pd.read_csv(): Liest eine CSV-Datei und lädt sie als DataFrame in pandas.(read the dataset)
pd.to_datetime(): Konvertiert eine Spalte in ein Datetime-Objekt.
apply(): Wendet eine Funktion auf jede Zeile oder Spalte eines DataFrames an.
loc[]: Ermöglicht den Zugriff auf eine Gruppe von Zeilen und Spalten anhand von Labels oder einer bedingten Auswahl.
plot_date(): Erstellt ein Datumsdiagramm mit matplotlib.
plot(): Erstellt ein Liniendiagramm mit matplotlib.
figure(): Erstellt eine neue Figur in matplotlib.
title(): Fügt einen Titel zu einem Diagramm in matplotlib hinzu.
xlabel(): Fügt eine Beschriftung zur x-Achse in matplotlib hinzu.
ylabel(): Fügt eine Beschriftung zur y-Achse in matplotlib hinzu.
grid(): Fügt ein Raster zu einem Diagramm in matplotlib hinzu.
show(): Zeigt das Diagramm in matplotlib an.
head(): Zeigt die ersten n=5 Zeilen eines DataFrames an.
describe(): Gibt eine statistische Zusammenfassung der numerischen Spalten eines DataFrames zurück.
isna(): Gibt einen DataFrame mit booleschen Werten zurück, die anzeigen, ob ein Wert NaN ist.
covid_data.isna().sum(): Gibt zurück wie viel Wertet NaN sind.
covid_data.isna().sum(axis=1): Um über Spalten zu summieren.
sum(): Summiert die Werte entlang einer Achse eines DataFrames.
drop(): Entfernt angegebene Zeilen oder Spalten aus einem DataFrame.
dropna(): Entfernt alle Zeilen oder Spalten mit fehlenden Werten.
drop_duplicates(): Entfernt doppelte Zeilen aus einem DataFrame.
covid_data.dropna(thresh=3): Zeilen mit weniger als 3 nicht fehlenden Werten entfernen
ffill(): Füllt fehlende Werte mit dem letzten bekannten Wert.
Ffillna(99): Füllt NaN werte mit 99 auf
duplicated(): Gibt eine boolesche Serie zurück, die anzeigt, ob eine Zeile ein Duplikat ist.
value_counts(): Gibt eine Serie zurück, die die Häufigkeit der eindeutigen Werte einer Spalte zählt.
unique(): Gibt die eindeutigen Werte einer Spalte zurück.
fillna(): Füllt fehlende Werte mit einem angegebenen Wert.
to_csv(): Speichert einen DataFrame als CSV-Datei.
urlparse(): Analysiert eine URL in ihre Komponenten.
scatter(): Erstellt ein Streudiagramm mit matplotlib.
wget: Ein Befehl zum Herunterladen von Dateien aus dem Internet.
print(): Gibt den angegebenen Text oder die angegebenen Werte aus.
len(): Gibt die Länge (Anzahl der Elemente) eines Objekts zurück.
sorted(): Gibt eine sortierte Liste der angegebenen Werte zurück.
lambda: Erstellt eine anonyme Funktion.
covid_data['time'].value_counts(): Zählen, wie oft jeder einzelne „time“-Wert erscheint
pd.DataFrame: stellt eine tabellenartige Struktur mit Zeilen und Spalten wie excel

Einführung in Machine Learning (ML)

Machine Learning (ML) ist ein Teilgebiet der Künstlichen Intelligenz (AI), das sich mit Algorithmen beschäftigt, die aus Daten lernen, um Vorhersagen oder Entscheidungen zu treffen, ohne explizit programmiert zu sein.

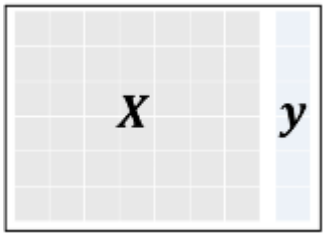
- **Modell:** Mathematische oder probabilistische Beziehung zwischen Variablen.
- **Training:** ML-Modelle passen sich basierend auf Trainingsdaten automatisch an.
- **Ziele von ML:** Mustererkennung, Vorhersagen treffen, Entscheidungsgrundlage schaffen.

Deep Learning ist eine spezielle Form von ML, die neuronale Netzwerke verwendet.

Supervised Learning

Ein Algorithmus wird mit **annotierten Daten** (Eingabe + erwartete Ausgabe) trainiert.

- **Beispiel:** Sentiment-Analyse von Texten (positive, negative, neutrale Sätze).
- **Ziel:** Eine Funktion f(x)f(x) finden, die Eingaben xx auf die erwarteten Ausgaben yy abbildet.
- **Datenstruktur:**
  - **Feature-Matrix X:** Enthält die Merkmale (z. B. Länge eines Textes).
  - **Zielvariable y:** Enthält die erwarteten Ausgaben (z. B. „positiv“ oder „negativ“).
- **Wichtige Algorithmen:** Support Vector Machines, Entscheidungsbäume, Neuronale Netzwerke.

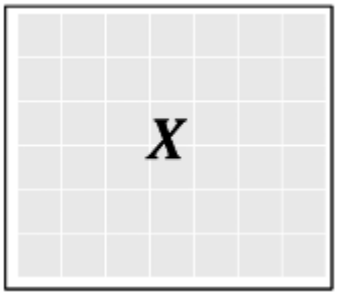


Unsupervised Learning

Reinforcement Learning

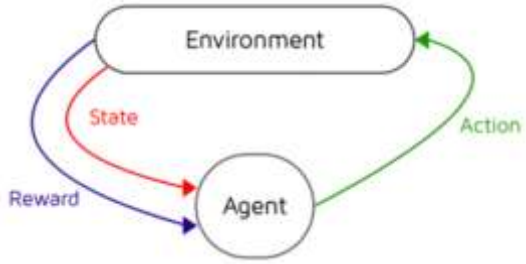
Daten enthalten keine vorgegebenen Labels – Ziel ist die Strukturierung und Mustererkennung.

- Beispiel: Kunden-Clustering basierend auf Alter und Ausgaben.
- Herausforderungen:
  - Keine klaren Labels zum Training.
  - Evaluierung ist schwieriger.
- Typische Methode: K-Means-Clustering.



Ein Agent interagiert mit seiner Umgebung, führt Aktionen aus und erhält Belohnungen oder Strafen.

- Ziel: Lernen einer optimalen Strategie (Policy) zur Maximierung der Belohnung.
- Beispiel: Ein autonomer Roboter lernt, Hindernissen auszuweichen.



Klassifikation vs. Regression

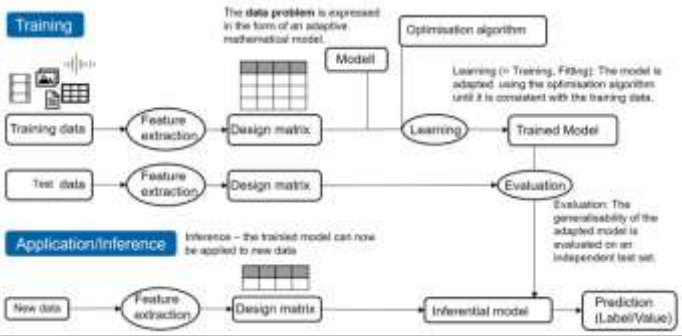
- Klassifikation: Zielvariable ist kategorisch (z. B. Spam vs. Nicht-Spam).
- Regression: Zielvariable ist numerisch (z. B. Vorhersage eines Autopreises).

Evaluierung von Supervised Learning

- Trainings- und Testset: Daten werden in Trainings- und Testdaten aufgeteilt, um die Generalisierungsfähigkeitdes Modells zu messen.
- Metriken:
  - Klassifikation: Accuracy, Precision, Recall.
  - Regression: Mean Squared Error (MSE).

Supervised Learning Pipeline

1. Feature-Extraktion: Daten in eine strukturierte Form bringen (Design-Matrix).
2. Modelltraining: Parameter iterativ anpassen.
3. Evaluation: Modell auf Testdaten prüfen.
4. Anwendung: Modell für neue Vorhersagen nutzen.
5. Iteration: Falls nötig, Modell verbessern oder an neue Daten anpassen.



Lineare Regression

$$h_{\theta_0,\theta_1}(x) = \theta_0 + \theta_1 x \quad == \quad \hat{y} = b \cdot x + a$$

Residual Sum of Squares *RSS*

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

Cost Function *J*

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

The Normal Equation

$$\begin{pmatrix} \varepsilon^{(1)} \\ \varepsilon^{(2)} \\ \vdots \\ \varepsilon^{(M)} \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(M)} \end{pmatrix} - \begin{pmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ 1 & \vdots \\ 1 & x^{(M)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$$

$$\varepsilon = y - X\theta$$

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

Multivariate Linear Regression

Linear Regression

1. The relationship between X and y is linear.
2. The residuals are independent of each other.
3. The expected output values are normally distributed.

