Week 1

- Week 1.1: Introduction
- Week 1.2: Regression
- Week 1.3: Methodology I
- Week 2.1: Classification I
- Week 2.2: Classification II
- Week 3.1: Methodology II
- Week 3.2: Neural networks & deep learning
- Week 4.1: Structure analysis
- Week 4.2: Density estimation
- Week 4.3: Intro to deployment

1. **What is Machine Learning?**
    - The ability to acquire knowledge, by extracting patterns from raw data
    - A set of tools for modeling and understanding complex datasets
2. **What is data?**
    - Data is the materialization of an observation or a measurement
    - Datasets are data formatted as collections of items described by a set of pre-defined attributes
3. **What is knowledge?**
    - **Proposition**(Statement) 命题
    - **Narrative**(description) 陈述
    - **Model**(mathematical or computer) 模型
4. **Notion of Machine Learning**
    - **Dataset-first view**: we start with a dataset, then produce a model
    - **Deployment-first view**: we start with a problem, then secure a dataset and finally produce a model.
    - **Definition of ML**: A set of tools together with a methodology for solving scientific, engineering and business problems using data.
5. **Two Stages of Machine Learning**
    - Learning Stage: the model is built
    - Deployment stage: the model is used
6. **ML Basic methodology:**
    - Training: a model is created using data and a quality metric
    - Testing: The performance of the model during deployment is assessed using new, unseen data
7. **Machine Learning task taxonomy**
    1. Supervised Learning:
        - **Definition**: Build a model that maps attributes x, known as the predictor, to another attribute y, which we call the label, using a dataset of labelled examples
        - **Classification**: The label is a discrete variable
        - **Regression**: The label is a continuous variable
    2. Unsupervised Learning:
        - **Definition**: set out to find the underlying structure of our dataset
        - **Structure Analysis:** Study the underlying structure of a dataset
            - Cluster analysis: Focus on groups of data points
            - Component analysis: Identify directions of interest

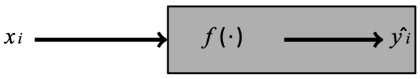
- **Density Estimation**: provide statistical models that describe the distribution of samples in the attribute space.

1.1 Regression

Definition： Regression is a supervised problem, our goal is to predict the value of one attribute(label) using the remaining attributes(predictors)

Goal: our goal is to find the **best model** that assigns a unique label to a given set of predictors

- Our ability to build predictors is due to association between attributes, rather than causation. Two attributes in a dataset appear associated:
  - One casue the other
  - Both have a common cause



Population:
- x is the **predictor** attribute
- y is the **label** attribute

Dataset:
- N is the number of samples, i identifies each sample
- x_i is the predictor of sample i
- y_i is the actual label of sample i
- (x_i, y_i) is sample i, $\{(x_i, y_i) : 1 \le i \le N\}$ is the entire dataset

Model:
- $f(\cdot)$ denotes the model
- $\hat{y_i} = f(x_i)$ is the **predicted label** for sample i
- $y_i - \hat{y_i}$ is the **prediction error** for sample i

Model quality metrics

- **SSE**(Sum of squared errors)

  $E_{SSE} = e_1^2 + e_2^2 + \cdots + e_N^2 = \sum_{i=1}^{N} e_i^2$
- **MSE**(mean squared errors)

  $E_{MSE} = \frac{1}{N} \sum_{i=1}^{N} e_i^2$
- **RMSE**

$$E_{RMSE} = \sqrt{\frac{1}{N} \sum e_i^2} \tag{1}$$

- **MAE**

$$E_{MAE} = \frac{1}{N} \sum |e_i| \tag{2}$$

- **R-squared**

$$E_R = 1 - \frac{\sum e_i^2}{\sum (y_i - \bar{y})^2}, \text{ where } \bar{y} = \frac{1}{N} \sum y_i \tag{3}$$

Nature of error

- The chosen of predictors might not include all the factors that determine the label

- The model might not be able to represent the true relationships between predictor and label
- Random noise may be present

Design Matrix and label vector

In **multiple linear regression**, the training dataset can be represented by the design matrix X:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,K} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,K} \end{bmatrix} \tag{4}$$

and the label vector $\mathbf{y}$:

$$\mathbf{y} = [y_1, \ldots, y_N]^T = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{5}$$

Least squares solution

The linear model that minimises the metric E_{MSE} on a training dataset defined by a design matrix X and a label vector y, has the parameter vector:

$$\mathbf{w}_{best} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{6}$$

Logistic function:

$p(d) = \frac{e^d}{1+e^d} = \frac{1}{1+e^{-d}}$

By using $d = w_0 + w_1 x$, we can translate the logistic function and change its gradient.

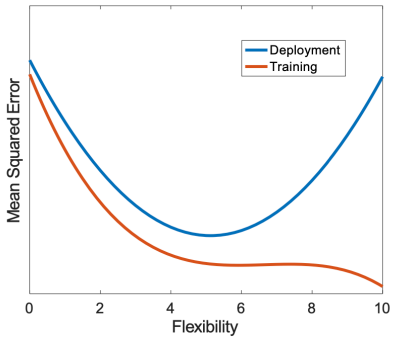
Flexibility & Interpretability

trade-off

Generalization

Ability of our model to successfully translate what we was learnt during the learning stage to deployment.

TIP: Generalisation can only be assessed by comparing training and deployment performance, not by just looking at how each model fits the training data.



- **Underfitting:** The model is unable to capture the underlying pattern
- **Overfitting**:The model is memorizing irrelevant details
- **Just Right**: The model is capable of reproducing the underlying pattern and ignores irrelevant details

1.2 Classification

- **Definition**: In classification(also known as decision or detection) problems, we want to build a model that produces a label when shown a set of predictors, and the label is discrete and each of its values is known as a class.
- **Binary classification**: A classification problem with only two classes. For example, sentiment analysis, which seeks to identify human opinions implied by a fragment of text, multiple opinions can be considered, but in its simplest form two are defined, namely positive and negative.
- **Multiclass classification**: A classification problem with more than two classes. For example, recognise digits in images containing handwritten representations is a classic multiclass classification problem, the predictor is an array of values (image) and there are 10 classes, namely 0, 1, 2, ... 9.
- **Representation of classification model**
  - **Dataset in the attribute space:** labels can be represented by numerical values on a vertical axis, be careful, the usual notions of ordering and distance do not apply to categorical variables.
  - **Dataset in the predictor space:** a more convenient representation consists of using different symbols for each label in the predictor space.
  - **Classifier:** can be represented as decision regions in the predictor space, a decision region is made up of points that are associated to the same label, regions can be defined by identifying their boundaries, and a solution model in classification is a partition of the predictor space into decision regions separated by decision boundaries.
- **Linear classifiers**: Classifiers that use linear boundaries between decision regions.
  - **Definition**: Linear boundaries are defined by the linear equation:

    $$\mathbf{w}^T\mathbf{x} = 0 \tag{7}$$

    The extended vector:

    $$\mathbf{x} = [1, x_1, x_2, \ldots]^T \tag{8}$$

    contains the predictors and w is the coefficients vector. To classify a sample we simply identify the side of the boundary where it lies.
  - **Model quality metrics**:
    - **Accuracy**:

      $$A = \frac{\#\text{correctly classif. samples}}{\#\text{samples}} \tag{9}$$

    - **Error**(misclassification) rate:

      $$E = 1 - A = \frac{\#\text{incorrectly classif. samples}}{\#\text{samples}} \tag{10}$$

    - **Linearly separable case**: the maximum accuracy ($A = 1, E = 0$) can be achieved.
    - **Non-linearly separable case**: the accuracy of a linear classifier is $A < 1$ ($E > 0$). The best one will achieve the highest accuracy.

