ML models

Summer 24

Christian Steineder

Contents

1 First mathematical tools

1.1 Maximum likelihood estimation

1.1.1 The basic idea

In classical (frequentistic) statistics parameters are (fixed but unknown) values which determine the distribution of the random variable modelling given data. One can define for a value θ the probability

$$\mathbf{P}_\theta(Data)$$

if the distribution of the data depends on θ. If, in particular, the data is a set of independent measurements $Data = \{x_1, \ldots, x_n\}$ who all satisfy the same distribution then the joint probability factors

$$\mathbf{P}_\theta(Data) = \mathbf{P}_\theta(x_1, \ldots, x_n) = \prod_i \mathbf{P}_\theta(x_i).$$

Such data is called i(ndependent) i(dentically) d(istributed) data.

The M(aximum) L(ikelihood) E(stimation) of the unknown parameter θ given the iid.data $\{x_1, \ldots, x_n\}$ looks for the value(s) for θ which maximises the probability that the measured data set occurred, i.e.,

$$MLE(\theta, \mathbf{x}) = \mathrm{argmax}_\theta \prod_i \mathbf{P}_\theta(x_i).$$

($\mathrm{argmax}_x f(x)$ stands here for arguments of the maxima, i.e., for the x which maximise $f(x)$.)

Remarks:

1. Since sums are in general simpler to handle than products one uses the logarithm to find the maximum, i.e., one uses (due to the monotonicity of ln),

$$\ln\left(\mathrm{argmax}_\theta \prod_i \mathbf{P}_\theta(x_i)\right) = \mathrm{argmax}_\theta \sum_i \ln(\mathbf{P}_\theta(x_i)).$$

2. The MLE satisfies several mathematical properties which justify its usefulness, we mention here just mention one, namely consistency, which says, that the MLE converges to the real value of θ if the sample size gets bigger and bigger.

3. In most applications, e.g. if θ is not a single parameter but even a (higher dimensional) set of parameters, the MLE cannot be computed explicitly – therefore numerical methods like the Gradient–descent method which are also used in other machine learning algorithms play an important role here.

1.1.2 An example

Assume the example flipping a coin, you observe in N repetitions k times heads. If θ is the (unknown) probability for heads of this coin the likelihood function of this outcome is

$$\mathbf{P}_\theta(k \text{ times heads in } N \text{ flips}) = \theta^k(1-\theta)^{N-k}.$$

To find the MLE of θ we maximise (for simplicity) the ln of the likelihood function w.r.t. θ, i.e.,

$$f(\theta) = k\ln(\theta) + (N-k)\ln(1-\theta).$$

Using differential calculus, i.e., by solving $f'(\theta) = 0$, it follows that the maximum occurs at $\theta = k/N$, i.e. the relative frequency of heads is the best estimator for θ.

1.2 Bayes rule

The definition of conditional probability , i.e.,

$$\mathbf{P}(A|B) = \frac{\mathbf{P}(A,B)}{\mathbf{P}(B)}$$

gives (since $\mathbf{P}(A,B) = \mathbf{P}(A \cap B) = \mathbf{P}(B \cap A) = \mathbf{P}(B,A)$) the central Bayes rule:

$$\mathbf{P}(A|B) = \frac{\mathbf{P}(B|A)\mathbf{P}(A)}{\mathbf{P}(B)}$$

for any events A and B. The terms in Bayes rule have the following names (as soon as we discuss applications of Bayes rule these names will become more clear):

1. The denominator $\mathbf{P}(B)$ is called the **evidence**. In many cases this is just seen as a normalisation constant, giving the notation

$$\mathbf{P}(A|B) \propto \mathbf{P}(B|A)\mathbf{P}(A), \tag{1}$$

   or

$$\mathbf{P}(A|B) = \alpha\mathbf{P}(B|A)\mathbf{P}(A),$$

   where $\propto$ stands for proportional and α denotes the normalisation constant $1/\mathbf{P}(B)$

2. The factor $\mathbf{P}(A)$ is called the **prior** probability.

3. The factor $\mathbf{P}(B|A)$ is called the **likelihood** probability.

4. The factor $\mathbf{P}(A|B)$ is called the **posterior** probability.

With this notation Bayes rule becomes

$$\underbrace{\text{posterior}}_{\mathbf{P}(A|B)} \propto \underbrace{\text{likelihood}}_{\mathbf{P}(B|A)} \times \underbrace{\text{prior}}_{\mathbf{P}(A)}$$

The importance of Bayes rule is based on the fact that we can compute unknown probabilities using known probabilities. We discuss here two important applications of Bayes rule:

1.3 Bayes rule and effect vs. cause

Interpreting in above formula 1

1. A as effect and

2. B the (unknown) cause of this effect then

Bayes rule becomes

$$\mathbf{P}(\text{cause}|\text{effect}) \propto \mathbf{P}(\text{effect}|\text{cause})\mathbf{P}(\text{cause})$$

EX: Assume cause is a virus and effect is a certain virus scan detects the virus. The prior $\mathbf{P}(\text{cause})$ as well as the likelihood $\mathbf{P}(\text{effect}|\text{cause})$ can be

determined (say in a test series) and this allows to compute the reliability of the test.

$\mathbf{P}(\text{effect})$ can be computed using so called marginalisation, i.e., by adding up all possible states of the other variable:

$$\begin{aligned}\mathbf{P}(\text{effect}) &= \mathbf{P}(\text{cause}, \text{effect}) + \mathbf{P}(\neg\text{cause}, \text{effect}) \\ &= \mathbf{P}(\text{effect}|\text{cause})\mathbf{P}(\text{cause}) + \mathbf{P}(\text{effect}|\neg\text{cause})\mathbf{P}(\neg\text{cause})\end{aligned}$$

1.4 Conditional distributions

Suppose given are two random variables X and Y. If both only take finitely many values we define the conditional distribution of X given Y in the obvious way based on the definition of conditional probability: We define for each x such that $\mathbf{P}(X = x) > 0$

$$\mathbf{P}(Y = y|X = x) = \frac{\mathbf{P}(Y = y, X = x)}{\mathbf{P}(X = x)}$$

and call the resulting distribution the (conditional) distribution of Y given X. However, if X and Y are continuous random variables we again must apply density functions. This directly leads to the definition

$$f_{Y|X}(y|x) = f(y|x) = \frac{f_{XY}(x, y)}{f_X(x)}.$$

The function $f_{Y|X}$ is called the conditional density. Here f_{XY} denotes the joint density of X and Y as well as f_X the density of X.

Similar to events we call X and Y independent it

$$f_{XY}(x, y) = f_X(x)f_Y(y)$$

The definitions above directly lead to a Bayes Theorem for densities:

$$f(y|x) = \frac{f(x|y)f(y)}{f(x)} = \frac{f(x|y)f(y)}{\int_{-\infty}^{\infty} f(x, y)dy} = \frac{f(x|y)f(y)}{\int f(x|y)f(y)dy}.$$

Using the proportional notation introduced earlier we obtain also for densities

$$\underbrace{f(y|x)}_{\text{posterior}} \propto \underbrace{f(x|y)}_{\text{likelihood}} \cdot \underbrace{f(y))}_{\text{prior}}$$

1.5 Conditional independence

Definition of conditional independence

Let A, B, C be events. We call A conditionally independent of B given C, notation $A \perp\!\!\!\perp B|C$ if

$$\mathbf{P}(A|B,C) = \mathbf{P}(A|C).$$

Note that this definition is equivalent to

$$\mathbf{P}(A,B|C) = \mathbf{P}(A|C)\mathbf{P}(B|C).$$

Remark: Independence equals conditional independence given $C = \varnothing$. However, if $C \neq \varnothing$, conditional independence given C and independence are not related. There are easy examples of dependent events which are conditionally independent and vice versa – see for instance the following informal settings:

Example:

1. Height of people under 20 and their math skills are dependent (typically very small persons are small kids whose math skills are limited), however, if conditioned on their age height and math skills are independent.

2. In a twofold dice roll the first and second roll are independent, however, if conditioned on the fact that their sum is, say, even they are conditionally dependent.

Let X, Y, Z be random variables. We call X conditionally independent of Y given Z, notation $X \perp\!\!\!\perp Y|Z$ if

$$\mathbf{P}(X = x|Y = y, Z = z) = \mathbf{P}(X = x|Z = z).$$

One says in this situation that X and Y are conditionally independent if Z is observed.

Queries

A probability query consists of two parts:

1. The evidence: A set of random variables E and a choice of values e which E can take.

2. The query (variables): A set of random variables Y.

Our task is to compute $\mathbf{P}(Y|E = e)$, i.e. the posterior distribution of Y given the evidence e. Particularly interesting are those values for Y which maximise $\mathbf{P}(Y|E = e)$, i.e. we are interested in an assignment y such that $\mathbf{P}(Y = y, E = e)$ and therefore, by Bayes rule, $\mathbf{P}(Y|E = e)$ is maximal.

1.6 The naive Bayes model

The joint distribution of a set of (finitely) many random variables contains all information of all the random variables – marginalisation and conditioning allows to compute several induced distribution.

Example: Assume $P(X, Y, Z)$ is given. Then one could, e.g., directly compute $\mathbf{P}(X, Y) = \sum_z \mathbf{P}(X, Y, Z = z)$ or $\mathbf{P}(X, Y|Z = z) = \frac{\mathbf{P}(X,Y,Z=z)}{\mathbf{P}(Z=z)}$.

However, given, a joint distribution on n variables, say, $\mathbf{P}(X_1, \ldots, X_n)$ where each random variable is only binary. Then the whole distribution, i.e., $\mathbf{P}(x_1, \ldots, x_n)$ for $x_i \in \{0, 1\}$, is given by the 2^n different combinations the different values of the variables x_i can take. In fact, since the distribution adds up to 1, as soon as we know the probability of $2^n - 1$ combinations, we know the whole distribution – still the space necessary to store the info contained in the distribution grows exponentially. Now, on the contrary suppose, that all the random variables X_i are independent. Then

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i)$$

and $\mathbf{P}(X_i)$ is determined by one value p_i, since the other one is $1 - p_i$. So overall we only need to store n values to have the entire probability distribution.

Independence per se is a strong assumption, however, conditional independence on some events is less strict and still allows a simpler description of the joint distribution based on factorisation – this leads to the so called Naive Bayes model:

Assume some effects that are all mutually conditionally independent given some cause. Then the joint probability distributions can be factorised as

$$\mathbf{P}(\text{cause}, \text{effect}_1, \ldots, \text{effect}_n) = \mathbf{P}(\text{cause}) \prod_n \mathbf{P}(\text{effect}_n|\text{cause})$$

due to the chain rule. Such a joint probability is called the naive Bayes model. This factorisation can be used in the following fashion: Assume there are some observed effects $\mathbf{e}$ and some other unobserved effects Y. Then, by Bayes rule

$$\begin{aligned}\mathbf{P}(\text{cause}|\mathbf{e}) &\propto \sum_y \mathbf{P}(\text{cause}, \mathbf{e}, y) \\ &= \sum_y \mathbf{P}(\text{cause})\mathbf{P}(y|\text{cause}) \prod_i \mathbf{P}(e_i|\text{cause}) \\ &= \left(\mathbf{P}(\text{cause}) \prod_i \mathbf{P}(e_i|\text{cause})\right)\left(\sum_y \mathbf{P}(y|\text{cause})\right) \\ &= \mathbf{P}(\text{cause}) \prod_i \mathbf{P}(e_i|\text{cause}).\end{aligned}$$

I.e., in order to know the probability distribution of the causes given some observed effects one can use the factorisation w.r.t. the the given effects.

1.7 Basics of Bayesian networks

Bayesian networks (BN) are graphs illustrating the conditional independencies of a given joint distribution. In fact, any joint distribution can be represented as Bayesian network.

Syntax of a Bayesian network

A BN is a graph G = (V,E) where

1. each vertex in V is one random variable,
2. G is a directed acyclic graph,
3. each vertex is labeled with a conditional probability table given its parents, i.e., its immediate predecessors in G,

The intuition between this graphical representation is if there is an edge from X to Y then X directly influences Y. In other words: Causes are parents of effects.

Semantics of a Bayesian network

The semantics of a BN defines how the joint distribution can be derived from

the syntax, namely, by definition:

$$\mathbf{P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathbf{P}(x_i | \text{parents of } X_i). \tag{2}$$

Here the small x_i denote the values the random variable X_i takes and parents of X_i the value of the parents of the random variable x_i.

Construction of a Bayesian network

Comparing the definition 2, i.e.,

$$\mathbf{P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathbf{P}(x_i | \text{parents of } X_i).$$

and the chain rule of conditional probabilities, i.e.

$$\begin{aligned}
&\mathbf{P}(x_1, x_2, \ldots, x_n) \\
&\quad = \mathbf{P}(x_n | x_1, x_2, \ldots, x_{n-1}) \mathbf{P}(x_{n-1} | x_1, x_2, \ldots, x_{n-2}) \cdots \mathbf{P}(x_2 | x_1) \mathbf{P}(x_1) \\
&\quad = \prod_{i=1}^{n} \mathbf{P}(x_i | x_1, x_2, \ldots, x_{i-1}).
\end{aligned}$$

we see that the factorisation of the joint distribution in 2 is guaranteed, if

1.

$$\mathbf{P}(x_i | x_1, x_2, \ldots, x_{i-1}) = \mathbf{P}(x_i | \text{parents of } X_i) \tag{3}$$

2. Parents of $X_i \subseteq \{X_1, X_2, \ldots, X_{j-1}\}$.

The latter condition is satisfied if we enumerate the nodes X_i of the graph (so called) topologically, i.e., if there is an edge from X_i to X_j then $i < j$. The former condition means that that each node is conditionally independent of all its predecessors given its parents.

Using the following algorithm to construct a BN satisfies these two conditions:

1. Vertices: Determine the set of vertices V necessary to model the domain. They can be ordered in any way, however, the rule cause precede effect leads in general to a optimal (compact) structure.

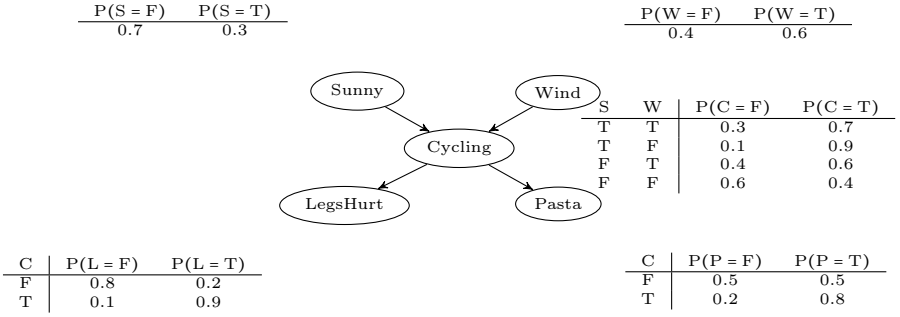
2. Edges: For $i = 1$ to n: choose a minimal set of parents of vertex X_i such that 3 holds. According to the chain rule this is always possible. For each parent insert an edge from the parent to X_i.

3. Conditional probability table: Write down the conditional probability table $\mathbf{P}(x_i|\text{parents of } X_i)$.

The idea of this representation is that the parents directly influence its children. Observe that this construction guarantees that the graph is acyclic since edges respect the order of the vertices, i.e. no edge connects a later vertex to an earlier one w.r.t. the ordering.

1.7.1 An example

Given is the following BN reflecting some parameters relevant for Christians cycling behaviour:



P(S = F)	P(S = T)
0.7	0.3

P(W = F)	P(W = T)
0.4	0.6

S	W	P(C = F)	P(C = T)
T	T	0.3	0.7
T	F	0.1	0.9
F	T	0.4	0.6
F	F	0.6	0.4

C	P(L = F)	P(L = T)
F	0.8	0.2
T	0.1	0.9

C	P(P = F)	P(P = T)
F	0.5	0.5
T	0.2	0.8

Given this information it is possible to compute the entire joint probability, e.g.,

$$\begin{aligned}\mathbf{P}(S, \neg W, C, L, P) &= \mathbf{P}(S)\mathbf{P}(\neg W)\mathbf{P}(C|S, \neg W)\mathbf{P}(L|C)\mathbf{P}(P|C) \\ &= 0.3 \cdot 0.4 \cdot 0.9 \cdot 0.9 \cdot 0.8 \approx 0.078\end{aligned}$$

Observe that row- wise each of the given probability tables necessarily adds up to 1, so half of the information is "free".

1.7.2 Further remarks and outlook

Independence in Bayesian networks

The definition of the BN guarantees the fact:

Each variable is conditionally independent of its predecessors, given its parents.

However, there are more independence relations which can be deduced only by the shape of the BN - since they also allow to compute the joint distribution but now entirely depend on the shape of the BN they are called topological semantics. We list here some of these independence properties which can be shown:

1. (Markov property) Each variable is conditionally independent of its non-decendents, given its parents.

2. (Markov blanket) A variable is conditionally independent of all other vertices given its parents, its children and its children's parents.

3. (d–separation) The most general independence criterion for a BN is d–separation which guarantees the conditional independence of two sets of vertices given a third set which satisfies some conditions which can be checked purely within the graph topology. We omit details here.

Inference in Bayesian networks

The aim of inference in Bayesian networks is to compute probability distributions related to queries, i.e. $\mathbf{P}(X|\mathbf{e})$, where X is the variable of interest and $\mathbf{e}$ is a list of given evidence. Using the joint distribution, which can be computed via products of the conditional probability tables given in the BN, queries can be computed up to normalisation by summation (marginalisation), i.e.,

$$\mathbf{P}(X|\mathbf{e}) \propto \mathbf{P}(X,\mathbf{e}) = \sum_{\mathbf{y}} \mathbf{P}(X,\mathbf{e},\mathbf{y})$$

Thus, any query can be answered by computing sums of products of entries of the given distribution tables.

Exact inference

There exists several algorithms for an exact computation of the queries. They are closely related to the DPLL algorithm discussed in the framework of propositional logic. The so–called enumeration algorithm and the variable elimination algorithm are mentioned here. However, it is shown (as in the case of the DPLL algorithm) that exact inference is a NP-hard problem, thus

especially for large BNs the exponential complexity can cause computational problems. Therefore often approximative inference is used.

Approximate inference

Given the NP-hardness of exact inference one chooses so called randomised sampling methods to provide approximate answers to queries. Some sampling methods are direct sampling as well as several M(arkov) C(hain) M(onte) C(arlo) sampling methods, in particular Gibbs sampling and Metropolis Hastings sampling.

1.8 Bayes statistics

Suppose you have some data and some hypothesis (e.g. on the value of an unknown parameter describing the distribution of interest). Interpreting in above formula 1

1. A as data and
2. B the (unknown) hypothesis of this effect then

Bayes rule becomes

$$\mathbf{P}(\text{hypothesis}|\text{data}) \propto \mathbf{P}(\text{data}|\text{hypothesis})\mathbf{P}(\text{hypothesis}).$$

Revising the terminology above, we have

1. the prior $\mathbf{P}(\text{hypothesis})$. This is the prior (current) state of knowledge, which values for our hypothesis are how likely.
2. the likelihood $\mathbf{P}(\text{data}|\text{hypothesis})$. This is the likelihood that we observe the (given) data given the hypothesis.
3. the posterior $\mathbf{P}(\text{hypothesis}|\text{data})$. This is the probability that our hypothesis (parameter) has a certain value, given the data.
4. the evidence $\mathbf{P}(\text{data})$. This parameter does not contain the value "hypothesis" of interest and is therefore often neglected in this discussion and replaced by the proportion relation.

1.9 Bayes vs. classical approach

Probability in an abstract/mathematical/axiomatic sense is clearly defined as function on the set of events. The question remains, how to bridge the abstract concept to applications, i.e., how to assign probability so that it displays real live.

The wish to establish a solid/objective framework for probability led to the so–called frequentistic probability approach: The probability of an event is a fixed but unknown value which is approximated by the long run relative frequency of the random experiment – the relative frequency of the infinitely many repeated trials equals this probability. Mathematically this approach is justified by a well established mathematical theorem, the so-called law of large numbers.

