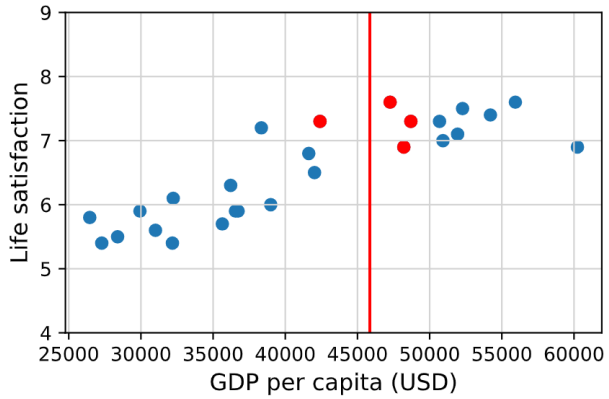
MACHINE LEARNING 1 – LINUS STUHLMANN

NEAREST NEIGHBOUR REGRESSION

Die Vorhersage eines Wertes y unter Verwendung der k nächstgelegenen Werte (Nachbarn) wird als "Nearest Neighbours Regression" bezeichnet. Man könnte z. B. $k = 4$ verwenden und den Durchschnittswert dieser Werte berechnen, um einen Wert x vorherzusagen. Es gibt keine theoretische Spezifikation für die optimale Anzahl von k, daher muss sie empirisch bestimmt werden.



Die Hauptnachteile dieses Ansatzes sind:

- Der Algorithmus muss für jede Vorhersage alle Trainingsdaten berücksichtigen.
- Er ist empfindlich gegenüber Ausreissern und Rauschen.
- Trainingsmuster ausserhalb des Bereichs von k werden bei der Vorhersage nicht berücksichtigt.

LINEAR REGRESSION

UNIVARIATE LINEAR REGRESSION

Das lineare Regressionsmodell ist definiert als:

$$h(x;\ \theta_0, \theta_1) = \theta_0 + \theta_1 x$$

θ_0 stellt den Schnittpunkt mit der y-Achse dar und θ_1 ist die Steigung einer Dehnungslinie.

Mit Werten für θ_0 & θ_1 und einem gegebenen Wert $\{x_m\}$, macht das Modell eine Vorhersage $\hat{y}_m$.

$$\hat{y}_m = h(x_m;\ \theta_0, \theta_1) = \theta_0 + \theta_1 x_m$$

TRAINING A SIMPLE REGRESSION MODEL

Das Training eines univariaten Regressionsmodells zielt darauf ab, die Werte für die Parameter θ_0 & θ_1 mit der besten Anpassung an die Trainingsdaten zu finden, d.h. ein Modell (eine gerade Linie) mit der kleinsten Abweichung von den Daten zu finden. Die Trainingsdaten bestehen aus M Stichproben von Werten (x_m, y_m). x_m stellt die Eingabewerte und y_m die bekannte Ausgabe dar.

LOSS FUNKTION

Eine Loss Function, misst den Fehler eines einzelnen Trainingsbeispiels. Eine Methode zur Messung des Fehlers oder der Anpassung eines Modells ist die RSS (Residual Sum of Squares).

$$\mathcal{L}_{RSS}(\theta_0, \theta_1; \{x_m, y_m\}) = \sum_{m=1}^{M} (y_m - \hat{y}_m)^2 = \sum_{m=1}^{M} \epsilon_m^2$$

$\mathcal{L}_{RSS}(\theta_0, \theta_1; \{x_m, y_m\})$ ist die Summe der quadrierten Residuen, d. h. die quadrierte Abweichung des vorhergesagten Wertes $\hat{y}_m$ vom tatsächlichen Wert von y_m.

COST FUNCTION

Das Lernen oder Trainieren des Modells besteht in der Minimierung einer Kostenfunktion J. In diesem Fall wird die Kostenfunktion durch die RSS dargestellt.

$$J(\theta_0, \theta_1) = \frac{1}{2M} \sum_{m=1}^{M} (y_m - \hat{y}_m)^2$$

$$= \frac{1}{2M} \sum_{m=1}^{M} (y_m - (\theta_0 + \theta_1 x_m))^2$$

Für das univariate Modell gibt es einen geschlossenen Ausdruck für den optimalen Modellparameter. Setzen Sie die partiellen Ableitungen der Kostenfunktion auf Null.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = 0$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = 0$$

was zu folgender Gleichung führt:

$$\hat{\theta}_0 = \mu_y - \theta_1 \mu_x$$

$$\hat{\theta}_1 = \frac{\sum_{m=1}^{M} (x_m - \mu_x)(y_m - \mu_y)}{\sum_{m=1}^{M} (x_m - \mu_x)^2} = \frac{\tilde{s}_{xy}}{\tilde{s}_x^2}$$

- $\tilde{s}_{xy}$ = Kovarianz von x und y
- $\tilde{s}_x^2$ = variance of x

$$\theta_0 = \bar{y} - m\bar{x}$$

MULTIVARIATE LINEAR REGRESSION

Die multivariate lineare Regression ist die Erweiterung der einfachen linearen Regression. Es werden N Merkmale $(x_1, x_2, \dots x_N)$ anstelle von nur einem verwendet.

$$\hat{y}_m = h(x_m;\ \theta) = \theta_0 x_{m0} + \theta_1 x_{m1} + \cdots + \theta_n x_{mN} = \theta^T X_m$$

Das multivariate lineare Regressionsmodell kann wie folgt formuliert werden:

$$y = X\theta + \epsilon$$

- y: Spaltenvektor mit allen Ausgabewerten
- θ: Modellparameter/ Koeffizienten
- ϵ: Fehler (Residuen) des Modells
- X: Matrix mit Eingabewerten
    - 1. Spalte 1en für y-Achsen Schnittpunkt

Bsp. $M = 3, N = 4$:

$$X = \begin{pmatrix} 1 & 29932.5 & 4.8 & 18.0 \\ 1 & 31007.8 & 1.0 & 16.4 \\ 1 & 32181.2 & 1.0 & 16.9 \end{pmatrix} \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix} y = \begin{pmatrix} 5.9 \\ 5.6 \\ 5.4 \end{pmatrix}$$

Um die optimalen Werte für θ zu finden, können sowohl die lineare Gleichung als auch der Gradient Descent verwendet werden.

NORMAL EQUATION

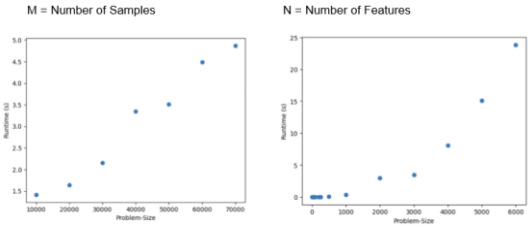
Die Normalgleichung ist die geschlossene Form der optimalen Parameter und führt zum optimalen Wert für Theta mit der kleinsten Fehlerquote.

$$\theta = (X^T X)^{-1} X^T y$$

NACHTEILE DER NORMAL EQUATION

- Arbeitsspeicher
    - Da alle Daten benötigt werden
- Rechenkomplexität (Inverse $(X^T X)^{-1}$)
    - Bis zu $O(n^3)$
    - < 20'000
- Numerische Instabilität
- Bei hinzufügen von neuen Daten müssen alle Daten wieder verwendet werden.

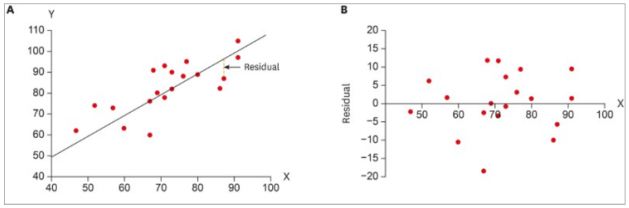
Rechenlaufzeit mit variierender Anzahl Samples (linear) vs. Anzahl Features (n^3).



RESIDUAL PLOTS

Das Residuen-Diagramm zeigt die Residuen für jedes Sample y_i auf der vertikalen Achse und die Eingabewerte auf der horizontalen Achse.

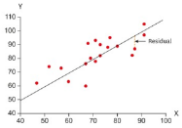
- Das Residuum beschreibt die Abweichung eines Samples y_m von der Regressionsgeraden, bzw. von der Vorhersage für den Wert x_m.
    - $\epsilon_m = y_m - \hat{y}_m$
- Sinnvoll für Modelle mit mehreren Variablen



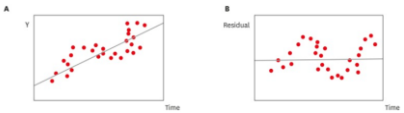
GRUNDLEGENDE ANNAHMEN

Die Anwendung von linearen Regressionsmodellen ist nur möglich, wenn folgende Annahmen für die Daten gültig sind.

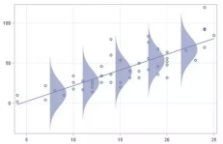
1. **Linearität**: Die Beziehung zwischen x und y ist linear.



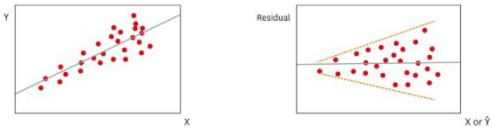
2. **Unabhängigkeit**: Die Residuen sind voneinander unabhängig.
    a) Sample Pearson Korrelationskoeffizient



3. **Normalverteilt**: Der erwartete Output ist normalverteilt.



4. **Homoskedastizität** (Gleichheit der Varianz): Die Varianz des Residuums ist für jeden Wert von X gleich gross.



EVALUIERUNG VON REGRESSIONS MODELLEN

Es gibt mehrere Metriken zur Bewertung der Qualität eines Regressionsmodells.

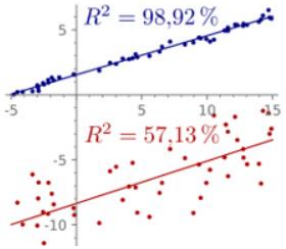
- Mean Absolute Error: $MAE = \frac{\sum_{i=1}^{I}|y_i - \hat{y}_i|}{I}$
- Mean Squared Error: $MSE = \frac{\sum_{i=1}^{I}(y_i - \hat{y}_i)^2}{I}$
- Root Mean Squared Error = $\sqrt{MSE}$

COEFFICIENT OF DETERMINATION

Der Koeffizient der Bestimmtheit, auch bekannt als R^2 ist ein Mass dafür, wie gut das Regressionsmodell die Variation der abhängigen Variablen erklärt. Er wird berechnet als das Verhältnis der durch das Modell erklärten Varianz zur gesamten Varianz der abhängigen Variablen.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \mu_i)^2}$$

- SS_{res}: Summe der quadrierten Residuen
- SS_{tot} : Summe des quadratischen Abstands zwischen jedem erwarteten Output y_i und dem Mittelwert

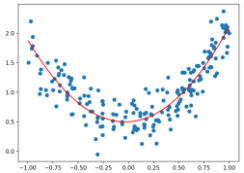


Wenn das Modell den Mittelwert vorhersagt, ist R^2 gleich Null.

