AI Foundation

Grundbegriffe

Artificial Intelligence (AI)

Ein grösseres Konzept, welches sich damit befasst, menschliches Denken und Verhalten nachzubilden.

⇨ z.B. Bilderkennung, Pflegeroboter, Voice-To-Text, etc.

Machine Learning (ML)

Ein Teilgebiet von AI, welches es Maschinen erlaubt, aus Daten zu lernen, ohne explizit programmiert zu werden.

- Supervised: Verwendung von Daten mit Labels, um z.B. lineare Funktionen zu bilden.
- Unsupervised: Verwendung von Daten ohne Labels, um z.B. Clusters zu erkennen.
- Reinforcement: Verwendung von Algorithmen, um eine «Belohnung» zu maximieren.

Artificial General Intelligence (AGI)

Eine «allgemeine» AI, welche jede mögliche Aufgabe lösen kann, die auch ein Mensch lösen könnte.

⇨ Traum von AI. Wir sind noch weit davon entfernt.

Übersicht

Wann ist eine AI intelligent?

Turing-Test

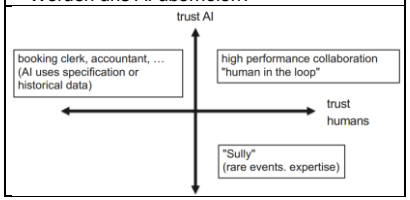
Eine Person stellt eine oder mehrere beliebige Fragen an einen Menschen und eine AI, ohne zu wissen, wer davon antwortet. Die AI gilt als intelligent, wenn die Person Mensch und AI nicht anhand der Antworten unterscheiden kann.

⇨ Sehr «informell» und mit einigen Tücken (z.B. fällt es auch auf, wenn die AI zu «intelligent» ist.)

Was sind die Herausforderungen?

Neben technischen Herausforderungen gibt es bei AI auch ethnische Fragen:

- Wo und wann werden AI akzeptiert?
- Können wir einer AI vertrauen?
- Werden uns AI überholen?



Was braucht es für Machine Learning?

ML besteht aus **4** zentralen und zahlreichen erweiterten Komponenten.

1. Data & Preprocessing

Ein Datenset, welches mit Preprocessing für die Aufgabe vorbereitet wurde. Dazu gibt es u.a. diese Schritte:

- Cleansing: Aufräumen, Korrektur und in manchen Fällen auch Sortierung der Daten
- Feature Engineering: Bestimmung der Features (z.B. Tierart und Gattung)
- Data Augmentation: Vervielfältigung der Daten durch Manipulation (z.B. Spiegeln)

2. Modell

Ein «Ding», welches aus einem Input einen Output generiert. Das Modell muss immer individuell bestimmt werden.

⇨ Kann einfach (z.B. linear) oder komplex (z.B. neuronal) sein.

3. Cost-Function (Loss)

Eine mathematische Funktion, welche Auskunft darüber gibt, ob eine Aufgabe «gut» oder «schlecht» gelöst wurde.

⇨ z.B. «Mean Squared Error» bei linearen Funktionen.

4. Optimization Procedure

Ein Algorithmus, welcher die Parameter des Modells so lange verändert, bis die Cost-Function minimal wird.

⇨ z.B. «Stochastic Gradient Descent»

Weitere Komponenten

Neben den obigen Punkten wurden 3 erweiterte Komponenten erwähnt:

- Performance Optimization
- Visualization of the learning Process
- Cross-Validation & Regularization

Dialogflow

Kontext & Verwendung

Dialogflow ist ein Service von Google für das Erstellen von AI-basierten Chatbots.

⇨ Einfaches User-Interface mit einem riesigen Backend.

Grundbegriffe

Intents

Kategorisieren die «Absichten» eines Benutzers. Bestehen u.a. aus:

- Trainingsätze (Was will der Benutzer?)
- Parameter (Welche Infos kann ich extrahieren?)
- Antworten (Was solch ich darauf antworten?)

Entities

Definieren die «Informationen» einer Nachricht. Beziehen sich im Normalfall auf ein Wort mit evtl. Synonymen.

⇨ z.B. Ort, Bestellung, Produkt, Datum
⇨ Es gibt auch einige vordefinierte Entities (z.B. Datum/Zeit)

Kontexte

Definieren, welche Intents als nächstes aufgerufen werden sollen. Gruppieren diese also in «logische» Einheiten.

⇨ So stehen z.B. Parameter im ganzen Kontext zu Verfügung.

Wahrscheinlichkeiten

Grundbegriffe

Zufallsvariable

Die Zufallsvariable X ist eine beliebige Funktion, welche jedem Zufallsereignis eine reelle Zahl x zuordnet.

⇨ z.B. «Die Zufallsvariable X sei die Augenzahl beim Würfeln.»

Wir unterscheiden zwei Arten:

- Diskret: Die x stammen aus einem klar definierten Zahlenbereich. z.B.: $x \in \{1, 2, 3.5, 10\}$
- Stetig: Die x stammen aus einen unzählbaren Zahlenbereich. z.B.: $x \in (2, 7)$

Wahrscheinlichkeitsfunktion Diskret

Eine beliebige Funktion, die jedem Wert x eine Wahrscheinlichkeit P zuordnet.

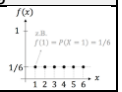
$$P(X = x) = f(x)$$

⇨ z.B. «Die Wahrscheinlichkeit eine 6 zu Würfeln ist 1 zu 6.»
⇨ Auch $P(x)$, $p(x)$ oder $P_x(x)$. Das kann manchmal verwirren.

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

Wahrscheinlichkeitsverteilung

Wird durch die Wahrscheinlichkeitsfunktion definiert und dient oft als Synonym dazu.



Zusammengesetzte Wahrscheinlichkeit

Mehrere Zufallsvariablen wie z.B. X und Y können gleichzeitig betrachtet werden.

⇨ z.B. «X sei der erste Würfel und Y der zweite Würfel»

	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

$P(X = 5, Y = 4) = \frac{1}{36}$
$P(X = 1) = \frac{6}{36}$
$P(X \vee Y = 6) = \frac{11}{36}$

⇨ $P(X, Y)$ Joint Prob.
⇨ $P(X)$ Marginal Prob.

Abhängigkeit

Zufallsvariablen wie z.B. X und Y sind voneinander abhängig, wenn das eine Ergebnis das andere beeinflusst.

⇨ z.B. 2 Würfelergebnisse sind «nicht» voneinander abhängig.

Bedingte Wahrscheinlichkeit

Beschreibt die Wahrscheinlichkeit für ein Ergebnis X, unter der Bedingung, dass Y eintritt, also $P(X|Y)$.

Rechenregeln (X und Y)

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

⇨ Falls X und Y unabhängig

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

⇨ Falls X und Y abhängig

Rechenbeispiel

	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

$P(X = 0, Y = 1) = 0.10$
$P(X = 2) = 0.20$
$P(Y = 2 \mid X = 1) = \frac{P(X=1, Y=2)}{P(X=1)} = 0.143$

⇨ $P(X|Y)$ Conditional Prob.

Satz von Bayes

Für abhängige Ereignisse gilt ebenso:

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

Verteilungen

Gleichverteilung (Uniform Distribution)

Die Verteilung einer Zufallsvariable, bei der jeder Wert gleichwahrscheinlich ist.

