Machine Learning

Samenvatting

Manuel Mol .

Samenvatting

This document provides a foundational introduction to machine learning concepts and algorithms. We'll delve into various tasks that machine learning can address, along with the models used to achieve them. We'll explore how these models are evaluated and some common challenges encountered in the machine learning process.

Samenvatting

Published Jun 08, 2024

Introduction:

Some important definitions for machine learning are:

- **Tasks:** Tasks are problems that can be solved with machine learning
- **Model**: A model is a mathematical representation of a real-world process. It is a set of rules that describe the relationship between input and output variables.

Tasks:

There are a few different types of tasks in machine learning. The first is **predictive tasks.** Here we predict a target value from a number of features:

- **Regression:** The target value is continuous
- **Classification:** The target value is discrete
- **Predictive clustering:** The target value is a cluster

We also have **descriptive tasks,** where we describe the data according to some underlying structure:

- **Descriptive clustering:** We cluster the data
- **Association rule learning:** We find rules that describe the data
- **Subgroup discovery:** We find subgroups of the data

Another way to categorize tasks is by the way they learn:

- **Supervised learning:** The model is trained on labeled data
- **Unsupervised learning:** The model is trained on unlabeled data

Model:

A model is a mathematical representation of a real-world process. It is a set of rules that describe the relationship between input and output variables. You can categorize models by there **main intuition:**

- **Geometric model:** The model is based on geometric function such as distance. All instances can be represented in in **instance space**. An example of a geometric model is the **Linear classifier** model.
- **Probabilistic models:** aim for reducing uncertainty using probability distributions. An example of a probabilistic model is the **Naive Bayes** model.
- **Logical models:** use logical expressions to describe the relationship between input and output variables. An example of a logical model is the **Decision tree** model.

You can alo categorize the models by the **modus operandi** (mode of operation):

- **Grouping models:** dividing the instance space into segments; in each segment a very simple (e.g., constant) model is learned. An example of a grouping model is the **tree model**. It can't distinguish between individual instances beyond this resolution.
- **Grading models:** A single, global model over an instance space. An example of a grading model is the **linear model**. It can distinguish between individual instances and the resolution is not limited.

Model phases:

- **Training/learning:** where te model is trained on the data
- **Inference:** where the model is used to make predictions

Features:

A measurement that can be performed on any instance. When features are not in the correct form, they can be transformed into a new feature space:

- **Feature construction:** turn images into pixels etc.
- **Discretisation:** Numerical features are transformed into categorical features
- **Feature transformation:** project the data into a new feature space
- **Feature selection:** removing irrelevant features

Binary classification:

In binary classification, the target value is binary.

How can we evaluate performance?

Contingency table: A table that shows the number of true positives, false positives, true negatives, and false negatives. We can calculate a few metrics from this table:

- **Accuracy:**

$$\text{Accuracy} = \frac{\text{correct}}{\text{total}} \tag{1}$$

- **Error rate:**

$$\text{Error rate} = \frac{\text{incorrect}}{\text{total}} \tag{2}$$

- **True positive rate / Sensitivity / Recall:** How many sick people are identified as having the illness, How many relevant items are selected

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

- **True negative rate / Specificity:** How many negative items are selected that are truly negative

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{4}$$

- **False positive rate:**

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5}$$

- **False negative rate:**

$$\text{FNR} = \frac{\text{FN}}{\text{FN} + \text{TP}} \tag{6}$$

- **Precision:** How many selected items are truly relevant

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

- **F score:** The harmonic mean of precision and recall

$$F = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

We can use a train test split to evaluate the model. We can split the data in a training set and a test set. We can then train the model on folds of the training set and evaluate the model on the test set.

We can also use a separate validation set to tune the hyperparameters of the model.

Train set error: The error on the training set
Test set error: The error on the test set

Overfitting: When the model performs well on the training set but poorly on the test set. This is because the model has learned the noise in the training set.
Underfitting: When the model performs poorly on the training set. This is because the model is too simple.

ROC curve: A curve that shows the true positive rate against the false positive rate. The area under the curve is a measure of the model's performance. The closer the area is to 1, the better the model. The points on the curve are the result of changing the threshold of the model. If the line is above the diagonal, the model is better than random.

Scoring and ranking:

A **scoring classifier** is a classifier that outputs a score for each instance.

Margin: The distance between the score of the correct class and the score of the incorrect class. The margin is a measure of the confidence of the model.
Loss function: A function that measures the error of the model.

Ranking: The order of the instances based on the score of the classifier.

Ranking error rate: The error rate of the ranking.

Class probability estimation:

The probability that an instance belongs to a certain class. The output of the classifier shows how likely it is that the instance belongs to a specific class rather than which class it will belong to.

To determine how good these probabilities are, we can use the **squared error loss function**. This function measures the difference between the predicted probability and the true probability. The **mean squared error** is the average of the squared error loss function.

Multi-class classification:

In multi-class classification, the target value is a class from a set of classes. There are 2 different types of algorithms for multi-class classification:

- **Inherently non-binary:** Algorithm like decision trees
- **Inherently binary:** Support Vector Machines

Turning a binary classifier into a multi-class classifier can be done. There are 2 main approaches:

- **One-vs-all:** Train a binary classifier for each class. You get an incorrect classification if one of the classifiers classifies the instance as the class incorrectly.
- **One-vs-one:** Train a binary classifier for each pair of classes

$$\text{num}_c = \frac{k(k-1)}{2} \tag{9}$$

num_c classifiers are needed to create a symmetric multi-class classifier with k classes.