POLYNOMIAL REGRESSION

Die polynomiale Regression wird zur Modellierung von Beziehungen bei nichtlinearen Daten verwendet. Dabei

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$



Um dies zu formalisieren, definieren wir zunächst künstliche Variablen $z_0 = 1,\ z_1 = x,\ z_2 = x^2$ und $z_3 = x^3$. So können wir die Hypothesenfunktion so formulieren: $h_0(x) = \theta_0 + \theta_1 z_1 + \theta_2 z_2 + \theta_3 z_3$ was uns wieder zu einem linearen Regressionsmodell führt.

VORGEHEN

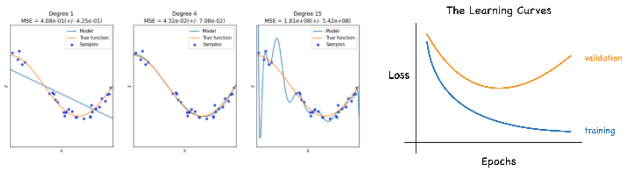
1. Grad des Polynoms ermitteln
2. Features transformieren
    a. bei einer unabhängigen Variable x und quadratischen Zusammenhang:
    b. $\theta_0 + \theta_1 x + \theta_2 x^2$
    c. $z_0 = 1,\ z_1 = x,\ z_2 = x^2$
    d. $\theta_0 z_0 + \theta_1 z_1 + \theta_2 z_2$

```
polynomial_features = PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(X_train)
```

3. Interaktionen zwischen Features berücksichtigen
    a. Bei mehreren unabh. Variablen können Interaktionsterme erstellt werden
    b. x = Länge, y = Breite → xy = Fläche
4. Lineares Modell trainieren
    a. Nachdem die Features transformiert wurden ($x \rightarrow z$) kann nun das «lineare Modell» $\theta_0 z_0 + \theta_1 z_1 + \theta_2 z_2$ trainiert werden.

OVER- AND UNDERFITTING

Wir wollen das perfekte Polynommodell (Leistung und Komplexität) finden, um Over- und Underfitting zu vermeiden. In der Mitte ist das perfekte Polynom-Modell für dieses Problem.



REGULARISIERUNG

Regularisierung wird verwendet, um Überanpassung zu verhindern, indem sie die Grösse der Modellparameter beschränkt. Sie fügt der Kostenfunktion einen Strafterm hinzu, der grosse Parameterwerte bestraft, und wird durch den Hyperparameter λ gesteuert.

$$J(\theta) = \frac{1}{2M}\left[\sum_{m=1}^{M}(y_m - \hat{y}_m)^2\right] + \left[+\frac{\lambda}{2M}\sum_{j=1}^{n}\theta_j^2\right]$$

Der Regulierungsparameter λ bestimmt das Gewicht von $\theta_{i>0}$.

- **Grosses λ → Kleines θ_i**

HYPERPARAMETER

- Grad des Polynoms
- Regularisierungsparameter λ
- Learning Rate α

GRID SEARCH

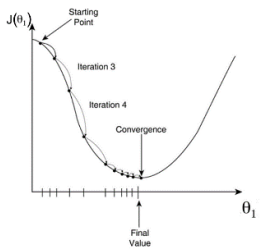
Ein simpler Ansatz die optimalen Hyperparameter zu finden ist Grid Search. Grad des Polynoms $\in \{3, 5, 8\}$, $\lambda \in \{10, 1, 0.1, 0.01\}$, $\alpha \in \{1, 0.3, 0.1, 0.03, 0.001\}$. In diesem Fall würden $3 * 4 * 5 = 60$ Modelle trainiert und validiert, um die optimalen Hyperparameter zu finden.

GRADIENT DESCENT

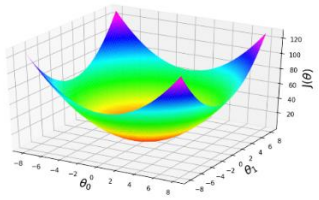
Aus den vorher genannten Gründen wie z.B. weil Matrixmultiplikationen und Invertierungen kubisch zur Grösse der Matrix ist, ist die Normalgleichung nur für Datensätze mit **weniger als 20'000 Samples** eine gute Wahl.

Daher gibt es einen anderen Ansatz, um die optimalen Parameter zu finden - den Gradient Descent Algorithmus. Er kann für die Optimierung vieler anderer Algorithmen als der linearen Regression verwendet werden.

Der Ansatz beginnt mit Zufallswerten für θ_0 und θ_1 für die Kostenfunktion $J(\theta)$ und nutzt die Eigenschaften des Gradienten, um den Fehler des Modells zu minimieren. Da der Gradient der Kostenfunktion die Richtung des steilsten Anstiegs der Kostenfunktion angibt, können wir die Kostenfunktion minimieren, indem wir den Gradienten der Kostenfunktion subtrahieren, bis die Kostenfunktion konvergiert.



Das Ziel ist es, ein Minimum der Kostenfunktion zu finden, wie der rote Bereich in der konvexen Funktion unten.



GD - ALGORITHMUS

Der generische Algorithmus für den Gradient Descent für n Parameter $\theta_0, \dots, \theta_n$ wie folgt:

1. $Initialize\ \theta_0, \dots, \theta_n$
2. $Repeat\ until\ convergence$:

$$Update\ \theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \qquad \forall j = 0, \dots, n$$

Bei einer univariaten linearen Regression sieht die Ableitung der Kostenfunktion wie folgt aus:

Für θ_0:

- $\frac{\partial}{\partial \theta_0} J(\theta) = \frac{\partial}{\partial \theta_0} \frac{1}{2M} \sum_{i=1}^{M} \left((\theta_1 x_i + \theta_0) - y_i\right)^2$
- $= \frac{1}{M} \sum_{i=1}^{M} \left((\theta_1 x_i + \theta_0) - y_i\right)$
- $= \frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - y_i)$

Für θ_1:

- $\frac{\partial}{\partial \theta_1} J = \frac{\partial}{\partial \theta_1} \frac{1}{2M} \sum_{i=1}^{M} \left((\theta_1 x_i + \theta_0) - y_i\right)^2$
- $= \frac{1}{M} \sum_{i=1}^{M} \left((\theta_1 x_i + \theta_0) - y_i\right) x_i$
- $= \frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - y_i) x_i$

Setzt man die beiden obigen Formeln ein, erhält man:

$$1.\ Initialize\ \theta_0, \dots, \theta_n$$

$$2.\ Repeat\ until\ convergence:$$

$$Update\ \theta_0 = \theta_0 - \alpha \frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - y_i)$$

$$Update\ \theta_1 = \theta_1 - \alpha \frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - y_i) x_i$$

LEARNING RATE α

Die Learning Rate ist ein Skalierungsfaktor, der angibt, wie gross die Schritte sind, die das Modell bei der Aktualisierung seiner Parameter in Richtung des Gradienten der Kostenfunktion macht.

- Lernrate α zu gross
  - Minimum wird übersprungen
  - Konvergiert nicht (Divergenz)
- Lernrate α zu klein
  - Sehr langsames Training/ Konvergenz
  - kann auf Plateau oder lokalem Minimum stecken bleiben.

LEARNING RATE OPTIMIERUNG

Indem wir die Lernrate α in jedem Schritt verringern, kann die Lernrate so optimiert werden, dass das Minimum so effizient wie möglich angenähert wird.

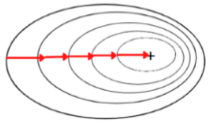
$$\frac{1}{1 + decay\_rate * epoch} \alpha_0, \qquad epoch = 1,2, \dots$$

- $decay\_rate$: Abklingrate
- $epoch$: ein Durchlauf durch gesamten Trainingssatz

VERSCHIEDENE ARTEN VON GD

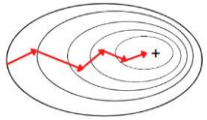
In der Optimierung, insbesondere beim Training von maschinellen Lernmodellen mittels Gradient Descent, unterscheidet man zwischen drei Hauptvarianten: **Batch Gradient Descent, Mini-Batch Gradient Descent** und **Stochastic Gradient Descent** (SGD).

BATCH GRADIENT DESCENT



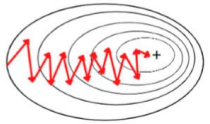
- Berechnet Gradienten der Kostenfunktion des gesamten Trainingsdatensatzes.
- Jede Iteration verwendet alle Samples, um die Parameter zu aktualisieren.
    - Iteration = Epoche
+ Batch Gradient Descent konvergiert stabil zum Minimum (langsam)
- Rechenintensiv, da alle Daten berücksichtigt werden.
- Benötigt viel Arbeitsspeicher

MINI-BATCH GRADIENT DESCENT



- Gradienten wird mit Teilmenge der Trainingsdaten berechnet.
- Typischerweise zwischen 50 & 250 Samples
+ Schnellere Konvergenz
+ Effizientere Arbeitsspeichernutzung
+ Balance zwischen Stabilität der Konvergenz und Recheneffizienz.

STOCHASTIC GRADIENT DESCENT – SGD



- Der Gradient wird auf Basis eines einzeln zufällig gewählten Trainingsbeispiels bei jedem Update berechnet.
- N Iterationen = Epoche
+ Kann helfen lokale Minima zu vermeiden
+ Besonders effektiv bei grossen Datensätzen
+ Schnell
- Kann durch Schwankungen das Minimum «übersehen» anstelle direkt zu konvergieren.

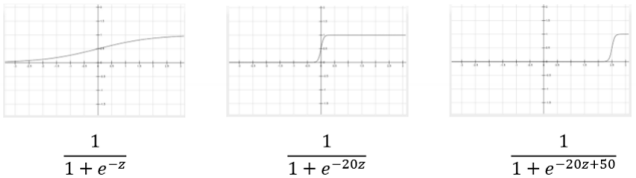
HYPERPARAMETER - GRADIENT DESCENT

1. Lernrate α
    a. Lernraten-Zeitplan (Learning Rate Schedule)
    b. Lernrate abklingen lassen.
2. Anzahl Epochen
    a. Wie häufig der Trainingsdatensatz durchlaufen wird
    b. Viele Epochen erhöhen Risiko zu Overfitting
3. Batch-Grösse
4. Early Stopping
    a. Stoppt, wenn der Lost des Validierungsset nicht mehr verbessert wird.
    b. Verhindert Overfitting

LOGISTIC REGRESSION

Die logistische Regression ist ein statistisches Modell, das verwendet wird, um die Wahrscheinlichkeit eines binären Ausgangs zu schätzen. Anstatt einen kontinuierlichen Ausgang wie bei der linearen Regression vorherzusagen, modelliert die logistische Regression die Wahrscheinlichkeit, dass die abhängige Variable den Wert 1 annimmt. Hierbei verwendet sie die logistische Funktion, auch Sigmoid-Funktion genannt, um die lineare Kombination der Eingabefeatures auf einen Wert zwischen 0 und 1 zu beschränken. Wenn eine Eingabe einen Wert über 0,5 erhält, wird sie als positiv eingestuft und umgekehrt.

$$g(z) = \frac{1}{1+e^{-z}}, \qquad 0 \le g(z) \le 1$$



EINDIMENSIONALE SIGMOID-FUNKTION

$$\hat{y} = h_\theta(x) = g(\theta_0 + \theta_1 x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$$

MULTIDIMENSIONALE SIGMOIDFUNKTION:

$$\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

$\theta^T x = \theta_o x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$ ist ein Skalarprodukt des Parametervektors θ und der Inputmatrix x.

