

Zusammenfassung AIFo 2021

Inhaltsverzeichnis

Inhaltsverzeichnis	1
Woche 1 : AI in science, technology and society	4
Was ist AI? Was ist Machine Learning?	4
Wie kann man testen ob eine AI intelligent ist oder nicht?	4
AI Heute	4
Herausforderung.....	4
Übersicht.....	4
Woche 1: Dialogflow.....	6
Intent	6
Entities	6
Dialog Control (Kontexte)	6
Woche 2: Natural Language Processing (NLP)	6
Was braucht jedes Machine-Learning-Projekt?	7
Data.....	7
Cost-Function (Loss).....	7
Model.....	7
Optimization Procedure.....	7
Performance optimization.....	7
Visualization and evaluation of the Learning Process	7
Cross-Validation & Regularization	7
Wie wird ein Wort in einem Computer repräsentiert?	7
Vorgehen 1: One-Hot Representation	8
Vorgehen 2: Indexing	8
Vorgehen 3: Distributed Representation (dense vectors).....	8
Woche 3: Introduction to Probability	9
Zufallsvariable	9
Beispiel Würfel werfen:	9
Wahrscheinlichkeitsfunktion (Probability Mass Function PMF).....	9
Zwei Zufallsvariablen	10
Zusammengesetzte Wahrscheinlichkeit (Joint Probability Mass Function)	10
Unabhängig	10
Bedingte Wahrscheinlichkeit (Correlated Random Variable).....	10
Woche 3: Naïve Bayes and Document Classification.....	11
Woche 4&5: Cheatsheets für Python, Conda, NumPy, Panda, Mathplotlib, Seaborn	13
Woche 5: Datenvisualisierung	20
Plots	20
Liniendiagramm	20

Balkendiagramm	20
Histogramm	20
Box Plots und Violin Plots	21
Steudiagramm (Scatter Plot)	21
Wann welcher Plot?.....	21
Woche 6: Linear Regression	22
Linear Regression und Machine Learning.....	22
Model.....	22
Mean squared Error MSE (Loss, Residuals)	22
Korrelation und Kausalität	22
Komplexere Lineare Modelle.....	23
Uniform Distribution, Normal Distribution, numpy's random number Generator	23
Seaborn Jointplot.....	23
Woche 7: Grundbegriffe	24
7 Steps of Machine Learning	24
Machine Learning Fundamentals: Bias and Variance.....	24
Bias.....	24
Variance	24
Woche 8: Optimization: Stochastic Gradient Descent (SGD)	25
Stochastic Gradient Descent (SGD).....	26
Stochastic Gradient Descent with annealed learning rate	26
Allgemeine Anmerkungen	26
Woche 9: Generalisation & Regularisierung.....	27
Overfitting.....	27
Underfitting	27
Generalization Error.....	27
Bias und Varianz.....	27
High Bias, Low Variance	27
Low Bias, High Variance	27
Regularisierung	27
Umsetzung mit Keras/Tensorflow	28
Woche 10: Cross-Validation.....	29
Ziel 1: Generalisierungsfehler Schätzen	29
Ziel 2: Auswahl von Hyper-Parameter	29
Scikit-learn Cross-Validation.....	29
Woche 10: Artificial Neural Networks	30
Woche 11: Logistic Regression	31
Binäre Klassifikation und deren Notation.....	31
Warum nicht Lineares Modell?.....	31
Sigmoid Funktion	31

Maximum Likelihood (Loss)	31
Woche 12: Classifier Evaluation	32
Confusion Matrix:	32
Precision vs Recall.....	32
Threshold	32
Woche 12: K-Nearest-Neighbours KKN	33
Funktionsweise	33
Distanz-Metriken	33
Das richtige k und die richtige Distanzmetrik.....	33
Vor und Nachteile	34
Beispiel IRIS.....	34
Woche 13: Clustering.....	34
Unsupervised Learning	34
Unterschied Supervised – Unsupervised Beispiel IRIS.....	34
Clustering (Mit KNN Klassifizierung)	35
Woche 13: Naive K-means.....	35
Stopping-Kriterium	35
Initialisierung	35
Standardisierung.....	35
Python.....	36
Qualität des Clusters: WCSS und Shilhouette.....	36
WCSS	36
Silhouette Score.....	36
Bestimmung welche Clustergrösse anhand von IRIS.....	36
Woche 14: Ensemble Methods.....	37
Unterschiedliche Dimensionen.....	37
Unterschiedliche Algorithmen.....	37
Unterschiedliche Hyperparameter	37
Unterschiedliche Trainingsdaten	37
Voting.....	37
Hard	37
Soft.....	37
Auswählen der Daten	37
Out of Bag (oob) Evaluation.....	38
Codebeispiele	38
No free lunch theorem	38
Komplettes Schaubild	39

Woche 1 : AI in science, technology and society

Was ist AI? Was ist Machine Learning?

Es gibt sehr verschiedene Auslegungen. Man kann es sowohl technisch, als auch philosophisch ansehen.

AI hat viele Forschungs- und Anwendungsfälle. In diesem Modul geht es um Algorithmen, also ein Computer lernt von Daten etwas ähnliches zu machen. ➔ **Statistical Machine Learning**

«AI is a bigger concept to create intelligent machines that can simulate human thinking capability and behavior, whereas, machine learning is an application or subset of AI that allows machines to learn from data without being programmed explicitly»

Ziel: Man Programmiert Algorithmus nicht mehr hart, sondern macht ihn lernfähig mit Beispielen. Dies ist wichtiges Merkmal von Machine Learning und AI.

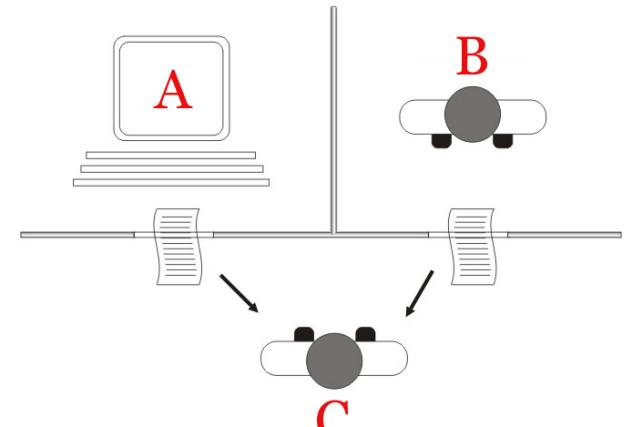
Traum: Das Entwickeln einer **AGI (Artificial general intelligence)**. Davon sind wir noch weit entfernt und AI können in der Regel nur ein spezifisches Problem auf mal lösen.

Wie kann man testen ob eine AI intelligent ist oder nicht?

Hierfür gibt es ein berühmter Test von Alan Turing. Den sogenannten **Turing-Test**.

Testbeschreibung: Person C stellt per Tastatur und Bildschirm eine beliebige Frage an einen anderen Menschen oder KI. Fragesteller weiß nicht wer ihm antwortet. Der Fragesteller entscheidet dann ob es eine Maschine oder der Mensch war. Dazu kann er mehrere Fragen stellen. Die KI ist intelligent, wenn der Mensch diese nicht erkennt.

Dieser Test hat einige Tücken: So muss KI lernen zu lügen und sich dumm zu stellen, da sie ja gleichintelligent wie ein Mensch sein muss.

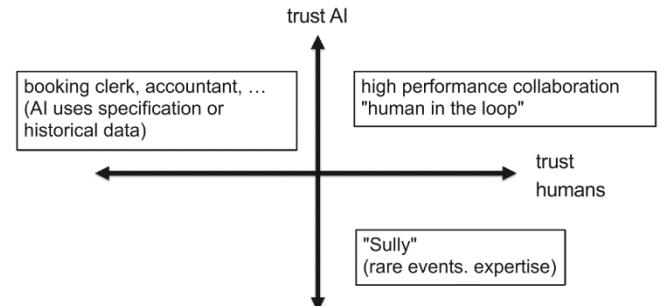


AI Heute

Bilderkennung (Google Fotos), Boston Dynamics, Roboter die Fussball spielen, Pflegeroboter, Personalisierung von News-Feed, Voice-To-Text, Automatischer Rechnungscan

Herausforderung

Werden Roboter akzeptiert? Sind diese ethisch vertretbar? Kann uns die KI mal überholen, wie können wir dagegen vorgehen? Wo vertrauen wir AI? Gibt es Dinge die wir nie einer AI überlassen werden?



Übersicht

Machine Learning ist ein Teilgebiet von AI. Dies wird auch noch weiter unterteilt. Häufig Unterteilung in drei Bereiche: **unsupervised, supervised** und **reinforcement learning**.

Reinforced Learning

Hier geht es unter anderem zu Spielen: Schachcomputer oder das bekannte GO-Spiel wo AI den Mensch schlägt.

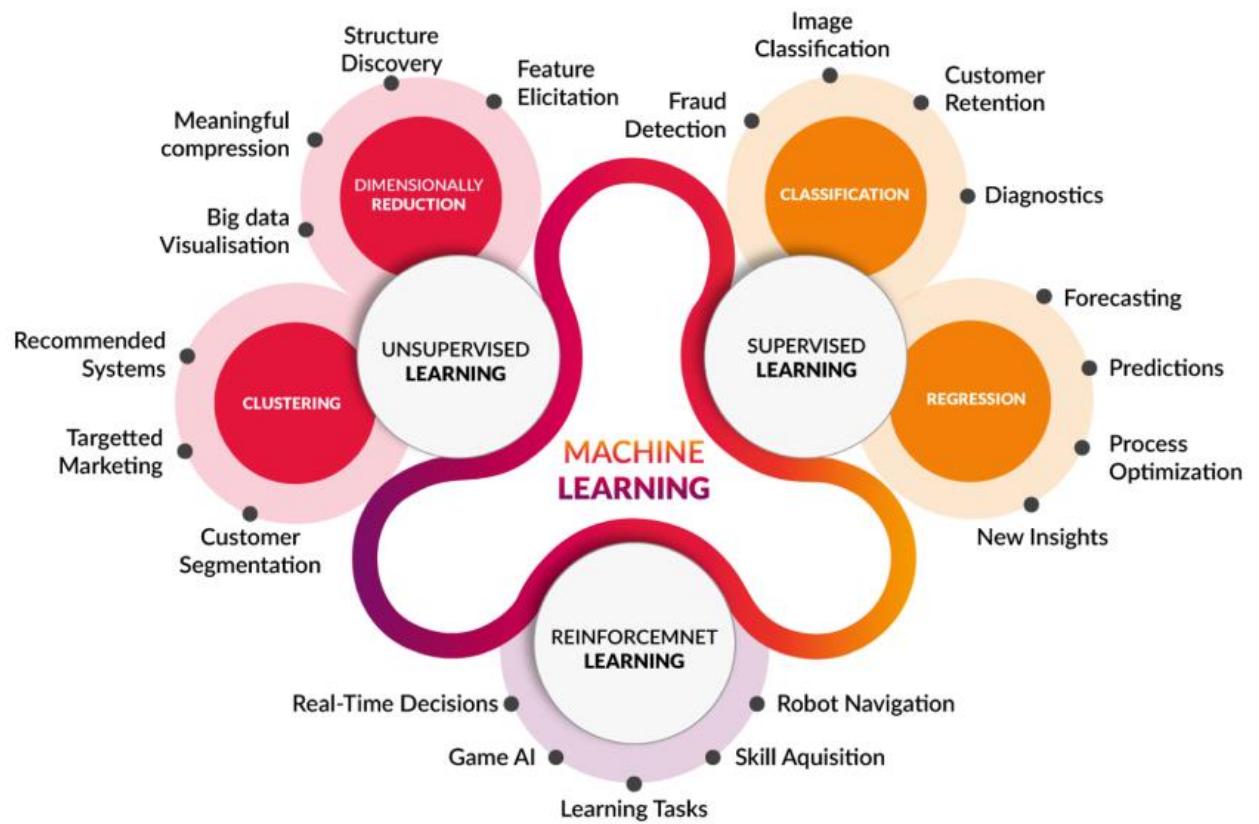
Unsupervised Learning

Hier nimmt man grosse Datenmengen, Segmentiert oder Clustert diese. Man möchte Daten visualisieren. Beispiel Politlandschaft.

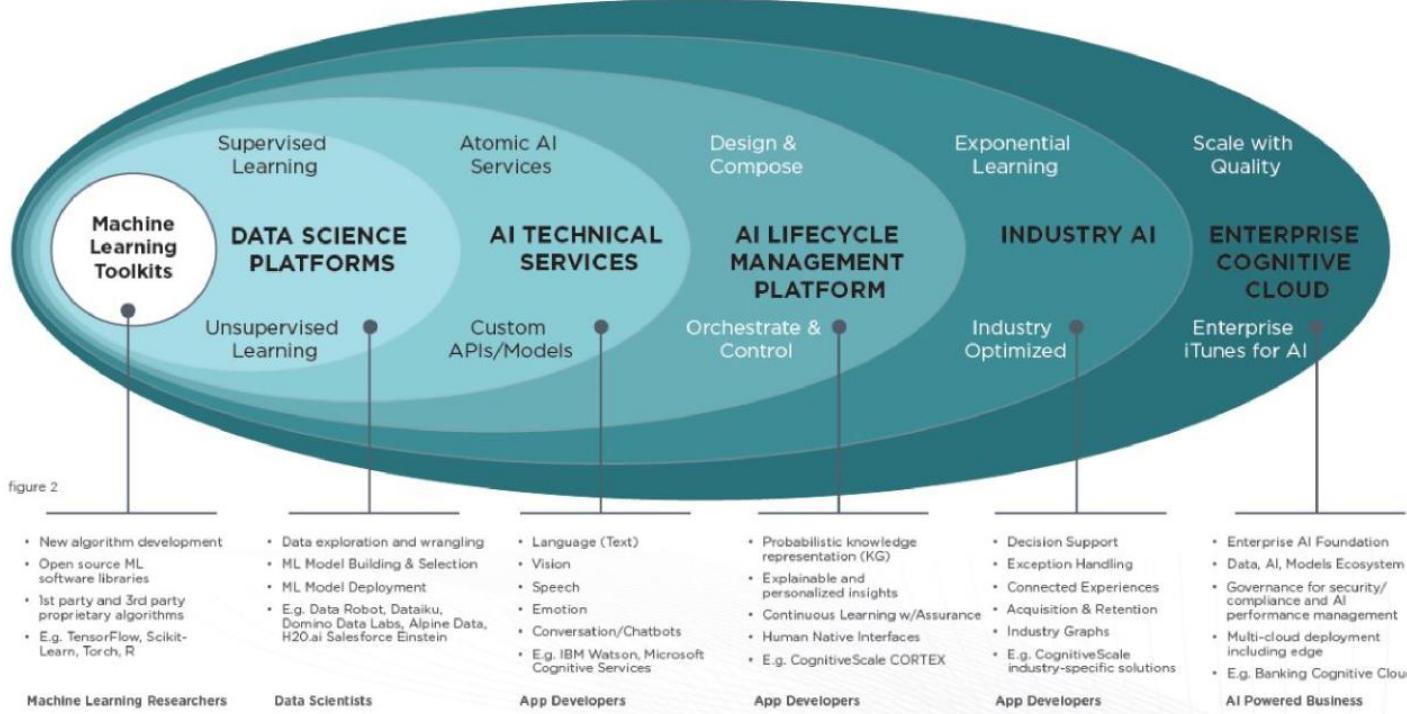
Supervised Learning

Ein Teil ist die Regression in Linearer Algebra. Hier gibt man Daten und Zusatzinformationen (Label). Zum Beispiel gibt man Bilder von Hunden und Katzen und lässt ihn lernen diese zu erkennen. Dazu gibt man jeweils auch noch ein Label mit, welches zeigt welche Bilder von Hunden und welche von Katzen sind.

Machine Learning: Tasks and Algorithms



AI Applications: a more complete picture



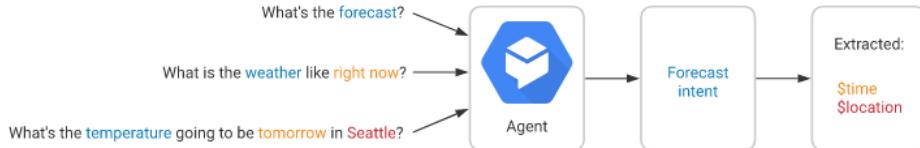
Woche 1: Dialogflow

Ein Service von Google, der eine AI im Hintergrund hat für das erstellen von Chatbots: «AI as a Service»

Das Interface ist sehr einfach, hat aber grosses Backend. Man kann auch JAVA Clients anbinden oder Telegram. Sehr umfangreich und dynamisch erweiterbar.

Intent

Kategorisiert Absicht eines Endbenutzers für Unterhaltungsrunde. Im Intent definiert man Trainingssätze, Aktionen und auch Parameter. So gibt es Dinge die dringend benötigt werden und wenn diese Fehlen, müssen diese Nachgefragt werden. Weiter kann man unterschiedliche Antworten definieren und der Bot wählt eine zufällig aus.