- **Logistic model**: Given a linear boundary w and a predictor vector $\mathbf{x}_i$, the quantity $\mathbf{w}^T\mathbf{x}_i$ can be interpreted as the distance from the sample to the boundary. If we set $d = \mathbf{w}^T\mathbf{x}_i$ in the logistic function, we get:

  $$p(\mathbf{w}^T\mathbf{x}_i) = \frac{e^{\mathbf{w}^T\mathbf{x}_i}}{1 + e^{\mathbf{w}^T\mathbf{x}_i}} \tag{11}$$

  For a fixed w, we will simply denote it as $p(\mathbf{x}_i)$ to simplify the notation.
  - When $\mathbf{w}^T\mathbf{x} \to \infty$, the logistic function $p(\mathbf{x}_i) \to 1$
  - When $\mathbf{w}^T\mathbf{x} \to -\infty$, the logistic function $p(\mathbf{x}_i) \to 0$
  - We will use the logistic function to quantify the notion of certainty in classifiers. This certainty is a quantity between 0 and 1.
  - $p(\mathbf{x}_i)$ is the classifier's certainty that $y_i =$ o is true.
  - $1 - p(\mathbf{x}_i)$ is the classifier's certainty that $y_i =$ x is true.
  - The certainty for a labelled dataset $(\mathbf{x}_i, y_i)$ is:

    $$L = \prod_{y_i=\text{o}} (1 - p(\mathbf{x}_i)) \prod_{y_i=\text{x}} p(\mathbf{x}_i) \tag{12}$$

    L is known as the **likelihood function** and defines a quality metric. Taking logarithms, we obtain the **log-likelihood**:

$$l = \sum_{y_i=\text{o}} \log[1 - p(\mathbf{x}_i)] + \sum_{y_i=\text{x}} \log[p(\mathbf{x}_i)] \tag{13}$$

The linear classifier that maximises L or l is known as the **Logistic Regression** classifier. It can be found using **gradient descent**.

- **Nearest neighbours**:
  - **Definition**: In nearest neighbours (NN), new samples are assigned the label of the closest (most similar) training sample.
  - **Characteristics**:
    - Boundaries are not defined explicitly (although they exist and can be obtained).
    - The whole training dataset needs to be memorised. That's why sometimes we say NN is an instance-based method.
  - **k Nearest Neighbours**:
    - **Definition**: kNN is a simple extension of NN that proceeds as follows. Given a new sample x:
      1. We calculate the distance to all the training samples $\mathbf{x}_i$.
      2. Extract the K closest samples (neighbours).
      3. Obtain the number of neighbours that belong to each class.
      4. Assign the label of the most popular class among the neighbours.
    - **Characteristics**:
      - There is always an implicit boundary, although it is not used to classify new samples.
      - **As K increases, the boundary becomes less complex. We move away from overfitting (small K) to underfitting (large K) classifiers.**
      - In binary problems, the value of K is usually an odd number. The idea is to prevent situations where half of the nearest neighbours of a sample belong to each class.
      - kNN can be easily implemented in multi-class scenarios.
- **The Bayes classifier**
  - Definition: The classifier that achieves the highest accuracy is the one that compares the posterior probabilities, defined as the probability that a sample belongs to a class given the value of its predictors, e.g.
  - $P(y = o|x = a) \leq P(y = x|x = a)$
  - Don't confuse a posterior probability with a class density!
  - The classifier that uses the true posterior probabilities is called the **Bayes classifier**. The Bayes classifier uses the **odds ratio**:
    - $\frac{P(y=o|x)}{P(y=x|x)} \leq 1$
    - If the ratio is greater than 1, the sample is o, if it is less, it is x. This is equivalent to assigning the sample to the most probable class.
- **Bayes rule**
  - If we happen to know the priors and the class densities, we can apply Bayes rule to obtain the posterior probabilities exactly:
    - $P(y = o|x) = \frac{p(x|y=o)P(y=o)}{p(x)}, P(y = x|x) = \frac{p(x|y=x)P(y=x)}{p(x)}$
    - The odds ratio can be expressed using the priors and the class densities:
    - $\frac{P(y=o|x)}{P(y=x|x)} = \frac{p(x|y=o)P(y=o)}{p(x|y=x)P(y=x)} \leq 1$
  - Hence, the problem of building posterior probabilities is equivalent to the problem of building priors and class densities.
- **Discriminant analysis**
  - In discriminant analysis, we assume that the class densities are Gaussian. If there is one predictor x, the o class density is:
    - $p(x|y = o) = \frac{1}{\sigma_o\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_o}{\sigma_o})^2}$
    - where μ_o is the mean and σ_o^2 the variance of the Gaussian density.
  - If there are K predictors, a Gaussian class density is expressed as:
    - $p(x|y = o) = \frac{1}{(2\pi)^{p/2}|\Sigma_o|^{1/2}} e^{-\frac{1}{2}(x-\mu_o)^T\Sigma_o(x-\mu_o)}$
    - we $x = [x_1, \ldots, x_K]^T$ contains all the predictors (note we have not prepended a 1), μ_o is the mean and Σ_o is the covariance matrix.
  - Similar expressions can be obtained for the x class densities.
  - The boundary in discriminant analysis depends on the covariance matrices Σ_o and Σ_x:
    - If $\Sigma_o = \Sigma_x$ the boundary is linear. We call this scenario **linear discriminant analysis (LDA)**.
    - **Otherwise, the boundary is quadratic. This is quadratic discriminant analysis (QDA).**
- **Beyond accuracy**

- **Class-sensitive costs**:
  - To account for misclassification costs, we can use the following comparison instead:
  - $\frac{C_o \times P(y=x|x)}{C_x \times P(y=o|x)} \leq 1$ or $\frac{P(y=x|x)}{P(y=o|x)} \leq \frac{C_x}{C_o}$
  - A classifier that follows this strategy will minimise the expected cost, rather than the accuracy.
  - In general, our Bayesian extension will be expressed as:
  - $\frac{P(y=o|x)}{P(y=x|x)} \leq T$
  - where T is a threshold value corresponding to the ration of misclassification costs.
- **Confusion matrix**:
  - In addition to using the misclassification cost for each class, we can assess how well the classifier deals with each class individually. This is precisely the information that a confusion or contingency matrix shows.

