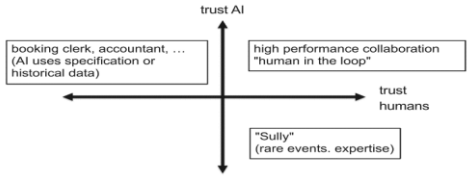
AGI (Artificial general intelligence)

«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 explicity»

Ist AI intelligent? ➔ Turing-Test

- Person darf nicht erkennen ob Maschine oder Mensch hinter Wand ist. Stellt fragen.
- AI muss sich je nach dem dumm stellen

Kann man AI vertrauen oder nicht?



Machine learning ist ein Teilgebiet. Unterteilt in:

- *Reinforced Learning*
  - Game AI, Robot Navigation, Skill
- *Unsupervised Learning*
  - Datenmenge segmentieren, clustern für Visualisierung (Politlandschaft)
- *Supervised Learning*
  - Regression mit linearer Algebra
  - Klassifizierung (Erkennung Hund/Katze)
  - Daten mit Zusatzinfos (Labels)

Dialogflow (Service von Google für Chatbot)

Intent:

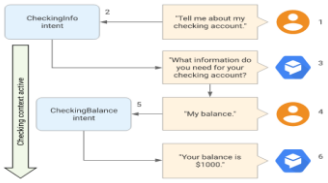
- Kategorisiert Absicht Enbenutzer
- Trainingsätze, Aktionen und Parameter
- Unterschiedliche Antworten möglich

Entities:

- Einzelne Wörter aus Sätze auslesen und Mappen
- Beispiel time und location
- Mehrere Trainingsbegriffe, sowie Synonyme möglich
- System-Entities: Vordefinierte Entities wie Datum

Dialog Control (Kontexte):

- Intent der als nächstes kommt nach Abschluss
- Übergabe von Parameter zwischen Intents



Zutaten für ein Machine-Learning-Projekt

Data

- Gegebenes Datenset inkl. Pre-Prozess-Pipeline mit
  - *Cleansing*: Aufräumen, korrigiern und sortieren
  - *Feature-Engineering*: Extrahieren von Features (z.B Katze ist ein Säugetier)
  - *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 Ausdruc: Wie gut oder schlecht ist Aufgabe gelöst anhand einer Zahl. Z.B MSE
- Falsch positiv und falsch negativ je nach dem unterschiedliche Auswirkungen
  - Gesichtserkennung CIA Zutritt oder Geschäft Vielzahler

Model

- "Ding" das aus Input ein Output generiert.
- Kann Linear oder komplex neuronal Netzwerk sein
- Typischerweise mit Frameworks (Tensorflow)
- Unterschiedliche Aufgaben, Anderes Modell

Optimization Procedure

- Modell muss mit ML optimiert werden
- Parameter Tuning
- Z.B Stochastic Gradient Descent, ADAM, …

Performance optimization

- Statt effiziente Pipelines (schwierig), Tool-Spezifische Empfehlungen und Referenz-Implementationen nutzen.

Visualization and evaluation oft the Learning Process

- Visualisierung vom Lernprozess (Optimierer) mit Tensorboard

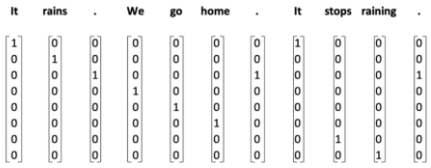
Cross-Validation & Regularization

- Modell zu trainieren die gut Generalisieren
- Flexibilität soll möglich sein.
  - z.B Hundebilderkenner erkennt neue Rassen

NLP (Natural Language Processing)

Ziel ist Verstehen was ein Mensch will (DeepL, Google)

One-Hot Representation



- Jedes Wort kriet ein Vektor mit einem Wert auf 1 gesetzt und alle anderen auf 0
- Vektordimension = Anzahl unterschiedliche Wörte
- Hat einige Nachteile
  - *High dimensional*: Vektoren schnell sehr gross
  - *Sparse*: Memory-Ineffizient, KI lernt nicht viel
  - *No generalization*: Kontext fehlt, Wörter sind unabhängig voneinander und bedeutungslos

Indexing

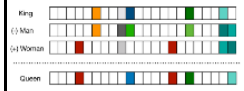
Example: "The cat sat on the mat."

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

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

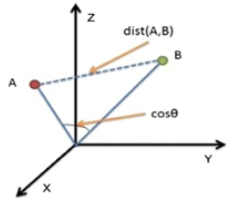
- Liste von Wörter wird erstellt, Nummer wird zugewiesen
- Sehr nahe bei One-Hot, aber besser, weshalb häufig als preprocessing genutzt.

Distributed Representation (Dense Vector)



«You shall know a word by the company it keeps»

- Wort wurde indexiert mit Algorithmus (Embedding Layer) und in ein Vektor umgewandelt.
- Mit guten Vektoren und Mathematik können Ähnlichkeiten erkannt werden oder durch Addition/Subtraktion neue Wörter gebildet werden.
- Distanz mittels Cosine Distance und Skalarprodukt



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

TI-Nspire: $\frac{\text{dotP}(a,b)}{\text{norm}(a)\cdot\text{norm}(b)}$

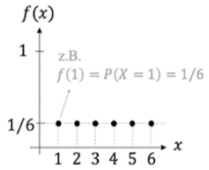
- Skalarprodukt Relationen zwischen zwei Vektoren
  - Maximal wenn in gleiche Richtung (*Normalvektor: 1*)
  - Null wenn senkrecht (orthogonal)
  - Minimal (Negativ), wenn entgegengesetzt (Normalvektor: *-1*)
- Meist wird vordefiniertes Modell verwendet oder Algorithmus wie word2vec

Probability

Zufallsvariable X ist Variable, die numerischen Wert x zuordnet aus Zufallsexperiment.

