Einführung

Artificial Intelligence = MachineLearning

Einige Methoden: DecisionTrees, Naive BayesClassification, Random Forest, Reinforcement Learning, Neural-Networks, Support Vector Machine, K-MeansClustering, Principal Component Analysis, Transformers

Free datasets on *kaggle.com*

Data Types (Merkmalstypen, Messniveaus)

Qualitativ (Kategoriell): Ausprägungen
1. **Nominal:** Ungeordnet, Labeling (z.B. Geschlecht)
2. **Ordinal:** Geordnet, diskret (z.B. Rangliste)

Quantitativ (Metrisch, Empirisch): Ausmass in Zahlen
3. **Diskret:** Abzählbar (z.B. Anzahl Kinder)
4. **Stetig:** Reelles Intervall (z.B. Körpergrösse)

Data Cleaning

Low-quality data will lead to low-quality results (garbage in, garbage out)

Detect (near) duplicates:
- Cosine Similarity: Measures similarity between two numerical vectors. Only consideres the angle between, not their lengths.
- Levenstein distance for text

Cosine Similarity

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

Missing values mit Mittelwert/Median berechnen oder Datensatz verwerfen.
Outliers should be removes as they disturb the results.

Machine Learning Paradigmen

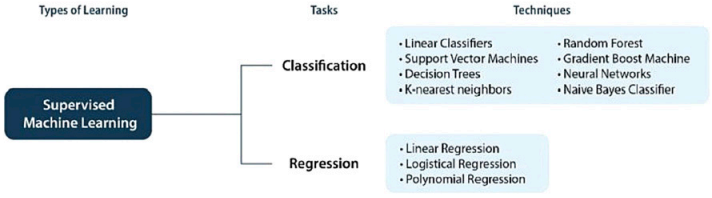
1. Supervised Learning → Viele Daten verfügbar, **labelled data (feature vec)**
The goal is to learn a function that maps input features to output labels.

Feature Vector: Umwandlung von Lables in numerischer Vector.
Represents the relevant properties of the input data.

- Regression: Prediction and forecasting
- Classification: Identifying to which category an observations belongs

Models:

- **Generative models:** Learns the joint probability for each class (probabilty of observing a class, given its features). Focus on understanding how the data is generated. Models: Naive Bayes, Dynamic Causal Modeling (DCM)
- **Discriminative models:** Learn a decision boundary and use it to separate classes. Models: Support Vector Machines, Logistic Regression



2. Unsupervised learning → Wenige Infos über Daten, **unlabelled data**
- Training data does not contain any output values (only input features given)
- The goal is to model the underlying distribution of the data *X*.
- Clustering: Grouping a set of objects (e.g. cars with same color)
- Dimensionality Reduction: Transforming into lower dimenstionality. Advantages: Data compression, better computation times, removas redundatn features, visualize data in 2d or 3d.

3. Reinforcement Learning → Computer spielt gegen sich selbst
- Reqard-System: Agent performs action and receives a reward or punischment from the observer.

Deep Learning: Both supervised and unsupervised learning

Classification Algorithms

Logistic Regression, Decision Tree, Random Forest, Neural Networks, Support Vector Machines, Naive Bayes, K-Nearest Neighbour, etc.
There is no best algorithm → try multiple algos and optimize best.

Quality Measures: Accuracy, Precision, Recall, F1

Lineare Regression

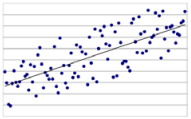
phonosync.github.io/mldm-notes

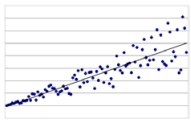
Basic Assumptions in Linear Regression

Requirement for linear regression:
1. **Linearity:** The relationship between x and y is linear
2. **Independence:** No repeating patterns, eg. no seasonality.
3. **Normality:** The expected output values are normally distributed
4. **Homoscedasticity** (equality of variance): The variance is the same for any x

Homoscedasdicity and Heteroscedasticity

Homoskedastizität (gewünscht) Heteroskedastizität (unerwünscht)



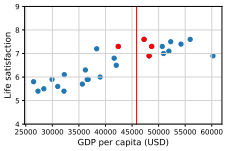


Nearest Neighbour Regression

Calculate the average from the *k* closest neighbours.
Sensitive to outliers and noise.

Instance-based learning (also called lazy-learning, because it defers the generalization step until prediction time).
Makes predictions based on specific examples (instances).

Similarity measures include **euclidean distance** and **cosine similarity**.

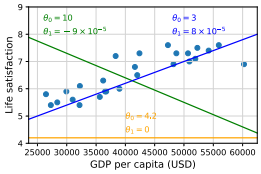


Simple (Univariate) Linear Regression

1 input variable. Model-based learning.

$$\hat{y}^{(m)} = h_{\theta_0,\theta_1}(x^{(m)}) = \theta_0 + \theta_1 x^{(m)}$$

$\hat{y}^{(m)}$ estimation / *hypothesis* function with paramters θ_0 and θ_1 (*x* is input)
$x^{(m)}$ input variable of *m*-th sample
θ_0 (theta$_0$) intercept with *y*-axis ← **trained**
θ_1 (theta$_1$) slope ← **trained**



Training

Loss Function: Measures/compares the predictions with the expected output

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

$\epsilon_m = y^{(m)} - \hat{y}^{(m)}$ is the residual of the *m*-th sample
RSS is the *residual sum of squares* (*sum of residuals*, *sum of squared errors*).

Cost Function (Mean Squared Error): Average error per sample
Asks: How good does the data fit? 2*M* weil es das Ableiten vereinfacht.

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

Closed Form Solution

Kein Training nötig, optimale Lösung durch Formel.

$$\hat{\theta}_1 = \frac{\sum_{m=1}^{M} (x^{(m)} - \mu_x)(y^{(m)} - \mu_y)}{\sum_{m=1}^{M} (x^{(m)} - \mu_x)^2} \qquad \hat{\theta}_0 = \mu_y - \theta_1 \mu_x$$

$$= \frac{S_{xy}}{S_x^2} = \frac{\overline{xy} - \bar{x}\,\bar{y}}{\overline{x^2} - \bar{x}^2} = \frac{\frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{n}}{\frac{\sum (x_i - \overline{x})^2}{n}}$$

where $\mu_x = \frac{1}{M} \sum_{m=1}^{M} x^{(m)}$ and $\mu_y = \frac{1}{M} \sum_{m=1}^{M} y^{(m)}$ are the sample means. $\tilde{s}_{xy}$ is the covariance and $\tilde{s}_x^2$ is the variance of x.

Multivariate (Multiple) Linear Regression

Bei mehrere Variablen (Features).

Normal Equation $X^T X\theta = X^T y$

Anwendbar wenn die Matrix nicht zu gross ist (bis 20'000 features).

Prognostizierte y-Werte (Matrix Form): $y = X\theta$

$$\hat{y}^{(m)} = h_\theta(x^{(m)}) = \theta_0 x_0^{(m)} + \theta_1 x_1^{(m)} + \theta_2 x_2^{(m)} + \ldots + \theta_N x_N^{(m)} = \theta^T X_m$$

Gesuchte Paramter

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

Residual*

$$e = y - X\theta$$

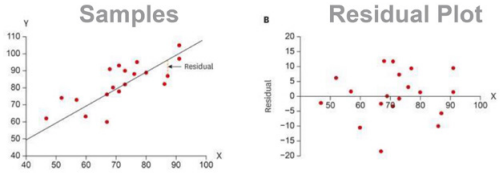
$$\epsilon^{(m)} = y^{(m)} - \hat{y}^{(m)}, \quad m = 1, \ldots, M$$

$$X = \begin{matrix} x_0 & x_1 & \ldots & x_n \end{matrix} \text{ Features}$$
$$\begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & x_{21} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \ldots & x_{in} \end{pmatrix}$$

$$X = \begin{matrix} x_0 & x_1 & x_2 & x_3 & x_4 \end{matrix}$$
$$\begin{pmatrix} 1 & 29932.5 & 4.8 & 70160 & 18 \\ 1 & 31007.8 & 1 & 104458 & 16.4 \\ 1 & 32181.2 & 1 & 232666 & 16.9 \end{pmatrix} \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix} \quad y = \begin{pmatrix} 5.9 \\ 5.6 \\ 5.4 \end{pmatrix}$$

Positives Koeffizienten (Theta) bedeutet einen positiven **linearen Zusammenhang** (zwischen Features und y-Target) und vise versa.

*** Residual-Plot:** Macht vor allem für **höher dimensionale** Daten Sinn. Höhere Dimensionen werden auf eine visuelle 2d Grafik reduziert.



Evaluating (linear) Regression Models

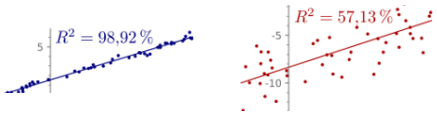
- Mean Absolute Error: $MAE = \frac{\sum_{i=1}^{I} |y^{(i)} - \hat{y}^{(i)}|}{I}$
- Mean Squared Error: $MSE = \frac{\sum_{i=1}^{I} (y^{(i)} - \hat{y}^{(i)})^2}{I}$
- Root Mean Squared Deviation: $RMSD = \sqrt{MSE} = \sqrt{\frac{\sum_{i=1}^{I} (y^{(i)} - \hat{y}^{(i)})^2}{I}}$

I = number of samples, Evalutaion: The lower the result, the better

Coefficient of Determination R^2 [0, 1]
Bestimmtheitsmass R^2

Not to be confused with *Pearson correlation coefficient* r_{xy} [-1, 1] (Korrelationkoeffizient). R^2 stimmt überein mit dem Quadrat des Korrelations-Koeffizienten r_{xy} (nach Bravais-Pearson).