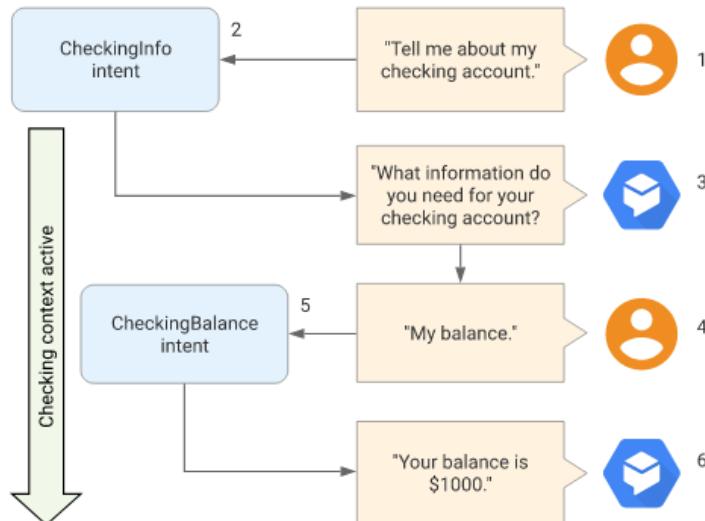


Entities

Dienen um einzelne Wörter aus Sätzen auszulesen und zu mappen. Siehe auch Beispiel oben mit **time** und **location**. Es gibt die Möglichkeit mehrere Trainingsbegriffe zu definieren und auch Synonyme für gleiche Wörter. Auch gibt es die Möglichkeit System-Entities zu verwenden (Vordefinierte Entities, wie Datum und Zeit)

Dialog Control (Kontexte)

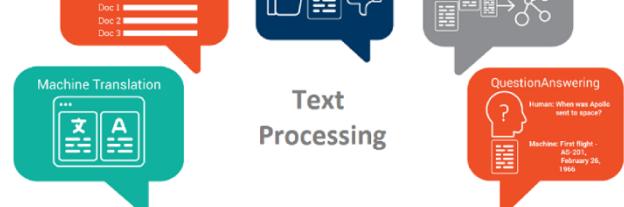
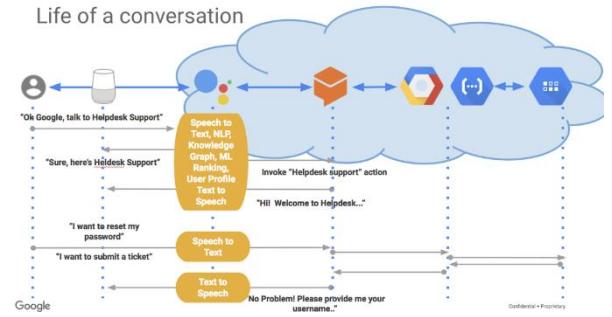
Mit Kontexten kann man definieren welcher Intent als nächstes aufgerufen werden sollte, wenn man ein Intent abgeschlossen hat und kann auch Parameter übergeben, die dann im nächsten Intent verfügbar sind:



Woche 2: Natural Language Processing (NLP)

Automatisches erfassen von menschlicher Sprache (Gesprochen oder geschrieben). Wichtiger Teilbereich in AI. Ziel ist zu verstehen was ein Mensch will. Kürzlich grösse Fortschritte gemacht. Derzeit ist aber verstehen und auch das Antworten immer noch sehr schwierig für eine KI.

Wo schon gut geht sind Suchmaschinen. Auch gibt es bereits Sprachassistenten (Google Assistant, Alexa). Oder DeepL als Übersetzer.



Was braucht jedes Machine-Learning-Projekt?

Die ersten 4 Punkte (Data, Cost-Function, Model, Optimization Procedure) sind die Grundlagen. Die anderen drei benötigt man für verbessertes Machine Learning

"classic" NLP / ML example

Task	Text Analysis, Sentiment Classification → Supervised learning, Binary classification
Example	comment: 'This is the worst movie I've ever seen' 🙄 label: 0 (=negative)
Link	https://www.tensorflow.org/text/guide/word_embeddings
Data & Preprocessing	labeled sentences (Words) from IMDB How to put words into numbers ? Embeddings !
Model	Multi-layer NN with ~160'000 parameters. Input Layer: sentences in an appropriate representation Output-Layer: single neuron. probability that the input belongs to class 1.
Loss	Binary Crossentropy
Optimizer	Adam

Data

Datenset das gegeben ist inklusive der Pre-Prozess-Pipeline unter anderem mit folgenden Funktionen

- **Cleansing:** Aufräumen, Korrigieren und sortieren
- **Feature-Engineering:** Extrahieren von Features, z.B Katze ist ein Säugetier (Beziehungen)
- **Data-Argumentation:** Daten vervielfältigen durch manipulation (Spiegeln, Farbänderung) oder hinzufügen neuer Daten

There's no magic: we can't do better than what's in this data!

Cost-Function (Loss)

Mathematischer Ausdruck der sagt wie gut oder schlecht Aufgabe gelöst wurde. Dies anhand Angabe einer Zahl. Mean Squared Error (MSE) wird häufig genutzt. Weiter gibt es Domain-Spezifische Kosten.

Beispiel: Gesichtserkennungsalgorithmus. Nun muss man Kosten schätzen wenn Gesichtserkennung falsch positiv oder falsch negativ ist. Je nach Ort kann es andere Auswirkungen haben. (Access-Control bei CIA oder Identifizierung von Kunden, die viel Geld ausgeben in einem Geschäft)

Model

Das «Ding» das aus einem Input einen Output generiert. Kann einfach sein (Linear $y_i = ax_i + b$) oder komplexes neuronales Netzwerk mit Millionen von Parameter.

Typischerweise verwendet man Framework wie Tensorflow oder Pytorch

Unterschiedliche Aufgaben benötigen unterschiedliches Modell (Regression, Entscheidungsbaum).

Optimization Procedure

Modell muss nun durch Machine-Learning optimiert werden. Dazu braucht es einen Algorithmus. Optimierungsprozedur nimmt alle oben genannte Elemente und schraubt so lange an Parameter rum, bis das Optionale Ergebnis rauskommt.

Beispiele: Stochastic Gradient Descent (SGD), ADAM, RMSProp, ...

Performance optimization

Bauen von effizienten Pipelines ist schwierig. Deshalb sollten Tool-spezifische Empfehlungen und Referenz-Implementationen zur Hilfe genommen werden.

Visualization and evaluation of the Learning Process

Man kann Lernprozess visualisieren. Man kann also Zeigen ob der Optimierer was sinnvolles tut. Hier kann Tensorboard genutzt werden

Cross-Validation & Regularization

Ziel ist Modelle zu trainieren, die Gut generalisieren. AI soll eine gewisse Flexibilität haben.

Beispiel: Hundebilderkennen soll auch neue Hunderassen automatisch erkennen.

Wie wird ein Wort in einem Computer repräsentiert?

Am einfachsten nutzt man dazu Vektoren, welche Wörter auf Grund Ihrer Meinung interpretieren

Vorgehen 1: One-Hot Representation

Jedem Wort wird ein Vektor zugewiesen mit einem einzigen 1-Wert und alle anderen Werte auf 0 gesetzt.

Vektordimension = Anzahl unterschiedliche Wörter

Nachteile

- **High dimensional:** Vektoren können schnell sehr gross werden (Wikipedia-Artikel hat schnell 100'000 Wörter) [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [1]
 - **Sparse:** Jeder Vektor hat nur eine 1 und dann viele Nullen. (1 zu 99'999 im Wikipediabeispiel). Sind sehr Memory-Ineffizient und KI kann davon nicht viel lernen
 - **No generalization:** Es fehlt der Kontext. Alle Wörter sind unabhängig voneinander und es kann keine Bedeutung abgeleitet werden (Z.B Ananas und Birne sind Essen)

Vorgehen 2: Indexing

Man macht eine Liste von Wörter und weist diese eine Nummer zu. So benötigt ein Satz nur ein einziger Vektor.

Sehr nahe beim One-Hot-Vektor. Aber etwas besser. Deshalb wird Indexing meist als Preprocessing-Step genutzt.

Vorgehen 3: Distributed Representation (dense vectors)

Example: "The cat sat on the mat."

cat	→ 0
mat	→ 1
on	→ 2
sat	→ 3
the	→ 4
.	→ 5

"The cat sat on the mat" → [4, 0, 3, 2, 4, 1, 5]

Ein Wort kann definiert werden mit einem Kontext. Wörter mit ähnlicher Schematik teilen oft den Kontext (cat and rat = Animals)

Beispiel: «Look at that little furry *mukawibuu* with white paws climbing a tree» → *Mukawibuu* wird wohl einem Koala ähneln

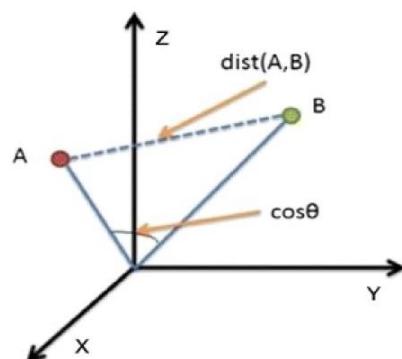
Bei Distributed Representation wird dieses Konzept zu nutze gemacht. Ein Wort, welches Indexiert wurde, wird mit einem Algorithmus (Embedding Layer) zu einem Vektor umgewandelt.

Mit guten Vektoren kann man dann durch Mathematik Ähnlichkeiten erkennen oder durch Addition/Subtraktion neue Wörter bilden. Dazu wird das Skalarprodukt verwendet. Somit kann der Computer mittels Mathematik die Ähnlichkeiten von unterschiedlichen Wörtern erkennen.

- Skalarprodukt zwischen zwei Vektoren ist maximal, wenn beide in die selbe Richtung zeigen
 - o Haben beide Vektoren die Norm 1, dann ist das Maximum 1
 - Skalarprodukt zwischen zwei Vektoren ist null, wenn die Vektoren senkrecht (orthogonal) zueinander stehen.
 - Skalarprodukt zwischen zwei Vektoren ist minimal (negativ), wenn beide Vektoren in entgegengesetzte Richtung zeigen
 - o Haben beide Vektoren die Norm 1, dann ist das minimum -1

Cosine Distance (Kosinusdistanz)

Das Skalarprodukt wird verwendet für die Cosine-Distance (Kosinusdistanz). Ähnliche Wörter haben eine ähnliche Kosinusdistanz.



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Mathlab: $\text{dot}(a,b)/(\text{norm}(a)*\text{norm}(b))$

Rechenbeispiel

Berechne die Kosinusdistanz zwischen Elefant und Maus, sowie zwischen Elefant und Bike.

```
>> dot(e,m) / (norm(e)*norm(m))
ans =
0.9740
>> dot(e,b) / (norm(e)*norm(b))
ans =
0.3444
```

Elefant und Maus sind sich sehr ähnlich, weshalb die Kosinusdistanz mit 0.974 nahe bei 1 ist. Elefant und Bike ähneln sich nicht.

Praktik

Das Trainieren eines solches Modell und generieren dieser Vektoren ist schwierig. Deshalb verwendet man meist ein vordefiniertes Modell. Für selbständige trainieren gibt es bereits Architekturen wie word2vec.

Woche 3: Introduction to Probability

Zufallsvariable

Eine Zufallsvariable X (oder anderer Grossbuchstabe) ist eine Variable, die einen **numerischen Wert** x (Kleinbuchstabe) zuordnet, welche aus einem Zufallsexperiment kommt.

Es gibt zwei Arten von Zufallsvariablen:

- **Diskret:** X nimmt einen fixen wert an aus einer fix definierten Zahlenmenge: {1.56, 2.93, 4, 6, 8}
- **Stetig:** X nimmt einen Wert aus einem unzählbaren bereich. Z.B alle Zahlen im Intervall (2, 7)

Vergleich mit anderen Variablen:

- **Algebra:** In einer expression ' $x+2=10$ ' ist eine variable ein wert, wo eine Gleichung in einen richtigen Ausdruck bringt.
- **Programmierung:** Hier ist einer Variable eine Zuweisung. Die Variable kann ihren Wert mit jeder Neuzuweisung ändern.

Vor dem ausführen eines Zufallsexperiments, das beste das wir wissen ist eine Liste von allen möglichen Werten mit ihren Wahrscheinlichkeiten.

Beispiel Würfel werfen:

Example: The discrete random variable X is the number observed when rolling a fair dice.

The possible values and the probabilities they take are:

Value x of the random variable X	1	2	3	4	5	6
Pr($X=x$)	1/6	1/6	1/6	1/6	1/6	1/6

$\text{Pr}(X=x)$ is the probability that the random variable X takes the value x .

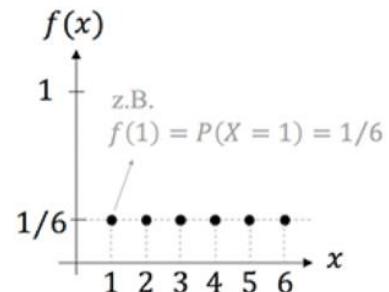
$\text{Pr}(X=x)$ is often written in the more compact form $P(x)$ or $p(x)$, or sometimes as $P_x(x)$ (there's no formal rule.
When reading articles, you'll notice that notation is often a bit sloppy, informal, and sometimes confusing)

Wahrscheinlichkeitsfunktion (Probability Mass Function PMF)

Die Wahrscheinlichkeitsfunktion einer diskreten Variable ist eine Funktion, welche die Wahrscheinlichkeit angibt für jeden möglichen x -Wert.

Value x of the random Variable X	1	2	3	4	5	6
Pr($X=x$)	1/6	1/6	1/6	1/6	1/6	1/6

Sie definiert auch eine Wahrscheinlichkeitsverteilung (Probability Distribution). Meist als Synonym verwendet. Siehe auch Bild rechts.



Zwei Zufallsvariablen

Man kann auch zwei Zufallsvariablen gleichzeitig ansehen. Z.B X ist Augenzahl von Würfel 1 und Y ist Augenzahl von Würfel 2.

Dann kann das Zufallsexperiment sein, dass man zwei Würfel wirft, was 36 Möglichkeiten gibt. Jede Möglichkeit ist gleich wahrscheinlich.

	X=1	X=2	X=3	X=4	X=5	X=6
Y=1	1/36	1/36	1/36	1/36	1/36	1/36
Y=2	1/36	1/36	1/36	1/36	1/36	1/36
Y=3	1/36	1/36	1/36	1/36	1/36	1/36
Y=4	1/36	1/36	1/36	1/36	1/36	1/36
Y=5	1/36	1/36	1/36	1/36	1/36	1/36
Y=6	1/36	1/36	1/36	1/36	1/36	1/36

Zusammengesetzte Warscheinlichkeit (Joint Probability Mass Function)

Die Zusammengesetzte Warscheinlichkeit oben beschreibt z.B die Warscheinlichkeit das Augenzahl Y=4 und Augenzahl X=5 ist.

Diese beträgt 1/36. Man schreibt dies auch $Pr(X=5, Y=4) = 1/36$ oder $P(5,4) = 1/36$

Unabhängig

Im Oben genannten Beispiel sind die beiden Zufallsvariablen unabhängig. Ein Würfelwurf beinflusst das andere Ergebnis nicht.

Also gilt folgende Formel:

$$P(X, Y) = P(X) \cdot P(Y) \quad (\text{if } X \text{ and } Y \text{ are independent})$$

For the case of 3 independent random variables:

$$P(X, Y, Z) = P(X) \cdot P(Y) \cdot P(Z) \quad (\text{if } X, Y \text{ and } Z \text{ are independent})$$

Bedingte Warscheinlichkeit (Correlated Random Variable)

Doch Unabhängig ist eher die Ausnahme als die Regel. Meist sind die Variablen Abhängig voneinander. Dann gilt folgende Regel

- By fundamental rules of probabilities, **joint** probability $P(X, Y)$ and **conditional** probability $P(X|Y)$ are related in the following way:

$$P(X, Y) = P(X|Y)P(Y)$$

- analogous: $P(X, Y) = P(Y, X) = P(Y|X)P(X)$
- We can rearrange it to calculate the conditional probability from the joint (and the marginal)
- $P(Y|X) = \frac{P(X,Y)}{P(X)}$

Satz von Bayes

Ebenso gilt die Bayes-Regel:

- $P(X, Y) = P(X|Y)P(Y)$
- $P(X, Y) = P(Y, X) = P(Y|X)P(X)$

We can rearrange the terms and obtain Bayes rule

$$P(X|Y)P(Y) = P(Y|X)P(X)$$



https://en.wikipedia.org/wiki/Thomas_Bayes

- Thomas Bayes, english mathematician & philosopher, 1701-1761

Beispiele

Look at the following joint probabilities. The random variables are:

X: The event to observe clouds (0= no clouds, 1= small clouds, 2= big clouds)

Y: The event that it rains (0=no rain, 1=light rain, 2=moderate rain, 3=heavy rain)

	X=0	X=1	X=2
Y=0	0.35	0.21	0.03
Y=1	0.10	0.07	0.04
Y=2	0.00	0.05	0.05
Y=3	0.00	0.02	0.08

- a) What is the probability that there are *small clouds* AND there is *moderate rain* ?

We can directly read this from the joint distribution: 0.05.

- b) What is the probability for "heavy rain"?

We marginalize over X:

$$\begin{aligned} P(Y=3) &= P(X=0, Y=3) + P(X=1, Y=3) + P(X=2, Y=3) \\ &= 0+0.02+0.08 = 0.10 \end{aligned}$$

- c) What is the probability for "no clouds" ?

Analogous: We marginalize over Y and obtain:

$$P(X = 0) = \sum_{y=0}^3 P(X = 0, Y = y) = 0.45$$

You observe "small clouds". What is the probability for "moderate rain"?

- a) Rewrite the question in terms of a mathematical expression.

We are interested in this quantity: $P(Y = 2|X = 1)$

- b) Calculate the result.