Hypothesis $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^\top x$

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

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

$$\hat{y} = b_1x_1 + b_2x_2 + \ldots + b_ix_i + \ldots + b_kx_k + a$$

$\hat{y}$ – Wert auf der Kriteriumsvariable, der vorhergesagt werden soll

b_1 – Regressionsgewicht des ersten Prädiktors

x_1 – Messwert auf dem ersten Prädiktor

b_i – Regressionsgewicht eines beliebigen Prädiktors i

x_i – Messwert auf dem beliebigen Prädiktor i

b_k – Regressionsgewicht des letzten Prädiktors k

x_k – Messwert auf dem letzten Prädiktor k

a – Regressionskonstante

Lineare Regression Bsp:

$$\theta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = \frac{(-2 \cdot -1.06) + (-1 \cdot -0.06) + (0 \cdot -0.76) + (1 \cdot 0.69) + (2 \cdot 0.19)}{(-2)^2 + (-1)^2 + 0^2 + 1^2 + 2^2} = 0.425$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x} = 2.06 - (0.425 \cdot 3) = 2.06 - 1.275 = 0.785$$

$$\bar{x} = \frac{1+2+3+4+5}{5} = 3$$

$$\bar{y} = \frac{1+2+2.3+3.75+2.25}{5} = 2.06$$

X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25

Evaluating Regression Models

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 Deviation: $RMSD = \sqrt{MSE} = \sqrt{\frac{\sum_{i=1}^{I}(y^{(i)} - \hat{y}^{(i)})^2}{I}}$

Coeicient of Determination

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Univariate Linear Regressio

Hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$

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

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

Learning Rate in Gradient Descent

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{for every j=1..n}$$

Learning Rate α

General Rules Learning Rate α

- **Learning Rate α too small -> slow convergence**
- **Learning Rate α too large -> might "jump too far", might not converge or even diverge**

Assumptions of Linear Regression

Linearity: The input and output values have a linear relationship

Independence: The outcome of one sample does not affect the others

Normality: Errors should be normally distributed, i.e. larger deviations from mean should be less likely

Equality of Variance ("Homoscedasticity"): Error distribution should be the same for all input values

Gradient Descent for Linear Regression

$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ ------> $\theta_j = \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

Batch Gradient Descent

Berechnet den Gradienten über den gesamten Datensatz, was genau, aber rechenintensiv ist.

$\theta_j = \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ for every j=1..n

Stochastic Gradient Descent (SGD)

for m = 1 to M {

$$\theta_j = \theta_j - \alpha(h_\theta(x^{(m)}) - y^{(m)})x_j^{(m)} \quad \text{for every j = 1..n}$$

}

Higher Order Polynomials

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 (x_2)^3 + \theta_3 \sqrt{x_1} + \theta_4 x_2{}^3 x_3 x_4$$

Overfitting: Das Modell passt sich zu stark an die Trainingsdaten an, einschließlich von Rauschen und unwichtigen Details. Es erzielt hohe Genauigkeit auf den Trainingsdaten, aber schlechte Leistung auf neuen Daten.

Underfitting: Das Modell ist zu simpel und kann die zugrunde liegenden Muster der Daten nicht erfassen. Es hat eine schlechte Leistung sowohl auf Trainings- als auch auf Testdaten.

Regularization

Hypothesis: $h_\theta(x) = \theta^T x = \sum_{j=1}^{n}\theta_j x_j$

Cost Function: $J(\theta) = \frac{1}{2M}\left[\sum_{i=1}^{M}\left(y^{(m)} - h_\theta(x^{(m)})\right)^2 + \lambda \sum_{j=1}^{n}\theta_j{}^2\right]$

Hyperparameter Tuning (or "Model Selection"):

Linear Regression vs Logistic Regression:

Lineare Regression:

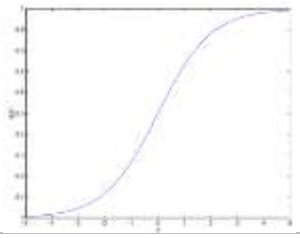
The goal of Model Selection is to find good hyperparameters for a given algorithm. These hyperparameters are then used for training the parameters of the algorithm.

Algorithm	Hyperparameter	Parameters
Nearest Neighbor Regression	- Number of neighbors k	- None
Polynomial Regression	- Polynomials of the input variables - Regularization parameter λ - Learning Rate α	- Values of $\theta_1 \dots \theta_n$
Logistic Regression	- Regularization parameter λ - Learning Rate α	- Values of $\theta_1 \dots \theta_n$

$$h_\theta(x) = \theta^T x$$

Logistic Regressio:

$$h_\theta(x) = g(\theta^T x)$$



Gradient Descent Bsp:

Given is a very small data set with 5 points.

X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25

1. We initialize Gradient Descent with $\theta_0 = 1$ und $\theta_1 = 0.5$. Draw a scatter plot of *X* and *Y* by hand and the corresponding regression line.
2. Calculate one step of gradient descent with $\alpha = 0.1$, and draw the resulting regression line

$$\theta_0 = \theta_0 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

= 1 – 0.1*1/5*((1+0.5*1-1) + (1+0.5*2-2) + (1+0.5*3-1.39) + (1+0.5*4-3.75) + (1+0.5*5-2.25)) = 0.958

Use OLD θ here!

$$\theta_1 = \theta_1 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

= 0.5 – 0.1*1/5*((1+0.5*1-1)*1 + (1+0.5*2-2)*2 + (1+0.5*3-1.3)*3 + (1+0.5*4-3.75)*4 + (1+0.5*5-2.25)*5) = 0.353

Generic Gradient Descent Algorithm

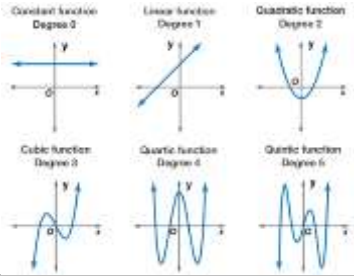
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \text{ for every j=1..n}$$

Gradient Descent for Linear Regression

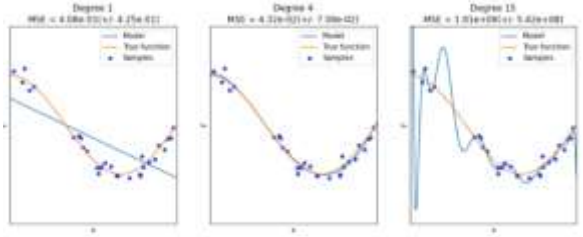
Repeat until convergence:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \longrightarrow \theta_j = \theta_j - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Higher Order Polynomials



Higher Order Polynomials might Overfit



Underfitting Overfitting

- Overfitting bedeutet, dass ein Modell die Trainingsdaten zu genau lernt und auf neue Daten schlecht generalisiert.
- Underfitting tritt auf, wenn ein Modell die zugrunde liegenden Muster nicht ausreichend erfasst und generell schlecht performt.

Regularization

Hypothesis: $h_\theta(x) = \theta^T x = \sum_{j=1}^{n} \theta_j x_j$

Cost Function: $J(\theta) = \frac{1}{2M} \left[\sum_{i=1}^{M} \left(y^{(m)} - h_\theta(x^{(m)}) \right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$

Regularization Parameter

Logistic Function

$$\frac{1}{1+e^{-z}} \qquad \frac{1}{1+e^{-20z}} \qquad \frac{1}{1+e^{-20z+50}}$$

Logistic Regression Bsp

Suppose we want to predict lung cancer from a person's exposure to radon and asbestos:

- $y = 1$ if person develops lung cancer; $y = 0$ otherwise.
- x_1 = kilograms of asbestos inhaled
- x_2 = average microCuries of radiation at home
- Hypothesis: $\hat{y} = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

We train the machine and get θ_0=0, θ_1=1.5, θ_2=0.22.

What is the machine's prediction for a person who inhales 2 grams of asbestos and whose home has an average of 4 microCuries?

- $y = 1$ if person develops lung cancer; $y = 0$ otherwise.
- x_1 = kilograms of asbestos inhaled = 0.002 (note the units!)
- x_2 = average microCuries of radiation at home = 4
- Hypothesis: $\hat{y} = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ = g(0 + 1.5*0.002 + 0.22*4) = g(0.883) =~ 0.707

Log Loss

Misst, wie gut ein Klassifikationsmodell die Wahrscheinlichkeiten für binäre (oder multiple) Klassen vorhersagt.

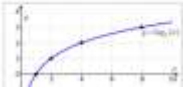
Loss Function "Log-Likelihood":

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

$$= -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

Cost Function

$$J(\theta) = \frac{1}{M} \sum_{m=1}^{M} Loss(h_\theta(x^{(m)}), y^{(m)})$$



Logistic Regression with Log Loss

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x^{(m)}$$

Gradient Descent for Logistic Regressio

$$J(\theta) = -\frac{1}{M} \sum_{m=1}^{M} [y^{(m)} \log h_\theta(x^{(m)}) + (1 - y^{(m)}) \log(1 - h_\theta(x^{(m)}))]$$

This leads[1] to the following update rule in Gradient Descent:

Repeat until convergence:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \longrightarrow \theta_j = \theta_j - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x_j^{(m)}$$

Linear Regression

Hypothesis: $h_\theta(x) = \theta^T x$

Cost Function: Mean Squared Error

$$J(\theta) = \frac{1}{2M}\sum_{m=1}^{M}(h_\theta(x^{(m)}) - y^{(m)})^2 = \frac{1}{2M}\sum_{m=1}^{M}(\theta^T x^{(m)} - y^{(m)})^2$$

Logistic Regression

Hypothesis: $h_\theta(x) = g(\theta^T x)$

Loss Function "Log-Likelihood":

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

Cost Function

$$J(\theta) = \frac{1}{M}\sum_{m=1}^{M} Loss(h_\theta(x^{(m)}), y^{(m)})$$

Accuracy BSP

Recall that a trained logistic regression machine produces an output $\hat{y}$ in (0,1).

We can apply a threshold τ to decide the final label, i.e.:
- If $\hat{y} > \tau$, then we predict the example is positive.
- If $\hat{y} \leq \tau$, then we predict the example is negative.

Suppose the true labels of a test set are y = [0, 0, 1, 1].
Suppose the machine's predictions are $\hat{y}$ = [0.2, 0.6, 0.5, 0.9].

What is the accuracy of the machine for a threshold of:
- τ = 0.5 ? Predictions = [0, 1, 0, 1] => Acc = 0.5
- τ = 0.7 ? Predictions = [0, 0, 0, 1] => Acc = 0.75

Accuracyn (Genauigkeit)

For regression problems, we care about how "close" the prediction was to the true answer. Over an entire dataset, we calculate the residual sum of squares, etc.:

$$RSS = \sum_{m=1}^{M}(h_\theta(x^{(m)}) - y^{(m)})^2$$

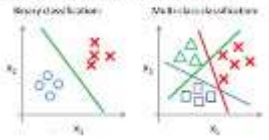
For classification problems, we care about whether the model predicted the correct class. Over an entire dataset, we thus calculate the proportion of examples classified correctly, sometimes called the accuracy:

$$Acc = \sum_{m=1}^{M} 1[h_\theta(x^{(m)}) = y^{(m)}]$$

where 1[...] is an indicator function that equals 1 if the condition is true, 0 if it is false.