However, there are some counter–arguments to this approach, we mention two:

1. The world is deterministic: An objective approach to probability seems unnecessary if one follows the point of view that randomness does not exist but undetermined outcomes are simply a consequence of unknown initial conditions. Randomness is just an inability to predict in fact deterministic processes. Thus our use of probability is never objective but just a subjective state of current knowledge.

2. A repeated random trial is most likely/never repeated under perfectly same conditions. Thus the concept of frequency is never objective/-precise.

The so called Bayesian approach based on Bayes rule replaces the objectivistic approach by the so called subjectivistic approach: Given the (subjective) prior knowledge and the likelihood allows to establish a distribution for the unknown hypothesis. The fact that Bayes rule directly gives a distribution of the hypothesis is a useful advantage which directly allows to make probabilistic statements about the hypothesis. In the frequentistic setting the theory of testing is necessary to obtain comparable results.

Still, mathematically, the main difference is the fact that in Bayes statistics the parameters of interest are themselves random variables whose distribution/ density explains the current state of believe.

Finally it should be noted that our way of establishing Bays rule from made use of the axioms which also lead to the frequentistic approach. There is also a direct way of axiomatically derive Bayes rule as starting point – these axioms were formulated by Cox (see e.g. Cox's theorem in Wikipedia).

1.10 The coin example again

Suppose you flip a coin and want to determine the probability for, say, heads. Associating the probability p with the (unknown) probability you flip the coin N times and you observe R times heads, i.e. your data d is R times heads and $N - R$ times tails. We have

1. the prior $\mathbf{P}(p)$: Assuming no prior knowledge every probability $p \in [0, 1]$ for heads is equally likely – therefore we could choose the uniform distribution for p:
$$\mathbf{P}(p) = 1 \quad \text{for } p \in [0, 1].$$
Of course other priors are also possible, e.g. a peaked one at $p = 0.5$ (if one assumes the coin to be fair). The Beta distribution is a frequently used prior – a discussion of this fact follows in section 1.11.

2. the likelihood $\mathbf{P}(d|p)$ of our data d which equals – due to independence of the single flips:
$$\mathbf{P}(d|p) = p^R(1-p)^{N-R}.$$

Applying Bayes rule we obtain:

$$\mathbf{P}(p|d) \propto p^R(1-p)^{N-R}$$

The remarkable point of this formula is that we obtained a probability distribution for the unknown parameter p based on the (concrete) data d.

1.11 Conjugate distributions

If a prior distribution/ density f_{prior} multiplied with a likelihood distribution/ density f_{likely} gives a posterior distribution/ density f_{post} of the same distribution family as f_{prior} we call f_{prior} a conjugate (prior) distribution of f_{likeli}. Conjugate distributions are important because they typically allow

a direct interpretation of the effect of the likelihood, i.e. the observed data set, on the prior knowledge.

Some examples

1. Beta-distribution: Following the example presented in section 1.10 assume that the prior knowledge of the parameter p is now $Be(\alpha, \beta)$, i.e., Beta distributed with parameters α and β, i.e.

$$f_{prior,(\alpha,\beta)}(p) = \underbrace{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}}_{=1/B(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1}$$

where $\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$ is the so called Gamma function which is defined for $t > 0$. If the likelihood is as in section 1.10, i.e, the probability of observing some data $\mathbf{x} = x_1, x_2, \ldots x_n$, with, say, r successes, then

$$f_{likeli}(\mathbf{x}|p) = p^r(1-p)^{n-r}.$$

To compute the denominator of

$$f(y|x) = \frac{f(x|y)f(y)}{\int f(x|y)f(y)dy}$$

observe that

$$\begin{aligned}
\int f(x|p)f(p)dp &= \int_0^1 f(x|p)f(p)dp \\
&= \int_0^1 p^r(1-p)^{n-r}\frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha,\beta)}dp \\
&= \frac{1}{B(\alpha,\beta)}\int_0^1 p^{r+\alpha-1}(1-p)^{n-r+\beta-1}dp \\
&= \frac{B(r+\alpha, n-r+\beta)}{B(\alpha,\beta)}.
\end{aligned}$$

This directly leads to the posterior

$$\begin{aligned}
f_{post} &= \frac{p^r(1-p)^{n-r}p^{\alpha-1}(1-p)^{\beta-1}B(\alpha,\beta)}{B(\alpha,\beta)B(r+\alpha,n-r+\beta)}. \\
&= \frac{p^{r+\alpha-1}(1-p)^{n-r+\beta-1}}{B(r+\alpha,n-r+\beta)}.
\end{aligned}$$

This means that f_{post} is $Be(\alpha + r, \beta + n - r)$ distributed, i.e., the parameters of the prior are updated according to the data.

2. Normal-distribution: Assume that f_{prior} is m-s-normally distributed, i.e.

$$f_{prior}(\mu) = \frac{1}{\sqrt{2\pi}s} e^{-\frac{(\mu-s)^2}{2s^2}}$$

and the likelihood for some data $\mathbf{x} = x_1, x_2, \ldots x_n$ also satisfies a normal distribution f with parameters μ and σ, where σ is assumed to be known, i.e., the likelihood is given by

$$\begin{aligned} f_{likel}(\mathbf{x}|\mu) =& f(x_1|\mu) \cdot f(x_2|\mu) \cdot \cdots \cdot f(x_n|\mu) \\ =& \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}} \\ =& \ldots \quad \text{(some steps are omitted)} \\ =& \frac{1}{\sqrt{2\pi/n}\sigma} e^{-\frac{(\bar{x}-\mu)^2}{2\sigma^2/n}}, \end{aligned}$$

where $\bar{x}$ stands, as usual, for the mean. Then it can be shown that the posterior f_{post} is given by

$$f_{post}(\mu|\mathbf{x}) \propto e^{-\frac{(\mu'-\mu)^2}{2(\sigma')^2}},$$

where

$$\mu' = \frac{m\sigma^2 + ns^2\bar{x}}{\sigma^2 + ns^2} \quad \text{and} \quad \sigma' = \frac{s^2\sigma^2}{\sigma^2 + ns^2}.$$

I.e., the posterior is also normally distributed and it is possible to observe how the data $\mathbf{x}$ updates the μ and σ to μ' and σ'

3. Dirichlet-distribution: The Dirichlet distribution gives the generalisation of the first example: Assume given are K categories and each occurs with probability p_i such that $\sum_{i=1}^{K} p_i = 1$. The Dirichlet distribution of the paramter $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$ then is a distribution for those probabilities p_i as the Beta distribution was a distribution for the probability p in the first example.

   For the sake for completeness we add the (multivariable) density function:

$$f_{Dir(\boldsymbol{\alpha})}(x_1, \ldots, x_K) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} p_i^{\alpha_i - 1},$$

   where $B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K}\alpha_i)}$.

The analogy to example 1 continues: It can be shown that the Dirichlet distribution is the conjugate prior distribution of the multinomial distribution (the distribution over observed counts of each possible category in a set of categorically distributed observations). This means that if a data point has multinomial distribution, and the prior distribution of the distribution's parameter (the vector of probabilities that generates the data point) is distributed as a Dirichlet, then the posterior distribution of the parameter is also a Dirichlet. Intuitively, in such a case, starting from what we know about the parameter prior to observing the data point, we then can update our knowledge based on the data point and end up with a new distribution of the same form as the old one.

In fact, the following can be computed: If $f_{prior}(p_1, \ldots, p_k)$ is Dirichlet distributed with parameter

$$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$$

and one observes c_i data in category i with the likelihood $f_{likeli}(\mathbf{c}|p_1, \ldots, p_K)$ being a multinomial distribution, i.e.,

$$\mathbf{P}(c_1, \ldots, c_K|p_1, \ldots, p_K) = \frac{n!}{c_1!c_2!\cdots c_K!} p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k},$$

then the posterior is also Dirichlet distributed with parameter

$$\boldsymbol{\alpha}' = (\alpha_1 + c_1, \ldots, \alpha_K + c_K).$$

1.12 MLE and Bayesian statistics

In Bayesian statistics there exists a concept similar to the MLE – one wants to fix the value of the hypothesis which is maximal likely, i.e. one computes the maximum of the posterior $\mathbf{P}(hypothesis|Data)$. This idea is called M(aximum) A P(osteriori) principle. Recap that

$$\mathbf{P}(Hypothesis|Data) \propto \mathbf{P}(Data|Hypothesis)\mathbf{P}(Hypothesis).$$

This shows that MLE and the maximisation approach in Bayesian statistics coincide if the prior hypothesis $\mathbf{P}(Hypothesis)$ is chosen constant (uniform distribution). Otherwise these approaches differ exactly by this prior information which displays again the subjectivity philosophy behind Bayesian statistics.

1.13 Entropy

1.13.1 The concept of entropy

Entropy in information theory is a tool to measure the informational value of a message. The underlying assumption related to probability is that the more probable a message is the less information it contains. For instance, knowing that a certain number does not win in the lottery contains almost no information and the probability that a number does not win is very large while knowing that a certain number wins contains lots of informations and the probability that a number does win is very small. Thus information can also be seen as uncertainty or as surprise.

The definition of the entropy

Given a random variable X its entropy $H(X)$ is defined as

$$\boxed{H(X) = E(-\log_2(\mathbf{P}(X)) = -\sum_x p(x)\log_2 p(x).}$$

Remark: The base of the logarithm in this definition varies, e or 10 are also frequently used.

An informal explanation of the entropy

Assume one wants to encode a string on the 4 letter alphabet, say:

$$aabcacda$$

If every letter comes equally likely, so with probability 1/4, encoding them with an equal amount of bits seems reasonable. In this example there are 4 letters, the number of bits to describe those is given by $2 = \log_2(4) = -\log_2(1/4)$ where 1/4 can be seen. Such an encoding would be, e.g., $a = 00$, $b = 01$, $c = 10$ and $d = 11$.

However, if the letters have variable probability it would be better (in the light of an overall compact encoding) to encode those which occur more frequently, i.e., those which have higher probability, i.e., those who carry less information, with a shorter set of bits, and vice versa, those, who occur less

likely with more bits. E.g., if a occurs with probability $1/2$, b with probability $1/4$ and c and d with probability $1/8$ then one could encode $a = 0$, $b = 10$, $c = 110$ and $d = 111$. In this example the number of bits b is directly related to the probability of the letters, namely $b = -\log(1/p)$. In general if one encodes (optimally – this is not possible if the probabilities are no powers of 2) a letter occurring which probability p with a string of length $\log(1/p)$ then an encoding of a string of length n where the letters occur according to those probabilities has lengths $nH(X)$. In simple terms $H(X)$ measures the optimal average string lengths of a binary encoding of a message.

An axiomatic derivation of the entropy

One defines information I of events by the following properties

1. I is a function of the probability of events.
2. $I(p)$ is monotonically decreasing.
3. $I(p) \geq 0$.
4. $I(1) = 0$ (if an event occurs with probability 1 it contains no information).
5. $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$ (if p_1 and p_1 are probabilities coming from two independent events, the information learned from both of them – whose probability is $p_1 \cdot p_2$ by independence – equals the sum of the single informations.).

Point 5. is the functional equation of the $\log_b$, overall those conditions fix $I(p) = -\log_b(p)$. Now, the entropy $H(X)$ of a random variable X is the average information of a random variable, i.e.,

$$H(X) = E(I(X)) = E(-\log_2(\mathbf{P}(X))) = -\sum_x \mathbf{P}(x) \log_2(\mathbf{P}(x),$$

where the base 2 is usually chosen because of its relation to a binary encoding as indicated above.

1.14 Connection to position weight matrices and sequence logos

The position weight matrix (PWM) is a commonly used representation of motifs (patterns) in biological sequences. PWMs are often computed from a list of aligned sequences which are potentially functionally related, and have replaced consensus sequences to be the most commonly used representation in motif discovery software and biological publications. PWM for DNA sequences has four rows, each representing a nucleotide, multiple columns, each representing a TFBS position.

To convert a PWM to the sequence logo, for each letter column j, the height of the whole letter stack is determined by the information content I of the column, which is calculated as follows:

One generally defines the **information content** at position j as

$$I_j = H_{\max} - H(\text{position } j),$$

where $H_{\max}$ is the maximal entropy (of uniform distribution), i.e. for 4 nucleotides $H_{\max} = 2$.

Then the sequence logo is given by a letters of size $I_j \cdot p_{ij}$, where p_{ij} is the probability of the i–th nucleotide at position j.

1.15 Conditional entropy and information gain

Suppose two random variables X and Y are given. Then the entropy of the random variable $X|Y = y$ can be computed according to the definition

$$H(X|Y = y) = -\sum_x \mathbf{P}(x|Y = y) \log_2(\mathbf{P}(X|Y = y)).$$

Then, the entropy of X given Y, $H(X|Y)$, is the average of $H(X|Y = y)$ over the values Y can take, i.e.,

$$H(X|Y) = \sum_y \mathbf{P}(y) H(X|Y = y) = -\sum_{x,y} \mathbf{P}(x, y) \log_2(\mathbf{P}(X|Y = y)).$$

Properties of the conditional entropy:

1. $X = f(Y)$ (i.e. X is determined by Y) if and only if $H(X|Y) = 0$.
2. X and Y are independent if and only if $H(X|Y) = H(X)$.
3. (chain rule) $H(X,Y) = H(X|Y) + H(Y)$. Here

$$H(X,Y) = \sum_y \mathbf{P}(x,y) \log P(X,Y)$$

denotes the so called joint probability of X and Y.

Information gain: Information gain, sometimes also expected mutual information, is defined as

$$I(X,Y) = H(X) - H(X|Y)$$

This quantity measures how much knowing Y reduces the uncertainty of X. Since $I(X,Y) = H(X) + H(Y) - H(X,Y) = I(Y,X)$ this can be seen as the information X and Y share. Information gain is used to quantify the added information.

EX: Assume you have some data, which contains 100 elements – you interpret this data as probability space equipped with relative frequency as probability. Assume among those 100 elements are 70 of class A and 30 of class B. Then the entropy of X measuring if an element is in class A or B equals, by definition,

$$H(X) = -7/10 \log(7/10) - 3/10 \log(3/10) \approx 0.88$$

Now split the data set twice:

1. Split 1: In two groups, group 1 is of size 40 and contains 20 elements of A and 20 of B. I.e. the entropy of this group equals

$$H(X|Y_1 = 1) = -1/2 \log(1/2) - 1/2 \log(1/2) = 1$$

and group 2 is of size 60 and contains 50 elements of A and 10 of B. I.e. the entropy of this group equals

$$H(X|Y_1 = 2) = -5/6 \log(5/6) - 1/6 \log(1/6) \approx 0.65$$

The conditional entropy of X given Y_1 is by definition

$$H(X|Y_1) = 40/100 H(X|Y_1 = 1) + 60/100 H(X|Y_1 = 2) \approx 4/10 + 6/10 \cdot 0.65 \approx 0.79$$

2. Split 2: In two groups, group 1 is of size 40 and contains 10 elements of A and 30 of B. I.e. the entropy of this group equals

$$H(X|Y_2 = 1) = -1/4 \log(1/4) - 3/4 \log(3/4) \approx 0.81$$

and group 2 is of size 60 and contains 60 elements of A and 0 of B. I.e. the entropy of this group equals

$$H(X|Y_2 = 2) = -1 \log(1) - 0 \log(0) = 0$$

The conditional entropy of X given Y_2 is by definition

$$H(X|Y_2) = 40/100 H(X|Y_2 = 1) + 60/100 H(X|Y_2 = 2) \approx 4/10 \cdot 0.81 + 6/10 \cdot 0 \approx 0.32$$

Observe that the conditional entropy of split 2 is lower – since split 2 creates more order, the information/surprise if we randomly pick according to the distribution is lower. If we compare now the information gains:

$$I(X, Y_1) \approx 0.09,$$
$$I(X, Y_2) \approx 0.55.$$

Remark: This concept is used when constructing decision trees – stepwise decide w.r.t the variable whose information gain is maximal.

1.15.1 Cross entropy

Suppose one uses an encoding of information based on a distribution q to model an (unknown real) distribution p – what will be the average number of bits then. Similarly to the definition of the entropy itself the expression

$$H(p, q) = -\sum_i p_i \log(q_i)$$

computes the expected number of bits. This expression is called the cross entropy of p over q.

EX: Assume the four letters a, b, c and d are the alphabet and they occur with (true) probabilities $p(a) = p_1 = 1/2$, $p(b) = p_2 = 1/4$, $p(c) = p_3 = 1/8$, and $p(d) = p_4 = 1/8$. If one (wrongly) assumes the distribution to be $p(a) = q_1 = 1/8$, $p(b) = q_2 = 1/2$, $p(c) = q_3 = 1/4$, and $p(d) = q_4 = 1/8$ one would choose

the binary encoding according to q_i as, say, $a = 110$, $b = 0$, $c = 10$ and $d = 111$. Then the average length would be

$$\sum \text{length} \cdot \text{frequency} = 3 \cdot 1/2 + 1 \cdot 1/4 + 2 \cdot 1/8 + 3 \cdot 1/8.$$

Remarks:

1. If X is a random variable with distribution p and Y is a random variable on the same sample space with distribution q, then sometimes one writes $H(X, Y)$ which should not be confused with the joint entropy discussed earlier (which is something different).

2. If $H(p)$ denotes the entropy w.r.t. distribution p, we have

   (a) $H(p, p) = H(p)$,

   (b) $H(p, q) \geq H(p)$,

   (c) $H(p, q) \neq H(q, p)$.

Relation to MLE

Let $A = \{1, 2, \ldots, i, \ldots\}$ be a (discrete) set of possible outcomes with assumed distribution $P_q(X = i) = q(i)$. Assume further this distribution depends on some parameter θ, i.e., $q(i) = q_\theta(i)$. Picking N times according to some true probability $P(X = i) = p(i)$, each outcome occurs, say, N_i times. Then, the likelihood function equals

$$L(\theta) = \prod_i q_\theta(i)^{N_i}$$

and, consequently, the log likelihood is

$$\log L(\theta) = \sum_i N_i q_\theta(i)^{N_i}$$

and therefore

$$\frac{1}{N} \log L(\theta) = \sum_i \frac{N_i}{N} \log(q_\theta(i)) \xrightarrow{N \to \infty} \sum_i p_i \log(q_\theta(i)).$$

I.e., maximizing the likelihood equals minimizing the cross entropy.

1.15.2 Max-Ent Principle

If a distribution of a random variable is only partly known, the principle of maximum entropy is frequently applied – it says:

The distribution with maximal entropy is the distribution should be chosen as least informative default.

Explanation of the maximal entropy principle

1. Intuitive: If entropy measures the average information value the outcomes of a random variable picking the distribution with maximal entropy means that in average the possible outcomes give us maximal information, i.e. we do not assume more than necessary a priori.

2. Physical: The most probable configuration of particles is often the one with maximal entropy – this fact is mentioned here because one of the founders of this maximum entropy principle – Jaynes (who, more generally, promoted the interpretation of probability as extension of logic and made many contributions to the Bayesian probability theory) – introduced it in the framework of statistical mechanics.