We want to calculate a **conditional** probability from the (given) **joint** probability. The fundamental formula is (see slides):

$$P(X, Y) = P(Y|X)P(X)$$

which is rearranged into

$$P(Y|X) = \frac{P(X, Y)}{P(X)}$$

$P(X)$ is the marginal. $P(X, Y)$ is given. Therefore:

$$P(Y = 2|X = 1) = \frac{P(X=1, Y=2)}{P(Y=2)} = \frac{0.05}{0.21+0.07+0.05+0.02} = 0.143$$

Woche 3: Naïve Bayes and Document Classification

Der Naive Bayes-Algorithmus ist ein probabilistischer Klassifikationsalgorithmus. Klassifikationsalgorithmus bedeutet, dass der Algorithmus Beobachtungen verschiedenen Klassen zuordnet. Und probabilistisch, dass es mit Wahrscheinlichkeiten zu tun hat. Denn der Naive Bayes-Algorithmus gibt uns für jede Klasse eine Wahrscheinlichkeit, dass die Beobachtung (x_1, \dots, x_n) zu dieser Klasse K_i gehört.

Der Naive Bayes-Klassifikator nimmt (naiverweise) an, dass die A-posteriori-Verteilung aus Wahrscheinlichkeiten aufgebaut ist, bei denen (bei gegebener Klasse) die Features unabhängig voneinander sind:

$$\hat{c}^{\text{Bayes}}(f_1, \dots, f_n) = \arg \max_c p(C = c | f_1, \dots, f_n) \propto \arg \max_c p(C = c) \prod_{i=1}^n p(f_i | C = c)$$

Aufgrund seiner schnellen Berechenbarkeit bei guter Erkennungsrate ist auch der naive Bayes-Klassifikator sehr beliebt. Mittels des naiven Bayes-Klassifikators ist es möglich, die Zugehörigkeit eines Objektes (Klassenattribut) zu einer Klasse zu bestimmen.

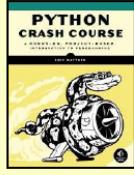
Er basiert auf dem Bayesschen Theorem. Man könnte einen naiven Bayes-Klassifikator auch als sternförmiges Bayessches Netz betrachten.

Die naive Grundannahme ist dabei, dass jedes Attribut nur vom Klassenattribut abhängt. Obwohl dies in der Realität selten zutrifft, erzielen naive Bayes-Klassifikatoren bei praktischen Anwendungen häufig gute Ergebnisse, solange die Attribute nicht zu stark korreliert sind.

Für den Fall starker Abhängigkeiten zwischen den Attributen ist eine Erweiterung des naiven Bayes-Klassifikators um einen Baum zwischen den Attributen sinnvoll. Das Ergebnis wird baumerweiterter naiver Bayes-Klassifikator genannt.

Hinweis: Nicht im Detail Prüfungsrelevant. Wichtig ist das man die Bayes Formel kennt. Dies ist einfach eine beispielhafte Anwendung bei Spam-Erkennung.

Woche 4&5: Cheatsheets für Python, Conda, NumPy, Panda, Mathplotlib, Seaborn

Beginner's Python Cheat Sheet													
Variables and Strings <i>Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.</i> Hello world <pre>print("Hello world!")</pre> Hello world with a variable <pre>msg = "Hello world!" print(msg)</pre> Concatenation (combining strings) <pre>first_name = 'albert' last_name = 'einstein' full_name = first_name + ' ' + last_name print(full_name)</pre>	Lists (cont.) List comprehensions <pre>squares = [x**2 for x in range(1, 11)]</pre> Slicing a list <pre>finishers = ['sam', 'bob', 'ada', 'bea'] first_two = finishers[:2]</pre> Copying a list <pre>copy_of_bikes = bikes[:]</pre>												
Lists <i>A list stores a series of items in a particular order. You access items using an index, or within a loop.</i> Make a list <pre>bikes = ['trek', 'redline', 'giant']</pre> Get the first item in a list <pre>first_bike = bikes[0]</pre> Get the last item in a list <pre>last_bike = bikes[-1]</pre> Looping through a list <pre>for bike in bikes: print(bike)</pre> Adding items to a list <pre>bikes = [] bikes.append('trek') bikes.append('redline') bikes.append('giant')</pre> Making numerical lists <pre>squares = [] for x in range(1, 11): squares.append(x**2)</pre>	Tuples <i>Tuples are similar to lists, but the items in a tuple can't be modified.</i> Making a tuple <pre>dimensions = (1920, 1080)</pre>												
	If statements <i>If statements are used to test for particular conditions and respond appropriately.</i> Conditional tests <table> <tbody> <tr><td>equals</td><td>x == 42</td></tr> <tr><td>not equal</td><td>x != 42</td></tr> <tr><td>greater than</td><td>x > 42</td></tr> <tr><td>or equal to</td><td>x >= 42</td></tr> <tr><td>less than</td><td>x < 42</td></tr> <tr><td>or equal to</td><td>x <= 42</td></tr> </tbody> </table> Conditional test with lists <pre>'trek' in bikes 'surly' not in bikes</pre> Assigning boolean values <pre>game_active = True can_edit = False</pre> A simple if test <pre>if age >= 18: print("You can vote!")</pre>	equals	x == 42	not equal	x != 42	greater than	x > 42	or equal to	x >= 42	less than	x < 42	or equal to	x <= 42
equals	x == 42												
not equal	x != 42												
greater than	x > 42												
or equal to	x >= 42												
less than	x < 42												
or equal to	x <= 42												
	Dictionaries <i>Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.</i> A simple dictionary <pre>alien = {'color': 'green', 'points': 5}</pre> Accessing a value <pre>print("The alien's color is " + alien['color'])</pre> Adding a new key-value pair <pre>alien['x_position'] = 0</pre> Looping through all key-value pairs <pre>fav_numbers = {'eric': 17, 'ever': 4} for name, number in fav_numbers.items(): print(name + ' loves ' + str(number))</pre>												
	Looping through all keys <pre>fav_numbers = {'eric': 17, 'ever': 4} for name in fav_numbers.keys(): print(name + ' loves a number')</pre> Looping through all the values <pre>fav_numbers = {'eric': 17, 'ever': 4} for number in fav_numbers.values(): print(str(number) + ' is a favorite')</pre>												
	User input <i>Your programs can prompt the user for input. All input is stored as a string.</i> Prompting for a value <pre>name = input("What's your name? ") print("Hello, " + name + "!")</pre> Prompting for numerical input <pre>age = input("How old are you? ") age = int(age) pi = input("What's the value of pi? ") pi = float(pi)</pre>												
<p align="center">Python Crash Course</p> <p align="center">Covers Python 3 and Python 2</p> <p align="center">nostarchpress.com/pythoncrashcourse</p> 													

While loops

A while loop repeats a block of code as long as a certain condition is true.

A simple while loop

```
current_value = 1
while current_value <= 5:
    print(current_value)
    current_value += 1
```

Letting the user choose when to quit

```
msg = ''
while msg != 'quit':
    msg = input("What's your message? ")
    print(msg)
```

Functions

Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

A simple function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
```

```
greet_user()
```

Passing an argument

```
def greet_user(username):
    """Display a personalized greeting."""
    print("Hello, " + username + "!")
```

```
greet_user('jesse')
```

Default values for parameters

```
def make_pizza(topping='bacon'):
    """Make a single-topping pizza."""
    print("Have a " + topping + " pizza!")
```

```
make_pizza()
make_pizza('pepperoni')
```

Returning a value

```
def add_numbers(x, y):
    """Add two numbers and return the sum."""
    return x + y
```

```
sum = add_numbers(3, 5)
print(sum)
```

Classes

A class defines the behavior of an object and the kind of information an object can store. The information in a class is stored in attributes, and functions that belong to a class are called methods. A child class inherits the attributes and methods from its parent class.

Creating a dog class

```
class Dog():
    """Represent a dog."""

    def __init__(self, name):
        """Initialize dog object."""
        self.name = name

    def sit(self):
        """Simulate sitting."""
        print(self.name + " is sitting.")

my_dog = Dog('Peso')

print(my_dog.name + " is a great dog!")
my_dog.sit()
```

Inheritance

```
class SARDog(Dog):
    """Represent a search dog."""

    def __init__(self, name):
        """Initialize the sardog."""
        super().__init__(name)

    def search(self):
        """Simulate searching."""
        print(self.name + " is searching.")

my_dog = SARDog('Willie')

print(my_dog.name + " is a search dog.")
my_dog.sit()
my_dog.search()
```

Infinite Skills

If you had infinite programming skills, what would you build?

As you're learning to program, it's helpful to think about the real-world projects you'd like to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. If you haven't done so already, take a few minutes and describe three projects you'd like to create.

Working with files

Your programs can read from files and write to files. Files are opened in read mode ('r') by default, but can also be opened in write mode ('w') and append mode ('a').

Reading a file and storing its lines

```
filename = 'siddhartha.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line)
```

Writing to a file

```
filename = 'journal.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

Appending to a file

```
filename = 'journal.txt'
with open(filename, 'a') as file_object:
    file_object.write("\nI love making games.")
```

Exceptions

Exceptions help you respond appropriately to errors that are likely to occur. You place code that might cause an error in the try block. Code that should run in response to an error goes in the except block. Code that should run only if the try block was successful goes in the else block.

Catching an exception

```
prompt = "How many tickets do you need? "
num_tickets = input(prompt)

try:
    num_tickets = int(num_tickets)
except ValueError:
    print("Please try again.")
else:
    print("Your tickets are printing.")
```

Zen of Python

Simple is better than complex

If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

More cheat sheets available at
ehmatthes.github.io/pcc/



CONDA CHEAT SHEET

Command line package and environment manager

Learn to use conda in 30 minutes at bit.ly/tryconda

TIP: Anaconda Navigator is a graphical interface to use conda. Double-click the Navigator icon on your desktop or in a Terminal or at the Anaconda prompt, type `anaconda-navigator`

Conda basics

Verify conda is installed, check version number

```
conda info
```

Update conda to the current version

```
conda update conda
```

Install a package included in Anaconda

```
conda install PACKAGENAME
```

Run a package after install, example Spyder*

```
spyder
```

Update any installed program

```
conda update PACKAGENAME
```

Command line help

```
COMMANDNAME --help
```

```
conda install --help
```

*Must be installed and have a deployable command, usually PACKAGENAME

Using environments

Create a new environment named py35, install Python 3.5

```
conda create --name py35 python=3.5
```

Activate the new environment to use it

```
WINDOWS: activate py35  
LINUX, macOS: source activate py35
```

Get a list of all my environments, active environment is shown with *

```
conda env list
```

Make exact copy of an environment

```
conda create --clone py35 --name py35-2
```

List all packages and versions installed in active environment

```
conda list
```

List the history of each change to the current environment

```
conda list --revisions
```

Restore environment to a previous revision

```
conda install --revision 2
```

Save environment to a text file

```
conda list --explicit > bio-env.txt
```

Delete an environment and everything in it

```
conda env remove --name bio-env
```

Deactivate the current environment

```
WINDOWS: deactivate  
macOS, LINUX: source deactivate
```

Create environment from a text file

```
conda env create --file bio-env.txt
```

Stack commands: create a new environment, name it bio-env and install the biopython package

```
conda create --name bio-env biopython
```

Finding conda packages

Use conda to search for a package

```
conda search PACKAGENAME
```

See list of all packages in Anaconda

```
https://docs.anaconda.com/anaconda/packages/pkg-docs
```

Installing and updating packages

Install a new package (Jupyter Notebook) in the active environment

```
conda install jupyter
```

Run an installed package (Jupyter Notebook)

```
jupyter-notebook
```

Install a new package (toolz) in a different environment (bio-env)

```
conda install --name bio-env toolz
```

Update a package in the current environment

```
conda update scikit-learn
```

Install a package (boltons) from a specific channel (conda-forge)

```
conda install --channel conda-forge boltons
```

Install a package directly from PyPI into the current active environment using pip

```
pip install boltons
```

Remove one or more packages (toolz, boltons) from a specific environment (bio-env)

```
conda remove --name bio-env toolz boltons
```

Managing multiple versions of Python

Install different version of Python in a new environment named py34

```
conda create --name py34 python=3.4
```

Switch to the new environment that has a different version of Python

```
Windows: activate py34  
Linux, macOS: source activate py34
```

Show the locations of all versions of Python that are currently in the path

```
Windows: where python  
Linux, macOS: which -a python
```

NOTE: The first version of Python in the list will be executed.

Show version information for the current active Python

```
python --version
```

Specifying version numbers

Ways to specify a package version number for use with conda create or conda install commands, and in meta.yaml files.

Constraint type

Specification

Result

Fuzzy

`numpy=1.11`

1.11.0, 1.11.1, 1.11.2, 1.11.18 etc.

Exact

`numpy==1.11`

1.11.0

Greater than or equal to

`"numpy>=1.11"`

1.11.0 or higher

OR

`"numpy=1.11.1|1.11.3"`

1.11.1, 1.11.3

AND

`"numpy>=1.8,<2"`

1.8, 1.9, not 2.0

NOTE: Quotation marks must be used when your specification contains a space or any of these characters: > < | *

Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com



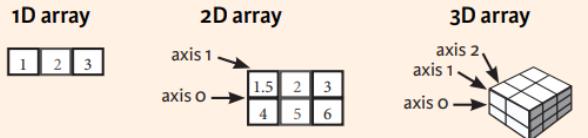
NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]),
      dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2X2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savetxt('array.npy', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

<code>>>> np.int64</code>	Signed 64-bit integer types
<code>>>> np.float32</code>	Standard double-precision floating point
<code>>>> np.complex</code>	Complex numbers represented by 128 floats
<code>>>> np.bool</code>	Boolean type storing TRUE and FALSE values
<code>>>> np.object</code>	Python object type
<code>>>> np.string_</code>	Fixed-length string type
<code>>>> np.unicode_</code>	Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
array([-0.5,  0. ,  0. ],
      [-3. , -3. , -3. ])
>>> np.subtract(a,b)
>>> b + a
array([ 2.5,  4. ,  6. ],
      [ 5. ,  7. ,  9. ])
>>> np.add(b,a)
>>> a / b
array([ 0.66666667,  1. ,
      [ 0.25      ,  0.4       ,  0.5      ],
      [ 1.5,  4. ,  6. ,  8. ]])
>>> np.divide(a,b)
>>> a * b
array([ 1.5,  4. ,  9. ],
      [ 4. , 10. , 18. ])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
array([[ 7.,  7.],
      [ 7.,  7.]])
```

Subtraction
Subtraction
Addition
Addition
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
array([[False,  True,  True],
      [False, False, False]], dtype=bool)
>>> a < 2
array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2]
3
>>> b[1,2]
6.0
```

1	2	3
1.5	2	3
4	5	6

Select the element at the 2nd index
Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

```
>>> a[0:2]
array([1, 2, 3])
>>> b[0:2,1]
array([ 2.,  5.])
```

1	2	3
1.5	2	3
4	5	6

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1

```
>>> b[:1]
array([[1.5, 2., 3.]])
>>> c[1,...]
array([[[ 3.,  2.,  1.],
      [ 4.,  5.,  6.]]])
```

1.5	2	3
4	5	6

Select all items at row 0 (equivalent to `b[0:1, :]`)
Same as `[1,:,:]`

```
>>> a[ : :-1]
array([3, 2, 1])
```

1	2	3
2	1	
3		

Reversed array `a`
Select elements from `a` less than 2
Select elements `(1,0),(0,1),(1,2)` and `(0,0)`
Select a subset of the matrix's rows and columns

Also see Lists

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a, [1])
```

Return a new array with shape `(2,6)`
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
array([[ 1.,  2.,  3.],
      [ 4.,  5.,  6.],
      [ 1.5, 2., 3.],
      [ 4., 5., 6.],
      [ 4., 5., 6.],
      [ 1.5, 2., 3.],
      [ 4., 5., 6.]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
array([[ 7.,  7.,  1.,  0.],
      [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
array([[ 1, 10],
      [ 2, 15],
      [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[ 1.5,  2.,  1.],
      [ 4.,  5.,  6.]]),
      array([[ 3.,  2.,  1.],
      [ 4.,  5.,  6.]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index



Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science [Interactively](#) at www.DataCamp.com



Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

	Country	Capital	Population
Index	Belgium	Brussels	11190846
0	India	New Delhi	1303171035
1	Brazil	Brasilia	207847528

A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   ...: 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   ...: 'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
   ...: columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')

Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-5
>>> df[1:]
   Country Capital Population
1  India    New Delhi  1303171035
2  Brazil   Brasilia  207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]
   Belgium
>>> df.iat[[0], [0]]
   Belgium
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
   Belgium
>>> df.at[[0], ['Country']]
   Belgium
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
   Country      Brazil
   Capital     Brasilia
   Population  207847528
>>> df.ix[:, 'Capital']
0    Brussels
1   New Delhi
2   Brasilia
```

Select single row of subset of rows

```
>>> df.ix[1, 'Capital']
   'New Delhi'
```

Select a single column of subset of columns

```
>>> df.ix[1, 'Capital']
   'New Delhi'
```

Select rows and columns

```
>>> Boolean indexing
>>> s~(s > 1)
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1

s where value is <-1 or >2

Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns (axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min() / df.max()
>>> df.idxmin() / df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cumulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
   a    10.0
   b    NaN
   c    5.0
   d    7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
   a    10.0
   b    -5.0
   c     5.0
   d     7.0
>>> s.sub(s3, fill_value=2)
   a    10.0
   b    -7.0
   c     5.0
   d     7.0
>>> s.div(s3, fill_value=4)
   a    10.0
   b    -5.0
   c     5.0
   d     7.0
>>> s.mul(s3, fill_value=3)
   a    10.0
   b    -6.0
   c     5.0
   d     7.0
```



Python For Data Science Cheat Sheet

Matplotlib

Learn Python Interactively at www.DataCamp.com



Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.