		Actual class	
		Positive	Negative
Predicted class	Positive	True positive	False positive
	Negative	False negative	True negative

  - **Detection problems**:
    - Many binary problems consider classes that represent the presence or absence of some property. For these problems, it is common to use the terms positive (presence) and negative (absence) for each class.
    - In addition, we use the terms:
      - **True positive (TP)** and **true positive rate (TPR)**.
      - **False negative (FN)** and **false negative rate (FNR)**.
      - **False positive (FP)** and **false positive rate (FPR)**.
      - **True negative (TN)** and **true negative rate (TNR)**.
- **Class-sensitive rates**:
  - In detection problems, the most commonly rates used are:
    - **Sensitivity** (recall or true positive rate): $TP/(TP+FN)$
    - **Specificity** (true negative rate): $TN/(TN+FP)$
    - **Precision** (positive predictive value): $TP/(TP+FP)$
  - These rates can be used as quality metrics.
  - The **F1-score** is another widely used performance metric that provides an average between precision and recall:
    - $F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
- **The ROC plane**
  - The **ROC (receiver operating characteristic)** plane is used to represent the performance of a classifier in terms of its sensitivity and 1-specificity.
  - The **area under the curve (AUC)** is a measure of goodness for a classifier that can be calibrated.

2.1 Structure Analysis

Cluster analysis

- **Definition**: Clustering is a family of unsupervised learning algorithms that describe the structure of a dataset as groups, or clusters, of similar samples.
- A notion of **similarity** is needed in order for us to partition a dataset into clusters.

Similarity as proximity

- Clusters can be defined as groups of samples that are **close** to one another. In this case, we use proximity as our notion of similarity.

- Mathematically, there are different ways of defining a distance. Given two samples $\mathbf{x}_i$ and $\mathbf{x}_j$ consisting of P attributes, $x_{i,1}$, ..., $x_{i,P}$ and $x_{j,1}$, ..., $x_{j,P}$, the **squared distance** $d_{i,j}$ is defined as

$$d_{i,j} = (x_{i,1} - x_{j,1})^2 + \cdots + (x_{i,P} - x_{j,P})^2 \tag{14}$$

A proximity-based quality metric

Using distance as our notion of similarity, samples within the same cluster should be close to one another and samples from different clusters should be far apart.

Assume we have two clusters C_0 and C_1. The **intra-cluster** sample scatter $I(C_0)$ and $I(C_1)$ is the sum of the square distances between samples in the same cluster:

$$I(C_0) = \frac{1}{2} \sum_{\mathbf{x}_i, \mathbf{x}_j \text{ in } C_0} d_{i,j}, \quad I(C_1) = \frac{1}{2} \sum_{\mathbf{x}_i, \mathbf{x}_j \text{ in } C_1} d_{i,j} \tag{15}$$

and the **inter-cluster** sample scatter $O(C_0, C_1)$ is defined as the sum of the distances between samples in different clusters:

$$O(C_0, C_1) = \sum_{\mathbf{x}_i \text{ in } C_0, \mathbf{x}_j \text{ in } C_1} d_{i,j} \tag{16}$$

The **best** clustering arrangement has the **lowest intra-cluster** sample scatter and **highest inter-cluster** sample scatter. We can show that reducing the intra-cluster scatter increases the inter-cluster scatter!

The intra-cluster and inter-cluster sample scatters allow us to compare and rank different clustering arrangements.

The question is, can we create an algorithm capable of automatically identifying the **best clustering** arrangement? This is an **optimisation** question.

K-means clustering: Prototypes

- **Definition**: A simple way to describe a cluster is by using **cluster prototypes**, such as the centre of a cluster.
- Given a cluster C_0 consisting of N_0 samples, its centre (or mean) μ_0 can be calculated as:

$$\mu_0 = \frac{1}{N_0} \sum_{\mathbf{x}_i \text{ in } C_0} \mathbf{x}_i \tag{17}$$

- Interestingly, the intra-cluster sample scatter can be calculated using the distance between each sample and the cluster prototype d_i:

$$I(C_0) = N_0 \sum_{\mathbf{x}_i \text{ in } C_0} d_i \tag{18}$$

- Therefore, our notion of clustering quality can be expressed as follows: in a good clustering arrangement, samples are **close to their prototype**.

K-means clustering

K-means partitions a dataset into K clusters represented by their **mean** and proceeds iteratively as follows:

- Prototypes are obtained as the centre (or mean) of each cluster.
- Samples are re-assigned to the cluster with the closest prototype.

As the K-means algorithm proceeds, we will see samples been reassigned to different clusters until at some point we reach a stable solution, where no sample is reassigned.

The final solution is a **local optimum**, not necessarily the global one.

How many clusters?

K-means requires that we specify the number of clusters K that we want to partition our dataset into.

A natural question is, what's the right number of clusters? The answer to this question depends on the application:

- In some cases we are given the number of clusters (e.g. t-shirt sizes).
- In a discovery scenario we want to find the underlying structure and the number of clusters is unknown.

Validation strategies can suggest a suitable value for the hyperparameter K. Choosing the value of K producing the lowest $I(C_0)$ would not work however, as $I(C_0)$ always decreases as the number of clusters increase.

The elbow method

Assume the true number of clusters is K_T. For $K > K_T$, we should expect the increase in quality to be slower than for $K < K_T$, as we will be splitting true clusters.

The true number of clusters can be identified by observing the value of K beyond which the improvement slows down.

Density-based clustering: DBSCAN

In a **non-convex** cluster, we can reach any sample by taking small jumps from sample to sample.

Non-convex scenarios suggest a different notion of cluster as group of samples that are **connected**, rather than simply close: *if I am similar to you, and you are similar to them, I am similar to them too*.

This notion of cluster as a group of connected samples is behind many clustering algorithms, such as DBSCAN (**density-based spatial clustering of applications with noise**).

DBSCAN belongs to the family of **density-based** algorithms, where an estimation of the density of samples around each sample is used to partition the dataset into clusters.

DBSCAN

DBSCAN defines two quantities, a **radius** r and a **threshold** t. A density is first calculated as the number of samples in a neighbourhood of radius r around each sample (excluding itself). Then, three types of samples are identified:

- **Core**: its density is equal or higher than the threshold t.
- **Border**: its density is lower than the threshold t, but contains a core sample within its neighbourhood.
- **Outlier**: Any other sample.