3. The monkey argument: Assume there are m buckets and (totally uninformed) monkeys throw independently N ($N >> m$) balls in these buckets. Then it is checked if the outcome meets the given constraints (the available prior information) – if it does the outcome is recorded, otherwise it is rejected. The denoted frequency in the i–th bucket is $p_i = \frac{n_i}{N}$, where n_i denotes the number of balls in the i–th bucket. If one repeats this experiment over and over again the expected frequencies are the probabilities that the balls land in the i–th bucket. This is computed by the so called multinomial distribution

$$\mathbf{P}(n_1, n_2, \ldots, n_m) = \underbrace{\frac{N!}{n_1! \cdot n_2! \cdots n_m!}}_{=\text{the multiplier } M} \left(\frac{1}{m}\right)^N.$$

Which outcomes are most likely given our constraints? Instead of directly maximising $\mathbf{P}(n_1, n_2, \ldots, n_m)$ we maximise the averaged logarithm of the

multiplier M. This means we maximise

$$\begin{aligned}
&\frac{1}{N}\log\left(\frac{N!}{n_1!\cdot n_2!\cdots n_m!}\right)\\
&= \frac{1}{N}\left(\log(N!)-\sum_{i=1}^{m}\log(n_i!)\right) \qquad \text{[log rules]}\\
&\approx \frac{1}{N}\left(N\log(N)-\sum_{i=1}^{m}n_i\log(n_i)\right)\\
&\qquad\qquad \text{[Stirlings formula: } \ln(n!)\approx n\ln(n)-n\text{]}\\
&= \frac{1}{N}\left((n_1+n_2+\ldots+n_m)\log(N)-\sum_{i=1}^{m}n_i\log(n_i)\right)\\
&= \frac{1}{N}\left(-\sum_{i=1}^{m}n_i\log(n_i/N)\right) \qquad \text{[log rules]}\\
&= -\sum_{i=1}^{m}p_i\log(p_i) \qquad \left[\text{ using the notation } p_i=\frac{n_i}{N}\right],
\end{aligned}$$

which means exactly we must maximise the entropy of the given distribution.

EX: Flipping a coin with no further information, i.e. $\mathbf{P}(H)=p$ and $\mathbf{P}(T)=1-p$. Then the Max Entropy distribution is maximising

$$f(p)=-p\log(p)-(1-p)\log(1-p).$$

Using $f'(p)=0$ one obtains $p=1/2$.

1.15.3 Probabilistic modus ponens

Recall the classical modus ponens

$$\frac{A \quad (A\to B)}{B}.$$

A probabilistic counterpart would be

$$\frac{\mathbf{P}(A) \quad \mathbf{P}(B|A)}{\mathbf{P}(B)}.$$

However, observe that $\mathbf{P}(B)$ is not fixed by $\mathbf{P}(A)$ and $\mathbf{P}(B|A)$ since the second summand of

$$\mathbf{P}(B) = \mathbf{P}(A, B) + \mathbf{P}(\neg A, B) = \mathbf{P}(B|A)\mathbf{P}(A) + \mathbf{P}(B|\neg A)\mathbf{P}(\neg A)$$

is not given. Introducing the distribution

1. $\mathbf{P}(A, B) = p_1$,
2. $\mathbf{P}(A, \neg B) = p_2$,
3. $\mathbf{P}(\neg A, B) = p_3$,
4. $\mathbf{P}(\neg A, \neg B) = p_4$,

one has the constraint $p_1 + p_2 + p_3 + p_4 = 1$,

$$p_1 = \mathbf{P}(B|A)\mathbf{P}(A)$$

and, since $\mathbf{P}(A) = \mathbf{P}(A, B) + \mathbf{P}(A, \neg B)$ also

$$p_2 = \mathbf{P}(A) - p_1.$$

Thus we must still fix p_3 and p_4 given the constraint $p_3 + p_4 = 1 - \mathbf{P}(A)$ (plugging $p_2 = \mathbf{P}(A) - p_1$ in $p_1 + p_2 + p_3 + p_4 = 1$). The Max Entropy principle tells us that we must maximise

$$-p_3 \ln p_3 - p_4 \ln p_4$$

given the constraint $p_3 + p_4 = 1 - \mathbf{P}(A)$. Using differential calculus again one obtains analogously to the last example

$$p_3 = p_4 = \frac{1 - \mathbf{P}(A)}{2}.$$

Thus, according to the Max Ent principle we have

$$\mathbf{P}(B) = \mathbf{P}(A)\mathbf{P}(B|A) + \frac{1 - \mathbf{P}(A)}{2}.$$

Observe that

1. If A is true, i.e., $\mathbf{P}(A) = 1$ and $A \to B$, i.e. $\mathbf{P}(B|A) = 1$ then $\mathbf{P}(B) = 1$, i.e. B is true which is the classical modus ponens.

2. However, if A is wrong, i.e. $\mathbf{P}(A) = 0$, then the classical modus ponens does not allow any conclusion. Here we obtain, independently of $\mathbf{P}(B|A)$, that $\mathbf{P}(B) = 1/2$. That is, if the premise is wrong, any outcome of B is equally likely.

3. Finally, if A is true but $A \to B$ is wrong, i.e. $\mathbf{P}(A) = 1$ and $\mathbf{P}(B|A) = 0$ then also $\mathbf{P}(B) = 0$.

2 Two first models – linear regression and logistic regression

2.1 Linear Regression - Least squares

2.1.1 The general idea

Suppose you want to model a characteristic y by a set of input characteristics $\mathbf{x}$, i.e. one wants to find a function depending on some parameters θ which satisfies $y_i \approx f_\theta(x_i)$ for a given data set $(\mathbf{x}_i, y_i)$. The least square method is a standard approach in many machine learning algorithms starting with regression analysis where a chosen model has less parameters than the data set measurements. This leads to an overdetermined set of equations which have no solution. Hence the "best" solution is searched – in the least square approach this is done by minimising the squared error of the residuals r_i, i.e., one wants to determine the parameters θ of the model f_θ such that

$$\sum_i r_i^2 = \sum_i (y_i - f_\theta(\mathbf{x}_i))^2$$

is minimised.

One nice feature of this sum of squares minimisation problem is that it can be seen as a problem in the context of functions (differential calculus) as well as a (since $\sum a_i^2 = \|\mathbf{a}\|^2$, i.e. sums of squares can be seen as the squared length of a vector) a geometrical problem.

2.1.2 Linear regression

The problem

Given is a set of n points $(x_i, y_i) \in \mathbb{R}^2$, $i = 1, \ldots, n$. The aim is to find a line $y = kx + d$ which approximates these points optimally. It turns out, measuring optimal approximation by so called least squares is a very fruitful approach:

For any line $y = kx + d$ the difference of the approximation and the real value

of the i th point (x_i, y_i) can be measured by

$$y_i - (kx_i + d).$$

This is just the difference in y direction between real value y_i and approximation $kx_i + d$. To get rid of cancellation effects and still have some nice computational properties (like differentiability) one squares these differences and says the line $y = kx + d$ optimally approaches the given set of points if the sum of square differences, so

$$\sum_{i=1}^{n} (y_i - (kx_i + d))^2,$$

is minimal.

Remark: This idea directly generalises to hyperplanes in the $\mathbb{R}^k$ approximating a point cloud of n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$.

Orthogonal Projection

Let $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k$ be some given vectors. They generate the set

$$E = \{\mathbf{x} = \lambda_1 \mathbf{a}_1 + \ldots + \lambda_k \mathbf{a}_k\}.$$

A vector $\mathbf{w}$ is orthogonal to E, notation $\mathbf{w} \perp E$, if $\mathbf{w}$ is perpendicular to all generating vectors $\mathbf{a}_i$, i.e., $\mathbf{v} \cdot \mathbf{a}_i = 0$ for $i = 1, 2, \ldots, k$.
For any vector $\mathbf{w}$ we call $Pr_E \mathbf{w}$ the orthogonal projection of $\mathbf{w}$ onto E if

$$(\mathbf{v} - Pr_E \mathbf{v}) \perp E,$$

i.e.,

$$(\mathbf{v} - Pr_E \mathbf{v}) \cdot \mathbf{a}_i = 0$$

for $i = 1, 2, \ldots, k$.

Remark: Pythagoras Theorem guarantees for any vector $\mathbf{w}$ and elements $\mathbf{x} \in E$ that

$$\|\mathbf{w} - \mathbf{x}\|^2 = \|\mathbf{w} - Pr_E \mathbf{w}\|^2 + \|Pr_E \mathbf{w} - \mathbf{x}\|^2.$$

Consequently

$$\|\mathbf{v} - \mathbf{x}\|^2 \geq \|\mathbf{v} - Pr_E \mathbf{v}\|^2,$$

i.e., $Pr_E\mathbf{w}$ is the closest point in E to any given point $\mathbf{w}$.

This idea can be applied to any non-solvable linear equation system $A\mathbf{x} = \mathbf{b}$:

$A\mathbf{x} = \mathbf{b}$ has no solution if there is no $\mathbf{x}$such that $A\mathbf{x} = \mathbf{b}$. The $\mathbf{x}$ which minimises

$$\|A\mathbf{x} - \mathbf{b}\|^2$$

is the "best possible" solution to $A\mathbf{x} = \mathbf{b}$. According to the discussion above this means that we must find $Pr_E\mathbf{b}$, where E is the set generated by the columns of A, i.e.,

$$E = \{\mathbf{x} = \lambda_1\mathbf{a}_1 + \ldots + \lambda_k\mathbf{a}_k\}.$$

analogously to above. Thus, $\mathbf{x}$ should satisfy

$$(A\mathbf{x} - \mathbf{b}) \cdot \mathbf{a}_i = 0$$

for all columns $\mathbf{a}_i$ of the matrix A. This is particularly guaranteed if

$$(A\mathbf{x} - \mathbf{b}) \cdot A\mathbf{y} = 0$$

for all vectors $\mathbf{y}$.

Writing the inner product in matrix form, i.e., $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^t\mathbf{y}$, together with $(AB)^t = B^tA^t$ gives

$$(A\mathbf{x}) \cdot \mathbf{y} = (A\mathbf{x})^t\mathbf{y} = \mathbf{x}^tA^t\mathbf{y} = \mathbf{x} \cdot A^t\mathbf{x}.$$

Consequently,

$$(A\mathbf{x} - \mathbf{b}) \cdot A\mathbf{y} = A^t(A\mathbf{x} - \mathbf{b}) \cdot \mathbf{y}.$$

$A^t(A\mathbf{x} - \mathbf{b}) \cdot \mathbf{y} = 0$ for all $\mathbf{y}$ is only possible if $A^t(A\mathbf{x} - \mathbf{b}) = \mathbf{0}$. Therefore the minimising $\mathbf{x}$ has to satisfy the so called normal equation:

$$A^tA\mathbf{x} = A^t\mathbf{b} \tag{4}$$

It can be shown that A^tA is an invertible matrix if the columns of A are linearly independent (this will be satisfied in linear regression later). In this case the normal equation (4) has the unique solution

$$\mathbf{x} = (A^tA)^{-1}A^t\mathbf{b}.$$

Remark: $(A^tA)^{-1} \neq (A^{-1}(A^t)^{-1}$, since A (and therefore also A^t) is not invertible.

The projection itself is then given by

$$\hat{\mathbf{b}} = Pr_E\mathbf{b} = \underbrace{A(A^tA)^{-1}A^t}_{=H}\mathbf{b}.$$

The matrix H is usually called the hat matrix – it projects $\mathbf{b}$ onto E.

Back to the line fitting problem:

Recall that

$$\sum_{i=1}^{n}(y_i - (kx_i + d))^2,$$

can be rewritten as

$$\left\| \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} - \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} d \\ k \end{pmatrix} \right\|^2 .$$

I.e. we want to minimize

$$\|A\mathbf{x} - \mathbf{b}\|$$

with

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} d \\ k \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Now,

$$A^TA = \begin{pmatrix} \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \\ n & \sum_{i=1}^n x_i \end{pmatrix}$$

and

$$A^T\mathbf{b} = \begin{pmatrix} \sum_{i=1}^n x_iy_i \\ \sum_{i=1}^n y_i \end{pmatrix}.$$

So the normal equation $A^T A\mathbf{x} = A^T\mathbf{b}$ reads

$$\begin{pmatrix} \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \\ n & \sum_{i=1}^n x_i \end{pmatrix}\begin{pmatrix} d \\ k \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n y_i \end{pmatrix}.$$

By the best approximation property of orthogonal approximation we know that the solution (d, k) will be a minimum.

Solving this system of linear equations explicitly for d and k gives the usual formulas

$$k = \frac{s_{XY}}{s_X^2} = \frac{r_{XY}s_Y}{s_X} \quad \text{and} \quad d = \bar{y} - k\bar{x}.$$

Here the following notation is used:

- (mean)
$$(\bar{x}, \bar{y}) = \left(\frac{\sum_{i=1}^n x_i}{n}, \frac{\sum_{i=1}^n y_i}{n}\right).$$

- (variance):
$$(s_X^2, s_Y^2) = \left(\frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x})^2, \frac{1}{n-1}\sum_{i=1}^n (y_i - \bar{y})^2\right).$$

- (standard deviation)
$$(s_X, s_Y) = (\sqrt{s_X^2}, \sqrt{s_Y^2}).$$

- (covariance)
$$\begin{aligned} s_{XY} &= \frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \\ &= \frac{1}{n-1}\left(\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}\right) \end{aligned}$$
Remark: $|s_{XY}| \leq s_X s_Y$ with $=$ if and only if the points are on a line.

- (correlation)
$$r_{XY} = \frac{s_{XY}}{s_X s_Y}$$
Remark: If $r_{XY} > 0$/ $r_{XY} \approx 0$ / $r_{XY} < 0$, X and Y are called positively correlated/ uncorrelated/negatively correlated. $r_{XY} > 0$ / $r_{XY} < 0$ shows that the point cloud is increasing/ decreasing.

Remark: The idea to solve the least square problem by orthogonal approximation and the normal equation directly generalises, e.g., to multidimensional linear regression, i.e.

$$y = \beta_0 + \beta_1 x^{(1)} + \beta_2 x^{(2)} + \ldots + \beta_l x^{(l)}.$$

2.1.3 MLE and least squares

After the discussion so far the following connection can be pointed out:

The standard assumptions are that the residuals $r_i = y_i - f_\theta(\mathbf{x}_i)$

1. are all 0-σ-normally distributed (with some fixed $\sigma > 0$) and
2. independent.

Given those assumptions the likelihood for observing a set of residuals is given by the density function of r_i, i.e.,

$$p(r_i)\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{r_i^2}{\sigma^2}}.$$

Using independence and plugging in for r_i the joint likelihood for a measurement $(\mathbf{x}_i, y_i)$ becomes

$$\begin{aligned} p_\theta(x_1, x_2, \ldots, x_n) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{r_i}{\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(\mathbf{x}_i))^2} \end{aligned}$$

Then the MLE approach is to fix the parameter θ such that the joint likelihood is maximised. But, due to the minus in the exponent, this is equivalent to solving the least square problem "minimise $\sum_{i=1}^{n}(y_i - f_\theta(\mathbf{x}_i))^2$".

2.2 Logistic Regression

2.2.1 First steps - logit function

Whilst linear regression models continuous response data (real valued y values), logistic regression targets classification problems. We focus here on binary logistic regression, i.e., the input data x is assigned to two categories (0 and 1). Moreover, we restrict ourself to one dimensional input data, i.e. we wish to fit a model to given points (x_i, b_i), $i = 1, \ldots, n$, $x_i \in \mathbb{R}$, $b_i \in \{0, 1\}$.

Instead of directly modelling the categorisation, i.e. "to which of the two classes does a given point x belong?", logistic regression models for a given point x the probability that x belongs to class 1. We denote this probability as $p(x)$.

Some sort linear regression model would be a nice choice since it is simple and well understood. However, a linear function obviously takes values in $\mathbb{R}$ while probabilities are only between 0 and 1. To overcome this, two preliminary steps are applied

1. (odds) Instead of probabilities, one works with odds, i.e. if event has probability p, its odds are

$$odds(p) = \frac{p}{1-p}.$$

   EX: The odds to roll a 6 are 1/5, i.e., "the odds are 1 to 5".

   $odds(p)$ is a monotonically increasing function with $odds(0) = 0$ and $odds(1) = \infty$. Moreover, for $p < 1/2$, $odds(p) < 1$.

2. (logit) One defines

$$logit(p) = \ln(odds(p)) = \ln\left(\frac{p}{1-p}\right).$$

   $logit(p)$ is now a monotonically increasing function taking values in $\mathbb{R}$, $logit(0) = -\infty$, $logit(1/2) = 0$, $logit(1) = \infty$.

Thus, for real input values x we have established an $\mathbb{R} \to \mathbb{R}$ map $x \to logit(p(x))$, where $p(x)$ still denotes the probability that x belongs to class 1, i.e. a way to use linear regression for probabilities:

The aim of logistic regression is to determine parameters β_0 and β_1 such that

$$logit(p(x)) = \beta_1 x + \beta_0.$$

Remark: The max-entropy principle can be applied to justify the chosen logit function. In this sense the logit function is the optimal choice among all functions

2.2.2 logit and sigmoid

Observe that

$$logit(p) = \ln(odds(p)) = \ln\left(\frac{p}{1-p}\right) = z$$

if and only if

$$p = \underbrace{\frac{1}{1+e^{-y}}}_{=\sigma(z)}.$$

This function $\sigma(z)$ is called the sigmoid function.

Thus we can reformulate the initial model formulation: We want to determine β_0 and β_1 such that

$$p(x) = \sigma(\beta_1 x + \beta_0) = \frac{1}{1+e^{-(\beta_1 x+\beta_0)}}\ , \tag{5}$$

the probability the point x belongs to class 1, fits the given data optimally.

2.2.3 Training the model

Given the data (x_i, b_i), $i = 1, \ldots, n$, $x_i \in \mathbb{R}$, $b_i \in \{0, 1\}$, we want to fit the model explained so far, i.e., given in equation 5. There are two ways to argue, both lead to the same result.

1. (Training using MLE) Since the response variable b is binary, we have a Bernoulli experiment, i.e., if $p(x)$ denotes the probability that the input x belongs to class 1, then $1 - p$ denotes the probability that the input x belongs to class 0. In short we can denote this distribution as

$$p^b(1-p)^{1-b}.$$

Applying this, we get the following likelihood function for the given data:

$$L = L(\beta_0, \beta_1) = \prod_{i=1}^{n} p(x_i)^{b_i} (1 - p(x_i))^{1-b_i},$$

where the unknown parameters β_0 and β_1 are parameters of $p(x)$. Taking, as usual, the log-likelihood, one wants to determine β_0 and β_1 which maximise

$$\ln L(\beta_0, \beta_1) = \sum_{i=1}^{n} b_i \ln(p(x_i)) + (1 - b_i) \ln(1 - p(x_i)).$$

Observe that in the last sum only one summand is not equal to 0 for each i.

2. (Training using cross entropy) Binary data can be seen as a probability distribution. If point x belongs to class b, $p'(x)$, the true probability of point x belonging to class 1 satisfies $p'(x) = b$. More detailed: If x_i belongs to class 0, i.e. $b_i = 0$, $p'(x)$, the true probability of point x_i belonging to class 1 satisfies $p'(x_i) = 0 = b_i$ and if x_i belongs to class 0, i.e. $b_i = 1$, $p'(x)$, the true probability of point x_i belonging to class 1 satisfies $p'(x_i) = 1 = b_i$. Since our model given by 5 should approximate the given true probability for each point x_i, we wish to minimise the difference between the distributions $p(x_i)$ and $p'(x_i)$ for each $i = 1, \ldots, n$. One tool to measure this difference is cross entropy, the overall cross entropy (for all points of the data) is the sum of the individual cross entropies. I.e. we wish to minimise

$$\sum_{i=1}^{n} H(p'(x_i), p(x_i)) = -\left(\sum_{i=1}^{n} b_i \ln(p(x_i)) + (1 - b_i) \ln(1 - p(x_i)) \right).$$

Observe that this is the negative version of the equation given by the MLE approach, i.e. minimising the cross entropy gives the same result as maximising the likelihood function.

Unfortunately there is no explicit way to compute the parameters β_0 and β_1 such that the likelihood function is maximised/ the cross entropy is minimised, numerical methods have to be applied. The usual numerical algorithm is gradient descent, which will be discussed in detail later in this course. For the sake of completeness the learning algorithm is still mentioned here:

1. Initialize (β_0^0, β_1^0) randomly.

2. Pick a so called mini batch (a subsample of the (training) data) (x_j, b_j), $j = 1, \ldots, m$ and update (β_0^l, β_1^l) for $l = 1, 2, \ldots$ until some termination condition is satisfied:

$$\beta_0^l \leftarrow \beta_0^{l-1} - \eta \sum_{j=1}^{m} \left(p_{(\beta_0^{l-1}, \beta_1^{l-1})}(x_j) - b_j \right)$$

and

$$\beta_1^l \leftarrow \beta_1^{l-1} - \eta \sum_{j=1}^{m} \left(p_{(\beta_0^{l-1}, \beta_1^{l-1})}(x_j) - b_j \right) x_j,$$

where η denotes the so called learning rate, which is a hyper-parameter of the algorithm.