1 Prepare The Data

Also see [Lists & NumPy](#)

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].barh([1,2,3],[3,4,5])
>>> axes[1,0].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horizontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

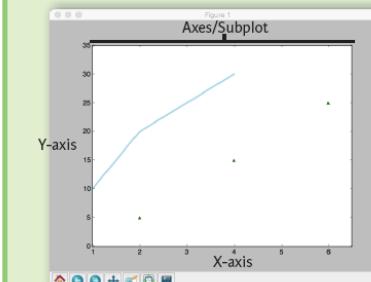
2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                  cmap='gist_earth',
                  interpolation='nearest',
                  vmin=-2,
                  vmax=2)
```

Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
- 2 Create plot
- 3 Plot
- 4 Customize plot
- 5 Save plot
- 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4] Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure() Step 2
>>> ax = fig.add_subplot(111) Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3, 4
>>> ax.scatter([2,4,6],
              [5,15,25],
              color='darkgreen',
              marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png') Step 5
>>> plt.show() Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".") 
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x*x2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1, -2.1, 'Example Graph', style='italic')
>>> ax.annotate("Sine", xy=(8, 0), xycoords='data',
               xytext=(10.5, 0), textcoords='data',
               arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Legends

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y', direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
                        hspace=0.3,
                        left=0.125,
                        right=0.9,
                        top=0.9,
                        bottom=0.1)
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible
Move the bottom axis line outward

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window



Python For Data Science Cheat Sheet

3) Plotting With Seaborn

Seaborn

Learn Data Science Interactively at www.DataCamp.com



Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on **matplotlib** and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> sns.set_style("whitegrid")           Step 2
>>> g = sns.lmplot(x="tip",
                    y="total_bill",
                    data=tips,
                    aspect=2)                         Step 3
>>> g.set_axis_labels("Tip", "Total bill (USD)") .
set(xlim=(0,10), ylim=(0,100))          Step 4
>>> plt.title("title")                  Step 5
>>> plt.show(g)
```

1) Data

Also see [Lists, NumPy & Pandas](#)

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                        'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2) Figure Aesthetics

```
>>> f, ax = plt.subplots(figsize=(5, 6)) Create a figure and one subplot
```

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                 {"xtick.major.size":8,
                  "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters
Return a dict of params or use with
with to temporarily set the style

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g.map(plt.hist, "age")
>>> sns.factorplot(x="pclass",
                     y="survived",
                     hue="sex",
                     data=titanic)
>>> sns.lmplot(x="sepal_width",
                     y="sepal_length",
                     hue="species",
                     data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
                     sns.distplot)
>>> sns.jointplot("sepal_length",
                     "sepal_width",
                     data=iris,
                     kind='kde')
```

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                     y="petal_length",
                     data=iris)
>>> sns.swarmplot(x="species",
                     y="petal_length",
                     data=iris)
```

Bar Chart

```
>>> sns.barplot(x="sex",
                     y="survived",
                     hue="class",
                     data=titanic)
```

Count Plot

```
>>> sns.countplot(x="deck",
                     data=titanic,
                     palette="Greens_d")
```

Point Plot

```
>>> sns.pointplot(x="class",
                     y="survived",
                     hue="sex",
                     data=titanic,
                     palette={"male":"g",
                             "female":"m"},
                     markers=["^","o"],
                     linestyles=["-","--"])
```

Boxplot

```
>>> sns.boxplot(x="alive",
                     y="age",
                     hue="adult_male",
                     data=titanic)
```

Violinplot

```
>>> sns.violinplot(x="age",
                     y="sex",
                     hue="survived",
                     data=titanic)
```

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Show point estimates and confidence intervals with scatterplot glyphs

Show count of observations

Show point estimates and confidence intervals as rectangular bars

Boxplot

Boxplot with wide-form data

Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                     y="sepal_length",
                     data=iris,
                     ax=ax)
```

Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                           kde=False,
                           color="b")
```

Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data, vmin=0, vmax=1) Heatmap
```

4) Further Customizations

Also see [Matplotlib](#)

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                     "Sex")
```

Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax, yticks=[0,5])
>>> plt.tight_layout()
```

Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5) Show or Save Plot

Also see [Matplotlib](#)

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                     font_scale=1.5,
                     rc={"lines.linewidth":2.5})
```

Also see [Matplotlib](#)

Set context to "talk"
Set context to "notebook",
scale font elements and
override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
>>> sns.set_palette(flatui)
```

Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

Close & Clear

Also see [Matplotlib](#)

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear an entire figure
Close a window



Woche 5: Datenvisualisierung

Datenvisualisierung bedeutet daten Grafisch darstellen. Dies ist ein erster Schritt in Machine Learning.

Visualisierung von Daten hilft um..

- ... ein intuitives Verständnis der Daten zu erhalten
- ... Trends, clusters und lokale Muster zu sehen und identifizieren, welche man bei Raw-Daten sehr schwer sieht
- ... Entdecken von Ausreissern und ungewöhnliche Gruppen
- ...Trends zu identifizieren, cluster zu sehen
- ...unsere Hypothesen/Vermutungen/Theorien zu validieren
- ... Resultate von Umfragen, etc. zu visualisieren. (Dies sollte man bei wichtigen Nachrichten tun, da Leute meist nur die Grafiken in einem Report ansehen, wenn sie diesen überfliegen.)

We don't want data. We want information.

Daten Visualisierungen können auch überraschende Fakten aufdecken.

Plots

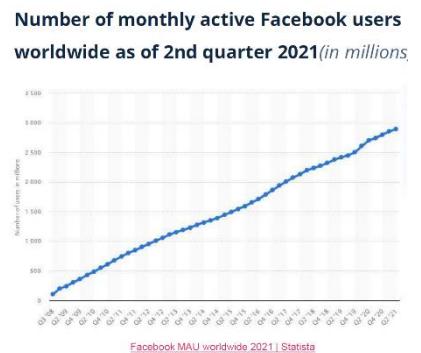
Mittels den Python Bibliotheken Mathplotlib und Seaborn (Moderner, Nutzerfreundlicher) können Daten visualisiert werden. Um daten zu Importieren ist Pandas eine sehr Hilfreiche Phyton-Bibliothek.

Ein Plot benötigt dringend nachfolgende Informationen:

- **Label der X-Achse:** Was für Daten werden in der X-Achse präsentiert?
- **Label der Y-Achse:** Was für Daten werden in der Y-Achse präsentiert?
- **Titel:** Was zeigt der Plot
- **Skala:** Es muss entschieden werden zwischen Linear und Logarithmisch. Je nach dem andere Aussagekraft und besser geeignet. *Beispiel Daten Moores Law: Transistoren pro Mikroprozessor. Hier ist logarithmisch besser geeignet, da es da den Trend der Verdoppelung besser zeigt. Linear sieht so aus als wäre lange nix passiert.*
- **Dimension der Daten:** 2D oder 3D ist das einzige was dargestellt werden kann und der Mensch auffassen. Mehr Dimensionen wie Wörtervektoren können nicht dargestellt werden oder müssen heruntergebrochen werden auf wenige Dimensionen

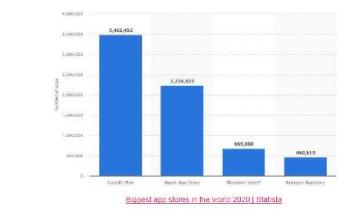
Liniendiagramm

- Bivariat, (kategorisch, kontinuierlich)
- Ein Trend ist definiert als ein Muster der Veränderung.
- Es handelt sich um die allgemeine Bewegung über Zeit einer statistisch nachweisbaren Veränderung.
- Die Linie verbindet mehrere unterschiedliche Datenpunkte und stellt sie als eine kontinuierliche Entwicklung. Das Ergebnis ist eine einfache, geradlinige Möglichkeit, Änderungen eines Wertes im relativ zu einem anderen Wert zu visualisieren.
- Beispiele: Aktienoptionspreise, Preis von Handys, Bevölkerung eines Landes, Einzelhandelsumsätze usw.



Balkendiagramm

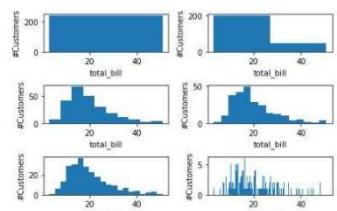
- Wird häufig für kategorische Daten verwendet, bei denen Zählung auf der Grundlage der einzelnen Kategorien erfolgt.
- Beispiel: die Anzahl der Apps in jeder Kategorie, d.h., GooglePlay, Apple Store, Windows Store



Histogramm

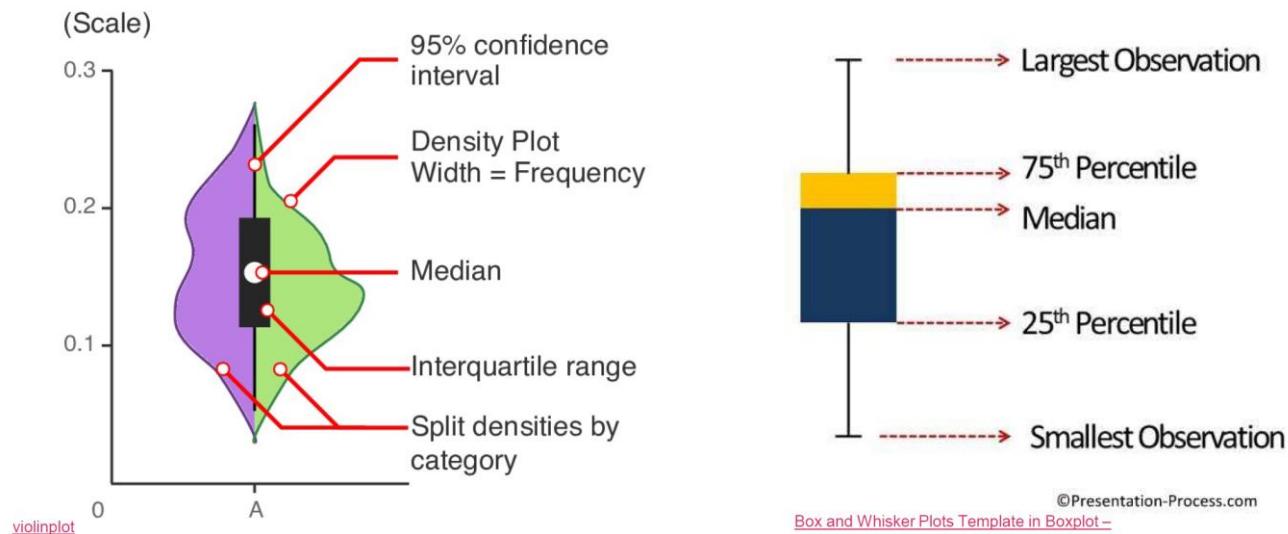
Ein Histogramm stellt die empirische Verteilung einer Variablen dar durch..

- ...das automatische erstellen von Bins (Intervalle) entlang des Wertebereichs
- ...anzeigen von vertikalen Balken zur Angabe der Anzahl der Beobachtungen in jedem Bin.



Box Plots und Violin Plots

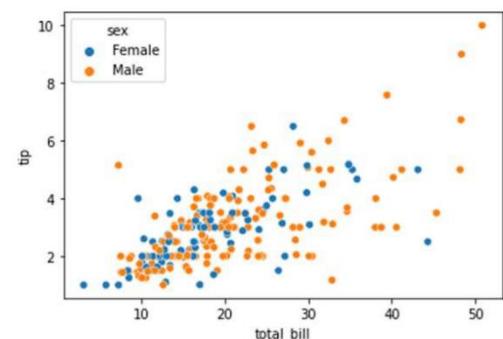
Anzeigen von Daten anhand der Aufteilung in Quartile und Median. Zeigt sehr gut Ausreißer und wo sich die meisten Daten befinden.



Studiogramm (Scatter Plot)

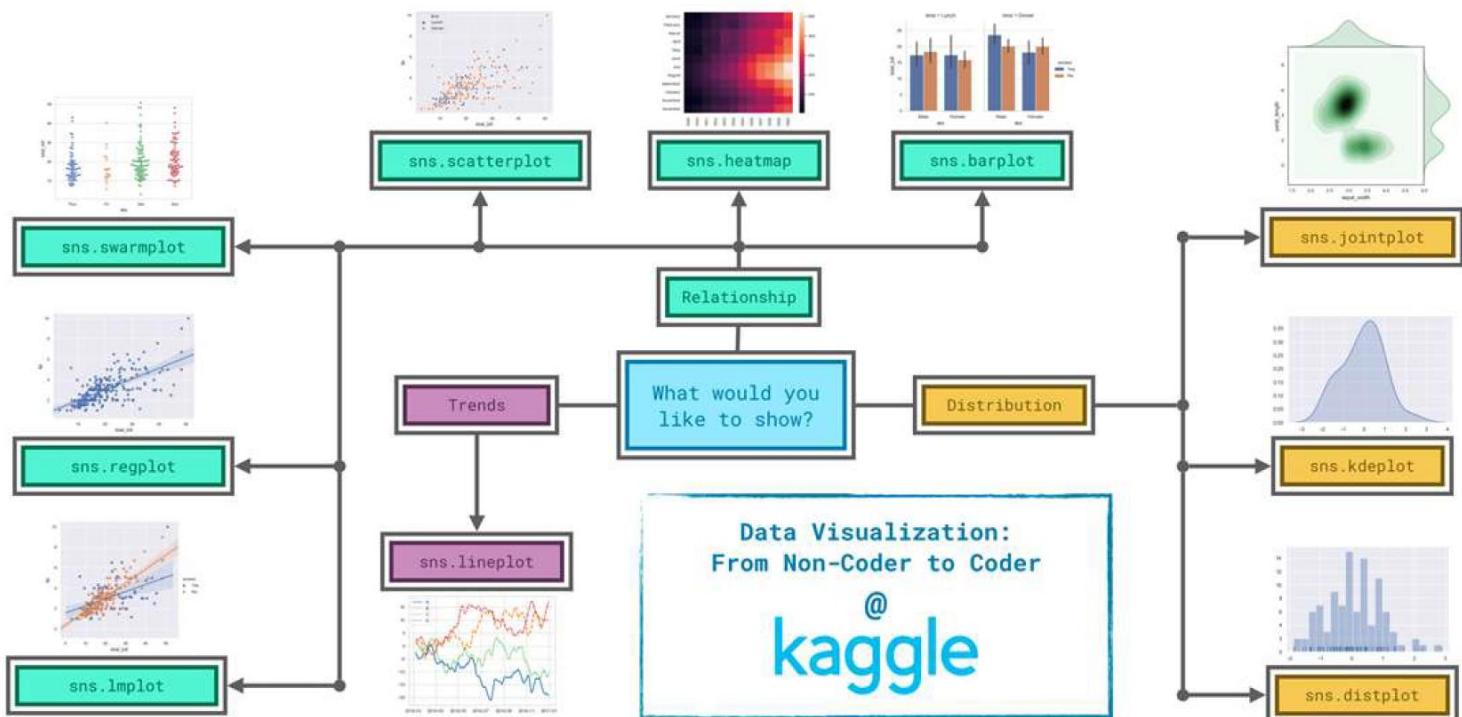
Zum Verständnis der Beziehung zwischen zwei kontinuierlichen Variablen kann ein Streudiagramm verwendet werden.

- Wir können Farbe verwenden, um eine dritte Variable zu kodieren
- Hilft dabei, eine Vorstellung vom Grad der Korrelation einer Variablen mit der anderen zu erhalten.



Wann welcher Plot?

Je nach Daten und was man zeigen will ist ein anderer Plot sinnvoll und aussagekräftiger.

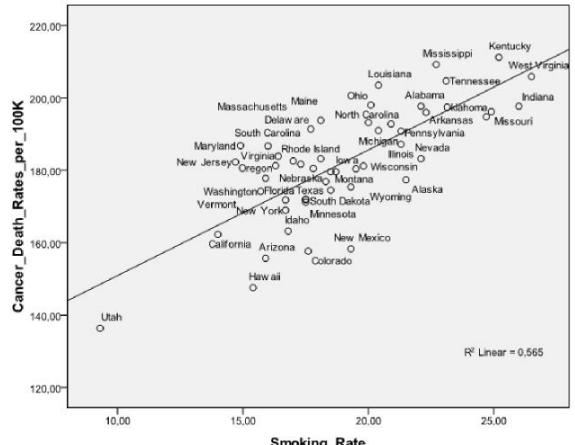


Woche 6: Linear Regression

Lineare Regression ist eine einfache Methode um Daten zu analysieren.

Man nutzt es hauptsächlich für:

- **Interpretation:** Verstehen ob ein bestimmter Input ein Effekt hat für den Output. Bsp: Gibt es eine Beziehung zwischen Raucher und Lungenkrebs?
- **Prediction:** Voraussehen wann was in Zukunft passieren könnte. Z.B. Wann geht eine Maschine kaputt anhand von Sensoren für Öldruck und Temperaturen.



Linear Regression und Machine Learning

Lineare Regression wird von Statistik «ausgeliehen». Hier ist es ein Beispiel von Algorithmen oder Techniken aus dem supervised learning. Wir haben also Inputdaten X und die Labels (In unserem Fall Y). Dann wird ein einfacher Algorithmus verwendet um daraus den Linearen Zusammenhang zu sehen. Das Modell soll also lernen Y vorauszusagen.

Model

In ML Model steht für irgendeine mathematische Funktion, welche die Daten erklärt. Z.B:

$$\begin{aligned} y_i &\approx f(x_i) \\ y_i &= f(x_i) + \varepsilon_i \end{aligned}$$

Das ε_i steht für «unerklärtes Rauschen». Man nimmt an, dass die der normalen Distribution folgt.

Die Funktion f kann beliebig kompliziert sein (Konstante oder Multi-Millionen-Parameter-Netzwerk). Machine Learning muss das korrekte Modell finden, welche die Daten möglichst gut interpretiert. Anstelle die Funktion näherungsweise zu bestimmen bestimmen wir ein y -Hut, welche eine Schätzfunktion von y_i ist.

$$\hat{y}_i = f(x_i)$$

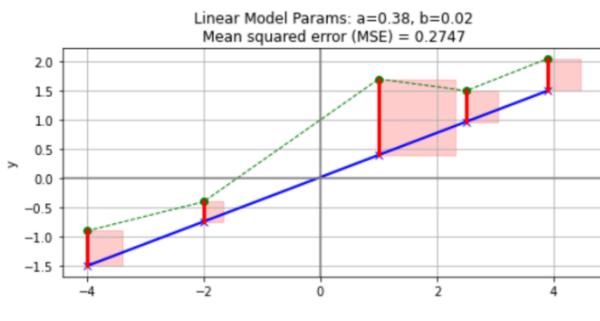
In der Linearen Regression wird nur die Lineare Repäsentation angeschaut zwischen input und output. Vor dem Lernen vom Algorithmus bestimmen wir den Model-Space (Funktionenschar). In einfachsten Fall sind x und y skalar und wir haben nur zwei frei wählbare, unbekannte Variablen a und b. Diese müssen wir nun mit Machine Learning bestimmen um möglichst nah an den Daten zu sein.

$$\hat{y}_i = a \cdot x_i + b$$

A wird meist **slope** benannt und b **intercept**.

Mean squared Error MSE (Loss, Residuals)

Dies ist das **Loss** das wir minimieren wollen. (Hinweis: MSE ist normalerweise mit 2 dividiert)



$$\hat{y}_i = a \cdot x_i + b$$

$$e_i = y_i - \hat{y}_i$$

$$\begin{aligned} E &= \frac{1}{2N} \sum_{i=1}^N e_i^2 \\ &= \frac{1}{2N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))^2 \end{aligned}$$

Die Abweichung wird auch **Residuals** (e_i) genannt.

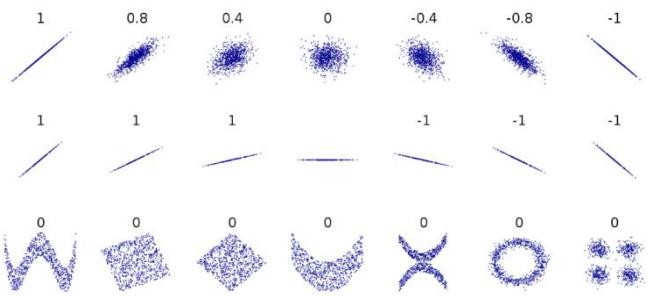
Korrelation und Kausalität

Korrelation ist NICHT Kausalität!!

Korrelation sagt aus wie zwei Variablen linear zueinander stehen. Dies kann mit Linearer Regression detektiert werden. Es gibt den Pearson Correlation Coefficient:

Auch wenn der Koeffizient 0 ist können die Daten strukturiert sein, wie man an der dritten Linie sieht.

Wichtig: Keine Aussagen über Kausalität machen. Immer Daten visualisiere und ebenso die Residuals.



Komplexere Lineare Modelle

More complex linear models

- We can have linear models that depend on more than one input variable. In this case, x is a vector with p features (=dimensions)

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \dots + \beta_p x_{ip}$$

- For many data (x_i, y_i) we can express the linear model in matrix notation:

$$X\beta = y,$$

where

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Uniform Distribution, Normal Distribution, numpy's random number Generator

Die Gleichverteilung (Uniform Distribution) beschreibt eine Zufallsvariable, die jeden Wert innerhalb eines bestimmten Intervalls annehmen kann. Jeder Wert in diesem Intervall ist gleich wahrscheinlich.

Bei der Normalverteilung (Gaussian Distribution) ist die Zufallsvariable an der Normalverteilung angelehnt.