One versus one is more accurate than one versus all, but one versus all is faster.

How to measure performance:

Accuracy can still be calculated in the same way as in binary.

Precision and recall are calculated for each class. The precision of the model is then calculated by taking the average of the precision of each class. The same goes for recall.

We can still use an **ROC curve** to evaluate the model. The ROC curve is calculated for each class. The area under the curve is then calculated by taking the average of the area under the curve of each class.

Regression:

Regression is a predictive task where the target value is continuous.

Regression functions are evaluated by using a loss function on the residuals. The residuals are the difference between the predicted value and the true value.

To prevent overfitting, the number of parameters should be considerably less than the number of data points

Bias
refers to the error introduced by approximating a real-world problem, which may be complex, by a simplified model. High bias means the model makes strong assumptions about the data, which can lead to systematic errors and underfitting.

$$\text{High bias} = \text{underfitting}$$

Variance
refers to how much your model's predictions change when you use different training data. High variance means the model is very sensitive to the training data and may not perform well on new, unseen data

$$\text{High variance} = \text{overfitting}$$

Unsupervised and descriptive learning:

In descriptive learning the task is to come up with a description of the data. We can use clustering to group the data.

Clustering:

Clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.

To evaluate performance with a ground truth we can use the **Rand index.** which is similar to the accuracy in classification. We can also use **purity**.

$$\text{purity} = \frac{\text{most frequent in cluster}}{\text{total}} \tag{10}$$

How can we check the performance without **ground truth?**

Silhouette Coefficient:

$$s = \frac{b - a}{\max(a, b)} \tag{11}$$

Where:
a: the mean distance between an instance and all other points in the same cluster.
b: the mean distance between an instance and all other points in the next nearest cluster.

The silhouette ranges from −1 to +1. A high value indicates that the instance is well matched to its own cluster and poorly matched to neighboring clusters.

Subgroup discovery:

Subgroup discovery is a supervised learning task but it is different from classification, as it addresses different goals → discovery of interesting population subgroups instead of maximizing classification accuracy of the induced rule set. *Finding patterns in traffic accident*

Figure 1: **(left)** classifier **(right)** subgroup discovery

A **Chi-squared test** can be used to determine if the subgroup is significant.

Tree models:

Decision trees:

A decision tree is a flowchart-like structure in which each internal node represents a feature(or attribute), each branch represents a decision rule, and each leaf node represents the outcome.

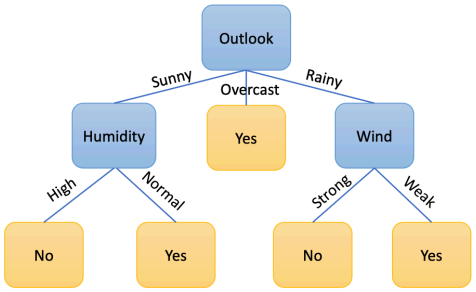

Figure 2: Decision tree

Dept and leave count
Every decision tree with d binary features can be represented with a decision tree with 2^d leaves and a dept of $d + 1$

A decision tree can be represented as an logical expression.

How to train a decision tree:

1. Start by placing the **best feature at the root** of the tree, this splits the data into subsets.
2. Select the next feature to **split** the data set
3. **Repeat**

What is the best split?

We can use purity to determine the best split:

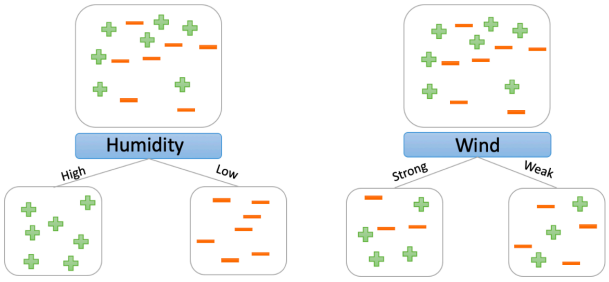

Figure 3: Purity

In Figure 3, the data on the right is pure. The data on the left is impure. There are a few different algorithms to determine the purity of the data:

1. **Gini impurity:** The probability of a random instance being misclassified
2. **Entropy:** The average amount of information produced by a random variable
3. **Classification error:** The fraction of instances that are misclassified

We can use the empirical probability (probability in the data set) of the positive classes. We can calculate this by dividing the number of positive instances by the total number of instances.

$$\dot{p} = \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus}} \tag{12}$$

Entropy:

Entropy is a measure of the uncertainty of a random variable. The entropy of a random variable is the average level of "information", "surprise", or "uncertainty" inherent in the variable's possible outcomes.We can calculate the entropy of a set of instances by using the following formula:

$$\text{Entropy} = -p^{\oplus} * \log_2(p^{\oplus}) - p^{\ominus} * \log_2(p^{\ominus}) \tag{13}$$

Where $p^{\oplus}$ is the probability of a positive instance and $p^{\ominus}$ is the probability of a negative instance.

We can use entropy to determine the best split. The left side of Figure 3 has a low entropy, while the right side has a high entropy. The lower the entropy of a split, the better the split.