LOSS FUNCTION OF LOGISTIC REGRESSION

Die Loss Function der Logistischen Regression heist Log Loss:

$$L(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$

Wir können das wie folgt formulieren:

$$J(\theta) = \frac{1}{M}\sum_{i=1}^{M} -y_i \log\big(h_\theta(x_i)\big) - (1-y_i)\log\big(1 - h_\theta(x_i)\big)$$

Immer wenn der Wert für y 1 ist, wird der linke Teil verwendet, und wenn y 0 ist, wird der rechte Teil verwendet.

$$\frac{\partial}{\partial\theta_j} J(\theta) =$$

$$-\frac{\partial}{\partial\theta_j}\left[\frac{1}{M}\sum_{m=1}^{M}\big(y_m * \log(\hat{y}_m) + (1-y_m)\log(1-\hat{y}_m)\big)\right]$$

Das führt zu:

1. $Initialize\ \theta_0, \dots, \theta_n$
2. $Repeat\ until\ convergence$:

- $Update\ \theta_0 = \theta_0 - \alpha\frac{1}{M}\sum_{i=1}^{M}(\hat{y}_m - y_m)$
- $Update\ \theta_1 = \theta_1 - \alpha\frac{1}{M}\sum_{i=1}^{M}(\hat{y}_m - y_m)x_m$

MODEL EVALUATION

Bei Klassifizierungsproblemen ist die gebräuchlichste Evaluierungskennzahl die Accuracy. Der Accuracy-Wert ist ein prozentualer Wert.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{number\ of\ right\ classifications}{all\ classifications}$$

HYPERPARAMETERS

- Learing rate α
- Anzahl Iterationen des Gradient Descent
- Der Regularisierungsfaktor

Der beste Ansatz, um die besten Werte für die Hyperparameter zu finden, besteht darin, den Datensatz in drei disjunkte (sich nicht überschneidende) Teile aufzuteilen - **Trainingsset**, **Validierungsset** und **Testset**.

- Trainingsset
  - Trainieren des Modells
- Validierungsset
  - Wenn das Modell trainiert wurde:
  - Validierung der Hyperparameter mit Validierungsset
- Testset
  - Wenn Modell trainiert und Hyperparameter festgelegt wurden:
  - Validierung des Modells durch Testset.

DATA AUGMENTATION

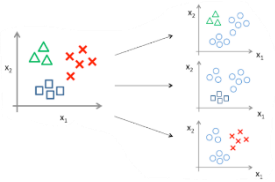
Vergrösserung des Datensatzes durch:

- Bilder rotieren/spiegeln
- Gaussian Noise/ Rauschen
- Farbveränderungen

MULTICLASS KLASSIFIKATION

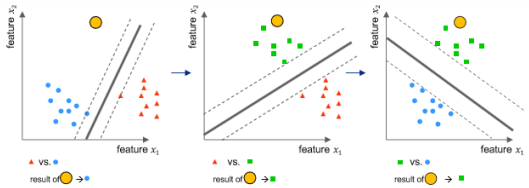
ONE-VS-REST CLASSIFICATION

Das Problem wird in 3 Probleme aufgeteilt und durch höchste die Wahrscheinlichkeit klassifiziert. Wenn $\hat{y}_1 = 0.8, \hat{y}_2 = 0.2, \hat{y}_3 = 0.3$, klassifizieren wir als $\hat{y}_1$.



ONE-VS-ONE

Es werden N Klassen erstellt und $(N-1)/2$ Modelle trainiert, welches jedes lernt nur diese Klassen zu unterscheiden.

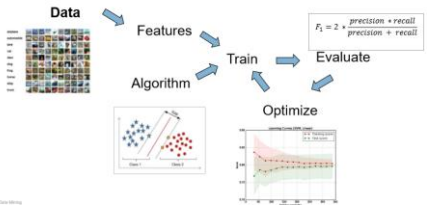


Die Klasse, die die meisten Stimmen von den verschiedenen Klassifikatoren erhält, wird als die endgültige Klassifikation für das Beispiel gewählt.

SOFTMAX-CLASSIFIER

Softmax-Regression, ist eine Verallgemeinerung der logistischen Regression auf Mehrklassenprobleme, d.h., sie kann verwendet werden, wenn es mehr als zwei Klassen gibt. Anstatt wie bei der binären logistischen Regression die Wahrscheinlichkeit für zwei Klassen zu modellieren, schätzt die Softmax-Regression die Wahrscheinlichkeiten für jede Klasse **innerhalb eines einzigen Modells**.

SUPERVISED LEARNING



FEATURE VEKTOREN & DATENREPRÄSENTATION

- Feature-Vektoren repräsentieren die relevanten Eigenschaften der Eingabedaten.
- Sie sind in der Regel n-dimensionale Vektoren von Zahlen.

BEISPIELE FÜR FEATURES

- **Textdaten**: Nutzung von Bag of Words (BoW), bei dem die Häufigkeit spezifischer Wortkombinationen in einem Text erfasst wird.
- **Audiodaten**: Einsatz von Mel Frequency Cepstral Coefficients (MFCCs), die wichtigen Charakteristika von Audiosignalen abbilden.
- **Bilddaten**: Verwendung von Gabor-Filtern, die zur Erfassung von Texturen und Kanten in Bildern dienen.

EVALUIERUNGSMETRIKEN KLASSIFIKATION

τ: Threshold bei dem ein Output als positiv oder negativ eingeschätzt wird.

- Accuracy
  - Gut wenn FP und FN gleich schlimm.
  - $\frac{TP+TN}{TP+TN+FP+FN}$
  - $\frac{number\ of\ right\ classifications}{all\ classifications}$
  - Error: 1 – Accuracy
  - Vorsicht bei unausgeglichenen Datensätzen (90%=1, 10%=0 → Modell sagt immer 1 → 90% Accuracy)
- Precision
  - Gut wenn FP schlimmer (Spamfilter)
  - Bestraft FP
  - Ignoriert FN
  - $\frac{TP}{TP+FP}$
- Recall
  - Gut wenn FN schlimmer (Diagnostik)
  - Bestraft FN
  - Ignoriert FP
  - $\frac{TP}{TP+FN}$
- F1-Score
  - Ausbalanciert zwischen Precision und Recall
  - $2 * \frac{precision*recall}{precision+recall}$
  - Schwer zu interpretieren
- Allgemeiner F1-Score
  - Wenn Recall wichtiger ist als Precision
  - $(1-\beta^2) * \frac{precision*recall}{(\beta^2*precision)+recall}$
  - Durch das β kann Gewichtung von Precision gesteuert werden.

MULTIKLASSEN CONFUSION MATRIX

Eine Confusion Matrix mit mehr als 2 Klassen, muss immer die jeweilige Klasse gegeben die anderen zwei berechnet werden (one-vs-all).

Predicted label

True label: c	Healthy	Cold	Flue	Support of c
Healthy	60	8	2	70
Cold	4	12	4	20
Flue	0	2	8	10
				100

True Class	Predicted Class: Yes	Predicted Class: No
Yes	TP	FN
No	FP	TN

- Healthy vs. Not Healthy
- TP = 60, TN=12+4+2+8, FP = 4, FN=8+2

EVALUIERUNGSMETRIKEN REGRESSION

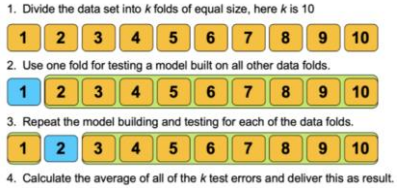
Es gibt mehrere Metriken zur Bewertung der Qualität eines Regressionsmodells.

- Mean Absolute Error: $MAE = \frac{\sum_{i=1}^{I}|y_i-\hat{y}_i|}{I}$
- Mean Squared Error: $MSE = \frac{\sum_{i=1}^{I}(y_i-\hat{y}_i)^2}{I}$
- Root Mean Squared Error = $\sqrt{MSE}$

DATA PARTITIONING UND MODELLVALIDIERUNG

K-FOLD CROSSVALIDATION

- Daten in **k gleich grosse Teile** (Folds) aufgeteilen
- Modell wird **k-mal trainiert** und für jedes Training wird ein Fold ausgelassen und als Testset verwendet.
- **Mittelwert** → Cross Validation Score
- Ganzes Datenset ist nutzbar (kleine Datensets)
- Braucht mehrere Durchläufe für valide Ergebnisse (Varianz der Daten,…)



LEAVE-ONE-OUT CROSS-VALIDATION (LOOCV)

- k = Gesamtzahl der Datenpunkte
- Bei jedem Durchlauf wird **ein einzelner Datenpunkt als Testset verwendet** und alle anderen Datenpunkte zum Trainieren.
- Dieser Ansatz ist rechenintensiv, bietet aber eine maximale Nutzung der verfügbaren Daten.
- Besonders nützlich bei sehr kleinen Datensätzen.

DATA PARTITIONIERUNG

Das Datenset wird in drei Sets aufgeteilt:

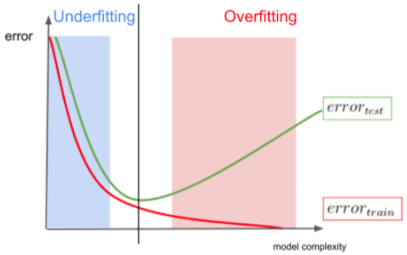
- Trainingsset
  - Trainieren des Modells
- Validierungsset
  - Wenn das Modell trainiert wurde:
  - Validierung der Hyperparameter mit Validierungsset
- Testset
  - Wenn Modell trainiert und Hyperparameter festgelegt wurden:
  - Validierung des Modells durch Testset.

