet Similaritätswert für zwei Datenpunkte, die in einen höher dimensional Raum projiziert wurden. Die definieren *implizit* ein Mapping in den hochdimensionalen Raum. Die Projektion wird nur hypothetisch gemacht. Bzw. eben einfach den Wert zurückgegeben ohne die Projektion effektiv auszuführen. Häufig verwendet:

- Lineare Kernels
- Polynomial Kernels
- RBF Kernels
- Sigmoid Kernels

10.6.2.1 Lineare Kernel

Spezialform des Polynomial Kernels, nämlich gewöhnliches Skalarprodukt

10.6.2.2 Polynomial Kernel

Dieser Kernel berechnet aus zwei Datenpunkten (Vektoren) aus dem Tiefdimensionalen Raum das Skalarprodukt, addiert Konstante r und rechnet das Ganze hoch d

$$(\vec{x} \cdot \vec{y} + r)^d$$

- r = Koeffizient
- d = Grad des Polynoms

Take 1D data such that $\mathbf{x} = [x]$ and $\mathbf{y} = [y]$ with $d = 2$ and $r = 0.5$

$$\left(xy + \frac{1}{2}\right)^2 = \left(xy + \frac{1}{2}\right) \cdot \left(xy + \frac{1}{2}\right) = xy + x^2y^2 + \frac{1}{4} = \left(x, x^2, \frac{1}{2}\right) \bullet \left(y, y^2, \frac{1}{2}\right)$$

calculation with 1D data
2 multiplications, 1 addition

calculation with 3D data
3 multiplications, 2 additions

Abbildung 10.21: Example Polynomial Kernel

Take 1D data such that $\mathbf{x} = [x]$ and $\mathbf{y} = [y]$ with $d = 2$ and $r = 1$

$$(xy + 1)^2 = 2xy + x^2y^2 + 1 = (\sqrt{2}x, x^2, 1) \bullet (\sqrt{2}y, y^2, 1)$$

- X-coordinate $\rightarrow 1.41x$ / Y-coordinate $\rightarrow x^2$ / Z-coordinate \rightarrow constant
- Here, the X-coordinate is scaled by a factor 1.41

Abbildung 10.22: 2. Example Polynomial Kernel

Take 2D data such that $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$ with $d = 2$ and $r = 1$

$$\begin{aligned} ([x_1, x_2] \bullet [y_1, y_2] + 1)^2 &= (x_1y_1 + x_2y_2 + 1)^2 \\ &= 2x_1y_1 + 2x_2y_2 + 2x_1y_1x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 1 \\ &= (\sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2, 1) \bullet \\ &\quad (\sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1y_2, y_1^2, y_2^2, 1) \end{aligned}$$

- Every 2D point is projected to 6D by this kernel
- Count the operations required for calculating in low and high dimensionality
- The more features, the more dimensions

Abbildung 10.23: 3. Example Polynomial Kernel

Output des Kernel ist das Skalarprodukt der beiden Koordinaten

10.6.2.3 RBF Kernel

Radial Basis Function (RBF) or Gaussian Kernel. Entspricht der Projektion in eine unendlichen dimensionalen Raum. Verhält sich ähnlich wie ein gewichteter Nearest Neighbor Model. Je näher die Distanz zu einem Trainingspunkt, desto mehr Einfluss auf die Klassifizierung.

$$\exp(-\gamma * \|\vec{x} - \vec{y}\|^2)$$

Das Gamma entspricht einem exponiellen Decay (Verfall) der skalierten Euclidean Distance or L^2 -Norm zwischen \vec{x} und \vec{y}

Take 1D data such that $\mathbf{x} = [x]$ and $\mathbf{y} = [y]$ with $y = 0.5$.

$$\begin{aligned} e^{-\frac{1}{2}(x-y)^2} &= e^{-\frac{1}{2}(x^2-2xy+y^2)} = e^{-\frac{1}{2}(x^2+y^2)} \cdot e^{xy} \\ &= f(x, y) \cdot e^{xy} \quad \text{with } f(x, y) = e^{-\frac{1}{2}(x^2+y^2)} \end{aligned}$$

We consider the Taylor expansion of the exponential function

$$\begin{aligned} f(x, y) \cdot e^{xy} &= f(x, y) \cdot \left[1 + \frac{1}{1!}xy + \frac{1}{2!}x^2y^2 + \frac{1}{3!}x^3y^3 + \frac{1}{4!}x^4y^4 + \dots \right] \\ &= f(x, y) \cdot \left[\left(1, \sqrt{\frac{1}{1!}x}, \sqrt{\frac{1}{2!}x^2}, \sqrt{\frac{1}{3!}x^3}, \sqrt{\frac{1}{4!}x^4}, \dots \right) \cdot \right. \\ &\quad \left. \left(1, \sqrt{\frac{1}{1!}y}, \sqrt{\frac{1}{2!}y^2}, \sqrt{\frac{1}{3!}y^3}, \sqrt{\frac{1}{4!}y^4}, \dots \right) \right] \end{aligned}$$

a scalar product of two vectors
with infinite dimensions

Abbildung 10.24: Example RBF Kernel

f in `ref{rbfkernel}` entspricht der Ableitung. Die Taylorentwicklung ist eine unedliche lange Approximation von e^{xy}

10.7 SVM with Kernel

Obwohl wir es nur für die Hard margin Fall anschauen, funktioniert es sehr ähnlich im soft margin Optimierungsproblem.

Problem ist, den Kernel in Optimierungsformel einzubetten. Eine solche Representation wird mit der *Lagrange Methode*.

10.7.1 Lagrange Methode

Siehe auch IMATH-stuff.

- Lagrange dual form
$$L(x, \alpha_i) = f(x) - \sum_i \alpha_i \cdot g_i(x)$$
- Observe that
$$\max_{\alpha_i \geq 0} (-\alpha_i \cdot g_i(x)) = \begin{cases} 0 & \text{if } g_i(x) \geq 0 \\ \infty & \text{otherwise} \end{cases}$$
- Let f^* be the minimum of $f(x)$ satisfying $g_i(x) \geq 0$, it follows that
$$f^* = \min_x \max_{\alpha_i \geq 0} L(x, \alpha_i)$$

Constraints have disappeared; now we can use standard techniques from calculus ☺

Abbildung 10.25: Lagrange Method

10.7.2 Lagrange Transformation

Wir berechnen die partiellen Ableitungen mit w , b und a_i . Eine Einschränkung ist, dass alle partiellen Ableitungen gleich null sind.

$$\underset{\mathbf{w}, b}{\text{minimize}} \frac{1}{2} \mathbf{w} \bullet \mathbf{w} \quad \text{subject to} \quad y^{(i)} (\mathbf{w} \bullet \mathbf{x}^{(i)} + b) - 1 \geq 0$$

transformed into

$$L(\mathbf{w}, b, \alpha_1, \dots, \alpha_m) = \frac{1}{2} \mathbf{w} \bullet \mathbf{w} - \sum_{i=1}^m \alpha_i [(\mathbf{w} \bullet \mathbf{x}^{(i)} + b) y^{(i)} - 1] \quad \text{with } \alpha_i \geq 0$$

Abbildung 10.26: Lagrange Transformation

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^m \alpha_i y^{(i)} \mathbf{x}^{(i)} = 0 \quad \text{and} \quad \frac{\partial L}{\partial b} = - \sum_{i=1}^m \alpha_i y^{(i)} = 0$$

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y^{(i)} \mathbf{x}^{(i)} \quad \text{and} \quad \sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- If $\mathbf{x}^{(i)}$ is a support vector, we must have $\mathbf{w} \bullet \mathbf{x}^{(i)} + b = y^{(i)}$
- Hence, if \mathbf{w} is known, we obtain b by $b = y^{(i)} - \mathbf{w} \bullet \mathbf{x}^{(i)}$

We insert these necessary conditions into the Lagrange formulation again

Abbildung 10.27: Partial Derivatives of Lagrange Formulation

$$\begin{aligned} L(\mathbf{w}, b, \alpha_1, \dots, \alpha_m) &= \frac{1}{2} \mathbf{w} \bullet \mathbf{w} - \sum_{i=1}^m \alpha_i [(\mathbf{w} \bullet \mathbf{x}^{(i)} + b) y^{(i)} - 1] \\ &= \frac{1}{2} \mathbf{w} \bullet \mathbf{w} - \sum_{i=1}^m \alpha_i y^{(i)} \mathbf{w} \bullet \mathbf{x}^{(i)} - \sum_{i=1}^m \alpha_i y^{(i)} b + \sum_{i=1}^m \alpha_i \\ &= \frac{1}{2} \mathbf{w} \bullet \mathbf{w} - \mathbf{w} \bullet \boxed{\sum_{i=1}^m \alpha_i y^{(i)} \mathbf{x}^{(i)}} - b \boxed{\sum_{i=1}^m \alpha_i y^{(i)}} + \sum_{i=1}^m \alpha_i \end{aligned}$$

We insert the two results from the previous slides (colored boxes)

$$\begin{aligned} L(\mathbf{w}, b, \alpha_1, \dots, \alpha_m) &= \frac{1}{2} \mathbf{w} \bullet \mathbf{w} - \mathbf{w} \bullet \boxed{\mathbf{w}} + \sum_{i=1}^m \alpha_i = \sum_{i=1}^m \alpha_i - \frac{1}{2} \mathbf{w} \bullet \mathbf{w} \\ &= \sum_{i=1}^m \alpha_i - \frac{1}{2} \boxed{\sum_{i=1}^m \alpha_i y^{(i)} \mathbf{x}^{(i)}} \bullet \boxed{\sum_{j=1}^m \alpha_j y^{(j)} \mathbf{x}^{(j)}} \end{aligned}$$

no \mathbf{w} and b anymore!