To find out if a split is useful at all, we can look at the **purity gain**: The higher the purity gain, the better the split.

$$\text{purity gain} = \text{original entropy} - \text{entropy after splitting} \tag{14}$$

Pruning:

There are a few ways to prevent overfitting. One is limiting the number of iterations, but another method is to prune the tree. Here we remove weak branches from the tree.

> **Pruning in practice**
> When we prune the tree, we start by replacing the leave notes with the majority class (the class that occurs most often). If the prediction accuracy is not affected, the change is kept. To check this we use a validation set.

Imbalance:

Imbalance is when the data set has a disproportionate ratio of one class to another. This can lead to a biased model. We can use **resampling** to fix this. We can either **oversample** the minority class or **undersample** the majority class. This works for Gini impurity and entropy, but not for $\sqrt{\text{Gini}}$

Regression trees:

For a regression tree we can't use the normal impurity measures. Instead we use the **variance** of the data. The variance is the average of the squared differences from the mean. We can use the variance to determine the best split. The lower the variance, the better the split.

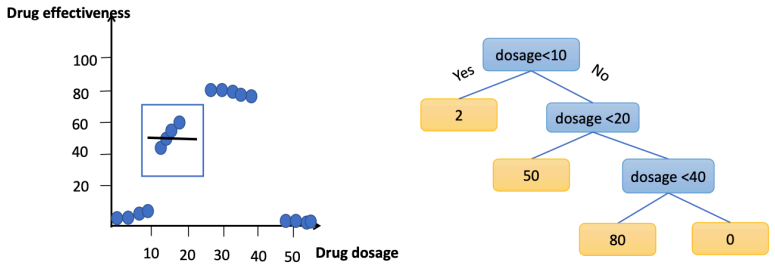

Figure 4: Regression tree

Distance-based models:

Distance-based algorithms are machine learning algorithms that classify instances by computing distances between these instances and a number of internally stored exemplars.

> **Exemplars**
> A typical or standard example. In machine learning, exemplars are the instances that are stored in the model.

The classification assigned to an instance is influenced the most by the **exemplars** that are closest to the instance. **Neighbors** can be near exemplars.

How do you calculate distance?

There are a few different ways to calculate distance:

1. **Minkowski distance:** A generalization of Euclidean and Manhattan distance, where the distance is calculated by the following formula.

$$\text{Minkowski distance} = \left(\sum(|x_i - y_i|^p)\right)^{\frac{1}{p}} \tag{15}$$

   If $p = 1$, the distance is the Manhattan distance. If $p = 2$, the distance is the Euclidean distance.
2. **Euclidean distance:** The distance between two points as a line.
3. **Manhattan distance:** The distance between two points as a grid.
4. **Hamming distance:** measures the number of positions at which two strings of equal length differ. It is used for comparing binary data or DNA sequences. For example, the Hamming distance between "1 0 1 1 1 0 1" and "1 0 **0** 1 **0** 0 1" is 2 because they differ at two positions.
5. **Chebyshev distance:** The maximum distance between two points. It is the maximum absolute difference between the coordinates of the points.

Defining our own distance metric
We can define our own distance metric. It should take in an x and a y such that:

1. Distances between a point and itself are zero
2. All other distances are larger than zero
3. Distances are symmetric
4. Triangle inequality: detours can not shorten the distance

Distance based classifiers:

Exemplars are prototypical instances within clusters/classes. The classification of an instance is influenced by the exemplars that are closest to the instance. Some exemplars include **Geometric mean (centroid)** and **Geometric median (medoid)**.

We can now classify an instance by looking at the nearest exemplar.

K-nearest neighbors:

The K-nearest neighbors algorithm is a simple algorithm that stores all available cases and classifies new cases based on the k amount of nearest neighbors. A larger k will make the model more robust to noise, but less sensitive to local patterns.

Curse of dimensionality
In high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative

Distance-based clustering:

Distance-based clustering is a clustering algorithm that groups instances based on their distance to each other. The goal is to find compact clusters.

Scatter
The distance between the instances in a cluster. The smaller the scatter, the better the cluster.

K-means clustering

K-means clustering is an algorithm used to group data points into 'k' clusters based on their features. Here's how it works:

1. **Initialization**: Choose 'k' initial cluster centroids randomly.
2. **Assignment**: Assign each data point to the nearest centroid, forming 'k' clusters.
3. **Update**: Calculate the new centroids by averaging the points in each cluster.
4. **Repeat**: Repeat the assignment and update steps until the centroids no longer change significantly or a set number of iterations is reached.

The goal is to minimize the variance within each cluster, making the clusters as distinct as possible.

Centroid and medoid

A **Centroid** is the average of all points in a cluster. A **Medoid** is the most central point (from the dataset) in a cluster. To find the medoid, we calculate the distance between each point and all other points in the cluster. The point with the smallest sum of distances is the medoid.

The K -means algorithm aims to choose centroids that minimize the **inertia** or within-cluster scatter. The inertia shows how well the data points are clustered around the centroids. The lower the inertia, the better the clustering. Very low inertia can indicate overfitting.