```

rng = default_rng()
# generate a uniform random distribution.
samples = rng.uniform(-5,5,100)

# generate a simple random number.
a_random_number = rng.normal(loc=0, scale=1, size=1)
print(a_random_number)

# generate 5 samples from Gaussian distribution with mean=100, standard-deviation = 20
many_random_numbers = rng.normal(loc = 100, scale = 20, size=5)
print(many_random_numbers)

```

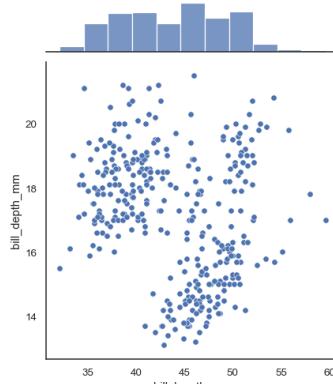
Seaborn Jointplot

Der Jointplot ist eine Verbindung von zwei Variablen mit bivariate und univariaten Graphen. Es zeigt die (empirischen) Randverteilungen als Histogramme. Das Ergebnis ist ideal: die Daten (x -Werte) sind gleichmäßig über den gesamten Bereich verteilt (Gleichverteilung in -50/+50), während die Residuen bei 0 zentriert sind (Standardnormalverteilung).

```

penguins = sns.load_dataset("penguins")
sns.jointplot(data=penguins, x="bill_length_mm", y="bill_depth_mm")

```



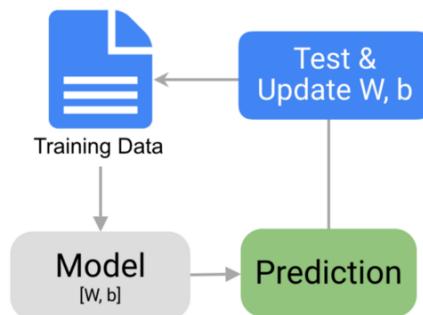
Woche 7: Grundbegriffe

In dieser Woche gab es keine Online-Vorlesung wegen Feiertag. Es gab zwei Videos zu schauen:

7 Steps of Machine Learning

<https://www.youtube.com/watch?v=nKW8Ndu7Mjw>

1. **Gathering Data:** Daten Sammeln, die fürs Training und Evaluieren genutzt werden können
2. **Data Preparation:** Daten randomisieren und zusammenstellen. Ebenso visualisieren um zu schauen ob komplett genug. Weiter müssen Daten gesplittet werden in Training Data und Evaluation Data (80%/20% oder 70%/30%). Somit kann man mit Daten Testen, welche nicht im Training vorhanden waren
3. **Choosing a model:** Ein Modell auswählen das passt (Musik, Text, Zahlen, Linear)
4. **Training:** Mit den Trainingsdaten das Modell trainieren. Dabei immer wiederholen bis zufrieden



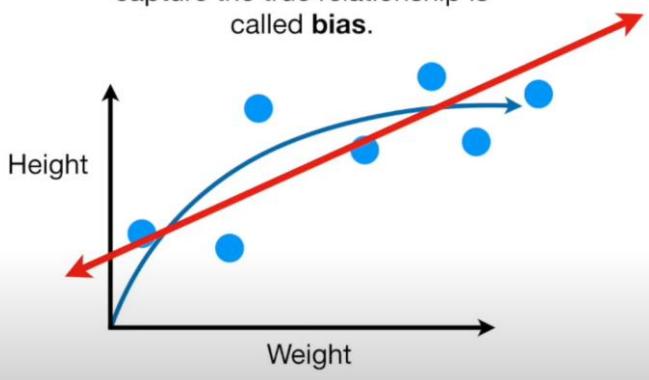
5. **Evaluation:** Mit Evaluationsdaten testen ob Training gut genug.
6. **Parameter Tuning:** Experimentieren mit weiteren Parametern um Training zu verbessern. «Hyperparameter»
7. **Prediction:** Modell nutzen um nun mit anderen Daten eine Aussage machen zu können (Modell anwenden)

Machine Learning Fundamentals: Bias and Variance

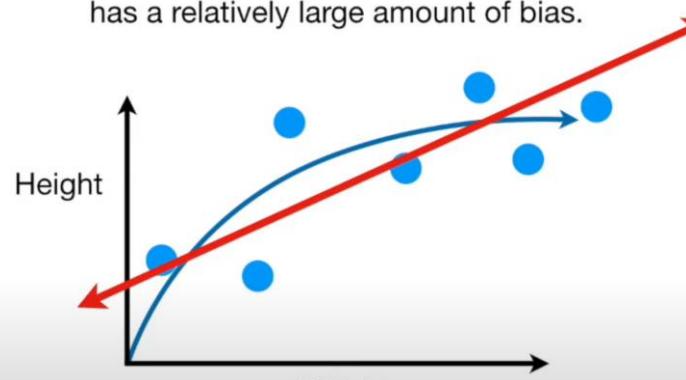
<https://www.youtube.com/watch?v=EuBBz3bl-aA>

Bias

The inability for a machine learning method (like linear regression) to capture the true relationship is called **bias**.



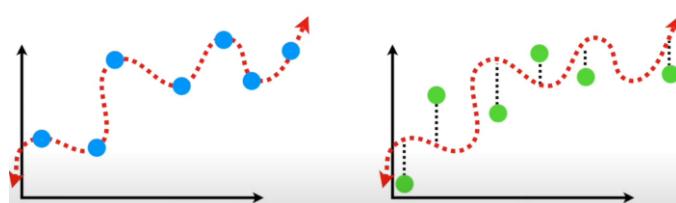
Because the **Straight Line** can't be curved like the "true" relationship, it has a relatively large amount of bias.



Variance

Differenz der Nähe der Daten (Sum of Squares) zwischen Test und Evaluationsdaten wird Varianz genannt.

In Machine Learning lingo, the difference in fits between data sets is called **Variance**.



Woche 8: Optimization: Stochastic Gradient Descent (SGD)

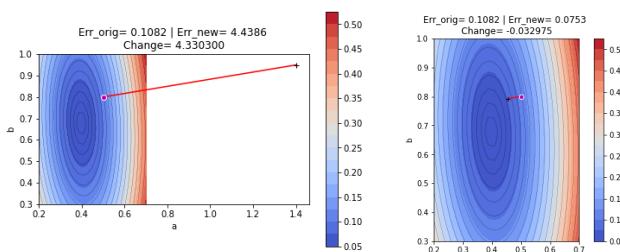
Fortsetzung von Woche 6. Model sowie die Loss Function bleibt die gleiche. Nur wollen wir nun eine andere Optimierungsfunktion. Wir nutzen dieses mal den Gradient Descent. Nachfolgend die Error-Function der linearen Gleichung:

$$\begin{aligned} E &= \frac{1}{2N} \sum_{i=1}^N e_i^2 \\ &= \frac{1}{2N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))^2 \end{aligned}$$

Der Gradient ist folgender (Ableitung einmal nach Variable a und einmal nach b und dies in einen Vektor platzieren):

$$\text{Gradient of } E = \begin{bmatrix} \frac{\partial E}{\partial a} \\ \frac{\partial E}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))(-x_i) \\ \frac{1}{N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))(-1) \end{bmatrix}$$

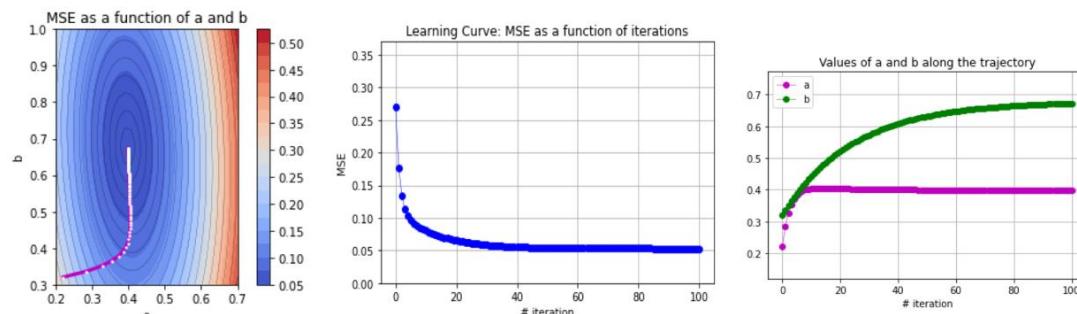
Problem hier: Der normale Gradient dient dazu von einem Tal auf einen Berg zu kommen, also vom Minimum zum Maximum, zudem macht er noch zu grosse Schritte (Bild 1). Dies lassen wir indem wir noch ein alpha hinzufügen und die Werte negieren (Bild 2):



Um nun weiterzukommen und immer näher ins Tal zu kommen zum minimalen Error, müssen wir nun am neuen Punkt wieder den Gradient berechnen:

$$\begin{bmatrix} a \\ b \end{bmatrix}_{t+1} = \begin{bmatrix} a \\ b \end{bmatrix}_t - \alpha \begin{bmatrix} \frac{\partial E}{\partial a} \\ \frac{\partial E}{\partial b} \end{bmatrix} \Big|_{\begin{bmatrix} a \\ b \end{bmatrix}_t}$$

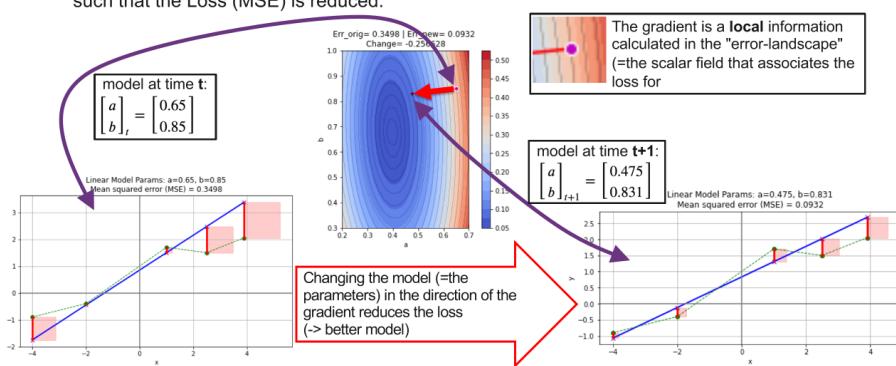
Dies geht so lange weiter bis man eine bestimmte Anzahl Iterationen erreicht hat, die vorgegeben wird. Daraus ergibt sich eine Learning-Kurve. In diesem Beispiel ist schon nach 20 iterationen das Ziel beinahe erreicht und nur noch am b wird optimiert.



Hinweis: Hat man mehr als ein Tal, dann muss man einfach an mehreren Punkten starten und so versuchen das Globale minimum zu finden (Mehrere Punkte unterschiedliches Minimum, dann den tiefsten nehmen).

Hier noch ein Bild, welches die Idee noch veranschaulicht:

- Gradient Descent is an iterative method. At each iteration, the model parameters are updated such that the Loss (MSE) is reduced.



Stochastic Gradient Descent (SGD)

Hier ist das Ziel, dass man nur eine kleine Menge an Datenpunkten nimmt um die Fehler und so den Gradient zu berechnen.

Man selektiert also random ein paar Datenpunkte aus dem Dataset ($J = \text{Teildatenset}$):

$$E = \frac{1}{2N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))^2$$

$$\frac{\partial E}{\partial a} = \frac{1}{N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))(-x_i)$$

$$\approx \frac{1}{n} \sum_{j \in J_n} (y_j - (a \cdot x_j + b))(-x_j)$$

$$\frac{\partial E}{\partial b} = \frac{1}{N} \sum_{i=1}^N (y_i - (a \cdot x_i + b))(-1)$$

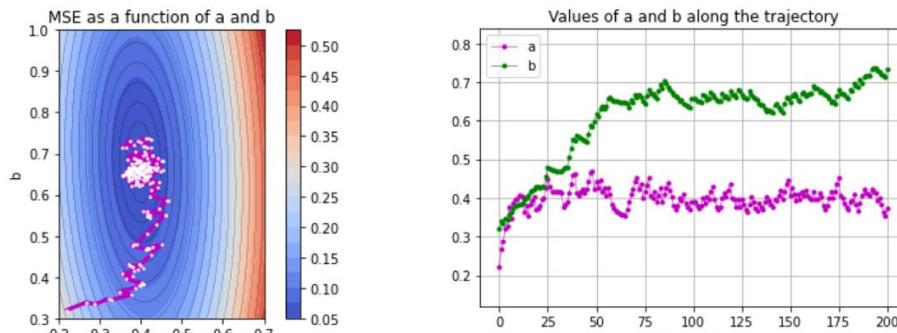
$$\approx \frac{1}{n} \sum_{j \in J_n} (y_j - (a \cdot x_j + b))(-1)$$

Dies führt zur folgender Update-Regel:

$$\begin{bmatrix} a \\ b \end{bmatrix}_{t+1} = \begin{bmatrix} a \\ b \end{bmatrix}_t - \alpha \begin{bmatrix} \frac{\partial E_j}{\partial a} \\ \frac{\partial E_j}{\partial b} \end{bmatrix} \Big|_{\begin{bmatrix} a \\ b \end{bmatrix}_t}$$

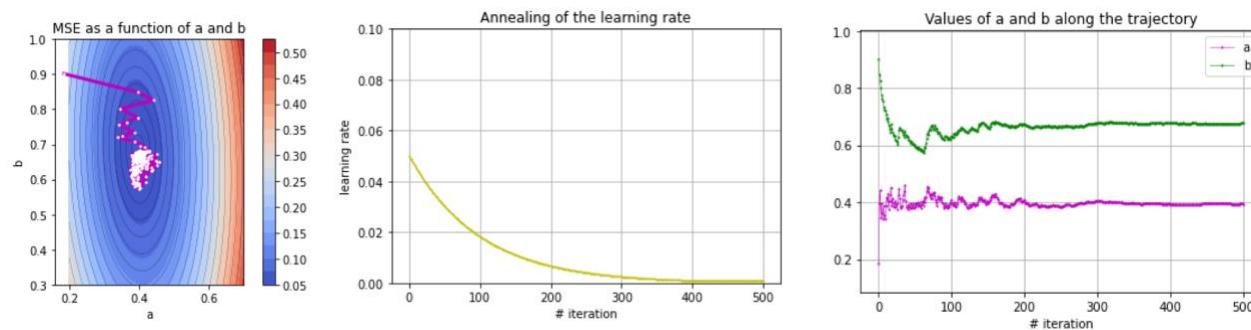
Nun versucht man den Batch auf 1 zu wählen oder möglichst klein (32 oder 64 werden häufig verwendet).

Hinweis: Bei Batch 1 kann es sehr grosse Zicksacke geben, da ja für einen Punkt mehr als eine Lösung gibt und MSE auch 0 sein kann. Dennoch kommt man irgendwann auf eine Lösung, doch es gibt Zickzackwerte bei der Lerning-Kurve. Und nahe beim Ziel geht er wieder weiter weg. Dieses Problem kann aber auch gelöst werden indem man Alpha immer kleiner wählt.



Stochastic Gradient Descent with annealed learning rate

Wie oben erwähnt gibt es eine grosse Streuung am Ende und man erreicht nicht mehr so genaue Werte. Dem kann man entgegenwirken indem man immer alle paar Iterationen alpha verkleinert. Somit flacht es nach ein paar Iterationen immer mehr aus und man kommt anfangs sehr schnell in die Mitte:



Allgemeine Anmerkungen

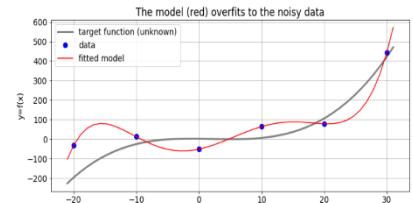
- Diese Methode geht nur, wenn die Loss-Function Ableitbar ist.
- SGD kann nur mit einem Datenset arbeiten, was sehr ineffizient ist und selten benutzt wird
 - o Batch oder Minibatch-Gradient ist also besser
- SGD muss den Gradient berechnen. Ist dies schwer?

- JA, wenn man es selbst tun muss und es viele Parameter gibt. (Neurale Netze können Millionen Parameter haben)
- Nein, wenn man ein Modell hat und ein Machine Learning Framework nutzt (Tensorflow, Pytorch).
- Gradient Descent ist ein Basis Baustein für viele weitere Algorithmen wie Adam, Adagrad, RMSProp, etc.

Woche 9: Generalisation & Regularisierung

Overfitting

Overfitting ist wenn alle Punkte genau getroffen werden. Dann stimmt es zwar für die Trainingsdaten genau, aber sobald weiter Datenpunkte dazukommen stimmt es für diese nicht mehr. Wir haben zwar einen MSE von 0, dies ist aber ein **In-Sample Error** (Training Error). Der Loss wurde sehr stark minifiziert in der Trainingsphase.

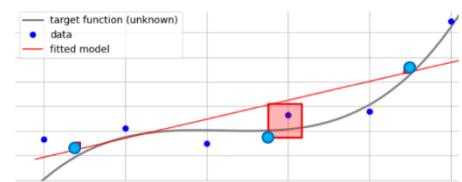


Sobald wir dann aber die Testdaten dazu nehmen (Neue noch nicht gesehene Daten, haben wir dann einen grossen **Out Of Sample Error** (Generalization Error). Ein gutes Modell hat ein tiefen Generalization Error.

Ziel ist also ein etwas grösserer In Sample Error, dafür ein tieferer Generalization Error.

Underfitting

Underfitting tritt dann auf, wenn das Modell zu einfach ist. Wir haben ein viel zu hoher In Sample Error, als auch ein zu grosser Generalization Error.

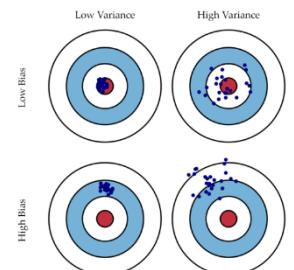


Generalization Error

Der Generalization Error kann nicht berechnet werden, sondern nur geschätzt werden. Wir müssen mit den Daten arbeiten, die wir haben. Deshalb müssen wir unsere Daten splitten in Test- und Trainingsdaten. Ein guter Split ist 80% Training und 2% Test. So nutzt man die Training-Daten um das Modell zu trainieren und wenden dann mit den Testdaten das Modell an. Der Test-Fehler ist eine Schätzung vom Generalization Fehler.

Bias und Varianz

Wenn wir die beiden Fehler genauer mathematisch analysieren können wir zwei Werte feststellen:
Bias und Varianz.



High Bias, Low Variance

In diesem Fall ist das Modell zu einfach. Egal wie viele Daten wir haben, das Modell gibt keine besseren Ergebnisse. Zum Beispiel wir nehmen ein Lineares Modell obwohl es passendere gibt. Dann ist dieses zu einschränkend. Die rote Linie wird bei vielen Daten gleich bleiben auch wenn ich ein neuer Punkt hinzufüge. Wir haben underfitting.

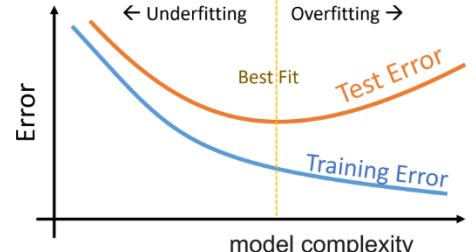
Low Bias, High Variance

Man nimmt ein (zu) komplexes Modell. Es kann die Daten besser beschreiben. Beispiel 6 Punkte und wir nehmen ein Polynom Grad 5. Hier passt das Modell genau zu den Testdaten. Aber wir haben nun Overfitting.

Die Varianz ist hoch. Ein weiter Datenpunkt führt dann zu einem sehr grossen Generalization error. Und sobald ein Punkt etwas verschoben ist, ist die Line komplett eine andere.

Regularisierung

Unser Ziel: Wir wollen möglichst eine tiefe Varianz. Der Bias ist eher sekundär, sollte aber auch möglichst tief sein. Man braucht aber die optimale Balance. Am besten schauen wir da den Training- und Testerror genauer an. Es gibt einen Punkt wo der Testerror am tiefsten ist und dann wieder steigt je genauer das Modell ist. Dazu verwenden wir Regularisierung.



Regularisierung fügt Constraints hinzu, ein Penalty-Term (Cost-Function). Der Optimizer optimiert die Daten (MSE minimalisieren) sowie auch den Constraint.

Es gibt zwei Zutaten die wir benötigen:

1. Weg zum Modell-Komplexität zu messen

- L1 Norm (Lasso)

$$\sum_{j=1}^p |\beta_j|$$

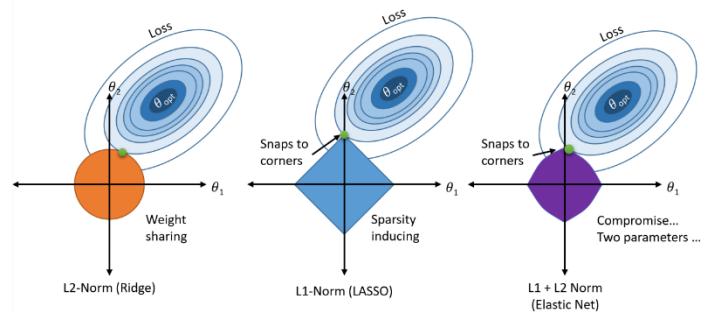
- L2 Norm (Ridge)

$$\sum_{j=1}^p \beta_j^2$$

2. Weg zum Model-Komplexität zu kontrollieren

- Penalty-Term wird zu Loss hinzugefügt
- Komplexere Modelle haben höhere Penalty
- Constrain wird zum Optimierungsprozess hinzugefügt

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$



«Lamda ist der Hyperparameter»

Lamda = 0: dann haben wir kein Constraint also nur MSE. Vergrössern von Lamda: zu grosse Bias, kleinere Varianz

Umsetzung mit Keras/Tensorflow

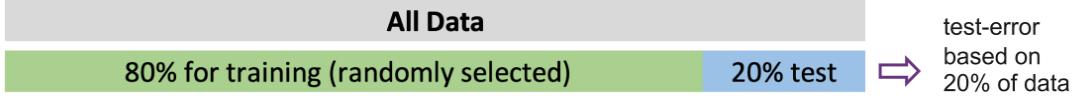
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
model = tf.keras.models.Sequential()

# define the input. we provide the dimensionality of the input: poly_order
inputs = keras.Input(shape=(poly_order_MODEL,))
model.add(inputs)
output_layer = layers.Dense(1, activation=None, use_bias=True,
kernel_regularizer=regularizers.l2(0.005))
model.add(output_layer)

model.compile(
    optimizer=keras.optimizers.Adam(Learning_rate=0.04), # Optimizer
    # Loss function to minimize
    Loss=keras.losses.MSE,
    # List of metrics to monitor
    metrics=[keras.metrics.MSE]
)
```