- *Disket*: X nimmt fixen Wert an aus def. Zahlenmenge (z.B {1.5, 2.98, 4, 6, 8})
- *Stetig*: X nimmt ein Wert aus einem unzählbaren Wert an (z.B alle Zahlen in einem Intervall [2,7])

Probability Mass Function (PMF)



- Gibt Warscheinlichkeit an für jeden möglichen X-Wert. (z.B Würfel werfen)
- Definiert auch Verteilung (Probability Distrubution)

Joint Probability Mass Function

- Benötigt zwei Zufallsvariablen X und Y.
  z.B zwei Würfel nacheinander werfen – 5 und 4:
  Pr(X=5, Y=4) = 1/36 oder *P(5,4) = 1/36*
- Wenn alle Variablen **unabhängig** gilt:

$$P(X,Y) = P(X) \cdot P(Y)$$
$$P(X,Y,Z) = P(X) \cdot P(Y) \cdot P(Z)$$

Correlated Random Variable (Bedingte Wahrscheinlichkeit)

- Relation zwischen Joint und Conditional

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

- Joint und marginal kann zur Berechnung nutzen:

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

- Satz von Bayes:

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

Anwendung: Naive Bayes Kategorisierung zwecks Spamfilterung

Datenvisualisierung

Visualisierung von Daten hilft um…

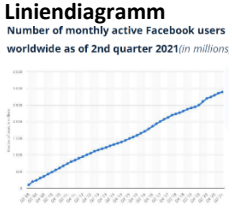
- … ein intuitives Verständnis der Daten zu erhalten
- … Trends, clusters und lokale Muster zu sehen und identifizieren
- …Entdecken von Ausreissern und ungewöhnliche Gruppen
- …unsere Hypothesen/Vermutungen/Theorien zu validieren
- … Resultate von Umfragen, etc. zu visualisieren (Insbes. bei wichtigen Daten, da viele Überfliegen)
- … Aufdecken von überraschenden Fakten

Plots benötigen

- *Label der X-Achse*: Was wird präsentiert?
- *Label der Y-Achse*: Was wird präsentiert
- *Titel*: Was zeigt der Plot?
- *Skala*: Linear oder Logarithmisch?
- *Dimension*: 2D oder 3D. Mehr geht nicht.

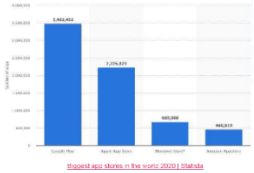
Mögliche Plots

- **Liniendiagramm**



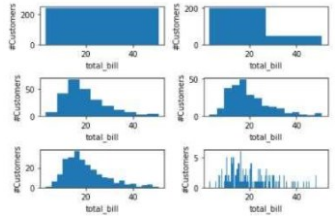
- Bivariant (kategorisch, kontinuierlich)
- Trend definiert als Muster der Veränderung
- Allgemeine Bewegung über Zeit statistisch nachweisbaren Veränderung
- Linie verbindet unterschiedliche Datenpunkte kontinuierlich
- Zeigt Änderungen relativ zu einem andern Wert
- Beispiele: Aktienpreise, Bevölkerungszahlen

- **Balkendiagramm**



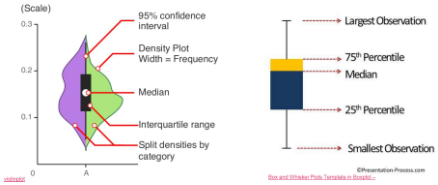
- Wird für kategorische Daten genutzt, bei der Zählung aufgrund von Kategorien erfolgt
- Beispiel: Anzahl Apps pro Kategorie

- **Histogramm**



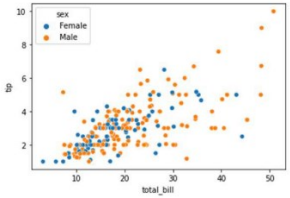
- Empirische Verteilung wird dargestellt durch:
  - Automatische Bins (Intervalle) entlang des Wertebereichs
  - Anzeige von vertikalen Balken zur Angabe der Anzahl Beobachtungen pro Bi

- Box Plot und Violin Plot

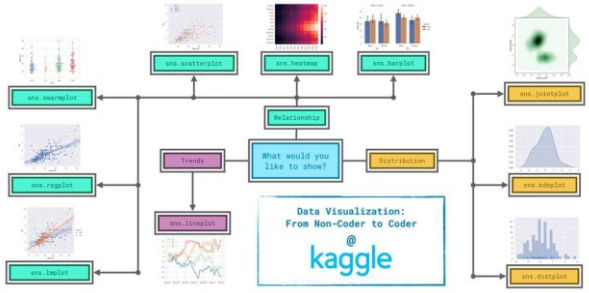


- Anzeigen von Daten durch Quartile und Median
- Zeigt Ausreisser und wo die meisten Punkte sind

- Streudiagramm (Scatter Plot)



- Beziehung zwischen zwei kontinuierlichen Variablen
- Farbe um dritte Variable ergänzen
- Grad und Korrelation kann erkannt werden



Lineare Regression

Wird von Statistik ausgeliehen und ist in ML ein Modell für Supervised Learning. Man nutzt es für:

- *Interpretation*: Verstehen ob ein bestimmter Input ein Effekt hat für den Output.
- *Prediction*: Voraussehen wann was in Zukunft passieren könnte.

Model (Mathematische Funktion, welche Daten erklärt)

$$y_i \approx f(x_i)$$
$$y_i = f(x_i) + \varepsilon_i$$

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

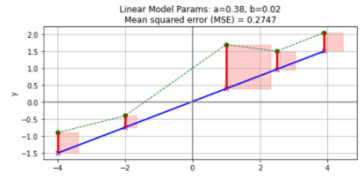
Funktion f kann beliebig kompliziert sein (Konstante, Multi-Millionen-Parameter-Netzwerk). ML muss korrektes Modell finden. Dazu wird ein y-Hut bestimmt (Schätzfunktion von y_i)

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

In Linearen Regression gibt es zwei freie Parameter:
a (*slope*) und b (*intercept*)

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

Mean Squared Error MSE (Loss)

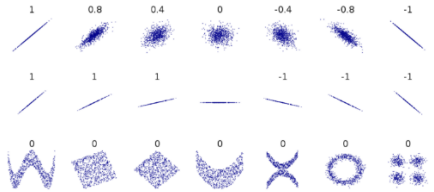


$$e_i = y_i - \hat{y}_i$$
$$E = \frac{1}{2N}\sum_{i=1}^{N} e_i^2$$
$$= \frac{1}{2N}\sum_{i=1}^{N} \left(y_i - (a \cdot x_i + b)\right)^2$$

Abweichung wird ***Residuals*** genannt (e_i)

Korrelation ist NICHT Kausalität!