Silhouette score:

The silhouette score is a measure of how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from −1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. We can visualize this by using a silhouette plot:

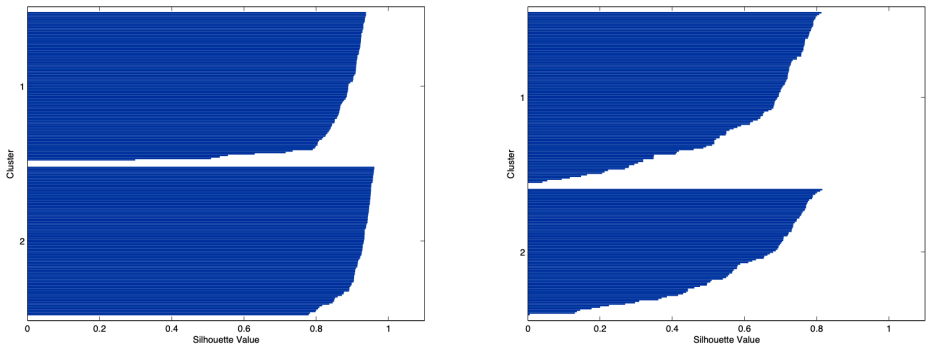

Figure 5: **(Left)** silhouette for cluster with high density, **(Right)** silhouette for cluster with low density

Hierarchical clustering:

K-means clustering is *flat*, which means that it only groups instances in one level. Hierarchical clustering is a *nested* clustering algorithm that groups instances in a tree-like structure. We typically use **dendograms** to visualize the clusters.

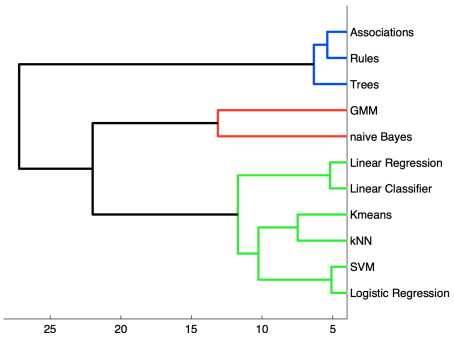

Figure 6: Dendogram

In order to create dendograms we need a way to compare clusters. We can use the **linkage function** to do this. A linkage function is a function that determines the distance between two clusters. There are a few different linkage functions:

1. **Single linkage:** The distance between two clusters is the distance between the closest two points in the clusters.
2. **Complete linkage:** The distance between two clusters is the distance between the two furthest points in the clusters.
3. **Average linkage:** The distance between two clusters is the average distance between all points in the clusters.
4. **Centroid linkage:** The distance between two clusters is the distance between the centroids of the clusters.

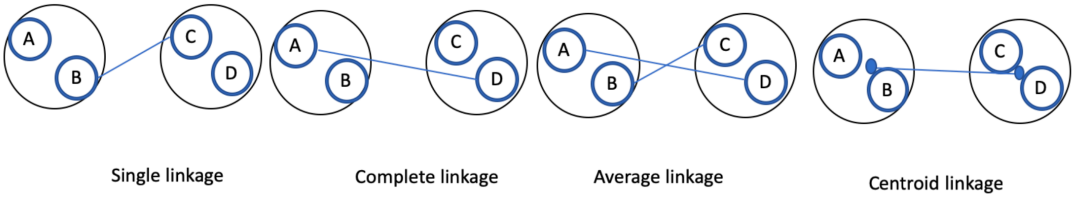

Figure 7: Linkage functions

Linear models:

Linear models use concepts like lines and planes to separate instances. Linear models exist for all predictive tasks, including classification, probability estimation and regression.

Linear models are simple, parametric and stable. They are prone to **underfitting** (high bias). Linear regression is about finding the parameters (a and b) such that sum of residuals error is minimized. We can use the **least squares** method to do this. The least squares method minimizes the sum of the squared residuals. We can use partial derivatives to find the minimum.

Evaluate the model:

We can evaluate the model by looking at the residuals. The residuals are the difference between the predicted value and the true value. We can use the **root mean squared error** to evaluate the model. The mean squared error is the average of the squared residuals. We can also use R^2 or the **coefficient of determination** to evaluate the model. The R^2 is the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

What can you do about outliers?

1. **ordinary least squares method:** use the ordinary method and then filter out the outliers
2. **total least squares method:** generalizes the least squares method to the situation that both x- and y-values are noisy, but this does have a unique solution.

Multiple features:

The approach used can be extended to multiple features.

Regularization on linear models:

Regularization is a technique used to prevent overfitting. It adds a penalty term to the loss function.

Using linear models for classification:

Linear models can be used for classification by using a threshold. You can for example do binary classification by using a threshold of 0.5. If the output of the model is above 0.5, the instance is classified as 1, otherwise it is classified as 0.

The perceptron:

A perceptron is a simple neural network with two parameters. It's a linear classifier that will achieve perfect separation on linearly separable data is the perceptron. The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

SVM (Support Vector Machine):

SVM is a linear model that tries to find the hyperplane that separates the data with the largest margin. The margin is the distance between the hyperplane and the closest data point. The larger the margin, the better the model.