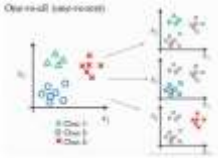
Multi-class Classification

Binary vs multi-class classification



One-vs-Rest approach

- K classes
- K binary classification problem: each class vs all the others
- K probabilities p_k
- Choose the highest probability to predict the class.



Examples of multi-class classification

- Visual digit recognition in OCR
- Obstacle recognition in autonomous driving

Clustering

Is a way of grouping things that are similar to each other.

Hard clustering: Each data point is assigned a unique cluster (belongs to one cluster);
Soft clustering: Each data point m is assigned a probability p_{km} that it belongs to cluster k for $k = 1 : K$ (K being the total number of clusters) such that $\sum_k p_{km} = 1$.

The K-Means Algorithm

Ist ein clustering algo.

Schritte:

1. Wähle k anfängliche Zentroiden
2. Wiederhole:
   - Weise jeden Punkt dem nächsten Zentroid zu
   - Berechne neue Zentroiden als Mittelpunkt aller zugewiesenen Punkte
3. Stoppe bei Konvergenz (keine Änderungen mehr)

Selection of initial centroids

- *Random points* randomly selects K points in $\mathbb{R}^N$;
- The *Forgy method* randomly chooses K observations from the dataset and uses these as the initial means;
- The *Random Partition* method first randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points.

K-means++:

1. Randomly select the first centroid from the data points
2. For each data point compute its distance from the nearest, previously chosen centroid (the minimum distance from all centroids)
3. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its squared distance from the nearest, previously chosen centroid. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)
4. Repeat steps 2 and 3 until K centroids have been sampled.

Stop-Criterion

- Stop when the coordinates of the centroids change only very little from one iteration to the next
- Stop when only very few data points are re-assigned to a different centroid in a new iteration.

Apart from this, we can also use time-boxed criteria:

- Stop after a fixed number of iterations (e.g. after 20-50 iterations)
- Stop after a certain time for the entire computation has elapsed (e.g. after 1 minute).

Quality of K-Means Clustering:

Durch die zufällige Initialisierung können K-Means-Durchläufe zu unterschiedlichen Ergebnissen führen

Zur Qualitätsbewertung wird die **Potentialfunktion Φ** verwendet

Die Potentialfunktion misst die Summe der quadratischen Abstände zwischen jedem Datenpunkt und seinem nächsten Zentroid:

- $\Phi(C, X) = \Sigma(\min(d(x^*(m), c)^2))$

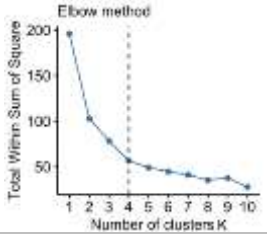
In der Praxis führt man K-Means mehrmals mit verschiedenen zufälligen Initialisierungen durch

Choosing the number of clusters K

- K-Means erfordert die Vorgabe der Clusterzahl K
- Möglichkeiten zur Bestimmung von K:
  1. Vorwissen über erwartete Clusteranzahl nutzen
  2. Domänenexperten befragen
  3. Spezifische Anzahl für nachfolgende Verarbeitung wählen
- In den meisten realen Anwendungen:
  - Die "korrekte" Anzahl ist nicht klar erkennbar
  - Verschiedene K-Werte können sinnvoll sein
  - Mehrere Interpretationen der Datenstruktur sind möglich (z.B. 2, 4 oder 6 Cluster)

The Elbow Method:

In this method, we run K-Means with diferent values of K and plot the value of the potential function Φ for each value of K.



The Silhouette Method:

Runtime Complexity K-Means

- Bewertet die Qualität einer Clusterzuordnung für jeden Datenpunkt
- Für Datenpunkt x^(m):
  - a_m = durchschnittlicher Abstand zu allen Punkten im eigenen Cluster
  - b_m = kleinster durchschnittlicher Abstand zu Punkten in anderen Clustern
  - Silhouette-Score s_m = (b_m - a_m) / max(b_m, a_m)
  - Wertebereich: [-1, 1]

Interpretation:

- Nahe 1: Punkt passt sehr gut in seinen Cluster und ist weit von anderen entfernt
- Nahe 0: Punkt liegt an der Grenze zwischen Clustern
- Nahe -1: Punkt ist vermutlich im falschen Cluster zugeordnet

Average Silhouette Score (Width):

- Durchschnitt aller Silhouette-Scores im Datensatz: (1/M)∑s_m
- Misst die Gesamtqualität des Clusterings

$\frac{1}{M}\sum_{m=1}^{M} s_m$.

Distance Metrics

The L_1 metric, known as the Manhattan distance defined as $d_1(p,q) = \sum_i |p_i - q_i|$ and illustrated here:

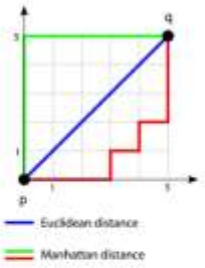


Figure 10.11: Manhattan vs Euclidean Distance

The L_{inf} or the maximum metric defined as $d_{inf}(p,q) = \max_i\{|p_i - q_i|\}$.

The Cosine similarity defined as $d_{cos}(p,q) = \frac{\sum_{i=1}^{N} p_i q_i}{\sqrt{\sum_{i=1}^{N} p_i^2}\sqrt{\sum_{i=1}^{N} q_i^2}}$, quantifies the similarity among two vectors by only accounting for the angle between them and ignoring their lengths. This similarity equals the cosine of the angle between the vectors, and thus, orthogonal vector have a cosine similarity of 1, proportional vectors have a similarity of 1, and opposite vectors have a similarity of -1.

Assume we are given M data points with N features. Assume further that K-Means stops after L iterations. Then the *runtime* of K-Means is $\mathcal{O}(LKNM)$.

- In the first step, we randomly select K initial centroids with N dimensions each: this step will run in $\mathcal{O}(KN)$.
- In each of the L iterations, we compute the distance between each data point and each centroid. The euclidean distance computation takes $\mathcal{O}(N)$. There are M data points and K centroids, thus we need total time $\mathcal{O}(KNM)$ per iteration.
- There are L iterations, each taking $\mathcal{O}(KNM)$, leading to a total runtime of $\mathcal{O}(LKNM)$.

DBSCAN

Grundkonzept:

- Density-based Spatial Clustering of Applications with Noise
- Identifiziert Cluster als dichte Regionen, getrennt durch dünn besetzte Bereiche
- Klassifiziert Datenpunkte in: Core Points, Border Points und Noise Points

Parameter:

- ε (Epsilon): Maximaler Abstand für Nachbarschaftsdefinition
- minPts: Mindestanzahl von Punkten für dichte Region

Punkttypen:

- Core Point: Hat mindestens minPts Nachbarn innerhalb ε
- Border Point: Weniger als minPts Nachbarn, aber mindestens ein Core Point in ε-Umgebung
- Noise Point: Weder Core noch Border Point

Algorithmus:

1. Wähle unbearbeiteten Punkt P
2. Falls P kein Core Point ist: klassifiziere als Noise, gehe zu Schritt 1
3. Falls P ein Core Point ist: bilde neuen Cluster mit P und allen seinen Nachbarn
4. Füge rekursiv alle erreichbaren Core Points mit ihren Nachbarn zum Cluster hinzu
5. Wiederhole bis alle Punkte verarbeitet sind

Vorteile:

- Keine Vorgabe der Clusteranzahl nötig
- Erkennt beliebige Clusterformen
- Identifiziert Ausreißer explizit

Nachteile:

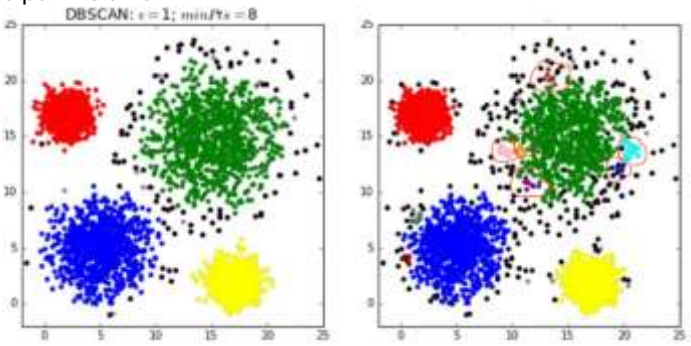
- Probleme bei unterschiedlichen Dichteverhältnissen

Komplexität:

- Naive Implementierung: O(M²)
- Optimiert mit geeigneter Datenstruktur: O(M·logM)

DBSCAN BSP.

Epsilon muss kleiner werden damit neue cluster entstehen können. MinPts wird kleiner da die neune cluster weniger als 8 punkte sind.



Binary Predictions

Suppose the true labels of a test set are y = [0, 0, 1, 1] and that the machine's predictions are $\hat{y}$ = [0.2, 0.6, 0.5, 0.9].

We are using two different thresholds, $\tau_1 = 0.5$ and $\tau_2 = 0.7$.