Beschreibt die Gesamtvarianz in den *y*-Daten.
R^2 = Prozentmass. 1 heisst alle Datenpaare liegen auf der Regressionslinie.
0 means the model always predicts the mean (straight line).
The larger R^2, the closer the points scatter to the regression line.



$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \qquad SS_{\text{res}} = \sum_i (y^{(i)} - \hat{y}^{(i)})^2 = \sum_i e^{(i)^2}$$

$$SS_{\text{tot}} = \sum_i (y^{(i)} - \mu_y)^2 \qquad \mu_y \text{ mean of all } y$$

$$R^2 = \frac{(\overline{xy} - \overline{x} \cdot \overline{y})^2}{(\overline{x^2} - \overline{x}^2)(\overline{y^2} - \overline{y}^2)} = r_{xy}{}^2$$

Problem

The Normal Equation method does not work in practice for larger input datasets. As a rule of thumb, the process is **too time-consuming** – exponential for features and linear for samples – or breaks for more than 20'000 features or samples → Ein Lösungsansatz: **Gradient Descent**

Standardization and Normalization

Normalization

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

e.g. for Neural Networks

Standardization

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$

e.g. for Linear and Logistic Regression or SVM's

Gradient Descent

Uses of **gradient (partial derivaties)** to determine the direction of adaption. The goal is to **reduce** the **cost function** J. The **learning rate** α determines how far we step each time (step opposite the direction of the gradient).

Works for any hypothesis function *h* that can be differentiated.

Cost function J (Average error per sample) M samples

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

Gradient (partial derivatives of J)

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

Abbruchkriterien:
- Cost function *J* does not decrease anymore or only slightly
- Gradient is close to 0

Univariate Linear Regression

1. Initialize $\theta_0, \ldots \theta_n$ with approximate values
2. Update simultaneously (use old θ_0 when computing $h_\theta(x^{(m)})$) and repeat until convergence

$$\theta_0 = \theta_0 - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) \qquad \theta_0^{(t+1)} = \theta_0^{(t)} - \alpha(\theta_0^{(t)} + \theta_1^{(t)} x_t - y_t)$$

$$\theta_1 = \theta_1 - \alpha \frac{1}{M} \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)}) x^{(m)} \qquad \theta_1^{(t+1)} = \theta_1^{(t)} - \alpha(\theta_0^{(t)} + \theta_1^{(t)} x_t - y_t) x_t$$

Multivariate (Multiple) Linear Regression

$$\theta = \theta - \alpha \frac{1}{M} \cdot \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)})\, x^{(m)} \qquad x^{(1)} = 1$$

n (θ_n features) · *M* (samples) calculations.

Learning Rate Optimization

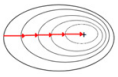
The learining rate should be smaller towards the end.

$$\alpha_t = \frac{\alpha_0}{1 + decay\_rate * t} \qquad t = 1, 2, \ldots \qquad \alpha_0 = 0.1$$

Flavors of Gradient Descent

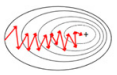
Batch Gradient Descent (BGD)
Disadvantage: Memory usage, needs to fit all samples into memory at once.

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



Stochastic Gradient Descent (SGD)
Randomly shuffles the examples.
Advantage: Usually faster close to the mininum than BGD.
Disadvantage: Convergence is harder to detect due to erro fluctuate heavily.

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



Mini-batch Gradient Descent (MBGD)
Combination of the two. Uses a fixed batch size of samples in each step.
Advantage: size of batches (samples) can be adapted to available memory.

Batch size: Number of training samples in a batch.
Iteration: Completing one batch.
Epoch: Completing the complete training set.

Gradient descent Type	Disadvantages	Advantages
Batch gradient descent Computes the gradient of the cost function w.r.t. to the parameters for the entire training dataset	- As we need to calculate the gradients for the whole dataset, batch gradient descent is slow - Is intractable for datasets that don't fit in memory. - Doesn't allow us to update our model online, i.e. with new examples on-the-fly. - Performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update.	- Guaranteed to converge to the global minimum for convex error surfaces - and to a local minimum for non-convex surfaces.
Stochastic gradient descent Performs a parameter update for each training example	- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily - SGD's fluctuation, enables it to jump to new and potentially better local minima. This ultimately complicates convergence to the exact minimum, as SGD will keep overshooting	- Is much faster and can also be used to learn online. - when we slowly decrease the learning rate, SGD shows the same convergence behavior as batch gradient descent
Mini-batch gradient descent Performs an update for every mini-batch of n training examples		- Reduces the variance of the parameter updates, which can lead to more stable convergence - 'Mini-batch gradient descent is typically the algorithm of choice when training a neural network

Polynomial Regression

For non-linear relationships.

Polynomial Model

We first define artificial variables $z_1 = x, z_2 = x^2$, and $z_3 = x^3$

$$h_\theta(z) = \theta_0 z_0 + \theta_1 z_1 + \theta_2 z_2 + \theta_3 z_3$$
$$y = \theta_0 \quad + \theta_1 x + \theta_2 x^2 + \cdots + \theta_m x^m$$

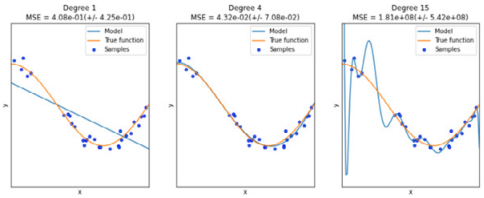
Vandermonde Matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \quad x_1, \ldots, x_n = \text{Features}$$

This is a multivariate linear regression model. We can apply our methods from before - e.g. normal equation, gradient descent etc.

Underfitting and Overfitting

Overfitting: Model (line) fits (almost) perfectly to the given training data, but does not work well for new (unseen) data. **Does not generalize well.**
Underfitting: Model lacks sufficient power or complexity



Regularization

The smaller the absolute values of parameters θ, the less extreme the curve. Regularization aims to ensure that the values of the parameters θ do not become too large. Prevent overfitting and better numerical stability.

Cost function J with new **regularization term**

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

λ = Regularization parameter (scalar).
We cannot know beforehand which values are the best

→ **Hyperparameter optimization:** Train a model for each of these values and then evaluate the quality using R^2.

Hyperparameter Optimization (Tuning)

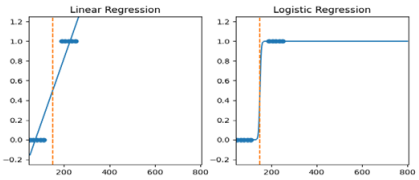
Algorithm	Hyperparameter	Parameters
Nearest Neighbor Regression	- Number of neighbors k	- None
Polynomial Regression	- Polynomials of the input variables - Regularization parameter λ - Learning Rate α	- Values of $\theta_1 \ldots \theta_n$
Logistic Regression	- Regularization parameter λ - Learning Rate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

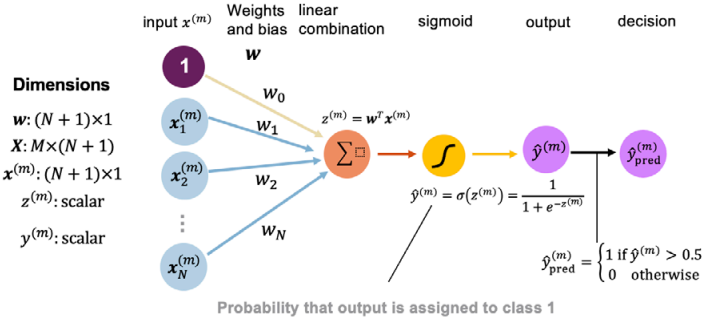
Methods

- **Grid Search:** search all combinations of hyperparameters in specified ranges (e.g. with GridSearchCV from sklearn). Example:
  degree of polynomial $\in \{3, 5, 8\}$, $\lambda \in \{10, 1, 0.1, 0.01\}$
  and $\alpha \in \{1, 0.3, 0.1, 0.03, 0.001\}$)
  → 3 · 4 · 5 = 30 models to train
- **Randomized Search:** train models on random hyperparameter combinations (e.g. with RandomSearchCV)
- **Genetic Algorithms:** using selection, crossover, and mutation to improve populations of hyperparameter settings over time

Logistic Regression Classification

Goal: To learn the optimal boundary between classes (Classification).





Outputs **probabilistic prediction.** Non-linear regression model.
One of the simplest (and oldest) algorithms for binary classification.

Unidimensional

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

Multidimensional

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

$$\boldsymbol{\theta}^T \boldsymbol{x} = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n \qquad x^{(1)} = 1$$

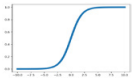
Binary Classification

When there are only two classes to distinguish for $y^{(1)}, \ldots, y^{(M)} \in \{0, 1\}$ where 0 and 1 can represent any class, for instance:
input x: cholesterol levels → output y: heart attack in 5 yrs {no, yes}

Logistic Sigmoid (Step Function) (0, 1)

Reason against hard-step function ⎍ : Differentiability

$$g(z) = \frac{1}{1 + e^{-z}}$$



This function is differentiable, monotonic (it grows larger) and bounded between 0 and 1.

Can be interpreted as how «**confident**» the model is about its prediction:
1: Very confident, **0.5:** Not confident, **0:** Very confident it's the other class.
e is weight, large → steeper. z is bias.

$$g'(z) = g(z) * (1 - g(z))$$

Training

There is no Normal Equation to find the optimal parameters θ (closed form). We cannot apply Mean Squared Error (MSE) loss function, because it assumes a linear relationship. Log-Loss is an **probabilistic** interpretation.