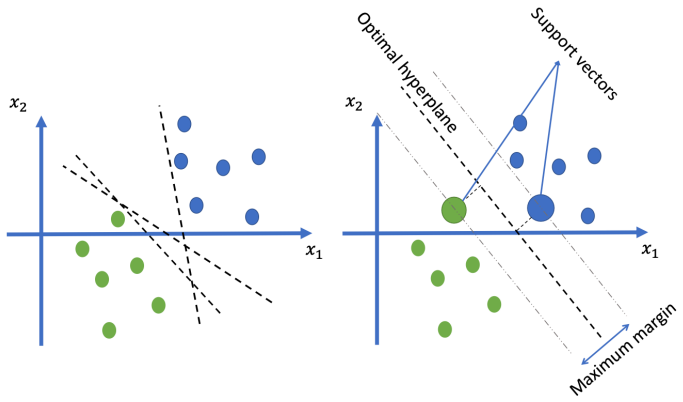

Figure 8: SVM

You can allow soft margins by introducing a **slack variable**. The slack variable allows for some misclassification. The larger the slack variable, the more misclassification is allowed. this can be used to prevent overfitting.

Kernel trick:

Sometimes our data is nonlinear. We can use linear transformations to transform the data into a higher dimension. We can then train a linear model on this higher dimension. This is called the **kernel trick**. The kernel trick allows us to use a linear model on nonlinear data.

Features:

We can distinguish features by their domain:

- **Real:** Features that are real numbers
- **Integer:** Features that are integers
- **Discrete:** Features that are discrete

There are multiple statistics we can calculate on features:

Statistics of central tendency:

- **Mean:** The average of the data
- **Median:** The middle value of the data
- **Mode:** The most common value in the data

Statistics of dispersion:

- **Variance:** The average of the squared differences from the mean
- **Standard deviation:** The square root of the variance
- **Range:** The difference between the maximum and minimum value
- **Interquartile range:** The difference between the 75th and 25th percentile

> **Median vs mean**
> The median is less sensitive to outliers than the mean. The mean is the average of the data, while the median is the middle value of the data.

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. If the skewness is positive, the data is skewed to the right. If the skewness is negative, the data is skewed to the left.

Kurtosis is a measure of the "tailedness" of the probability distribution of a real-valued random variable. If the kurtosis is positive, the data is more peaked than a normal distribution. If the kurtosis is negative, the data is less peaked than a normal distribution.

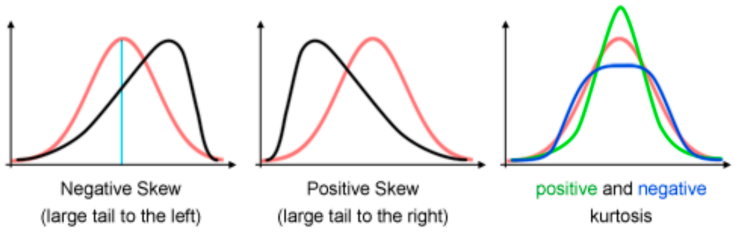

Figure 9: Skewness and kurtosis

Different types of features:

Given these various statistics we can distinguish four main kinds of features:

- **Boolean:** Features that are either true or false
- **Quantitative features:** Features that are real numbers (e.g., age), can be either continuous ($i \in \mathbb{R}$) or discrete ($i \in \mathbb{N}$)
- **Ordinal features:** Features that have a natural order (e.g., grades)
- **Categorical/nominal features:** Features that are categories (e.g., colors)

Structured features:

Structured features are features that are not independent. For example word count in a text. Structured features can be constructed prior to the model or during the models training.

Feature transformations:

A feature transformation is a function that maps a feature to a new feature. We can use feature transformations to transform the data into a new feature space.

↓ to, from →	*Quantitative*	*Ordinal*	*Categorical*	*Boolean*
Quantitative	**normalisation**	**calibration**	**calibration**	**calibration**
Ordinal	**discretisation**	**ordering**	**ordering**	**ordering**
Categorical	**discretisation**	**unordering**	**grouping**	
Boolean	**thresholding**	**thresholding**	**binarisation**	

Figure 10: Feature transformations

- **Normalization & Calibration:** adapt the scale of quantitative features, or add a scale to features that don't have one
- **Ordering:** adds or adapts the order of feature values without reference to a scale.
- **Unordering, Binarisation:** abstract away from unnecessary detail
- **Discretisation & thresholding:** introducing new information

one-hot encoding
A method used to convert categorical data into binary data. Each category is represented by a binary vector, where all elements are zero except for the element corresponding to the category, which is one.

Thresholding:

Thresholding transforms a quantitative or an ordinal feature into a Boolean feature by finding a feature value to split on.

You can use **unsupervised** thresholding, which create sensible threshold based on the mean and median. You can also use **supervised** thresholding, which uses the target value to find the best threshold.

Discretisation:

Discretisation transforms a quantitative feature into a categorical feature. This can be done with **binning**. Binning is the process of dividing the data into bins. You have **unsupervised** discretization, which uses the feature values to create the bins, for example bins with equal width, and **supervised** discretization, which use a scoring function.

Feature normalization:

- **Min-Max normalization:** Scales the data to a fixed range, usually between 0 and 1.

- **z-scores:** Scales the data to have a mean of 0 and a standard deviation of 1.

Principal component analysis:

PCA is a feature-construction technique. It works by computing the principal components (i.e., new features) and using them to perform a change of basis on the data. PCA can be performed on quantitative features

Principal components are new features constructed as a linear combination of original features.