Eine Alternative eines fixen Splits wäre, eine Double Cross Validation die einen Nested Loop verwendet:

- Äussere Schleife
  - Daten warden in Folds unterteilt
  - Ein Teil Testset
  - Rest für die innere Schleife
- Innere Schleife
  - Wird wieder in Folds aufgeteilt
  - Ein Fold wird als Validation Set verwendet für Hyperparameter
  - Der Rest zum trainieren des Modells

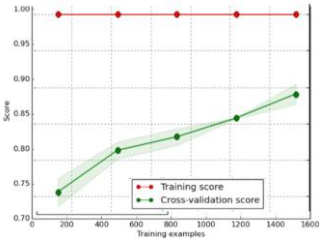
OVER- AND UNDERFITTING

- **Underfitting**
  - Zu stark generalisiert
  - Hoher Bias → erkennt systematisch gewisse Aspekte der Daten nicht
  - Schlechten Score in Trainingsdaten und Testdaten
- **Overfitting**
  - Zu sehr auf Trainingsdaten angepasst
  - Hohe Varianz → Überanpassung
  - Gut auf Trainingsdaten, schlecht auf neuen Daten (Testdaten)
- **Bias-Varianz Trade-Off**
  - Gleichgewicht zwischen Under- und Overfitting
  - Genug komplexität um Muster zu erkennen, gleichzeitig genug Generalisierung um auf neue Daten angewendet zu werden.
  - Bevor Trainings-Error und Test-Error divergieren.

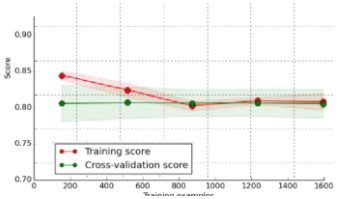


EVALUATION VON LERNKURVEN

- hoher Trainingsscore (Idealerweise)
  - (steigend mit mehr Trainingssamplen)
  - Oder konstant hoch
- hoher Testscore
  - gut generalisert
- grosser «Gap» zwischen Test- und Trainingsscore
  - Overfitting → mehr Generalisierung



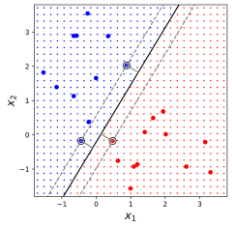
- Verhalten über Zeit
  - Testscore sollte sich verbessern
  - Andernfalls Probleme beim Lernprozess. (falsches Modell, fehlerhafte Daten, Overfitting)



SUPPORT VECTOR MACHINES

Support Vector Machines (SVMs) sind Supervised Learning Modelle, die für Klassifikations- und Regressionsanalysen verwendet werden. Die Idee hinter SVMs ist es, die beste Entscheidungsgrenze, genannt Hyperplane, zu finden, die zwei Klassen von Datenpunkten trennt. Dieser Grenzbereich wird so gewählt, dass er den grösstmöglichen Abstand (Margin) zwischen den Datenpunkten beider Klassen hat.

HYPERPLANE



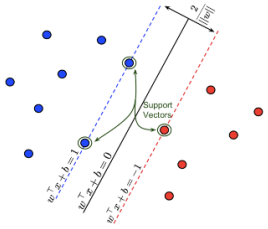
In einem n-dimensionalen Raum ist eine Hyperplane eine flache affine Untermenge der Dimension $N-1$, die den Raum in zwei Teile teilt. Die mathematische Definition der Hyperplane ist:

$$\mathbf{w}^T\mathbf{x} + b$$

- $\mathbf{x} = (x_1 \;\; x_2 \;\; \dots \;\; x_N)$
  - Ein Inputvektor (einzelner Datenpunkt) aus Datenraum
- $\mathbf{w} = (w_1 \;\; w_2 \;\; \dots \;\; w_n)$
  - Gewichtungsvektor (orthogonal zur Hyperplane)
- b = Bias-Term gibt Versatz zum Ursprung an

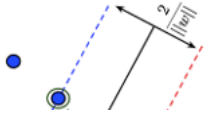
SUPPORT VECTORS

Das sind die Datenpunkte, die am nächsten an der Hyperplane liegen und den Abstand zur Entscheidungsgrenze definieren.



MARGIN

Der Margin ist der Abstand zwischen der Hyperplane und den Support Vektoren. SVMs maximieren diesen Margin, um die generalisierende Fähigkeit des Klassifikators zu verbessern.



TRAININGSDATEN

Bei der binären Klassifikation gibt es M Trainingsdatenpunkte im N-dimensionalen Raum:

$$\begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_N^{(1)} \end{pmatrix}, \dots, \begin{pmatrix} x_1^{(M)} \\ \vdots \\ x_N^{(M)} \end{pmatrix}$$

- Mit Klassen-Labels $y^{(1)}, \dots, y^{(M)} \in \{-1, +1\}$.

MAXIMAL MARGIN CLASSIFIER

Ein Maximal Margin Classifier zielt darauf ab, die Hyperplane so zu positionieren, dass die Margin maximiert wird, was die grösstmögliche Trennung der Klassen ermöglicht.

TRAINING (KONSTRUKTION) DER HYPERPLANE

1. $\mathbf{w}$ und b sollen so optimiert werden, dass Margin maximal ist.
2. Maximieren der Distanz $r^{(m)}$ zu den am naheliegendsten Trainingsdatenpunkten.
   a. $r^{(m)} = y^{(m)}\left(\frac{1}{\|\mathbf{w}\|}\left(\mathbf{w}^T\mathbf{x}^{(m)} + b\right)\right)$
   b. $\max_{b,\boldsymbol{w}}\left\{\frac{1}{\|\mathbf{w}\|}\min_{m=1}^{M}\left[y^{(m)}(b + \mathbf{w}^T x^{(m)}\right]\right\}$
3. Mit dem Skalierungsparameter κ wird sichergestellt, dass:
   a. $y^{(m)}\left(b + \mathbf{w}^T x^{(m)}\right) = 1$
4. Daraus folgt:
   a. $\min_{m=1}^{M}\left[y^{(m)}(b + \mathbf{w}^T x^{(m)}\right] = 1$
   b. $\Rightarrow \max_{b,\mathbf{w}}\left\{\frac{1}{\|\mathbf{w}\|} * 1\right\}$
   c. $\max_{b,\mathbf{w}} \frac{1}{\|\mathbf{w}\|} \Leftrightarrow \min_{b,\mathbf{w}} \|\mathbf{w}\|^2$
5. Das Optimierungsproblem von Max Margin Classifier ist also:
   a. $\min_{b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2$
   b. Unter der Nebenbedingung
   c. $y^{(m)}\left(b + w^T x^{(m)}\right) \geq 1, \; \forall\, m$
   d. Optimierungsproblem mit **Lagrange-Ansatz** lösen.

VORHERSAGE

Sind die Parameter $b, \mathbf{w}$ für die Hyperplane gefunden, kann eine Vorhersage bzw. Klassifizierung durchgeführt werden:

$$f\left(\mathbf{x}, \hat{b}, \hat{\mathbf{w}}\right) = \hat{b} + \hat{\mathbf{w}}^T\mathbf{x}$$

- $\mathbf{w}^T\mathbf{x} + b > 0$: Datenpunkt liegt **über** Hyperplane
- $\mathbf{w}^T\mathbf{x} + b < 0$: Datenpunkt **unter** Hyperplane
- $\mathbf{w}^T\mathbf{x} + b = 0$: Datenpunkt liegt **auf** Hyperplane

SOFT MARGIN CLASSIFIER

Da es in den meisten Fällen keine gerade Linie gibt, die zwei verschiedene Klassen in einem Datensatz trennt, brauchen wir einen neuen Ansatz, um Datenpunkte der anderen Klasse auf der anderen Seite zuzulassen.

- Soft Margin lässt Anzahl von «falschen» Klassifikationen zu.

Aus diesem Grund warden die **Slack Variablen** $\epsilon = (\epsilon_1\ \epsilon_2\ \dots\ \epsilon_M)^T, \epsilon_m \geq 0$ dem Support Vector Klassifikator hinzugefügt, der definiert, wie weit die Trainingsdatenpunkte innerhalb der falschen Seite liegen dürfen.

$$\min_{b,w,\epsilon} \frac{1}{2}\|w\|^2 + C\sum_{m=1}^{M} \epsilon_m$$

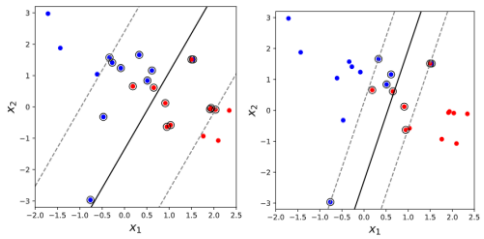
Mit der Bedingung:

$$\mathbf{y}^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right) \geq 1 - \epsilon_m, \qquad \forall m$$

HYPERPARAMETER C

Durch den **nicht-negativen** Hyperparameter C, kann gesteuert werden, wie viele Punkte die Randbedingung verletzen dürfen. (Cross Validation)

- kleines C → lässt viele Verstösse zu (links)
  - beugt Overfitting vor
- grosses C → wenig Verstösse (rechts)

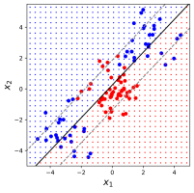


In der Praxis wird der Hyperparameter C über eine **Kreuzvalidierung** ausgewählt, die einen **Kompromiss** zwischen:

- Maximierung der Margin (kleines C)
  - Bessere Generalisierung
- Strafe für Fehlklassifikationen (grosses C)
  - Präzision des Modells

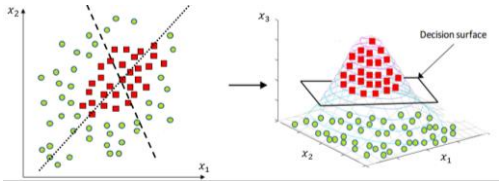
KERNEL TRICK

DEFINITION



Viele Klassifizierungsprobleme sind nicht linear trennbar. Ein Support-Vector-Klassifikator oder ein beliebiger linearer Klassifikator wird hier schlecht abschneiden. Das Problem kann durch die **Erweiterung des Feature Space** unter Verwendung quadratischer, kubischer oder polynomialer Funktionen höherer Ordnung für die Inputwerte (x_1, x_2) geschehen. Dabei kommen sogenannte Mapping Funktionen zum Einsatz, die den Datensatz in höhere Dimensionen befördern.

$$\varphi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1 \\ x_2 \\ \sqrt{x_1^2 + x_2^2} \end{pmatrix}$$



VERSCHIEDENE KERNELS

Statt den Merkmalsraum explizit um alle polynomiellen Kombinationen der Features zu erweitern (was zu einem quadratisch, kubisch oder höheren Anstieg der Feature-Anzahl führen würde), verwendet der Kernel-Trick eine Funktion, die das Skalarprodukt im höherdimensionalen Raum berechnet, ohne die Transformation durchführen zu müssen. (N bleibt konstant→ **effizient**)

POLYNOMIELLE KERNEL

$$\mathcal{K}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \left(\gamma \mathbf{x}^{(i)T}\mathbf{x}^{(j)} + b\right)^d$$

- $\mathcal{K}$: ist die Kernelfunktion, die das Skalarprodukt im höher dimensionalen Raum berechnet.
- $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$: zwei Vektoren aus dem Feature Space
- γ: Skalierungsfaktor
- b konstanter Term
- d Grad des Polynoms

Der Parameter d bestimmt den Grad des Polynoms, mit welchem die nicht separierbaren Daten in eine separierbare Dimension gehoben werden sollen.