Abbildung 10.28: Towards the Dual Problem Formulation

This is where we arrived ...

$$L(\alpha_1, \dots, \alpha_m) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \bullet \mathbf{x}^{(j)} \quad \text{with } \alpha_i \geq 0$$

- The solution to this dual problem formulation yields α_i for $i = 1, \dots, m$
- Afterwards we can compute \mathbf{w} and b as suggested on the previous slides
- Note that this now contains a scalar product between training data points
- Remember that the scalar product is the linear kernel

Abbildung 10.29: Dual Problem Formulation

10.7.3 Kernel Hard-Margin SVM

Als Skalarprodukt in 10.29 kann nun ein beliebiger Kernel eingefügt werden. Der Kernel via Hyper-Parameter Optimierung eruiieren. Anstatt das Primal Problem zu minimieren, *maximieren* wird die Lagrange Dual Funktion. Zur Klassifizierungszeit verwenden wir

K entspricht einem Kernel. Das Resultat ist eine Zahl (Similarität im höheren Raum).

$$L(\alpha_1, \dots, \alpha_m) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y^{(i)} y^{(j)} \cdot \mathbf{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \text{ with } \alpha_i \geq 0$$

Abbildung 10.30: Kernel Hard-Margin SVM

Zur Klassifizierungszeit wird der Karsupel nicht mehr benötigt.

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \bullet \mathbf{x} + b) = \text{sign}\left[\sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)} \bullet \mathbf{x} + b\right]$$

Abbildung 10.31: Kernel Hard-Margin SVM at Classification Time

10.7.4 API Check

```
sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0,...)
```

- C is the regularization parameter
- kernel choice (linear, poly, rbf, ...)
- degree is the degree d of polynomial kernel
- gamma is the RBF kernel coefficient
- coef0 is the coefficient r of polynomial kernel

[<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>] [siehe online]

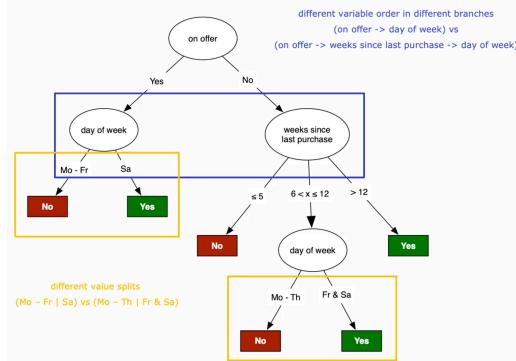
11 Tree Models

Supervised Learning mit Decision-, Regression Trees and Random Forests

11.1 Decision Trees

Entscheidungsbäume sind Bäume die zum Beispiel Kaufanalyse machen und eine Voraussage treffen, ob ein gewisses Produkt am nächsten Samstag gekauft wird oder nicht. Entspricht einem Klassifikationsproblem.

Features können hier sein, Dauer seit dem letzten Kauf des Produktes, gab es Rabatt und Wochentag des Kaufs. Der Algorithmus wählt die Features selber. Gleiche Variablen und Wertesplits siehe ref{decisiontree} können



unterschiedliche Ausprägungen haben.

11.1.1 Which Features and in which order?

Ja-Instanzen sind Ereignisse, wo ein Produkt *gekauft* wurde. Nein-Instanzen sind Ereignisse wo das Produkt *nicht* gekauft wurde. Der Algorithmus muss filtern, welche Features für ein Ja entscheidend sind, da einige *nicht informativ* sind.

Als Beispiel nicht informativ wäre die *Jahreszeit*. Ein informatives könnte *Angebot* sein.

11.1.2 Adding Decisions

Nein-Instanzen können laufend weiter unterteilt werden, falls es Sinn macht.

11.1.3 ID3 - Tree Construction Rules

Einen der ersten Algorithmen um ein Entscheidungsbaum zu konstruieren.

1. Wenn nur *Positive oder Negative* Instanzen übrig sind, stop splitting und Entscheid dem Leaf zuweisen
2. Wenn einige *Positive und Negative* Instanzen übrig sind, ein weiteres Feature wählen und weiteres Child hinzufügen
3. *Spezialfall:* Wenn *keine* Instanzen übrig sind, zum Beispiel wenn diese Kombi im Trainingset nicht vorkommt, dann auf Parent schauen und da Mehrheitswahl treffen

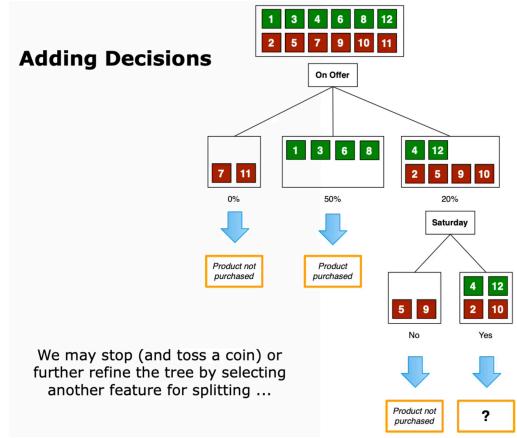


Abbildung 11.1: Adding Decisions

4. *Spezialfall:* Wenn noch Instanzen vorhanden sind, aber *keine* Features mehr, sind die Daten verschmutzt. Das [Data Quality Assessment](#) wurde ungenügend durchgeführt. Auch in diesem Fall wird Mehrheitswahl gemacht

11.1.3.1 History

- ID3 (Iterative Dichotomiser 3); Information Gain wurde erstes Mal für Featurewahl verwendet
- C4.5; nutzt auch Information Gain, integriert numerische Features (z.B. Preis) und partitioniert diesen, beschneidet Tiefe der Bäume und Kostenzuweisung zu Features
- CART (Classification and Regression Trees); Nachfolger von C4.5, kann auch für Regression verwendet werden, nutzt Gini impurity, nutzt Varianzreduzierung für Featurewahl, konstruiert Binary-Trees für jeden möglichen Split

11.1.4 Splitting Criterion

Es gibt also zwei Herangehensweisen um das beste Splitting-Feature zu finden:

1. Anhand des Informationsgehaltes, wähle den, welchen den höchsten Informationsgewinn in Bezug auf die Zielvariable bietet
2. Gini Index oder Impurity (Unreinheit), Instanzen mit Ja und nein Stimmen ist unrein. Der Index misst also die Reinheit eines Knotens

11.2 Gini Impurity

Die Gini Impurity wird für jedes Feature (On Offer in `ref{giniimpurity}`) gemessen. Für jedes Label wird zufällig ein Wert gezogen und die Wahrscheinlichkeit berechnet, welchen Zug man macht. Die Wahrscheinlichkeit einer richtigen Wahl wird mit der W'keit einer falschen Wahl multipliziert. Die W'keit der beiden Labels wird summiert. Die Gini Impurity ergibt sich schlussendlich aus der Summe der Labelverteilung multipliziert mit der Summe der W'keit der Labels.

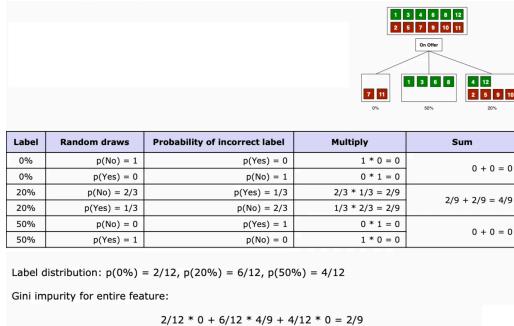


Abbildung 11.2: Gini Impurity

11.2.1 Gini Impurity gets efficient

p_i entspricht einem Random draw.

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

Label	$p(\text{No})$	$p(\text{Yes})$	$1 - p(\text{No})^2 - p(\text{Yes})^2$
0%	1	0	0
20%	2/3	1/3	4/9
50%	0	1	0

Abbildung 11.3: Gini efficient

Die Komplexität der Gini Impurity lässt sich aus der Tabelle auslesen. Pro Feature-Wert gibt es eine Zeile und pro Zielvariablenwert, eine Spalte.

11.2.2 Gini Impurity of continuous Variables

Wenn nummerische Werte vorkommen, kann ja nicht für jede mögliche Zahl eine Impurity berechnet werden (unendliches Vorhaben). Vorgehen um das zu managen:

1. Tabelle nach diesen Werten sortieren.
2. Die Übergänge zwischen zwei Werten werden nun genommen und daraus der Durchschnitt berechnet.
3. Der Durchschnitt der Übergänge wird nun als Splitkriterium verwendet
4. Diese Durchschnitte werden nun als Featurevalues verwendet, um die Gini Impurity zu berechnen

11.2.3 Splitting Feature Selection and Stop Criterion

- je kleiner die Gini Impurity, desto besser
- wenn Purity nicht verbessert werden kann, wird aufhört
- der Split wird anhand des bestens Features gemacht

11.3 Regression Trees

Beim CART Regression Tree hat man Regressionswerte in den Leafs. Um ein Mass zu erhalten wird nun die Varianz dieser Daten gemessen. Je kleiner umso besser, je breiter umso unsicherer ist die Wahl.

Die Varianz ersetzt den Gini Impurity, der Rest ist gleich wie bei [Decision Trees](#)

11.4 Discussion

11.4.1 Advantages of Decision and Regression Trees

Vorteile von Bäume sind und sie können...

- einfach verstanden und interpretiert werden (mehr Vertrauen als in Neuronalen Netzen (weil Blackbox))
- nummerische und kategorische Daten behandeln und Klassifizierung und Regression
- mit beinahe ohne Datenaufbereitung umgehen (keine Normalisierung und Dummy Variablen (kein Vector Space Model))
- gut Performen auch auf grossen Datensets

Nachteile der Bäume sind ...

- nicht so akkurat oder genau
- tendieren auf überspezialisiert und so zu tief werden (overfitting). Es muss Pruning (abscheiden von Knoten) vorgesehen werden
- sehr sensitiv auf Datenqualität - eine Änderung auf Datenpunkt kann Resultat massiv beeinflussen

11.4.2 Scikit Learn API

`sklearn.tree.DecisionTreeClassifier` mit den wichtigen Parameter `criterion='gini'` sowie `max_depth=None`

`sklearn.tree.DecisionTreeRegressor` mit den wichtigen Parameter `criterion='mse'` sowie `max_depth=None`

- `mse` steht für [MSE - Mean Squared Error](#)
- `max_depth` kontrolliert pruning und schützt so vor overfitting

11.5 Random Forests

Wurde bei Kinect (XBox) verwendet. Wieso aber Random Forests? Baummodelle tendieren auf Overfitting und sind sehr sensibel auf Datenqualität (Verschmutzung, Anomalien, etc.).