The DBSCAN algorithm proceeds as follows:

- Identify core, border and outlier samples.
- Pair of core samples that are within each other's neighbourhood are connected. Connected core samples form the **backbone** of a cluster.
- Border samples are assigned to the cluster that has more core samples in the neighbourhood of the border sample.
- Outlier samples are not assigned to any cluster.

Hierarchical clustering

Given a dataset consisting of N samples, there exist two **trivial** clustering solutions: **one single cluster** that includes all the samples, and the solution where **each sample is a cluster** on its own.

K-means produces K clusters, but we need to choose K within $1 \leq K \leq N$. In DBSCAN clusters are discovered automatically, but the final number of clusters depends on the values of the radius r and the threshold value t.

This ambiguity ultimately reveals that **the structure of a dataset can be explored at different levels that expose different properties**.

Hierarchical clustering is a family of clustering approaches that proceed by progressively building clustering arrangements at **different levels**.

The resulting collection of clustering arrangements is hierarchical in the sense that a cluster in one level contains all the samples from one or more clusters in the level below.

The representation of the relationship between clusters at different levels is called a **dendrogram**. At the bottom we find the arrangement where each sample is one cluster and at the top, the whole dataset.

There exist two basic strategies to build a dendrogram:

- The **divisive** or top-down approach splits clusters starting from the top of the dendrogram and stops at the bottom level.
- The **agglomerative** or bottom-up merges two clusters, starting from the bottom until we reach the top level.

There are different options to decide which clusters to merge or split at each level. Common strategies in agglomerative clustering include:

- **Single linkage**: uses the distance between the two closest samples from two clusters. This option results in clusters of arbitrary shapes.
- **Complete linkage**: uses the distance between the two further samples from each pair of clusters. This choice produces clusters that tend to have a spherical shape.
- **Group average**: uses the average distance between samples in two cluster and also produces spherical shapes, although they are more robust to outliers.

Summary

- Unsupervised learning provides with answers for the basic question **where is my data?** (in the attribute space)

- Our answer is a mathematical/computer model. This model can tell us where in the space we have samples (clustering) or the probability to find a sample in a region within the space (density estimation).
- Sometimes we say that our data is unlabelled. What we **really** mean is that we don't treat any attribute as a label that we want to predict. Datasets are neither labelled nor unlabelled.
- Lacking such a target as a label means that **our quality metric is not obvious**.

Clustering

- **K-means** is a prototype-based clustering that produces spherical clusters where K is a hyperparameter that has to be set.
- **DBSCAN** is a density-based option suitable for non-convex scenarios and does not require specifying the number of clusters. We need to set r and t and they determine the final number of clusters.
- **Hierarchical clustering** allows to explore the structure of a dataset at multiple levels.

Comparing clustering solutions

- We could consider **comparing** the solutions from two different algorithms. However, if they use different definitions of clustering quality, this comparisons will make little sense.
- Clustering is ultimately implemented with an **application** in mind so we should create a final notion of clustering quality based on the specific goals of the application.

Component analysis
Component analysis allows us to identify the **directions in the space** that our data are aligned with. This can be useful to transform our dataset, clean it and reduce its dimensionality.

2.2 Density Estimation

Definition: Density estimation is an unsupervised learning problem that aims to model the underlying probability density function that governs the distribution of data points in a given dataset. It involves estimating the probability of observing data points within specific regions of the attribute space.

- **Probability densities** are models that describe the underlying true distribution of our data. They allow us to quantify:
  - The probability of extracting a sample within a region of the space.
  - The fraction of population samples that lie within a region of the space.
- **Divide and count**

  Partitioning the space into smaller regions and counting the number of samples within each region is a simple way of describing the observed distribution.

  It also gives an indication of what to expect if we extract more samples from the same population, i.e. the true distribution. Counts will not give useful quantitative answers, but we can transform them into rates.
- **Non-parametric methods**
  - **Definition:** Non-parametric methods for density estimation do not assume any specific shape for the probability density.
  - **The histogram** is the simplest and best known non-parametric method for density estimation. A histogram is built by dividing the feature space into equal-sized regions called **bins**. The density is approximated by the fraction of samples that fall within each bin. If bins are small, the estimated density will be **spiky**. If they are big, the estimated density will be **flat** and the underlying structure will be lost.
  - **Kernel methods** proceed by building an individual density around each sample first and then combining all the densities together. Individual densities have the same shape (the kernel), for instance a Gaussian.
- **Parametric density estimation**
  - **Definition:** Parametric approaches **specify the shape** of the probability density. The problem of density estimation consists of **estimating its parameters**.
  - The **Gaussian distribution**, usually denoted by $\mathcal{N}(\mu, \Sigma)$.
  - The Gaussian or normal distribution $\mathcal{N}(\mu, \sigma)$ is defined by two parameters μ and σ describing its **location** and **width**. Its mathematical expression in a 1D attribute spaces is:

    $$p(x_1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x_1-\mu}{\sigma})^2} \tag{19}$$

    μ is known as the **mean** and σ is the **standard deviation**. The value σ^2 is known as the **variance**.

- The **Central Limit Theorem (CLT)** states that the **sum** of a large number of independent, random quantities has a **Gaussian** distribution. The CLT is perhaps the **main reason** why the Gaussian distribution is one of our favourite density models.
- The Gaussian distribution can be extended to 2D, 3D... attribute spaces:

$$p(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{20}$$

  Where $x = [x_1, \ldots, x_P]^T$ contains all the attributes. The mean μ and covariance matrix Σ describe the **position** and **shape** of the distribution.
- Given a Gaussian distribution $p(x_1, x_2)$, the following statements are equivalent:
  - Attributes x_1 and x_2 are **independent** (e.g. we cannot predict the value of one based on the other).
  - The covariance matrix Σ is **diagonal**.
  - $p(x_1, x_2)$ can be obtained as the **product** of the **marginal densities** $p(x_1)$ and $p(x_2)$, which are themselves Gaussian.
- If we are given a dataset consisting of N samples x_i, the parameters of a Gaussian distribution can be estimated using **maximum likelihood** approaches.
  - In 1D attribute spaces, the parameters μ and σ^2 can be estimated as:

$$\hat{\mu} = \frac{1}{N} \sum_i x_i, \text{ and } \hat{\sigma}^2 = \frac{1}{N} \sum_i (x_i - \hat{\mu})^2 \tag{21}$$

  - In higher dimensional spaces, μ and Σ are estimated as:

$$\hat{\mu} = \frac{1}{N} \sum_i x_i, \text{ and } \hat{\Sigma} = \frac{1}{N} \sum_i (x_i - \hat{\mu})(x_i - \hat{\mu})^T \tag{22}$$

- **Mixture models**: Datasets can exhibit more than one mode (clumps) for which single Gaussian densities are not suitable. In such cases, mixture densities such as **Gaussian Mixture Models (GMM)** constitute a convenient choice. A GMM probability density is formulated as a **combination** of Gaussian densities $g_m(x)$ with their own mean μ_m and covariance matrix Σ_m:

$$p(x) = \sum_m g_m(x) \pi_m \tag{23}$$

  where π_m are the mixing coefficients.