3 General aspects of machine learning

3.1 Basic definitions

Artificial intelligence

From Wikipedia:

Artificial intelligence (AI) is the intelligence of machines or software, as opposed to the intelligence of other living beings, primarily humans. It is a field of study in computer science that develops and studies intelligent machines. Such machines may be called AIs.

The general problem of simulating (or creating) intelligence has been broken into sub-problems. These consist of particular traits or capabilities that researchers expect an intelligent system to display, like

1. Reasoning, problem-solving
2. Knowledge representation
3. Planning and decision making
4. Learning

Machine learning

Machine learning (ML) is a field of study in artificial intelligence concerned with the development and study of algorithms that can learn from data and generalize to unseen data, and thus perform tasks without explicit instructions

3.1.1 Types of ML

Machine learning approaches are traditionally divided into three broad categories, which correspond to learning paradigms, depending on the nature of the "signal" or "feedback" available to the learning system:

1. Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn

a general rule that maps inputs to outputs.

2. Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

3. Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize.

3.2 Further concepts

3.2.1 Cross validation

Usually the data set is split in three groups

1. Training set: A training data set is a data set of examples used during the learning process and is used to fit the parameters (e.g., weights).

2. Validation data set: A validation data set is a data-set of examples used to tune the hyper-parameters (i.e. the architecture) of a model. The model is never trained on the validation set, however the model sees the validation set several times and the layout of the model is adjusted based on this set.

3. Test data set: A test set is a set of examples used only to assess the performance (i.e. generalisation) of a fully specified model. To do this, the final model is used to predict examples in the test set. Those predictions are compared to the examples' true classifications to assess the model's accuracy.

4. Cross validation: In order to get more stable results and use all valuable data for training, a data set can be repeatedly split into several training and a validation datasets. This is known as cross-validation. To validate the model performance, an additional test data set held out from cross-validation is normally used.

Usual ratios are around x = 70% of the total data set as training data and $\frac{100-x}{2}$ % for validation and testing.

3.2.2 Bias Variance tradeoff

An important fact of ML models is the so called bias variance trade-off. The error a chosen model makes when predicting outcomes can be written as sum of three errors:

1. Bias: The error the model cannot learn because of wrong assumptions of the models. Typically the bias is bigger the simpler the model is chosen since the model cannot learn the detailed properties of the data. A high-bias model is most likely to under-fit the training data.

2. Variance: The error which is the consequence of the sensitivity of the model to unexplainable variations in the training data. Complex models tend to have a bigger variance and are sensitive to overfitting.

3. Irreducible Error: This is the unexplainable random error due to the noisiness of the data itself. Unknown underlying features as well as measurement errors are typical sources of this error.

As indicated the more complex a model is, the smaller is the bias but the larger is the variance, this fact is called the bias-variance-tradeoff. A suitable model for given data/ setting satisfies that both errors are low, i.e. one which lies at the intersection of the curves bias vs model complexity and variance vs model complexity.

3.2.3 Generative vs discriminative models

In statistical classification there are generative models as well as discriminative models. In probabilistic terms, the aim of classification tasks is to compute $\mathbf{P}(Y|X)$, where X are the input features and Y the classes in question.

1. Informally, generative models learn the distribution of each class. More precisely: generative models learn the joint distribution $\mathbf{P}(X, Y)$ (or, equivalently, $\mathbf{P}(X|Y)$ and $\mathbf{P}(Y)$) and use this (i.e. Bayes theorem) to obtain $\mathbf{P}(Y|X)$.

Example: Naive Bayes classifier

2. Discriminative models, on the other hand, directly learn (some functional form) of $\mathbf{P}(Y|X)$ from the training data.

   Example: Logistic regression

3.2.4 Imbalanced data

In classification tasks one typical problem is that some classes are highly overrepresented in the data which - in worst case - causes the model to ignore the underrepresented classes and still get good results on the training data. In this context one calls

1. the majority class: the class with the highest number of samples,
2. the minority class the class with the lowest number of samples
3. the class ratio: the ratio of the size of the minority class and the majority class. A class ratio (much) lower than 25% can become problematic

To deal with such data sets there exist several approaches, we mention three:

1. Random undersampling: One downsamples the larger classes by picking randomly from each class. The rate of downsampling is a hyperparameter which must be optimized while the model is (fine-) tuned.
2. Random oversampling: Randomly picked elements from the smaller classes are copied and added to the data set until the data set is more balanced.
3. S(ynthetic) M(inority) O(versampling) TE(chnique): Smaller classes are oversampled by generating new (sythnetic) data points according to the following idea:
   (a) Select a random point x_i of the minority class and for this point a second randomly chosen point x_j of the k–nearest neighbours of x_i.
   (b) Pick a random number t in $[0, 1]$.
   (c) Add the new point $x_i + t(x_j - x_i)$ to the minority class.

This process is again repeated until the data set is more balanced.

3.2.5 Curse of dimensionality

The problem that a finite number of data points cannot fill every "corner" of a potentially very high dimensional feature space leads to the fact that that more included features do not necessarily improve the predictive power of the model.

In machine learning he curse of dimensionality is equivalent to the so called peaking phenomenon: This phenomenon states that with a fixed number of training samples, the average (expected) predictive power of a classifier or regressor first increases as the number of dimensions or features used is increased but beyond a certain dimensionality it starts deteriorating instead of improving steadily

Some rules of thumbs concerning the sample sizes (which are more often wrong than right):

1. Classification tasks: For C classes and n features use at least $10 * n * C$ data points.

2. Regression tasks: For n features use $50 * n$ data points.

3. If the data set is small (< 1000): Simple models like Naive Bayes, Decision trees, Regression, Logistic Regression, ...

4. If the data set is of intermediate size (> 1000 and < 10000): Random forest, boosted trees, ...

5. If the data set is larger: (> 10000): Neural networks

At this point the so called **no free lunch theorem** of optimization can be mentioned: It states that the cost (in the light of complexity) for solving an optimization problem in a given class is the same for every solution method available. This means there is no single generic optimal model which works for all problems of one type (e.g. classification tasks). It is therefore important to find the best model with the optimally chosen hyper-parameters for a given concrete problem.

3.3 Model Evaluation

To evaluate a model one distinguishes between classification tasks and regression tasks:

1. (**Classification tasks**:)

   (a) **Accuracy**: Accuracy is the ratio of correctly predicted instances to the total instances in the dataset. It is calculated as:

   $$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Instances}}$$

   Rule of Thumb:

   - High accuracy may indicate a good overall performance, but it can be misleading in imbalanced datasets.

   (b) **Precision**: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It is calculated as:

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

   Rule of Thumb:

   - High precision indicates that the model is making fewer false positive predictions.
   - Precision is useful when the cost of false positives is high.

   (c) **Recall (Sensitivity)**: Recall measures the proportion of true positive predictions among all actual positive instances in the dataset. It is calculated as:

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

   Rule of Thumb:

   - High recall indicates that the model is capturing most of the positive instances.
   - Recall is useful when the cost of false negatives is high.

(d) **F1 Score**: The F1 score is the harmonic mean of precision and recall. It is calculated as:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Rule of Thumb:

- F1 score provides a balance between precision and recall.
- A high F1 score indicates good overall performance.

Remark: : The harmonic mean is a type of average that is calculated by taking the reciprocal of the arithmetic mean of the reciprocals of a set of numbers. It is particularly useful when dealing with rates or ratios.

Given a set of n positive numbers $x_1, x_2, \ldots, x_n$, the harmonic mean H is calculated as:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

For example, if we have two numbers x_1 and x_2, the harmonic mean is:

$$H = \frac{2}{\frac{1}{x_1} + \frac{1}{x_2}} = \frac{2x_1x_2}{x_1 + x_2}$$

The harmonic mean tends to be lower than the arithmetic mean and is more influenced by smaller values. It is often used in situations where the rates or ratios need to be averaged, such as in the case of average speed, average rates of return, or in combining precision and recall to compute the F1 score in classification evaluation.

(e) **ROC Curve (Receiver Operating Characteristic Curve)**: The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

(f) **Area Under the ROC Curve (AUC-ROC)**: AUC-ROC quantifies the overall performance of a binary classification model by

calculating the area under the ROC curve.

(g) **Confusion Matrix**: A confusion matrix provides a tabular summary of the actual versus predicted class labels produced by the classification model. It includes true positives, true negatives, false positives, and false negatives.

2. (**Regression tasks**:)

(a) **Mean Absolute Error (MAE)**: MAE measures the average absolute difference between the predicted and actual values. It is calculated as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Rule of Thumb:

- MAE is easy to interpret and gives a good indication of average prediction error.

(b) **Mean Squared Error (MSE)**: MSE measures the average squared difference between the predicted and actual values. It is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Rule of Thumb:

- MSE penalizes larger errors more than smaller ones and is useful for identifying outliers.

(c) **Root Mean Squared Error (RMSE)**: RMSE is the square root of the MSE and provides the same unit of measurement as the target variable. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Rule of Thumb:

- RMSE is widely used as it is interpretable in the same units as the target variable.

(d) **Mean Absolute Percentage Error (MAPE)**: MAPE measures the average percentage difference between the predicted and actual values. It is calculated as:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%$$

Rule of Thumb:

- MAPE is useful for interpreting prediction errors as a percentage of the actual values.

(e) **Coefficient of Determination (R^2 Score)**: R^2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the observed data. **Rule of Thumb**:

- R^2 score ranges from 0 to 1, where 1 indicates a perfect fit and 0 indicates no improvement over the mean model.

From Wikipedia, the free encyclopedia

Sources: [1][2] [3][4][5][6][7][8] **view** · talk · edit

		Predicted condition			
	Total population $= P + N$	**Predicted Positive (PP)**	**Predicted Negative (PN)**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)** [a]	**True positive** (TP), hit[b]	**False negative** (FN), miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate type II error [c] $= \frac{FN}{P} = 1 - TPR$
	Negative (N)[d]	**False positive** (FP), false alarm, overestimation	**True negative** (TN), correct rejection[e]	False positive rate (FPR), probability of false alarm, fall-out type I error [f] $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{PN} = 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2\,PPV \times TPR}{PPV + TPR} = \frac{2\,TP}{2\,TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

Figure 1: Overview of Error measurments

4 Classification and decision trees

Another supervised learning architecture are the so called classification and decision trees (CART). Analogous to the regression we differentiate between a categorial and a quantitive target value.

4.1 Regression trees

Regression trees deal with the problem of modelling data

$$D = \{(\mathbf{x}_i, y_i), i = 1, \dots m\},$$

where $\mathbf{x}_i$ is an n dimensional input vector and $y_i \in \mathbb{R}$ is a numerical target value.

The tree T is built iteratively, the error is typically measured using again the mean squared error

$$E(T, D) = \frac{1}{m} \sum_{i=1}^{m} (f_T(\mathbf{x}_i) - y_i)^2,$$

where $f_T(\mathbf{x}_i)$ is the prediction given by the tree model.

The tree is then build by the following steps:

1. The leaves l of the (currently constructed) tree define a partition of D into sets R_l, at the beginning of the algorithm the only leaf is the root and the set is whole D.

2. The elements of each of the sets R_l then define the output of the model by
$$f_T(\mathbf{x}_i) = mean(y_k \ : \mathbf{x}_k \in R_k \text{ and } \mathbf{x}_i \in \mathbb{R}_k) = m_k$$

3. To find a next split in the tree take one leaf l and its set R_l and find for each feature j, i.e. dimension, of $\mathbf{x}$ the best split s such that the split leads to the lowest possible error, i.e. s satisfies
$$s = \arg\min_s \left(\sum_{x_j \in R_l^{<s}} (c_1 - y_i)^2 + \sum_{x_j \in R_l^{>s}} (c_2 - y_i)^2 \right),$$

where $R_l^{<s}$ denotes those elements of R^l such that $x_j < s$ and $R_l^{>s}$ denotes those elements of R^l such that $x_j > s$ and $c_1 = mean(y_k : \mathbf{x}_k \in R_l^{<s})$ as well as $c_2 = mean(y_k : \mathbf{x}_k \in R_l^{>s})$. Finally fix the feature whose new split gives the overall lowest error as a new split and the sets $R_l^{<s}$ and $R_l^{>s}$ become the new leaves.

4. Continue computing new leaves until the leaves contain only one element or (typically) until a termination criterion is met (like maximal depth, minimal number of elements or a regularisation condition as discussed in the next paragraph).

Pruning

One concept which prevents overfitting is to include a so called regularization term which punishes the complexity of the tree. Such a regularisation, also called pruning, can also be independent of the given data, however, here we mention the following approach:

Let T be a tree with $|T|$ many leaves. Let the error at each leaf l be, similar to above, be given by

$$E(T, R_l) = \frac{1}{|R_l|} \sum_{\mathbf{x}_i \in R_l} (m_l - y_i)^2.$$

Then reduce the tree T to some subtree by successively deleting the last splits to find a tree S which minimizes

$$C(\alpha, S, D) = \sum_{l=1}^{|S|} |R_l| E(S, R_l) + \alpha |S|$$

in dependence of the regularisation parameter α.

4.2 Decision trees

We discuss the case if the output variables are categorial. Given is again data

$$D = \{(\mathbf{x}_i, y_i), i = 1, \dots m\},$$

where $\mathbf{x}_i$ is an n dimensional input vector but now $y_i \in \{1, 2, \dots, K\}$ is an element of K classes. In this case a tree which models the output based on the data is called a decision tree. As before the tree is build iteratively,

in this case the error is measured by quantifying the purity of the data corresponding to the leaves of the tree.

We define the probability that a randomly picked element of the set R_l of data points corresponding to leaf l belongs to class k as the relative frequency, i.e.

$$p_l(k) = \frac{|\{y_i \ : \ y_i = k \text{ and } \mathbf{x}_i \in R_l\}|}{|R_l|}.$$

The model returns the class with highest probability, i.e.

$$f_T(\mathbf{x}_i \in R_l) = \arg\max_k p_l(k).$$

Here we mention two possibilities of error-quantification, i.e. two ways of measuring the impurity:

1. Entropy: As introduced in section 1.15 the conditional entropy and information gain can be used to quantify the purity of a split - as in the case of regression trees one defines new leaves along splits which maximise the information gain or, equivalently, minimise the conditional entropy.

2. Gini impurity: The Gini impurity is based on the idea that in a pure set the probability of picking two elements (with replacement) of different classes should be low. The probability of picking elements of two different classes in a twofold pick equals

$$G(\mathbf{p}) = 1 - \sum_{k=1}^{K} p_l(k)^2.$$

Entropy as well as Gini impurity are 0 if a class is pure. Therefore, to construct the decision tree one proceeds similarly to the construction of the regression tree - split the tree in each step such that the error is minimized.

4.3 Random forests and boosting

Decision trees are simple models (sometimes also called weak learners). Small trees do have a big bias. However, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have

low bias, but very high variance. However, it is possible to improve the performance of such trees with the following ideas.

4.3.1 Random forests

Random forest is an ensemble learning method where many trees are trained simultaneously. To make a prediction one takes for classification tasks the class which is predicted by most of the trees and for regression the mean of the predictions of the individual trees.

To construct the forest one applies a technique called bagging. One either uses a bootstrapping technique for the given data, i.e. one picks from the given data several sub-datasets and trains with each of those one decision tree. This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Alternatively the method of feature-gagging is also commonly used. Here trees are trained on random selections of the available features.

Furthermore there exists the technique of so called extra trees. Here the splits are chosen randomly and from several of those random splits the best possible is chosen. This also prevents overfitting.

4.3.2 Boosting

Boosting is also an ensemble learning method where several trees are combined to obtain a better model. In boosting the errors (i.e. residuals) of the previously constructed tree used to train a next tree. There are several ways to train and combine trees constructed based on this idea leading to algorithms like gradient boosting, ada boosting or xgboost.

5 Hidden Markov Model

5.1 Preliminaries

A Hidden Markov model (HMM) is a probabilistic graph based model consisting of the ingredients

1. A finite Markov model of (hidden) states h_i fixed by its transition probabilities given as a matrix A where a_{ij} denotes the probability to move from state h_i to state h_j.

2. A finite set of possible observations o_l and for each hidden state h_i the emission probability distribution $b_i(o_l)$ which is the probability that observation o_l will occur when in state h_i. These distributions can be summed up in a matrix B. In this text we write the distributions column wise, leading to am $m \times n$ matrix, if there are m possible observations and n hidden states.

Here is one example of an HMM (A, B) with three hidden states h_1, h_2, h_3, a set of observations $\mathbf{o}$ and transition as well as emission probabilities A and B:

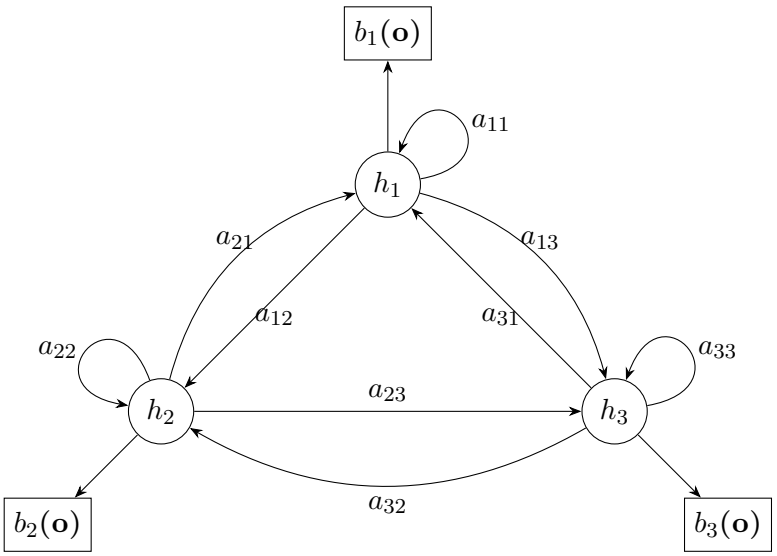


Figure 2: Hidden Markov Model with three hidden states and its emission distributions

A HMM allows to make statements based on a chain of observations $O = o_1, o_2, \ldots, o_T$ which are related to a chain of hidden states $Q = q_1, q_2, \ldots, q_T$. In order to work with HMMs two important assumptions are made:

1. Markov assumption: $\mathbf{P}(q_i|q_1, q_2, \ldots, q_{i-1}, o_1, o_2, \ldots, o_{i-1}) = \mathbf{P}(q_i|q_{i-1})$, i.e., the probability to state q_i to appear depends only on the previous state.

2. Output independence: $\mathbf{P}(o_i|q_1, q_2, \ldots, q_T, o_1, o_2, \ldots, o_{i-1}) = \mathbf{P}(o_i|q_i)$, i.e., the probability to observe o_i depends only on the hidden state underneath it.

There are three fundamental problems related to HMMs which will be discussed in the following sections.

5.2 Likelihood

Given an HMM (A, B) and an observation sequence $O = o_1, o_2, \ldots, o_T$, what is it's likelihood, i.e. $\mathbf{P}(O)$?

The main tool for working with HMMs is the chain rule for probabilities, i.e.,

$$\mathbf{P}(A_1 \ldots A_n) = \mathbf{P}(A_n|A_1 \ldots A_{n-1})\ \mathbf{P}(A_{n-1}|A_1 \ldots A_{n-1}) \cdots \mathbf{P}(A_2|A_1)\mathbf{P}(A_1)$$

combined with the assumptions of HMMs.

From this we immediately obtain

$$\begin{aligned}\mathbf{P}(O) &= \sum_{q_1,\ldots,q_T} \mathbf{P}(o_1, o_2, \ldots, o_T, q_1, q_2, \ldots, q_T) \\ &= \sum_{q_1,\ldots,q_T} P(o_1|q_1)P(o_2|q_2) \cdots P(o_T|q_T)\end{aligned}$$

where the sum is taken over all N^T hidden state combinations.