Ein Wald beinhaltet ganz viele Bäume. Und das wird hier auch so gemacht. Ein Random Forest ist eine Collection von [Decision Trees](#). Diese werden aus dem gleichen Trainingsset trainiert. Allerdings werden zufällig Daten und die Attribute ausgewählt. Man erhält viele unterschiedliche Bäume mit unterschiedlichen Subdatenmengen.

Die Voraussage wird so gemacht, dass ein neuer Datenpunkt allen Bäumen zur Klassifizierung gegeben wird und die Wahl am Schluss wird auf Basis einer Mehrheitswahl aller Subbäumen getroffen.

11.5.1 Building a Random Forest

Training

1. Wähle zufälliges Set D^* aus Trainingsdaten D
2. Wähle zufälliges Set A^* aus Attributen A
3. Erstelle decision oder regression Tree aus D^* und A^*
4. Schritte wiederholen für gewünschte Anzahl Bäumen

Decision Phase

1. Separate Entscheid für jeden Baum erhalten
2. Resultate kombinieren und finales Ergebnis ausspucken (Durchschnitt aus Regression oder Mehrheit bei Klassifikation)

Durch das einfügen des Zufalls wird die Performance besser. Das Risiko von overfitting, sowie auch unsaubere Daten - welche nur noch in einigen Bäume vorhanden sind, wird verteilt.

11.5.2 Scikit Learn API

`sklearn.tree.RandomForestClassifier` mit den wichtigen Parameter `n_estimators=100, criterion='gini'` sowie `max_depth=None`

`sklearn.tree.RandomForestRegressor` mit den wichtigen Parameter `n_estimators=100, criterion='mse'` sowie `max_depth=None`

- `n_estimators` steht für die Anzahl Bäume
- `mse` steht für [MSE - Mean Squared Error](#)
- `max_depth` kontrolliert pruning und schützt so vor overfitting

12 Model Diagnostics

Ziel ist das Model weiter zu optimieren, sobald das Model ausgewählt wurde.

12.1 Model Training

In Supervised Learning trainieren wir normalerweise unser Model mit **Gradient Descent**. Durch den Trainingsstep prüfen wir die Cost Function wie sie sich entwickelt.

Folgende Techniken können genutzt werden, um das Training des Systems zu verbessern.

12.1.1 Training Curves

Die TRainingskurve zeigt die entwicklung der Kostenfunktion über die Zeit. Idealerweise *konvergiert* die Funktion to einem kleinen Wert in nützlicher Zeit. Wir prüfen die *Dynamik* der Kurve um die Trainingsperformance zu verstehen (z.B. linear Regression Trainigs Curves).

12.1.1.1 Training Epochs

Eine *Epoche* beschreibt ein Zyklus, indem wir *alle* Daten verarbeiten.

- Batch: Eine Epoche ist eine Iteration
- Stochastic: Eine Epoche sind N Iterationen



Abbildung 12.1: Different Epochs

12.1.2 Mini-Batch Gradient Descent

Mini-Batches sind kleinere Datenstücke aus dem Datenset. Wir wählen ein B (normalerweise 32) und teilen die Daten durch B . Das gibt die Anzahl Iterationen von B . Mini-Batches geben einen Performanceboost in Batch und Stochastic Training. Die Kurve konvergiert massiv schneller.

Optimization Method	Samples in each Gradient Calc	Weight updates per epoch
---------------------	-------------------------------	--------------------------

12.1.2.1 Epochs vs. Batches

Eine Epoche ist normalerweise einen Loop über das *gesamte* Datenset. Ein Batch oder Minibatch sind gleichgrosse Subsets aus dem Datenset wo der Gradient berechnet und die Gewichte aktualisiert werden.

Optimization Method	Samples in each Gradient Calc	Weight updates per epoch
Batch Gradient Descent	das gesamte Datenset	1
Mini-Batch Gradient Descent	fortlaufende Subset aus dem Datenset	$\frac{n}{\text{size of minibatch}}$
Stochastic Gradient Descent	jedes Sample aus Datenset	n

12.1.3 Step-Size Tuning

In [Gradient Descent](#) können wir mit Hilfe von α die Trainings beeinflussen. Je kleiner α umso länger dauert das Training, aber man ist auch genauer im Minimum. Falls zu gross, verpasst man es.

12.2 Model Validation: Learning Curve Diagnostics

Validation wird gemacht, sobald das Training gemacht fertig ist. Es wird mit *unseen* Daten gemacht. Dazu haben wir vor dem Training ein Validationset zurückgelegt. Aber was kann man noch versuchen, wenn die Validation schlechte Resultat liefert?

Es gibt viele Möglichkeiten das Training zu wiederholen. Man sollte nicht wahllos andere Parameter oder Models oder was auch immer verwenden sondern systematisch vorgehen. Normalerweise ist das Model underfittet (hohen Bias) oder overfittet (hohe Varianz). Mit welcher Methode können wir herausfinden was die Ursache ist?

- High Bias: das Model ist zu wenig komplex, schlechte Performance
- High Variance: perfekt gut auf Trainingsdaten, aber generalisiert nicht auf Validation-Daten. Eine kleine Anpassung in den Trainingsdaten hat grossen Einfluss auf das Modell.

12.2.1 Learning Curves

Learning Curves prüfen den Effekt von mehr Daten die genutzt werden könnten. Sie können genutzt werden, um under- & overfitting zu diagnostizieren.

12.2.1.1 Learning Curves: Training Set

Im Training, reduzieren mehr Daten die Performance, weil es schwieriger wird die Daten zu fitten.

12.2.1.2 Learning Curves: Validation Set

In der Validierung, erhöhen mehr Daten die Performance. Mehr Daten erfassen die Verteilung der Daten besser.

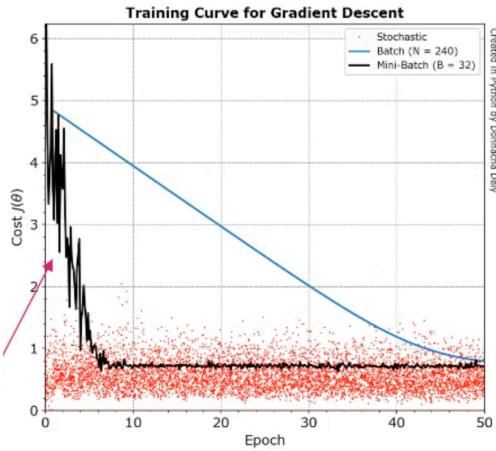


Abbildung 12.2: Mini-Batch Gradient Descent

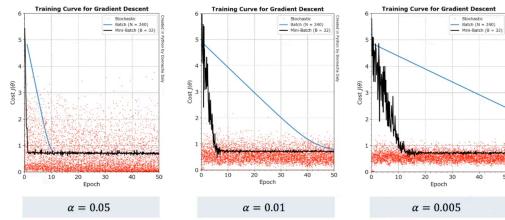


Abbildung 12.3: Step-Size Tuning Gradient Descent

12.2.2 Diagnosing Under-Fitting (High Bias)

Das Modell ist zu simple und performt schlecht. Mehr Daten helfen hier nicht. Komplexeres Modell auswählen oder mehr Features generieren.

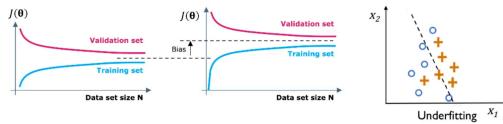


Abbildung 12.4: Diagnosing Under-Fitting

12.2.3 Diagnosing Over-Fitting (High Variance)

Kommt oft vor und schwierig zu beheben. Mehr Daten und saubere Daten können helfen. Evtl. ein kleineres Set der Features versuchen oder [Regularization](#) anwenden.

12.3 Model Tuning: Regularization and Other Tricks

Wir hoffen, dass unseres Modell gut generalisiert - auch auf hoch-Dimensionalen Daten. Je höher die Dimension der Training-Features, umso grösser ist aber die W'keit, dass wir overfitten.

Dies wird *The Curse of Dimensionality* genannt.

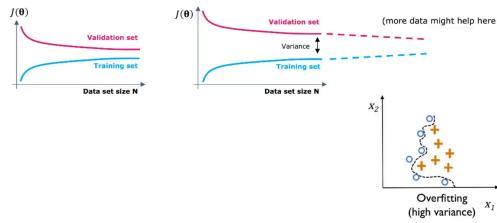


Abbildung 12.5: Diagnosing Under-Fitting

12.3.0.1 Regularization

Mit Hilfe der Regularisierung können wir grosse Parameter bestrafen, um eine bessere Generalisierung erreichen. Dieser Effekt beeinflusst die Kostenfunktion so, dass die wichtigen Features das Training vorwärts treiben (nicht die unwichtigen).

12.3.0.2 Cross-Validation for Regularization Tuning

weil λ ein Hyperparameter ist, können wir Cross-Validation machen.

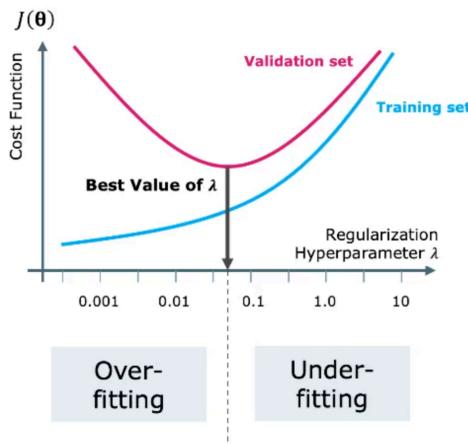


Abbildung 12.6: Cross-Validation for Regularization Tuning

Weitere Tricks können sein:

- Tuning step-Size oder Batch-size
- Tuning der Hyperparameter
- Validation Daten zur Diagnose nutzen

12.4 Ethical Considerations

Nur weil wir es könnten, heißt nicht dass man es machen soll.

12.4.1 Ethical Questions

Bevor man einen Auftrag ausführt sollte man sich folgende Fragen stellen:

- Wozu wird die Applikation eingesetzt?
- Sind die Daten parteiisch
- welche Partei hat die Daten zur Verfügung gestellt und wurden sie vergütet
- Wissen die Personen, um welche Daten es sich handelt, dass ihre Daten verwendet werden?
- Hat man das Recht/Zustimmung?