Cost Function $J(\theta)$

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

$$\text{Log-Loss}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if} \quad y = 1 \\ -\log(1 - h_\theta(x)) & \text{if} \quad y = 0 \end{cases}$$
$$= -y \log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x))$$
$$= -y \log(p) - (1 - y)\log(1 - p) \qquad p\text{: Probability of } y = 1$$

Gradient Descent

(Equivalent to linear models)

1. Initialize $\theta_0, \ldots \theta_n$ with approximate or random values
2. Update simultaneously (use old θ_0 when computing $h_\theta(x^{(m)})$) and repeat until convergence

$$\theta = \theta - \alpha \frac{1}{M} \cdot \sum_{m=1}^{M} (h_\theta(x^{(m)}) - y^{(m)})\, x^{(m)} \qquad x^{(1)} = 1$$

Accuracy

Proportion of correctly classified examples. For **classification problems,** we care about whether the model predicted the correct class.

We sum up 1s if the predicted class is correct (above τ), 0 otherwise:

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

Hyperparameters Optimization

Hyperparameters are:

- Learning rate α
- Number of gradient descent iterations T
- Threshold τ (positive example if > τ)

All affect the final values of θ (parameters of the model).

Howto: Split dataset into three disjoint (non-overlapping) parts:

```
|------------ Training 80% ------------|- Val. 10% -|- Test 10% -|
```

1. Training Set
Use fixed value of hyperparameters ($\alpha = 0.1$)
Used to optimize the model's parameters θ.

2. Validation Set (Development Set)
Used for **Hyperparameters Optimization** and to measure how good (accuracy) the model for the particular choice of hyperparameters is and then pick the best one.

3. Testing
Used to estimate how good your model is on unseen data.

```
bestAccuracy = -1
for alpha in [ 0.0001, 0.001, 0.01 ]:
  # 1. Train with current hyperparameter alpha
  model = train(dataTrain, alpha)
  # 2. Evaluate on validation data
  accuracy = validate(dataValid, model) # 0 or 1
  # Keep track of best alpha
  if accuracy > bestAccuracy:
    bestAlpha = alpha
    bestAccuracy = accuracy
# 3. Test
bestModel = train(dataTest, bestAlpha)
accuracy = validate(dataTest, bestModel)
```

Multi-Class Classification

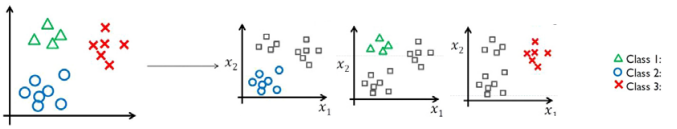
$y \in \{$ smile, frown, neutral, sad, ... $\}$.

Classes are **mutually exclusive.** Each input x belongs to exactly one class.

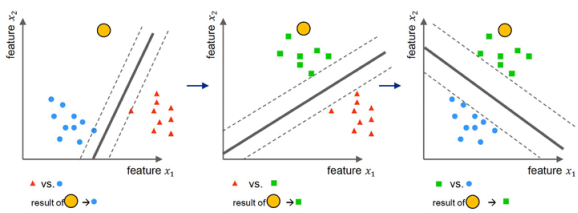
Cannot use logistic regression because that model is for binary classification.

Strategies:
1. One-vs-rest: Decompose a multi-class problem into different bin. classifi.
2. Multi-Class Classification: Softmax Regressino or Neural Networks

One-vs-Rest (One-vs-All):
Man wählt die Klasse mit der **höchsten probability.**



One-vs-One:
Classify unknown instance by majority vote among all pairwise comparisons.
k(*k*–1)/2 pairwise comparisons with *k* classes.



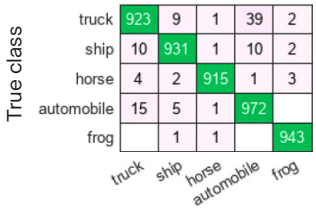
Supervised Learning

Lifecycle

Raw Data → Preprocess → Features → Train → Evaluate → Optimize → Train ...

Classification Quality Measures

Confusion Matrix



For **multiclass** (>2) problems. Describes the models accuracy.

Accuracy: Proportion of examples that are correctly classified.

Prescicion: Of all the positive predictions made by the model, how many are actually correct? How much we can trust the classifier. High precision means a classification has a high probablity, but the model might have many misses. Detect **mostly correct,** but migh have **many misses.** Example: Spam detection or Medical diagnosis (avoid diagnosing a healthy person).

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$= \frac{|\text{PredictedPositive} \cap \text{ActualPositive}|}{|\text{PredictedPositive}|}$$

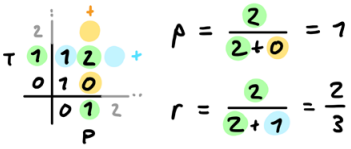
Recall: Of all the actual positive cases, how many were correctly identified? How often the classifier correctly classifies. High recall means many correct classifications but maybe many false positive. **Detect most,** but also **many wrong.** Example: Cancer detection (ensure most cancer cases are detected).

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

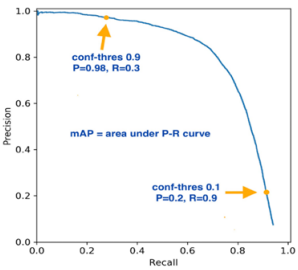
$$= \frac{|\text{PredictedPositive} \cap \text{ActualPositive}|}{|\text{ActualPositive}|}$$

True Positives: The value in the *confusion matrix* where the predicted matches the actual class.
False Positives: The sum of all predictions for *class_of_interest* that are incorrect.
False Negatives: The sum of all actual *class_of_interest* instances that were predicted wrongly.

Beispiel:



Precision-Recall Curve



F1-Score [0, 1]

Expresses how good a classifier is.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

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

β = Weight of precision score.

For **multiclass** we use the average of each class:

$$F_1 = \frac{F_{C_1} + F_{C_2} + ... + F_{C_n}}{n}$$

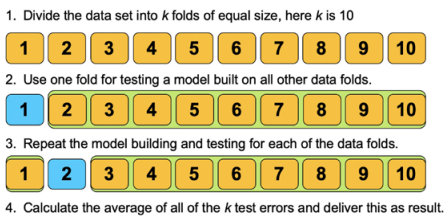
$\text{Accuracy}_{\text{Class A}} > \text{Accuracy}_{\text{Class B}}$ does not imply $\text{F1}_{\text{Class A}} > \text{F1}_{\text{Class b}}$ as they measure different aspects.

Data Partitioning

Two strategies:
1. Fixed data splits (for larger datasets, >= 1000, 80% training, 20% testing)
2. Cross validation (for smaller datasets).

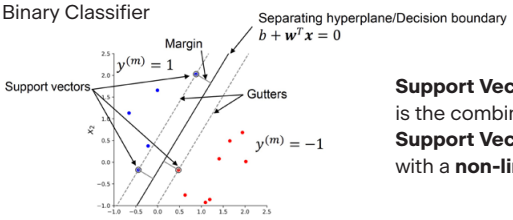
k-Fold Cross Validation

For **smaller datasets,** a single test set may not give a reliable accuracy estimation. Instead, we can «rotate» which data are used for training vs. testing. Score (F1) is **averaged** over all these runs for all data folds.



Support Vector Machines

Binary Classifier



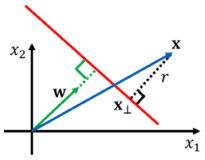
Support Vector Machine is the combination of **Support Vector Classifier** with a **non-linear kernel**.

In N dimensional space, a **hyperplane** is a flat affine* subspace of dimensions $N - 1$. In three dimensions, a hyperplane is a plane, a flat two-dimensional subspace.

* Affine: Subspace, that does not need to pass through the origin

Hyperplane Equation for N dimensions:

$$f(\mathbf{x}) = b + \mathbf{w}^T\mathbf{x}$$
$$b + w_1x_1 + w_2x_2 + \cdots + w_Nx_N$$



Returns **signed distance** of the sample $\mathbf{x}$ from the hyperplane.
$f(\mathbf{x}) = 0$ means sample $\mathbf{x}$ lies exactly on the hyperplane.

$\mathbf{w}$ is the normal vector (perpendicular) to the descition boundry = margin
Goal is to minimize $\mathbf{w}$.

Classification Using a Separating Hyperplane

M training samples in N-dimensional space. $\mathbf{x}$ are our input features:

$$\mathbf{x}^{(1)} = \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_N^{(1)} \end{pmatrix}, \ldots, \begin{pmatrix} x_1^{(M)} \\ \vdots \\ x_N^{(M)} \end{pmatrix}$$

Class labels $y^{(1)}, y^{(2)}, \ldots, y^{(M)} \in \{-1, +1\}$

b takes the role of θ_0 (to be trained)
$\mathbf{w}$ takes the role of $(\theta_1 \ldots \theta_n)^T$ (to be trained)

The hyperplane divides the space is two halves:

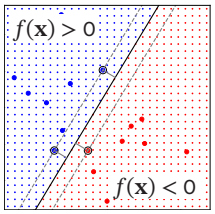
$$f(\mathbf{x}) = b + \mathbf{w}^T\mathbf{x}^{(m)} = 0$$ **Descision boundary:** Sample $\mathbf{x}$ lies exactly on the hyperplane.

$$= -1$$
$$= 1$$ } **Support vectors,** Margin boundaries (gutter)

A separating hyperplane has the property that $y^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right) > 0$

Maximal Margin Classifier

There will exist an infinite number of such hyper-planes → Maximal margin hyperplane:
The hyperplane that is farthest (maximum margin) from the training samples. There are always **at least 2 support vector** (black circles).
They support the maximal margin. hyperplane.
The separating line is calles **descision boundary.**



We want to maximise r (distance between sample and hyperplane):

$$r^{(m)} = y^{(m)}\left(\frac{1}{||\mathbf{w}||}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right)\right) \qquad r = \frac{f(\mathbf{x})}{||\mathbf{w}||}$$

$r_{\min} = \frac{1}{||\mathbf{w}||}$ Distance of all support vectors.