- **Applications**
  - **Noise and outliers**: Samples extracted from the same population will always exhibit some level of randomness and deviate from the underlying pattern. Such deviations are known as noise. Sometimes a sample can be so different that we doubt its deviation is just due to noise. We call these samples outliers or anomalies. Outliers are samples that belong to a different population altogether.
  - **Basic anomaly detection algorithm**: The main idea behind an anomaly detection algorithm is to quantify the probability of observing samples some distance away from the general pattern. If this probability is low, the sample is an anomaly. The main pattern is described by a probability density $p(x)$. If $p(x)$ is a multivariate Gaussian distribution, we proceed as follows:
    - Estimate μ and Σ from the dataset.
    - Agree on a threshold value T.
    - If $p(x_i) < T$, x_i is an anomaly.
  - **Classification: Estimating class densities**: Classifiers that apply Bayes rule turn posterior probabilities into priors and class densities. A class density $p(x|C)$ describes the distribution of samples in the predictor space for each class C. Class densities are obtained using density estimation methods. We need to fit a probability distribution for each class separately.
  - **Naive Bayes classifiers**: Gaussian distributions are the most popular choice for class densities. In high-dimensional scenarios, the total number of parameters is very large: for P predictors, we have P (mean) + P^2 (covariance) parameters. Naive Bayes classifiers make the (naive) assumption that predictors are independent, hence a P-dimensional Gaussian distribution can be expressed as the product of its P marginal distributions. As a result of this additional constraint, we need to obtain P (means) + P (variances) parameters, which reduces the risk of overfitting.
  - **Clustering**: K-means can be seen as a version of GMM fitting, where the Gaussian distributions have the same diagonal covariance matrix and hence clusters tend to be spherical. GMM can be used as a clustering method that produces ellipsoidal clusters. First we fit K Gaussian densities and then we assign each sample to the most likely density.
- **Summary**
  - Probability densities are models that allow us to calculate the probability of finding a sample in a region of the attribute space.
  - Non-parametric methods do not assume any particular shape for the probability density, whereas parametric methods do.
  - The Central Limit Theorem and other mathematical properties, make the Gaussian distribution one of the most popular choices.
  - Probability densities can be used in many machine learning problems, such as anomaly detection, classification and clustering.

3.1 Deep Learning and Neural Network

What is neural network?

- **Definition**: A neural network is a computing system loosely inspired by the human nervous system, commonly used as a family of Machine Learning models.
- From a **functional** point of view, a neural network:
  - Produces an **output** (label) given an **input** (predictors), .
  - Has **weights** (parameters) that can be **tuned** (trained).

$$\mathbf{W} \text{ denotes all the weights of the neural network and } h_{\mathbf{W}}(\mathbf{x}) \text{ indicates explicytely that the prediction } \hat{y} \text{ depends on } \mathbf{W}. \quad (24)$$

- From an **computational** angle, neural networks consist of interconnected units that (loosely) mimmic **neurons**. This architecture is appealing since:
  - **Neuroscience** suggests biological neural networks can solve any problem.
  - **Mathematics** suggests artificial neural networks can reproduce any input/output relationship, provided they are complex enough.
- Hence, the family of neural network models can be seen as a **universal machine**.

The perceptron

- **Definition**: The perceptron is the **basic unit** of a neural network. It is defined by a **weight vector** $\mathbf{w}$ and an **activation function** $h(\cdot)$ that map an extended vector $\mathbf{x}$ to an output a. The coefficient w_0 is known as the **bias**.
  $a = h(\mathbf{w}^T\mathbf{x})$
- The activation function is **non-linear**.

Layers

- **Definition**: A **layer** is a collection of perceptrons that use the **same input**. Each perceptron within a layer produces a separate output.
- A layer consisting of L perceptrons and K inputs has $L \times (K+1)$ weights.

The architecture

- **Definition**: The **architecture** of a neural network describes how layers are connected. The **input layer** is the predictor vector, **hidden layers** produce internal features and the **output layer** produces the prediction.
- Vector $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_M]^T = h_{\mathbf{W}}(\mathbf{x})$ is the output of the neural network. (Note that here $\hat{y}_j$ is not the prediction for the j-th sample!)

The neural network

- A neural network consists of **perceptrons** arranged in **layers** that are connected according to an **architecture**.
- There is one **parameter** per connection (the connections's **weight**).
- Each layer produces intermediate **features**.
- The number of layers determines the **depth** of the neural network (hence the distinction between **shallow** and **deep** neural networks).

The perceptron as a linear classifier

- A perceptron that uses a **step** activation function is actually a **linear classifier**.
- If it uses a **logistic** function, the perceptron is a **logistic classifier**.

$$h_{step}(d) = \begin{cases} 1, & \text{if } d > 0. \\ 0, & \text{otherwise.} \end{cases}$$

The perceptron as a basic logical function

- We can use perceptrons to implement **logical functions**.

Adding depth to derive new logical functions

- We can add depth to derive new logical functions.

The perceptron as a grid pattern detector

- We can use perceptrons to detect grid patterns.

Adding depth to detect derived grid patterns

- We can add depth to detect derived grid patterns.

Combining linear classifiers and logical functions

- We can combine linear classifiers and logical functions to build a neural network classifier.

Versatility of neural networks as a computing system

- Using a single perceptron we can implement:
  - **Linear boundaries**.
  - **Logical functions**.
  - **Grid pattern detectors**.
- Connecting perceptrons allows us to implement more complex operations, e.g. complex boundaries, complex logical functions and complex grid patterns. This can be seen as building complexity from basic operations.
- So far we have seen neural networks from a computation angle and have specified each network by ourselves. Note this is **not** machine learning!

Machine learning pipelines

- A machine learning pipeline is a sequence of data operations.

$$\mathbf{x}_i \rightarrow \boxed{T} \rightarrow \mathbf{z}_i \rightarrow \boxed{f} \rightarrow \hat{y}_i \tag{25}$$

- A simple pipeline consists of a first transformation stage followed by a machine learning model:
  - The transformation stage produces **derived features**.
  - The model uses the derived features as input.
  - The transformation stage can also be trained.

Neural networks as tunable pipelines

- A neural network can be seen as an entire tunable machine learning pipeline, where:
  - Each perceptron defines one **derived feature** or **concept** using other features, raw or derived.
  - Increasing the **number of perceptrons** in a layer allows to create more new concepts per layer.
  - Increasing the **number of layers** (deeper networks) allows to create concepts of increasing complexity.