Je mehr Personen von diesem Algorithmus betroffen sind umso wichtiger werden diese Fragen.

12.4.2 Machine Learning SWOT

Viele AI/ML Risiken haben Etnische Komponenten.



Abbildung 12.7: Machine Learning SWOT

12.4.3 Bias in the Data

Größtes Problem, die Realität in der ganzen Welt wird oft nicht in den Daten widerspiegelt (Hauptfarbe, Traditionen (Hochzeitskleider)).

12.4.4 Ethics Guide for Business

- Kontrollinstanzen sind essentiell
- Immer "Dual-Use" Szenarios betrachten (gute Idee missbrauchen)
- Audit log implementieren um Sichtbar zu werden
- Erklärbarkeit garantieren
- Rückmeldungen von Personen mit ethnischen Background einholen

12.4.5 ML Workflow

1. Data Quality Assessment (DQA) is always worth the time
2. Ethics is not an after-thought! Address ethical issues from the beginning
3. Split your data into Train- Validate- Test- sets: Lock away your test set!
4. Normalize your data, Visualise your data, Understand your data
5. Define the performance metrics you want to use for your application
6. Implement a quick- and dirty baseline classifier or regressor
7. Examine your training curves and fine-tune your training
8. Use your validation set for model diagnostics with learning curves
9. Learn and improve e.g. hyperparameter tuning, cross validation
10. Only use your test data once at the end!

13 Neural Networks

Eine der wichtigsten Methoden um ML zu betreiben.

13.1 The Perceptron - An Artificial Neuron

Ein Neuron ist eine Zelle, die elektrisch aktiviert werden kann und mit anderen über *Synapsen* kommunizieren kann. Ein Neuron besteht aus dem Zellkörper (Soma), *Dendriten* (Rezeptoren) und einem einzigen *Axon* (haben Synapsen). Die Dendriten und das Axon zweigen vom Körper ab.

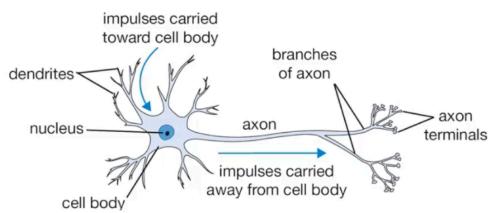


Abbildung 13.1: The Biological Neuron

Neuronen erhalten ein Inputsignal via die Dendriten und senden ein Output-Signal an das Axon. Über Axon-Terminals kann das Signal via die Synapsen an andere weiter geleitet werden. Der Signalprozess ist teilweise *elektrisch*, teilweise *chemisch*. Wenn die Spannung in einem kleinen Intervall stark ändert, generiert das Neuron einen *All-or-Nothing* elektro-chemischen Impuls welcher *Action Potential* genannt wird. Dieses Potential travesiert entlang der Axone und *aktiviert* mehr synaptische Verbindungen. Neuronen-Wechsel ist sehr schnell < 1ms. Face Recognition benötigt ungefähr 100 Schritte und das Gehirn kann dies unter 100ms erledigen.

13.1.1 How do we learn?

Wir lernen indem Synapsen gleichzeitig Impulse senden. Die Verbindung zwischen den Synapsen wird dadurch stärker. Wenn das passiert, lernt man.

Synapses that fire together, wire together

Das Gehirn hat verschiedene Areal, wo unterschiedliche Fähigkeiten gelernt werden.

13.1.2 The Perceptron

Frank Rosenblatt implementierte als erster die Idee ein Model von Neuronen in den Computer zu übertragen - es wurde Perceptron genannt.

13.1.2.1 The Perceptron as a Logic Unit

- Inputs X_j von vorherigen Axone werden an den Synapsen multipliziert mit den Gewichten w_j
- Der Bias b macht extra tuning des Aktivierungslayers möglich
- Die Aktivierung eines einfachen Perceptrons ist ein Boolean True wenn ein Threshold erreicht

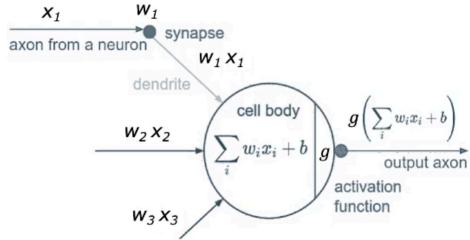


Abbildung 13.2: The Perceptron as a Logic Unit

Die Funktion $g(z)$ wird *Activation Function* genannt. Sie produziert ein *Action Potential* oder *Activation a*. Als Aktivierungsfunktion können z.B. Sigmoid Funktion oder Soft-Max Funktion verwendet werden. Als Aktivierung wird ein *Hard Threshold* verwendet.

Mit der Sigmoid-Funktion erhalten wir eine [Logistic Regression](#) als Aktivierungsfunktion. Damit kann der Perceptron als linearen Klassifier verwendet werden.

13.1.2.2 Logic using a Single Layer Perceptron

Siehe Wahrheitstabelle.

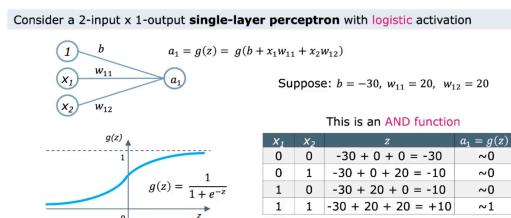


Abbildung 13.3: Logic using a Single Layer Perceptron

13.1.2.3 Linear vs. Non-Linear Models

Eine Single linear Network kann kein XOR implementieren, weil *Non-linear*. Mit einem multi linear wäre es möglich.

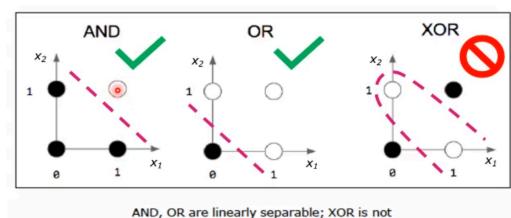


Abbildung 13.4: Linear vs. Non-Linear Models

13.2 Feed Forward Neural Networks

Wenn Daten in einem Netzwerk von links nach rechts fliessen ist es ein *feed-forward network*.

13.2.1 Add a second Neuron

Schreibweise in Vektor bzw. Matrixform.

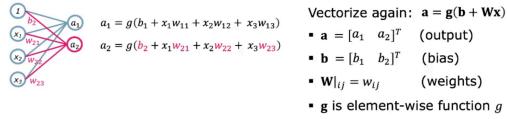


Abbildung 13.5: Add a second Neuron

13.2.2 Add a third Neuron

3 Inputs und drei Outputs!!

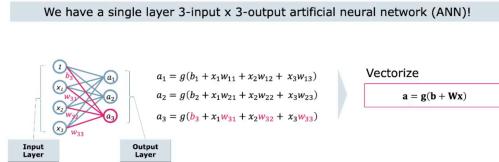


Abbildung 13.6: Add a third Neuron

13.2.3 Single Layer Neural Network

Generalisiert zu einer $M \times N$ und löst bereits jedes linear separierbares Klassifikationsproblem. Falls $g(z)$ eine logistische Funktion ist, kann jeder Output a_j als *One-vs-All Klassifizierung* betrachtet werden. Bedeutet dass a_1 aus Klasse 1 oder nicht, a_2 aus Klasse 2 oder nicht, ...

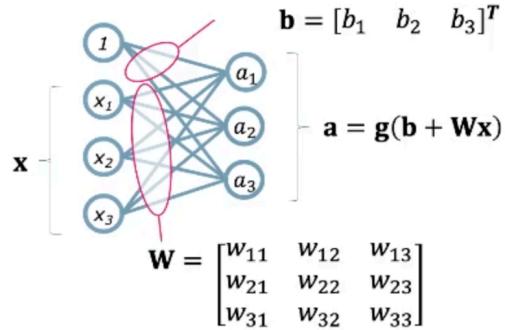


Abbildung 13.7: Add a third Neuron

13.2.4 Introducing a Hidden Layer

Einem Netzwerk können mehr Neuronen hinzugefügt werden. Bei einem Input-, Hidden und Output-Layer spricht man von einem *two-layer*-Network. Der Input-Layer wird nicht berücksichtigt, weil da keine Gewichte berechnet werden. Die Aussgabe eines Neurons formen den Input des nächsten Neurons - Simuliert das Gehirn.

13.2.4.1 Hidden Layers Create new Features

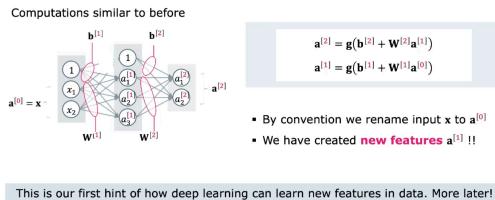


Abbildung 13.8: Hidden Layers Create new Features

13.2.4.2 Universal Approximators

- Es können jede Art von Non-Linear Klassifizierung mit Multilayered Networks gemacht werden
- Hidden Layer lernen neue Features, welche evtl. nicht identifiziert wurden
- Training ist eher langsam, Implementierung aber schnell und einfach
- Multi-Layer ANN sind universale Approximationen
- Führt aber eher zu overfitting
- Als Gegenmittel **Regularization**
- sehr Fehlerresistent und Robust

13.2.5 Deep Learning

Hidden Layers sind die Core-Idee in Deep Learning. Sie lernen ihre Features selber mit den Hidden Layers. Dazu braucht man kein Expertenwissen um Features zu extrahieren, aber viele viele Daten.

13.3 Neural Network Training by Gradient Descent and Back-Propagation

13.3.1 Feed Forward Error Calculation

In Supervised Learning schmeissen wir die gelabelten Daten in das Netzwerk und schauen am Ende was kommt raus. Falls das nicht dem entsprechenden entspricht, werden die Parameter W und b so angepasst, dass das Resultat stimmt.