The **distance** between the two margin bounderies (gutter) is given by: $\frac{2}{||\mathbf{w}||} = 2\,r_{\min}$

Objective is to evaluate the minimal value

$$\min_{b,\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 = \max_{b,\mathbf{w}}\left\{\frac{1}{||\mathbf{w}||}\min_{m=1}^{M}\left[y^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right)\right]\right\}$$

so, dass (subject to / Nebenbedingung)

$$y^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right) \geq 1 \;\forall\; m = 1, \ldots, M$$

The factor 1/2 and power of 2 is added for mathematical convenience.

Solve for $\mathbf{w}$ and b

1. Optimise the Langrangian Multiplier
The Lagrange mutiplier is a strategy for finding local maxima and minima of a function subject to equality constraints.
https://www.youtube.com/watch?v=6-ntMlaJpm0

Primal form (This form is not used for optimization)

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) := \frac{1}{2}||\mathbf{w}||^2 - \sum_m^M \alpha_m\left(y^{(m)}\left(\mathbf{w}^T\mathbf{x}^{(m)} + b\right) - 1\right), \qquad \alpha_m \geq 0$$

Bedingung 1: Wird minimiert, Bed. 2: geht gegen 1, geht insgesamt gegen 0

Find extrema of derivates (=0):

$$\frac{\partial\mathcal{L}(\mathbf{w}, b)}{\partial\mathbf{w}} = \mathbf{w} - \sum_{m=1}^{M}\alpha_m y^{(m)}\mathbf{x}^{(m)} = 0 \qquad \Rightarrow \mathbf{w} = \sum_{m=1}^{M}\alpha_m y^{(m)}\mathbf{x}^{(m)}$$

$$\frac{\partial\mathcal{L}(\mathbf{w}, b)}{\partial b} = -\sum_{m=1}^{M}\alpha_m y^{(m)} = 0$$

Dual form (This form is used for optimization)
Terms are merged back into Langrangian Multiplier and simplified.

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

$\Rightarrow \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha})$ subject to $\sum_{m=1}^{M}\alpha_m\, y^{(m)} = 0$ and $\alpha_m \geq 0$ for $\mathrm{m} = 1..M$

Find values for α_m that maximise this expression. Can be solved numerically as a quadratic optimization problem. $y^{(i)}$ and $y^{(j)}$ are 1 or -1 (class).

2. Compute $\mathbf{w}$

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

3. Compute b
Only support vectors are used (M_S).

$$\hat{b} = \frac{1}{M_s}\sum_{m=1}^{M_s}\left(y^{(m)} - \hat{\mathbf{w}}^T\mathbf{x}^{(m)}\right) \qquad \hat{\alpha}_m > 0$$

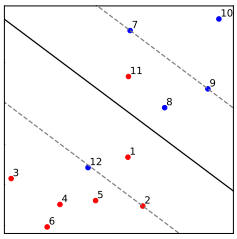
Predictions can then be made using:

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

Problems with maximal margin classifier:
a. Can only be used if the classes are exactly separable (in a straight line).
b. Extremely sensitive to a change in a single sample.
→ Soft Margin Classifier

Soft Margin Classifier (Support Vector Classifier)

The margin is called soft because it is allowed to be violated by some of the training samples.
The **decision boundary** can be non-linear when combined with a **non-linear kernel.**



Slack Variables

Used to **avoid overfitting.** Allow a few individual samples to be on the wrong side of the margin.

$$\epsilon = (\epsilon_1 \quad \epsilon_2 \quad \ldots \quad \epsilon_M)^T$$

The larger their value, the further away from the margin $\epsilon_m > 0$ or the wrong side of the hyperplane (misclassified sample) $\epsilon_m > 1$.
All samples within the margin are support vectors.

Objective

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

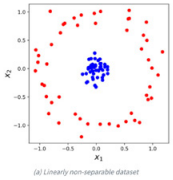
$$\text{subject to } \epsilon_m \geq 0, \quad y^{(m)}\left(b + \mathbf{w}^T\mathbf{x}^{(m)}\right) \geq 1 - \epsilon_m$$

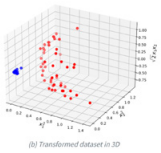
C is nonegative and acts as a **regularisation parameter.**
Controls how many points are allowed to violate the margin constraint.

Bias-variance trade-off: When C is **small**, the margin is wide (many samples may violate the margin) → **low variance, high bias**

Kernel-Trick

For non-linearly separable datasets. Solution: Original space is transformed into a higher dimensionalty, where the examples become linearly separable.
Problem: Hard to know which transformations will work for a given dataset.





Example transformation:

$$\phi : \mathbf{x} \to \phi(\mathbf{x})$$

$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

Kernel Functions

$\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$ (scalar products) can be replaced by a kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. Kernels allows to efficiently work in higher-dimensional spaces without going through the transformations explicitly.

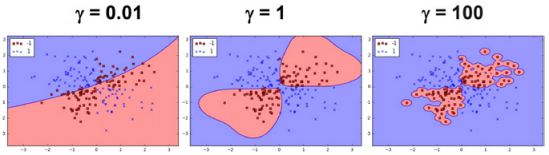
Polynomial Kernel
Hyperparameters d and C.

$$\mathcal{K}(\mathbf{x}^{(m)}, \mathbf{x}^{(m')}) = \left(1 + \sum_{n=1}^{N} x_n^{(m)} x_n^{(m')}\right)^d$$

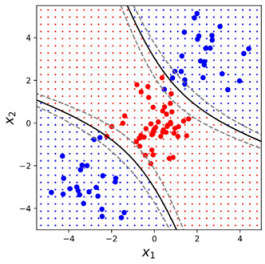
Radial Basis Function Kernel (RBF)
(also called Gaussian kernel)
Hyperparameters γ, d and C.

$$\mathcal{K}\left(\mathbf{x}^{(m)}, \mathbf{x}^{(m')}\right) = \exp\left(-\gamma||\mathbf{x}^{(m)} - \mathbf{x}^{(m')}||^2\right) \qquad \gamma = \frac{1}{\sigma^2}$$

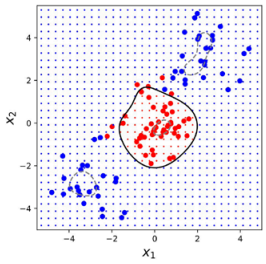
Larger γ (smaller variance σ²): **Smaller spread,** rougher decision boundary typically, low bias, but high variance (small change in data has large effect).

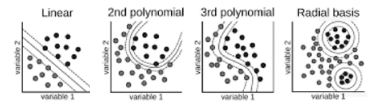


SVM with a **polynomial kernel** of degree 2.



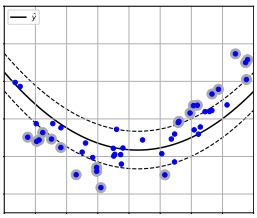
SVM with a **RBF-kernel**.





SVMs for Regression

Tries to fit as many instances as possible on the street while limiting margin violations. The width of the street is controlled by a hyperparameter ε. Reducing ε increases the number of support vectors. It will not affect the model's predictions, the model is ε-insensitive.



SVMs vs Logistic Regression

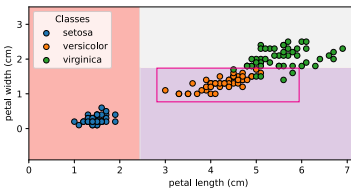
- Both are binary classifiers and are based on linear separation.
- When used as linear classifiers, they perform similarly.
- Both can be combined with non-linear features (leading to non-linear classifiers).
- **SVM deals much more efficiently with non-linear features.**

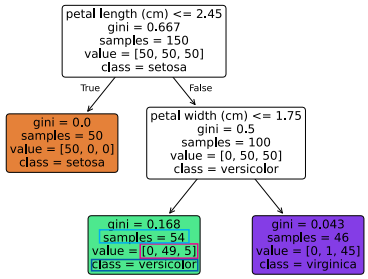
SVM vs Random Forest

- SVM: binary classifier (only 2 classes)
- SVM: data needs standadization
+ RF: regression and multiclass-classification
+ RF: parallel processing
- RF: black box algorithm

Decision and Regression Trees

Can perform both **classification** (multiclass) and **regression** tasks.





Value: How many training instances of each class are in this node.

Probability that a sample belongs to class k.
1. Find leave node of this sample. Let's say node is in class=versicolor
2. Take ratio between samples of class k and training samples. E.g. 49/54

Impurity Measures

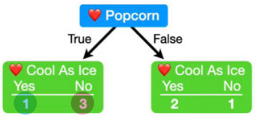
Gini Impurity G [0, 0.5]
gini = 0: **Pure**, all training instances it applies to belong to the same class, node perfectly predict the lable.

$$G(Q_i) = 1 - \sum_{k=1}^{K} p_{i,k}^2 \qquad p_{ik} = \frac{1}{M_i} \sum_{y \in Q_i} I(y = k)$$

Q_i data at node i.
M_i samples

p_{ik} = Proportion of k observations in node i.
Predicted probability for the region of the feature space.