1. **Binary prediction:** $\tau_1 = 0.5$ -> [0, 1, 0, 1] $\tau_2 = 0.7$ -> [0, 0, 0, 1]
2. **Correctly classified samples:** τ_1 -> 2/4 τ_2 -> 3/4

Accuracy(Genauigkeit)

$$\text{Accuracy} = \frac{\#\ correct\ samples}{\#\ all\ samples}$$

Confusion Matrix:

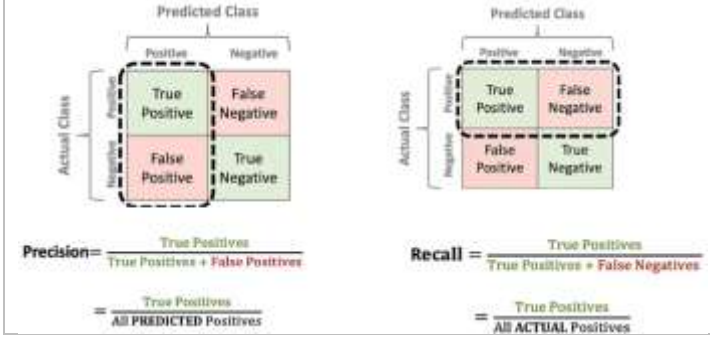
		Actual Value		
		Car	House	Ball
Predicted Value	Car	12	3	4
	House	2	8	1
	Ball	17	20	143

1: Accuracy is:

(12+8+1430) / (12+3+4+2+8+1+17+20+1430) = 1450 / 1487 = 0.968

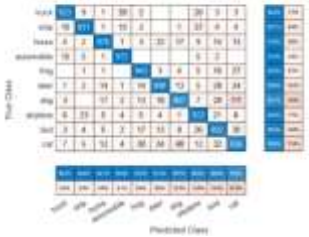
2. A "Dummy-Classifier" that always outputs "neutral" achieves an accuracy of : 1430/1487 = 0.96, although it does not do anything.

Precision and Recall



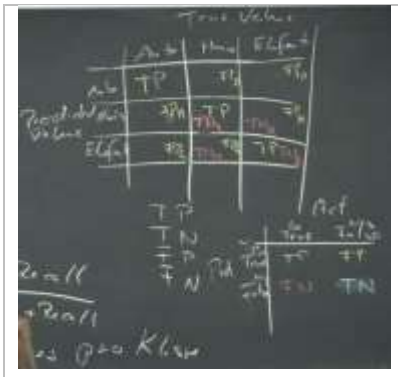
Multi-Class Confusion Matrix

For multi-class (>2) problems, we can use a confusion matrix to describe the model's accuracy:



How many fals positives are ther for Class «Frog»?

Enter the number at https://b.socrative.com/student Room 432779



	diagnosed sick	diagnosed healthy
sick	1000 true positives	200 false negatives
healthy	800 false positives	800 true negatives

a)

Precision: 55,5%

Recall: 83,3%

b) High recall, since we want to find and treat at least all sick people – even if there might be some false-positives

Precision (Genauigkeit der positiven Vorhersagen)

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1000}{1000 + 800} = \frac{1000}{1800} \approx 0{,}555 \Rightarrow 55{,}5\%$$

Recall (Sensitivität / Trefferquote)

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1000}{1000 + 200} = \frac{1000}{1200} \approx 0{,}833 \Rightarrow 83{,}3\%$$

Ein falsch-positiver Patient bekommt vielleicht unnötige Tests.
Ein falsch-negativer Patient bleibt unbehandelt was lebensgefährlich sein kann.

Precision and Recall: Solution:

τ=0.2

Index (m)	1	2	3	4
Label (y)	1	1	0	1
Prediction ($\hat{y}$)	0.8	0.4	0.7	0.95
Binary Prediction	1	1	1	1

$$precision = \frac{|\text{PredictedPositive} \cap \text{ActualPositive}|}{|\text{PredictedPositive}|} = \frac{|\{1,2,3,4\} \cap \{1,2,4\}|}{|\{1,2,3,4\}|} = \frac{|\{1,2,4\}|}{|\{1,2,3,4\}|} = 3/4$$

$$recall = \frac{|\text{PredictedPositive} \cap \text{ActualPositive}|}{|\text{ActualPositive}|} = \frac{|\{1,2,3,4\} \cap \{1,2,4\}|}{|\{1,2,4\}|} = \frac{|\{1,2,4\}|}{|\{1,2,4\}|} = 1$$

Combination of Recall and Precision

Definition of F1-Score for Class C: $F_1\ for\ C = 2 * \frac{precision * recall}{precision + recall}$

Definition of F_β -Score:

$$F_\beta\ for\ C = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall}$$

The «F-Score» is then the average over all or some classes:

$$F = \frac{F_{pos} + F_{neg}}{2} \quad \text{or} \quad F = \frac{F_{Cat} + F_{House} + F_{Ball}}{3}$$

F-Score for Multiclass Classification

		Actual Value				
		Positive	**Negative**	**Neutral**	**PRECISION**	**RECALL**
Predicted Value	**Positive**	12	3	4	0.63	0.38
	Negative	2	8	1	0.72	0.26
	Neutral	17	20	143	0.79	0.96

For each class an F-score can be computed

$$F_{pos} = 2\frac{P_{pos} * R_{pos}}{P_{pos} + R_{pos}} = 0.474 \qquad F_{neg} = 0.382 \qquad F_{neutral} = 0.867$$

Then we can use the average of all F-scores for the entire model:

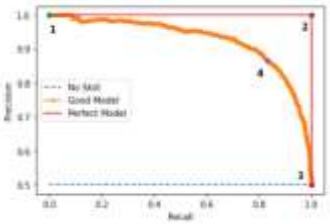
$$F_1 = \frac{F_{pos} + F_{neg} + F_{neu}}{3} = (0.474 + 0.382 + 0.867) / 3 = 0.574$$

Precision-Recall Curve

a) Person 1: Explain to your partner what points 1, 3 and 4 are.

b) Person 2: Explain to your partner why the red line represents a "perfect" model.

c) How would you find the "best" value for threshold τ?

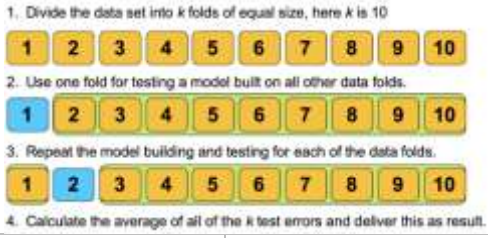


k-fold Cross Validation (CV)

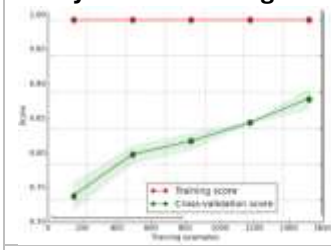
Ein einzelner Testdatensatz ist oft nicht repräsentativ. Stattdessen wird das gesamte Dataset in k gleich große Teile (Folds) aufgeteilt (z. B. k=10).

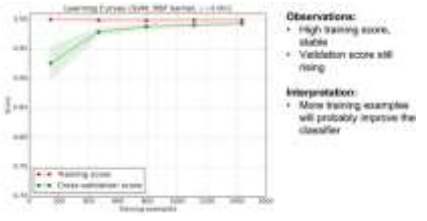
Ablauf:

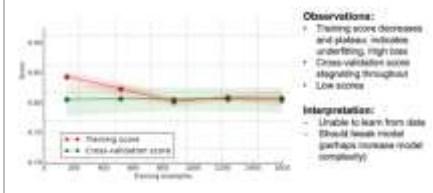
1. **Dataset in k Folds aufteilen.**
2. **Einer der Folds wird als Testdaten verwendet, die anderen als Trainingsdaten.**
3. **Dieser Vorgang wird k-mal wiederholt, wobei jeweils ein anderer Fold als Testdaten dient.**
4. **Die Testfehler aus allen k Durchläufen werden gemittelt → ergibt die Cross-Validation-Genauigkeit.**



Analyze the following Learning Curve:

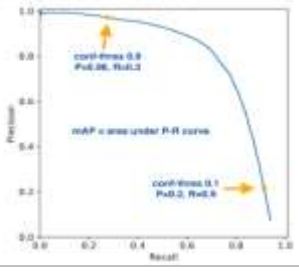






Precision-Recall Curve

By systematically varying the threshold over [0,1], we can trace out a Precision-Recall Curve.
It gives a holistic view of the classifier's behavior.



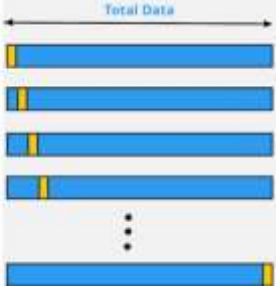
Leave-one-out Cross Validation LOOC

De-facto n-fold cross validation, with n number of training samples

Train the model on *n*-1 samples and evaluate it on the single data point

Computing expensive

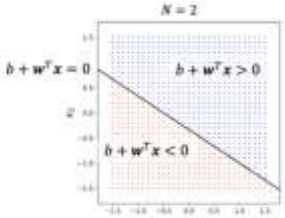
Useful especially for small datasets



Support Vector Machines (SVM)

Eine SVM sucht nach der besten Linie, die diese beiden Arten trennt, und zwar so, dass der Abstand von der Linie zu den nächsten Punkten beider Klassen maximal ist.

Classification using a Separating Hyperplane:



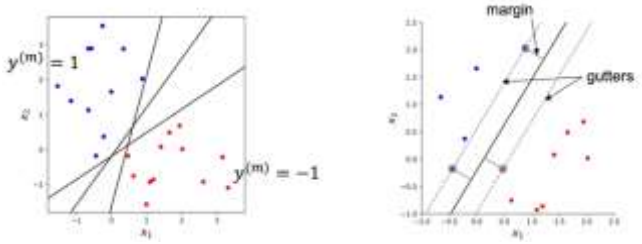
If the parameters b, $\mathbf{w}$ are known, a sample $\mathbf{x}^{(*)}$ can classified according to

$$y^{(*)} = \text{sign}(\mathbf{w}^T\mathbf{x}^{(*)} + b)$$

Maximal Margin Classifier: The Approach:

Goal: Find a hyperplane that separates the two classes of training samples with the largest possible margin.