Korrelation sagt aus wie zwei Variablen linear zusammenstehen
➔ Pearson Correlation Coefficient



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)**

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

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

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y},$$

where

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

In Python kann das Modell mit numpy, bzw. Panda-Array erstellt werden

```
from sklearn import linear_model

# create a LinearRegression object
lin_model = linear_model.LinearRegression()
lin_model.fit(data_X_array, data_Y)
# we can use the model to predict values.
y_hat = lin_model.predict(data_X_array)
```

Uniform Distrubution, Normal distribution

Gleichverteilung (Uniform Distrubution)

Zufallsvariable die jeden Wert eines bestimten Intervalls gleich Warscheinlich annimmt.

Normalverteilung (Gausian Distrubution)

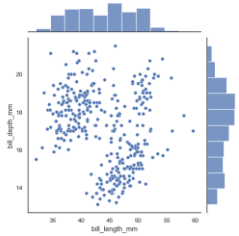
Zufallsvariable an einer Normalverteilung angelehnt.

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

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

Seaborn Jointplot

- Verbindung von zwei Variablen mit bivariaten und univariaten Graphen
- 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")
```



Optimization: Stochastic Gradient Descent (SGD)

Der Gradient des Mean Squared Errors MSE (Ableitung einmal nach Variable a und einmal nach b und dies in einen Vektor platzieren):

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

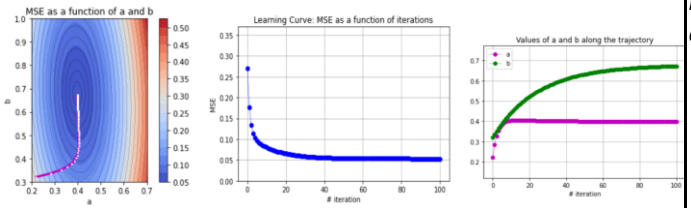
Problem damit:

- Macht zu grosse Schritte
- Geht von Tal zu einem Berg

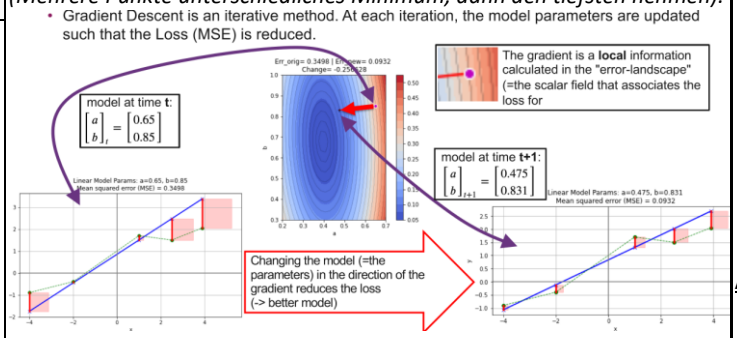
⇨ Lösung: hinzufügen von Alpha und Negierung des Wertes Gradient wird an neuem Punkt immer weiter berechnet:

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

- Dies dauert an bis bestimmte anzahl Iterationen erreicht ist. Ergibt Learing-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).



Stochastic Gradient Descent (SGD)

- Ziel: nur eine kleine Menge and Datenpunkte nimmt um die Fehler und Gradient zu berechnen.
- Man selektiert zufällig paar Datenpunkte aus Dataset (J = Teildatenset):

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

$$\frac{\partial E}{\partial a} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (a \cdot x_i + b))(-x_i) \approx \frac{1}{n}\sum_{j \in J_n}\Big(y_j - \big(a \cdot x_j + b\big)\Big)(-x_j)$$

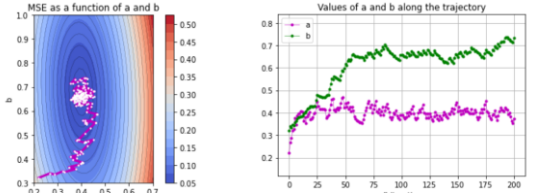
$$\frac{\partial E}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (a \cdot x_i + b))(-1) \approx \frac{1}{N}\sum_{j \in J_n}\Big(y_j - \big(a \cdot x_j + b\big)\Big)(-1)$$

- Update-Regel wird entsprechend angepasst:

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

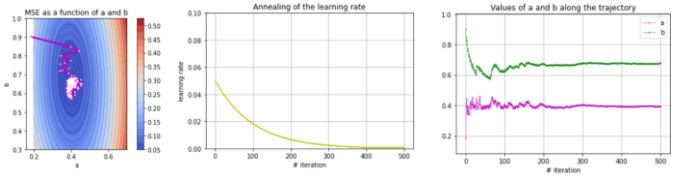
- Batch möglichst klein (32 oder 64 werden häufig verwendet), wenn es geht ist Batchgrösse 1 am optimalsten.

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.



Stochastic Gradient Descent with annealed learning rate

- Löst Problem von der grossen Streuung am ende
- Alle paar Iterationen wird Alpha verkleinert.
- Führt zu einer Abflachung, dennoch zu Beginn grosse Schritte



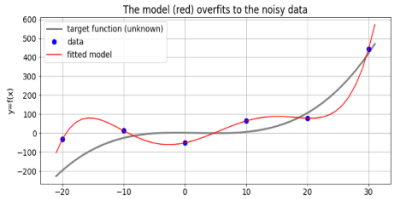
Allgemeine Anmerkungen

- Loss-Function muss ableitbar sein.
- SGD kann nur mit einem Datenset arbeiten, was sehr ineffizient ist und selten benutzt wird
  - Batch oder Minibatch-Gradient ist also besser
- SGD muss der 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.

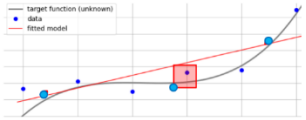
Generalisation & Regularisierung

Overfitting



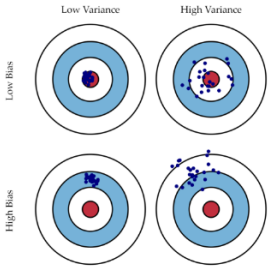
- Alle Punkte genau getroffen
- MSE ist 0 , Loss wurde zu stark minimalisiert
  ➔ **In-Sample-Error** (Training Error)
- Testdaten (Neue, ungesehene Daten) sind dann weit weg
  ➔ **Out Of Sample Error** (Generalization Error)
  - Kann nicht berechnet werden, sondern nur geschätzt
  - Damit wir Schätzen können ist Splitting Notwendig von Daten (80/20 ist ein guter Split). Test-Daten dürfen NICHT für Training, sondern nur fürs Testen verwendet werden und Fehlerschätzung
- Ziel wäre In Sample Error grösser, Generalization Error tiefer

Underfitting



- Zu einfaches Modell
- Viel zu hoher In Sample sowie Gerneralization Error

Bias und Varianz (Mathematische analyse Der Fehler

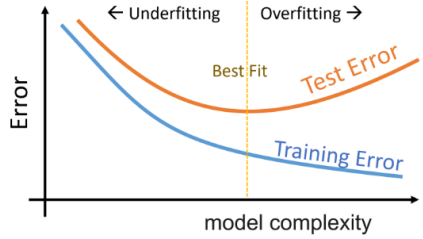


- High Bias, Low Variance
  - Modell zu einfach und limitiert
    ➔ Underfitting
  - Z.B Lineares Modell obwohl es passendere gibt
- Low Bias, High Variance
  - Modell (zu) komplex.
  - Z.B: 6 Punkte und Polynom-Grad 5 wird genutzt.
  - Modell passt genau zu Testdaten ➔ Overfitting
  - Varianz ist hoch, was zu Generalisierungsfehler führt

Regularisierung

- Ziel: möglichst eine tiefe Varianz.
- Bias ist eher sekundär, sollte aber auch möglichst tief sein.
- Man braucht aber die optimale Balance.

- Optimaler Punkt wo Test-Error tief ist bevor er wieder steigt



- Man kann dies durch hinzufügen von Constrains erreichen
  - Penalty-Term (Cost-Function).
  - Optimizer optimiert die Daten (MSE minimalisieren) sowie auch den Constraint
- Zwei benötigte Zutaten:
  - Weg zum Modell-Komplexität zu messen

L1 Norm (Lasso)

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

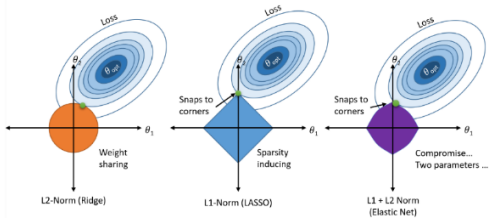
L2 Norm (Ridge)

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

  - Weg zum Model-Komplexität zu kontrollieren
    - Penalty-Term wird zu Loss hinzugefügt
    - Komplexere Modelle haben höhere Penalty
    - Constrain wird zum Optimierungsprozess hinzugefügt

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

- «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

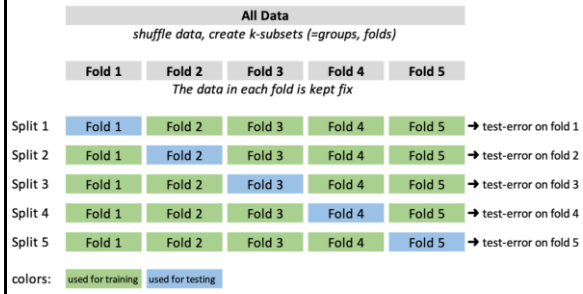
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
model = tf.keras.models.Sequential()

# 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]
)
```