Deep neural networks: Computational angle

- From a cognitive point of view, large neural networks are appealing, as they give us the necessary **flexibility** to create **new** and **increasingly complex concepts** that might be relevant to make a prediction.
- However:
  - Higher flexibility increases the risk of **overfitting** (which we can see as a network creating and using irrelevant concepts).
  - Large number of parameters need to be tuned, therefore the **computational requirements** might be too high.
- For **complex inputs**, such as pictures consisting of millions of pixels, this is even more severe.

Training neural networks: Cost function

- Every Machine Learning algorithm needs a **model**, a **cost function** and an **optimisation** method.
- Given a dataset $\{(\mathbf{x}_i, y_i), 1 \leq i \leq N\}$, where labels can take on the values 0 or 1, a common cost function for classification is the **negative log-likelihood function**, defined as:

$$l(\mathbf{W}) = -\frac{1}{N}\sum_{n=1}^{N} y_i \log[\hat{y}_i] + (1-y_i)\log[1-\hat{y}_i] \tag{26}$$

  where $\hat{y}_i = h_{\mathbf{W}}(\mathbf{x}_i)$. This cost function can be extended to multi-class classifiers.

Gradient descent and back-propagation

- **Gradient descent** is the method of choice to find the optimal set of coefficients $\mathbf{W}$ for the cost function $l(\mathbf{W})$.
- Obtaining the gradient is easy, but can be computationally expensive. **Back-propagation** is an efficient algorithm to **compute the gradient**. This gradient is then used by the optimisation algorithm to update $\mathbf{W}$.

- Back-propagation exploits the **chain rule of calculus**. It turns out that to compute the gradient in one layer, we just need information from the next layer. Back-propagation starts from the output: it obtains the cost and proceeds backwards calculating the gradients of the hidden units.

Training considerations

- **Initialisation**: If the initial weights are zero, back-propagation fails. Initial weight values should be random.
- **Overfitting**: Neural networks can have millions of parameters. Use regularisation and validation-based early stop to avoid overfitting.
- **Non-convexity**: The cost function has multiple local minima. Retrain from different random starting values.
- **Scaling of the inputs**: Range of input values can affect the values of the weights. Standardise to ensure inputs are treated equally.
- **Architecture**: Different architectures suit different problems.

Transfer learning

- A neural network that has been successfully trained for a problem A can be reused for a related problem B, for instance:
  - We can leave the early stages **unchanged**, becoming a fixed transformation stage $T(\mathbf{x})$.
  - We can **retrain** the late stages using new data $f(\mathbf{z})$.

$$\mathbf{x}_i \rightarrow \boxed{T} \rightarrow \mathbf{z}_i \rightarrow \boxed{f} \rightarrow y_i \tag{27}$$

- We are in essence **transferring an already learnt transformation** and reusing it for a different problem:
  - No need to train $T(\mathbf{x})$ (same parameters for problems A and B).
  - The optimal parameters of $f(\mathbf{z})$ for problem B will be close to the ones found for problem A (shorter training time!).

Fully-connected layer

- So far we have considered **fully-connected** (FC) layers are, where all the perceptrons receive all the outputs from the previous layer. FC layers have a **large number of parameters** and training them can be challenging.

Equivariance in grid data

- Images and time series are complex data types consisting of individual attributes associated to a **regular grid** defining a **spatial relationship**.
- Some grid data exhibit the **equivariance** property, according to which the same pattern can be expected in different locations of the grid.

Equivariance in grid data: FC layer

- How would a FC layer identify a short horizontal segment in the following 8×8 grid?
  - Each perceptrion connected to all inputs: 8×8+1 weights.
  - As many perceptrons as potential locations, L.
  - Total of $L \times (8 \times 8 + 1)$ weights.

Equivariance in grid data: The convolutional layer

- Convolutional layers impose additional restrictions:
  - Perceptrons are arranged as a grid known as **feature map**,
  - focus on different **limited regions** in the input grid and
  - **share** their parameters, represented as a grid called **kernel**.
- The feature map is efficiently calculated as a **convolution** of the kernel and the input or in other words, **filtering** the input with the kernel.

Convolutional layers

- Convolutional layers can have **several feature maps**, each of which is associated to a **different concept**. They form a **stack** of maps.
- The **dimensions of a kernel** are $H \times W \times D$, there H is the height, W is the width and D is the depth (number of input feature maps).
- The total number of **weights per kernel** is $H \times W \times D + 1$ (bias).
- **Training** a convolutional layer means using data to tune the weights of each kernel.

Pooling layers

- Pooling **reduces the size** of feature maps. Pooling layers are defined by size of the area they are reducing to a single number and are inserted between successive convolutional layers.
- They come in two flavours:
  - **Max pooling**: The output is the largest value within the filter area.

    - **Average pooling**: The output is the average of the values within the filter area.
- Note that pooling layers do not need to be trained!

Deep learning architectures

- Deep neural networks are **not arbitrary** sequences of arbitrary layers. On the contrary, they have a **predefined architecture** that is suitable for a specific goal.
- In classification, it is common to see architectures in which:
    - The first layers define a **few, simple** concepts, the last layers define **many, complex** concepts.
    - Feature maps **shrink** as we move deeper into the network.
- The same intermediate concepts can be useful for different goals. We can use **transfer learning** to reuse existing solutions.

Example: The VGG16 network

- VGG16 was designed for the ImageNet Large-Scale Visual Recognition Challenge (dataset of images belonging to 1000 classes).
- VGG16 has 16 layers, 3×3 kernels and 138 million weights.

Neural networks: the three views

- We can approach neural networks from three angles:
    - **Functional**: system that **maps** an input $\mathbf{x}$ to an output $\hat{y}$. By tuning its set of parameters $\mathbf{W}$, we change the mapping.
    - **Cognitive**: system that **creates new concepts** from raw data and other concepts. By tuning $\mathbf{W}$, we create different concepts.
    - **Computational**: **pipeline** of interconnected computing units called **perceptrons**. By tuning $\mathbf{W}$, we change the computation.

Neural networks: A family of machine learning models

- Neural networks should not be seen as a machine learning model in the same sense as, for instance, a logistic regression classifier.
    - Neural networks are a **family** of machine learning models.
    - Each neural network has an **architecture**, the simplest one of which is one single perceptron.
    - Some architectures are **not substantially different** from other machine learning models (e.g. the perceptron is a linear classifier).
    - We can see neural networks as a **framework** to create new models.
    - We **train** one specific architecture and can use **validation** approaches to choose the right architecture.
- We should **avoid** saying *neural networks perform well for a given problem*. Instead, we should say *this neural network architecture performs well*.