$d = 1$ (standard SVM)

- Berechnet Beziehung zwischen jedem Datenpunkt in 1-Dimension
-
- $d = 2$
  - Fügt zusätzliche Dimension hinzu
  - Berechnet quadratische Beziehung zwischen Datenpunkten
- $d = \cdots$

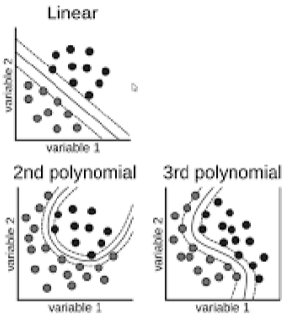
Durch die Validierung des Datensets wird der optimale Wert für d gewählt.

In diesem Fall braucht es zwei Gewichtungsvektoren $\mathbf{w}_1 = (w_{11} \ \ldots w_{N1})^T$ and $\mathbf{w}_2 = (w_{12} \ \ldots w_{N2})^T$

$$\min_{b,w_1,w_2,\epsilon} \frac{1}{2}\|w\|^2 + C\sum_{m=1}^{M} \epsilon_m$$

Mit der Bedingung:

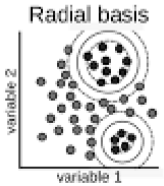
$$y^{(m)}\left(b + \mathbf{w}_1^T x^{(m)} + \mathbf{w}_2^T x^{(m)}\right) \geq 1 - \epsilon_m, \qquad \forall m$$



RBF KERNEL

$$\mathcal{K}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \exp(-\gamma\left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|^2)$$

- $\mathcal{K}$: ist die Kernelfunktion, die das Skalarprodukt im höher dimensionalen Raum berechnet.
- $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$: zwei Vektoren aus dem Feature Space
- γ: positiver Skalierungsfaktor der die **Breite** des RBF-Kernel steuert.

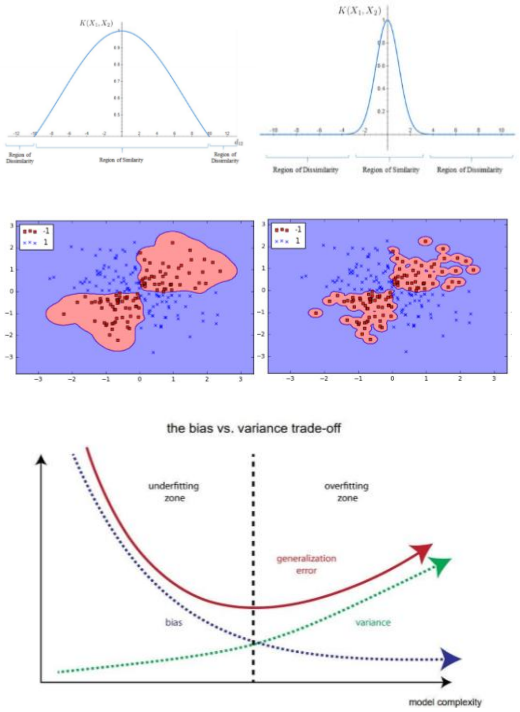


HYPERPARAMETER γ

γ ist der Skalierungsfaktor der die **Breite** des RBF-Kernel steuert.

Optimale Werte für die Hyperparameter C, γ können durch **Grid Search** gefunden werden, indem die **Cross Validation-Scores** verglichen werden. (C, γ müssen zusammen verwendet werden)

- kleines γ → grosses σ^2 → breite Region die gleich klassifiziert wird (Risiko für Underfitting)
- grosses γ → kleines σ^2 → kleine Region die mit gleicher Klassifizierung (Risiko für Overfitting



SVM WITH MORE THAN TWO CLASSES

- One-vs-Rest
- One-vs-One

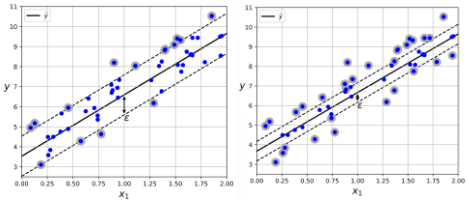
SVM FOR REGRESSION

Anstatt zu versuchen, die grösstmögliche Tube «Strasse» zwischen zwei Klassen zu passen, zielt die SVR darauf ab, so viele Datenpunkte wie möglich innerhalb dieser Strasse zu haben. Für nichtlineare Regressionsaufgaben können kernellisierte SVM-Modelle eingesetzt werden, um nichtlineare Beziehungen zu modellieren.

HYPERPARAMETER ϵ

Die Breite der Tube wird reguliert durch den Hyperparameter ϵ.

- Grosses ϵ → breiterer Tube



(links $\epsilon = 1$, rechts $\epsilon = \frac{1}{2}$)

HYPERPARAMETER C

Der Hyperparameter C ähnelt dem der SVM-Klassifikation und bestimmt das Ausmaß der Strafe für Abweichungen außerhalb der Epsilon-Tube.

- Grosses C → lässt weniger Fehler zu
  - empfindlich für Ausreisser
- Kleines C → lässt mehr Fehler zu
  - Weniger empfindlich für Ausreisser

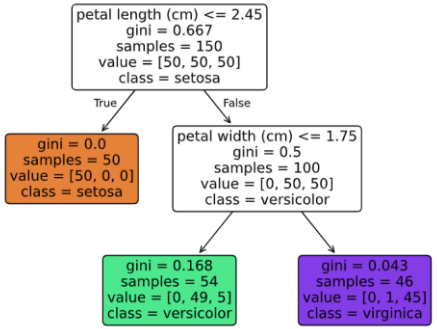
DECISION TREES

Decision Trees können sowohl für **Klassifikations-** als auch für **Regressionsaufgaben** verwendet werden können.

- Einteilung in Klassen aufgrund verschiedener Attribute

Die wichtigsten Aspekte eines Entscheidungsbaums sind:

- **Wurzelknoten**: Der Knoten, repräsentiert das gesamte Trainingsset
- **Entscheidungsknoten**: Knoten, die auf der Grundlage bestimmter Bedingungen oder Kriterien weitere *Aufteilungen* vornehmen.
- **Blattknoten**: Die *endgültigen Ausgänge* oder Entscheidungen, die nach Durchlaufen aller Entscheidungsknoten getroffen werden.



TRAINING

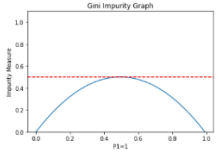
- **Auswahl von Attributen**, die die Daten am besten nach bestimmten Kriterien wie der Informationsgewinn (bei Klassifikation) oder der Varianzreduktion (bei Regression) teilen.
- **Definition von Entscheidungen**, die die Daten aufgrund ihrer Merkmale unterteilen.

GINI-IMPURITY

Um die Reinheit oder Unreinheit (Impurity) eines Datensatzes zu berechnen, verwenden wir die Gini-Impurity $G(Q_i)$. Ziel ist es, die reinste Teilmenge des Wurzelknotens $Q_{i=0}$.

$$G(Q_i) = 1 - \sum_{k=1}^{K} p_{i,k}^2$$

- p : Wahrscheinlichkeit, dass Klasse in Set vorkommt
- $G(Q_i) = 0$: höchste Reinheit
- $G(Q_i) = 1$ tiefste Reinheit
  - Bei 2 Klassen: $1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right) = 0.5$
  - Bei 3 Klassen: $1 - \left(\sum_{i=1}^{3} \left(\frac{1}{3}\right)^2\right) = 0.67$

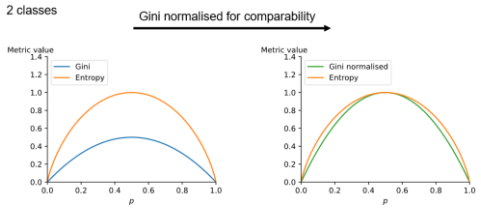


Bsp: Setosa = $\left(\frac{0}{54}\right)$, Versicolor = $\left(\frac{49}{54}\right)$, Virginica = $\left(\frac{5}{54}\right)$

$$G(Q_i) = 1 - \left(\left(\frac{0}{54}\right)^2 + \left(\frac{49}{54}\right)^2 + \left(\frac{5}{54}\right)^2\right) = 0.168$$

ENTROPY

Entropy ist ein alternatives Reiheitsmass, das sich ähnlich wie Gini-Impurity verhält. $H(Q) = -\sum_{i=1}^{n} p_i \log_2(p_i)$



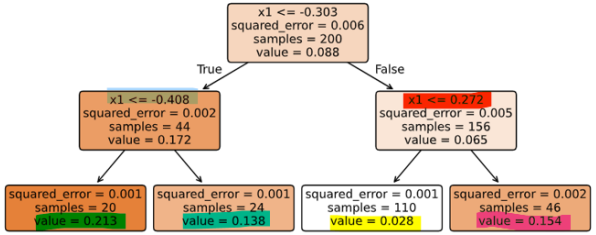
TRAINING MIT CART-ALGORITHMUS

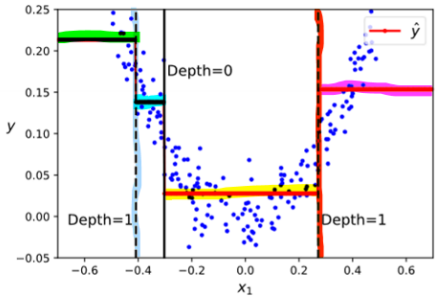
- CART-Algorithmus produziert binäre Bäume
  - Jeder Entscheidungsknoten genau 2 Kinder
- Datensatz Q
  - Die Daten am Knoten i werden mit Q_i bezeichnet
  - Bestehend aus M Samples
  - Der Algorithmus teilt den Datensatz auf Basis eines Features und eines Schwellenwerts $\theta = (n, t_i)$.
  - n = Feature des Datensatzes
    - Attribut
  - t_i = Schwellenwert
    - z.B. ≤ 0.5
- Aufteilung der Daten
  - Jeder Knoten eines Baumes entscheidet, wie die Daten aufgeteilt werden sollen.
  - Sucht nach Paar (n, t_i) mit grösster Reinheit
- Kostenfunktion $J(Q_i, \theta) =$
  - $\frac{M_i^{left}}{M_i} G\left(Q_i^{left}(\theta)\right) + \frac{M_i^{right}}{M_i} G\left(Q_i^{right}(\theta)\right)$
  - Beispiel erster Split Iris-Daten
  - $n =$ Petal Length, $t_i \leq 2.45$
  - $\frac{50}{150} * \left(1 - \left(\left(\frac{50}{50}\right)^2 + \left(\frac{0}{50}\right)^2 + \left(\frac{0}{50}\right)^2\right)\right) + \frac{100}{150} * \left(1 - \left(\left(\frac{50}{100}\right)^2 + \left(\frac{50}{100}\right)^2 + \left(\frac{0}{100}\right)^2\right)\right) = \frac{1}{3}$
  - Berechnet die Kosten, indem die **gewichtete Impurity** der beiden Sets summiert wird.
  - Der Algorithmus sucht $\min_{Q_i} J(Q_i, \theta)$
- Wurde das optimale Paar θ gefunden, wird dasselbe **rekursiv** für die Subsets durchgeführt, bis die gewünschte Tiefe erreicht ist.