13.3.2 Choosing a Cost Function

Wir brauchen eine Kostenfunktion, welche die Performance des Models misst. Die ist abhängig vom aktuellen Problem:

- Regression

- Mean Squared Error
- Mean Absolute Error
- Mean Absolute % Error
- Binary Classification
 - Cross Entropy
 - Hinge Loss
 - Squared Hinge Loss
- Multi Classification
 - Multilabel Cross Entropy
 - Kullback Leibler Divergence

13.3.3 Recap - Feed Forward Neural Net

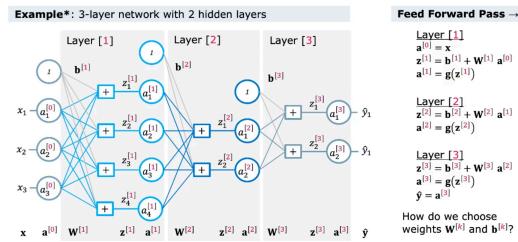


Abbildung 13.9: Feed Forward Neural Net

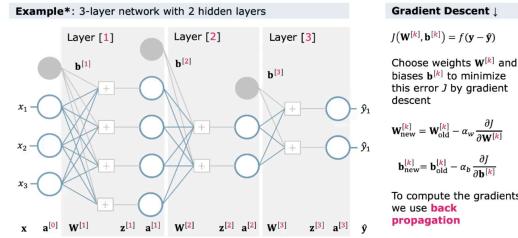


Abbildung 13.10: Labeled Data to drive Gradient Descent

13.3.4 Back Propagation

Back Propagation mit Hilfe der Kettenregel $\delta^{[3]}$.

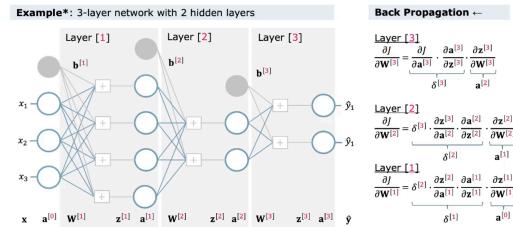


Abbildung 13.11: Back Propagation

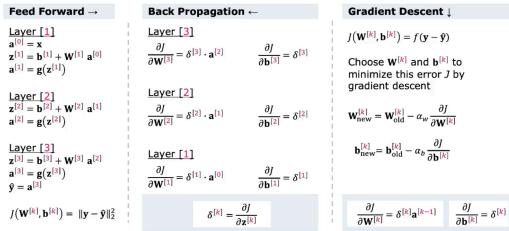


Abbildung 13.12: NN all together

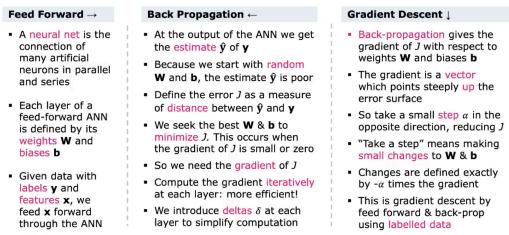


Abbildung 13.13: NN all together in Words

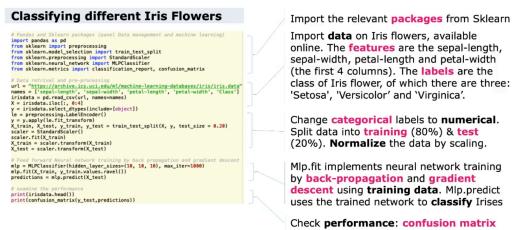


Abbildung 13.14: Training a Feed Forward Neural Network

13.3.5 Training a Feed Forward Neural Network

13.4 Activation Functions and the Soft-Max Classifier

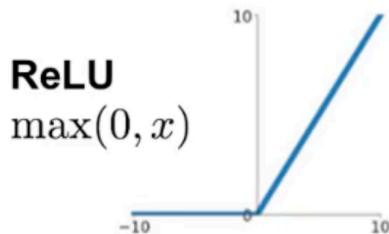
Um den Output zu erhalten benötigen wir eine Aktivierungsfunktion.

13.4.1 Choosing an Activation Function

Normalerweise möchten wir als Output eine W'keit ($0 < \alpha < 1$). Weil wir Gradient Descent für das Training verwenden sollte die Aktivierungsfunktion *stetig* und *differierbar* sein. Wenn die Funktion auf flach ist, verschwinden die Gradienten. Dies verlangsamt das Training markant und sollte vermieden werden. Wenn die Funktion *monotonic*, die Fehlerebene in einem single-layer model ist *konvex*. Eine geeignete ist die Sigmoid, weitere werden nachfolgend behandelt.

13.4.1.1 The Rectified Linear Unit (ReLU)

Ist stückweise linear und wird oft bei Hidden Layers verwendet. Sofern Wert grösser 0, wird sofort zurückgegeben, sonst 0. Vermindert dass Gradienten verschwinden. Standard Activation Function für viele Typen von Neural Networks. Modelle mit ReLU sind einfach zu trainieren und erzielen oft bessere Resultate.



```
1 if input > 0:  
2     return input  
3 else:  
4     return 0
```

Abbildung 13.15: The Rectified Linear Unit (ReLU)

Allerdings führen sie zum *dying units* Problem, weil der Gradient für negative Werte immer null ist und nicht mehr aktiviert wird.

13.4.1.2 The Leaky ReLU

Wird, oder Varianten davon, von den meisten DL Models genutzt, weil verschwinden der Gradienten nicht auftritt. Führt *non-zero activation* für negative Werte ein. Verhindert Auslöschung des Gradienten und *dying unit* Problem. Gute Performance, aber Resultate sind nicht immer konsistent.

Sind aber generell eine gute Wahl für Hidden Layer Activation.

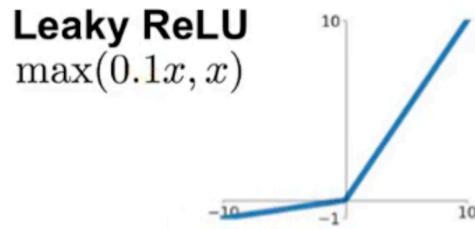


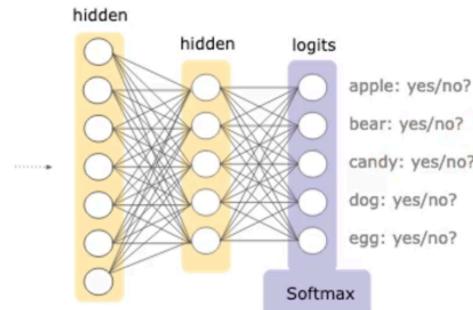
Abbildung 13.16: The Leaky ReLU

13.4.1.3 Soft-Max for Multi-Label Classification

Wird für Output bei multi-labels verwendet. Damit kann W'keit bestimmt werden über alle Inputwerte, egal wie die Aussehen (positiv, negativ, grösser 1, usw.). Softmax modifiziert ein Vektor von n reellen Werten so dass die Summe 1 gibt.

Die Softmax-Funktion ist eine Generalisierung der Logistic Function für mehrere Dimensionen und wird auch in *multinomial* Logistic Regression verwendet. Wird normalerweise bei der letzten Aktivierungsfunktion eines NN verwendet. Die Inputs werden *logits* genannt. Die Funktion *normalisiert* die Outputs eines Netzwerks in eine Wahrscheinlichkeitsverteilung.

We can use a **Soft-Max** activation
at the output layer



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

Abbildung 13.17: Soft-Max for Multi-Label Classification

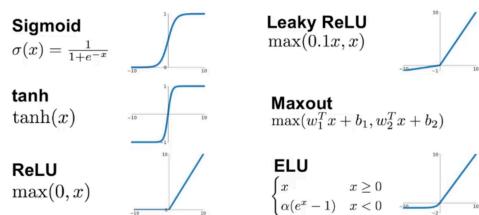


Abbildung 13.18: Summary of Activation Functions

14 Back Propagation

Wird genutzt um Neural Networks zu trainieren und beschleunigt die Berechnung der Gradient Descent-Algorithmus.

14.1 Motivation

Back Propagation ist sehr wichtig in ML. Viele Modelle (NN, CNN, RNN) nutzen BP für das Training - überall wo es Mehrere Layers gibt. Sie steigern die Berechnung der Gradientengleichung.

BP ist aber schwierig. Ein einzelner Layer ist eher einfach, das grosse ganze macht es Komplex.

BP ist built-in in den open-source ML-Frameworks.

14.2 The Chain Rule

Methode um die Ableitung einer zusammengesetzten Funktion zu finden. Der Output der einen Funtion ist der Input der nächsten.

14.2.1 The Logistic Function

Recall: Die Logistische Funktion ist eine *Sigmoid*, welche wir auf Daten einer logistic Regression fitten, um binary Classification zu betreiben.

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

Die Ableitung der Sigmoid ist sehr einfach zu berechnen, weshalb sie auch oft verwendet wird:

$$\frac{dg(z)}{dz} = g(z)(1 - g(z))$$

14.2.2 The Multivariate Chain Rule

Bei einer Mehrvariablen Funktion $f(x, y)$ kann die partielle Chain-Rule angewendet werden.

$$\frac{df}{dt} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

14.3 Gradient Descent again

Die Abbildung 14.1 zeigt einem Drei Layer Neural Network mit einem Hidden-Layer. Dieser wird dann mit gelabelten Daten mit Hilfe von Gradient Descent trainiert.