Examples:
$G(Q_i) = 1 - P(class=1)^2 - P(class=2)^2 - ... - P(class=N)^2$
$G(Q_i) = 1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$



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

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

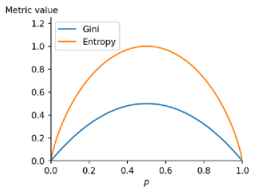
Gini Impurity for a Decision: Weighted average of Ginis of the leaves:

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

Entropy H [0, 1]
Both work, but Gini is slightly faster to compute.

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

Maximum entropy when the classes have the same number of samples.



Selecting the Root Node (First Descision)

The decision with the lowest Gini Impurity

Training a Decision Tree for Classification

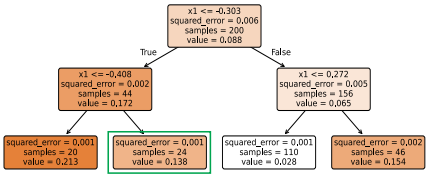
Classification and Regression Tree Algorithm (CART)

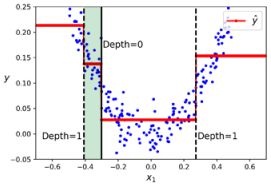
Produces binary trees. Nodes always have exactly two children. Questions only have True/False answers. Can handle both numerical and categorical features. Greedy algorithm: Searches for an optimum split at each level. No guarantee that the solution is optimal.

The algorithm starts at depth=0 with a root node $Q_{i=0}$ where the training set is recursively into two subsets using a single feature n and a threshold $t_{i=0}$ (e.g., petal length ≤ 2.45 cm). In each step the algorithm searches for the pair (n, t_i) that produces the **purest subsets, weighted by their size.**

Regression Trees

Decision trees can also be used for **regression tasks,** i.e. the target is a continuous value. The main difference to a classification tree is that instead of predicting a class in each node, it **predicts a numerical value.**





Prediciton = **Average** value of the training samples in each class.

Training Regression Trees

To train a decision tree for a **regression task.** Instead of the Gini impurity or entropy, the **Mean Squared Error (MSE)** is used. The predicted value of leaf nodes is set to the learned mean value y_i of the node.

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

Regularization

Avoid overfitting. Decision trees are nonparametric model because the number of parameters is not determined prior to training, so the model structure is free to stick closely to the data. This increases the risk of overfitting.

Pre-Pruning (Hyperparameters):
Increasing *min_** hyperparameters or reducing *max_** hyperparameters will regularize the model:

max_depth: The maximum depth of the tree.
min_samples_split: Minimum number of samples a node must have before it can be split.
min_samples_leaf: Minimum number of samples a leaf node must have to be created.
max_leaf_nodes: Maximum number of leaf nodes.
max_features: Number of features to consider when looking for the best split.

Post-Pruning:
Use validation data to decide for each leaf whether it is reasonable.
Leafs werden ovnunten nach oben abgeschnitten.

White vs. Black Box Algorithms

White Box: Decisions are easy to interpret. Examples: Decision Trees

Black Box: It is usually hard to explain why the predictions were made. Examples: Random Forests and Neural Networks.

This is important in many domains, for example, to ensure the system does not make unfair decisions.

Random Forests

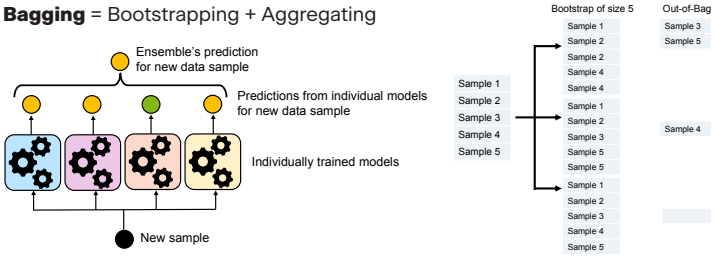
Composition of indvidual decision trees **(ensemble methods)**.
Often leads to better predictions than from the best individual model alone.
Decision Trees have the tendency to overfit the data.

Bootstrapping: Each tree is trained on a different subset randomly sampled from the complete training (same samples may appear in the subsets).
Each tree use random subset of all features.

Aggregating: The predictions of all trees are then aggregated. Aggregation reduces both bias and variance.

- For **regression** task: **Averaging** the predicted values.
- For **classification** task: **Hard** (Majority vote) or **soft voting.**

Bagging = Bootstrapping + Aggregating



Out-of-Bag samples can be used to evaluate a bagging ensemble, without the need for a separate validation set.

Out-of-Bag Error: Proportion of all incorrectly classfied Out-of-Bag samples

Encoding (Output)

Lables: Category is converted to an integer (e.g. Romance → 2)
Ordinal: Values based on order or hierarchy are converted to integer.
One-hot: For every category a binary value is created.
Thriller → 0 0 1, Romance → 0 1 0, Action → 1 0 0

Summary

Advantages
- Easy to understand and visualize
- Works well with mixed input feature types (categorical, continuous)
- No feature normalization (scaling) needed
- Can address (multiclass) classification and regression tasks

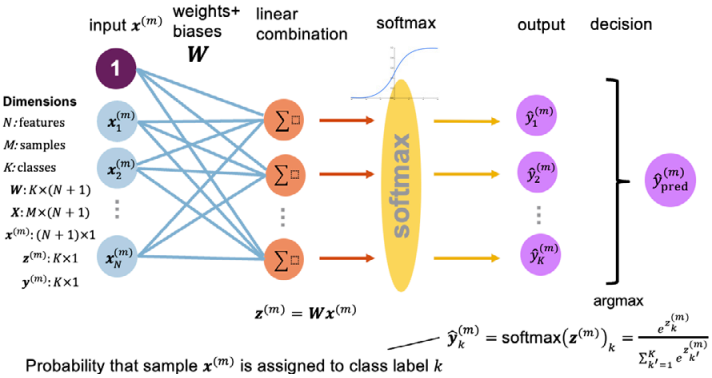
Disadvantages
- Easily overfit the data
- Not robust, small changes in the training set can result in different tree

Neural Networks

A neural network is just a (parameterisierbare) function that maps an input to output. **Depth:** Number of hidden layers. **Deep:** More than 1 hidden layer.

Softmax Regression

Generalizing Logistic Regression to **Multiclass Classification**.



For each class a separate linear combination (sum [-∞, ∞]) of the inputs is computed and then the **softmax** applied. ŷ is a probability.

$$softmax(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)} \quad \rightarrow [0, 1], \text{ so that } \Sigma \hat{y}_k = 1$$

The decision boundaries of softmax regression are **linear.**
A neural network model is **non-linear only** when it has **at least one hidden layer with non-linear activation function.** The predicted label is the class with the highest value/probability (argmax). Softmax Regression has no hidden layers. Softmax is **differentiable,** this allows for backpropation.

Cross Entropy Loss (Cost Function)

Quantifies how good the prediction is.

$$\text{Loss}(\hat{\mathbf{y}}^{(m)}, \mathbf{y}^{(m)}) = -\sum_{k=1}^{K} y_k^{(m)} \log(\hat{y}_k^{(m)}) \qquad J(\mathbf{W}) = \frac{1}{M} \sum_{m=1}^{M} \text{Loss}(\hat{\mathbf{y}}_m, \mathbf{y}_m)$$

y is the true label, ŷ is the predicted label (classification problems). The cross entropy cost function is minimized by gradient descent. The loss quantifies how much the predicted value differs from the true observed value.

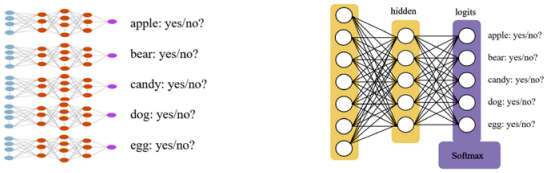
Rule of thumb: Use **Mean Squared Error (MSE)** for **regression,** and **Cross-Entropy** for **multiclass**-classification problems.

One-hot Encoding

For class labels. If we have three classes {1, 2, 3}, and the first sample belongs to class 2 then its label will be $y^{(1)}$ = [0, 1, 0].

Softmax vs One-vs-Rest

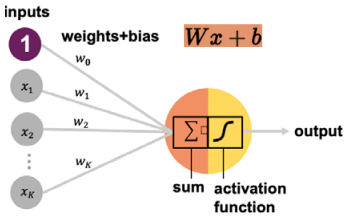
One-vs-rest: Each label has its own classifier. (left)
Softmax: One model for all labels. (right)



Perceptron

Linear Classifier.
Multiple inputs and one output.

1) **Preactivation:** The inputs (including an extra bias term w_0, the offset) are weighted and then summed.
2) An activation function on this linear combination is applied.



Biases: Allow the model to shift the activation function, improving flexibility.

Activation Functions

The nodes in a neural network can have different activation functions.
Linear activation function can only solve linear classification problems.

$$relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \qquad \frac{\mathrm{d}}{\mathrm{d}x} relu(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

$$sigmoid'(x) = sigmoid(x)\big(1 - sigmoid(x)\big)$$

Which Activation Function on the output layer?
Regression → Linear Function
Binary Classification → Sigmoid
Multiclass Classification → Softmax

Training

Vorteil von NN ist, dass die Daten nicht annotiert werden müssen.

Goal is to optimize the model parameters **(weights and biases)** that minimize the cost function. This minimization is computed using gradient descent.