Cross-Validation

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

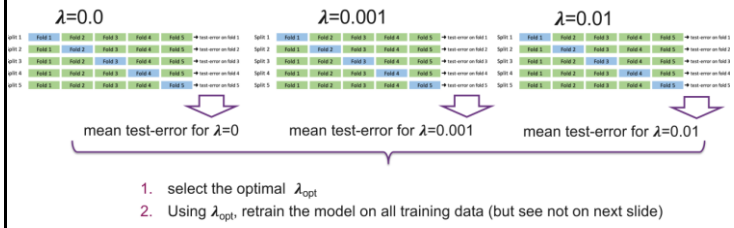
k-Fold-Crossvalidation:



- Daten werden zuerst geshuffelt.
- Anschliessend aufgeteilt in k-Fold
  (Üblich: 5-Fold, 10-Fold. Gibt auch N-Fold mit 1 Datenpunkt pro Fold)
- Jeder Trainingsdurchgang anderer Fold als Test-Datensatz
-

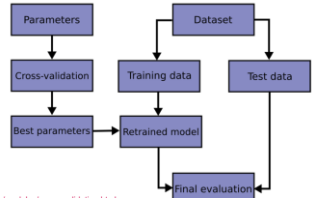
Hinweis: Daten in einem Fold ändern sich während den einzelnen Splits nicht. Diese bleiben über die ganzen Trainings gleich. Weiter muss die Preprocessing-Pipeline während den einzelnen Splits laufen und nicht davor, da sonst Ergebnisse verzerrt werden.