⇨ z.B. bei einem Würfel

```
rng = default_rng()
samles = rng.uniform(low=0, high=10, size=100)
```

Normalverteilung (Gausian Distribution)

Die Verteilung einer Zufallsvariable, bei dem die Werte um einen bestimmten Erwartungswert streuen.

⇨ Standardverteilung in der «Natur», z.B. bei Messungen

```
rng = default_rng()
samles = rng.normal(loc=0, scale=1, size=100)
```

⇨ `loc` = Mittelwert, `scale` = Standardabweichung

Natural Language Processing (NLP)

Kontext & Verwendung

Bei NLP wird versucht, die menschliche Sprache (als Text oder Audio) zu erfassen und einzuordnen.

⇨ z.B. Suchmaschinen, Sprachassistenten (Alexa), DeepL
⇨ Allgemein ein sehr wichtiger Teilbereich in der AI

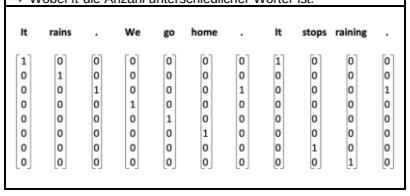
Wie werden Wörter in ML repräsentiert?

Wörter werden im Bereich von Machine Learning als Vektoren repräsentiert.

Variante 1: One-Hot Vektoren

Bei One-Hot besteht jedes Wort aus einem Vektor mit einer 1 und $n - 1$ Nullen.

⇨ Wobei n die Anzahl unterschiedlicher Wörter ist.



One-Hot kommt mit einigen Nachteilen:

- High Dimensional: Vektoren werden schnell sehr gross (z.B. Artikel mit 100'000 Wörter)
- Sparse: Jeder Vektor hat nur eine einzige Information (die 1). Dies ist sehr ineffizent.
- No Generalization: Es fehlt Kontext und Bedeutung. Wir können von einem Wort nicht auf ein anderes schliessen.

Variante 2: Indexing Vektoren

Beim Indexing wird jedem Wort ein Index zugeordnet. Danach werden diese Indexe in einen Vektor geschrieben.

⇨ Oft werden die Wörter zuerst alphabetisch sortiert. (optional)

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

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

Indexing ist nur etwas besser als One-Hot, da der Kontext immer noch fehlt.

⇨ Wird aber meistens als Preprocessing-Schritt benutzt.

Variante 3: Distributed Representation

Diese Variante macht sich zu Nutze, dass sich Wörter meistens durch ihren Kontext definiert lassen.

⇨ «You shall know a word by the company it keep»

Konzept & Aufbau

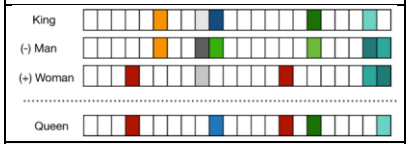
Die verschiedenen Wörter (bzw. ihre Indexe) werden über einen Algorithmus in einen «Dense» Vektor umgewandelt.

⇨ Ein Algorithmus dazu ist z.B. der «Embedding Layer».
⇨ Wie das genau funktioniert, ist in diesem Modul nicht relevant.

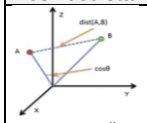
Bestimmung von Ähnlichkeiten

Bei «guten» Algorithmen stehen semantisch ähnliche Wörter nahe zueinander im Vektorraum. Wir können dadurch:

- Mit Addition/Subtraktion neue Wörter bilden.
- Durch die Berechnung der Kosinusdistanz die Ähnlichkeit von Wörtern bestimmen.



Kosinusdistanz



$$\cos\theta = \frac{A \circ B}{\|A\| * \|B\|} = \frac{\sum A_i * B_i}{\sqrt{\sum A_i^2} * \sqrt{\sum B_i^2}}$$

1: Starke Ähnlichkeit
0: Keine Ähnlichkeit (orthogonal)
−1: Umgekehrte Ähnlichkeit (falls nicht normalisiert)

Berechnung der Ähnlichkeit

elephant, mouse, bike

$$\cos\theta = \frac{E \circ M}{\|E\| * \|M\|} = \frac{52}{53.4} = 0.974$$

$$\cos\theta = \frac{E \circ B}{\|E\| * \|B\|} = \frac{26}{75.5} = 0.344$$

Hinweis: $\frac{3*2+6*7+2*2+1*0}{\sqrt{3^2+6^2+2^2+1^2}*\sqrt{2^2+7^2+2^2+0^2}}$

⇨ Also: Elefant und Maus sind ähnlicher als Elefant und Velo.

Datenvisualisierung

Kontext & Verwendung

Datenvisualisierung, also die graphische Darstellung von Daten, ist ein erster und wichtiger Schritt in ML. Man kann damit:

- Intuitives Verständnis der Daten erhalten
- Trends, Clusters und Muster erkennen
- Ausreisser & Spezialfälle identifizieren
- Hypothesen & Vermutungen validieren
- Resultate eindrücklich visualisieren

⇨ Für Python: «matplotlib» und dessen Erweiterung «seaborn»

Grafiken (Plots)

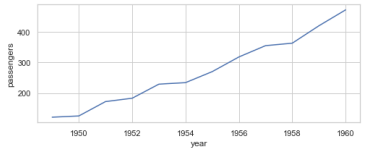
Grundbausteine

Jede sinnvolle Grafik benötigt mindestens diese 5 Informationen:

- Label der X-Achse
- Label der Y-Achse
- Titel bzw. Beschreibung
- Skala (z.B. linear oder logarithmisch)
- Dimension der Daten (z.B. 2D oder 3D)

Liniendiagramm

Ein Liniendiagramm zeigt ein Trend als ein Muster von Veränderung, normalerweise über eine Zeitachse hinweg.

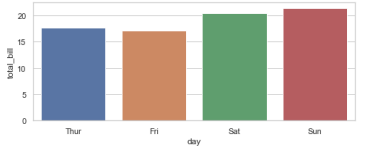



```
flights = sns.load_dataset("flights")
may = flights.query("month == 'May'")
sns.lineplot(data=may, x="year", y="passengers")
```

Balkendiagramm

Ein Balkendiagramm zeigt oftmals kategorisierte Daten, bei denen Zählungen basierend auf den Kategorien erfolgt.

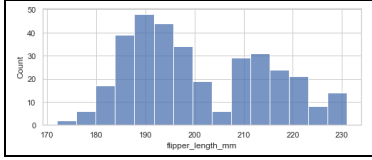
⇨ z.B. Trinkgeld pro Tag, Apps pro App-Store, etc.




```
tips = sns.load_dataset("tips")
sns.barplot(data=tips, x="day", y="total_bill")
```

Histogramm

Ein Histogramm zeigt die empirische Verteilung einer Variablen, aufgeteilt in ein oder mehrere Intervalle (Bins).

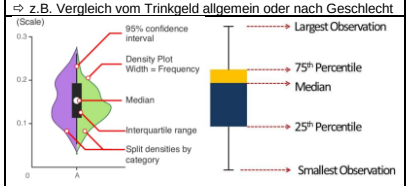



```
penguins = sns.load_dataset("penguins")
sns.histplot(data=penguins,
             x="flipper_length_mm", bins=15)
```

Violine Plot & Box Plot

Der Violine und der Box Plot zeigen beide eine informationsdichte Darstellung der wichtigsten Daten.