1. Initialize all network parameters (the weights) randomly.
2. Compute cost/loss and then the gradient of the cost/loss function with respect to each of the model parameters.
3. **Backpropagation:** Adjust the model parameters by a small step α (the learning rate) in the opposite direction of the gradient:

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$$

Cost/Loss Functions

Goal is to minimise this function.

Mean Squared Error for **regression task:**

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

Logistic Function for **binary classification:**

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

Cross-Entropy Cost for **multiclass (multinomial) classification:**

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

where c_m is the correct class (label) for the m-th sample.

Epoch: A forward pass and backward pass completed for all training data.
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.
Iterations per Epoch: training data / size of batch

Regularization (Dealing with Overfitting)

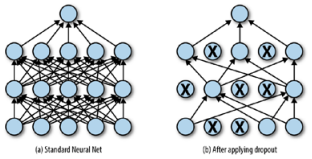
Neural networks are prone to overfitting.

Early Stopping

Terminates the training process before the model overfits. Performs periodical performance evaluation (e.g. every 5 epochs). If the model's performance on the **validation set starts to worsen** (indicating potential overfitting), the training is stopped.

Dropout

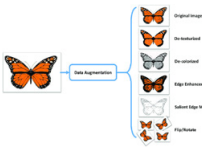
Deactivates randomly selected proportion of the neurons in in each training iteration. This promotes a more distributed and robust learning across all neurons. The drop out percentage is a hyperparameter. Typical is 20-50%.



Data Augmentation

Increases the size and diversity of training data without collecting or labelling new data.

For images: rotations, scaling, cropping, flipping, brightness, contrast, decolorizing, etc.



Regression and Multiclass Classifiaction Models

Regression: One output layer

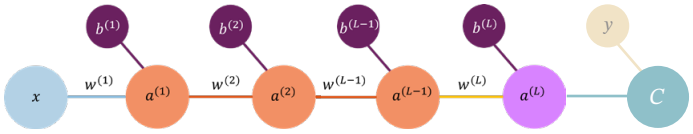
Multiclass Classifiaction:
- Many output layers, or
- One output layer with One-vs-rest method

Backpropagation

Backpropagation uses the chain rule for partial derivatives.

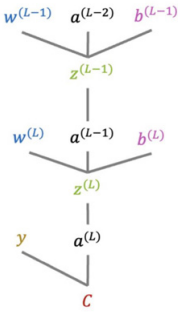
$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial z}{\partial y} \qquad \text{x — y — z} \qquad z = f(y), y = g(x)$$

The final parameter updates are obtained by averaging the computed gradients over all training samples (in a batch). The partial derivatives are averaged among all training samples. The cost function is computed as the average over the individual costs for all training samples.



Preactivation

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



Activation

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

Cost/Loss

$$C = \left(y - a^{(L)}\right)^2 \quad \text{[MSE]}$$

Change of *w* (weights)

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

In practical implementations, the factor **2** is omited because it doesn't change the direction of the gradient (but only scales it). It anyways will be absorbed into the learning rate.

Change of *b* (bias)

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

Change of previous activation

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

To obtain the derivative of the full cost function we average the costs of all training samples.

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n}\sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

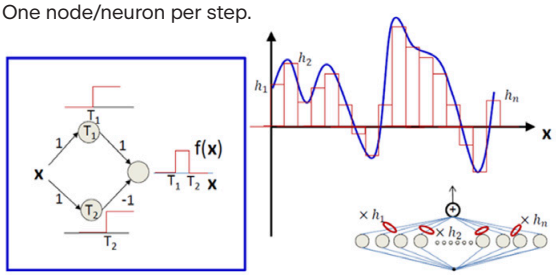
Vanishing Gradient Problem

If we have many hidden layers, and if all partial derivatives are between –1 and 1, then multiplying them makes them exponentially small. Changes in the first layers will diminish since the gradient becomes extremely small. This results in very slowly changing weights and thus very long training times.

Solution: Do not use randomly initialized weights, but rather «carefully pre-trainied weights», e.g. with auto encoders. Backpropagation reuses computations from earlier steps for efficiency

Universality Theorem (Hornik, 1991)

A neural network with **one hidden layer** and arbitrary number of neurons using **nonlinear activation function** can **approximate any given continuous function** to any desired level of accuracy (given enough hidden nodes).
A continuous function can be approximated by cutting it to many tiny pieces. One node/neuron per step.



Probability Theory

Sample Space S**:** Set of all possible outcomes (events) of an experiment.
$S = \{Heads, Tails\}$

Events: Any subset of the sample space is called an event, e.g. $\{Heads\}$

Probability of an Event: *P(A)*

Example: Consider rolling a die. The sample space is $S = \{1, 2, 3, 4, 5, 6\}$.
Event $A = \{2, 4, 6\}$ represents rolling an even number.
Event $B = \{1, 2, 3\}$ represents rolling a number less than or equal to 3.
$A \cup B = \{1, 2, 3, 4, 6\}$ is the event of rolling either an even number or a number less than or equal to 3.
$A \cap B = \{2\}$ is the event of rolling a 2.
$A^c = \{1, 3, 5\}$ is the event of rolling an odd number (the complement of event A as defined above).
The probability of rolling an even number is given by
$P(A) = P(2) + P(4) + P(6) = 1/6 + 1/6 + 1/6 = 1/2.$

Marginal probability *P(A)*

Wahrscheinlichkeit eines Ereignisses. The probability of event occuring, not conditioned on other events.

$$P(A) = \frac{|A|}{|\Omega|}$$

Joint Probability *P(A and B)*

Probability of two or more events happening at the same time.

Independent events (stochastisch unabhängig)
Product Rule / Multiplikationssatz

$$P(A \cap B) = P(A) \cdot P(B) = \frac{|A|}{|\Omega|} \cdot \frac{|B|}{|\Omega|} = \frac{|A \cdot B|}{|\Omega|^2}$$

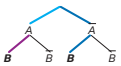
Bsp: Flipping heads on a coin: *P(A)*, and rolling a 4 on a die: *P(B)*

Dependent events (stochastisch abhängig / bedingte Wahrscheinlichkeit)
Conditional Probability / Multiplikationssatz (Pfadwahrscheinlichkeit)

$$P(A \cap B) = P(A) \cdot P(B|A) = P(B) \cdot P(A|B)$$

Satz der Totalen Wahrscheinlichkeit *P(B)*

$$P(B) = P(A) \cdot P(B|A) + P(\bar{A}) \cdot P(B|\bar{A})$$



Bayes' Theorem (Conditional Probability) *P(B given A)*

Posterior propability

$$P(H|E) = \frac{P(H)P(E|H)}{P(E)} = \frac{P(H)P(E|H)}{P(H)P(E|H) + P(\neg H)P(E|\neg H)}$$

$$= \frac{P(H \cap E)}{P(E)} = \frac{|H \cap E|}{|E|} = 1 - P(\neg H|E)$$

$P(H|E)$: The **propability** of hypothesis *H* being true, given that evidence *E* has occurred.

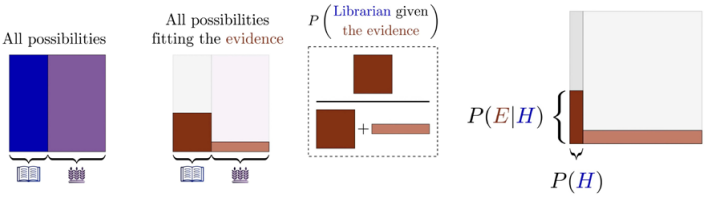
$P(E|H)$: The **likelihood** of observing evidence *E* given that hypothesis *H* is true. Asks how probable is the observed data *E*, based on a specific hypothesis. Quantifies how well the hypopthesis *H* explains the observed data *E*. Higher likelihood suggests that the hypothesis *H* is more likely to be true.

$P(H)$ = Probability a hypothesis is true (before any evidence)

$P(E|H)$ = Probability of seeing the evidence if the hypothesis is true

$P(E)$ = Probability of seeing the evidence

$P(H|E)$ = Probability a hypothesis is true given some evidence



Likelihood vs Probability

Probability refers to the chance that a particular outcome occurs based on the values of parameters in a model.

Likelihood refers to how well a sample provides support for particular values of a parameter in a model.

Law of Total Probability

Let $B_1, B_2, \ldots, B_n$ be mutually exclusive and collectively exhaustive events, meaning no two events can occur at the same time, and one of these events must occur. Then, the probability of an event *A* can be expressed as:

$$P(A) = P(A \mid B_1)P(B_1) + P(A \mid B_2)P(B_2) + \ldots + P(A \mid B_n)P(B_n)$$

Sums the probabilities of *A* occurring in each scenario B_i, weighted by the probability of each scenario B_i occurring. Use this if we don't know $P(A)$.

Example:

Likelihood: $P(\text{Positive}|\text{Disease}) = 0.99$
A disease test is 99% accurate (it correctly identifies 99% of the time).
(Probability of getting a positive test given that the person has the disease).

Prior probability: $P(\text{Disease}) = 0.01$, 1% of the population has the disease.