- Ziel 1: Generalisierungsfehler Schätzen
  - Durchschnitt aus einzelnen Testfehler nehmen als Berechnung des Mean und die Variance. Ist genauer als mit einzelnen Tests.
- Ziel 2: Auswahl von Hyper-Parameter
  - Varieren mit den Hyper-Parameter (Regularisierungslamda, etc.) um so optionalen Parameter herausfinden



Scikit-learn Cross-Validation

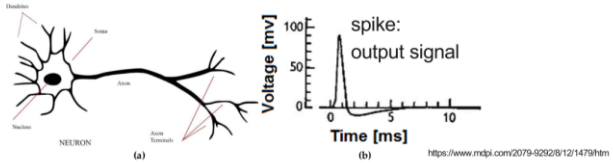
- Einzelne Daten werden vorher ganz rausgenommen als Testdaten. Dann wird mit anderen Cross-Validation gemacht. Ist empfohlen.



Artificial Neural Networks

AI ist inspiriert von unserem Gehirn und den Neuronen.

Biological Neurons integrate and communicate information

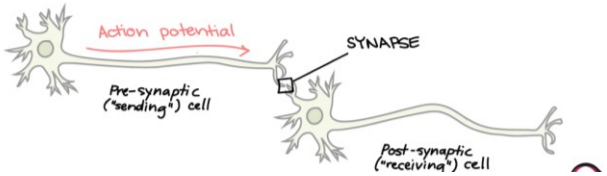


https://www.mdpi.com/2079-9292/8/12/1479/htm

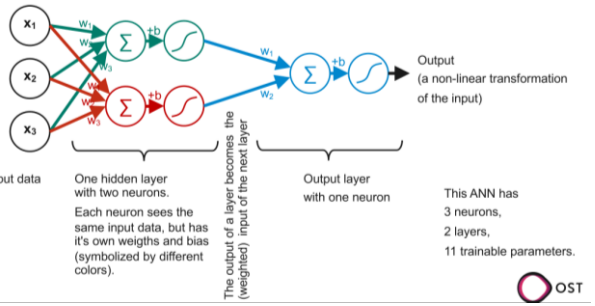
- *Dendrites*: Collect (chemical) **input** from (many) other neurons.
- *Cell Body (Soma)*: If the collective input signal reaches a (voltage) threshold, an **output** signal, called a "spike" is generated.
- *Axon*: the electrical pulse (spike) travels along the axon to other neurons.

Synapses: Where neurons connect (and learning happens)

- The human brain has ~ 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).



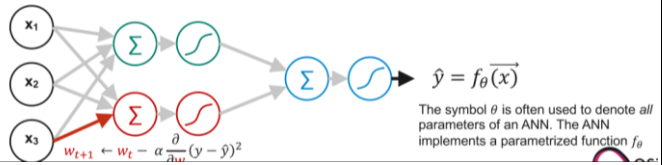
Artificial Neuron (Vereinfachung des Biologischen)



Bei zahlreiche Hidden Layer, wird von 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)



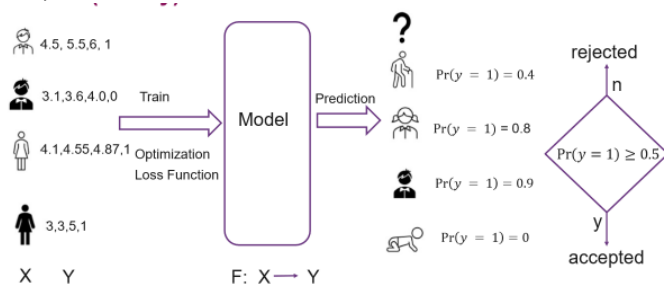
ANN's from a computer science perspective

- 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

Mit Logistic regression können Daten Binär klassifiziert werden (JA-Nein Fragen). Z.B Wird es hageln aufgrund von Satellitenbilder

Beispiel Binärer Klassifikation:



Notation

x_{n_1} = *years of Work experience of applicant n*
x_{n_2} = *BS grade of applicant n*
x_{n_3} = *Sec.School grade of applicant n*

$\mathbf{x}_n = (x_{n_1}, x_{n_2}, x_{n_3})$

$X = \{(x_{1_1}, x_{1_2}, x_{1_3}), (x_{2_1}, x_{2_2}, x_{2_3}), \ldots, (x_{n_1}, x_{n_2}, x_{n_3})\}$

$y_n = \begin{cases} 1 & \text{application n accepted} \\ 0 & \text{application n NOT acccpted} \end{cases}$

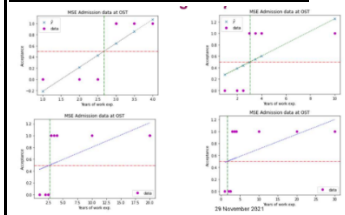
$Y = \{y_1, y_2, \ldots, y_n\}$

What is the probability that applicant #44 is accepted?
Given the notation defined above, we can formulate this question as follows:

$\Pr(y_{44} = 1 \mid (x_{44,1}, x_{44,2}, x_{44,3})) = \Pr(y_{44} = 1 \mid \mathbf{x}_{44}) = \Pr(y = 1 \mid \mathbf{x})$

Note that as soon as the context is set, notation is often minimized

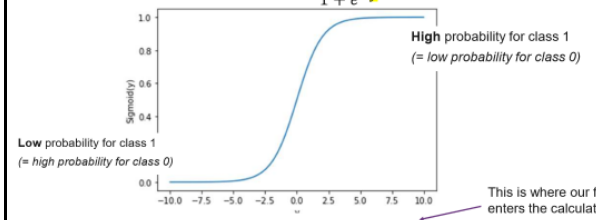
Warum nicht Lineares Modell?



- Problem ist, dass die Kurve ganz anders aussieht, sobald ein weiterer Datensatz dazu kommt und Tresholding hier falsche Ergebnisse liefern kann.
- Die Kostenfunktion (Loss) MSE funktioniert gar nicht
- Wir sind interessiert in eine Warscheinlichkeits-Output. ➔ Sigmoid

Sigmoid Funktion

$$sigmoid(y) = \frac{1}{1+e^{-y}}$$



$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4 + \cdots$

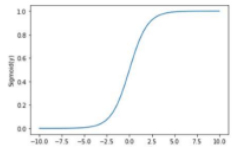
This is where our features (**data x**) enters the calculation.
The **w**'s are unknown! That's what needs to be **learned**.