Missing values:

Imputation is the process of replacing missing data with substituted values. There are a few different ways to impute missing values:

- **Mean imputation:** Replace missing values with the mean of the feature
- **Regression imputation:** Use a regression model to predict the missing values
- **Expectation maximization:** Use the expectation maximization algorithm to estimate the missing values

Probabilistic models:

There are 2 different views on probability: the **frequentist** view and the **Bayesian** view. Think of frequentists as observers and Bayesians as learners who adapt their thinking. Both are useful for different situations!

Bayes:

- Prior: $P(Y)$
- Posterior: $P(Y|X)$
- Likelihood: $P(X|Y)$
- Evidence: $P(X)$

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)} \tag{16}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \tag{17}$$

Probabilistic models:

A probabilistic model is a model that tries to predict based on decision functions:

- **Posterior distribution:** a value of Y based on the posterior distribution $P(Y|X)$.
- **Likelihood function:** How likely is it that a spam email would contain the word lottery?

From these functions, rules can be made:

- **Maximum a posteriori:** It chooses the class label (Y) that maximizes the posterior probability of that class label given the observed features.
- **Maximum likelihood:** It chooses the class label (Y) that maximizes the likelihood of observing the features (X) given that class. In other words, it picks the class that makes the observed features the most likely.

$$\text{posterior odds} = \text{likelihood ratio} \times \text{prior odds} \tag{18}$$

Maximum likelihood estimation:

Maximum likelihood estimation is a method used to estimate the parameters of a statistical model. It works by finding the parameter values that maximize the likelihood of the observed data. The likelihood is the probability of observing the data given the model and the parameter values.

Probability distributions for categorical variables:

Multivariate Bernoulli distribution: Models whether or not a word occurs in a document.

Multinomial distribution: models how many times a word occurs in a document.

Naïve Bayes model:

Features are not really independent from each other. The Naïve Bayes doesn't account for order. Naive Bayes has a high bias, but low variance. It is a simple model that is easy to interpret. It is often used for text classification.

Logistic regression

Logistic regression is a linear model that is used for binary classification. It works by finding the parameters that maximize the likelihood of the observed data. The output of the model is the probability that an instance belongs to a certain class. The output is then transformed into a binary output by using a threshold.

Gaussian mixture models:

A Gaussian mixture model is a probabilistic model that assumes that the data is generated by a mixture of several Gaussian distributions. The model tries to find the parameters that maximize the likelihood of the observed data. The model can be used for clustering.

Ensemble models:

Ensemble models are models that combine multiple models to improve performance. The wisdom of crowds: multiple opinions are better than one.

Ensemble methods have the following attributes:

- They construct multiple, diverse predictive models from adapt versions of the training data.
- They combine the predictions of the individual models to produce a single prediction. Often thru the use of voting or averaging.

Bootstrapping

is any test or metric that uses random sampling with replacement. Bootstrapping works in 4 steps:

1. Make a bootstrapped dataset by sampling with replacement from the original dataset
2. Calculate some statistic on the bootstrapped dataset
3. Keep track of the calculation
4. Repeat

Bagging:

Bagging is a method that uses bootstrapping to create multiple models. The models are then combined by averaging the predictions. Bagging can be used for regression and classification.

Subspace sampling
is a method used in ensemble learning to create diverse models. It works by selecting a random subset of features for each model. This makes the models more diverse and less likely to overfit.

Boosting:

Boosting is a method that uses multiple models to improve performance. The models are trained sequentially, with each model trying to correct the errors of the previous model. Boosting can be used for regression and classification.

Adaboost
is a boosting algorithm that works by training a series of weak learners. Each weak learner tries to correct the errors of the previous weak learner. The final model is a combination of all the weak learners. Adaboost can be used for binary classification.

How can we improve learning? → giving the misclassified instances a higher weight, and modifying the classifier to take these weights into account (e.g., by using a weighted loss function).

Variance and bias:

Some sources of misclassification errors:

- **Unavoidable bias**: if instances from different classes are described by the same feature vectors
- **Bias due to low expressiveness of models:** if the data is not linearly separable, a linear model will have high bias
- **High variance:** Tree models have high variance: the change of training data can lead to a different root of the tree and different tree

Bagging is predominantly a variance-reduction technique. It is often used with models that have a high variance, like tree models.

Boosting is predominantly a bias-reduction technique. It is often used with models that have a high bias, like linear models.

Stacking:

Stacking involves training a learning algorithm to combine the predictions of several other learning algorithms. A learning algorithm is trained to make a final prediction using (combining) the predictions of the other algorithms.

Machine learning experiments:

Some questions we like to answer:

- Which algorithm is better on one specific dataset?
- Which algorithm is better on a varied set of datasets?

We can use **cross-validation** to evaluate the performance of the model. Cross-validation works by splitting the data into multiple folds. The model is then trained on all but one fold and evaluated on the remaining fold. This process is repeated for all folds. The performance of the model is then averaged over all folds.

If we have a really small dataset, we can use **Leave one out cross-validation** instead. This works by training the model on all but one instance and evaluating the model on the remaining instance. This process is repeated for all instances.

Comparing the performance of a pair of algorithms:

We can use a **paired t-test** to compare the performance of two algorithms. A t-test is a type of inferential statistic used to determine if there is a significant difference between the means of two groups, which may be related in certain features.