We want to know the probability one actually has the disease, given the test is positive:

$P(\text{Disease}|\text{Positive}) = \frac{P(\text{Positive}|\text{Disease}) \cdot P(\text{Disease})}{P(\text{Positive})}$

1. Use the law of total probability to find: $P(\text{Positive})$

$= P(\text{Positive}|\text{Disease}) \cdot P(\text{Disease}) + P(\text{Positive}|\text{No Disease}) \cdot P(\text{No Disease})$

$P(\text{Positive}|\text{No Disease}) = 0.01$
$P(\text{No Disease}) = 0.99.$
$P(\text{Positive}) = (0.99 * 0.01) + (0.01 * 0.99) = 0.0198$

Posterior probability using the Bayes' theorem:

$P(\text{Disease}|\text{Positive}) = \frac{0.99 * 0.01}{0.0198} \approx 0.5$

Let's say we test 10,000 people. 1% have the disease. Among the 100 people who have the disease: 99% will test positive → 99 **true positives.** 1% will test negative → 1 false negative. Among the 9,900 people who don't have the disease: 1% will test positive → 99 **false positive.**

$$\frac{\text{True Positives}}{\text{Total Positives}} = \frac{99}{198} = 0.5$$

Estimating the parameters in probabilistic models

Likelihood Function $L(\theta; X)$

Idea: Find parameter values that maximize the likelihood function.
Likelihood is a measure of how well a statistical model explains observed data.

Example: Our Hypothesis is that the normal distribution might best fit the data:

$$L(\mu, \sigma^2; X) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x^{(i)} - \mu)^2}{2\sigma^2}\right)$$

Asks how likely it is to observe the data *X* given the parameters
μ (mean) and σ² (standard deviation).

Goal is to find μ and σ, the normal distribution that best fits the data points
→ Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation (MLE)

Method for finding the parameter values that maximize the likelihood function.
Negative log-likelihood:

$$\theta_{\text{MLE}} = \underset{\theta}{\text{argmin}}\{ -\log P(X|\theta)\} = \underset{\theta}{\text{argmax}}\, P(X|\theta)$$

Example with a Bernoulli distribution:
We observe 10 coin flips with 7 heads and 3 tails.
$L(p; X) = p^7(1-p)^3$
Negative Log-Likelihood:
$NLL(p; X) = -7log(p) - 3log(1-p)$

Maximum a Posteriori (MAP) Estimation

Seeks the parameter values that maximize the posterior probability.

$$\theta_{\text{MAP}} = \underset{\theta}{\text{argmax}}\, P(\theta|X) = \underset{\theta}{\text{argmax}}\, \frac{P(X|\theta) \cdot P(\theta)}{P(X)}$$

$P(\theta|X)$: Posterior probability of the parameters given the data
$P(X|\theta)$: Likelihood (how probable is the observed data)

Naïve Bayes

Generative model (supervised learning) for **classification tasks.**

Generative Models

Learns the underlying **probability distribution** of the data within each class. Focus on understanding how the data is generated. Assumes that all features are **conditionally independent** given the class label. This probability is then used to make predictions or generate syntethic data.

Learning:

Prior probabilities: Frequency of each **class** in the training data, e.g. $P(Cat)$

Likelihoods: Calculates probabilities of **features** occurring in different classes. e.g. calculates $P(pointyears|Dog)$ and $P(pointyears|Cat)$.

Prior probabilities and likelihoods are used to calculate **posterior probability:** e.g. $P(Cat|observed features)$

Generative vs Discriminative

Generative: Learns the **joint probability** distribution for each class *P*(*X*, *Y*). Thats is the probabilty of observing class *k*, given some input features.

Discriminative: Learns a **decision boundary** between two classes. They learn the **conditional probability** distribution *P*(*Y* | *X*) directly.

Model	Advantages	Disadvantages
Generative 	Can generate new data samples	Computationally expensive
	Provides a complete understanding of the data distribution.	Strong assumptions about feature independence
	Robust with limited data.	
Discriminative 	More accurate and efficient for classification.	Cannot generate data samples
	Weaker assumptions about feature independence.	Less interpretable
	Well-suited for high-dimensional data	Performance depends on the choice of features

Bayes' Theorem

$$P(class|features) = \frac{P(features|class) * P(class)}{P(features)}$$

Posterior probability: The probability of the data point belonging to the class given its features. What the model wants to predict. The probability of observing class *k* changes after we observe feature *x*. Observing feature *x* changes how likely we think class *k* is.

Likelihood: The probability of observing those features given that the data point belongs to that class. Can be used to generate new features.

Prior propability: Initial belief before considering the new evidience (feautres).

Evidence: How likely it is to observe the feature on its own.

Uses **maximum likelihood estimation (MLE)**.

Types of Naïve Bayes

Bernoulli Naive Bayes: Where features are represented as **1 or 0**.

Multinomial Naive Bayes: For count data, such as **word counts** in text documents. e.g. spam detection.

Gaussian Naive Bayes: For **continuous data** with numerical features, and assumes that features follow a Gaussian (normal) distribution.

Smoothing

Technique to handle the problem of **zero probabilities** (when a feature value is not encountered during training is encountered during testing).

Solution: Add a small number to the probability estimates, ensuring that no probability is ever exactly zero.

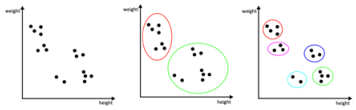
Training Complexitites

Naïve Bayes: $O(n \cdot m)$, *n* samples, *m* features
Linear Regression: $O(n \cdot m^2)$ for normal equation
or $O(n \cdot m \cdot k)$ with gradient descent wiht *k* interations.
Neural Networks: umbestimmt.

Clustering

Unsupervised Learning

Goals: data understanding, class identification, data reduction, outlier detection



Clustering is the task of grouping a set of similar objects (cluster) togehter. Clustering is a collection of subsets of all data points *X*. Goal is that samples in the same cluster are similar, i.e. have a small distance to the centroid.
Für > 2D-Daten wird **Principle Component Analysis (PCA)** verwendet.

Distance measure

Euclidean distance (L2-norm) for *N*-dimensional points *p* and *q*.

$$d(p,q) = \sqrt{\sum_{i=1}^{N}(q_i - p_i)^2}$$

Types of Clustering

- **Hard clustering:** Each data point belongs to one cluster only. Methods: **K-Means**, **DBSCAN** and OPTICS
- **Soft clustering:** Each data point *m* is assigned a probability p_{km} that it belongs to cluster *k*.

K-Means

Hard clustering. Expectation Maximisation. Tends to find spherical clusters. K-Means ≠ K-Nearest Neighbour (Supervised learning)

Selection of initial centroids

- *Random points:* Randomly selects *K* points in $\mathbb{R}^N$
- *Forgy method:* Randomly chooses *K* observations from the dataset.
- *Random Partition:* Randomly assigns a cluster to each observation.
- *K-means++:* Tries to find better inital centroids.

Algorithm

Input:

- $X = \{x^{(1)}, \ldots, x^{(M)}\}$ set of M data points
- The number of clusters K

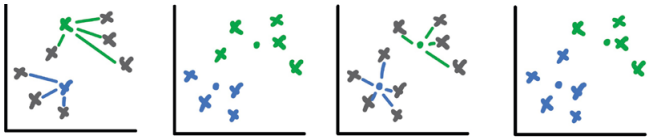
Output: A set of K centroids $\{c_1, \ldots, c_K\}$

Algorithm:
REPEAT

- Compute closest centroid c_k for each data point and "assign" data point to that centroid
- FOR EACH centroid c_k
  - A_k = set of data points currently assigned to c_k
  - $m_k = \frac{1}{|A_k|} \sum_{x^{(m)} \in A_k} x^{(m)}$ (mean of all data points in A_k)
  - $c_k = m_k$
- END FOR

UNTIL stop-criterion is met
RETURN $c_1, \ldots, c_K$ as the centroids of the K clusters

Konvergiert in der Theory. Die Summe der Abstände wird in jedem Schritt kleiner. In der Praxis aber nicht wegen Rechenungenauigkeit.



Runtime: $O(L \cdot k \cdot n \cdot m)$ *k* Clusters, *m* Datapoints, *n* features, *L* iterations

Stop-Criterions

- When the coordinates of the centroids change only very little
- When only few data points are re-assigned to a different centroid

Time-boxed criteria:

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

Quality of *K*-Means Clustering

Potential Function Φ (phi): Measures the **squared distance** between each data point and its **closest centroid**.

Let $C = \{c_1, \ldots, c_K\}$ denote a set of K centroids,

then $\Phi(C, X) = \sum_{m=1}^{M} min_{c \in C}(d(x^{(m)}, c)^2)$

In practice, one will run K-Means clustering algorithm several times with different randomly selected initial centroids, and then choose the best clustering.

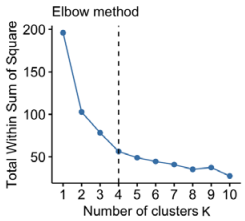
Choosing the Number of Clusters *K*

Methods: **Elbow Method** and the **Silhouette Method**

Elbow Method

Run K-Means with different values of *K* and plot the potential function Φ.

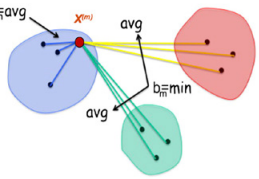
Number of clusters: Where the **slope of the curve significantly decreases.**



Silhouette Method

Measures how well a data *x* point fits into its own cluster.
High value of the **Silhouette Score** means better clustering.

$$s_m = \frac{b_m - a_m}{max(b_m, a_m)} \quad s_m \in [-1, 1]$$

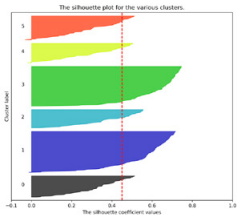


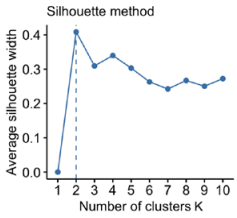
b_m Smallest average distance of $x^{(m)}$ to all points in any other clusters. The bigger (>0) the better the point matches the current cluster.

a_m Average distance of $x^{(m)}$ with all other points in the same cluster.