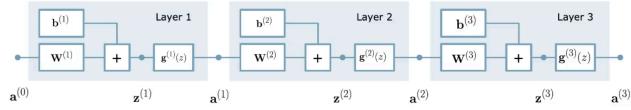


Abbildung 14.1: Three Layer Neural Network

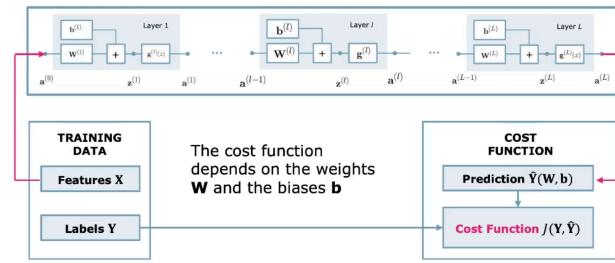


Abbildung 14.2: L -Layer Neural Network

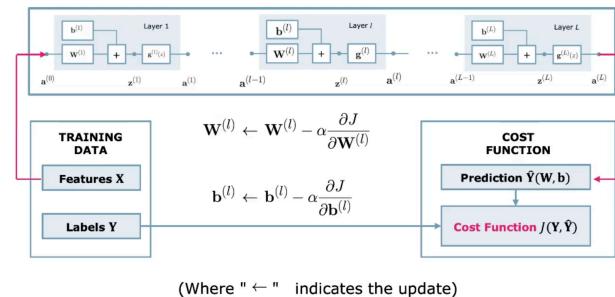


Abbildung 14.3: Gradient Descent Training

14.4 Back Propagation

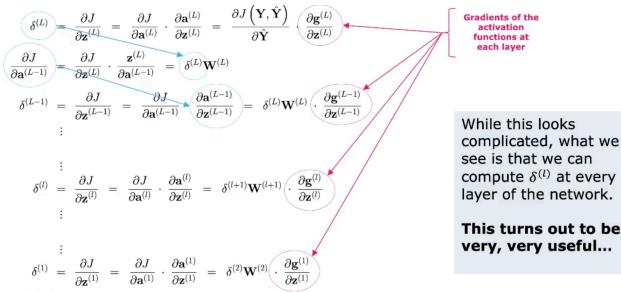


Abbildung 14.4: Back Propagation Derivation

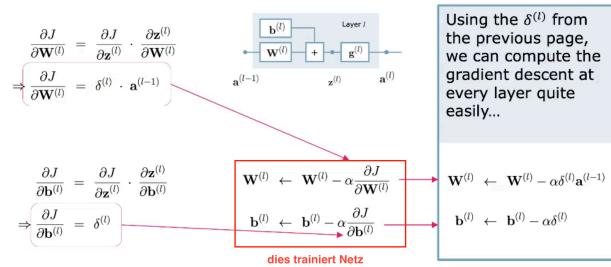


Abbildung 14.5: Revisiting Gradient Descent

14.4.1 Familiar Cost & Activation Functions

Damit wir mit Back Propagation starten können benötigen wird die Kostenfunktion J und die Aktivierungsfunktion g . Daraus erhalten wir ihre Gradienten wodurch wir die Kostenfunktion optimieren können. Dazu benötigen wir Funktionen die differenzierbar sind z.B. MSE oder Logistic Function (sigmoid).

14.4.2 The Multi-Variate Chain Rule

Weil die Kostenfunktion J auf allen Werten von z in jedem Level basiert, muss die Partielle Kettenregel verwendet werden.

We have specified the weight updates using vector calculus based gradient descent.

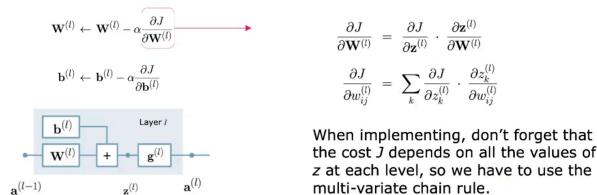


Abbildung 14.6: The Multi-Variate Chain Rule

15 Convolutional Neural Networks

Wie neuronale Netze Bildinformationen verarbeiten.

15.1 ImageNet Challenge

öffentliche Bildsammlung mit annotierten Bildern. 22k Kategorien. Zur Klassifizierung wird Error-Rate verwendet. Top-5 bedeutet, Fehlerrate bei einem Bild min. eines von 5 Labels entdecken (Bild mit Auto und Ampel).

15.2 What an Image really is

Bild in drei Kanälen (Matrizen) RGB, mit Zahlen welche die Farbintensität repräsentiert.

15.3 The Naive Approach

Pixelwerte direkt ins Netz ein neuronales Netz eingeben? Unmöglich, weil bereits ein kleines Bild sehr viele Werte hätte (240x240x3). Dazu wären die gleiche Zahl Neuronen nötig. Ein typischer Hidden Layer hat 1024 Neuronen. Man bräuchte mehrere 100 Mio. Gewichte um ein sehr kleiner Bild zu verarbeiten.

15.3.1 Tackled the Problem in the Old Times

Durch Feature Engineering konnte Anzahl Features reduziert werden. Durch Kantenextraktion konnten die Features gewählt werden. Die Performance ist aber nur so gut, wie die von Menschen gewählten Filter sind.

15.3.2 Invariance to Position, Scaling, Rotation

Bilder können skaliert, gedreht, usw. werden. Ein Klassifier muss also das selbe Resultat erzielen ob das Bild Original entspricht oder skaliert wurde.

15.3.2.1 The MNIST Dataset

Berühmtes Set für die Erkennung der Handschrift.

15.4 Convolutions & Pooling

Werden für Klassifizierung verwendet

15.4.1 Filter Matrices

Pixelwerte sind im Zusammenhang mit ihren Nachbarn am informativesten. Mit einem filter, werden diese also zusammen verarbeitet. Mathematisch benötigt diese viele Operationen, weil die Pixel- und Matrixwerte elementweise multipliziert und addiert werden. Manchmal wird zusätzlich eine Aktivierungsfunktion genutzt.

15.4.1.1 Images and Filters have different Size

Der Filter wird nun von links nach rechts über das Bild geschoben. Die Versatz der Verschiebung wird *stride* genannt (wenn grösser, wird Bild kleiner). Das Bild wird aber kleiner. Um das zu vermeiden, muss *gepaddet* werden.

15.4.1.2 Effect of Convolutions

Mit Filter kann man Ecken detektieren (z.B. Sobel Filter). Die blenden gewisse Eigenschaften ein, bzw. aus.

15.4.2 Convolutional Layers

Ein Conv-Layer wendet viele Filter parallel an. Durch das Anwenden des Filters, schrumpft das Bild ein wenig. Es werden acht Filter angewendet, die alle 3x3 gross sind. Dadurch erhalten wir $3 \times 3 \times 8 = 28$ Gewichte, und das Netz muss die Gewichte lernen. Obwohl viele Inputs und Output-Werte hat es nur wenige Gewichte in diesem System

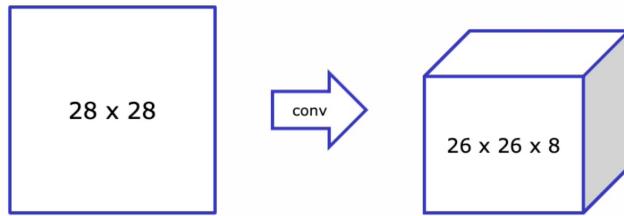


Abbildung 15.1: Transformation to Convolutional Layers

15.4.2.1 Conv as Neural Nets

Filtergrösse von 3 weil jeder Layer input von drei hat. Conv1D(1,3) bedeutet, dass es ein 1D convolutional Layer mit 1 Filter und Filtergrösse 3 ist. Stride ist auch eins, würde separat als «Parameter» angegeben.

Die Filter reduzieren die Anzahl Parameter für die nächste Schicht, weil die Gewichte jeweils gleich sind.

15.4.3 Pooling

Nachbarspixel sind jeweils sehr ähnlich und auch das convolution von Nachbarspixel würde ähnliche Pixelwerte ergeben (hohe Redundanz). Wenn wir Objekte entdecken wollen, müssen wir diese aus der Distanz betrachten - ein Auto können wir nicht anhand wenige Pixel erkennen.

Pooling nimmt nun aus z.B. 3x3 Fläche den max, min, oder avg der Pixelnachbar und wendet diese an. Die Grösse wird über den Pool-Size Parameter bestimmt.

Pooling bewirkt einem Zoom out Effekt.

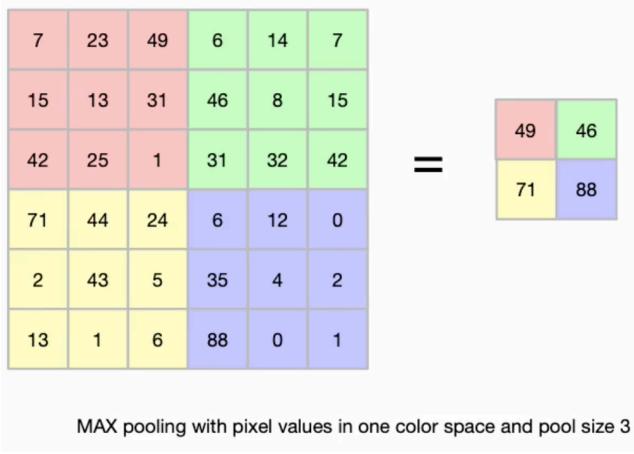


Abbildung 15.2: Pooling

15.4.3.1 Pooling Layers

Pooling dividiert Höhe und Breite durch die Pool Size. Im Bild 15.3 wird die pool size 2 angewendet.

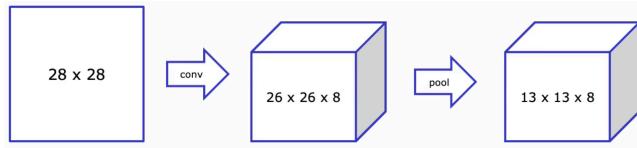


Abbildung 15.3: Pooling Layers

15.5 Model Architectures

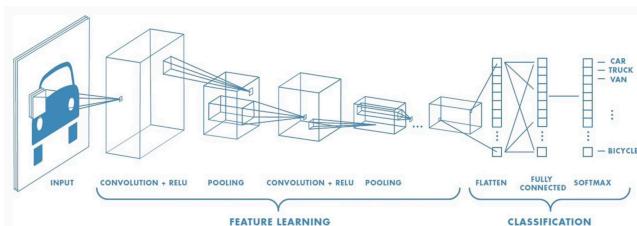


Abbildung 15.4: The Big Picture of CNN

15.5.1 Convolutions on RGB Images

Mit einem 3D-Filterl werden die Farbräume zusammengemergt. Der Filter enthält 27 Werte. Der Filter wird über Bild gelegt und jeder überdeckte Punkt (27 Stk.) wird mit dem Filter multipliziert und danach aufaddiert. Dies ist der neue Wert. Der Filter wird über das ganze Bild verschoben und jeweils neu berechnet.

Das Resultat ist ein 2D mit Werten der Korrelationen.