Steps we take to significance test a pair of algorithms:

- We calculate the difference in accuracy on each fold; this difference is normally distributed if the two accuracy's are.
- Our **Null hypothesis** is that true difference is 0 and any differences in performance are attributed to chance
- Calculate a p-value using the normal distribution, and reject the null hypothesis if the p-value is below our significance level α.

A t-test cannot be used across multiple datasets, only on a single dataset.

Comparing the performance of multiple algorithms:

To compare the performance of multiple algorithms, we can use a test called the **Wilcoxon's signed-rank test**.

1. For each pair of observations, calculate the difference between the two values.
2. Rank the absolute values of the differences from smallest to largest, ignoring the signs.
3. **Sum the ranks**, not the values of the positive differences and the ranks of the negative differences separately.
4. The test statistic (W) is the smaller of the two sums of ranks.
5. Compare the calculated test statistic to a critical value from a Wilcoxon signed-rank table (based on the number of pairs and desired significance level). If the calculated test statistic is less than or equal to the critical value, you reject the null hypothesis.

Multiple algorithms multiple datasets:

Friedman test: the goal is to rank the performance of all k algorithms per data set, from best performance (rank 1) to worst performance (rank k). We can then calculate the average rank of each algorithm. Our null-hypothese is that these ranks are on average equal.

We calculate:

1. The **average rank** of each algorithm
2. The **sum of squared differences** between the average rank and the rank of each algorithm
3. The **sum of squared differences** between the average rank and all the ranks

The **Friedman statistic** is the ratio of the second to the third quantity.

The Friedman test tells us whether the average ranks as a whole display significant differences.

Post-hoc tests:

If the Friedman test is significant, we can use a post-hoc test to determine which algorithms are significantly different from each other. We can use the **Nemenyi test** for this. The Nemenyi test is a non-parametric test that compares the average ranks of all algorithms. The idea is to calculate the **critical difference**

Neural networks:

The goal of a neural network is to approximate a non-linear function. The deep learning approach is to try to learn the kernel function of a support vector machine.

Representation learning
is a set of techniques that allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep learning is a type of representation learning.

Image recognition is a common use case for neural networks. The goal is to classify images into different categories. Before the use of neural networks, features were extracted from the images (e.g., edges, corners, etc.). With neural networks, the features are learned automatically.

Neuron:

A neuron consists of a few different parts:

- **Inputs:** The input to the neuron (vector x)
- **Weights:** The weights of the inputs (vector w)
- **Activation function:** The activation function of the neuron (function g)
- **Output:** The output of the neuron ($\hat{y}$)

$$\hat{y} = g(w^T x + b) \tag{19}$$

This is just a linear function without the activation function. By introducing the activation function, we can model non-linear functions.

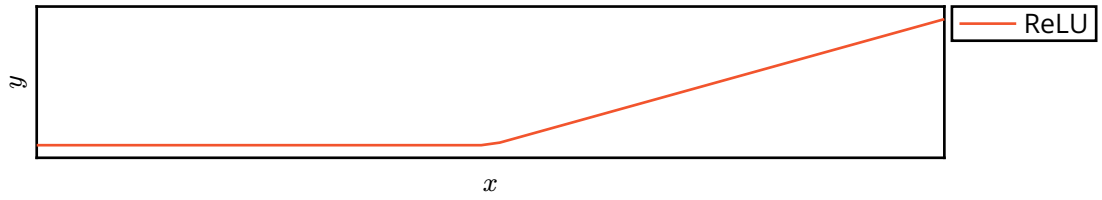

Figure 11: ReLU activation function

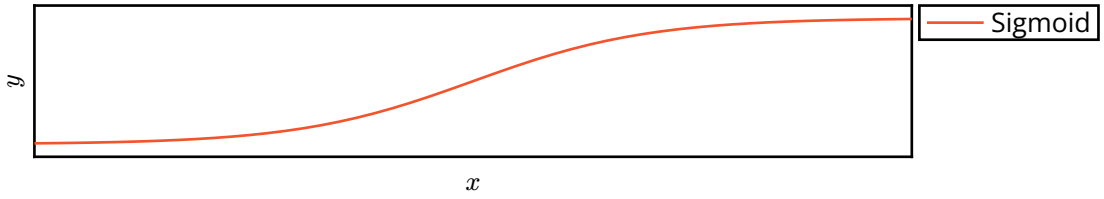

Figure 12: Sigmoid activation function

Multi-layer perceptron:

A multi-layer perceptron is a feed-forward network. It tries to find a mapping between input x and output y. Layers in between input and outputs are called **hidden layers**.

The dimensionality of these hidden layers is called the **width** of the model and the number of layers determines the **depth** of the network.

We call these networks feed forward because the data flows one way.

Architecture
refers to the overall structure of the network, including the number of layers, the number of neurons in each layer, and the connections between neurons.

Training a neural network:

The goal of training a neural network is to find the weights that minimize the loss function. The loss function is a measure of how well the model is performing. The loss function is a function of the weights of the model. The **cross-entropy** function is a common loss function for classification tasks.