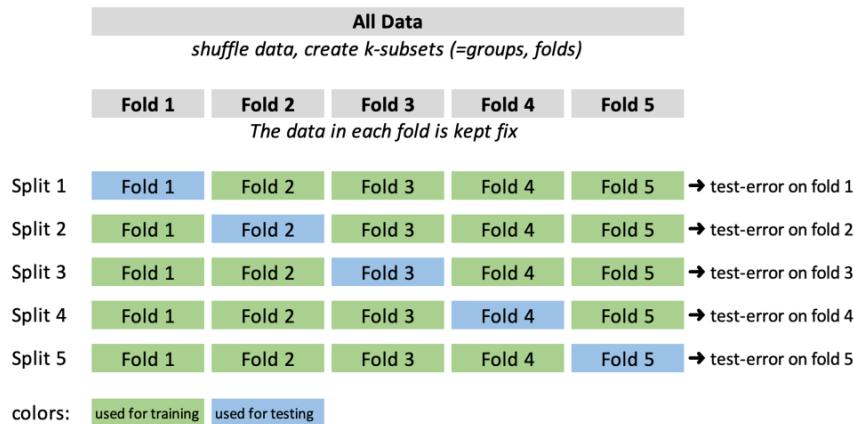
Woche 10: Cross-Validation

Ohne Cross-Validation: Test-Fehler basiert auf 20% der Daten.



Funktionsweise der k-Fold-Crossvalidation: Daten werden zuerst geshuffelt. Anschliessend aufgeteilt in k-Fold (Üblich: 5-Fold, 10-Fold. Gibt aber auch N-Fold wo jeder Fold nur ein Datenpunkt hat). Dann wird jedes mal ein anderer Fold als Test-Datensatz genommen und das Modell trainiert.

Hinweis: Daten in einem Fold ändern sich während den einzelnen Splits nicht. Diese bleiben über die ganzen Trainings gleich. Weiter muss die Preprocessing-Pipeline während den einzelnen Splits laufen und nicht davor, da sonst Ergebnisse verzerrt werden.

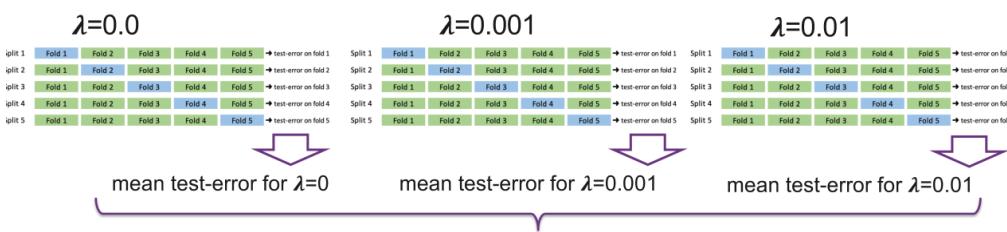


Ziel 1: Generalisierungsfehler Schätzen

Hier wird aus den einzelnen Testfehlern ein durchschnitt genommen und so der Mean und die Variance des Testfehlers genommen. Ist genauer als mit einzelnen Tests.

Ziel 2: Auswahl von Hyper-Parameter

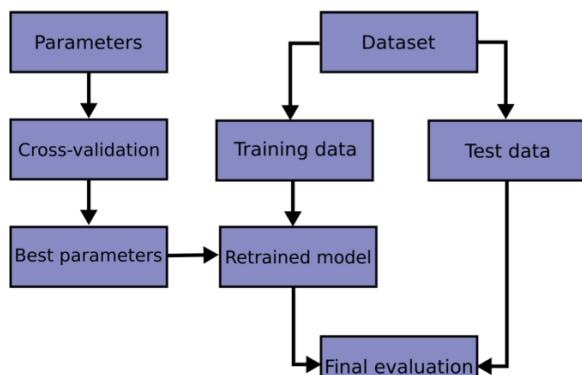
Man variiert mit den Hyper-Parameter (Regularisierungslamda, etc.). Findet so den optimalen Parameter heraus:



1. select the optimal λ_{opt}
2. Using λ_{opt} , retrain the model on all training data (but see not on next slide)

Scikit-learn Cross-Validation

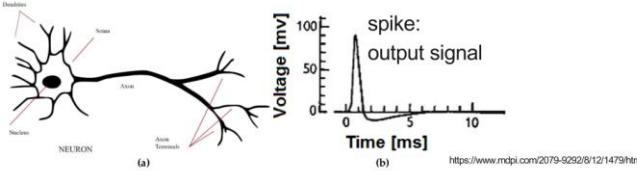
Einzelne Daten werden vorher ganz rausgenommen als Testdaten. Dann wird mit anderen Cross-Validation gemacht. Ist meist empfohlen.



Woche 10: Artificial Neural Networks

AI ist inspiriert von unserem Gehirn und den Neuronen. (Gogli hat erstmals Neuronen sichtbar gemacht durch staining in 1873)

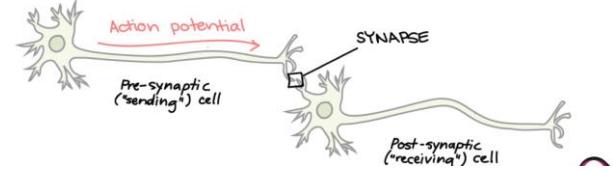
Biological Neurons integrate and communicate information



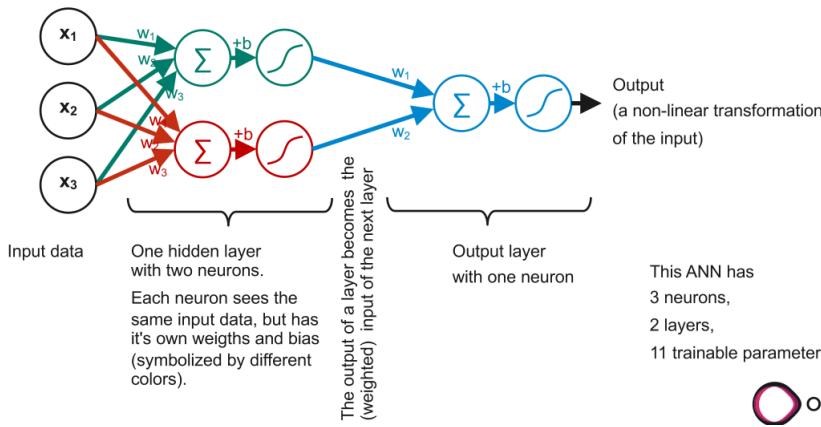
- Dendrites: Collect (chemical) **input** from (many) other neurons.
- Cell Body (Soma):** If the collective input signal reaches a (voltage) threshold, an **output** signal, called a "spike" is generated.
- Axon: the electrical pulse (spike) travels along the axon to other neurons.

Synapses: Where neurons connect (and learning happens)

- The human brain has $\sim 10^{11}$ neurons (100 billions).
- Each neuron receives input from up to 10^4 other neurons. The connections are called synapses. The human brain has trillions of synapses.
- When a brain "learns", the synapses are strengthened (they can grow).



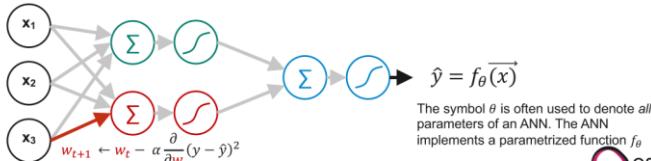
Die Biologischen Neuronen wurden vereinfacht in ein Artificial Neuron. Diese bieten ein Input-Vektor. Jedes Neuron hat seine eigenen Input-Gewichte und ein Bias (Intercept). Sie berechnendie Summe von den gewichteten Inputs (Skalarprodukt), fügen Bias hinzu und passen es in eine nichtlineare Aktivierungsfunktion. Nachfolgend ein Beispiel eines einfachen Netzes:



Hat ein Netz zahlreiche Hidden Layer, dann wird von einem Deep Neural Network gesprochen.

How to train an ANN?