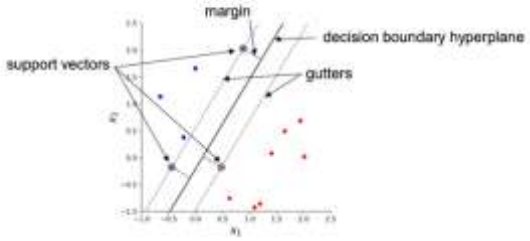
Margin = distance of the hyperplane to the closest training samples (support vectors)



Support Vectors:

are equidistant to the separating hyperplane

in contrast to other machine learning methods, the maximal margin hyperplane depends directly on the support vectors, but not on the other samples



Dual Form:

$$\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i\alpha_j\, y^{(i)}y^{(j)}x^{(i)^T}x^{(j)} \quad + \sum_{m=1}^{M}\alpha_m$$

$$\Rightarrow \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha})$$

subject to $\sum_{m=1}^{M}\alpha_m\, y^{(m)} = 0$ and $\alpha_m \geq 0$ for m = 1..M

Predictions from the Optimised Lagrangian

With the optimal $\hat{a}_m$ we can compute the optimal $\hat{\mathbf{w}}$ from the equation in the previous slide

$$\hat{\mathbf{w}} = \sum_{m=1}^{M}\hat{a}_m y^{(m)}\mathbf{x}^{(m)}$$

and

$$\hat{b} = \frac{1}{M_s}\sum_{m=1}^{M_s}\left(y^{(m)} - \hat{\mathbf{w}}^T\mathbf{x}^{(m)}\right)$$

for the support vectors.

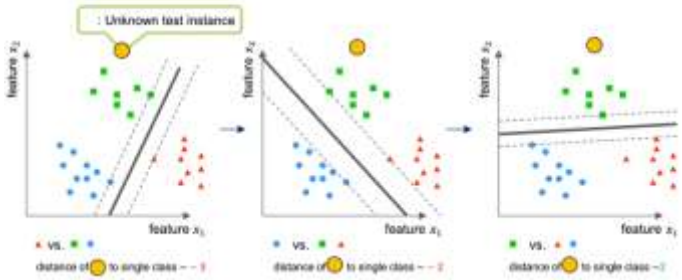
Number of support vectors $\hat{a}_m > 0$

Then, a new sample $\mathbf{x}^{(*)}$ can be classified using $y^{(*)} = \text{sign}(\hat{\mathbf{w}}^T\mathbf{x}^{(*)} + \hat{b})$

The distance from the hyperplane can be interpreted as a confidence for the prediction.

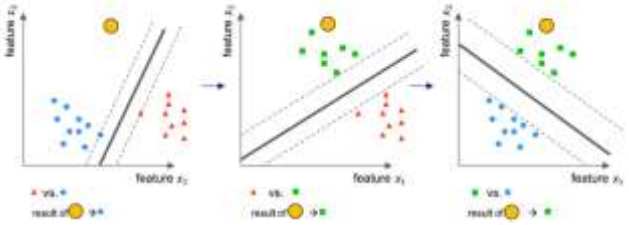
One vs. Rest (One vs. All)

- SVM is a binary classifier, so it can only separate two classes
- more than 2 classes? → #*classes* times 'one vs. rest'
- has highest positive distance (confidence) in green case



One vs. One:

- assume k classes and learn all $k(k-1)/2$ pairwise comparisons
- classify unknown instance by majority vote among all pairwise comparisons



The Cost Function in Soft-Margin SVM:

Slack variables $\varepsilon^{(m)} \geq 0$ allow samples to be on the wrong side of the margin

Hard constraint $y^{(m)}(\mathbf{w}^T\mathbf{x}^{(m)} + b) \geq 1 \rightarrow$ soft constraint $y^{(m)}(\mathbf{w}^T\mathbf{x}^{(m)} + b) \geq 1 - \varepsilon^{(m)}$

The new cost function includes a regularization term:

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{m=1}^{M}\varepsilon^{(m)} \quad \text{subject to} \quad y^{(m)}\left(\mathbf{w}^T\mathbf{x}^{(m)} + b\right) \geq 1 - \varepsilon^{(m)}$$

for $m = 1, \ldots, M$

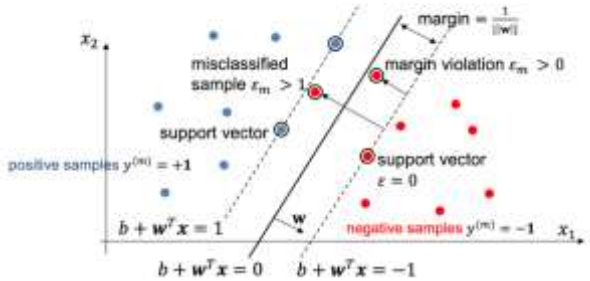
Note that there is only one parameter C

Every constraint can be satisfied if $\varepsilon^{(m)}$ is sufficiently large

Optimisation of the cost function: This is still a quadratic optimisation problem which can be solved essentially the same way as the hard margin classifier.

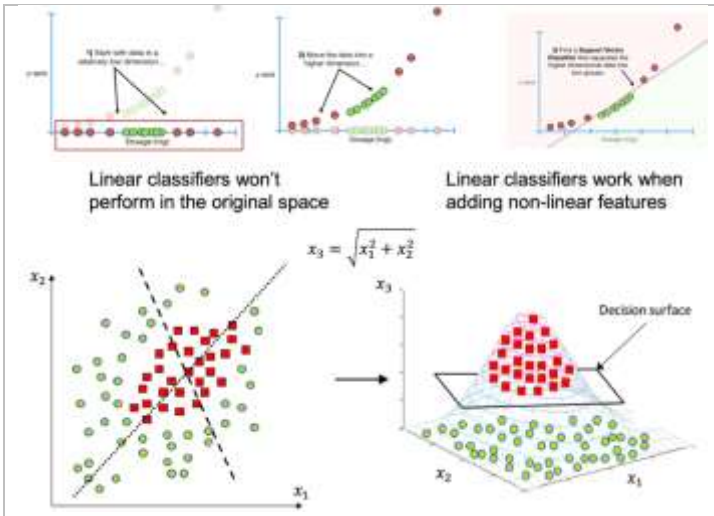
The Values of the Slack Variables $\boldsymbol{\varepsilon_m}$

Introduce new variables ε_m that "keep track" of how much each point is misclassified



kernel trick:

Lifting Trick: Projection into Higher Dimensions:

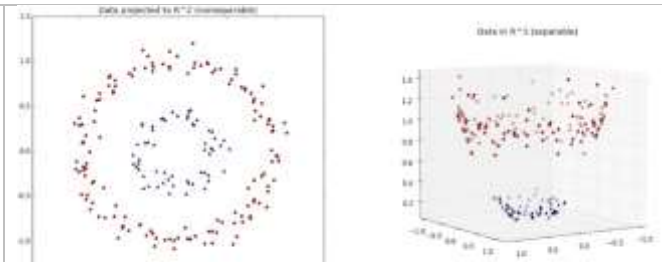


Linear classifiers won't perform in the original space

Linear classifiers work when adding non-linear features

$x_3 = \sqrt{x_1^2 + x_2^2}$

Decision surface



Transformation: $(x_1, x_2) \rightarrow (x_1, x_2, {x_1}^2 + {x_2}^2)$

Transformation:

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}x_1x_2 \\ {x_2}^2 \end{pmatrix}$$

How can we use these kernels?:

Hard margin

primal form: $\min \frac{1}{2}\|\mathbf{w}\|^2$ subject to $y^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right) \geq 1$ for $m = 1, \ldots, M$

dual form: $\max \mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + \sum_{m=1}^{M}\alpha_m$

subject to $\sum_{m=1}^{M}\alpha_m y^{(m)} = 0$ and $\boldsymbol{\alpha}_m \geq 0$ for $m = 1, \ldots, M$

Soft margin

primal form: $\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{m=1}^{M}\varepsilon^{(m)}$ subject to $y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(m)} + b\right) \geq 1 - \varepsilon^{(m)}$ for $m = 1, \ldots, M$

dual form: $\max \mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + \sum_{m=1}^{M}\alpha_m$

subject to $\sum_{m=1}^{M}\alpha_m y^{(m)} = 0$ and $0 \leq \alpha_m \leq C$ for $m = 1, \ldots, M$

The constraints are different!

The scalar products $\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}$ can be replaced by a kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

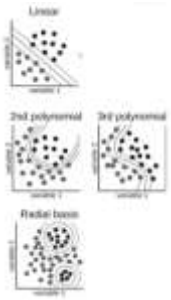
Common Kernel Functions:

Linear: $K(a,b) = a^T \cdot b$

Polynomial: $K(a,b) = (a^T \cdot b + t)^r$
with parameters t, r

RBF = Radial Basis Function:

$K(a,b) = e^{-\gamma\|a-b\|^2}$
with parameter γ



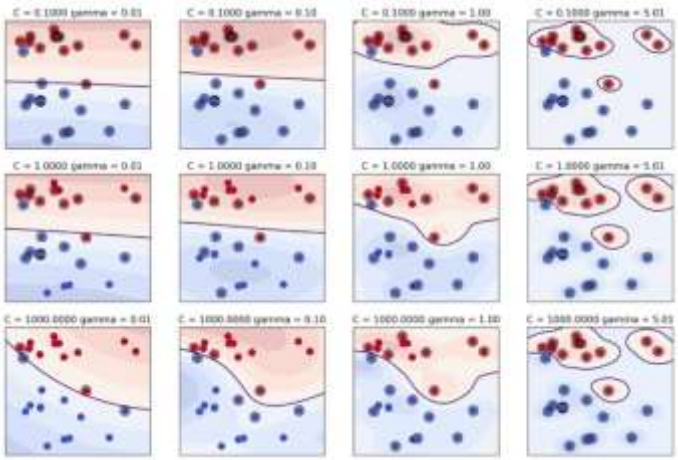
RBF Kernel:

$$K(a,b) = e^{-\gamma\|a-b\|^2}$$

$$= e^{-\frac{\|a-b\|^2}{2\sigma^2}}$$

Small γ = Large width σ

Large γ = Small width σ

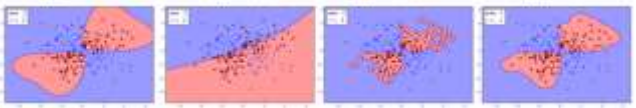


ConcepTest: RBF Kernel:

In the SVM below we are using C = 1. Which image has the highest value for hyperparameter γ?

$$K(\mathbf{x}, \mathbf{x}^{(m)}) = \exp(-\gamma\|\mathbf{x}^{(m)} - \mathbf{x}\|^2)$$

a) γ = 1 b) γ = 0.01 c) γ = 100 d) γ = 10



SVMs vs Logistic Regression:

Both classifiers are based on linear separation

When used as linear classifiers, they perform similarly

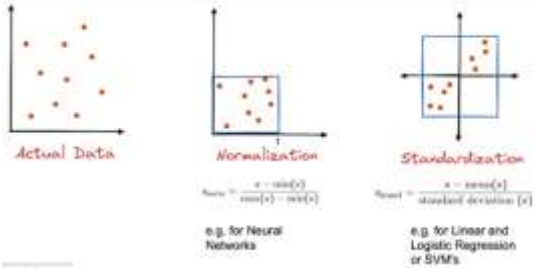
Both can be combined with non-linear features leading to non-linear classifiers

The kernel trick is a good way to construct non-linear features

SVM deals much more efficiently with these features

SVM is a popular choice for non-linear classification

Standardization and Normalization

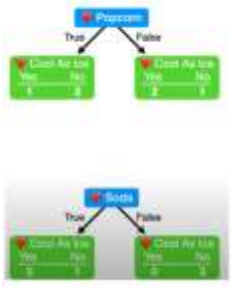


Kernle Trick:

$$\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)^T}x^{(j)} + \sum_{m=1}^{M}\alpha_m$$

$$\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i\alpha_j y^{(i)}y^{(j)}K(x^{(i)}, x^{(j)}) + \sum_{m=1}^{M}\alpha_m$$

Gini Impurity for Yes/No Decisions



Gini Impurity for a Leaf:
1 – (probability of "Yes")² – (probability of "No")²

Popcorn->True: $1-\left(\frac{1}{1+3}\right)^2-\left(\frac{3}{1+3}\right)^2=0.375$

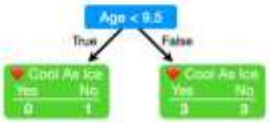
Popcorn->False: $1-\left(\frac{2}{2+1}\right)^2-\left(\frac{2}{2+1}\right)^2=0.444$

Gini Impurity for a Decision:
Weighted average of Gini Impurities of the leaves

Popcorn: $\left(\frac{4}{4+3}\right)0.375+\left(\frac{3}{4+3}\right)0.444=0.405$

Soda: 0.214

Gini Impurity of Numeric Nodes



Gini Impurity of "True"
= 1 – (probability of "Yes")² – (probability of "No")²
$=1-\left(\frac{0}{0+1}\right)^2-\left(\frac{1}{0+1}\right)^2=0$

Gini Impurity of "False"
$=1-\left(\frac{3}{3+3}\right)^2-\left(\frac{3}{3+3}\right)^2=0.5$

Gini Impurity of Decision "Age < 9.5"
= Weighted average of Gini Impurities of the leaves
$=\left(\frac{1}{1+6}\right)0+\left(\frac{6}{1+6}\right)0.5=0.429$

Node Impurity for Classification (Knoten Verunreinigung für Klassifizierung):

Gini impurity for data Q_i at node i:

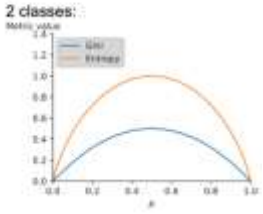
$$G(Q_i)=1-\sum_{k=1}^{K}p_{i,k}^2$$

proportion of class k samples in node i: $p_{i,k}=\frac{1}{N_i}\sum_{y\in Q_i}I(y=k)$

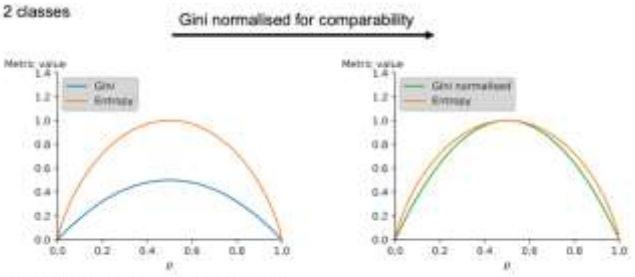
Alternative: Entropy

$$H(Q_i)=-\sum_{k=1}^{K}p_{i,k}\log_2 p_{i,k}$$

2 classes:



Gini Impurity and Entropy Behave Similarly (Gini-Unreinheit und Entropie verhalten sich ähnlich)



Gini impurity increase is slightly steeper

Gini Impurity
Gibt an, wie oft ein zufällig ausgewähltes Element falsch klassifiziert würde, wenn die Klassifizierung nach der Verteilung der Klassen im Datensatz erfolgen würde.

Formel für zwei Klassen:

$$\text{Gini}=1-(p^2+(1-p)^2)$$

wobei p die Wahrscheinlichkeit ist, dass ein Element zur Klasse 1 gehört.

Interpretation:

- Gini = 0: Alle Elemente gehören zur gleichen Klasse (perfekte Reinheit).
- Gini ist maximal (~0.5 für zwei Klassen): Elemente sind gleichmäßig auf die Klassen verteilt (größte Unreinheit).

Encoding von kategorischen Variablen:

1. Label Encoding
   - Jede Kategorie wird einer Ganzzahl zugeordnet (z.B. nach Reihenfolge).
   - **Achtung:** Es entsteht eine künstliche Ordnung, die das Modell falsch interpretieren könnte!
2. Ordinal Encoding
   - Kategorien werden entsprechend einer echten **Reihenfolge oder Hierarchie** codiert (z.B. Ratings: 1–5 Sterne).
   - Hier **macht die Reihenfolge Sinn** und sollte vom Modell beachtet werden.
3. One-hot Encoding
   - Jede Kategorie bekommt eine eigene **binäre Variable** (0 oder 1).
   - Keine künstliche Reihenfolge, aber mehr Variablen entstehen.
   - Typisch bei Kategorien **ohne natürliche Ordnung**.

Regression Tree:
Ein Regressionsbaum ist ein Entscheidungsbaum, der verwendet wird, um Zahlen vorherzusagen (nicht Klassen). Er teilt die Daten Schritt für Schritt auf, sodass am Ende in jedem Blatt ein durchschnittlicher Zahlenwert steht.
Die Aufteilungen werden so gewählt, dass der Fehler (z.B. mittlere Abweichung) möglichst klein wird.

Pruning (Beschneiden von Entscheidungsbäumen):
Pre-Pruning (vor dem Baumaufbau stoppen):

- Min-Sample Pruning: Knoten nur splitten, wenn mindestens k Beispiele vorhanden sind.
- Max-Depth Pruning: Baum darf nur eine bestimmte maximale Tiefe erreichen.

Post-Pruning (nach dem Baumaufbau kürzen):

- Baum vollständig wachsen lassen und danach anhand von Validierungsdaten entscheiden, ob ein Blatt nötig oder überflüssig ist.

Random Forest Überblick
Training: Viele Entscheidungsbäume auf zufälligen Subsets der Daten.
Vorhersage: Neue Daten durch alle Bäume schicken → Mehrheitsentscheid.

Random Feature Selection

- Für jeden Split wird nur ein zufälliges Teilset der Features betrachtet.
- Standard: $\sqrt{N}$ Features (bei Klassifikation).

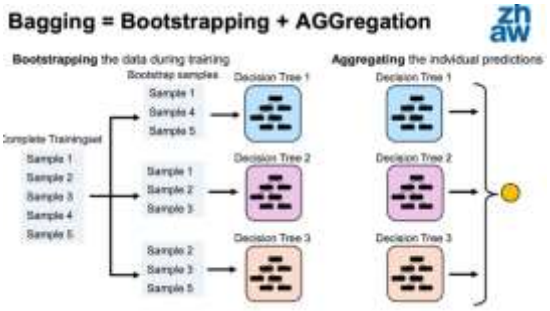
Bagging (Bootstrap Aggregating)
Bootstrap: Ziehe mit Zurücklegen zufällige Stichproben aus den Trainingsdaten.

Hyperparameter von Random Forests:
Anzahl Bäume: Typisch 100–500.
Sample Size: 20–100 % der Trainingsdaten pro Baum.
Features pro Split: Standard $\sqrt{N}$
Maximale Tiefe: Volle Tiefe oder begrenzt (z.B. 3–7).

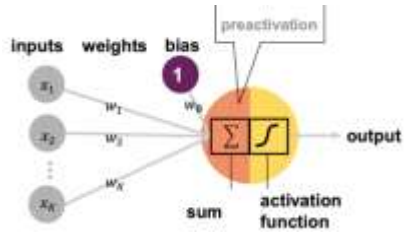
Out-of-Bag (OOB) Error

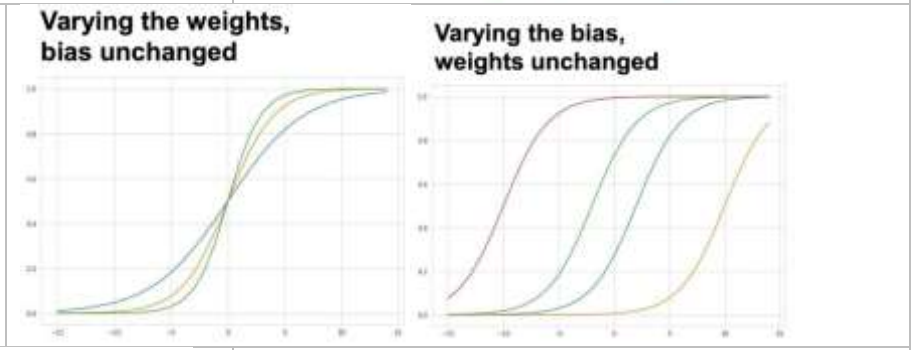
- Jedes Sample wird von den Bäumen bewertet, die es nicht zum Trainieren benutzt haben.
- OOB-Fehler: Anteil der OOB-Samples, die falsch klassifiziert wurden.