To speed up this naive approach an dynamic programming approach is taken which leads to the so called forward algorithm:

The forward algorithm of a given HMM (A, B) recursively computes the values

$$\alpha_t(j) = \mathbf{P}(o_1, o_2, \ldots, o_t, q_t = i),$$

i.e. the probability to observe the output chain $o_1, o_2, \ldots, o_t$ and be in the hidden state i.

This can be computed easily recursively since (using Markov and independence assumption as well as the chain rule):

$$\begin{aligned}
\alpha_t(j) &= \mathbf{P}(o_1, \ldots, o_t, q_t = j) \\
&= \sum_{i=1}^{N} \mathbf{P}(o_1, \ldots, o_t, q_{t-1} = i, q_t = j) \\
&= \sum_{i=1}^{N} \mathbf{P}(o_t | o_1, \ldots, o_{t-1}, q_{t-1} = i, q_t = j) \mathbf{P}(q_t = j | o_1, \ldots, o_{t-1}, q_{t-1} = i) \\
&\qquad \mathbf{P}(o_1, \ldots, o_{t-1}, q_{t-1} = i) \\
&= \sum_{i=1}^{N} \mathbf{P}(o_t | q_t = j) \mathbf{P}(q_t = j | q_{t-1} = i) \mathbf{P}(o_1, \ldots, o_{t-1}, q_{t-1} = i) \\
&= \sum_{i=1}^{N} b_j(o_t) a_{ij} \alpha_{t-1}(i)
\end{aligned}$$

where the sum is taken over all hidden states of the model.

Thus, given some initial distribution $\pi_1, \ldots \pi_N$, where π_i is the probability of being in hidden state i at the start (usually the equilibrium distribution is chosen, i.e. the eigenvector of A to the Eigenvalue 1 of A), the forward algorithm reads as follows:

1. Start: $\alpha_1(j) = \pi_j b_j(o_1)$ for $1 \leq j \leq N$, the probability for observing o_1 when starting according to π.
2. Forward step: $\alpha_t(j) = \sum_{i=1}^{N} b_j(o_t) a_{ij} \alpha_{t-1}(i)$ for $1 \leq t \leq T$ and $1 \leq j \leq N$.
3. Result: $\mathbf{P}(O) = \sum_{i=1}^{N} \alpha_T(j)$.

Remarks:

1. Observe how the algorithm uses previously computed values. This tabulation property is why the algorithm is seen as a dynamic programming algorithm.
2. The runtime of the forward algorithm is $O(N^2 T)$: For a fixed $t > 1$ every $\alpha_t(j)$ is computed as a sum of N elements, i.e. in $O(N)$ and there are N different such $\alpha_t(j)$'s, which are therefore all computed

in $O(N^2)$ steps. Finally there are T such t's which gives the claimed runtime.

5.3 Decoding

Given an HMM (A, B) and an observation sequence $O = o_1, o_2, \ldots, o_T$, what is the most likely sequence of underlying hidden states.

This problem is solved by the so called Viterbi algorithm, which is also a dynamic programming algorithm.

Obviously one could derive the most like sequence of hidden states by computing the likelihood for each possible hidden state sequence using the forward algorithm. But since there are exponentially many hidden state sequences, this, again, is not feasible.

The idea is closely related to the idea of the forward algorithm – find the most likely sequence recursively. For this define the so called Viterbi path probability

$$v_t(j) = \max_{q_1,\ldots,q_{t-1}} \mathbf{P}(q_1, q_2, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j)$$

which is the probability that the most probable path to hidden state j at time t takes.

Observe that, analogously to the previous section, we can compute

$$\begin{aligned}
v_t(j) &= \max_{q_1,\ldots,q_{t-1}} \mathbf{P}(q_1, q_2, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j) \\
&= \max_{i=1}^{N} \max_{q_1,\ldots,q_{t-2}} \mathbf{P}(q_1, q_2, \ldots, q_{t-2}, o_1, o_2, \ldots, o_t, q_{t-1} = i, q_t = j) \\
&= \max_{i=1}^{N} \max_{q_1,\ldots,q_{t-2}} \mathbf{P}(o_t | q_t = j) \mathbf{P}(q_t = j | q_{t-1} = i) \\
&\qquad \mathbf{P}(q_1, q_2, \ldots, q_{t-2}, o_1, o_2, \ldots, o_{t-1}, q_{t-1} = i) \\
&= \max_{i=1}^{N} b_j(o_t) a_{ij} v_{t-1}(i).
\end{aligned}$$

I.e., we have established a recursion similar to the forward algorithm. The splitting of the max Operator is where dynamic programming comes in – to obtain the global optimum use the local optimal values.

Using this recursion we are able to formulate the Viterbi algorithm:

1. **Initialization:** Set $v_1(j) = \pi_j b_j(o_1)$ for $1 \leq j \leq N$, where N is the number of states, π_j is the initial probability of state j, and $b_j(o_1)$ is the emission probability of observation o_1 given state j.

2. **Recursion:** For $t = 2$ to T, where T is the number of observations:

   - For each state j:

     – Compute $v_t(j) = \max_{1 \leq i \leq N}[v_{t-1}(i)a_{ij}]b_j(o_t)$, where a_{ij} is the transition probability from state i to state j, and $b_j(o_t)$ is the emission probability of observation o_t given state j.

     – Store the index $\iota_t(j)$ that maximized the above expression.

3. **Termination:** Set $\mathbf{P}^* = \max_{1 \leq i \leq N} v_T(i)$ as the probability of the most likely sequence of states.

4. **Backtracking:** Set $q_T^* = \arg\max_{1 \leq i \leq N} v_T(i)$. For $t = T - 1$ down to 1, set $q_t^* = \iota_{t+1}(q_{t+1}^*)$ as the state in the most likely sequence.

5. The most likely sequence of states is $q^* = (q_1^*, q_2^*, \ldots, q_T^*)$, and its probability is $\mathbf{P}^*$.

Remark: The running time of the Viterbi algorithm is again $O(N^2T)$.

5.4 Learning

Learning is the most general task - it is one example of an unsupervised algorithm. Given is only an observed sequence O and a predefined number of hidden states – which HMM (A, B) describes the given sequence O best, i.e. which HMM (A, B) maximizes $\mathbf{P}(O)$? The algorithm discussed here is the Baum Welch Algorithm which iteratively updates a given HMM (A, B) to obtain a new HMM (A', B')

Remarks:

1. It can be shown that $\mathbf{P}_{(A',B')}(O) \geq \mathbf{P}_{(A,B)}(O)$ and that the Baum Welch Algorithm always converges to some local maximum.

2. The Baum Welch Algorithm is an instance of a class of algorithms called Expectation-Maximisation Algorithms.

The Baum Welch Algorithm is based on the so-called forward - backward algorithm. The forward algorithm was discussed in Section 5.2, the backward algorithm is its direct counterpart, which also addresses the question: Given is an HMM (A, B) and an $O = o_1, o_2, \ldots, o_T$, what is it's likelihood, i.e. $\mathbf{P}(O)$? but now backwards:

Let

$$\beta_t(i) = \mathbf{P}(o_{t+1}, o_{t+1}, \ldots, o_T | q_t = i)$$

be the probability to observe the tail of O given that the hidden state at t is i.

Then one computes similarly to the forward algorithm

1. **Initialization:** Set $\beta_T(i) = 1$ for $1 \leq i \leq N$, where N is the number of states.

2. **Recursion:** For $t = T - 1$ down to 1:

   For each state i: Compute $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$, where a_{ij} is the transition probability from state i to state j, $b_j(o_{t+1})$ is the emission probability of observation o_{t+1} given state j, and $\beta_{t+1}(j)$ is the backward probability at time $t+1$ for state j.

3. **Termination:** The probability of observing the entire sequence O given the model is $\Pr(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$, where π_i is the initial probability of state i and $b_i(o_1)$ is the emission probability of observation o_1 given state i.

The two probabilities

$$\alpha_t(j) = \mathbf{P}(o_1, o_2, \ldots, o_t, q_t = i),$$
$$\beta_t(i) = \mathbf{P}(o_{t+1}, o_{t+1}, \ldots, o_T | q_t = i)$$

are now the main ingredients for the forward-backward algorithm to compute the updated estimates for the transition probabilities A as well as the emission probabilities B:

1. **(Transition probabilities:)** One estimates the transition probabilities by the fraction

$$a'_{ij} \approx \frac{\text{Expected transitions from state } i \text{ to state } j}{\text{Expected overall transitions from state } i}.$$

This is done by computing for each time t the expression

$$\eta_t(i,j) = \mathbf{P}(q_t = i, q_{t+1} = j|O).$$

Using $\alpha_t(i)$ and $\beta_t(i)$ one can directly compute

$$\begin{aligned}
&\mathbf{P}(q_t = i, q_{t+1} = j, O) \\
&\quad = \mathbf{P}(q_t = i, q_{t+1} = j, o_1, \ldots, o_T) \\
&\quad = \mathbf{P}(o_{t+2}, \ldots, o_T|o_1, \ldots, o_{t+1}, q_t = i, q_{t+1} = j) \\
&\qquad\quad \mathbf{P}(o_1, \ldots, o_{t+1}, q_t = i, q_{t+1} = j) \\
&\quad = \mathbf{P}(o_{t+2}, \ldots, o_T|q_{t+1} = j)\mathbf{P}(o_{t+1}|q_{t+1} = j) \\
&\qquad\quad \mathbf{P}(q_{t+1} = j|q_t = i)\mathbf{P}(o_1, \ldots, o_t, q_t = i) \\
&\quad = \alpha_t(i)a_{ij}b_{t+1}(j)\beta_{t+1}(j)
\end{aligned}$$

and, using the definition of conditional probability $\mathbf{P}(A|B) = \frac{\mathbf{P}(A,B)}{\mathbf{P}(B)}$, as well as $P(O) = \sum_{i=1}^{N} \alpha_t(j)\beta_t(i)$, which also is a consequence of the definition of conditional probability since

$$\begin{aligned}
\mathbf{P}(o_1, \ldots, o_T) &= \sum_{i=1}^{N} \mathbf{P}(o_1, \ldots, o_T, q_t = i) \\
&= \sum_{i=1}^{N} \mathbf{P}(o_{t+1}, \ldots, o_T|o1, \ldots o_t, q_t = i)\mathbf{P}(o_1, \ldots o_t, q_t = i) \\
&= \sum_{i=1}^{N} \mathbf{P}(o_{t+1}, \ldots, o_T|q_t = i)\mathbf{P}(o_1, \ldots o_t, q_t = i) \\
&= \sum_{i=1}^{N} \alpha_t(j)\beta_t(i),
\end{aligned}$$

we obtain

$$\eta_t(i,j) = \mathbf{P}(q_t = i, q_{t+1} = j|O) = \frac{\alpha_t(i)a_{ij}b_{t+1}(j)\beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_t(j)\beta_t(i)}.$$

$\eta_t(i,j)$ computes the probability of the transition from i to j at time t based on the (current) HMM and the given observation. To get the expected transition probability over O one sums over all t, i.e.

$$a'_{ij} = \frac{\sum_{t=1}^{T} \eta_t(i,j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \eta_t(i,j)}.$$

2. **(Emission probabilities:)** The computation is similar to the discussion so far - one gets the updated emission probability by

$$b'_j(o) \approx \frac{\text{Expected number of times in state } j \text{ and observing } o}{\text{Expected number of times in state } j}.$$

This is computed by

$$\gamma_t(j) = \mathbf{P}(q_t = i|O) = \frac{\mathbf{P}(q_t = i, O)}{\mathbf{P}(O)} = \frac{\alpha_t(j)\beta_t(j)}{P(O)}.$$

Summing $\gamma_t(j)$ over all t gives the denominator of $b'_j(o)$ and summing over those t where O equals o the numerator. Thus

$$b'_j(o) = \frac{\sum_{t\epsilon\{1,\ldots,T\} \text{ such that } o_t=o} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}.$$

Using the typical E-M Algorithm approach the Baum Welch Algorithm for a given observation O after initializing A and B repeats through the following steps

1. (**E step**) Set

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O)},$$
$$\eta_t(i,j) = \frac{\alpha_t(i)a_{ij}b_{t+1}(j)\beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_t(j)\beta_t(i)}.$$

2. (**M step**) Set

$$a'_{ij} = \frac{\sum_{t=1}^{T} \eta_t(i,j)}{\sum_{t=1}^{T}\sum_{j=1}^{N} \eta_t(i,j)},$$
$$b'_j(o) = \frac{\sum_{t\epsilon\{1,\ldots,T\} \text{ such that } o_t=o} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)},$$

until convergence.

6 Introduction to higher dimensional calculus

6.1 Higher dimensional functions

6.1.1 Basics

Let $A \subseteq \mathbb{R}^k$ and $B \subseteq \mathbb{R}^m$. We call a rule (or correspondence or relation) $\mathbf{f} : A \to B$, which associates with each $\mathbf{x} \in A$ a unique element $\mathbf{f}(\mathbf{x}) \in B$ a (real) function in one variable.

$A = D(\mathbf{f})$ is called the domain and $B = ran(\mathbf{f})$ the target space or range of the function $\mathbf{f}(\mathbf{x})$. We write $\mathbf{f}(\mathbf{x}) = \mathbf{y}$.

If we do not want to specify domain or target space we shorty write $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ for a function mapping from a domain $A \subseteq \mathbb{R}^k$ to a target space $B \subseteq \mathbb{R}^m$.

Remarks:

1. If the domain A is a subset of $\mathbb{R}^k$, then the input is a vector $\mathbf{x}$ with n components.

2. if the target space B is a subset of $\mathbb{R}^m$, then the output is a vector $\mathbf{y}$ with n components. That is the function itself has m components.

According to this observation a general function $\mathbf{f}$ from $\mathbb{R}^k$ to $\mathbb{R}^m$ has the shape

$$\mathbf{f}(\mathbf{x}) = \mathbf{y} \quad \Longleftrightarrow \quad \begin{array}{c} f_1(x_1, x_2, \ldots, x_k) = y_1, \\ f_2(x_1, x_2, \ldots, x_k) = y_2, \\ \vdots \\ f_m(x_1, x_2, \ldots, x_k) = y_m. \end{array}$$

6.1.2 Geometric representation of functions

Let $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ be a function in n variables mapping to the $\mathbb{R}^m$.

- If $k = 2$ and $m = 1$ the function can be sketched as a three dimensional figure – each point in the $x-y$ plane is assigned to its z– value $f(x, y,)$.

- Slices: For every $l \in 1, 2, \ldots, k$, fix all but the l–th input variables as

constants c_i and sketch the two dimensional slice

$$g(x) = f_i(c_0, c_1, \ldots, c_{l-1}, x_l, c_{l+1}, \ldots, c_k)$$

for every component of the function $\mathbf{f}$.
If $k = 2$ and $m = 1$ one fixes only one coordinate and varies the other, i.e. there are slices in x and y direction.

- Level curves/sets: Fix an output vector $\mathbf{b} \in \mathbb{R}^m$ and sketch (slices of) the level set

$$\mathbf{f}^{-1}(\mathbf{b}) = \{\mathbf{x} \in A \ : \ \mathbf{f}(\mathbf{x}) = \mathbf{b}\}.$$

  If $k = 2$ and $m = 1$ a level set for level l is the set of all $(x, y)^T$ such that $f(x, y) = l$, that is the set of all point isn the $x-y$ plane for which f takes height l.

- If $k = 1$ and m is arbitrary then $\mathbf{f}(t)$ assigns to every $t \in A$ a point of $\mathbb{R}^m$. This describes a curve $\mathbb{R}^m$, like $\mathbf{f}(t) = \mathbf{w} + t\mathbf{v}$ was a line.

6.1.3 Special kinds of functions

Linear functions

A function $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ is called linear if it satisfies the two properties

- $\mathbf{f}(c\mathbf{x}) = c\mathbf{f}(\mathbf{x})$, for all $c \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^k$,
- $\mathbf{f}(\mathbf{x} + \mathbf{y}) = \mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$.

Matrizes are linear functions since $A(c\mathbf{x}) = cA(\mathbf{x})$ and $A(\mathbf{x} + \mathbf{y}) = A(\mathbf{x}) + A(\mathbf{y})$. However, all linear functions have a matrix representation (w.r.t to a fixed basis): Let $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ be a linear function and $B = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k)$ and $C = (\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m)$ be two basis of the R^k and $\mathbb{R}^m$. Firstly, observe that $\mathbf{f}(\mathbf{b}_i)$ can be written as a unique linear combination of elements of C, i.e.

$$\mathbf{f}(\mathbf{b}_i) = a_{1i}\mathbf{c}_1 + \ldots a_{mi}\mathbf{c}_m,$$

or, shortly, $\mathbf{f}(\mathbf{b}_i) = C\mathbf{a}_i$. Let $\mathbf{x} = \lambda_1\mathbf{b}_1 + \ldots + \lambda_k\mathbf{b}_k = B\boldsymbol{\lambda}$ be the representation

of $\mathbf{x}$ w.r.t. the basis B, i.e. $\mathbf{x}$ has coordinates $\boldsymbol{\lambda}$. Then

$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \mathbf{f}(\lambda_1\mathbf{b}_1 + \ldots + \lambda_k\mathbf{b}_k)\\ &= \lambda_1\mathbf{f}(\mathbf{b}_1) + \ldots + \lambda_k\mathbf{f}(\mathbf{b}_k)\\ &= \lambda_1 C\mathbf{a}_1 + \ldots + \lambda_k C\mathbf{a}_k\\ &= CA\boldsymbol{\lambda},\end{aligned}$$

where A is the (uniquely determined) $k \times n$ matrix with columns $\mathbf{a}_i$. Thus if $\mathbf{x}$ has coordinates $\boldsymbol{\lambda}$ w.r.t. the basis B then $\mathbf{f}(\mathbf{x})$ has coordinates $A\boldsymbol{\lambda}$ w.r.t the basis C.

Remarks:

1. The matrix A depends on the chosen basis B and C. If one picks different basis B' and C' one has to d a change of coordinates to obtain the matrix representation of $\mathbf{f}$ w.r.t the new basis:

   $$A' = C'^{-1}CAB^{-1}B' \quad \text{as well as} \quad A = C^{-1}C'A'B'^{-1}B.$$

   In particular, if A is an $n \times n$ matrix and $B = C$ is the standard basis, and A' is the same mapping w.r.t. to an other basis P (i.e. $B' = C' = P$), then

   $$A = PA'P^{-1}.$$

   This is exactly the shape of the diagonalisation – if A has a basis P of Eigenvectors, then the linear function represented by A only stretches w.r.t. the basis P. I.e. A can be written as

   $$A = PDP^{-1}.$$

2. In particular linear $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}$ have the representation $f(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x}$ where $\mathbf{a}$ depends again on the Basis of $\mathbb{R}^k$

3. $ranA$ denotes the range of the function $\mathbf{f}(x) = A\mathbf{x}$.

Quadratic forms

A quadratic form Q on $\mathbb{R}^k$ is a function $Q : \mathbb{R}^k \to \mathbb{R}$ of the form

$$Q(x_1, \ldots, x_k) = \sum_{i,j=1}^{k} a_{ij}x_ix_j.$$

Each quadratic form has a matrix representation:

$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x},$$

where A is the $k \times k$ symmetric matrix with components b_{ij} such that $b_{ii} = a_{ii}$ and $b_{ij} = b_{ji} = \frac{a_{ij}}{2}$ where the a_{ii} are the coefficients from the definition of Q.