- We consider a simple case of **supervised learning**: for each input \vec{x}_i we are given the output y_i (know outputs are usually called **target values or labels**)
- The ANN is initialized with random weights and produces some output \hat{y} . Then, an optimizer (e.g. SGD) reduces a cost-function (e.g. MSE).
- That is, at every iteration, and for every single weight w (and every bias b), the partial derivative $\frac{\partial}{\partial w} (y - \hat{y})^2$ needs to be calculated. Luckily there's an algorithm which is doing this very efficiently: **Backpropagation**. (We will not study BP here, it's used behind the scenes in Keras)



ANN's from a computer science perspective

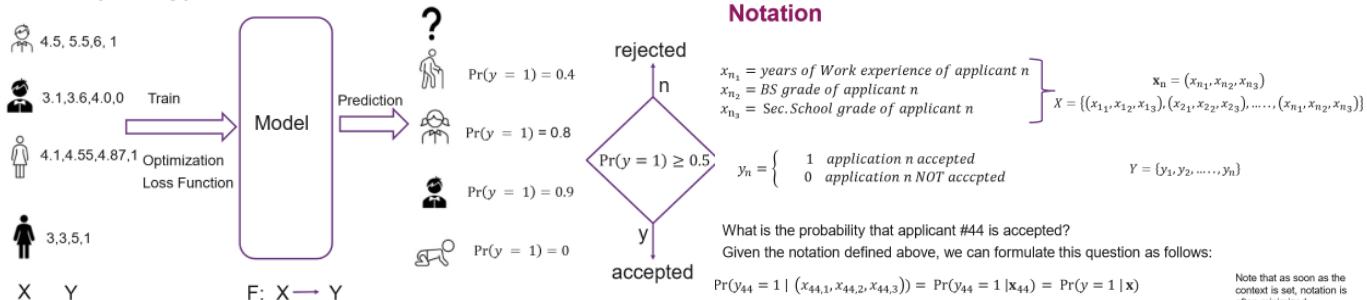
- An ANN is a **data-structure** to define arbitrarily complex mathematical functions.
- ANNS are composed of (many) relatively simple expressions.
- An ANN is an expression-tree. Each node can be evaluated.
- Finding the optimal weights of an ANN requires calculation of the gradient (partial derivatives w.r.t. every single parameter). Backpropagation (aka Backprop) is an efficient **algorithm** for automatic differentiation of ANNs.
 - From the math perspective, Backpropagation is basically the chain rule (german: Kettenregel)
 - From the CS perspective, Backpropagation is a recursive, Dynamic-Programming algorithm.
 - Without the efficiency of Backprop, we would not see deep learning.
- There are software packages (e.g. Keras or Pytorch) that make the creation and training of neural networks extremely efficient and relatively simple.

Woche 11: Logistic Regression

In dieser Woche ging es um Daten die Binär klassifiziert werden. (JA-Nein Fragen). Beispiel: Wird es Hageln aufgrund des Satellitenbild. Oder hat jemand bald ein Epilepsie-Anfall wegen einem EEG reading.

Bei solchen fällen sehen die Daten etwas anders aus, weshalb wir ein anderes Modell und Loss-Funktion brauchen. (Lineare Funktion geht nicht) Für den Optimizer können wir aber weiterhin Gradient Descent verwenden, da die andere Loss-Funktion convex ist.

Binäre Klassifikation und deren Notation

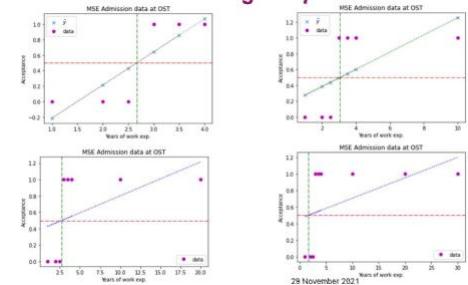
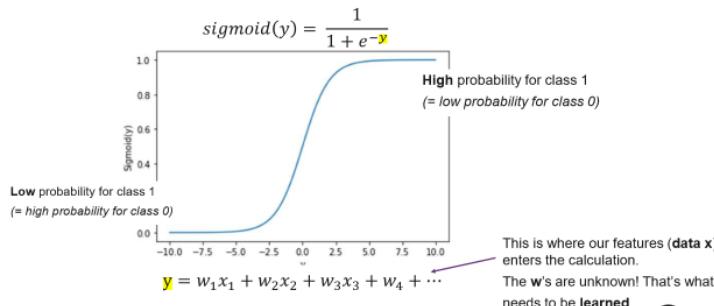


Warum nicht Lineares Modell?

Problem der Linearen regression ist, dass die Kurve ganz anders aussieht, sobald ein weiterer Datensatz dazu kommt und Tresholding hier falsche Ergebnisse liefern kann. Die Kostenfunktion (Loss) Min Squared Error funktioniert hier gar nicht und liefert diese falschen, verzerrten Ergebnisse. Wir sind aber interessiert in eine Wahrscheinlichkeits-Output. Somit brauchen wir ein Modell, dass die Wahrscheinlichkeit abbildet.

Sigmoid Funktion

Die Lösung ist die Sigmoid-Funktion. Sie bietet genau das benötigte Modell:



Das y bzw. manchmal auch z kann alle Dimensionen enthalten. Z.B

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \dots$$

Maximum Likelihood (Loss)

Die Lösung um das W zu finden. (negativer log-loss) Dazu können wir auch Gradient descent benutzen, da die Funktion convex ist.

Maximum Likelihood

- Given all the data points (X, Y) , we want to maximize the probability that all the predictions are correct (or minimize the probability that all predictions are wrong!).
- For each of the training data, we want to **maximize the likelihood** of correct prediction
- The objective of training is to set the coefficients W so that p is close to 1 when $y = 1$ and p is close to 0 when $y = 0$

Reminder:

$$\Pr(y = 1 | x) = 1 - \Pr(y = 0 | x)$$

$$\text{Maximize cost}_1(W) = \prod_{i:y_i=1}^N p(x_i) \prod_{i:y_i=0}^N 1 - p(x_i)$$

$$\text{Maximize cost}_2(W) = \sum_{i:y_i=1}^N \log(p(x_i)) + \sum_{i:y_i=0}^N \log(1 - p(x_i))$$

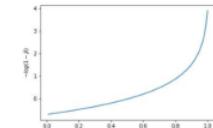
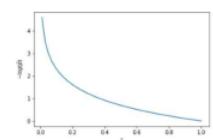
$$\text{Minimize cost}_3(W) = - \left(\sum_{i:y_i=1}^N \log(p(x_i)) + \sum_{i:y_i=0}^N \log(1 - p(x_i)) \right)$$

$$\text{Minimize cost}(W) = \frac{-1}{N} \sum_{i=1}^N (y_i \cdot \log(p_i)) + (1 - y_i) \cdot \log(1 - p_i)$$

Taking a closer look at the cost

$$\square -\log(p) \begin{cases} \text{large when } p=0 \text{ and } y=1 \\ \text{small when } p=1 \text{ and } y=1 \end{cases}$$

$$\square -\log(1-p) \begin{cases} \text{large when } p=1 \text{ and } y=0 \\ \text{small when } p=0 \text{ and } y=0 \end{cases}$$



Woche 12: Classifier Evaluation

Wir stellen uns die Frage ob das Model gut genug ist, weshalb wir das Modell genauer evaluieren. Dazu nutzen wir unter anderem eine Confusion Matrix.

Confusion Matrix:

- **False Positive:** Der berechnete Wert ist 1 und der wahre Wert ist 0. (Fälschlicherweise richtig)
- **False Negative:** Der berechnete Wert ist 0 und der wahre Wert ist 1. (Fälschlicherweise falsch)
- **True Negative:** Der berechnete Wert ist 0 und der wahre Wert ebenso. (Korrekt)
- **True Positive:** Der berechnete Wert ist 1 und der wahre Wert ebenso. (Korrekt)

- **Mean Accuracy:**
How often is the classifier correct?
$$\text{Accuracy} = (t_p + t_n) / n$$
- **Mean Error:**
How often is the classifier wrong
$$\text{Error} = (f_p + f_n) / n$$
- **Precision:**
When the prediction is "1", how often is it correct?
$$\text{Precision} = t_p / (t_p + f_p)$$
- **Sensitivity, Recall, True Positive Rate(TPR):**
How often the prediction is "1", when it's actually "1"
$$\text{Recall} = t_p / (t_p + f_n)$$
- **Miss Rate, False Negative Rate(FNR):**
$$\text{Miss Rate} = 1 - \text{TPR}$$

		Predicted condition		
		Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP),	False negative (FN),	
	Negative (N)	False positive (FP),	True negative (TN),	

Source: [Wikipedia](#)

6 December 2021



Bei oben genannten Formeln ist die Mean Accuracy nicht genug um bestimmen zu können, ob wir ein gutes Modell haben. Beispiel: Dataset wo 90% Nein und 10% Ja sind. Das Modell sagt nun einfach immer Nein. Hier ist dann die Accuracy bei 90%. Nun schauen wir die weiteren Werte an Precision und Recall. Diese sind in diesem Modell jeweils 0...

Precision vs Recall

Es kommt drauf an worauf man den Fokus legt je nach Applikation. Nachfolgend die Folie dazu:

- Increasing precision reduces recall and vice versa
 - Decided by the application what is desired.
 - Application 1: classify videos safe for kids (Kids Youtube).
 - Low recall (rejects many safe videos), i.e., high precision (keeps only safe ones)
 - High recall (rejects many safe videos), i.e., low precision (keeps unsafe videos)
 - Application 2: classify shoplifters from a surveillance camera
 - Low precision but High recall (false alerts are ok)
- **Precision:**
When the prediction is "1", how often is it correct?
$$\text{Precision} = t_p / (t_p + f_p)$$
 - **Sensitivity, Recall, True Positive Rate(TPR):**
How often the prediction is "1", when it's actually "1"
$$\text{Recall} = t_p / (t_p + f_n)$$

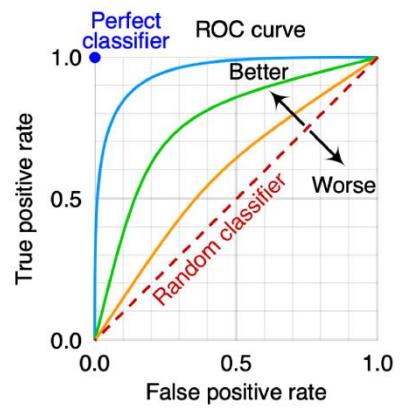
Threshold

Was der Threshold angeht gibt es keine allgemeingültige Lösung mittels Mathematik.

Hier kommt es wieder auf die Applikation drauf an. (Threshold ab wann True/False).

Vorgehen um das Threshold zu bestimmen:

1. Modell trainieren
2. Predictions machen mit dem Test-Set
3. Versuchen unterschiedliche Threshold-Werte zu verwenden und damit folgendes Berechnen:
 - a. True Positive Rate (Wie oft richtig akzeptiert) $TPR = t_p / (t_p + f_n)$
 - b. False Positive Rate (Wie oft falsch akzeptiert) $FPR = f_p / (f_p + t_n)$
4. Wir erhalten unterschiedliche TPR und FPR Werte. Diese können wir Plotten → Ergibt ROC Space (Anderer Plot mit Precision und Recall könnte ein Schnittpunkt ergeben, was sicher ein sehr guter Punkt ist)



Woche 12: K-Nearest-Neighbours KKN

Haben wir Plots, welche mehr als zwei Klassen haben brauchen wir eine weiter Variante und Logistic Regression wird hier nicht funktionieren. Weiter wird ein Lineares Modell nicht zufriedenstellende Ergebnisse liefern und nicht überall funktionieren:

Decision Boundary, linearly separable data

- For the sake of simplicity, let us use **two features** for classification in our admission data (bsc grades and workex).

- Using sklearn's logistic regression, we get the following model

$$p = \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}} \text{ where } w_0 = 8.87, w_1 = 1.5, w_2 = 1$$

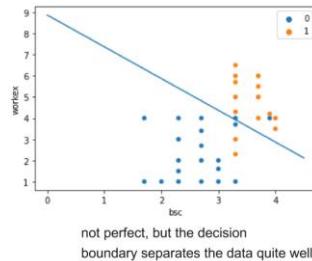
- For a threshold of $p = 0.5$

- $w_0 + w_1x_1 + w_2x_2 = 0$

- This results in a line $x_2 = 8.87 - 1.5x_1$

- This line is the **linear decision boundary**!

- If a simple line perfectly separates the classes, then the classes are said to be **linearly separable**!



Non-linear decision boundary?

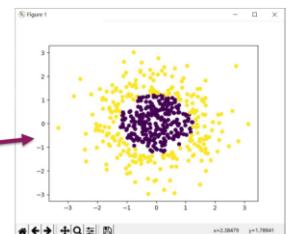
- What to do when the classes are NOT linearly separable!

- If we have additional information about the structure of the data, we can apply a non-linear transformation.

- Non-linear decision boundaries!

$$p = \frac{1}{1+e^{-(w_0+w_1x_1^2+w_2x_2^2)}}$$

- We resort to polynomial terms. e.g., the following logistic regression term might result in a good classifier for the dataset



Beispiele für mehrere Klassen: SARS-CoV-2 Varianten (Alpha, Beta, Gamma, etc.), Wetter (Sonnig, Wollig, Regen, Schnee)

Logistic Regression kann hier funktionieren, ist aber sehr umständlich. Man muss viel trainieren (Rekursiv), was langsam ist. So braucht es eine Methode ohne training, mit Predictions, einfacher Support mehrere Klassen, und auch möglichkeit Nicht-Lineare-Modelle zu trainieren. Hier kommt KKN ins Spiel.

Funktionsweise

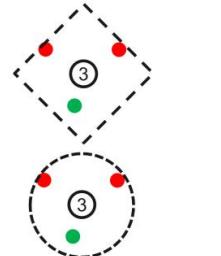
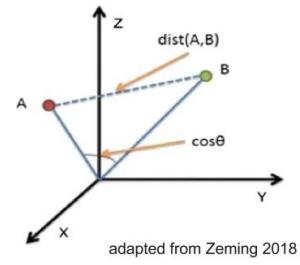
- Training und Testdaten Laden
 - Value von k bestimmen (Nummer von Nachbarn, welche berücksichtigt werden sollten (1, 5, 10, 15, ...))
 - Für jeden Test-Datenpunkt
 - Für alle Trainingsdaten muss Distanz berechnet werden $d(x_{\text{test}}, x_{\text{train}})$: Mit Euclidean, Manhattan, cosine, ...
 - Trainingdaten müssen in ascending Order sortiert werden
 - Die ersten k Datenpunkte werden von der Sortierung genommen
 - Von diesen Punkten werden die am häufigsten vorkommende Klasse genommen als Klassifikation.
- ⇒ Key Elemente: Die Anzahl Nachbarn k, Distanz-Metrik

Distanz-Metriken

Es gibt unterschiedliche Definitionen von Distanzen, die verwendet werden können.

Distance Metric

- Given $\mathbf{x}_1 = (x_{1,1}, x_{1,2}, \dots, x_{1,n})$ and $\mathbf{x}_2 = (x_{2,1}, x_{2,2}, \dots, x_{2,n})$
- Cosine distance, $\cos \theta = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}$
- Manhattan Distance, $d_M = \sum_{i=1}^n |x_{1,n} - x_{2,n}|$
- Euclidean Distance, $d_E = \sqrt{\sum_{i=1}^n (x_{1,n} - x_{2,n})^2}$
- Remember week 2?
- Minkowski Distance $d_{MIN} = (\sum_{i=1}^n (|x_{1,n} - x_{2,n}|^p))^{1/p}$
 - Special case p = 1, Manhattan distance
 - Special case p = 2, Euclidean distance



Das richtige k und die richtige Distanzmetrik

Um dies zu bestimmen sollte man folgende Elemente von AI verwenden (Beschrieben in vorangehenden Kapitel):

- Test-Train split
- Cross Validation
- Test Performance Überwachung (Mean accuracy, Precision, Recall)

Vor und Nachteile

Vorteile:

- Einfaches Modell
- Wenige Hyper-Parameter

Nachteile:

- K muss gut gewählt werden
- Grosse Berechnungsaufwand, wenn Sample gross ist
- Nicht effizient bei vielen Dimensionen
- Richtige Skalierung muss verfügbar sein

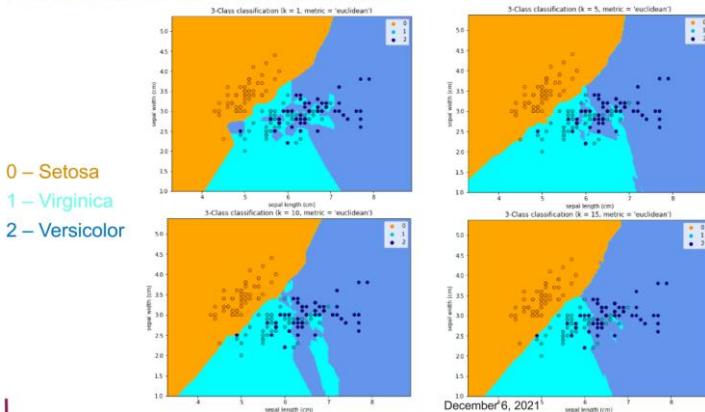
Beispiel IRIS

• Iris is a flowering plant, the researchers have measured and recorded various features of different iris flowers

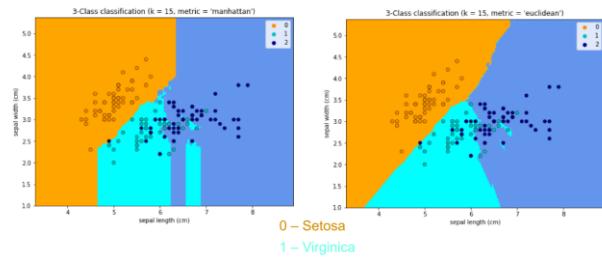
• Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type.

	sepal_length	sepal_width	petal_length	petal_width	species
1	5.8	2.7	5.1	1.9	virginica
2	7.0	3.0	5.9	2.1	virginica
3	6.3	2.9	5.6	1.8	virginica
4	6.5	3.0	5.8	2.2	virginica
5	7.6	3.0	6.6	2.1	virginica

Effect of K on classification



Effect of distance Metric



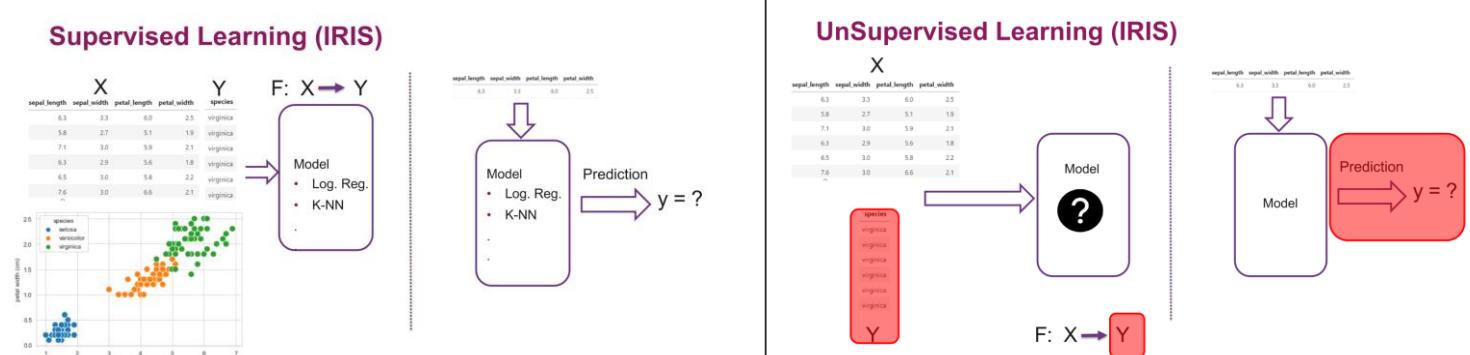
Woche 13: Clustering

Unsupervised Learning

Alles bisher war Supervised Learning. Nun ist das Thema Unsupervised Learning. Hier haben wir nur die Features (X-Werte) und uns sind die Labels (Y-Werte) unbekannt.