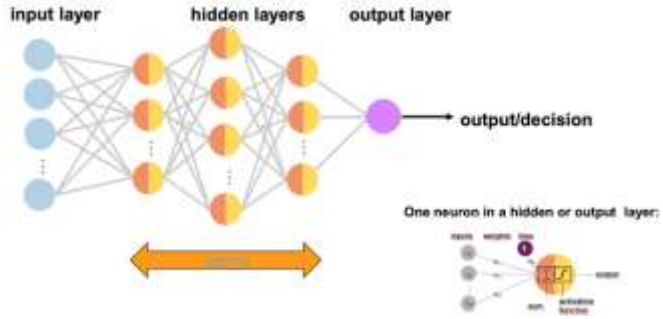
Aggregation: Mittlere oder mehrheitliche Vorhersage aller Modelle.



A Neuron:







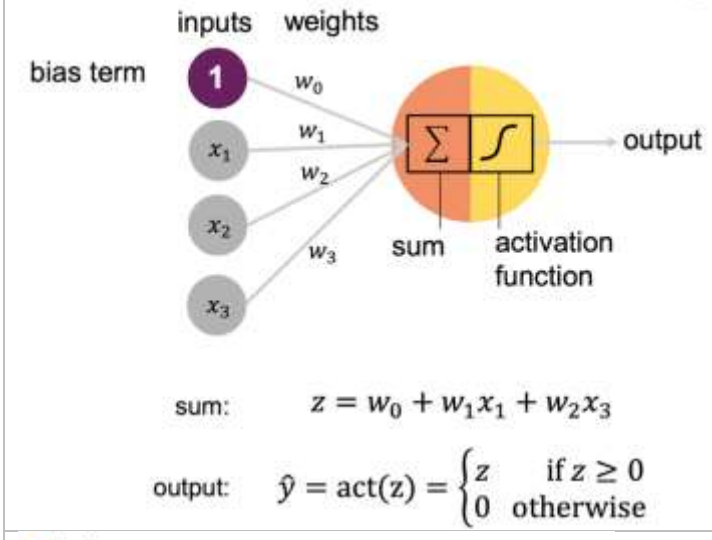
Activation Functions:

LINEAR
like linear regression
(only used in output layer)

LOGISTIC / SIGMOIDAL / TANH
Smooth, differentiable, saturating functions

RECTIFIED LINEAR (ReLU)
Cheap to compute, very popular

Neuron with ReLU activation

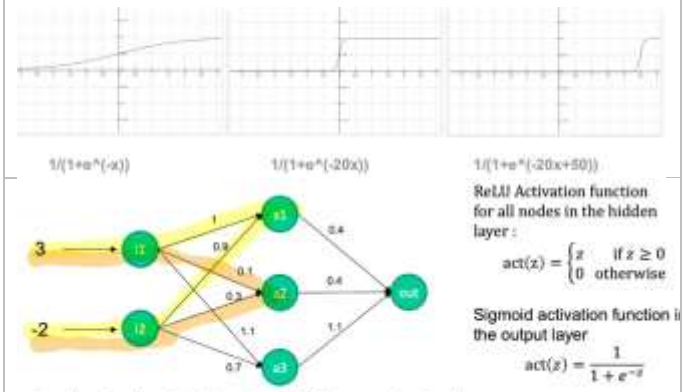


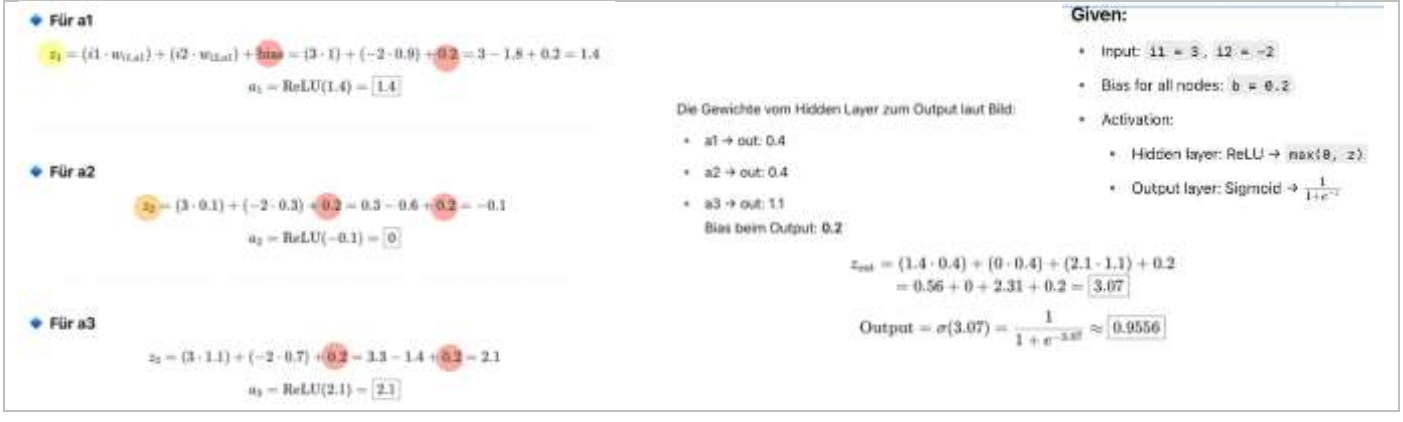
sum: $z = w_0 + w_1 x_1 + w_2 x_3$

output: $\hat{y} = \text{act}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

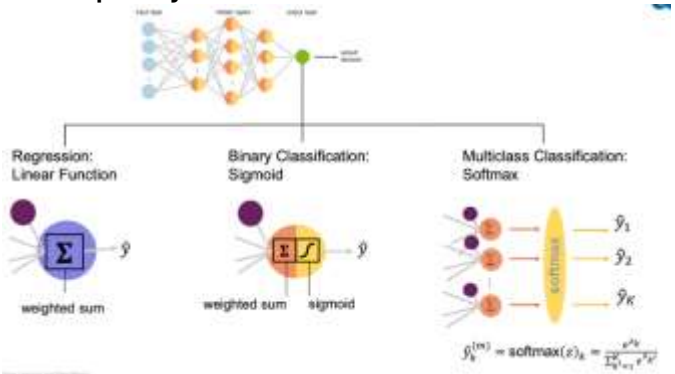
Sigmoid Function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$





The Output Layer:



Regression tasks:

- Output is a single node, the predicted value

Classfication with two classes:

- output a single node with values between 0..1
- decision boundary e.g. at 0.5

Classification with many classes:

- number of output nodes = number of classes
- sum over all output values = 1
- each output value between 0..1
- largest output value is predicted class

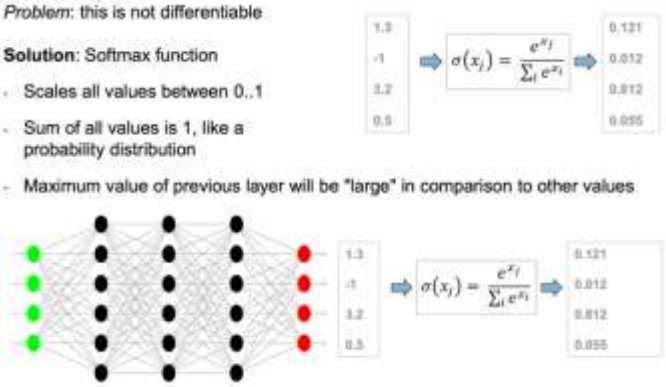
Softmax:

Goal: For classification tasks with mutually exclusive classes, in the output layer we want to have 1.0 for the correct class, and 0 for all other classes

Idea: A "Max-Layer" could set 1 for maximum value of previous layer, and 0 to all other outputs.
Problem: this is not differentiable

Solution: Softmax function

- Scales all values between 0..1
- Sum of all values is 1, like a probability distribution
- Maximum value of previous layer will be "large" in comparison to other values



Goal: For classification tasks with mutually exclusive classes, in the output layer we want to have 1.0 for the correct class, and 0 for all other classes

Logistic Regression

Hypothesis: $h_\theta(x) = g(\theta^T x) \quad \text{where } g(z) = \frac{1}{1+e^{-z}}$

Loss Function "Log-Likelihood":

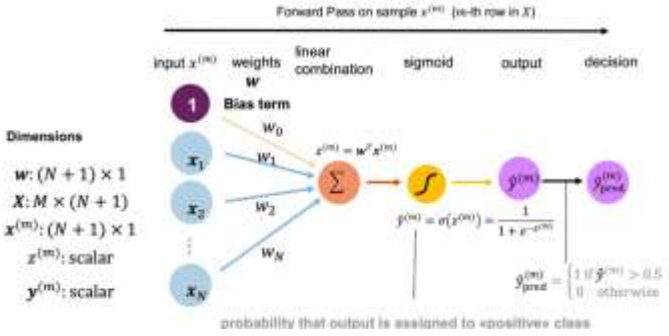
$$Loss(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$

$$= -y \log(h_\theta(x)) - (1-y)\log(1 - h_\theta(x))$$

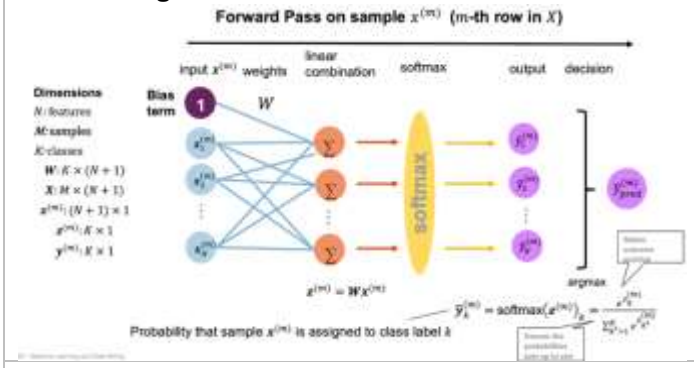
Cost Function

$$J(\theta) = \frac{1}{M} \sum_{m=1}^{M} Loss(h_\theta(x^{(m)}), y^{(m)})$$

Logistic Regression as "Neuron":