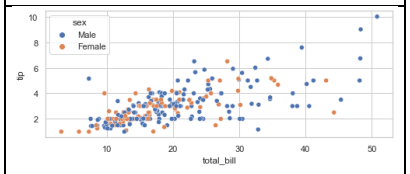
⇨ z.B. Vergleich vom Trinkgeld allgemein oder nach Geschlecht




```
tips = sns.load_dataset("tips")
sns.violinplot(data=tips,x="day",y="total_bill",
               hue="sex", split=True)
sns.boxplot(data=tips, x="total_bill", y=None)
```

Streudiagramm

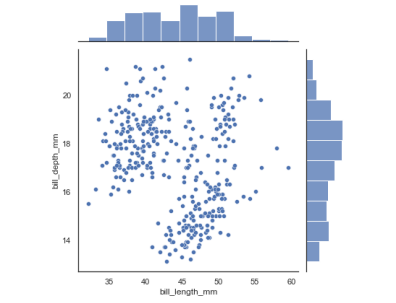
Das Streudiagramm zeigt die Beziehung zwischen zwei kontinuierlichen Variablen und visualisiert u.a. Korrelationen.




```
tips = sns.load_dataset("tips")
sns.scatterplot(data=tips, x="total_bill",
                y="tip", hue="sex")
```

Jointplot

Der Jointplot zeigt ein Streudiagramm, in welchem die empirische Verteilung der beiden Achsen am Rand als Histogramm dargestellt wird.




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

Wann verwendet man welchen Plot?

Meistens stimmt die Intuition beim Wählen eines passenden Plots. Alternativ kann auch die Grafik von Kaggle helfen.

⇨ Diese befindet sich am Ende der Zusammenfassung.

Lineare Regression

Kontext & Verwendung

Die lineare Regression ist eine einfache Methode für die Datenanalyse. Oftmals will man damit einer dieser zwei Dinge:

- Interpretation: Herausfinden, ob ein Input einen Einfluss auf den Output hat.
- Prediction: Voraussagen, wann etwas in der Zukunft passieren könnte.

Anwendung in Machine Learning

Die lineare Regression ist eine Methode aus dem «Supervised Learning». Ziel ist es, basierend auf den Inputdaten (X, Y) eine lineare Funktion zu finden, welche diese Datenpunkte optimal abbildet.

⇨ So kann z.B. für beliebige X-Werte ein Y gefunden werden.
⇨ Das Konzept stammt aus der Statistik.

1. Daten

Als Daten haben wir die $X = \{x_1, x_2 \dots\}$ mit einem oder mehreren Features, sowie die Labels $Y = \{y_1, y_2 \dots\}$.

⇨ Ein «Feature» ist ein «Merkmal», welches uns interessiert.
⇨ z.B. Person $x_i = (x_{i1} = \text{Alter},\ x_{i2} = \text{Gewicht},\ x_{i3} = \text{Grösse})$

2. Modell

Als Modell wird die lineare Funktion aus der Statistik / Mathematik verwendet:

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

⇨ Wobei a «Slope» und b «Intercept» genannt wird.
⇨ In diesem Fall gehen wir von nur einem Feature aus.

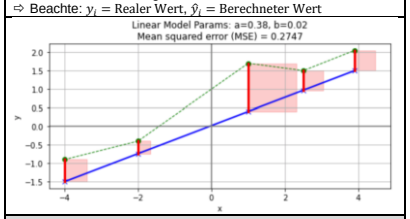
3. Cost-Function (Loss)

Wie «gut» die Funktion auf die Inputdaten passt, können wir über den «Mean Squared Error» (MSE) herausfinden:

$$E = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 \quad \text{wobei } e_i = y_i - \hat{y}_i$$

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

⇨ Wobei e «Residual» genannt wird.
⇨ Beachte: y_i = Realer Wert, $\hat{y}_i$ = Berechneter Wert



4. Optimization Procedure

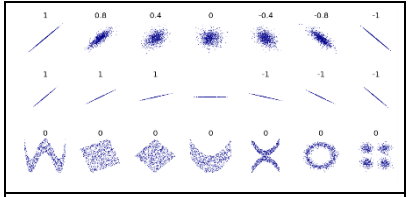
Um nun das optimale Modell zu finden, müssen wir bei der lin. Regression das optimale a und b finden. Dazu verwenden wir «Stochastic Gradient Descent».

⇨ Siehe «Stochastic Gradient Descent» weiter unten.

Korrelationen

Die lin. Regression kann ausschliesslich lineare Zusammenhänge (Korrelationen) erkennen. Wie stark diese ist, sagt uns der «Pearson Correlation Coefficient».

⇨ Wichtig: Korrelation ≠ Kausalität



Selbst wenn der Koeffizient 0 ist können die Daten dennoch strukturiert sein.

⇨ Dies lässt sich z.B. oben in der letzten Zeile erkennen.

Komplexere Modelle

Wie erwähnt kann ein x auch mehrere Features besitzen. In diesem Fall wird x zu einem Vektor mit p Dimensionen. Das Modell ändert sich wiefolgt:

$$\hat{y}_i = \beta_0 + \beta_1 * x_{i1} + \beta_2 * x_{i2} + \dots + \beta_p * x_{ip}$$

Oder in der Matrix-Schreibweise:

$$X * \beta = y$$

⇨ Wobei p der Anzahl von Features entspricht.

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

Stochastic Gradient Descent

Kontext & Verwendung

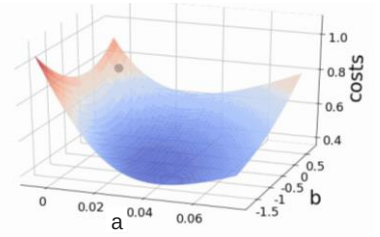
Gradient Descent ist eine iterative Optimierungsfunktion, in der bei jeder Iteration die Modell-Parameter angepasst werden, um den Loss zu reduzieren.

⇨ Für lin. Regression also: Bei jeder Iteration werden a und b angepasst, um den MSE zu reduzieren.

Was ist eine «Error Landscape»?

Erstellen wir aus Loss und Modell-Parametern eine Grafik, so ergibt sich eine Landschaft mit «Bergen» und «Tälern».

⇨ In den «Tälern» ist der Loss minimal. Da wollen wir hin!



«Mathematische» Gradient

Ein Gradient besteht aus den partiellen Ableitungen eines Punktes und gibt die Richtung der grössten Änderung an.

⇨ Oder anders: Ein Gradient zeigt uns den Weg ins «Tal».
⇨ Bei lin. Regression: MSE nach a und b ableiten und in Vektor.

$$\text{Gradient } E = \begin{bmatrix} \frac{\delta E}{\delta a} \\ \frac{\delta E}{\delta b} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (a * x_i + b)\right)(-x_i) \\ \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (a * x_i + b)\right)(-1) \end{bmatrix}$$

Problem

Der Gradient geht in die verkehrte Richtung (vom Min. zum Max.) und macht zu grosse Schritte. Wir können dies mit einem negativen Skalierungsfaktor lösen:

$$\text{Gradient } E = -\alpha * \begin{bmatrix} \frac{\delta E}{\delta a} \\ \frac{\delta E}{\delta b} \end{bmatrix}$$ wobei α «Learning Rate» heisst.