Average Silhouette Score (Width): Average of the Silhouette Scores of all data points in the training set. $\frac{1}{M}\sum_{m=1}^{M} s_m$

Silhouette Plot: Scores for all points in the dataset





DBSCAN

Density-based Spatial Clustering of Applications with Noise
Identifies points in «densely populated» regions. Hard clustering.

Advantages:
- We do not need to specify the number of clusters in advance.
- Find clusters with arbitrary shapes and it explicitly identifies outliers (noise)

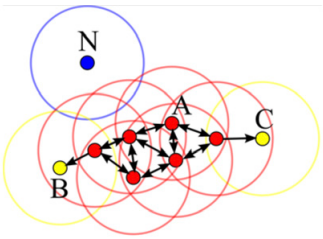
Disadvatage: If there are large differences in the data densities then it does not detect meaningful clusters

Has **two parameters** that determine the neighborhood of the points:

- *minPts:* The minimum number of points (a threshold) clustered together for a region to be considered dense
- ϵ: A distance measure to define the neighborhood of any point. Point *p* is reachable from point *q* if $distance(p, q) < \epsilon$.

Three **types of points**

- *Core point:* has at least *minPts* points within distance from itself (A)
- *Border point:* has at least one core point within distance (B, C)
- *Noise point:* a point that is neither a core nor a border point (N).



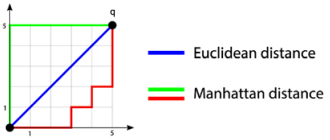
The DBSCAN Algorithm

1. Select an unprocessed point P
2. If P is not a Core Point (i.e. there are less than *minPts* points within range ϵ), then classify P as noise and go back to Step 1
3. Otherwise, if P is a core point, a new cluster is formed as follows:
   - Assign all neighbors of P (i.e. all points within distance ϵ from P) to the new cluster
   - Repeat previous assignment step for all newly-assigned neighbors that are core points
4. Go back to Step 1
5. Continue algorithm until all data points have been processed.

Runtime: Naive impl. $O(M^2)$, with better data structure $O(M \log M)$, M samples

Distance Metrics

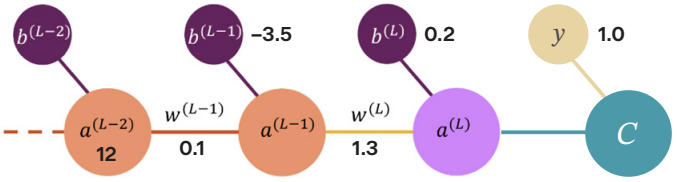
Manhattan distance can be used as well. Its faster to compute than Euclidean distance.



Comparison

Algorithm	k-Means	DBSCAN	Hierarchical
Advantages	• relatively easy and efficient to implement • works for very large datasets • computationally faster then other clusterings	• no need to define number of clusters • can discover arbitrarily shaped clusters • robust to outliers and noise	• no need to define number of clusters • calculates a whole hierarchy of clusters • uses dendrogram to visualize the clusters
Disadvantages	• sensitive to initialization • number of clusters is hard to choose • unable to handle outliers or noisy data	• not partitionable for multiprocesses • datasets with different densities are tricky • can depend on the order of the data	• algorithm decisions cannot be undone • not very scalable • does not work well with outliers and noise

Appendix

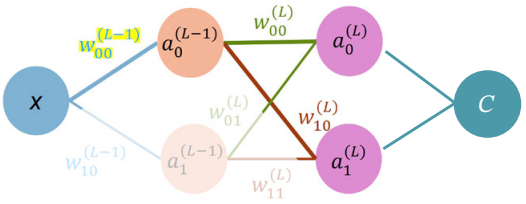


σ = sigmoid, $\sigma' = \sigma(1-\sigma) = \sigma - \sigma^2$

$$a^{(L-1)} = \sigma(\underbrace{12 \cdot 0.1 - 3.5}_{-2.3 = z^{(L-1)}}) = 0.0911$$

$$a^{(L)} = \sigma(\underbrace{0.0911 \cdot 1.3 + 0.2}_{0.318 = z^{(L)}}) = \underline{0.578}$$

$$\frac{\partial C}{\partial w^{(L)}} = a^{(L-1)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y), \quad y = 1$$
$$= 0.0911 \cdot (\sigma(0.318) - \sigma(0.318)^2) \cdot 2(0.578 - 1)$$
$$= \underline{-0.0187}$$



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

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

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

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

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

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

A father claims it snowed last night. His daughters have some insights. What is the probability it actually snowed? Determine P(S | C), the probability of snowfall given the father's claim.

S: Event of snowfall last night.
C: Event of the father claiming snowfall.

Known Probabilities (the daughters insights):

P(S) = 1/8 (Probability of snowfall)
P(¬S) = 7/8 (Probability of no snowfall, calculated as 1 – P(S)
P(L) = 5/6 (Probability of the father lying)
P(T) = 1/6 (Probability of the father telling the truth, calculated as 1 – P(L))

$$P(S|C) = \frac{P(C|S) \cdot P(S)}{P(C)} = \frac{P(T) \cdot P(S)}{P(C)} = \frac{1}{36}$$

Law of Total Probability

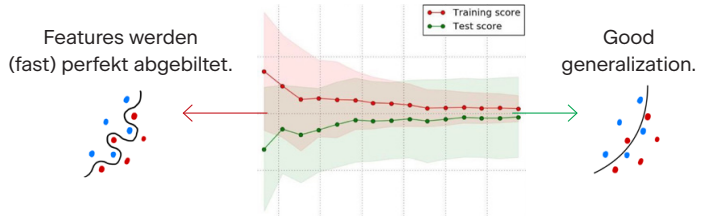
$$P(C) = P(C|S) \cdot P(S) + P(C|\neg S) \cdot P(\neg S)$$
$$= P(T) \cdot P(S) + P(L) \cdot P(\neg S) = \frac{3}{4}$$

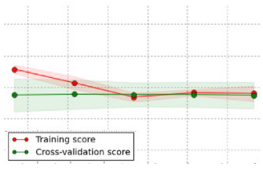
Evaluation of Learing Curves

Potential underfitting (high bias): If both curves are **close to each other** and both of them have a **low score.**

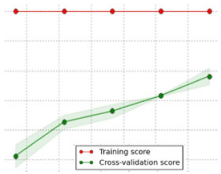
Potential overfitting (high variance): If training curve has a much better score than testing curve. The model does not generalize well.

A good model: If both curves are **close together** and both of them have a **high score.**

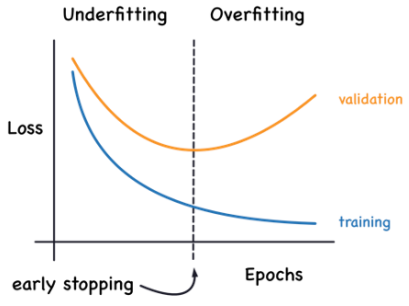


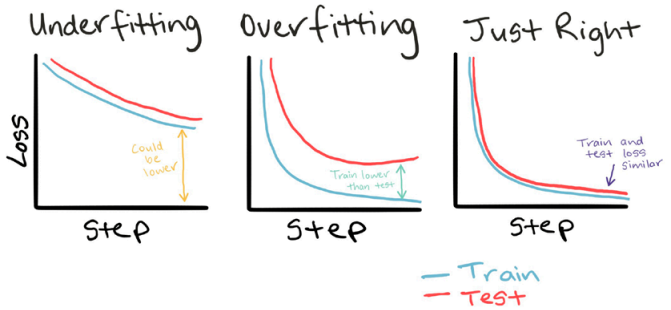


Underfitting: Model lernt nicht.
→ Model mächtiger machen (mehr Nodes / hidden layers, etc.)



Overfitting: Model-Komplexität reduzieren oder mehr Daten sammeln (more data decrease the risk of overfitting) → Data Augmentation





Normal Equation

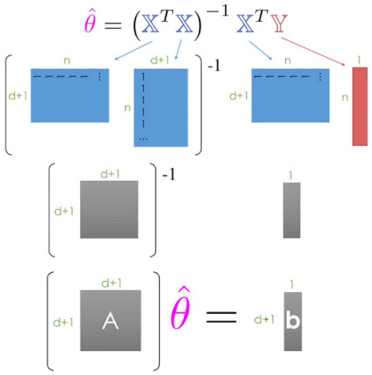
Closed form solution to a multilinear regression model.

$$\hat{\mathbb{Y}} = \mathbb{X}\,\theta$$

Normal equation:

$$\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$$

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\,\mathbb{X}^T\mathbb{Y}$$



The sum of squared residuals, i.e. the loss, can be expressed as the scalar product of the residual vector with itself:

$$\boldsymbol{\varepsilon} = \boldsymbol{y} - \boldsymbol{X\theta}$$

$$\begin{aligned} J &= \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} = (\boldsymbol{y} - \boldsymbol{X\theta})^T(\boldsymbol{y} - \boldsymbol{X\theta}) \\ &= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X\theta} + \boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X\theta} \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{y} + \boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X\theta} \end{aligned}$$

Using some calculus, the gradient of the squared residuals yields

$$\frac{\partial}{\partial\boldsymbol{\theta}} J = -2\boldsymbol{X}^T\boldsymbol{y} + 2\boldsymbol{X}^T\boldsymbol{X\theta}$$

Equating the gradient with zero produces the so-called normal equations:

$$\boldsymbol{X}^T\boldsymbol{X\theta} = \boldsymbol{X}^T\boldsymbol{y}$$