Softmax Regressio



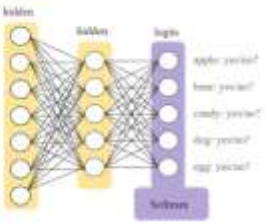
Softmax vs One-vs-rest

One-vs-rest

Each label has its own classifier

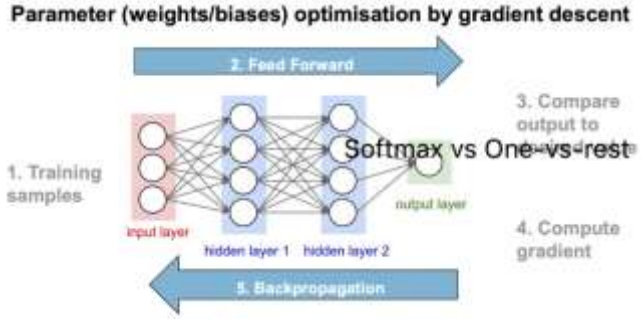
Softmax

One model for all labels



Training in Feedforward Neural Networks

Parameter (weights/biases) optimisation by gradient descent



- **Epoch** = a forward pass and backward pass completed for *all* the training examples
- **Batch size** = the number of training examples in a batch. The bigger the batch size, the more memory space you'll need.
- **Iteration** = forward and backward pass each using a batch with [batch size] number of examples
- **Iterations per Epoch** = #training data / size of batch
- **Optimizer:** (Batch) Gradient Descent
- **Learning Rate Adaption,** e.g. with Momentum, Adagrad, Adadelta, RMSprop, Adam, Nesterov accelerated gradient, AdaMax, Nadam, AMSGrad etc

The Cost Function:

Regression Task: Mean Squared Error $MSE = \frac{1}{M}\sum_{m=1}^{M}(y^{(m)} - \hat{y}^{(m)})^2$

Binary Classification: Logistic Loss

$$L_{logistic} = -\sum_{m=1}^{M}[y^{(m)}log(\hat{y}^{(m)}) + (1-y^{(m)})log(1-\hat{y}^{(m)})]$$

Mulitclass-Classification: Cross Entropy Loss

$$L_{CE} = -\sum_{m=1}^{M}\sum_{k=1}^{K} y_k^{(m)} log(\hat{y}_k^{(m)}) = -\sum_{m=1}^{M} log\frac{\exp(\mathbf{w}_{c_m}^T \mathbf{x}^{(m)})}{\sum_{k=1}^{K}\exp(\mathbf{w}_k^T \mathbf{x}^{(m)})}$$

How many parameters does the neural network have?

- Input: 28x28 greyscale image (784 pixels)
- 2 hidden layers with 16 neurons each
- Output layer with 10 neurons, one for each potential digit
- Training Data: MNIST, 60'000 training and 10'000 for evaluation

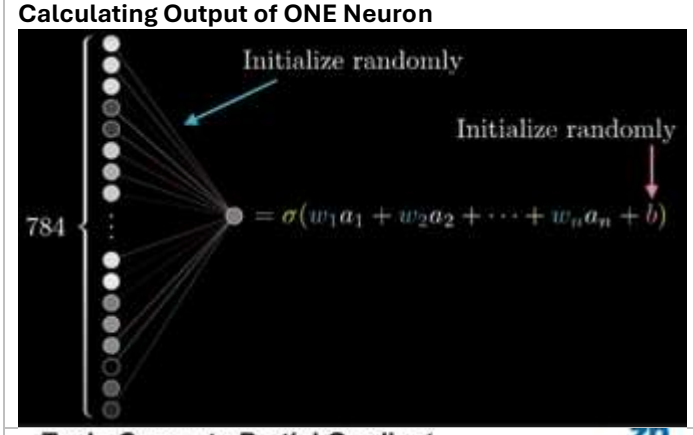
How many parameters does the neural network have?

$784\times16 + 16\times16 + 16\times10$ weights

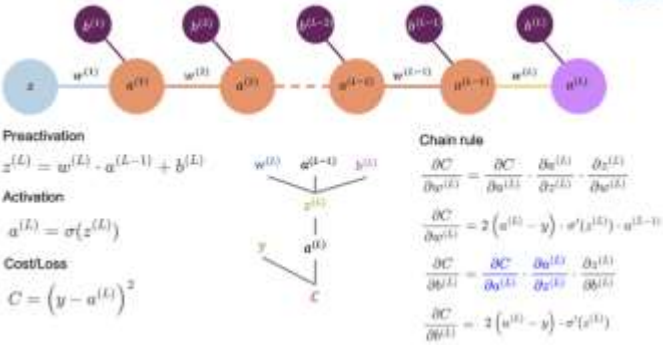
$16 + 16 + 10$ biases

13,002

Calculating Output of ONE Neuron



Backpropagation for a Chain: Last Layer



Preactivation

$z^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)}$

Activation

$a^{(L)} = \sigma(z^{(L)})$

Cost/Loss

$C = \left(y - a^{(L)}\right)^2$

Chain rule

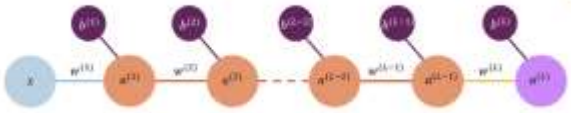
$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

$$\frac{\partial C}{\partial w^{(L)}} = 2\left(a^{(L)} - y\right) \cdot \sigma'(z^{(L)}) \cdot a^{(L-1)}$$

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

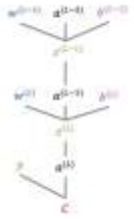
$$\frac{\partial C}{\partial b^{(L)}} = 2\left(a^{(L)} - y\right) \cdot \sigma'(z^{(L)})$$

Task: Compute Partial Gradient



Assume we have the following setting (we skip irrelevant parameters): $a^{(L-2)} = 12$, $w^{(L-1)} = 0.1$, $b^{(L-1)} = -3.5$, $w^{(L)} = 1.3$, $b^{(L)} = 0.2$ with the sigmoid activation function.

1. Compute the output $a^{(L)}$
2. Assume that we would except for $a^{(L)}$ an output value of 1.0. Compute the partial derivative $\frac{\partial C}{\partial w^{(L)}}$



$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

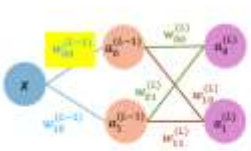
$$\frac{\partial C}{\partial w^{(L-1)}} = \underbrace{2\left[a^{(L)} - y\right]}_{\frac{\partial C}{\partial a^{(L)}}} \underbrace{\sigma'(z^{(L)})}_{\frac{\partial a^{(L)}}{\partial z^{(L)}}} \underbrace{w^{(L)}}_{\frac{\partial z^{(L)}}{\partial a^{(L-1)}}} \underbrace{\sigma'(z^{(L-1)})}_{\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}} \underbrace{a^{(L-1)}}_{\frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}}$$

The **reuse of previous computations** (from later layers when computing the partial derivatives for earlier layers as those marked in blue) **makes backpropagation efficient**.

Multiple Training Samples

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{M}\sum_{m=1}^{M}\frac{\partial C_m}{\partial w^{(L)}}$$

TASK: Backpropagation With More Neurons



Preactivation

$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + b_j^{(L)}$$

Activation

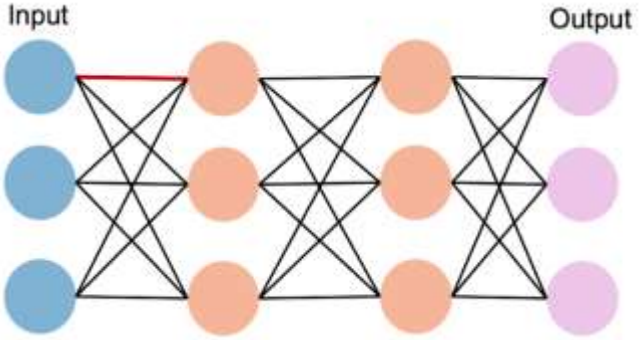
$$a_j^{(L)} = \sigma\left(z_j^{(L)}\right)$$

Cost

$$C = \sum_{j=1}^{n_L}(a_j^{(L)} - y_j)^2$$

Compute the derivatives of the cost function with respect to the yellow-highlighted weight $w_{00}^{(L-1)}$

$$\frac{\partial C}{\partial w_{00}^{(L-1)}} = \cdots$$

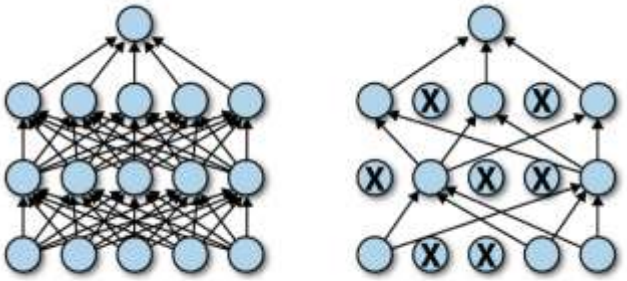


Number of Paths = 9

Compute the derivatives of the cost function with respect to the yellow-highlighted weight

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

Dropout



(a) Standard Neural Net

(b) After applying dropout

Size of Neural Networ

Assume a neural network with *k* hidden layers, where all layers (incl. input and output layer) each have *w* neurons.

In the example: k = 3 hidden layers, width = 4.

1. How many trainable parameters does such a network have?
Number of weights and biases = $w^2 (k+1) + w(k+1)$

2. How many paths do exist that start in any input neuron and end in any output neuron?
Number of paths = w^{k+2}