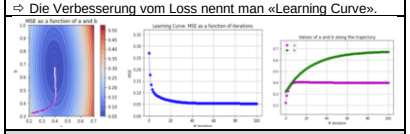
Iterationsprozess

Um nun das Minimum vom Loss zu finden, berechnen wir Schrittweise den Weg entlang des Gradienten ins «Tal».

$$\begin{bmatrix} a \\ b \end{bmatrix}_{t+1} = \begin{bmatrix} a \\ b \end{bmatrix}_{t} - \alpha * \begin{bmatrix} \frac{\delta E}{\delta a} \\ \frac{\delta E}{\delta b} \end{bmatrix}_{t}$$

Dies läuft nun so lange, bis eine vordefinierte Anzahl Iterationen erreicht wurde.

⇨ Die Verbesserung vom Loss nennt man «Learning Curve».



Und bei mehreren «Tälern»?

Hat ein Gradient mehrere «Täler», also Wertepaare, die den Loss minimieren, so müssen wir auch diese finden.

⇨ Lösung: Beginne an verschiedenen Startpunkten und wähle am Ende einfach das kleinste Resultat.

Problem

Wenn wir uns die Gradient-Funktion ansehen, fällt auf, dass wir bei jeder Iteration den MSE neu berechnen müssen.

⇨ Da wir bei jeder Iteration ein neues a und b erhalten.
⇨ Dies ist bei grossen Datensätzen enorm ineffizient.

Stochastic Gradient Descent (SGD)

Beim Stochastic Gradient Descent wird nur eine Teilmenge der Datenpunkte für die Berechnung vom Loss verwendet.

⇨ Diese Teilmenge wird bei jeder Iteration zufällig gewählt.

Bei der Teilmenge J mit n Elementen, geht man dabei davon aus, dass:

$$\frac{\delta E}{\delta a} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (a * x_i + b)\right)(-x_i)$$
$$\approx \frac{1}{n}\sum_{j \in J_n}\left(y_j - (a * x_j + b)\right)(-x_j)$$

Sowie für den Parameter b:

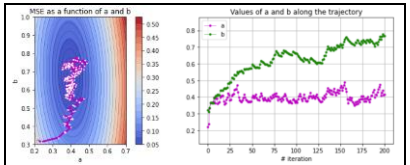
$$\frac{\delta E}{\delta b} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (a * x_i + b)\right)(-1)$$
$$\approx \frac{1}{n}\sum_{j \in J_n}\left(y_j - (a * x_j + b)\right)(-1)$$

Batch-Size

Die Grösse n der Teilmenge J (auch «Batch-Size») kann frei gewählt werden. Es gibt keinen optimalen Wert für n.

- Kleines n: Der Gradient wird unruhig.
- Grosses n: Die Rechenkosten steigen.

⇨ Oft wählt man den Wert $n = 32$ oder $n = 64$.

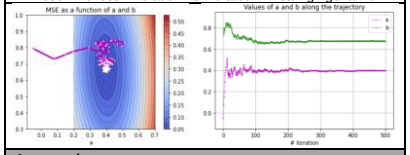


⇨ Ein unruhiger Gradient mit kleiner Batch-Size

Annealed Learning Rate

Wie oben erkennbar ist der Gradient besonders beim Minimum sehr unruhig. Wir können dieses Problem lösen, indem wir die Schrittgrösse α bei jeder Iteration etwas verkleinern.

⇨ Dieser Prozess wird auch «Simulated Annealing» genannt.



Anmerkungen

Weitere Anmerkungen zu SGD:

- Die Loss-Funktion muss ableitbar sein.
- Das Berechnen des Gradienten kann trotz Optimierungen sehr aufwändig sein.
- SGD dient insbesondere als Baustein für andere Methoden wie Adam, Adagrad, etc.

Generalisierung & Regularisierung

Kontext & Verwendung

Wir haben in den vorherigen Schritten immer den Loss, also den Fehler innerhalb der definierten Inputdaten (X, Y), angeschaut. Doch wie können wir herausfinden, ob unser Modell auch gut auf neue, unbekannte Daten funktioniert?

Generalization Error

Der Generalization Error gibt an, wie «gut» ein Modell mit neuen, unbekannten Daten funktioniert.

⇨ Doch wie können wir «unbekannte» Daten evaluieren?

Wir können diesen Error annäherungsweise bestimmen, wenn wir die vorhandenen Daten in zwei Teile unterteilen:

- Trainings-Daten: Trainieren das Modell und definieren den Loss (In-Sample Error)
- Test-Daten: Testen das Modell und definieren den Generalization Error (Out-Of-Sample Error)

⇨ Oft trennt man die Daten im Verhältnis 80% zu 20%.

Wann sind Modelle «schlecht»?

Grundsätzlich wollen wir den Generalization Error immer möglichst tief halten. Ist dieser Error hoch, so liegt meistens eines der folgenden zwei Probleme vor.

Overfitting

Das Modell ist zu komplex und beschreibt Daten zu genau. Es folgt:

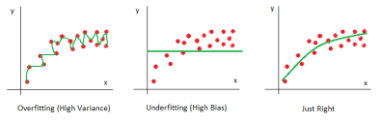
- Der «In-Sample Error» ist minimal und kann teilweise (je nach Modell) sogar 0 sein.
- Der «Out-Of-Sample Error» ist sehr gross.

Underfitting

Das Modell ist zu einfach und beschreibt die Daten zu wenig genau. Es folgt:

- Der «In-Sample Error» ist sehr gross.
- Der «Out-Of-Sample Error» ist sehr gross.

Beispiel

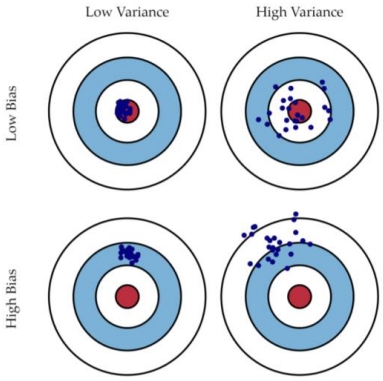


Bias & Variance

Die zuvor beschriebenen Phänomene lassen sich auch mathematisch mit «Bias» und «Variance» ausdrücken.

- Bias: Wie «gut» passt das Modell auf die bekannten Daten (Loss)
- Variance: Wie «gut» passt das Modell auf unbekannte Daten (Generalization Error)

Darstellung im Diagramm



Beispiel für Bias & Variance

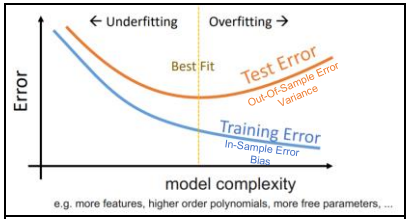
Ich stelle eine Frage im Klassenchat:

- Low B., Low V.: Alle Mitstudis geben die gleiche, korrekte Antwort.
- High B., Low V.: Alle Mitstudis geben die gleiche, fast oder teilweise korrekte Antwort.
- Low B., High V.: Alle Mitstudis geben eine andere, manchmal korrekte und manchmal falsche Antwort.
- High B., High V.: Alle Mitstudis geben eine andere, aber alle eine inkorrekte Antwort.