Neural networks: Universal machines

- For any problem, we can find a neural network architecture that solves it. In this sense, neural networks are said to be universal machines.
    - Note that this does **not** mean that a specific neural network architecture can solve any problem.
    - The existence of a suitable neural network architecture does **not** imply that we will be able to find it.
    - Flexibility is achieved adding **complexity**, which increases the risk of **overfitting**.
    - **Neural network experts** design the right architecture for a problem and reuse existing solutions using the principles of transfer learning.
    - **Neural network brutes** are unaware of the Monkey Theorem and use all the computational power and data available to train as many complex architectures as possible. Their carbon footprint is huge.

0.1 Methodology I

What's different about machine learning?

- Machine learning aims at building solutions that work well on **samples** coming from a **target population**.
- In machine learning, we **lack a description** of the target population.
- All we can do is extract samples from the population (known as **sampling the population**).
- A dataset provides an **empirical** description of our target population.

Sampling a population

- Datasets are **representative**, i.e. provide a complete picture of the target population.
- Sampling mimics the mechanism that generates samples during deployment: Samples need to be extracted **independently**.
- Samples need to be **independent and identically distributed** (iid).

Deployment performance

- Machine learning models can be built, sold and deployed.
- The best model is the one with the highest **deployment performance**, i.e. the one that works best on the **target population**.
- A performance evaluation strategy includes:
    1. A **quality metric** used to quantify the performance.
    2. How **data** will be used to assess the performance of a model.
- Test datasets are extracted randomly. Hence, the **test performance** is itself **random**, as different datasets generally produce different values.
- Caution should be used, as the test performance is a random quantity, hence some models might appear to be **superior by chance**!

Optimisation theory

- Assume that we have:
    - A collection of **candidate models**, e.g. all the models resulting from tuning a linear model.
    - A **quality metric**, e.g. a notion of error.
    - An **ideal description** of the target population.
- Optimisation allows us to identify among all the candidate models the one that achieves the highest quality on the target population, i.e. the **optimal** model.

The error surface

- The error surface (a.k.a. error, objective, loss or cost function) denoted by $E(\mathbf{w})$ maps each **candidate model** $\mathbf{w}$ to its **error**.
- The **gradient** (slope) of the error surface, $\nabla E(\mathbf{w})$, is zero at the optimal model. Hence we can look for it by identifying where $\nabla E(\mathbf{w}) = 0$.

Gradient descent

- Gradient descent is a numerical optimisation method where we **update iteratively** our model using the gradient of the error surface.
- The gradient provides the direction along which the error increases the most. Using the gradient, we can create the following update rule:
  $\mathbf{w}_{new} = \mathbf{w}_{old} - \epsilon \nabla E(\mathbf{w}_{old})$
  where ϵ is known as the **learning rate** or **step size**.
- With every iteration we adjust the parameters $\mathbf{w}$ of our model. This is why this process is also known as **parameter tuning**.
- The learning rate ϵ controls how much we change the parameters $\mathbf{w}$ of our model in each iteration of gradient descent:
    - Small values of ϵ result in slow convergence to the optimal model.
    - Large values of ϵ risk overshooting the optimal model.
- For gradient descent to start, we need an initial model. The choice of the initial model can be crucial. The initial parameters $\mathbf{w}$ are usually chosen **randomly** (but within a sensible range of values).
- In general, gradient descent will not reach the optimal model, hence it is necessary to design a **stopping strategy**. Common choices include:
    - Number of iterations.
    - Processing time.
    - Error value.
    - Relative change of the error value.
- Error surfaces can however be complex and have
    - **Local** optima (model with the lowest error within a region).
    - **Global** optima (model with the lowest error among all the models).
- Gradient descent can get **stuck** in local optima. To avoid them, we can repeat the procedure from several **initial models** and select the best.

Training a model

- In machine learning we have:
    - A family of **candidate models** (e.g. linear models).
    - A **quality metric** (e.g. the error).

    - **Data** extracted/**sampled** from the population (i.e. **not** an ideal description).
- We use a subset of data, known as the **training dataset**, to (implicitly or explicitly) **reconstruct** the error surface needed during optimisation. We will call this the **empirical error surface**.
- The empirical and true error surfaces are in general different. Hence, their optimal models might differ, i.e. the best model for the training dataset might not be the best for the population.
- Least squares provides one of the few analytical solutions defining an optimal model from data. A linear model applied to a training dataset can be expressed as:
  $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
  where $\mathbf{X}$ is the design matrix, $\hat{\mathbf{y}}$ is the predicted label vector and $\mathbf{w}$ is the coefficients vector. The MSE on the training dataset can be written as
  $E_{MSE}(\mathbf{w}) = \frac{1}{N}(\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}}) = \frac{1}{N}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$
  where $\mathbf{y}$ is the true label. The resulting gradient of the MSE is
  $\nabla E_{MSE}(\mathbf{w}) = \frac{-2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$
  and it is zero for $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.
- In general, we will not have analytical solutions. We can reconstruct the empirical error surface by evaluating each model on training data. This is called **brute-force** or **exhaustive search**. Simple, but often **impractical**.
- Gradient descent can be implemented by **estimating the gradient** using our training dataset.
- Depending on the amount of data used in each iteration, it is common (although not really useful) to distinguish between:
    - **Batch** gradient descent (the whole training dataset is used).
    - **Stochastic** (or online) gradient descent (one sample is used).
    - **Mini-batch** gradient descent (a small subset from the training dataset is used).
- Small batches produce noisy versions of the gradient of the empirical error surface, which can help to escape local minima.
- Other popular gradient-based optimisation algorithms include:
    - **Momentum** defines a velocity (direction and speed) for the update step, which depends on past gradients.
    - **RMSProp** adapts the learning rate by scaling them using the past gradients.
    - **Adam** combines some features from the Momentum and RMSProp approaches.

Overfitting and fooling ourselves

- The empirical and true error surfaces are in general different. When **small datasets** and **complex models** are used, the differences between the two can be very large, resulting in trained models that work very well for the empirical error surface but very poorly for the true error surface.
- This is, of course, another way of looking at **overfitting**. By increasing the size of the training dataset, empirical error surfaces becomes closer to the true error surface and the risk of overfitting decreases.
- **Never** use the same data for testing and training a model. The test dataset needs to remain **inaccessible** to avoid using it (inadvertently or not) during training.

Regularisation

- Regularisations modifies the empirical error surface by adding a term that constrains the values that the model parameters can take on. A common option is the regularised error surface $E_R(\mathbf{w})$ defined as:
  $E_R(\mathbf{w}) = E(\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$
- For instance, the MSE in regression can be regularised as follows:
  $E_{MSE+R} = \frac{1}{N}\sum_{i=1}^{N} e_i^2 + \lambda\sum_{i=1}^{K} w_k^2$
  and its MMSE solution is
  $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + N\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$
- As λ increases, the complexity of the resulting solution decreases and so does the risk of overfitting.