Remarks:

1. Conic section like ellipsoid or hyperbola occur as the level curves of quadratic forms.

2. The matrix corresponding to a quadratic form is always symmetric. Such matrices have only real Eigenvalues and it is possible to find a orthonormal basis of Eigenvectors. Consequently,

   $$A = PDP^T \quad \text{or} \quad D = P^T AP.$$

   Writing the quadratic form

   $$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x},$$

   refers to coordinates $\mathbf{x}$, typically w.r.t. the standard basis. Then $\mathbf{x}' = P^T\mathbf{x} = P^{-1}\mathbf{x}$ are the coordinates w.r.t the basis P. Using $(\mathbf{x}')^T = (P^T\mathbf{x})^T = \mathbf{x}^T P$ we obtain

   $$\begin{aligned} Q(\mathbf{x}) &= \mathbf{x}^T A\mathbf{x} \\ &= \mathbf{x}^T PP^T APP^T\mathbf{x} \\ &= (\mathbf{x}')^T D\mathbf{x}', \end{aligned}$$

   i.e.

   $$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} = \lambda_1(x_1')^2 + \lambda_2(x_2')^2 + \ldots + \lambda_n(x_n')^2. \tag{6}$$

   Thus, using a coordinate change to an orthonormal basis given by the Eigenvectors of A, every quadratic form can be written sum of multiples of squares, where the coefficients are the Eigenvalues of A. This fact is called the principal axis theorem.

Polynomials

A monomial is a function $f(x_1, \ldots, x_k) = cx_1^{a_1} \cdots x_k^{a_k}$. The degree of such a monomial is $a_1 + \ldots + a_k$.

A polynomial of degree d is a sum of monomials such that the highest degree of the monomials in the sum equals d.

Particular polynomials are affine functions, i.e. functions $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ with $\mathbf{f}(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$.

6.1.4 Continuous functions

Continuous function are functions where points can be approximated arbitrarily well. The concept of limit of functions allows to define this properly:

For a function $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ we write

$$\lim_{\mathbf{x} \to \mathbf{a}} \mathbf{f}(\mathbf{x}) = \mathbf{b}$$

if for every $\varepsilon > 0$ there exists a $\delta > 0$ such that for all $\mathbf{x} \in B_\delta(\mathbf{a})$ such that $\mathbf{x} \neq \mathbf{a}$ holds $\mathbf{f}(\mathbf{x}) \in B_\varepsilon(\mathbf{b})$. This means that $\mathbf{f}(\mathbf{x})$ is arbitrarily (i.e. ε–) close to $\mathbf{b}$ provided that $\mathbf{x}$ is sufficiently (i.e. δ–) close to $\mathbf{a}$ but not equal to $\mathbf{a}$.

Remark:

Above definition of $\lim_{\mathbf{x} \to \mathbf{a}} \mathbf{f}(\mathbf{x}) = \mathbf{b}$ is equivalent to $\lim_{n \to \infty} \mathbf{f}(\mathbf{x}_n) = \mathbf{b}$ for every sequence $\mathbf{x}_n$ sucht that $\lim_{n \to \infty} \mathbf{x}_n = \mathbf{a}$ and $\mathbf{x}_n \neq \mathbf{a}$.

A function is called continuous at $\mathbf{a}$ if

$$\lim_{\mathbf{x} \to \mathbf{a}} \mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{a}),$$

i.e. if $\mathbf{f}(\mathbf{x})$ approximates $\mathbf{f}(\mathbf{a})$ arbitrarily well if $\mathbf{x}$ is sufficiently close to $\mathbf{a}$ (but not equal to $\mathbf{a}$). In other words, one knows $\mathbf{f}(\mathbf{a})$ well if one knows $\mathbf{f}(\mathbf{x})$ for $\mathbf{x}$ close to $\mathbf{a}$.

$\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ is called continuous on A if it is continuous for every $\mathbf{a} \in A$.

1. Continuity behaves nicely w.r.t. operations of functions: If $\mathbf{f}$ and $\mathbf{g}$ are two continuous functions so are $\mathbf{f} \pm \mathbf{g}$, $\mathbf{f} \cdot \mathbf{g}$, $\mathbf{f}/\mathbf{g}$.

2. $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ is continuous if and only if every component f_i of $\mathbf{f} = (f_1, f_2, \ldots, f_m)$ is continuous as a real valued function.

6.1.5 Vocabulary of functions

For a function $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$ we define

1. the image of a set $C \subseteq A$ by

$$\mathbf{f}(C) = \{\mathbf{b} \in B \ : \ \mathbf{f}(\mathbf{x}) = \mathbf{b} \text{ for some } \mathbf{x} \in C\}.$$

2. the pre-image of a set $V \subseteq B$ by

$$\mathbf{f}^{-1}(V) = \{\mathbf{x} \in A \ : \ \mathbf{f}(\mathbf{x}) \in V\}.$$

We call a function $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$

1. one–to–one or injective if different starting points $\mathbf{x} \neq \mathbf{y}$ lead to different target values, i.e., $\mathbf{f}(\mathbf{x}) \neq \mathbf{f}(\mathbf{y})$. Equivalently, if $\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{y})$ implies $\mathbf{x} = \mathbf{y}$.

2. onto or surjective if every element of the target space is reached, i.e., if $\mathbf{f}(A) = B$.

3. bijective if $\mathbf{f}$ is surjective and injective, i.e., if $\mathbf{f}$ is every $\mathbf{b} \in B$ is reached under $\mathbf{f}$ from exactly one $\mathbf{x} \in A$.

For two functions $\mathbf{f} : A \to B$ and $g : B \to C$ we define analogous to real functions the composition of f and g, notation $\mathbf{g} \circ \mathbf{f}$ by

$$\mathbf{g} \circ \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{f}(\mathbf{x})).$$

and consequently the inverse function of $\mathbf{f}$, notation $\mathbf{f}^{-1}$, to be the function such that

$$\mathbf{f} \circ \mathbf{f}^{-1}(\mathbf{x}) = \mathbf{f}^{-1} \circ \mathbf{f}(\mathbf{x}) = \mathbf{x}.$$

That is, the composition puts one function after the other and function and inverse function cancel out if put one after the other.

Remarks:

1. Composition of linear maps and matrix multiplication belong to each other:
   Let l_1 and l_1 be two linear maps and A_1 and A_2 the matrices representing those maps. Then $A_1 A_2$ is the matrix representing $l_1 \circ l_2$.

2. If f and g are continuous so is $\mathbf{g} \circ \mathbf{f}$.

3. A function has an inverse function if and only if it is bijective.

4. The inverse function of a matrix interpreted as a linear function is the inverse matrix.

Finally observe that a sequence is function whose domain is $\mathbb{N}^+$.

6.2 Calculus of Several Variables

6.3 Definitions and first results

When defining differentiability in higher dimensions one has to be careful as we will discuss now – the one dimensional derivative can be seen as

- a linear function ($l(x) = f'(a)(x-a)$ for some point a) and
- as number which measures the relative change rate along the x axis.

Generalising these ideas one ends up with two concepts:

- the (total) differential of a function which is the linear approximation, and
- the partial derivatives (which are the relative change rates along the coordinates of the domain).

Whilst in $\mathbb{R}$ these concept naturally coincide a more careful approach is necessary for functions $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$.

Definition of the total derivative

Using limits we can define the differential of f at at point $\mathbf{x} \in X$ to be the linear approximation of the mapping f at $\mathbf{x}$. This gives the following definition:

Let $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$ and $\mathbf{a} \in intA$. If there exists a linear function $Df(\mathbf{a}) : \mathbb{R}^k \to \mathbb{R}^m$ such that the limit

$$\lim_{\mathbf{h} \to \mathbf{0}} \frac{\|\mathbf{f}(\mathbf{a}+\mathbf{h}) - \mathbf{f}(\mathbf{a}) - Df(\mathbf{a})\mathbf{h}\|}{\|\mathbf{h}\|} = 0$$

exists, the we call $\mathbf{f}$ differentiable at $\mathbf{a}$ and $Df(\mathbf{a})$ its (total) derivative.

Remarks:

1. The limit–definition of the derivative of $\mathbf{f}$ is equivalent to the existence of a function $\mathbf{r} : \mathbb{R}^k \to \mathbb{R}^m$ such that

$$\mathbf{f}(\mathbf{a}+\mathbf{h}) - \mathbf{f}(\mathbf{a}) = Df(\mathbf{a})\mathbf{h} + \mathbf{r}(\mathbf{h}) \quad \text{and} \quad \lim_{\mathbf{h}\to\mathbf{0}} \frac{\mathbf{r}(\mathbf{h})}{\|\mathbf{h}\|} = \mathbf{0}.$$

2. If the differential $Df(\mathbf{a})$ exists it is unique.

   Since: Assume otherwise, i.e. assume there are two linear functions $Df_1(\mathbf{a})$ and $Df_2(\mathbf{a})$ satisfying the definition of the differential. Then

$$\frac{(Df_1(\mathbf{a}) - Df_2(\mathbf{a}))\mathbf{h}}{\|\mathbf{h}\|} = \frac{\mathbf{r}_1(\mathbf{h}) - \mathbf{r}_2(\mathbf{h})}{\|\mathbf{h}\|} \to \mathbf{0}$$

   for all $\mathbf{h} \to \mathbf{0}$. Then

$$\frac{(Df_1(\mathbf{a}) - Df_2(\mathbf{a}))\mathbf{h}}{\|\mathbf{h}\|} = \frac{(Df_1(\mathbf{a}) - Df_2(\mathbf{a}))t\mathbf{h}}{\|t\mathbf{h}\|} \to \mathbf{0}$$

   if $t \to 0$ and thus $(Df_1(\mathbf{a}) - Df_2(\mathbf{a}))\mathbf{h} = \mathbf{0}$ (at least for all sufficiently small $\mathbf{h}$ such that $\mathbf{a}+\mathbf{h} \in intA$. But then $Df_1(\mathbf{a}) = Df_2(\mathbf{a})$.

3. If the differential $Df(\mathbf{x})$ exists at every $\mathbf{x} \in A \subseteq \mathbb{R}^k$ we call $\mathbf{f}$ differentiable.

4. If $m = 1$ then $Df(\mathbf{a})$ is a k–dimensional vector and $z = Df(\mathbf{a})\mathbf{x}$ is a hyperplane in $\mathbb{R}^{k+1}$ such that $\mathbf{f}(\mathbf{a}) + Df(\mathbf{a})\mathbf{x}$ approximates $\mathbf{f}(\mathbf{a}+\mathbf{x})$.

5. Differentiability implies continuity: To see this use

$$\mathbf{f}(\mathbf{a}+\mathbf{h}) - \mathbf{f}(\mathbf{a}) = Df(\mathbf{a})\mathbf{h} + \mathbf{r}(\mathbf{h}) \to 0 \text{ if } \mathbf{h} \to \mathbf{0},$$

   i.e., $\lim_{\mathbf{h}\to\mathbf{0}} \mathbf{f}(\mathbf{a}+\mathbf{h}) = \mathbf{f}(\mathbf{a})$.

6. We call the derivative of the first derivative Df the second derivative of f, notation $D^2 f$,....

Definition of the partial derivatives

Let $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$, i.e.,

$$\mathbf{f(x)} \quad = \quad \begin{matrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_m(x_1, x_2, \ldots, x_n) \end{matrix} .$$

Motivated by the differential quotient approach used in one dimensional functions we can define the partial derivatives component- and coordinate-wise at a point $\mathbf{a} \in intA$: For the partial derivative of f_i with respect to x_j at $\mathbf{a} = (a_1, \ldots, a_k)^T$, notation $\frac{\partial f_i}{\partial x_j}(\mathbf{a})$, fix all but the j–th coordinates at the given point a_l $(l = 1, 2, \ldots, k,\ l \neq j)$ and compute the differential quotient (if it exists), i.e., the relative change rate with respect to the j–th component at a_j. I.e., the partial derivative of f_i with respect to x_j is defined by

$$\begin{aligned} \frac{\partial f_i}{\partial x_j}(\mathbf{a}) &= \lim_{h \to 0} \frac{f_i(a_1, a_2, \ldots, a_j + h, \ldots, a_n) - f_i(a_1, a_2, \ldots, a_j, \ldots, a_n))}{h} \\ &= \lim_{x \to a_j} \frac{f_i(a_1, a_2, \ldots, x, \ldots, a_n) - f_i(a_1, a_2, \ldots, a_j, \ldots, a_n))}{x - a_j} \end{aligned}$$

if the limits exists. (Otherwise we say the partial derivative does not exist.)

Remarks:

1. Thus the partial derivative computes $\frac{\partial f_i}{\partial x_j}(\mathbf{a})$ equals the one dimensional derivative $g'(a_j)$ of the one dimensional function

   $$g(x) = f_i(a_1, a_2, \ldots, x, \ldots, a_n).$$

   That is, the partial derivatives are obtained by working with two dimensional sections of the original function $\mathbf{f}$.

2. Being in fact one dimensional differentials, the partial derivatives are computed using the same differential rules from one dimensional differentiation, i.e., to compute $\frac{\partial f_i}{\partial x_j}$ one takes the derivative w.r.t. the variable x_j while all other variables x_i for $i \neq j$ are treated like constants.

3. Usual notation for partial derivatives are

   $$\frac{\partial f_i}{\partial x_j}, \frac{\partial}{\partial x_j} f_i, D_{x_j} f_i, (f_i)_{x_j}, .$$

4. Partial derivatives are again functions $\frac{\partial f_i}{\partial x_j} : \mathbb{R}^k \to \mathbb{R}$. If all partial derivatives are continuous, we call $\mathbf{f}$ continuously differentiable (or C^1). Analogously to one dimensional calculus one defines also higher partial derivatives by iterating this definition. Notation:

$$\frac{\partial^2 f_i}{\partial x_j^2}, , \frac{\partial^2}{\partial x_j \partial x_k} f_i, D_{x_j,x_k} f_i, (f_i), (f_i)_{x_j,x_k}, \dots$$

5. All partial derivatives of a function $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$ are collected in the so called Jacobian matrix , i.e.

$$\mathbf{J} = \begin{pmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_k} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_k} \end{pmatrix}.$$

   Other notation for the Jacobian:

$$D\mathbf{f},\ \mathbf{J}_\mathbf{f},\ \nabla \mathbf{f},\ \frac{\partial(f_1, .., f_m)}{\partial(x_1, .., x_k)}.$$

6. If $f : \mathbb{R}^k \to \mathbb{R}$, the Jacobian is only a vector, namely

$$\begin{pmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_k} \end{pmatrix}$$

   This vector is called the gradient of f and denoted by ∇f.

   We write ∇f as a row–vector or as an (ordinary column) vector according to the context, however, this should not cause confusion.

Total differential vs. partial derivatives

Facts:

1. If $\mathbf{f}$ is differentiable then the Jacobian of $\mathbf{f}$, $\mathbf{J}_\mathbf{f}$, evaluated at an inner point $\mathbf{a}$ is the total derivative $Df(\mathbf{a})$.
   To see this, let A be the matrix representing the linear function $Df(\mathbf{a})$ (w.r.t. the Euclidean standard bases $\mathbf{e}_j$ of the $\mathbb{R}^k$ and $\mathbb{R}^m$), i.e.,

$$Df(\mathbf{a}) = A = \begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mk} \end{pmatrix}.$$

According to the definition of $Df(\mathbf{a})$ via

$$\mathbf{f}(\mathbf{a}+\mathbf{h}) - \mathbf{f}(\mathbf{a}) = Df(\mathbf{a})\mathbf{h} + \mathbf{r}(\mathbf{h}) \quad \text{and} \lim_{\mathbf{h}\to\mathbf{0}} \frac{\mathbf{r}(\mathbf{h})}{\|\mathbf{h}\|} = \mathbf{0}.$$

for the special choice $\mathbf{h} = h\mathbf{e}_j$ and taking the i–th component of $\mathbf{f}$ we obtain

$$f_i(\mathbf{a}+h\mathbf{e}_j) - f_i(\mathbf{a}) = a_{ij}h + \mathbf{r}(\mathbf{h})$$

and consequently $a_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{a})$.

2. However, it is possible that all partial derivatives of a function $\mathbf{f}$ exist at some point $\mathbf{a}$, even if $\mathbf{f}$ itself is not continuous. I.e.: The partial derivatives do not guarantee differentiability.
   See for instance the counter example

$$f(x,y) = \begin{cases} \frac{xy}{x^2+y^2} & \text{for}(x,y) \neq (0,0), \\ 0 & \text{for}(x,y) = (0,0). \end{cases}$$

3. If $\mathbf{f}$ is of class C^1 (at least in some in some neighborhood of $\mathbf{a}$), i.e., if all partial derivatives are continuous sufficiently close to $\mathbf{a}$, then $\mathbf{f}$ is differentiable at $\mathbf{a}$. (The converse is not true).
   To see this let $\mathbf{a}$ be the point where we want to show differentiability, i.e. we want to estimate

$$\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{a}) - Df(\mathbf{a})(\mathbf{x}-\mathbf{a}),$$

   where $\mathbf{x}$ in the neighbourhood of $\mathbf{a}$ where f is C^1. If $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ let, for $j = 0, 1, \ldots, k-1$,

$$\mathbf{x}_j = (x_1, x_2, \ldots, x_j, a_{j+1} \ldots, a_k),$$

   i.e. $\mathbf{x}_0 = \mathbf{a}$ and $\mathbf{x}_k = \mathbf{x}$. and

$$\mathbf{r}_j(t) = \mathbf{x}_{j-1} + t(\mathbf{x}_j - \mathbf{x}_{j-1}).$$

   The mean value theorem for one dimensional functions implies

$$\begin{aligned} f_i(\mathbf{x}_j) - f_i(\mathbf{x}_{j-1}) &= f_i(\mathbf{r}_j(1)) - f_i(\mathbf{r}_j(0)) \\ &= \frac{\partial f_i}{\partial x_j}(r_j(\xi_{ij}))(x_j - a_j) \end{aligned}$$

for some real ξ_{ij} in $[0,1]$. Then,

$$\begin{aligned}
f_i(\mathbf{x}) - f_i(\mathbf{a}) &= f_i(\mathbf{x}_k) - f_i(\mathbf{x}_0) \\
&= f_i(\mathbf{x}_k) - f_i(\mathbf{x}_{k-1}) + f_i(\mathbf{x}_{k-1}) - f_i(\mathbf{x}_{k-2}) + \ldots + f_i(\mathbf{x}_1) - f_i(\mathbf{x}_0) \\
&= \frac{\partial f_i}{\partial x_k}(r_k(\xi_{i,k}))(x_k - a_k) + \frac{\partial f_i}{\partial x_{k-1}}(r_{k-1}(\xi_{i,k-1}))(x_{k-1} - a_{k-1}) + \ldots \\
&\quad + \frac{\partial f_i}{\partial x_1}(r_1(\xi_{ik}))(x_1 - a_1) \\
&= \nabla f_i(\mathbf{r}(\boldsymbol{\xi}_i))(\mathbf{x} - \mathbf{a}).
\end{aligned}$$

Therefore

$$\begin{aligned}
\frac{|f_i(\mathbf{x}) - f_i(\mathbf{a}) - (Df(\mathbf{a})(\mathbf{x} - \mathbf{a})_i|}{\|\mathbf{x} - \mathbf{a}\|} &= \frac{|(\nabla f_i(\mathbf{r}(\boldsymbol{\xi}_i)) - \nabla f_i(\mathbf{a}))(\mathbf{x} - \mathbf{a})|}{\|\mathbf{x} - \mathbf{a}\|} \\
&\leq \frac{\|\nabla f_i(\mathbf{r}(\boldsymbol{\xi}_i)) - \nabla f_i(\mathbf{a})\| \, \|\mathbf{x} - \mathbf{a}\|}{\|\mathbf{x} - \mathbf{a}\|} \\
&= \|\nabla f_i(\mathbf{r}(\boldsymbol{\xi}_i)) - \nabla f_i(\mathbf{a})\|
\end{aligned}$$

which converges to 0, since $\boldsymbol{\xi}_i \to 0$ as $\mathbf{x} \to \mathbf{a}$ due to the continuity of the partial derivatives.
(The inequality is the Cauchy Schwarz inequality).

6.3.1 The chain rule

The chain rule is one of the most important tools in multivariable calculus - therefore we develop it slowly and stepwise.

Starting point is a curve $\mathbf{x} : \mathbb{R} \to \mathbb{R}^k$.