⇨ Man erkennt, dass eine tiefe Varianz meistens besser ist.

Regularisierung

Wie wir beim Over- & Underfitting gesehen haben, stehen Bias & Variance immer miteinander im Konflikt. Doch wie finden wir nun das Optimum?



An der Grafik oben ist zu erkennen, dass es irgendwo in der Mitte ein Optimum gibt. Dieses können wir mithilfe von Regularisierung finden.

⇨ Wichtig ist besonders die Variance, Bias ist eher sekundär.

Funktionsweise

Wir benötigen dazu zwei Komponenten:

1. Ein Weg, die Komplexität zu messen.
2. Ein Weg, die Komplexität zu kontrollieren.

Messung der Komplexität

Die Komplexität lässt sich über einen Normierungsfaktor messen:

$\sum_{j=1}^{p}|\beta_j|$ L1-Norm (Lasso)

$\sum_{j=1}^{p}\beta_j^2$ L2-Norm (Ridge)

⇨ Siehe «Komplexe Modelle» für Details (Erinnerung: $X * \beta = y$)

Kontrolle der Komplexität

Wir können die Komplexität kontrollieren, indem wir einen «Penalty-Term» der Loss-Funktion hinzufügen.

$$\text{Regularized Loss} = \text{Loss} + \lambda * \text{Norm}$$

⇨ Somit erhalten komplexere Modelle eine höhere Strafe.
⇨ Den Hyperparameter λ können wir beliebig festlegen.

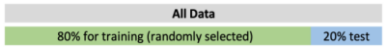
Beispiel eines MSE mit L2-Norm:

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

Cross-Validation

Kontext & Verwendung

Ohne Cross-Validation berechnen wir Modelle und deren Generalization Error mit Daten, welche einmal zu Beginn in Test- und Training-Set unterteil wurden.



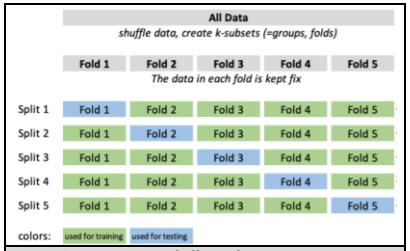
Die Idee von Cross-Validation ist, dass wir stattdessen mehrere Sets aus den Daten bilden und mit diesen mehrere Modelle berechnen können.

⇨ Wir können dann z.B. das beste Modell auswählen.

k-Fold Cross-Validation

Bei diesem Verfahren werden alle Datenpunkte zufällig in k Folds unterteilt. Anschliessend werden k Trainingseinheiten (Splits) durchgeführt, in welchen jeder Fold genau einmal als Test-Datensatz verwendet wird. Mit den anderen Daten werden die Modelle trainiert.

⇨ Die Daten in einem Fold werden nicht mehr verändert.
⇨ Weiteres Preprocessing muss pro Split angewant werden.



Use-Case 1: Modellanalyse

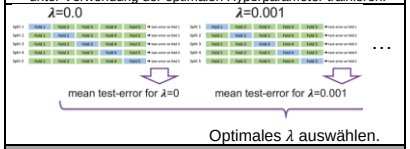
Jeder Split liefert uns einen anderen Generalization Error. Wir können so den durchschnittlichen Error so wie dessen Standardabweichung bestimmen.

⇨ Das ist genauer als mit nur einem «modellspezifischen» Error.

Use-Case 2: Hyperparameter

Weiter können wir Cross-Validation verwenden, um die optimalen Hyperparameter für unser Modell zu bestimmen.

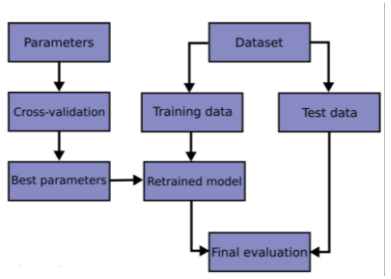
⇨ Wir trainieren mehrere Modelle mit anderen Hyperparameter.
⇨ Anschliessen kann man z.B. ein Modell mit allen Testdaten unter Verwendung der optimalen Hyperparameter trainieren.



Cross-Validation (scikit-learn)

Je nach Use-Case können einige Daten zu Beginn herausgenommen werden, die man nie zum Training verwendet.

⇨ Diese Daten werden dann für die Endvalidierung verwendet.
⇨ In den meisten Fällen ist dieser Ansatz zu empfehlen.



Logistic Regression

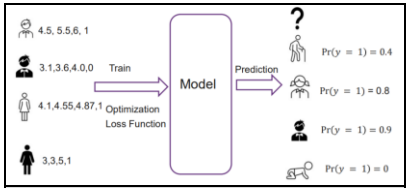
Kontext & Verwendung

Die logistische Regression ist eine Methode zur Binärklassifizierung, also die Aufteilung von Daten in zwei Klassen.

⇨ z.B. Wird es basierend auf den Daten regnen? (Ja/Nein)

Anwendung in Machine Learning

Die logistische Regression kommt aus dem «Supervised Learning». Ziel ist es, basierend auf den Inputdaten (X, Y) eine Funktion zu finden, welche die Daten optimal in zwei Klassen abbildet.



1. Daten

Als Daten haben wir die $X = \{x_1, x_2 \dots\}$ mit einem oder mehreren Features, sowie die binären Labels $Y = \{y_1, y_2 \dots\}$.

⇨ Man kann also «mathematisch» sagen, dass $y \in \{1,0\}$.
⇨ Erinnerung: Ein Feature kann z.B. $x_i = (x_{i1}, x_{i2}, x_{i3})$ sein.

2. Modell

Als Modell wird die sigmoidale Funktion aus der Mathematik verwendet. Sie gibt die Wahrscheinlichkeit P an, dass ein x der Klasse $y = 1$ angehört.

$$P(x_i) = \frac{1}{1 + e^{-(W^T * x_i)}}$$

⇨ Das $(W^T * x_i)$ wird oft auch y oder z genannt.

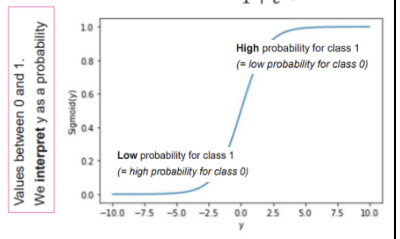
Ausserdem ist definiert, dass:

$$W^T * x_i = w_1 * x_{i1} + w_2 * x_{i2} + \cdots$$

⇨ Ziel vom ML-Algorithmus wird sein, W zu optimieren.

Sigmoidfunktion

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



Thresholding

Da wir als Resultat von unserem Modell aber keine Wahrscheinlichkeit, sondern eine Klasse erwarten, müssen wir noch einen «Threshold» definieren. z.B.:

- Wenn $P(x_i) > 0.5$, dann Klasse 1.
- Wenn $P(x_i) < 0.5$, dann Klasse 2.

⇨ Wie man diesen Threshold bestimmt, siehe weiter unten.

3. Cost-Function (Loss)

Wie «gut» unser Modell ist, hängt davon ab, wie wahrscheinlich eine Zuweisung in die korrekte Klasse ist.

⇨ Der Loss soll also inkorrekte Zuweisungen bestrafen.