Probabilities

- We can write the estimated probability

$$Pr(y = 1|x; W) = \Pr(y = 1 \mid x) = p(x) = \frac{1}{1+e^{-(\mathbf{W}^T\mathbf{x})}}$$

- For a prediction, we can write
  - Predicted probability $p(x) = {}^{1}/_{1+e^{-(w^T x)}}$
  - Note that p is a real number between 0 and 1
- **We have the model, how to find W?**



$sigmoid(0) = \frac{1}{1+e^{-0}} = 0.5$

$sigmoid(\infty) = \frac{1}{1+e^{-\infty}} = 1$

$sigmoid(-\infty) = \frac{1}{1+e^{\infty}} = 0$

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

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

Maximum Likelihood (Loss)

- Die Lösung um das W zu finden. (negativer log-loss)
- Gradient descent nutzbar, 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)!

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

- For each of the training data, we want to **maximize the likelihood** of correct prediction

$$Maximize\ cost_2(W) = \sum_{i:y_i=1}^{N} \log(p(x_i)) + \sum_{i:y_i=0}^{N} \log(1 - p(x_i))$$

- 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 $y = 0$

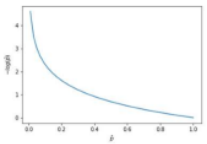
$$Minimize\ cost_3(W) = -\left(\sum_{i:y_i=1}^{N} \log(p(x_i)) + \sum_{i:y_i=0}^{N} \log(1 - p(x_i))\right)$$

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

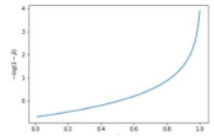
$$Minimize\ cost(W) = \frac{-1}{N}\sum_{i=1}^{N} (y_i * \log(p_i)) + (1 - y_i) * \log(1 - p_i))$$

Taking a closer look at the cost

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



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



Classifer Evaluation (Ist das Modell gut genug?)

Confusion Matrix

- **False Positive**: Berechnet: 1 Wahr: 0 (Fälschlicherweise richtig)
- **False Negative**: Berechnet: 0, Wahr: 1. (Fälschlicherweise falsch)
- **True Negative**: Berechnet: 0, Wahr: 0 (Korrekt)
- **True Positive**: Berechnet 1, Wahr 1 (Korrekt)

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

- Mean Accuracy reicht nicht um bestimmen ob Modell gut ist. Z.B: Dataset wo 90% Nein und 10% Ja sind. Bei einem Modell das immer Nein sagt ist die Accurracy bei 90%. Dank Precision und Recall welche 0 sind, wissen wir, dass Modell schlecht ist.

Precision vs Recall

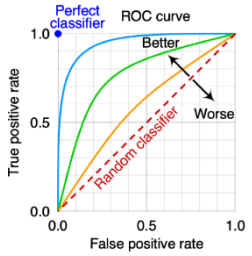
Es kommt drauf an worauf man den Fokus legt je nach Applikation

- 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?
  $Precision = t_p / (t_p + f_p)$
- **Sensitivity, Recall, True Positive Rate(TPR)**:
  How often the prediction is "1", when it's actually "1"
  $Recall = t_p / (t_p + f_n)$

Threshold (ab wann True/False)

Es gibt keine allgemeingültige Lösung mittels Mathematik. Es kommt Applikation drauf an. Ein Weg: Threshold bestimmen mit ROC-Kurve (Area under the Curve)



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

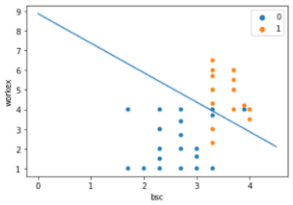
- (Anderer Plot mit Precision und Recall könnte ein Schnittpunkt ergeben, was sicher ein sehr guter Punkt ist)

K-Nearest-Neighbours KKN

Weitere Methode für Klassenbestimmung, bei mehr als zwei Klassen. (z.B Wetter, SARS-COV Varianten)

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



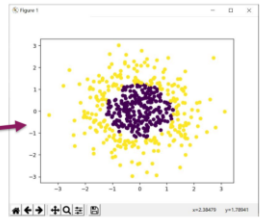
not perfect, but the decision boundary separates the data quite well

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



- Logistic Regression kann hier funktionieren, ist aber sehr umständlich. Mann muss viel trainieren (Rekursiv), was langsam ist.
- Methode ohne Training mit Predictions und einfacher Support mehrerer Klasse gesucht, welches auch bei nicht-Linearer Modelle funktioniert ➔ Diese Methode ist KKN

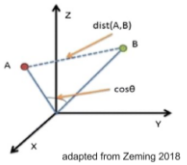
Funktionsweise von KKN

1. Training und Testdaten laden
2. Value von k bestimmen (Nummer von Nachbaren, welche berücksichtigt werden sollten (1, 5, 10, 15, …))
3. Für jeden Test-Datenpunkt x_{test}
   a. Für alle Trainingsdaten muss Distanz berechnet werden $d(x_{test}, x_{train})$: Euclidean, Manhatten, cosine, ...
   b. Trainingdaten müssen in ascending Order sortiert werden
   c. Die ersten k Datenpunkte werden genommen
   d. Von diesen Punkte werden die am häufigsten vorkommende Klasse genommen als Klassifikation.