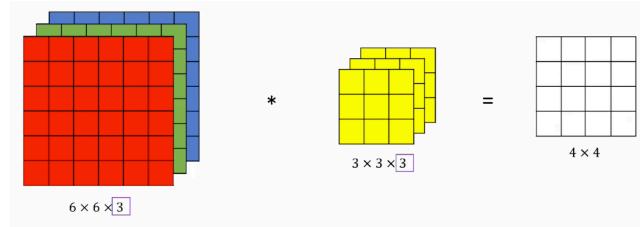


Abbildung 15.5: The Big Picture of CNN

15.5.2 Computer Vision Disciplines

Nicht nur was sondern auch *wo* ist das Objekt! Reihenfolge nach Schwierigkeit links oben, rechts oben, links unten, recht unten.

Bei Segmentierung wird jeder Pixelpunkt zugeordnet.

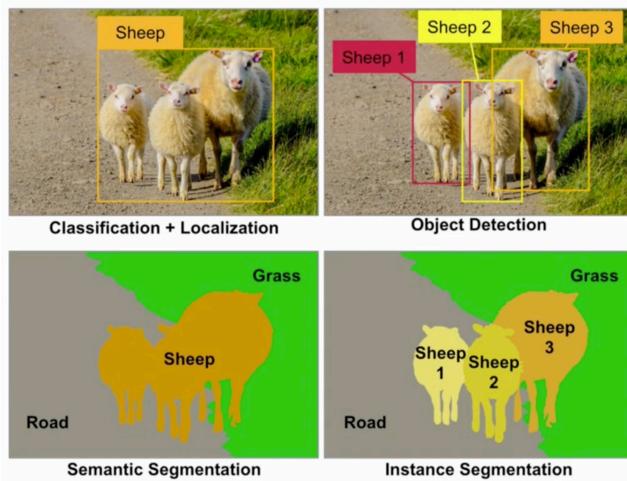


Abbildung 15.6: The Big Picture of CNN

16 Transfer Learning

Filter in Deep CNN lernen in frühen Layers abstrakte Features wie zum Beispiel Kantenfilter, Color Blobs etc. Je weiter der Layer rechts liegt, umso mehr steigt die Komplexität dieses Features. Der finale Layer erlert dann Prototypen Objekte. Ein neuronales Netz sind als Feature Extractors. Dieser Part im Netz wird *Backbone* genannt (alle Hidden ohne Output). Daran kleben wir Klassifikatoren, Lokalisierung oder was auch immer. Diese sind also Task agnostisch (göttlich) und können für alle Arten von CNN in den ersten Layers verwendet werden.

task dedizierter werden die Strukturen je weiter rechts im Netz.

16.1 Transfer Learning

Das ist die Idee von Transfer Learning. Um gute Modelle zu trainieren braucht man ein tiefes Netz. Mit der Anzahl Schichten Menge an Optimierungsparameter und damit der Rechenaufwand. Um ein Model zu trainieren, braucht man also annotierte Daten.

Weil die ersten Phasen relativ generisch sind, können diese in einem *Pre-Training* verwendet werden. Die Domänen spezifischen Bilder werden danach in einem kleineren Umfang trainiert. Der Datensatz bildet genau das ab, was schlussendlich erreicht werden soll. Trainiert wird nur der finale Layer, evtl. ein paar wenige mehr.

16.2 Transfer Learning Approaches

Es gibt zwei Vorgehensweisen. Quelle für Datensets sind Liste auf [Wikipedia](#), oder [datasetsearch](#) von Google.

16.2.1 Libraries of pre-trained Models

Vortrainierte Datenmodelle von unterschiedlichen Architekturen [TensorFlow](#) erhältlich. Für den Zweck des Transfer Learnings verwenden.

16.2.2 Model Repurposing

Strategie 1 um einen Auftrag ausführen zu können:

1. Pre-Trained model für Objekt Detection auswählen
 - Zum Beispiel YOLO (geeignet für Video) trainiert auf COCO-Datenset
2. Den letzten Layer für die Objekt lokalisierung abtrennen
3. Neuen Objekt lokalisierungs-Layer mit zwei Kategorien hinzufügen
4. Nur den letzten Layer mit den domänenspezifischen Datenset trainieren

16.2.2.1 Pros and Cons

- sehr effizient, weil nur ein Layer trainiert wird
- am wenigsten Datenhungrige Technik
- Übernahme der gesamten Backbones aus dem vortrainierten Model
- Das bedeutet, dass diese Layers *frozen* sind

Der Nachteil ist, dass die Backpropagation von den frozen Layers gestoppt wird. Weil diese nicht mit trainiert werden. Man nimmt also an, dass der Backbone gut auf den Zweck passt.

Falls man mit der Performance nicht zufrieden ist, kann man Strategie 2 [Fine-Tuning](#) anwenden.

16.2.3 Fine-Tuning

Dann wird nicht nur der letzte Layer trainiert, sondern einige der letzten, weil diese die komplexe Strukturen lernen. Diese werden also unfreezed und neu trainiert.

Die Gewichte sind dabei nicht zufällig, sondern stammen aus dem Pre-Training. Die Optimierung muss also nicht von ganz vorne neu beginnen, sondern man hat schon einen akzeptablen Stand (Transfer Learning Effekt). Das Training wird aber Ressourcen- und Datenhungriger.

Jeweils Performance messen.

Das Fine-Tuning kann im Extremfall auf alle Backbone-Layers ausgeweitet werden.

16.2.4 Transfer Learning in Industry

Löst Probleme von ungeheuren Kosten und benötigten Daten bei einem Training von Scratch.

16.3 Issues with Supervised Learning

Das Problem ist nicht die Datenmenge, die Daten sind vorhanden. Das Problem ist das annotieren. Dies gilt für

1. labelled image datasets
2. labelled video datasets
3. labelled text datasets
4. labelling quality - auch Ärzte können unterschiedlicher Meinung sein

16.4 Unsupervised Pre-Training

Aktuell grosses Forschungsthemen.

- Pre-Training benötigen immense Datensets
- Daten wären vorhanden, aber das labelling skaliert nicht

16.4.1 Masked Language Modelling

Man nehme eine riesige Menge an unlabeled Sätzen in natürlicher Sprache und zufällig werden Wörter maskiert. Ein pre-trained Model wird verwendet um dieses mit den Sätzen mit den Masken weiter trainiert wird. Der Clou daran ist, dass man weiß was maskiert wurde und damit kann man das Training (Optimierung) gestalten.

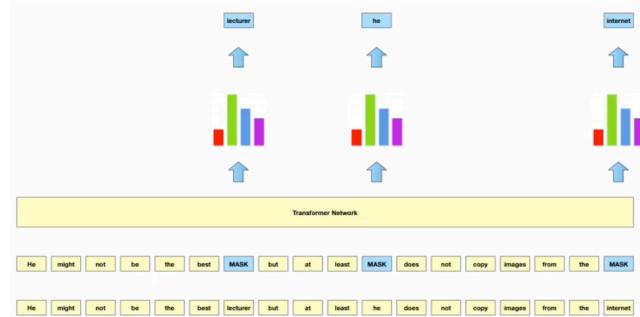


Abbildung 16.1: Masked Language Modelling

16.4.2 Contrastive Learning

Für Bilder oder Videos. Man nehme grossen Satz an unlabeled Bildern aus z.B. Social Media. Man berechnet nicht einen konkreten Wert, sondern vergleicht die beiden Bilder.

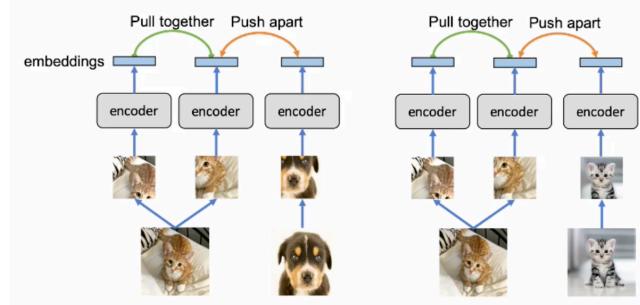


Abbildung 16.2: Masked Language Modelling

Man samplet aus einem Bild zwei *Tiles A* und *B*. Alle *B'* werden in einen Batch gelegt. Wir fordern für das Neuronale Netz eine hohe Similarität zwischen *A* und *B* des gleichen Bildes und eine tiefe bei anderen.

Bei Videos wird es ähnlich gemacht. Das Video wird einfach in zwei gleich lange Teile gesampelt (z.B. je 3s).

Die Bilder können noch transformiert werden (Farbe, Spiegelung, usw.).

17 Recurrent Neural Networks

Mit rekurenten wiederkehrende Netzen für Zeitreihen. Zum Beispiel Sprache.

17.1 Unsupervised NLP

Smeantik, Word Embeddings und Anwendung.

17.1.1 Form Syntax to Semantics

Die Syntax vergleicht Rechtsschreibung von Wörter. Dabei wird die Similarität mit der Levenshtein-Distanz gemessen. Die Levenshtein Distanz vergibt Punkte wie viele Schritte nötig sind, um ein Wort in ein anderes zu transformieren.

Semantik klärt die Bedeutung eines Wortes, zum Beispiel Fahrrad/Velo. ist das Gleiche.

17.1.2 Word Relatedness

Worte haben eine Beziehung wenn sie im gleichen *Dokument* vorkommen. Messung ob zwei Wörter einen Bezug zusammenpassen. Als Beispiel Hund und Leine. Tauchen diese Begriffe im gleichen Dokument auf, werden die gezählt. Die Beziehung von Synonymen im gleichen Dokument ist sehr rar.

Als anderen Ansatz werden Worte als verwandt gewertet, wenn sie das gleiche *Thema* bearbeiten. Hier wird die Wichtigkeit jedes Wortes in Bezug auf jeden Artikel (Wikipedia) gewertet ([Term Frequency-Inverse Document Frequency \(TF-IDF\)](#)). Verwandtschaft von Synonymen ist besser.

17.1.3 Word Similarity

Die TF-IDF ist sehr langsam weil wir einen x-Millionen Dimensionsraum operieren. Mit der Idee der Word Similarity wird der *Kontext* der Wörter verwendet. Dabei werden die Worte vor und nach diese Wort betrachtet und so eingestuft. Der zweier Kontext schaut sich also die beiden Worte vor und nach dem Wort an.

Der Kontext wird zum Training in einem Neuronalen Netzwerk verwendet um fehlende Worte vorauszusagen.