Maximum Likelihood

Max. Likelihood ist eine Funktion, die oft in der Binärklassifizierung verwendet wird. Sie minimiert den Loss, wenn:

- $P(x) = 1$ und $y = 1$ ■
- $P(x) = 0$ und $y = 0$ ■

Und maximiert den Loss wenn:

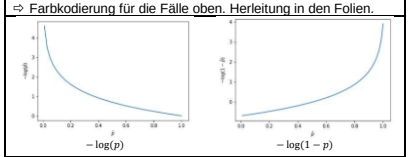
- $P(x) = 0$ und $y = 1$ ■
- $P(x) = 1$ und $y = 0$ ■

⇨ We want to «maximize the likelihood» of a correct prediction.

Die Funktion sieht wiefolgt aus:

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

⇨ Achtung: Hier steht log() für den natürlichen Logarithmus ln().
⇨ Farbkodierung für die Fälle oben. Herleitung in den Folien.



4. Optimization Procedure

Um das optimale Modell zu finden, müssen wir bei der log. Regression das optimale W finden. Da die Loss-Funktion konvex ist, können wir auch hier «Stochastic Gradient Descent» verwenden.

⇨ «konvex» heisst «gewölbt», bzw. die Funktion hat «Täler».

Classifier Evaluation

Kontext & Verwendung

In der «Classifier Evaluation» wollen wir die Qualität unseres Modells evaluieren.

⇨ Im Unterricht diente «Logistic Regression» als Beispiel.

Confusion Matrix

Die Conf. Matrix erlaubt die Evaluation eines «Logistic Regression»-Modells. Sie gibt das Verhältnis zwischen korrekten und inkorrekten Zuteilungen an.

		Voraussage der AI	
		Positiv	Negativ
Reale Wert	Positiv	t_p **True** Positiv	f_n **False** Negativ
	Negativ	f_p **False** Positiv	t_n **True** Negativ

Parameter

Aus der Conf. Matrix lassen sich nun verschiedene Parameter bestimmen.

- **Mean Accuracy:** Wie oft sind wir korrekt?
  $$\text{Accuracy} = (t_p + t_n)/n$$
- **Mean Error:** Wie oft sind wir inkorrekt?
  $$\text{Error} = (f_p + f_n)/n$$
- **Precision:** Wenn wir «Wahr» sagen, wie oft sind wir in diesem Fall korrekt?
  $$\text{Precision} = t_p/(t_p + f_p)$$
- **Recall (TPR):** Wie oft sagen wir «Wahr», wenn der reale Wert «Wahr» sein sollte.
  $$\text{Recall} = t_p/(t_p + f_n)$$
- **Fall-Out (FPR):** Wie oft sagen wir «Wahr», wenn der reale Wert «Falsch» sein sollte.
  $$\text{Fall Out} = f_p/(f_p + t_n)$$
- **Miss Rate:** Wie oft sagen wir «Falsch», wenn der reale Wert «Wahr» sein sollte.
  $$\text{Miss Rate} = 1 - \text{Recall}$$

⇨ Recall heisst auch Sensitivity / True Positive Rate (TPR)
⇨ Fall Out heisst auch False Positive Rate (FPR)
⇨ Miss Rate heisst auch False Negative Rate (FNR)

Precision vs. Recall

Das Erhöhen der Precision reduziert den Recall und umgekehrt. Die Applikation entscheidet, was wichtiger ist.

- CIA-Gesichtsscan: Hohe Precision. Soll ausschliesslich korrekte Personen erkennen.
- Dieb-Erkennung: Hoher Recall. Kann auch mal eine inkorrekte Warnung auslösen.

Threshold

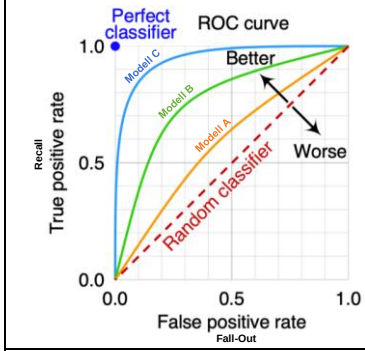
Der Threshold bei der log. Regression lässt sich nicht allgemein «mathematisch» bestimmen, da er stark vom jeweiligen Use-Case abhängt.

⇨ Wir können ihn jedoch anhand Mathematik «begründen».

Receiver Operating Characteristics

Um einen optimalen Threshold zu finden, können wir mehrere Modelle mit unterschiedlichen Thresholds trainieren und bei jedem Modell «Miss Rate» und «Recall» ausrechnen.

⇨ Wir können diese nun in einem «ROC-Space» darstellen.



Grundsätzlich gilt nun: Je grösser die Fläche unter der Kurve, desto besser ist der Threshold und somit das Modell.

⇨ Alternative: Schnittpunkt mit einem Precision-Recall Plot.
⇨ Schlussendlich entscheidet aber immer der Use-Case.

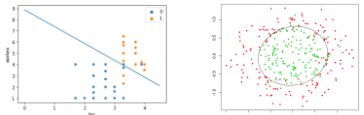
K-Nearest Neighbors (KNN)

Kontext & Verwendung

Decision Boundary

Lassen sich die Datenpunkte in zwei Klassen unterteilen, so können wir oftmals eine Funktion finden, welche die Daten sinnvoll klassifizieren kann.

⇨ Diese Funktion nennen wir «Decision Boundary».
⇨ Ist die Funktion linear, nennt man die Daten «linear trennbar».



Problem

Lassen sich die Daten über eine «Decision Boundary» klassifizieren, so können wir log. Regression anwenden. Dies hat jedoch einige Nachteile:

- Das Training der Modelle ist aufwändig.
- Die Decision Boundary ist schnell komplex.
- Wir sind auf zwei Klassen beschränkt.

⇨ Andere Methoden wie KNN können diese Probleme lösen!

K-Nearest Neighbors

KNN kommt zum Zug, wenn man nicht-binäre Daten klassifizieren will, welche keine «Decision Boundary» besitzen.

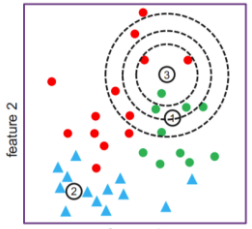
Anwendung in Machine Learning

K-Nearest Neighbors kommt aus dem «Supervised Learning». Bei dieser Methode wird kein Modell trainiert, stattdessen wird die «Predicition» basierend auf den Inputdaten (X, Y) berechnet.

Funktionsweise

Bei KNN wird die Klassifizierung eines neuen Datenpunktes anhand der k nächsten Nachbaren berechnet. Das Resultat ist die Klasse, welche unter den Nachbaren am meisten vorkommt.

⇨ «A datapoint is known by the company it keeps.»



Was bedeutet «nearest»?

Die «Nähe» eines Nachbaren wird anhand einer Distanz-Metrik bestimmt.

⇨ D.h. Berechne zuerst die Distanz zu allen Datenpunkten.
⇨ Anschliessend wählt man die k nächsten Nachbaren aus.