⇨ Key Elemente: Die Anzahl Nachbarn k, Distanz-Metrik

Distance Metric

- Given $\mathbf{x}_1 = (x_{1,1}, x_{1,2}, \ldots, x_{1,n})$ and $\mathbf{x}_2 = (x_{2,1}, x_{2,2}, \ldots, x_{2,n})$
- Cosine distance, $cost\ \theta = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{||\mathbf{x}_1|| \ ||\mathbf{x}_2||}$
- Manhattan Distance , $d_M = \sum_{i=1}^{n} |x_{1,n} - x_{2,n}|$
- Euclidean Distance, $d_E = \sqrt{\sum_{i=1n}(x_{1,n} - x_{2,n})^2}$
- 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



adapted from Zeming 2018

Das richtige k und die richtige Distanzmetrik

Dazu folgende Elemente von AI nutzen für die Bestimmung:

- Test-Train split
- Cross Validation
- Test Performance überwachung (Mean accurarcy, Precision, Recall)

Vor und Nachteile von KKN

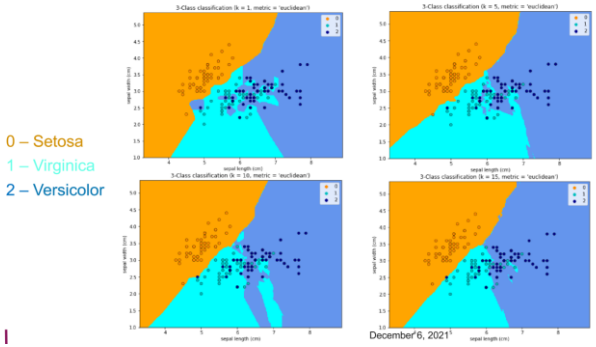
Vorteile:	Nachteile:
- Einfaches Modell - Wenige Hyper-Parameter	- K muss gut gewählt werden - Grosse Berechnungsaufwand, wenn Sample gross ist - Nicht effizient bei vielen Dimensionen - Richtige Skalierung muss verfügbar sein

Beispiel IRIS

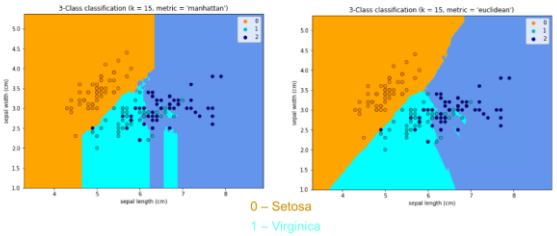
- Iris is a flowering plant, the researchers have measured and recorded various features of different iris flowers
- Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type.

sepal_length	sepal_width	petal_length	petal_width	species
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
6.5	3.0	5.8	2.2	virginica
7.6	3.0	6.6	2.1	virginica

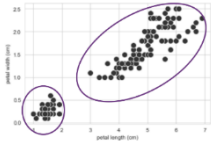
Effect of K on classification



Effect of distance Metric



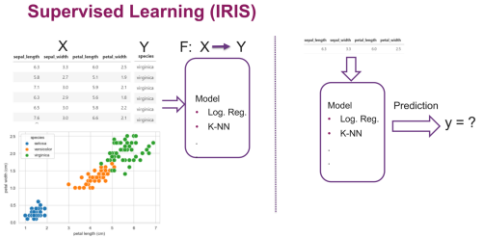
Clustering

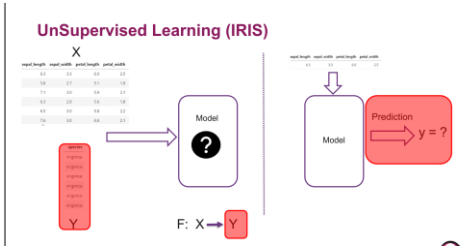


Unsupervised Learning

- Es sind nur Features (X-Werte) bekannt
- Die Labels (Y-Werte) sind unbekannt.
- Damit können wir die Struktur der Daten lernen
- Wir wollen das Pattern der Daten selbst entdecken
- Struktur kann auch durch Noise Verborgen sein
- Menschen können Struktur viel leichter erkennen als Algorithmen
- Algorithmen unterstützten bei vielen Daten mit vielen Dimensionen
- Mensch kann mit maximal 3 Dimensionen arbeiten

Unterschied Supervised – Unsupervised Beispiel IRIS





Clustering (Mit KNN Klassifizierung)

- Datenpunkte mit Ähnlichen Werte. fallen in einer Gruppe zusammen
- Z.B: Soziale Netzwerke, Astromische Daten, Google News Kategorien
- Dazu hilft KNN Klassifierung
  - Annahme: ähnliche Datenpunkte nahe zusammen. (Distanzmetriken: Eculidean, Manhatten)

Group n data points into k_c number of clusters.

Naive K-means (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) $(\mathbf{C_1}, \mathbf{C_2}, \ldots . \mathbf{C}_{k_c})$
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 $((\mathbf{C'_1}, \mathbf{C'_2}, \ldots . \mathbf{C'}_{k_c})$ = Average of all the data points in the cluster$(1,2,..,k_c)$
5. If some stopping criterion met, Done
6. Else, go to Assignment step 3

Some Formal Maths

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

$$\forall_{k=1}^{k_c} \forall_{i=1}^{N} \; d_{i,k} = (\mathbf{x}_i - \mathbf{C}_k)^2 \qquad \textit{(squared euclidean distance)}$$

$$d_{i,k}^{min} = \underset{x_i \in \textit{Cluster } k}{\textit{Minimum}}\left(d_{i,1}, d_{i,2}, \ldots, d_{i,k_c}\right)$$

- Update Step: **(centre$_k$ = mean of all data in cluster$_k$)**

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

Stopping-Kriterium

- Wenn «Centres» nicht mehr ändern (Zeitaufwändig)
- Datenpunkte in Cluster ändern sich nicht mehr (Geht viel zu lange)
- Distanz vom Datenpunkt zum Zentrum >= gewählten Threshold
- Fixe Anzahl an Iterationen erreicht
  - Zu wenig: Schlechte Cluster ➔ Muss gescheit gewählt werden.

Intialisierung

- 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 mehrmals ausführen und wenn Cluster stabil bleibt dann haben wir gutes Cluster.

Standardisierung

- Die Daten müssen normalisiert werden.
- Fehlt dies, dominieren Features mit grossen Werten
- Beispiel: Noten und Berufserfahrung. Ohne normalisierung hat die Berufserfahrung ein Übergewicht und die Noten spielen keine Rolle.