REGRESSION TREES

Regression Trees sind so strukturiert, dass sie einen kontinuierlichen Wert vorhersagen, der auf den **durchschnittlichen Werten** der Trainingsbeispiele **in jedem Blattknoten** basiert.

- Der Wert im Wurzelknoten ist das Mittel des gesamten Datensatzes.





TRAINING VON REGRESSION TREES

- Gleich wie Decision Trees
- Statt Gini-Impurity → MSE

$$MSE(Q_i) = \frac{1}{M_i} \sum_{y \in Q_i} (y - \bar{y}_i)^2$$

Mit Mittelwert als Vorhersage $\bar{y}$.

$$\bar{y} = \frac{1}{M_i} \sum_{y \in Q_i} y$$

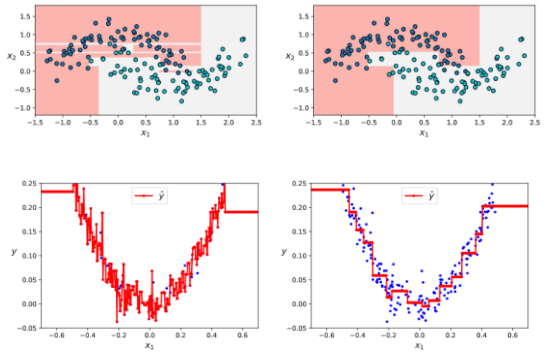
HYPERPARAMETER DECISION & REGRESSION

- **max_depth**: Begrenzt die Tiefe des Baumes.
- **min_samples_split**: Definiert die minimale Anzahl von Samples, die erforderlich sind, um einen Knoten zu teilen.
- **min_samples_leaf**: Gibt die minimale Anzahl von Samples an, die ein Blattknoten haben muss.
- **min_weight_fraction_leaf**: Ähnlich wie min_samples_leaf, aber als Bruchteil der gewichteten Instanzen ausgedrückt.
- **max_leaf_nodes**: Begrenzt die maximale Anzahl von Blattknoten.
- **max_features**: Bestimmt die Anzahl der Merkmale, die bei der Suche nach der besten Aufteilung berücksichtigt werden.

REGULARIZATION

- Entscheidungsbäume sind nichtparametrische Modelle, was bedeutet, dass sie die Anzahl ihrer Parameter während des Trainings anpassen und eng an die Daten anlehnen können, was zu Overfitting führen kann.
- Um **Overfitting zu vermeiden**, muss die **Komplexität** der Entscheidungsbäume durch Regularisierung **eingeschränkt werden**, ähnlich wie bei polynomialer Regression oder SVMs.
- Die Regularisierung eines Baumes kann durch die oben genannten Hyperparameter gesteuert werden.

Erhöhen der **min_*** Hyperparameter oder Reduzieren von **max_*** Hyperparameter wird das Modell regulieren.



(links: unreguliert (zu komplex), rechts reguliert)

PRE- OR POST-PRUNING

Vorbeugen von zu hoher Komplexität des Baumes.

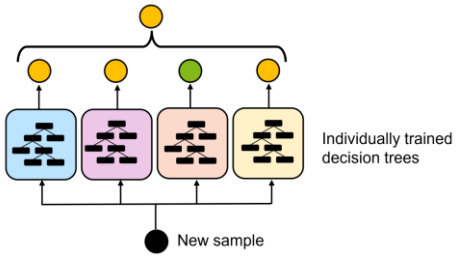
Zu Deutsch: «Bescheidung»

- **Pre-Pruning** (Vorab-Beschneidung): Beim Pre-Pruning wird der Baum während des Trainingsprozesses vorzeitig beschnitten. Das bedeutet, dass der Trainingsprozess gestoppt wird, bevor der Baum zu komplex wird.
  - **max_depth**
  - **min_samples_split**
  - **min_samples_leaf**
- **Post-Pruning** (Nachträgliches Beschneiden): Post-Pruning findet nach dem vollständigen Training des Baumes statt. Dabei werden Knoten oder ganze Zweige des Baumes entfernt, die wenig zur Vorhersagegenauigkeit beitragen.

RANDOM FORESTS

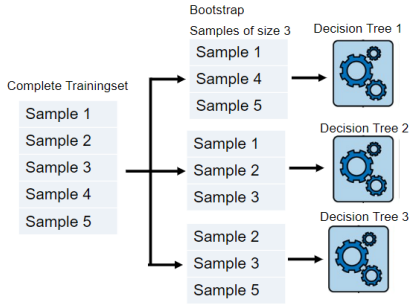
Random Forests sind ein Ensemble-Lernverfahren, das für Klassifikations- und Regressionsaufgaben eingesetzt werden kann und auf mehreren Entscheidungsbäumen basiert.

- Individuell trainierte Decision/ Regression Trees



BAGGING

- **Jeder Baum** wird auf einer zufälligen Auswahl von Daten trainiert (Bootstrap-Sample, mit Zurücklegen)



OUT-OF-BAG ERROR

Die Samples, die durch Bootstrapping nicht im Training des jeweiligen Baumes berücksichtigt wurden, werden Out-of-Bag genannt.

Der **Out-of-Bag-Error** wird pro Decision Tree berechnet, indem diese nicht verwendeten Samples durch den Random Forest laufen gelassen werden.

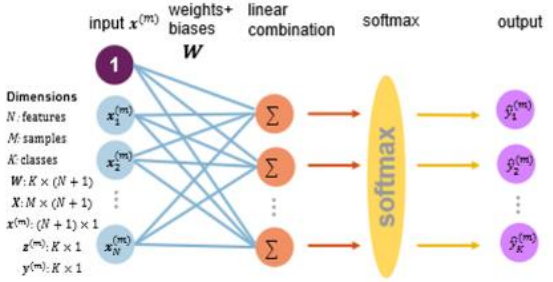
- niedriger OOB-Fehler → gut generalisiert

HYPERPARAMETER

- Anzahl Trees
  - 100-500
- Sample Grösse
  - Anteil von Samples verwendet in einem Decision Tree (20%-100%)
- Anzahl Features pro Knoten
  - Wie viele Features pro Knoten berücksichtigt werden. Normal ($\sqrt{n}$)
- Maximale Tiefe
  - Unendlich möglich
  - Meistens 3-7

SOFTMAX REGRESSION

Softmax Regression ist eine Erweiterung der logistischen Regression und wird für **lineare** Multiklassenklassifikation verwendet.



KONZEPT

- Klassifizierung von **> 2 Kategorien**
- Für jede Klasse wird eine Wahrscheinlichkeit berechnet ob gegebener Datenpunkt zu Klasse gehört.

MODELL

- **Eingabe**: $\mathbf{x} \in \mathbb{R}^n$
- **Ausgabe**: $\hat{\mathbf{y}} \in \mathbb{R}^K$: Vektor mit Wahrscheinlichkeit für jede Klasse k unter K möglichen Klassen
- Lineare Kombination: $\mathbf{z}_k = \mathbf{w}_k^T \mathbf{x} + b_k$
  - $\mathbf{z}_k$: Ausgabevektor der Linearkombination
  - $\mathbf{w}_k$: Vektor mit Gewichtungen
  - b_k: Bias-Term
- Softmax Funktion (**nicht linear**)
  - $\hat{y}_k = softmax(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}$
  - Skaliert alle Outputs [0, 1]

SOFTMAX KOSTENFUNKTION

Ziel: Minimierung des Fehlers bei Klassifikation

Cross-Entropy-Loss:

$$Loss(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

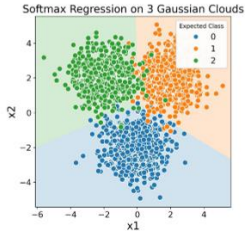
- $\mathbf{y}$**:** One-Hot-Vektor mit tatsächlicher Klasse
- $\hat{\mathbf{y}}$**:** Vorhersagevektor mit Wahrscheinlichkeiten

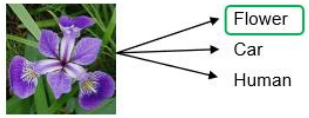
Über alle Trainingsdatenpunkte:

$$J(W) = -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} y_k^{(m)} \log\left(\hat{y}^{(m)}{}_k\right)$$

- M: Anzahl Trainingsdatenpunkte

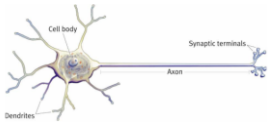
Durch Gradient Descent führt zur Minimierung der Kostenfunktion.





FEED-FORWARD NEURAL NETWORKS

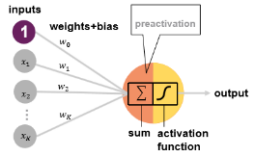
Die Architektur künstlicher Neuronale Netze lehnt sich an biologischen Neuronalen Netzen an. Durch die Verkettung von verschiedenen künstlichen «Neuronen», kann praktisch jede Funktion approximiert werden (Universality Theorem). (Feed-Forward → nur in eine Richtung)



NEURON

Ein Neuron in einem neuronalen Netzwerk kann als eine grundlegende Verarbeitungseinheit betrachtet werden.

- Mehrere Eingänge
- Einen Ausgang



FUNKTIONSWEISE

- Lineare Kombination
  - Jedes Eingabesignal wird mit einem Gewicht multipliziert und zusammen mit einem Bias-Term aufsummiert
  - $z = (\sum_{i=1}^{n} w_i x_i) + b$
  - w_i: Gewichte
  - x_i: Eingabewerte
  - b: Bias-Term

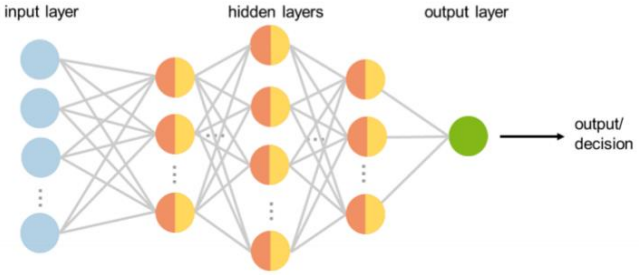
AKTIVIERUNGSFUNKTION

Die resultierende Linearkombination z wird einer Aktivierungsfunktion übergeben $act(z)$, um den Ausgang des Neurons zu erzeugen.

Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	0	$\{0,1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \triangleq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$	$(0,1)$	C^{∞}
Hyperbolic tangent (tanh)		$\tanh(x) \triangleq \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1-g(x)^2$	$(-1,1)$	C^{∞}
Soboleva modified hyperbolic tangent (smht)		$\operatorname{smht}(x) \triangleq \frac{e^{ax}-e^{-bx}}{e^{cx}+e^{-dx}}$		$(-1,1)$	C^{∞}
Rectified linear unit (ReLU)[20]		$(x)^+ \triangleq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0,x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0,\infty)$	C^{0}

AUFBAU EINES NEURONALEN NETZES

Ein neuronales Netz besteht immer aus einem «Input-Layer», einem oder mehreren «Hidden-Layers» und einem «Output-Layer». Diese Layer werden mit n-n Verbindungen verbunden.



INPUT LAYER

Die erste Schicht, nimmt die Eingabedaten in Form eines Vektors auf und übergibt sie den Hidden Layers. Im Input Layer werden keine Rechenoperationen durchgeführt.

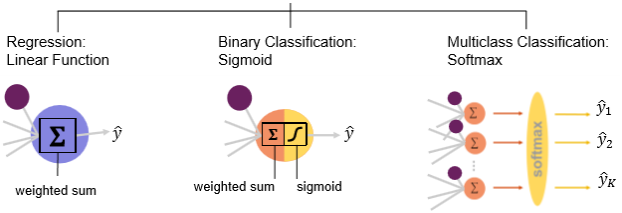
HIDDEN LAYERS

Die «Hidden Layers» bestehen aus einer oder mehreren Schichten, die zwischen dem Input Layer und dem Output Layer liegen.

- Jede Schicht beinhaltet n Neuronen
- Jedes Neuron ist normalerweise mit jedem Neuron im vorherigen Layer verbunden.
- Die Anzahl Schichten und Neuronen in jeder Schicht bestimmt die Tiefe und Kapazität des Netzwerks.
- In jedem Neuron wird eine Linearkombination der Eingabe berechnet und durch die Aktivierungsfunktion geleitet.

OUTPUT LAYER

Der Output Layer variiert je nach Problem, für das das Netzwerk eingesetzt wird.



- Regression (lineare Activation-Funktion)
  - $\hat{y} \in \mathbb{R}$ (stetiger Wert)
- Binäre Klassifikation
  - $\hat{y} \in [0,1]$
  - Mit Entscheidungsgrenze z.B. 0.5
- Multiklassen Klassifikation (Softmax)
  - $\hat{\mathbf{y}} \in \mathbb{R}^K$
  - Vektor mit Wahrscheinlichkeiten
  - $\sum_{i=1}^{K} \hat{y}_i = 1$
  - Höchste Wahrscheinlichkeit = Klassifikation

BERECHNUNG TRAINIERBARE PARAMETER

Die Anzahl trainierbare Parameter in einem Netzwerk berechnet sich, indem alle Verbindungen zwischen den Neuronen zusammengerechnet werden und die Bias-Terms hinzugerechnet werden.

- Weights (Gewichte):
  - Anzahl Neuronen * Anzahl Neuronen im vorherigen Layer + … + …
- Bias Terms
  - Anzahl Neuronen im Hidden Layer und Output
- Trainierbare Parameter
  - Weights + Bias-Terms

KOSTENFUNKTION

Je nach Problem wird eine andere Kostenfunktion verwendet:

REGRESSION

$$MSE = \frac{1}{M}\sum_{m=1}^{M}\left(y^{(m)} - \hat{y}^{(m)}\right)^2$$

BINÄRE KLASSIFIKATION

$$L_{logist.} = -\sum_{m=1}^{M}\left[y_m * \log\left(\hat{y}_m\right) + \left(1 - y_m\right)\log(1 - \hat{y}_m)\right]$$

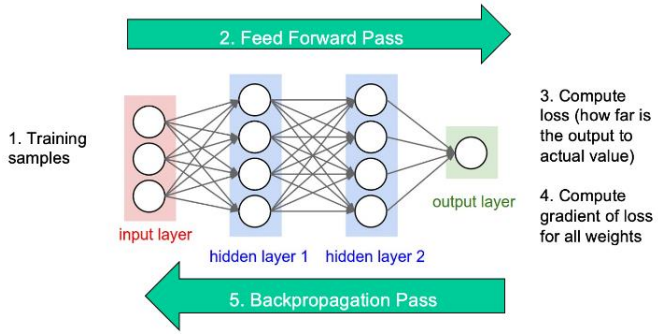
MULTINOMINALE KLASSIFIKATION

$$L_{CE} = -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} y_k^{(m)} \log\left(\hat{y}^{(m)}{}_k\right)$$

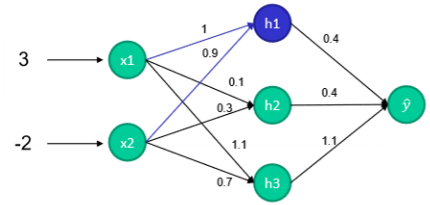
TRAINING EINES NEURONALEN NETZWERKS

Das Netzwerk lernt durch Anpassung der Gewichte und Biases basierend auf einem Optimierungsprozess, typischerweise Gradient Descent. Die Verlustfunktion eines Neuronalen Netzes ist im Normalfall **nicht konvex**.

1. **Initialisierung** der Netzwerkparameter (zufällig)
2. **Feed-Forward-Pass**: Das gesamte Datenset wird durch das Netzwerk geleitet.
3. **Verlustrechnung**: Differenz zwischen Vorhersage und tatsächlichem Wert
4. **Gradientenberechnung:** der Kostenfunktion bezüglich aller Modellparameter
5. **Anpassen** der Modelparameter um α (simultan) mithilfe von Backpropagation
   a. $\mathbf{w} = \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$
   b. $b = b - \alpha \frac{\partial L}{\partial b}$



FEEDFORWARD-PASS



- $\mathbf{b}_h = b_{out} = 0.2$
- $act_h(z) = ReLu = \max(0, x)$
- $act_{out}(z_{out}) = Sigmoid = \frac{1}{1+e^{-z}}$

NEURON H1

- $z_1 = x_1 * 1 + x_2 * 0.9 + b_{h1} = 3 * 1 - 2 * 0.9 + 0.2 = 1.4$
- $h_1 = act(z_1) = act(1.4) = 1.4$

NETZ

1. $\mathbf{x}^T \mathbf{w}_1 + \mathbf{b}_h = \mathbf{z}$
   - $\begin{bmatrix} 3 \\ -2 \end{bmatrix}^T \begin{bmatrix} 1 & 0.1 & 1.1 \\ 0.9 & 0.3 & 0.7 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}^T = \begin{bmatrix} 1.4 \\ -0.1 \\ 2.1 \end{bmatrix}$
2. $act_h(\mathbf{z}) = \mathbf{h}$
   - $act\left(\begin{bmatrix} 1.4 \\ -0.1 \\ 2.1 \end{bmatrix}\right) = \begin{bmatrix} \max(1.4, 0) \\ \max(-0.1, 0) \\ \max(2.1, 0) \end{bmatrix} = \begin{bmatrix} 1.4 \\ 0 \\ 2.1 \end{bmatrix}$
3. $\mathbf{h}^T \mathbf{w}_2 + b_{out} = z_{out}$
   - $\begin{bmatrix} 1.4 \\ 0.1 \\ 2.1 \end{bmatrix}^T \begin{bmatrix} 0.4 \\ 0.4 \\ 1.1 \end{bmatrix} + 0.2 = 3.07$
4. $act_{out}(z_{out}) = \hat{y}$
   - $act_{out}(3.07) = \frac{1}{1+e^{-3.07}} = 0.956$

BACKPROPAGATION

Das Verfahren wird verwendet, um die Gewichte und Biases in Netzwerk so anzupassen, dass der Vorhersagefehler minimiert wird. Backpropagation nutzt die Kettenregel für partielle Ableitungen, um die verschiedenen Parameter anzupassen.

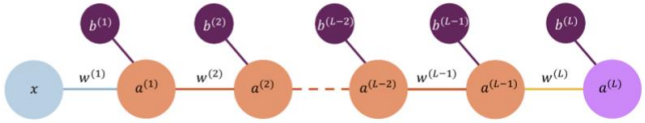
x — y — z

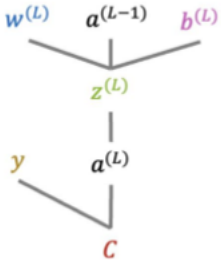
$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial z}{\partial y}$$

$z = f(y), y = g(x)$

- z hängt von y und von x ab
- Um z nach x abzuleiten muss die partielle Ableitung von y nach x und von z nach y multipliziert werden

Um nun von der Kostenfunktion C auf das $w^{(L)}$ zu kommen:



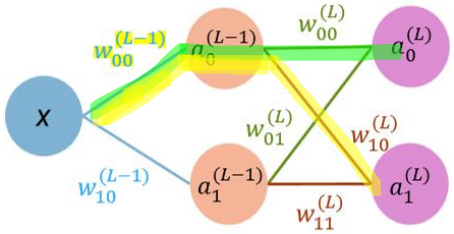


- $C = \left(y - a^{(L)}\right)^2$
- $a^{(L)} = \sigma(z^{(L)})$
- $z^{(L)} = w^{(L)} * a^{(L-1)} + b^{(L)}$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}}$$

$$\frac{\partial C}{\partial w^{(L)}} = 2\left(a^{(L)} - y\right)\sigma'\left(z^{(L)}\right)a^{(L)}$$

BACKPROPAGATION IN NETZWERK



- $z_i^{(L)} = w_{i0}^{(L)} * a_0^{(L-1)} + w_{i1}^{(L)} * a_1^{(L-1)} + b_i^{(L)}$
- $a_i^{(L)} = \sigma(z_i^{(L)})$
- $C\left(y_0 - a_0^{(L)}\right)^2 + \left(y_1 - a_2^{(L)}\right)^2$
  - $= \sum_{i=1}^{n_L}\left(y_i - a_i^{(L)}\right)^2$

Um die Gewichtung $w_{00}^{(L-1)}$ zu berechnen, müssen alle Pfade von denen $w_{00}^{(L-1)}$ mit der Kettenregel berechnet werden. Der Teil des Pfades am Ende kann dabei wiederverwendet werden, da die Pfade sich an dieser Stelle überschneiden.