Cost vs quality

- Regularisation provides an example where we use a notion of quality during training (E_{MSE+R}) that is different from the notion of quality during deployment (E_{MSE}). Doesn't it sound strange?
- Our goal is always to produce a model with that achieves the highest **quality** during deployment. How we achieve it, is a different question.
- Factors such as overfitting might result in models that are optimal during training, but not deployment. A well-designed notion of quality during training can produce models that perform better during deployment.
- We usually call our notion of quality during training **cost** or **objective function**, to distinguish it from the **target quality metric**.

Validating our models

- Machine learning uses datasets for different purposes, for instance to assess the deployment performance of a final model (**test dataset**) or to tune a model (**training dataset**).
- Often, we need to explore different options before training a final model. For example, consider polynomial regression. The polynomial degree D is a **hyperparameter**, as for each value of D a different family of models is obtained. How can we select the right value of a D?
- Validation methods allow us to use data for **assessing** and **selecting** different families of models. The same data used for validation can then be used to train a final model.
- Validation involves one or more **training** and **performance estimation** rounds per model family followed by **performance averaging**.
- The **validation set** approach is the simplest method. It randomly splits the available dataset into a **training** and a **validation** (or hold-out) dataset.
- **Leave-one-out cross-validation (LOOCV)** method also splits the available dataset into training and validation sets. However, the validation set contains **only one sample**.
- **k-fold cross-validation** approach the available dataset is divided into k groups (also known as folds) of approximately equal size:
    - We carry k rounds of training followed by validation, each one using a different fold for validation and the remaining for training.
    - The final estimation of the performance is the average of the performances from each round.

Summary

- In machine learning we can identify the following tasks:
    - **Test**: This is the **most** important task. It allows us to estimate the deployment performance of a model.
    - **Training**: Used to find the best values for the parameters of a model, i.e. to tune a model.
    - **Validation**: Necessary to compare different modelling options and select the best one, the one that will be trained.
- In machine learning we do not have an ideal description of the target population, all we can do is **extract data**.
- Datasets need to be **representative** of the target population and its samples need to have been extracted **independently**.
- Any test strategy has to be **designed before training**. Avoid looking at the test dataset during training and test a final model only once (Monkey Theorem!).
- Any performance estimation that is obtained from a dataset is a **random quantity**: use with caution.
- The quality of a final model depends on the **type** of model, the **optimisation** strategy and the representativity of the **training data**.

0.2 Methodology II

Pipeline

- **Definition**: The term pipeline describes a sequence of operations. A supervised machine learning pipeline is the sequence of operations that produce a prediction (output) using a set of predictors (input).
- A pipeline can consist of multiple stages, including:
    - **Transformation stages** (where input data is processed).
    - **Several machine learning models running in parallel**.
    - **A final aggregation stage** (where individual outputs are combined to produce one single output).
- The stages in a pipeline can be either hand-crafted or trained.
- After training, the parameters of a pipeline remain fixed.
- Pipelines can be tested and deployed.
- The quality of the prediction depends on **all the pipeline stages.**

Normalization

- **Definition**: Data normalization is a tunable transformation stage that allows us to scale attributes so that their values belong to similar ranges.
- It helps to ensure that attributes with different units, dimensions, or dynamic ranges do not disproportionately influence the model.
- **Min-max normalization**:
    - **Definition**: produces values within the same continuous range [0, 1].
    - **Formula**: $z = \frac{x - \min(x)}{\max(x) - \min(x)}$
- **Standardization**:

    - **Definition**: produces values with 0 mean and unit standard deviation.
    - **Formula**: $z = \frac{x-\mu}{\sigma}$, where μ is the average of x and σ is its standard deviation.
- **Observations**:
    - Outliers can have a negative impact on normalization.
    - Non-linear scaling options exist, such as softmax scaling and logarithmic scaling.
    - The original values in your dataset might be relevant, so consider unintended effects and distortions.
    - Many machine learning algorithms might produce different solutions after scaling.

Transformations

- **Definition**: Transformations are data manipulations that change the way that we represent our samples. They can be seen as moving samples from one space (original space) to another (destination space).
- **Types**:
    - **Linear transformations**: can be seen as a rotation and scaling.
    - **Non-linear transformations**: have no unique description.
- **Dimensionality reduction**: If the destination space has fewer dimensions than the original one.
    - **Feature selection**: the destination space is defined by a subset of the original attributes.
        - **Filtering**: consider each attribute individually.
        - **Wrapping**: evaluate different subsets of features together.
    - **Feature extraction**: the new attributes are defined as operations on the original attributes.
- **Complex models and kernel methods**:
    - Many complex machine learning models can be interpreted as a transformation followed by a simple model.
    - Kernel methods, such as support vector machines, implicitly define such transformations using so-called kernel functions.

Ensembles

- **Definition**: An ensemble is a machine learning strategy that combines the output from individual models (base models).
- **Principles**:
    - Base models need to be as diverse as possible.
    - Diversity can be achieved by training:
        - A family of models with random subsets of the data.
        - Different models with random subsets of attributes.
        - Different families of models altogether.
- **Bagging (Bootstrap Aggregating)**:
    - **Definition**: generates K sub-datasets by **bootstrapping** and trains K simple base models with each sub-dataset.
    - The final model combines the predictions of the base models by averaging or voting.
- **Decision trees**:
    - **Definition**: partition the predictor space into multiple decision regions by implementing sequences of splitting rules using one predictor only.
    - The goal is to create pure leaves, i.e. containing as many samples from the same class as possible.
    - Decision trees are built recursively.
    - Splits are axis-parallel (decisions using one predictor).
    - The chosen split is such that the purity of the resulting regions is higher than any other split.
    - We stop when a given criterion is met, such as the number of samples in a region.
- **Random forests**:
    - **Definition**: an ensemble of decision trees.
    - Random forests train many individual trees by randomising the training samples and the predictors.
    - Predictions are obtained by averaging the individual predictions.
- **Boosting**:

    - **Definition**: generates a sequence of simple base models, where each successive model focuses on the samples that the previous models could not handle properly.
    - Each new base model generated by boosting depends on the previous models.

Summary

- **Deploying machine learning pipelines**:
    - Pipelines define sequences of operations on data.
    - In machine learning we deploy pipelines, not just models.
    - Complex machine learning models can be seen as pipelines.
- **Transformations**:
    - The purpose is to use a more convenient representation for our data.
    - Normalisations are transformations where each attribute is scaled individually.
    - In feature selection we select a subset of attributes, whereas in feature extraction we produce a new set of attributes by processing the input attributes.
- **Ensembles**:
    - Combine the output from individual models.
    - The idea is that each model learns different aspects of the true underlying model.
    - Can be created by using different subsets of samples, different subsets of attributes or different families of models.