Output units: The output units of the model are the units that produce the output of the model. The output units depend on the task:

- **Regression:** The output unit is a linear unit, minimizing the *mean squared error*
- **Binary classification:** for binary classification, the output unit is a *sigmoid unit* (Figure 12), minimizing the *cross-entropy loss*
- **Multi-class classification:** for multi-class classification, the output unit is a *softmax unit*, minimizing the *cross-entropy loss*

Softmax
is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities: example: $[0.02, 0.90, 0.05, 0.03]$. These add up to 1

Gradient decent:

Gradient descent is an optimization algorithm used to minimize the loss function. We want to find the local minimum (valley).

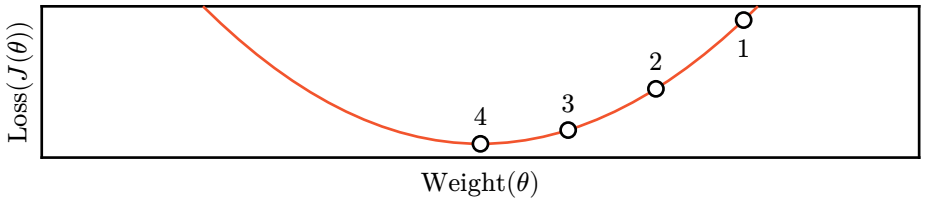

Figure 13: Gradient decent

Given the gradient, you can calculate the change in the parameter using the **learning rate**. The learning rate determines how big the steps are that you take in the direction of the gradient. Repeat this.

Backpropagation:

To implement gradient decent we need to be able to calculate the gradient of the loss with respect to all weights. This can be done using the **backpropagation** algorithm. The backpropagation algorithm works by calculating the gradient of the loss with respect to the weights of the model. It does this by using the chain rule of calculus.

Improving efficiency:

- **Batch:** All the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters.
- **Mini-batch gradient descent:** Instead of calculating the gradient on the entire dataset, we calculate the gradient on a mini-batch of the data. This makes the algorithm faster.
- **Stochastic gradient descent:** uses a single datapoint (batch with size of 1) to calculate the gradient.

Explainable AI:

We need to understand why a model makes predictions for a few reasons:

- **Preventing bias:** If we understand why a model makes predictions, we can prevent bias.
- **Verifying decisions:** If we understand why a model makes predictions, we can verify the decisions of the model.
- **Building trust:** If we understand why a model makes predictions, we can build trust in the model.
- **Debugging:** If we understand why a model makes predictions, we can debug the model.

Interpretable models can easily be understood by humans: (*decision tree*). **Black box models** are models that are hard to understand (*neural networks*). Some other models that are not explainable:

- **Random forests**
- **SVM kernels**

Model specific methods:

Interpretable models have a few properties:

- **Linearity:** The relationship between the input and output is linear.
- **Monotonicity:** The relationship between a feature and a target outcome always goes in the same direction.
- **Interaction:** Automatically includes interactions between features to predict the target outcome.

A linear model can be interpreted by looking at the weights of the model. The weights of the model tell us how much each feature contributes to the prediction.

Model-agnostic methods:

Model-agnostic methods are methods that can be used with any model. These methods work by looking at the model as a black box and trying to understand how the model makes predictions.

LIME
is a model-agnostic method that explains the predictions of any classifier. It works by training a **local model** around the instance that we want to explain. The local model is a simple model that approximates the behavior of the black box model.

Fidelity: The local model should be a good approximation of the black box model. The local model should be able to predict the same output as the black box model.

LIME steps:

- We have the **black box model** where you can input data points and get the predictions of the model.
- We can probe the box as often as you want to understand why the machine learning model made a certain prediction. LIME tests what happens to the predictions when you give variations of your data (**perturbation**) into the machine learning model.
- LIME generates a new data-set consisting of **perturbed samples** and the corresponding predictions of the black box model.
- LIME then trains an **interpretable model** on the new data-set, which is weighted by the proximity of the sampled instances to the instance of interest.

AutoML:

AutoML is a process that automates the process of applying machine learning to real-world problems. AutoML can be used to automate the process of feature engineering, model selection, hyperparameter tuning, and model evaluation.

We want to, Given a dataset D, find an algorithm A^* and its hyperparameters λ^* that minimizes a given loss function L.

The search algorithm:

The search algorithm is the algorithm that searches for the best model and hyperparameters. The search algorithm can be a grid search, random search, or a more advanced search algorithm like Bayesian optimization.

Bayesian Optimization: Bayesian optimization is a method used to optimize black-box functions. It works by modeling the objective function as a Gaussian process and using the model to find the best hyperparameters.

Search space:

The search space is the space of all possible models and hyperparameters. The search space can be discrete or continuous. The search space can be defined by the user or automatically.

Evaluation:

The evaluation of the model is the process of evaluating the performance of the model. The evaluation can be done using cross-validation or a hold-out set.

Successive Halving: A method used to evaluate multiple models in parallel. The method works by training multiple models on a subset of the data and evaluating the performance of the models. The models are then ranked based on their performance, and the best models are selected for further evaluation.

Configuration Spaces and What are Important Hyperparameters:

The configuration space is the space of all possible hyperparameters. The configuration space can be defined by the user or automatically. The configuration space can be discrete or continuous.

Metalearning:

Metalearning is a method used to learn from past experiments. Metalearning works by using the results of past experiments to guide the search for the best model and hyperparameters.