Fact:

The component wise derivative is denoted by $\mathbf{x}'(t)$, i.e.,

$$\mathbf{x}'(t) = \begin{pmatrix} x_1'(t) \\ x_2'(t) \\ \vdots \\ x_k'(t) \end{pmatrix}$$

evaluated at a point $t_0 \in \mathbb{R}$ the tangent to the curve at $\mathbf{x}(t_0)$.

A curve is called regular if each component x_i is continuously differentiable and $\mathbf{x}'(t) \neq \mathbf{0}$ for all t.

Let $f : \mathbb{R}^k \to \mathbb{R}$ be a real valued function. Then the real function

$$g(t) = f(\mathbf{x}(t))$$

is the function f along the curve $\mathbf{x}(t)$. If $g(t)$ is differentiable, $g'(t)$ computes the change rate of f in the direction of $\mathbf{x}(t)$. The chain rule tells us how to derive this derivative:

$$\boxed{g'(t) = \nabla f(\mathbf{x}(t)) \cdot \mathbf{x}'(t)}$$

or, more detailed,

$$\boxed{g'(t) = \frac{\partial f}{\partial x_1}(\mathbf{x}(t))x_1'(t) + \frac{\partial f}{\partial x_2}(\mathbf{x}(t))x_2'(t) + \ldots + \frac{\partial f}{\partial x_k}(\mathbf{x}(t))x_k'(t)}$$

We will give a concise proof of the general chain rule at the end of this section, however the following lines give a nontechnical informal idea:
We use the approximations of f and $\mathbf{x}$ given by the derivatives, i.e.,

$$\begin{aligned} f(\mathbf{x} + \mathbf{h}) &\approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{h}, \\ \mathbf{x}(t + h) &\approx f(\mathbf{x}) + \mathbf{x}'(t)h. \end{aligned}$$

In order to keep notation simple we use $\approx$ for "approximately". Then

$$\begin{aligned} g'(t) = (f(\mathbf{x}(t)))' &= \lim_{h \to 0} \frac{f(\mathbf{x}(t+h)) - f(\mathbf{x}(t))}{h} \\ &\approx \lim_{h \to 0} \frac{f(\mathbf{x}(t)) + \mathbf{x}'(t)h) - f(\mathbf{x}(t))}{h} \\ &\approx \lim_{h \to 0} \frac{f(\mathbf{x}(t)) + \nabla f(\mathbf{x}(t)) \cdot \mathbf{x}'(t)h - f(\mathbf{x}(t))}{h} \\ &= \nabla f(\mathbf{x}(t)) \cdot \mathbf{x}'(t). \end{aligned}$$

Remark:

The computation only holds if f is of class C^1 since otherwise it is not guaranteed that $\mathbf{x}(t+h) \approx f(\mathbf{x}) + \mathbf{x}'(t)h$ for any (small) $\mathbf{h} \in \mathbb{R}^k$. The following is a good example illustrating this: Let, as before,

$$f(x, y) = \begin{cases} \frac{xy}{x^2+y^2} & \text{for } (x, y) \neq (0, 0), \\ 0 & \text{for } (x, y) = (0, 0). \end{cases}$$

and consider the curve $\mathbf{x}(t) = \binom{t}{t^2}$. When computed by directly plugging in with the differential quotient one obtains $g'(0) = 1$ whilst the chain rule would give $g'(0) = 0$.
One therefore generally asks in the chain rule for either differentiability or that all involved functions are of C^1.

This basic version $g'(t) = \nabla f(\mathbf{x}(t)) \cdot \mathbf{x}'(t)$ of the chain rule immediately generalises to higher dimensional functions:

Let $f : \mathbb{R}^k \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^l \to \mathbb{R}^k$ and

$$h(\mathbf{x}) = f \circ \mathbf{g}(x_1, x_2, \ldots, x_l); \qquad \mathbb{R}^l \to \mathbb{R}$$

The first version immediately yields

$$\boxed{\frac{\partial h}{\partial x_i}(\mathbf{x}) = \frac{\partial f}{\partial x_1}(\mathbf{g}(\mathbf{x}))\frac{\partial g_1}{\partial x_i}(\mathbf{x}) + \frac{\partial f}{\partial x_2}(\mathbf{g}(\mathbf{x}))\frac{\partial g_2}{\partial x_i}(\mathbf{x}) + \ldots + \frac{\partial f}{\partial x_k}(\mathbf{g}(\mathbf{x}))\frac{\partial g_k}{\partial x_i}(\mathbf{x})}$$

Finally, if $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^m$, $\mathbf{g} : \mathbb{R}^l \to \mathbb{R}^k$ and

$$\mathbf{h}(\mathbf{x}) = \mathbf{f} \circ \mathbf{g}(x_1, x_2, \ldots, x_l); \qquad \mathbb{R}^l \to \mathbb{R}^m$$

then the discussion so far leads to

$$\boxed{\frac{\partial h_i}{\partial x_j}(\mathbf{x}) = \frac{\partial f_i}{\partial x_1}(\mathbf{g}(\mathbf{x}))\frac{\partial g_1}{\partial x_j}(\mathbf{x}) + \frac{\partial f_i}{\partial x_2}(\mathbf{g}(\mathbf{x}))\frac{\partial g_2}{\partial x_j}(\mathbf{x}) + \ldots + \frac{\partial f_i}{\partial x_k}(\mathbf{g}(\mathbf{x}))\frac{\partial g_k}{\partial x_j}(\mathbf{x})}$$

Therefore

$$\boxed{\mathbf{J}_h(\mathbf{x}) = \mathbf{J}_f(\mathbf{g}(\mathbf{x}))\mathbf{J}_g(\mathbf{x}))}$$

After this stepwise discussion here also a concise proof based on the definition of differentiability: Let $\mathbf{f} : A \subseteq \mathbb{R}^k \to B \subseteq \mathbb{R}^m$ and $\mathbf{g} : B \subseteq \mathbb{R}^l \to C \subseteq \mathbb{R}^k$. Assume $\mathbf{g}$ is differentiable at $\mathbf{x}$ and $\mathbf{f}$ at $\mathbf{g}(\mathbf{x})$. Then $\mathbf{l} = \mathbf{f} \circ \mathbf{g} : A \to C$ is differentiable and

$$Dl(\mathbf{x}) = Df(\mathbf{g}(\mathbf{x})) \cdot Df(\mathbf{x}).$$

Set $\mathbf{y} = \mathbf{g}(\mathbf{x})$, $A = Dg(\mathbf{x})$ and $B = Df(\mathbf{y})$. Then the differentiability of $\mathbf{f}$ and $\mathbf{g}$ can be rewritten as

$$\begin{aligned}
\mathbf{g}(\mathbf{x}+\mathbf{h}) &= \mathbf{g}(\mathbf{x}) + A\mathbf{h} + \mathbf{r}(\mathbf{h}) \quad \text{and} \lim_{\mathbf{h}\to\mathbf{0}} \frac{\mathbf{r}(\mathbf{h})}{\|\mathbf{h}\|} = \mathbf{0}, \\
\mathbf{f}(\mathbf{y}+\mathbf{h}') &= \mathbf{f}(\mathbf{y}) + B\mathbf{h}' + \mathbf{s}(\mathbf{h}') \quad \text{and} \lim_{\mathbf{h}'\to\mathbf{0}} \frac{\mathbf{s}(\mathbf{h}')}{\|\mathbf{h}'\|} = \mathbf{0}.
\end{aligned}$$

We need to show that

$$\mathbf{l}(\mathbf{x}+\mathbf{h}) = \mathbf{l}(\mathbf{x}) + BA\mathbf{h} + \mathbf{t}(\mathbf{h}) \quad \text{and} \quad \lim_{\mathbf{h}\to\mathbf{0}} \frac{\mathbf{t}(\mathbf{h})}{\|\mathbf{h}\|} = \mathbf{0}.$$

Note that

$$\begin{aligned}\mathbf{l}(\mathbf{x}+\mathbf{h}) &= \mathbf{f}(\mathbf{g}(\mathbf{x}+\mathbf{h})) = \mathbf{f}(\mathbf{y} + A\mathbf{h} + \mathbf{r}(\mathbf{h})) \\ &= \mathbf{f}(\mathbf{y}) + B(A\mathbf{h} + \mathbf{r}(\mathbf{h}))) + \mathbf{s}(A\mathbf{h} + \mathbf{r}(\mathbf{h})),\end{aligned}$$

i.e., $\mathbf{t}(\mathbf{h}) = B\mathbf{r}(\mathbf{h}))) + \mathbf{s}(A\mathbf{h} + \mathbf{r}(\mathbf{h}))$. Clearly

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{B\mathbf{r}(\mathbf{h})}{\|\mathbf{h}\|} = \mathbf{0}.$$

Let σ be a function such that $\mathbf{s}(\mathbf{h}) = \|\mathbf{h}\|\sigma(\mathbf{h})$. Then $\sigma(\mathbf{h}) \to \mathbf{0}$ if $\mathbf{h} \to \mathbf{0}$. $A\mathbf{h} + \mathbf{r}(\mathbf{h})) \to \mathbf{0}$ as $\mathbf{h} \to \mathbf{0}$. Thus $\sigma(A\mathbf{h} + \mathbf{r}(\mathbf{h})) \to \mathbf{0}$ as $\mathbf{h} \to \mathbf{0}$. Moreover,

$$\frac{\|A\mathbf{h} + \mathbf{r}(\mathbf{h}))\|}{\|\mathbf{h}\|} \le \left\| A\frac{\mathbf{h}}{\|\mathbf{h}\|} \right\| + \frac{\|\mathbf{r}(\mathbf{h}))\|}{\|\mathbf{h}\|}$$

which is bounded, if $\mathbf{h} \to \mathbf{0}$. Hence

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{s(A\mathbf{h} + r(\mathbf{h}))}{\|\mathbf{h}\|} = \lim_{\mathbf{h}\to\mathbf{0}} \frac{\|A\mathbf{h} + r(\mathbf{h}))\|}{\|\mathbf{h}\|}\sigma(A\mathbf{h} + r(\mathbf{h})) = \mathbf{0}.$$

6.3.2 The directional derivative and gradients

Let $\mathbf{v} \in \mathbb{R}^n$ be a direction, i.e. $\|\mathbf{v}\| = 1$, and $\mathbf{x}(t) = \mathbf{x}^* + t\mathbf{v}$ be the line starting form a fixed point $\mathbf{x}^* \in \mathbb{R}^n$ in direction $\mathbf{v}$. Let $f : \mathbb{R}^n \to \mathbb{R}$. Then the derivative of f at $\mathbf{x}^*$ in direction $\mathbf{v}$, notation $\frac{\partial f}{\partial \mathbf{v}}(\mathbf{x}^*)$, is the change rate of f along the curve $\mathbf{x}(t)$ at $t = 0$, i.e.,

$$\frac{\partial f}{\partial \mathbf{v}}(\mathbf{x}^*) = f(\mathbf{x}(t))'(0),$$

which is, by the chain rule,

$$\boxed{\frac{\partial f}{\partial \mathbf{v}}(\mathbf{x}^*) = \nabla f(\mathbf{x}^*) \cdot \mathbf{v}}$$

Facts:

1. Since the inner product $\nabla f(\mathbf{x}^*) \cdot \mathbf{v}$ is maximal if $\mathbf{v} = \nabla f(\mathbf{x}^*)$, the last equation immediately gives an important interpretation of the gradient of f:
   Given that $\nabla f(\mathbf{x}) \neq \mathbf{0}$, the gradient $\nabla f(\mathbf{x})$ points at $\mathbf{x}$ to the direction where f increases most rapidly.

2. If $f : \mathbb{R}^n \to \mathbb{R}$ has continuous partial derivatives we know that
   $$f(\mathbf{x}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a})(\mathbf{x} - \mathbf{a}).$$
   $f(\mathbf{a}) + \nabla f(\mathbf{a})(\mathbf{x} - \mathbf{a})$ is a hyperplane, called the tangent plane of f at $\mathbf{a}$. Observe that the normal vector to this plane is given by
   $$\begin{bmatrix} -1 \\ \nabla f(\mathbf{a}) \end{bmatrix}$$

3. Let $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{x}(t) : \mathbb{R} \to \mathbb{R}^n$ be a curve such that $f(\mathbf{x}(t)) = c$, i.e. along $\mathbf{x}(t)$ the function f takes constant "height" c. Then, by the chain rule
   $$0 = f'(\mathbf{x}(t)) = \nabla f(\mathbf{x}(t)) \cdot \mathbf{x}'(t).$$
   $\mathbf{x}'(t)$ is the tangent of $\mathbf{x}(t)$, therefore $\nabla f(\mathbf{x}(t))$ is perpendicular to the curve. Thus:
   $\nabla f(\mathbf{x}(t))$ is perpendicular to a level set $\mathbf{x}(t)$.

6.3.3 Higher order derivatives

In section 6.3 we introduced higher order derivatives recursively. Second order partial derivatives $\frac{\partial^2 f_i}{\partial x_j \partial x_k}$ are partial derivatives of the (first order) partial derivatives $\frac{\partial f_i}{\partial x_j}$, third order partial derivatives $\frac{\partial^3 f_i}{\partial x_j \partial x_k \partial x_l}$ are partial derivatives of the (second order) partial derivatives $\frac{\partial^2 f_i}{\partial x_j \partial x_k}$, ...

We call a function $\mathbf{f}$ of class C^p, $p \in \mathbb{N}$, if all possible partial derivatives of $\mathbf{f}$ of order p exist and are continuous.

Fact: If $f(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$ is of class C^2 then the cross derivatives do not depend on the order of differentiation, i.e.
$$\frac{\partial^2 f}{\partial x_j \partial x_k} = \frac{\partial^2 f}{\partial x_k \partial x_j}.$$

An important object is the so called Hessian matrix H (or H_f or $D^2 f$) which contains all partial second derivatives of a function $f(x_1, \dots, x_n) : \mathbb{R}^n \to \mathbb{R}$, i.e.

$$H_f = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$

6.3.4 Multivariable Taylor theorem

Using the multi-index notation

$$|\alpha| = \alpha_1 + \cdots + \alpha_n, \quad \alpha! = \alpha_1! \cdots \alpha_n!, \quad \boldsymbol{x}^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

for $\alpha \in \mathbb{N}$ and $x \in \mathbb{R}$.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^{k+1} function and $\mathbf{a} \in \mathbb{R}^n$. Then there exists a function $h_\alpha : \mathbb{R}^n \to \mathbb{R}$ such that

$$f(\boldsymbol{x}) = \sum_{|\alpha| \le k} \frac{D^\alpha f(\boldsymbol{a})}{\alpha!} (\boldsymbol{x} - \boldsymbol{a})^\alpha + \underbrace{\sum_{|\alpha| = k+1} \frac{D^\alpha f(\mathbf{c})}{\alpha!} (\boldsymbol{x} - \boldsymbol{a})^\alpha}_{= R_{k+1}(\mathbf{a}, \mathbf{x})}.$$

where $\mathbf{c}$ is a suitable element on the segment $[\mathbf{x}, \mathbf{a}]$ fixes the $k+1$–st error term $R_{k+1}(\mathbf{a}, \mathbf{x})$.

In particular we will use the second order multivariable Taylor formula for a C^3 function $f : \mathbb{R}^n \to \mathbb{R}$:

$$f(\mathbf{x}) = f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) + \frac{1}{2} (\mathbf{x} - \mathbf{a})^t H_f(\mathbf{a}) (\mathbf{x} - \mathbf{a}) + R_3(\mathbf{a}, \mathbf{x}), \qquad (7)$$

with the error term $R_3(\mathbf{a}, \mathbf{x}) = \frac{1}{3!} D^3 f(\mathbf{c})(\mathbf{x} - \mathbf{a})$ where $\mathbf{c}$ is a suitable element on the segment $[\mathbf{x}, \mathbf{a}]$.

Remark: The multivariable Taylor theorem follows directly from its one dimensional counterpart. To approximate f at a given point $\mathbf{x}$ starting from $\mathbf{a}$ take the line $\mathbf{x}(t) = \mathbf{a} + t(\mathbf{x} - \mathbf{a})$ for $t \in [0, 1]$ and compute the one dimensional Taylor polynomial of $g(t) = f(\mathbf{x}(t))$ with the chain rule. This gives the claimed formula.

6.4 Unconstrained Optimization

6.4.1 Definitions

Let $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$.

1. A point $\mathbf{x}^* \in U$ is called a max (or maximizer, maximum) of F on U if $F(\mathbf{x}^*) \geq F(\mathbf{x})$ for all $\mathbf{x} \in U$.

2. A point $\mathbf{x}^* \in U$ is called a strict max (or maximizer, maximum) of F on U if $F(\mathbf{x}^*) > F(\mathbf{x})$ for all $\mathbf{x} \in U$.

3. A point $\mathbf{x}^* \in U$ is called a local (or relative) max (or maximizer, maximum) of F on U if $F(\mathbf{x}^*) \geq F(\mathbf{x})$ for all $\mathbf{x} \in B_\varepsilon(\mathbf{x}^*) \cap U$ for some $\varepsilon > 0$.

4. A point $\mathbf{x}^* \in U$ is called a strict local (or relative) max (or maximizer, maximum) of F on U if $F(\mathbf{x}^*) > F(\mathbf{x})$ for all $\mathbf{x} \in B_\varepsilon(\mathbf{x}^*) \cap U$ for some $\varepsilon > 0$.

To emphasise that a point $\mathbf{x}^*$ is a max on whole U one also uses the words global max or absolute max.

Analogous definitions can be made min (or minimizer, minimum) by replacing the inequality signs to $<$ or $\leq$.

A point $\mathbf{x}^*$ which is either a min or a max will be call extrema (or extreme point).

6.5 Definiteness

In section 6.1.3 we introduced quadratic forms as quadratic polynomials $Q : \mathbb{R}^n \to R$ which have the special shape

$$Q(\mathbf{x}) = \mathbf{x}^T A \mathbf{x},$$

where A is a symmetric $n \times n$ matrix.

The following concept will be important for the next sections. Since symmetric $n \times n$ matrices and quadratic forms are directly related the following definitions hold similarly for quadratic forms Q as well as symmetric $n \times n$ matrices A.

We call the quadratic form Q/the symmetric $n \times n$ matrix A

- positive definite if
$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} > 0 \qquad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$
- positive semidefinite if
$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} \geq 0 \qquad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$
- negative definite if
$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} < 0 \qquad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$
- negative semidefinite if
$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} \leq 0 \qquad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$
- indefinite if there exist $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that
$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} > 0 \quad \text{and} \quad Q(\mathbf{x}) = \mathbf{y}^T A\mathbf{y} < 0.$$

Using the principal axis theorem, 6, one can immediately see the definiteness of $Q/\ A$ from its Eigenvalues: Q/A is

- positive definite if all Eigenvalues of A are positive,
- negative definite if all Eigenvalues of A are negative,
- positive semidefinite if all Eigenvalues of A are non-negative,
- negative semidefinite if all Eigenvalues of A are non-positive,
- indefinite if some Eigenvalues of A are positive whilst others are negative.

To verify definiteness one can also use so called leading principal minors of the $n \times n$ matrix A.

- Any $k \times k$ sub-matrix formed from A by deleting the same $n - k$ rows and columns from A is called a principal sub-matrix of k–th order of A.
- The determinant of a principal sub-matrix of k–th order of A is called a k–th order principal minor of A.
  Remark: In the Laplace expansion of Determinants one applies first order principal minors.
- The $k \times k$ sub-matrix formed from A by deleting the last $n - k$ rows and columns from A is called the k–th order leading principal sub-matrix of A.
- The determinant of a k–th order leading principal sub-matrix of A is called the k–th order leading principal minor of A.

Thus, if

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix},$$

the leading principle minors are

$$Det(a_{11}),\ Det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},\ Det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \ldots,\ Det(A).$$

Facts: Let A be a symmetric $n \times n$ matrix.

1. A is positive definite if all of its n leading principal minors are > 0.
2. A is negative definite if the k order leading principal minors have the same sign as $(-1)^k$, i.e. the leading principal minors have alternating sign, starting with $Det(a_{11}) < 0$.