Python

- 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 Shilhouette

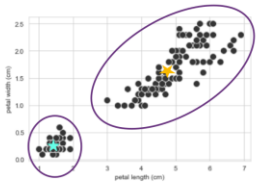
- Anzahl Cluster ist ein Hyperparameter (Muss intelligent gewählt sein)
  - K_c = 1 bringt uns nicht weiter (Alle Punkte gehören ein Cluster).
  - K_c = N ist viel zu gross (Minimal, aber jeder Punkt eigenes Cluster)
- Unsupervised Learning kennt keine Testdaten (Andere Methode!)
- WCSS (Distanz von den Punkten zum Zentrum minimieren)

- Goal of good clustering:

$$\textbf{Minimize } \sum_{k=1}^{k_c} \sum_{x_i \in Member(C_k)} d(C_k, \mathbf{x})$$

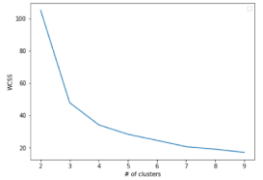
  - Make clusters so that for each cluster the distance of each cluster member from its centre is minimized
- **Inertia or within-cluster sum-of-squares (WCSS)**
  - WCSS = Sum of squared distances of samples to their closest cluster centre
  - how far away the points within a cluster are
  - a small of 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 $k_c = N, inertia\ or\ WCSS = 0$
- $k_c changed\ from\ 2\ to\ 3$ 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 von Punkt zu anderen Cluster)

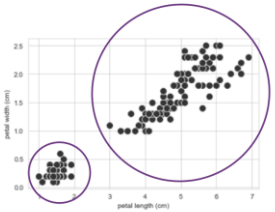
Silhouette Score

- 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

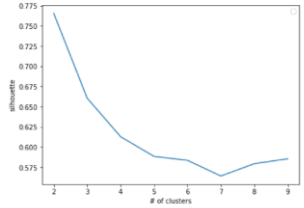
- a= average intra-cluster distance i.e the average distance between each point within a cluster.
- b= average inter-cluster distance i.e the average distance between a cluster and its nearest neighbour cluster

- 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



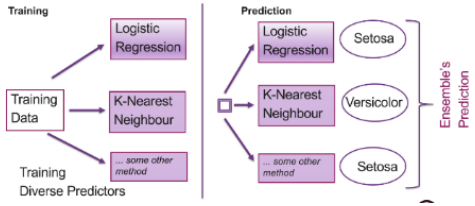
- Beispiel IRIS: 2-3 = Perfekte Clustergrösse. (Knick bei WCSS ab da so nicht mehr grosse Verbesserung, dafür Shilhouette immer schlechter.

Ensemble Methods

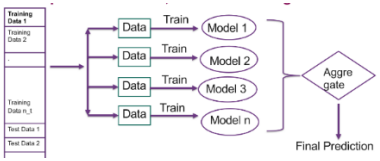
- Ziel: Mehrere Modelle zu nutzen um ein Problem zu lösen. Anstelle das perfekte Modell zu nutzen, werden mehrere (schlechtere) Modelle trainiert und die Ergebnisse aggregiert. Analog **Wisdom of Crowd**: Bei einer komplexen Frage mehrere zufällig ausgewählte Personen fragen und Ergebnisse aggregieren. Dies funktioniert wahrscheinlich sehr gut und ist auch einfacher als die am besten passende Person, den Experten zu finden.
- Ensemble steht für «A group of Predictors», «Ensemble Learning», «Ensemble Method»
- Meist gegen Ende des Projektes (mit guten Predictors)

Unterschiedliche Dimensionen

- Unterschiedliche Algorithmen



  - Z.B Kombination KNN und Logistic Regression.
- Unterschiedliche Hyperparameter
  - Z.B Bei KNN verschiedene k verwenden. Oder bei Logistic Regression diverse Regularisierungsparameter.
- Unterschiedliche Trainingsdaten

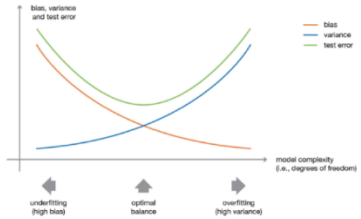


  - Cross Validation Ergebnisse kombinieren
  - Features aus Feature Engineering kombinieren. (Splitting)

Voting

- *Hard*: Klasse mit den meisten Votes (Setosa im Bild oben)
- *Soft:* Falls Ergebnisse Warscheinlcihkeiten: Durschnitt nehmen

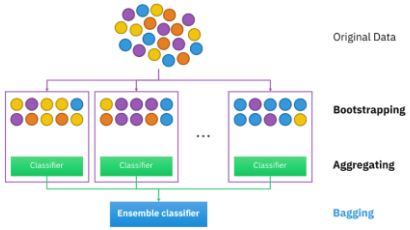
Auswählen der Daten



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 recudes the bias

- Splitting:
  - **Sampling without Replacement**: Datenpunkt kann nur einmal selektiert werden.
    ➔ Pasting
  - **Sampling with Replacement**: Datenpunkt kann mehrmals selektiert werden.
    ➔ Bagging (Bootstrap Aggregating)

Ensemble Method with Bagging (=Bootstrap + Aggregating)



- Bagging auch für Features möglich Daten. (max_features und bootstrap_features).
  - Sampling both features and training data ➔ Random Patches
  - Sampling only features ➔ Random Subspaces

Out of Bag (oob) Evaluation

- Wenn Datenpunkte zufällig, gibt es welche die nie gewählt werden. Dies sind dann oob-Punkte.
- Können als Testdaten genutzt werden (Spart splitting vorher)

Codebeispiele

- Einfaches skicit-Learn beispiel mit Voting

```
log_clf = 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')
```

- Hier werden Modelle mit n_jobs=-1 parallel evaluiert. Ansonsten würde es zu lange geben. Ebenso OOB/Bagging aktiviert

```
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 (Wozu Ensemble statt ein perfekt optimiertes Modell?)

- 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.