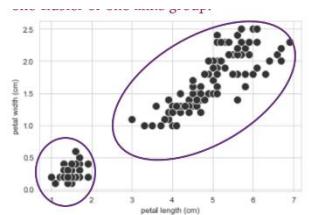
Damit können wir die Struktur der Daten lernen. Wir wollen das Pattern der Daten selbst entdecken. Die Struktur kann auch durch Noise verborgen sein. Menschen können Struktur sehr leicht erkennen und Algorithmen haben es schwerer. Wo uns aber Algorithmen unterstützen ist bei grossen Daten mit vielen Dimensionen. Wir können maximal 3 Dimensionen haben.

Unterschied Supervised – Unsupervised Beispiel IRIS



Clustering (Mit KNN Klassifizierung)

Ein Beispiel, was wir erkennen können mit unsupervised Learning sind Clusters: Datenpunkte mit Ähnlichen Werte. fallen in einer Gruppe zusammen.



Beispiele:

- Soziale Netzwerke analysieren
- Astromische Daten
- Google News kategorisierung (Gruppieren von News in Kategorien)
- Market Segmentation

Wir können KNN Klassifizierung zur Hilfe nehmen. Denn wir nehmen an, dass ähnliche Datenpunkte nahe zusammen sind. (Distanzmetriken: Eculidean, Manhatten)

Group n data points into k_c number of clusters.

Woche 13: Naive K-means

Ein Algorithmus um Cluster zu erkennen.

1. Let us assume we know the number of clusters k_c
2. Initialize the value of k cluster centres (aka, means, centroids) (C_1, C_2, \dots, C_{k_c})
3. Assignment :
 1. Find the squared Euclidean distance between the centres and all the data points.
 2. Assign each data point to the cluster of the nearest centre.
4. Update: Each cluster now potentially has a new centre (mean). Update the centre for each cluster
 1. New Centres ($(C'_1, C'_2, \dots, C'_{k_c})$) = Average of all the data points in the cluster($1, 2, \dots, k_c$)
5. If some stopping criterion met, Done
6. Else, go to Assignment step 3

Some Formal Maths

- Initialization: choose $(C_1, C_2, \dots, C_{k_c})$. (C_1, \dots are vectors in the same space as the features. They can be randomly initialized)
- Assignment Step:

$$\forall_{k=1}^{k_c} \forall_{i=1}^N d_{i,k} = (x_i - C_k)^2 \quad (\text{squared euclidean distance})$$

$$d_{i,k}^{\min} = \underset{x_i \in \text{Cluster } k}{\text{Minimum}}(d_{i,1}, d_{i,2}, \dots, d_{i,k_c})$$
- Update Step: $(\text{centre}_k = \text{mean of all data in cluster}_k)$

$$\forall_{k=1}^{k_c} C_k = \frac{1}{\text{size}(C_k)} \sum_{x_i \in \text{cluster } k} x_i$$

Stopping-Kriterium

- Wenn «Centres» nicht mehr ändern (Zeitaufwändig)
- Datenpunkte in einem Cluster ändern sich nicht mehr (Geht viel zu lange)
- Distanz vom Datenpunkt zum Zentrum \geq gewählten Threshold
- Fixe Anzahl an Iterationen erreicht
 - o Zu wenig Iterationen: Schlechte Cluster → Muss gescheid gewählt werden.

Initialisierung

Die Anfangs-Zentrumswerte sind meist zufällig. Somit ist die Performance davon abhängig. Bestimmte Anfangswerte führen auch zu schlechten Convergence-Raten oder auch schlechtes Clustering.

Wenn die Zentrumswerte nahe beieinander sind braucht es viel mehr Iterationen.

Am besten also mehrmals ausführen und wenn Cluster stabil bleibt dann haben wir gutes Cluster.

Standardisierung

Die Daten müssen normalisiert werden. Ist wichtig, dass Features mit grossen Werten nicht dominieren.

Beispiel: Noten und Berufserfahrung. Ohne normalisierung hat die Berufserfahrung ein Übergewicht und die Noten spielen keine Rolle.

Python

Sklearn kmeans

- Initialization
 - Init = Kmeans++
 - only the initialization of the centroids will change, rest everything is similar to conventional KMeans.
 - The initial chosen centroids should be far from one another.
 - Details beyond the scope
[sklearn.cluster.KMeans — scikit-learn 1.0.1 documentation](#)
- max_iter : number of iterations of (assignment and update) to perform before stopping
- n_init : Number of time the k-means algorithm will be run with different centroid seeds.
 - The final results will be the **best** output of n_init consecutive runs.
- from sklearn.preprocessing import MinMaxScaler : data scaled

Qualität des Clusters: WCSS und Silhouette

Die Nummer des Cluster ist ein Hyperparameter, der intelligent gewählt werden muss.

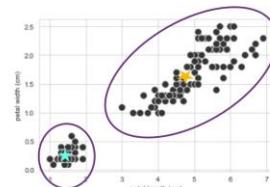
Was man bereits im voraus sagen kann. $K_c = 1$ bringt uns nicht weiter (Alle Punkte gehören zum selben Cluster). $K_c = N$ ist viel zu gross, dann haben wir minimalen Abstand zum Zentrum, aber jeder Punkt ist ein eigenes Cluster.

Bei Unsupervised Learning haben wir keine Testdaten die wir prüfen können, da wir keine Labels haben zum gegenchecken.
Somit benötigen wir andere Methoden.

WCSS

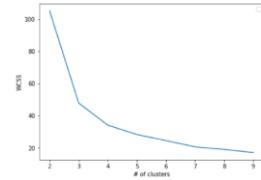
Distanz von den Punkten zum Zentrum muss minimalisiert werden.

- Goal of good clustering:
 - Minimize** $\sum_{k=1}^{K_c} \sum_{x_i \in \text{Member}(C_k)} d(C_k, x)$
 - Make clusters so that for each cluster the distance of each cluster member from its centre is minimized
 - Inertia or within-cluster sum-of-squares (WCSS)**
 - WCSS = Sum of squared distances of samples to their closest cluster centre
 - how far away the points within a cluster are
 - a small of inertia is desired
- ```
for i in range(0, N):
 kmeans = Kmeans(n_clusters = i)
 kmeans.fit(X)
 kmeans.inertia_
```



### Inertia/ WCSS applied to the Iris Data Set

- What happens to inertia as we increase the number of clusters?
- We draw a plot  $k_c$  vs inertia
- Remember  $k_c = N$ ,  $\text{inertia}$  or  $\text{WCSS} = 0$
- $k_c$  changed from 2 to 3 what is the effect on WCSS?
- $k_c$  changed from 3 to 4 what is the effect on WCSS?
- The drop in WCSS is not sharp
- Margin of returns falls
  - Smaller decrease in WCSS as the number of clusters increases

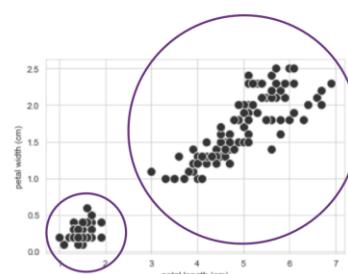


### Silhouette Score

Entfernung der Punkte von einem Cluster zum anderen Cluster. Desto näher bei 1, desto besser.

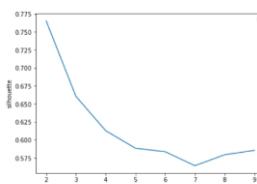
### Silhouette Score

- how far away the datapoints in one cluster are, from the datapoints in another cluster.
  - Silhouette Score of a point* =  $\frac{b-a}{\max(a,b)}$
- where
- $a$  = average intra-cluster distance i.e the average distance between each point within a cluster.
  - $b$  = average inter-cluster distance i.e. the average distance between a cluster and its nearest neighbour cluster
  - The **Silhouette score for a set of samples** the mean of the Silhouette Coefficient for each sample.



### Iris Dataset: How to choose number of clusters?

- Silhouette Score
  - Averaged over all datapoint.
  - Fewer clusters look better



### Bestimmung welche Clustergrösse anhand von IRIS

In diesem Beispiel ist die optionale Clustergrösse 2-3 Cluster. Denn ab drei Cluster gibt's beim WCSS einen Knick und es verbessert sich nicht mehr so stark. Silhouette fällt aber noch weiter.

## Woche 14: Ensemble Methods

Ziel: Mehrere Modelle zu nutzen um ein Problem zu lösen. Anstelle das perfekte Modell zu nutzen, werden mehrere (schlechtere) Modelle trainiert und die Ergebnisse aggregiert.

**Wisdom of Crowd:** Bei einer komplexen Frage mehrere zufällig ausgewählte Personen fragen und Ergebnisse aggregieren. Dies funktioniert wahrscheinlich sehr gut und ist auch einfacher als die am besten passende Person, den Experten zu finden.

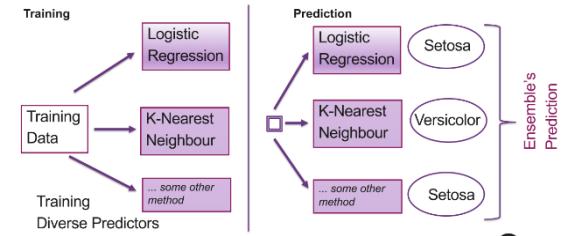
Dieses Konzept kann mit Ensemble auf Machine Learning übertragen werden. Wir erhalten je nach dem sogar ein viel besseres Resultat als mit nur einem Modell. Ensemble steht für «A group of Predictors», «Ensemble Learning», «Ensemble Method»

Ensemble Methoden werden meist gegen Ende eines Projekts benötigt, wenn man bereits einige gute Predictors hat und diese kombinieren will.

### Unterschiedliche Dimensionen

#### Unterschiedliche Algorithmen

Wir kennen beispielsweise zwei Methoden für die Klassifizierung. Diese können wir kombinieren. Z.B. KNN und Logistic Regression.



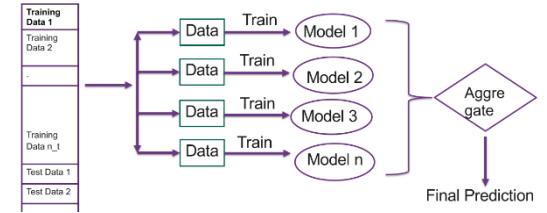
#### Unterschiedliche Hyperparameter

Wir können bei KNN verschiedene k verwenden. Oder bei Logistic Regression diverse Regularisierungsparameter.

#### Unterschiedliche Trainingsdaten

Wir können Cross Validation Nutzen und dann die Ergebnisse kombinieren.

Oder diverse Features aus Feature Engineering kombinieren. So haben wir unterschiedliche Splits.



### Voting

#### Hard

Beim harten Voting wird die Klasse genommen mit den meisten Votes. Beispiel in der Grafik oben wird Setosa genommen, da dies zwei mal vorkommt und Versicolor nur einmal.

#### Soft

Kann nur genommen werden, wenn Ergebnisse Wahrscheinlichkeiten sind, nicht Klassen. Dann kann man einen Durchschnitt aus allen Modellen nehmen.

### Auswählen der Daten

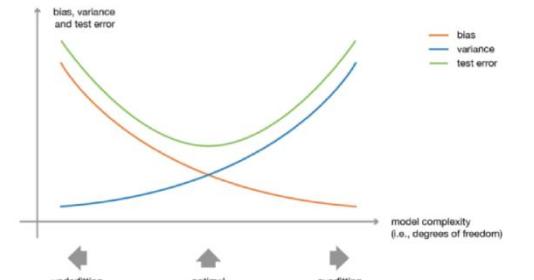
Für die verschiedenen Modelle müssen die Daten aufgesplittet werden. Da gibt es grob zwei Varianten:

**Sampling without Replacement:** Datenpunkt kann nur einmal selektiert werden.

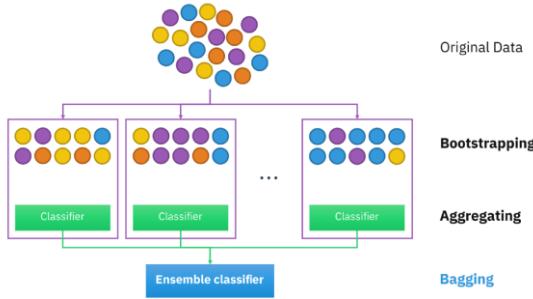
→ Pasting

**Sampling with Replacement:** Datenpunkt kann mehrmals selektiert werden.

→ Bagging (Bootstrap Aggregating)



1. the individual (simple!) models have a relatively high bias and low variance
2. Bootstrap (reuse of data) reduces variance
3. Aggregation potentially increases the expressiveness of the model and therefore reduces the bias

**Ensemble Method with Bagging (=Bootstrap + Aggregating)**

Bagging kann auch für Features gemacht werden nicht nur für die Daten. So kriegt man noch mehr unterschiedliche Modelle. (`max_features` und `bootstrap_features`).

Sampling both features and training data → Random Patches

Sampling only features → Random Subspaces

**Out of Bag (oob) Evaluation**

Es gibt Datenpunkte die nie gewählt werden wenn man die Datenpunkte zufällig wählt. Dies sind dann oob-Punkte. Diese können genutzt werden als Testdaten um das Modell zu trainieren. So braucht man nicht extra vorher die Daten aufzuteilen in Trainings- und Testdaten.

**Codebeispiele**

*Links:* Einfaches scikit-Learn Beispiel mit Voting

*Rechts:* Hier werden Modelle mit `n_jobs=-1` parallel evaluiert. Ansonsten würde es zu lange geben. Ebenso wurde nun OOB aktiviert. Auch wird Bagging genutzt

```

log_clf = LogisticRegression()
log_clf.fit(X_train, Y_train)
knn_clf = KNeighborsClassifier(n_neighbors=3, metric="Euclidean")
knn_clf.fit(X_train, Y_train)
voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('knn', knn_clf)], voting='hard')

```

```

from sklearn.ensemble import BaggingClassifier
ensemble_classifier =
 BaggingClassifier(LogisticRegression(), #uses logistic regressions
 n_estimators=3, #creates 3 weak classifiers
 max_samples=60, #The number of samples to draw from X
 bootstrap=True, #sampling with replacement
 n_jobs = -1, # -1 means using all processors
 oob_score=True) # out of bag evaluation

```

**No free lunch theorem**

Warum Ensemble statt alle Zeit investieren in die Optimierung eines einzelnen Modells? Antwort: Weil kein einzelner Algorithmus die Lösung ist für ein Problem und es keinen besten Algorithmus gibt. Wichtige Punkte die zu beachten sind:

- Kein einzelner Algorithmus kann alle Probleme besser lösen als jeder anderer Algorithmus
- Machine Learning muss richtig verstanden werden, genauso die involvierten Daten, bevor man den Algorithmus wählt
- Jedes Modell ist nur so gut, wie die Daten die wir haben und den schlüssen daraus.
- Einfachere Modelle, wie Logistic Regression, tendieren eher dazu zu underfitten und haben mehr Bias. Zu komplexe Modelle führen zu grösseren Varianzen und tendieren zum Overfitting.
- Die besten Modelle sind jeweils in der Mitte von zwei Bias-Varianzen-Extremen
- Um ein gutes Modell zu finden, muss man mehrere unterschiedliche Modelle probieren und vergleichen mittels Cross-Validation.

## Bagging

(a more complete picture)