17.1.4 Continuous Bag of Words applied to 3-Grams

Genannt CBoW. Wie in CNN wird Weight-Sharing verwendet, Gewichte von left und right neighbour sind gleich.

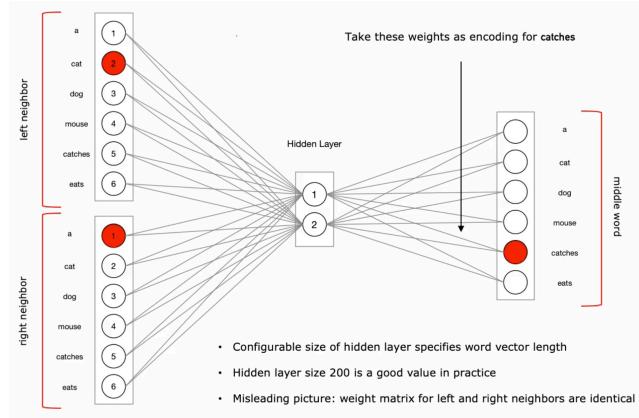


Abbildung 17.1: CBoW

17.1.5 Mathematics with Text - Word embeddings

Man kann semantisches Kalkulationen mit Wortersetzung. Weil es sich um Vektoren handelt kann man damit rechnen. Dies kann folgendes ergeben: Erhalten wird man nicht genau dieses Wort, aber mit einer Umgebungs-suche und Similaritätsberechnung ergeben folgendes

- King - Man + Women = Queen
- Paris - France + Poland = Warschau
- Computer Programmer - Man + Woman = Homemaker (nicht über alles erhaben)

Um das Modell zu trainieren braucht es extrem viele Daten, dass diese nicht von Menschen auf Qualität geprüft werden kann. Deshalb erben wir Fehler.

CBoW hat nur ein Hidden-Layer, das Training ist also nicht sehr Rechenintensiv.

17.2 Supervised NLP

Zeitreihenanalyse auf Textdaten angewandt.

1. Sequence-to-Vector Models

Sentiment Analysis; Input ist eine Sequenz ist fixed-size Vektor -> z.B. positive oder negative Annotation

2. Vector-to-Sequence Models

Image Captioning, Input fixed-size Vector, output eine Sequence (Satz)

3. Sequence-to-Sequence Models

Text summarization, Translations; Input und Output sind Sequenzen

Vanilla NN und CNNs erwarten als In- und Output immer konstante Größen.

17.2.1 Recurrent Neural Networks

RNN lassen sich auf jede Art von timeseries-Daten anwenden. Die Hidden-States (h_n) fliessen in die Berechnung der nachfolgenden Hidden-State ein. Die Outputs von y_4 ist also abhängig von den Vorgänger.

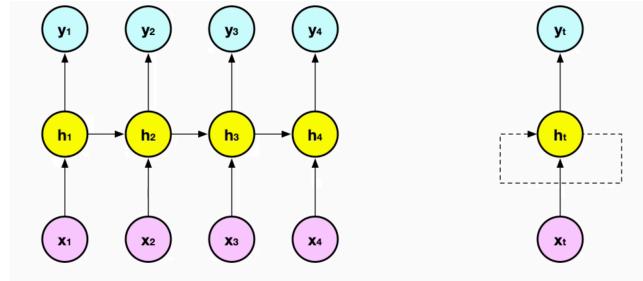


Abbildung 17.2: Recurrent Neural Networks

17.2.2 Deep Inside a Sequence-to-Sequence RNN

RNN verwendet die gleichen Gewichtsmatrizen und Biases für jeden Step.

17.2.2.1 Vanishing Gradient Problems of RNNs

Durch die Rückkoppelung kann der Gradient immer kleiner werden und verschwinden. Es kann aber auch vorkommen, dass der Gradient explodiert. Ein weiteres Problem von RNNs ist das «langzeitgedächtnis» sagt es eine Sequence (Satz) voraus, muss es sich das Subjekt (zB Einzahl/Mehrzahl) am Anfang merken.

17.2.3 Gated Recurrent Units

Verhindern vanishing Problem von Gradient. Berechnen nicht den Kandidaten h_t , sondern *wie viel* vom alten Wert übernommen werden soll.

Durch das einfügen einer σ (Sigmoid) Multiplikation kann das Netz unterscheiden ob der vorherige Wert übernommen werden soll (1) oder nicht (0).

17.2.4 Long Short-Term Memory Model

LSTM ist ein alternatives Model für GRU. Beide verwenden Hidden States als Memory, LSTM nutzt erweiterten Memorystatus. GRUs sind generell schneller, LSTM zeigen aber bessere Performance.

17.2.5 Bidirectional RNNs

Können in die «Zukunft» schauen, wir müssen wissen, was wir später schreiben möchten.

17.2.6 Encoder-Decoder Architecture

Die vorherigen RNN-modelle können nur verwendet werden wenn Input und Output die gleiche Länge haben. Wenn Input und Output verschiedene Länge haben (z.B. Übersetzung oder Sentiment Analysis) benötigen wir eine encoder-decoder Architektur.

17.2.6.1 Unidirectional Encoder-Decoder Architecture

Die RNN sind miteinander verbunden und Output aus erstem, wird zu Input des zweiten. Die Hidden Layers können vanilla RNNS, GRUs oder LSTMs sein. der gesamte Input wird encoded und als fixed-size context Vektor fürs decoding verwendet. Jeder Output von y_i ist Input von y_{i+1} . Der Input y_1 ist der Context-Vektor plus ein Start-Token.

Der Context-Vektor enthält die gesamte Information aus dem Input.

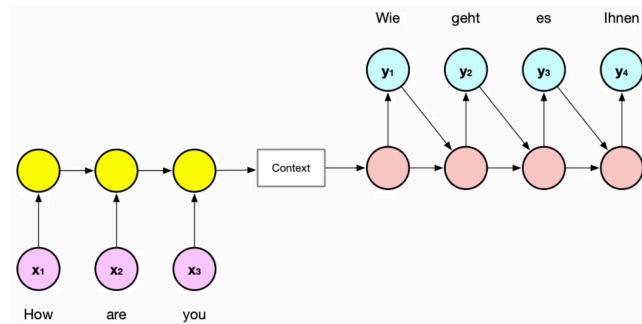


Abbildung 17.3: Unidirectional Encoder-Decoder Architecture

Links den Encoder, rechts den Decoder.

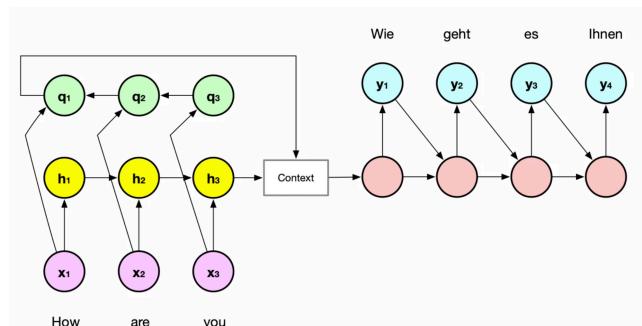


Abbildung 17.4: Bidirectional Encoder-Decoder Architecture

17.3 Attention Models

Die Attention Mechanismen merkt sich wo relevante Informationen von vorherigen Worten herangezogen werden können.

17.3.1 RNNs do not parallelize

Es muss sich also nicht alles merken, sondern weiss wo nachschauen. Allerdings lassen sie sich nicht gut parallelisieren.

17.3.2 Transformers

Attention is all you need

– Paper aus 2017

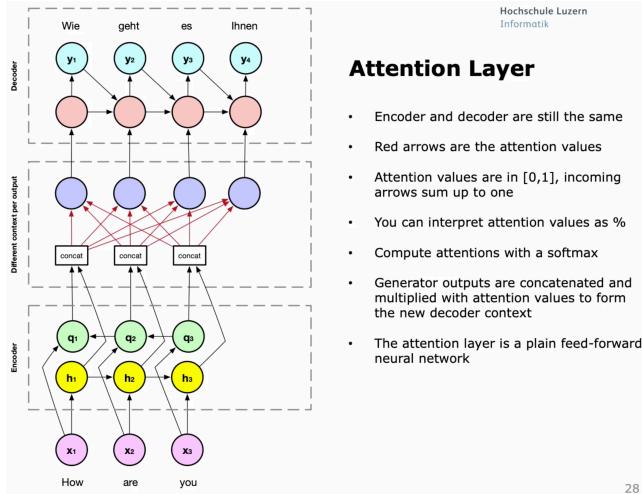


Abbildung 17.5: Attention Layer

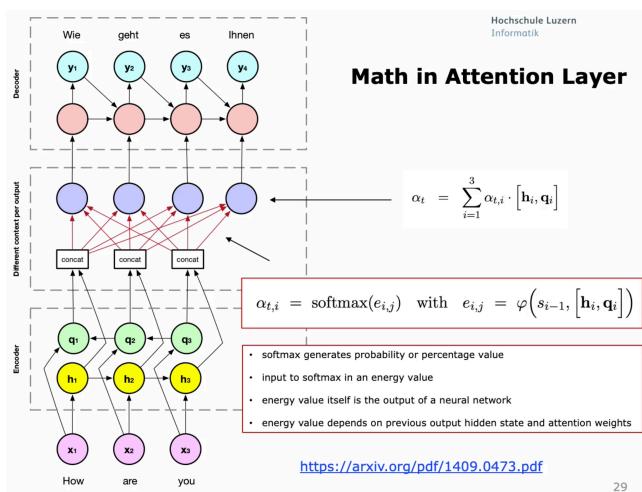


Abbildung 17.6: Math in Attention Layer

18 Code CheatSheet

Contains several code snippets and useful library functions.

18.1 Python

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

18.2 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

18.3 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
<code>(160,3)</code> means 160 data points in 3 dimensions		
<code>np.roll(a, shift, axis=None)</code>		

18.4 Sklearn

Code	Parameters	Description
<pre>sklearn.neighbors.KNeighborsRegressor() train_test_split(X,y, test_size=0.2,random_state=42)</pre>	<p>weights=Gewichtung der k-nahesten Datenpunkte, metric=DistanceMetric $X \in \mathbb{R}^n$ and y must have the same size,*</p>	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.