Distanz-Metriken

Diese Metriken wurden behandelt:

- Kosinusdistanz: $\cos\theta = \frac{x_1 \circ x_2}{\|x_1\| * \|x_2\|}$
- Manhattan-Distanz: $d_M = \sum_{i=1}^{n}|x_{1i} - x_{2i}|$
- Euclidean-Distanz: $d_E = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$
- Minkowski: $d_{MIN} = (\sum_{i=1}^{n}|x_{1i} - x_{2i}|^p)^{1/p}$

⇨ Minkowski mit $p = 1$: Manhattan-Distanz
⇨ Minkowski mit $p = 2$: Euclidean-Distanz

Welche Distanz und welches k?

Die Wahl dieser zwei Parameter ist wie so oft frei. Mit diesem Vorgehen können wir sie aber gut bestimmen:

- Test- und Trainingsdaten definieren
- Cross-Validation durchführen
- Distanz-Metrik und k variieren
- Performance überwachen

⇨ Hinweis: Wird $k = n$ gewählt, so werden alle Datenpunkte der «gleichen» Klasse mit den meisten Punkten zugeordnet.

Vor- und Nachteile von KNN

Diese Punkte sind zu beachten:

\+ Das Modell ist enorm einfach
\+ Es hat wenige Hyperparameter
\- Das k muss gut gewählt werden
\- Die Distanzberechnung ist aufwändig
\- Ineffizent bei vielen Features (Dimensionen)
\- Die Skalierung der Features muss stimmen

Clustering

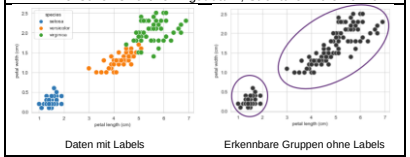
Kontext & Verwendung

Selbst wenn man Daten ohne Informationen (Labels) hat, können wir etwas von ihnen lernen: Die Struktur.

⇨ Alle Daten haben eine Struktur (evtl. durch Noise verborgen).

Beim Clustering versuchen wir, die Daten in «ähnliche» Gruppen zu unterteilen. Diese nennen wir «Cluster».

⇨ Wir Menschen sind enorm gut darin, Strukturen zu erkennen.



Anwendung in Machine Learning

Clustering kommt aus dem «Unsupervised Learning». Ziel ist es, basierend auf den Inputdaten (X) Clusters zu finden, welche die Daten optimal zuordnen.

⇨ z.B. Soziale Netzwerke, Google News, etc.
⇨ Idee: Ähnliche Datenpunkte sind nahe beieinander (KNN).

Naïve K-means (NKM)

Der «Naïve K-means»-Algorithmus basiert auf dem KNN und ermöglicht es, die Cluster in den Daten zu bestimmen.

1. Definiere die Anzahl der Cluster k_c
2. Initialisiere die k_c Zentren $(C_1, C_2 \ldots C_{k_c})$
3. Berechne die Distanz zwischen jedem der n Datenpunkte und den k_c Zentren.
4. Ordne jeden Datenpunkt seinem nächsten Zentrum C zu.
5. Berechne die neuen Zentren C aus dem Durchschnitt der zugeordneten Punkte.
6. Wiederhole ab 3. bis zum Abbruchkriterium

Was sind die Abbruchkriterien?

Es gibt verschiedene Kriterien, wann wir den Algorithmus abbrechen wollen:

- Wenn die Zentren nicht mehr ändern.
- Die Punkte im Cluster nicht mehr ändern.
- Die Distanz der Datenpunkte >= einem definierten Threshold sind.
- Eine fixe Anzahl Iterationen erreicht wurde.

⇨ Die ersten zwei Punkte sind dabei sehr Zeitintensiv.

Wie initialisieren wir die Zentren?

Die NKM-Perfomance hängt stark mit der Wahl der Startzentren zusammen.

- Einige Zentren können zu schlechten Clustern oder «Convergence Rates» führen.
- Nahe Zentren brauchen mehr Iterationen.

Als Grundsatz gilt daher:

- Initialisiere zufällige und mehrmals.
- Kontrolliere, dass die Cluster stabil bleiben.

⇨ «Convergence» bedeutet die «Annäherung» an die Cluster.

Wieso braucht es Standardisierung?

Beim Clustering kann es vorkommen, dass Features mit grossen Werten, die mit kleinen Werten dominieren.

$$d_E = \overset{x_{i1}}{(3-6)^2} + \overset{x_{i2}}{(4-6)^2} + \overset{x_{i3}}{(10-30)^2}$$
$$= 9 + 4 + 400 \Rightarrow x_{i3} \text{ dominiert!}$$

Wir müssen die Werte also vor dem Clustering normalisieren.

⇨ «Normalisieren» heisst, in dieselbe Skala bringen.

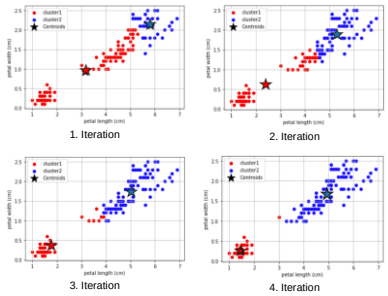
Wie viele Cluster k_c?

Die Anzahl der Cluster lassen sich über die nachfolgenden Qualitätskriterien bestimmen. Grundsätzlich gilt aber:

$$1 < k_c < N$$

⇨ Bei $k_c = 1$ wären alle Datenpunkte im selben Cluster.
⇨ Bei $k_c = N$ wären alle Datenpunkte ein eigenes Cluster.

Visualisierung im Diagramm



⇨ Resultat eines «Naïve K-means»-Algorithmus

Qualität des Clustering

Wir bezeichnen ein Clustering als «gut», wenn die Distanz von jedem Punkt im Cluster zu dessen Zentrum minimal ist.

⇨ Anders als beim «Supervised Learning» können wir hier leider nicht auf Metriken wie Accuracy oder Recall zurückgreifen.

Inertia / WCSS

Das Inertia ist die quadrierte Distanz aller Punkte zum nächsten Zentrum.

$$\sum_{i=1}^{n} \|x_i - C_{\text{Closest}}\|^2$$

Diesen Wert wollen wir *minimieren*.

⇨ WCSS bedeutet «within-cluster sum-of-squares»

	Squared Euclidean distance from red centre	Squared Euclidean distance from red centre
	1.25,2	2.5, 1
1,3	$(1.25-1)^2 + (3-2)^2 = 1.0625$	-
1,2	0.0625	-
1,1	1.0625	-
2,1	-	0.25
2,2	0.5625	-
3,1	-	0.25
WCSS	**2.75**	0.5

Das Inertia wird kleiner («besser»), je mehr Cluster wir haben. Zu viele Cluster sind jedoch nicht immer optimal.

Silhouette Score

Der Silhouette Score beschreibt den Abstand von den Datenpunkten in einem zu den Punkten im anderen Cluster.

$$\text{Silhoutte von } x_i = \frac{b-a}{\max(a,b)}$$

- a = Die durchschnittliche Distanz von x_i zu allen anderen Punkten im Cluster.
- b = Die durchschnittliche Distanz von x_i zu allen Punkten im «nächsten» Cluster.

Der Silhouette Score vom gesamten Modell sieht dann wiefolgt aus:

$$\frac{1}{n}\sum_{i=1}^{n} \text{Silhoutte}(x_i)$$

Diesen Wert wollen wir *maximieren*.

⇨ Der Silhouette Score liegt immer im Bereich 1 bis –1.