$$\frac{\partial C}{\partial w_{00}^{(L-1)}} = \overbrace{\frac{\partial C}{\partial a_0^{(L)}}\frac{\partial a_0^{(L)}}{\partial z_0^{(L)}}\frac{\partial z_0^{(L)}}{\partial a_0^{(L-1)}}}^{Pfad\ 1}\overbrace{\frac{\partial a_0^{(L-1)}}{\partial z_0^{(L-1)}}\frac{\partial z_0^{(L-1)}}{\partial w_{00}^{(L-1)}}}^{identisch}$$

$$+\overbrace{\frac{\partial C}{\partial a_1^{(L)}}\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}}\frac{\partial z_1^{(L)}}{\partial a_0^{(L-1)}}}^{Pfad\ 2}\overbrace{\frac{\partial a_0^{(L-1)}}{\partial z_0^{(L-1)}}\frac{\partial z_0^{(L-1)}}{\partial w_{00}^{(L-1)}}}^{identisch}$$

$$= \overbrace{2\left(a_0^{(L)} - y_0\right) * \sigma'\left(z_0^{(L)}\right) * w_{00}^{(L)} * \sigma'\left(z_0^{(L-1)}\right) * x}^{Pfad\ 1}$$

$$+\overbrace{2\left(a_0^{(L)} - y_1\right) * \sigma'\left(z_1^{(L)}\right) * w_{10}^{(L)} * \sigma'\left(z_0^{(L-1)}\right) * x}^{Pfad\ 2}$$

$$= \left(2\left(a_0^{(L)} - y_0\right) * \sigma'\left(z_0^{(L)}\right) * w_{00}^{(L)} + 2\left(a_0^{(L)} - y_1\right) * \sigma'\left(z_1^{(L)}\right) * w_{10}^{(L)}\right) * \sigma'\left(z_0^{(L-1)}\right) * x$$

NACHTEILE VON BACKPROPAGATION

Bei vielen Hidden Layers werden aus den partiellen Ableitungen sehr lange Multiplikationsketten.

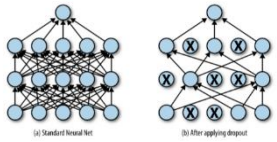
- Vanishing Gradient Problem
  - Wenn die Ableitungen sehr klein sind (z.B. mit Sigmoid), führt das zu sehr kleinen Änderungen der Gewichtungen → Training sehr langsam
  - Lösung: Verwendung von ReLu
- Exploding Gradient Problem
  - Bei zu grossen partiellen Ableitungen führt die Multiplikation zu sehr grossen Werten.

VERMEIDUNG VON OVERFITTING

DROPOUT

Zufällig ausgewählte Neuronen werden während des Trainings deaktiviert.

- Während jeder Trainings-Iteration werden andere ausgewählt
- Das zwingt das Modell sich nicht zu sehr auf bestimmte Neuronen zu verlassen.
- 20-50%



EARLY STOPPING

Lässt das Modell stoppen bevor es overfittet.

- Implementierung durch Tests alle 5 oder 10 Epochen, ob Validationset schlechter wird als Trainingsset

DATA AUGMENTION

Erweiterung der Daten durch Veränderung oder Anreicherung.

- Bilder: Flipping, Kontrast, …
- Text: Leichte Veränderung durch Synonyme
- Gausian Noise …

HYPERPARAMETER

- Anzahl Layers
- Anzahl Neuronen in Layer
- Activation Function
- Optimierungsmethode inkl. Parameter
- Batch Size
- Anzahl Epochen
- Dropout
- Early Stopping

Testen durch Cross Validation oder ein Validationset.

ARCHITEKTUREN VON NEURAL NETWORKS

- Vanilla Feed-Forward Neural Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformers

UNSUPERVISED LEARNING

Im Vergleich zu Supervised Learing stehen bei den Trainingsdaten keine Ziel- oder Ausgabewerte (also keine Label) zur Verfügung.

Ziele sind:

- Mustererkennung
- Clustering
- Dimensionsreduktion

Herausforderungen sind:

- Keine klaren Zielvorgaben
  - Bewertung schwierig

CLUSTERING

Clustering zielt darauf ab, eine Menge von Objekten bzw. Datenpunkte in Gruppen einzuteilen, sogenannte Cluster.

- Die Datenpunkte sollten eine **hohe Ähnlichkeit** zueinander aufweisen.
  - Outlier Detection
  - Datenklassen Identifikation
  - Datenverständnis
  - Datenkomprimierung

Gegeben ist ein Set aus M Datenpunkten $X = \{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$ und jeder Datenpunkt besteht wiederum aus $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \dots, x_N^{(i)}\}$ Features. Ziel ist es K Cluster zu finden, indem die Datenpunkte mit der geringsten Distanz d zusammengefasst werden.

ARTEN VON CLUSTERING

- Hard Clustering
  - Jeder Datenpunkt gehört zu nur einem Cluster
- Soft Clustering
  - Jeder Datenpunkt hat eine Wahrscheinlichkeit p_{km} zu einem Cluster zu gehören.
  - $\sum_k p_{km} = 1$

K-MEANS

Der meistgenutzte Clustering-Algorithmus ist K-Means.

- **Input**: Set aus M Datenpunkten
  - $X = \{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$
- **Output**: eine Menge aus K «Centroids»
  - $\{c_1, \dots, c_K\}$

ALGORITHMUS

1. Initialisieren von K Centroids $c_1, \dots, c_K$.
   a. Zufällig oder nach Methode (K-Means++)
2. WHILE:
   a. Berechne nächsten Centroid c_k für jeden Datenpunkt und weise den Datenpunkt diesem Centroid zu.
   b. FOR EACH Centroid c_k
      i. A_k: Menge Daten pro c_k
      ii. $m_k = \frac{1}{M_{A_k}} \sum_{x^{(m)} \in A_k} x^{(m)}$
      iii. $c_k = m_k$
   c. END FOR
3. END WHILE (Stopp-Kriterium erfüllt)
4. RETURN $c_1, \dots, c_K$

DISTANZMESSUNG

Als Distanzmessung wird die können verschiedene Masse verwendet werden:

- Euklidische Distanz
  - $d_E(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$
- Manhatten Distanz
  - $d_M(\vec{p}, \vec{q}) = \sum_{i=1}^{n} |p_i - q_i|$
- Maximum (Chebychev) Distanz
  - $d_{Cheb}(\vec{p}, \vec{q}) = \max_i |(p_i - q_i)|$

STOPP-KRITERIEN

- Wenn die Veränderung der Centroids nur noch gering ist.
- Wenn nur noch sehr wenige Neuzuweisungen durchgeführt werden.
- Stopp nach einer festen Anzahl Iterationen
- Stopp nach einer festen Zeit (z.B. 1 min)

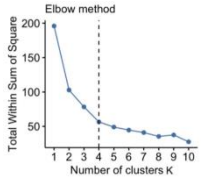
QUALITÄTSMESSUNG K-MEANS

Um einen optimalen Hyperparameterwert K zu finden, suchen wir die besten Distanzeigenschaften. Die Potential Funktion misst die Summe^2 der Distanz von jedem Datenpunkt zu seinem naheliegendsten Centroid.

- Potential function
  - $\Phi(C, X) = \sum_{m=1}^{M} \min_{c \in C} \left(d\left(x^{(m)}, c\right)^2\right)$

ELBOW-METHOD

K-Means wird mehrmals durchgeführt mit unterschiedlichen K-Werten und durch die Potentialfunktion die Qualität gemessen. Wähle K dort wo die Steigung der Kurve signifikant abnimmt.

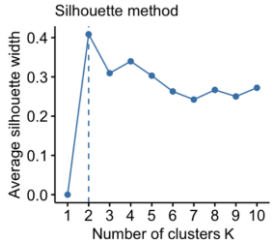


SILHOUETTE METHODE

Ist ein Ansatz zur Interpretation und Validierung der Konsistenz innerhalb von Clustern in einem Datensatz. Er liefert eine knappe grafische Darstellung, **wie gut ein Punkt innerhalb seines Clusters passt**.

- Silhouette Score $s_m = \frac{b_m - a_m}{\max(a_m, b_m)}$
  - $s_m \in [-1, 1]$
  - a_m: mittlerer Abstand des Punktes m zu den anderen Punkten im Cluster
  - b_m: geringster mittlerer Abstand zu Punkten in einem anderen Cluster
  - $\max(a_m, b_m)$: grösserer Wert
- s_m nahe 1 → Punkt passt gut ins Cluster
- s_m nahe -1 → Punkt passt schlecht (falsches Cluster)
- s_m nahe 0 → an Grenze zwischen zwei Clustern

Mit dem mittleren Silhouette-Score können auch unterschiedliche Werte für K probiert werden.

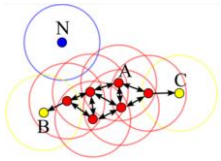


RUNTIME COMPLEXITY

- K-Means ist NP-Hard, was bedeutet, dass es kein effizientes Verfahren gibt, das garantiert die optimale Lösung in einer Zeit findet, die nur polynomial mit der Größe des Problems wächst.
- Wenn K zufällige Centroids ausgewählt werden mit M Datenpunkten mit N Features, wobei das Verfahren L Iterationen durchläuft, ist die Runtime:
  - $\mathcal{O}(LKNM)$

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) ist ein Klassifikationsalgorithmus, der sich gut eignet, um Cluster in unregelmässigen Formen zu finden.



DBSCAN ist **dichtebasiert.** Im Gegensatz zu K-Means definiert DBSCAN Cluster als Bereich grosser Dichte, die durch Bereiche geringer Dichte von anderen Clustern getrennt sind.

- **Kernpunkte**
  - Mindestanzahl benachbarte Punkte in Radius (minPts).
  - **Radius** ϵ
- **Grenzpunkte**
  - Kein Kernpunkt
  - Liegt in Radius von Kernpunkt
- **Noise**
  - Weder Kernpunkt noch Grenzpunkt

ABLAUF DBSCAN

1. Kernpunkte identifizieren
   a. Alle Punkte die genug Punkte in Radius haben
2. Cluster bilden
   a. Verbinde alle Punkte die innerhalb des Radius ϵ eines Kernpunktes liegen zu einem Cluster
3. Grenzpunkte zuordnen
   a. Alle nicht-Kernpunkte in Radius dem Cluster zuordnen
4. Noise kennzeichnen
   a. Alle übriggebliebenen Punkte sind Noise

HYPERPARAMETER

- Radius ϵ
  - Grösse des Radius um Datenpunkt
  - Kleines ϵ → mehr Cluster
- minPts
  - Mindestanzahl Punkte für Kernpunkt
  - Kleines minPts → mehr Cluster

VORTEILE

- Keine Annahmen über Clusterformen
  - Keine feste Anzahl Cluster
- Robust gegenüber Ausreissern