	Distance from other points in the cluster				
	1,3	1,2	1,1	2,2	a
1,3	-	1	2	$\sqrt{2}$ = 1.41	1.47
1,2	1	-	1	1	1
1,1	2	1	-	$\sqrt{2}$	1.47
2,2	$\sqrt{2}$	1	$\sqrt{2}$	-	1.27

	Distance from points in the GREEN cluster		
	2,1	3,1	b
1,3	$\sqrt{5}$	$\sqrt{8}$	3.65
1,2	$\sqrt{2}$	$\sqrt{5}$	1.8
1,1	1	2	1.5
2,2	1	$\sqrt{2}$	1.2

z.B. $\text{Silhoutte von } (1,3) = \frac{3.65-1.47}{3.65} = 0.59$

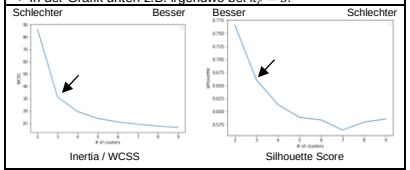
Errinnerung «Distanz»: $d_E = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$

Die Silhouette wird kleiner («schlechter»), je mehr Cluster wir haben.

Die optimale Anzahl von Cluster?

Die «beste» Anzahl für k_c ist ein Kompromiss zwischen den beiden Scores.

⇨ In der Grafik unten z.B. irgendwo bei $k_c = 3$.



Ensemble Methods

Kontext & Verwendung

Beim Ensemble wollen wir unsere Prediction nicht mehr von einem einzigen, «perfekten» Modell erhalten, sondern von mehreren Modellen aggregieren.

⇨ Diese Modelle können auch «schlechter» sein.
⇨ Ensemble steht für «A group of Predictors».

Das Konzept basiert auf «Wisdom of the Crowd». Wir haben mehrere Modelle und wählen das Ergebnis aus, welches am häufigsten berechnet wurde.

⇨ Diese Variante ist oft besser als ein «perfektes» Modell!

Ensemble wird vor allem am Ende eines Projekts eingesetzt, wenn:

- Bereits gute Modelle erstellt wurden.
- Genügend viele Modelle vorhanden sind.
- Die Modelle divers sind.

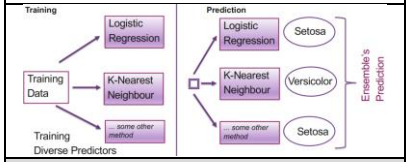
Was lässt sich kombinieren?

Beim Ensemble können sich die Modelle stark voneinander unterscheiden.

⇨ Wir können z.B. Algorithmen, Daten und Parameter variieren.

Algorithmen

Wir können z.B. in einem Modell KNN und in einem anderen die logistische Regression verwenden.



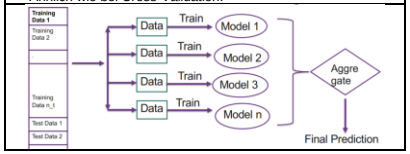
Hyperparameter

Wir können z.B. pro Modell eine unterschiedliche Anzahl Klassen k_c oder Regularisierungsparameter definieren.

Daten

Wir können z.B. pro Modell unterschiedliche Datensätze oder Features haben.

⇨ Ähnlich wie bei Cross-Validation.



Welches Resultat wird gewählt?

Hard-Voting

Beim Hard-Voting wird einfach das Resultat genommen, welches von den meisten Modellen berechnet wurde.

Soft-Voting

Ist das Resultat eine Wahrscheinlichkeit, so können wir auch den Durchschnitt von den Modellberechnungen nehmen.

⇨ Dieses Verfahren nennt man «Soft-Voting»

Wie werden die Daten verteilt?

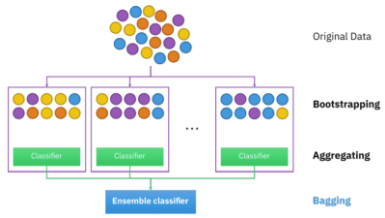
Sampling Without Replacement Pasting

Beim «Pasting» kann ein Datenpunkt nur von einem einzigen Modell verwendet werden.

Sampling With Replacement Bagging

Beim «Bagging» können dieselben Datenpunkte von mehreren Modellen verwendet werden.

⇨ Der Begriff kommt von «Bootstrap Aggregating».



⇨ Die Daten werden dabei meistens zufällig eingeteilt.

Variationen von Sampling

Wir können Sampling sowohl für Daten als auch für Features verwenden. Die Variationen heissen dabei:

- Random Patches: Sampling mit Daten und Features
- Random Subspaces: Sampling nur mit Features

Out of Bag (oob) Evaluation

Bei der zufälligen Einteilung der Datenpunkte kann es vorkommen, dass einige Punkte nie für das Training verwendet werden. Wir nennen sie OOB-Punkte.

⇨ Wir können diese Punkte für die Evaluation brauchen!

No-free-Lunch-Theorem

Warum sollen wir überhaupt mehrere Modelle trainieren, wenn wir auch ein «perfektes» Modell suchen könnten?

Dafür gibt es eine informelle Begründung, das No-free-Lunch-Theorem:

«no single machine learning algorithm is universally the best-performing algorithm for all problems»

Codebeispiel

```
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
```

Artificial Neural Networks (ANN)

Ein ANN ist ein moderner ML-Algorithmus, welcher aus mehreren, vom Gehirn inspirierten Neuronen besteht.

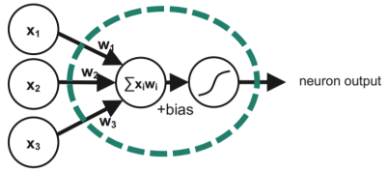
Artificial Neurons

Ein Artificial Neuron ist eine «einfache» Recheneinheit bestehend aus:

- Einem Inputvektor $X = [x_1, x_2 \dots]$.
- Gewichte für die Inputs $W = [w_1, w_2 \dots]$.
- Ein Bias / Intercept b.
- Eine nicht-lineare Aktvierungsfunktion.

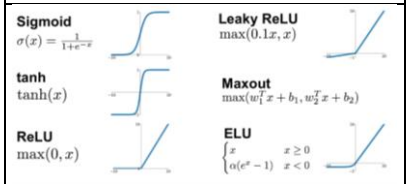
Jedes Neuron berechnet das Skalarprodukt der Inputs & Gewichte, addiert den Bias und sendet den Wert durch die Aktivierungsfunktion.

⇨ Das b und $W = [w_1, w_2 \dots]$ sind die trainierbaren Parameter.
⇨ Eine Gruppe von ANs mit denselben Inputs heisst «Layer».



Aktivierungsfunktion

Es gibt zahlreiche Aktivierungsfunktionen für ANNs. Die bekannteste heisst «Rectified Linear Unit» (ReLU).



Training

Grundsätzlich funktioniert das Training von ANNs genau gleich wie bei anderen ML-Algorithmen. Wir haben Inputdaten (X,Y), einen Loss (z.B. MSE) und einen Optimizer (z.B. SGD).

⇨ Wir optimieren also in jeder Iteration alle b und W.
⇨ Die Ableitungen werden mit «Backpropagation» berechnet.
⇨ Ohne den «BP»-Algorithmus gäbe es keine ANNs.

Diverses