Whilst definiteness can be verified by only evaluating the leafing principle minors, semi-definiteness requires to check every principle minor of A. One has

1. A is positive semidefinite if all of its n principal minors are ≥ 0.

2. A is negative semidefinite if all of its kth order principal minors are ≥ 0 if k is even and ≤ 0 if k is odd.

Remarks: If A is symmetric, definiteness is checked immediately – the principle minors are simply the product of the diagonal elements.

6.5.1 First order conditions

The idea from one dimensional functions carries over to calculus in higher dimensions – if a point $\mathbf{x}^*$ is a min or max then the tangent hyperplane must be parallel to the base plane $z = 0$, i.e. all partial derivatives need to be 0:

Fact: Let $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$ be a C^1 function. If $\mathbf{x}^*$ is a min or a max in the interior of U then

$$\frac{\partial F}{\partial x_i}(\mathbf{x}^*) = 0 \qquad \text{for all } i = 1, \ldots, n.$$

Points such that $\frac{\partial F}{\partial x_i}(\mathbf{x}^*) = 0$ for all $i = 1, \ldots, n$, are called critical points or stationary points.

Being a critical point is only a necessary condition for being an extrema, i.e., not all critical points are indeed extreme values. To obtain sufficient conditions, second order conditions are helpful.

6.5.2 Second order conditions

Suppose $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$ is a C^2 function. Let $\mathbf{x}^*$ be a critical point.

The main idea for understanding the second order condition is the concept of Taylor polynomials as discussed in 6.3.4– recall

Suppose that $F : U \to \mathbb{R}$ is three times continuously differentiable, then the second order Taylor approximation

$$f(\mathbf{x}) = f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) + \frac{1}{2}(\mathbf{x} - \mathbf{a})^t H_f(\mathbf{a})(\mathbf{x} - \mathbf{a}) + R_3(\mathbf{a}, \mathbf{x}),$$

with the error term $R_3(\mathbf{a},\mathbf{x}) = \frac{1}{3!}D^3 f(\mathbf{c})(\mathbf{x}-\mathbf{a})$, where $\mathbf{c}$ is a suitable element on the segment $[\mathbf{x},\mathbf{a}]$, holds. Also recall that $H_F(\mathbf{x})$ denotes the Hessian of F at $\mathbf{x}$), i.e.,

$$H_F(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial^2 F}{\partial x_1^2}(\mathbf{x}) & \dfrac{\partial^2 F}{\partial x_1\,\partial x_2}(\mathbf{x}) & \cdots & \dfrac{\partial^2 F}{\partial x_1\,\partial x_n}(\mathbf{x}) \\ \dfrac{\partial^2 F}{\partial x_2\,\partial x_1}(\mathbf{x}) & \dfrac{\partial^2 F}{\partial x_2^2}(\mathbf{x}) & \cdots & \dfrac{\partial^2 F}{\partial x_2\,\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 F}{\partial x_n\,\partial x_1}(\mathbf{x}) & \dfrac{\partial^2 F}{\partial x_n\,\partial x_2}(\mathbf{x}) & \cdots & \dfrac{\partial^2 F}{\partial x_n^2}(\mathbf{x}) \end{pmatrix},$$

Remark: Again, the tangent plane to F at a point $\mathbf{x}$ is given by the equation $z = F(\mathbf{x}) + \nabla F(\mathbf{x}) \cdot (\mathbf{y}-\mathbf{x})$.

So applying the Taylor approximation for a $\mathbf{y}$ close to a critical point $\mathbf{x}^*$ yields

$$F(\mathbf{y}) = F(\mathbf{x}^*) + \underbrace{\nabla F(\mathbf{x}^*)\cdot(\mathbf{y}-\mathbf{x}^*)}_{=0 \text{ on critical point } \mathbf{x}^*} + \frac{1}{2}(\mathbf{y}-\mathbf{x}^*)^t H(\mathbf{x}^*)(\mathbf{y}-\mathbf{x}^*) + R(\mathbf{x}^*,\mathbf{y})$$

one immediately obtains

1. If $H(\mathbf{x}^*)$ is positive definite, so $(\mathbf{y}-\mathbf{x}^*)^t H(\mathbf{x}^*)(\mathbf{y}-\mathbf{x}^*) > 0$, then $F(\mathbf{y}) > F(\mathbf{x}^*)$ for any $\mathbf{y}$ close to $\mathbf{x}^*$, meaning that $\mathbf{x}^*$ is a strict local minimum.

2. If $H(\mathbf{x}^*)$ is negative definite, so $(\mathbf{y}-\mathbf{x}^*)^t H(\mathbf{x}^*)(\mathbf{y}-\mathbf{x}^*) < 0$, then $F(\mathbf{y}) < F(\mathbf{x}^*)$ for any $\mathbf{y}$ close to $\mathbf{x}^*$, meaning that $\mathbf{x}^*$ is a strict local maximum.

3. If $H(\mathbf{x}^*)$ is indefinite, so $(\mathbf{y}-\mathbf{x}^*)^t H(\mathbf{x}^*)(\mathbf{y}-\mathbf{x}^*)$ is both >0 and <0, then $F(\mathbf{x}^*)$ is neither a min nor a max. for any $\mathbf{y}$ close to $\mathbf{x}^*$, meaning that $\mathbf{x}^*$ is a local minimum.

Remarks:

1. If $H(\mathbf{x}^*)$ is only semidefinite, no conclusion is possible.

2. If $H(\mathbf{x}^*)$ is indefinite, $\mathbf{x}^*$ is called a saddle point of F.

3. To check definiteness of $H(\mathbf{x}^*)$ one uses the criterion introduced in section 6.5, i.e. one checks the signs of the leading principal minors of

$H(\mathbf{x}^*)$, i.e.,

$$\frac{\partial^2 F}{\partial x_1^2}(\mathbf{x}^*),$$

$$Det\left(\begin{pmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} \end{pmatrix}(\mathbf{x}^*)\right),$$

$$Det\left(\begin{pmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \frac{\partial^2 F}{\partial x_1 \partial x_3} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \frac{\partial^2 F}{\partial x_2 \partial x_3} \\ \frac{\partial^2 F}{\partial x_3 \partial x_1} & \frac{\partial^2 F}{\partial x_3 \partial x_2} & \frac{\partial^2 F}{\partial x_3^2} \end{pmatrix}(\mathbf{x}^*)\right)$$

$$\vdots$$

$$\vdots$$

and

(a) if all signs are positive, then $\mathbf{x}^*$ is a min,

(b) if the signs are alternating, starting with a negative sign, then $\mathbf{x}^*$ is a max.

Remark: The second order necessary condition are slightly weaker – if an interior point $\mathbf{x}^*$ of U is

1. a max of F on U then $\nabla F(\mathbf{x}^*) = 0$ and $H_F(\mathbf{x}^*)$ is negative semidefinite,

2. a min of F on U then $\nabla F(\mathbf{x}^*) = 0$ and $H_F(\mathbf{x}^*)$ is positive semidefinite.

6.5.3 Global minima and maxima

The first and second order conditions discussed so far can be used to find local extrema. To find global extrema one must verify properties of the function globally – the desired concept property is convexity/concavity.

Let us start with definitions:

1. A set $C \subset \mathbb{R}^n$ is called convex if with any two points $\mathbf{x}$ and $\mathbf{y} \in C$ the

whole line segment connecting $\mathbf{x}$ and $\mathbf{y}$ is in C, i.e.,

$$\mathbf{x}, \mathbf{y} \in C \implies \underbrace{t\mathbf{x} + (1-t)\mathbf{y}}_{(*)} \in C \text{ for all } t \in [0,1]$$

Here, for $t \in [0,1]$, the expression $(*)$ is called convex combination of $\mathbf{x}$ and $\mathbf{y}$.

2. Let U be a convex set. A function $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$ is called convex, if the line segment connecting any two points on the graph of F lies above the graph of F, i.e. if for all $\mathbf{x}, \mathbf{y} \in U$ and all $t \in [0,1]$

$$F(t\mathbf{x} + (1-t)\mathbf{y}) \leq tF(\mathbf{x}) + (1-t)F(\mathbf{y}).$$

F is therefore convex, if the epigraph of F, i.e. the set of points above the graph of F, is a convex set.
If the $\leq$ can be replaced by a $<$, F is called strictly convex.

3. Let U be a convex set. A function $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$ is called concave, if the line segment connecting any two points on the graph of F lies below the graph of F, i.e. if for all $\mathbf{x}, \mathbf{y} \in U$ and all $t \in [0,1]$

$$F(t\mathbf{x} + (1-t)\mathbf{y}) \geq tF(\mathbf{x}) + (1-t)F(\mathbf{y}).$$

F is therefore concave, if $-F$ is convex.
If the $\leq$ can be replaced by a $<$, F is called strictly concave.

Facts:
Let $F : U \subseteq \mathbb{R}^n \to \mathbb{R}$ be a C^2 function defined on an open, convex set U.

1. The following are equivalent:
   (a) F is a concave function on U.
   (b) $F(\mathbf{y}) - F(\mathbf{y}) \leq DF(\mathbf{x})(\mathbf{y} - \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in U$.
   (c) $D^2F(\mathbf{x})$ is negative semidefinite for all $\mathbf{x} \in U$.
2. The following are equivalent:
   (a) F is a convex function on U.
   (b) $F(\mathbf{y}) - F(\mathbf{y}) \geq DF(\mathbf{x})(\mathbf{y} - \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in U$.

(c) $D^2F(\mathbf{x})$ is positive semidefinite for all $\mathbf{x} \in U$.

3. If F is concave on U and $DF(\mathbf{x}^*) = 0$ for some $\mathbf{x}^* \in U$ then $\mathbf{x}^*$ is a global max for F on U.

4. If F is convex on U and $DF(\mathbf{x}^*) = 0$ for some $\mathbf{x}^* \in U$ then $\mathbf{x}^*$ is a global min for F on U.

6.5.4 Two examples

1. Let $f : 4x^2 - 4xy + 2y^2$. Then

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 8x - 4y \\ -4x + 4y \end{pmatrix}.$$

Therefore the stationary point is found by

$$\begin{aligned} 8x - 4y &= 0, \\ 4x - 4y &= 0, \end{aligned}$$

i.e., $(x, y) = (0, 0)$ is the stationary point. Since the Hessian

$$H_f(x, y) = \begin{pmatrix} 8 & -4 \\ -4 & 4 \end{pmatrix}$$

has determinant of $48 > 0$ and $f_{xx}(0, 0) = 8 > 0$ we have a minimum at $(0, 0)$.

2. Let $f : x^2y - 2xy^2 + 3xy + 4$. Then

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2xy - 2y^2 + 3y \\ x^2 - 4xy + 3x \end{pmatrix},$$

and the stationary points are found by

$$\begin{aligned} 2xy - 2y^2 + 3y &= 0, \\ x^2 - 4xy + 3x &= 0. \end{aligned}$$

This gives the critical points $P_1 = (0, 0)$, $P_2 = (-3, 0)$, $P_3 = (0, 3/2)$ and $P_4 = (-1, 1/2)$. The second order conditions show that all of these points are saddle points except D which turns out to be a minimum.

6.6 Intermezzo: Least squares again

6.6.1 The problem

Given is a set of n points $(x_i, y_i) \in \mathbb{R}^2$, $i = 1, \ldots, n$. The aim is to find a line $y = mx + b$ which approximates these points optimally. It turns out, measuring optimal approximation by so called least squares is a very fruitful approach:

For any line $y = mx + b$ the difference of the approximation and the real value of the i th point (x_i, y_i) can be measured by

$$y_i - (mx_i + b).$$

This is just the difference in y direction between real value y_i and approximation $mx_i + b$. To get rid of cancellation effects and still have some nice computational properties (like differentiability) one squares these differences and says the line $y = mx + b$ optimally approaches the given set of points if the sum of square differences, so

$$\sum_{i=1}^{n} (y_i - (mx_i + b)^2,$$

is minimal.

Remark:

This idea directly generalises to hyperplanes in the $\mathbb{R}^k$ approximating a point cloud of n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$.

6.6.2 Calculus approach

Set $f(m, k) = \sum_{i=1}^{n} (y_i - (mx_i + b)^2$. Obviously f is defined on $\mathbb{R}^2$, so any extreme value is in the interior and must satisfy the first order conditions

$$\frac{\partial f}{\partial m} = \sum_{i=1}^{n} 2(y_i - (mx_i + b))x_i = 0,$$
$$\frac{\partial f}{\partial b} = \sum_{i=1}^{n} 2(y_i - (mx_i + b) = 0.$$

These first order conditions simplify to

$$\begin{aligned} m\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i y_i, \\ m\sum_{i=1}^{n} x_i + bn &= \sum_{i=1}^{n} y_i \end{aligned}$$

which is a system of two linear equations in unknowns m and b. Cramer's rule gives

$$\begin{aligned} m &= \frac{Det\begin{pmatrix} \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} y_i & n \end{pmatrix}}{Det\begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{pmatrix}} = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}, \\ b &= \frac{Det\begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i \end{pmatrix}}{Det\begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{pmatrix}} = \frac{\sum_{i=1}^{n} x_i^2 y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} x_i y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}. \end{aligned}$$

Remark: Since

$$\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2, \tag{8}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the usual mean value, m and b exist if and only if the values x_i are non-constant for all i.

The Hessian H of $f(k, m)$ is

$$\begin{pmatrix} \sum x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n. \end{pmatrix}$$

Since

$$Det(H) = n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2 = n\left(\sum_{i=1}^{n}(x_i - \bar{x})^2\right),$$

similar to equation 8, $Det(H) > 0$ if the x_i's are not all equal. Also the first leading principal minor, $\sum x_i^2$, is > 0. Thus H is positive definite, i.e., our solution for m and b is a minimizer for the sum of squares $f(m, k)$.

Remark: We arrived at the same solution as in the linear algebra approach discussed in the framework of linear regression.

6.7 Approximative Optimization

For optimisation we need to find stationary points. However, sometimes the occurring equations cannot be solved explicitly (with reasonable effort) - so iterative optimization algorithms must be used. This holds especially for optimisation in the context of ML.

6.7.1 Gradient Descent

Gradient descent is a widely used algorithm for finding a minimum. The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$, so ∇f, points in direction of the steepest ascent. This fact leads to the idea of gradient descent:

The Gradient Descent Algorithm:

Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

1. Fix a starting point $\mathbf{x}_0$ (any point where f is defined) and set $k \leftarrow 0$.
2. While some predefined stopping condition is not met (like $k \geq K$, for some $K \in \mathbb{N}$ or $\|\nabla f(\mathbf{x}^k)\| \geq \varepsilon$ for some small $\varepsilon > 0$)
   (a) Fix a step–width $t_k \in \mathbb{R}$ (possibly depending on so far computed data, in particular $\mathbf{x}^k$, $f(\mathbf{x}^k)$, $\nabla f(\mathbf{x}^k)$).
   (b) $\mathbf{x}^{k+1} = \mathbf{x}^k - t_k \nabla f(\mathbf{x}^k)$.
   (c) $k \leftarrow k + 1$.
3. print($\mathbf{x}^k$).

Remarks:

1. Observe that according to $\mathbf{x}^{k+1} = \mathbf{x}^k - t_k \nabla f(\mathbf{x}^k)$ the next point is computed in direction $-\nabla f(\mathbf{x}^k)$ – this is the direction of the steepest descent. The core idea of the algorithm is to reach the minimum by always "going down" as fast as possible.
2. Choosing the step size t_k: It is reasonable to pick t_k by

$$t_k = \arg\min_{t \geq 0} f\left(\mathbf{x}^k - t \nabla f(\mathbf{x}^k)\right)$$

However, this computation is not always possible analytically, also for computational simplicity in many cases a constant step size is fixed for the whole algorithm.

3. There exist classes of functions (like strongly convex functions) for which the gradient descent algorithm always converges, if the step size is chosen suitably. However, unfortunately, in general there is the possibility that gradient descent does not find the desired minimum due to a complicated shape of f or a badly chosen starting point, stopping condition or step size.

Example: Let $f : 4x^2 - 4xy + 2y^2$ be a given function. I.e.

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 8x - 4y \\ -4x + 4y \end{pmatrix}.$$

Picking $\mathbf{x}^0 = (3, 4)$ gives

$$\begin{pmatrix} x^1 \\ y^1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \end{pmatrix} - t_0 \begin{pmatrix} 8 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 - 8t_0 \\ 4 - 4t_0 \end{pmatrix}.$$

To find t_0 as arg min as mentioned in above remarks, we should find t which minimises $f\left(\mathbf{x}^k - t\nabla f(\mathbf{x}^k)\right)$, i.e., for

$$e(t) = f\left(\begin{pmatrix} 3 - 8t \\ 4 - 4t \end{pmatrix}\right)$$

then, by the chain rule,

$$e'(t) = (8(3 - 8t) - 4(4 - 4t))(-8) + (-4(3 - 8t) + 4(4 - 4t))(-4) = 320t - 80,$$

therefore $e'(t) = 0$ for $t_0 = 1/4$. Using this step size we arrive at

$$\begin{pmatrix} x^1 \\ y^1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}.$$

And so on ....

6.7.2 Newton's Method

Let $\mathbf{f} : \mathbb{R}^n \to R^n$ be a differentiable function. The aim of Newtons method is to find those $\mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. The idea is to use the linear approximation given by the differential

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{y}) + D_{\mathbf{f}}(\mathbf{y})(\mathbf{x} - \mathbf{y}).$$

If $\mathbf{x}$ is such that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ then

$$\mathbf{x} \approx \mathbf{y} - (D_{\mathbf{f}}(\mathbf{y}))^{-1}\,\mathbf{f}(\mathbf{y}).$$

This gives rise to **Newton's algorithm**:

1. Pick a starting point $\mathbf{x}^0$ and set $k \leftarrow 0$.
2. While some predefined stopping condition is not met (like while $k \geq K$, for some $K \in \mathbb{N}$ or $\|\mathbf{f}(\mathbf{x}^k)\| \geq \varepsilon$ for some small $\varepsilon > 0$)
   (a) $\mathbf{x}^{k+1} = \mathbf{x}^k - \left(D_{\mathbf{f}}(\mathbf{x}^k)\right)^{-1}\mathbf{f}(\mathbf{x}^k)$.
   (b) $k \leftarrow k + 1$.
3. print($\mathbf{x}^k$).

Applying Newton's method to find a stationary point:

For a stationary point $\mathbf{x}$ one has to find solution of $\nabla\mathbf{f}(\mathbf{x}) = 0$. Therefore Newton's method translates to

1. Pick a starting point $\mathbf{x}^0$ and set $k \leftarrow 0$.
2. While some predefined stopping condition is not met (like while $k \geq K$, for some $K \in \mathbb{N}$ or $\|\nabla\mathbf{f}(\mathbf{x}^k)\| \geq \varepsilon$ for some small $\varepsilon > 0$)
   (a) $\mathbf{x}^{k+1} = \mathbf{x}^k - \left(H_{\mathbf{f}}(\mathbf{x}^k)\right)^{-1}\nabla\mathbf{f}(\mathbf{x}^k)$.
   (b) $k \leftarrow k + 1$.
3. print($\mathbf{x}^k$).

Remark: Observe that in the latter algorithm the second order Hessian H_f being the derivative of $\nabla\mathbf{f}$ replaced $D_{\mathbf{f}}$ from the original version of Newton's algorithm.

Example Let again $f : 4x^2 - 4xy + 2y^2$ be a given function. I.e.

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 8x - 4y \\ -4x + 4y \end{pmatrix}$$

and

$$H_f(x, y) = \begin{pmatrix} 8 & -4 \\ -4 & 4 \end{pmatrix}$$

This H_f is constant for all points (x, y). Now

$$H_f(x, y)^{-1} = \frac{1}{12}\begin{pmatrix} 3 & 1 \\ 3 & -2 \end{pmatrix}$$

Picking, e.g., as starting point again $\mathbf{x}^0 = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ gives

$$\mathbf{x}^1 = \begin{pmatrix} 3 \\ 4 \end{pmatrix} - \frac{1}{12}\begin{pmatrix} 3 & 1 \\ 3 & -2 \end{pmatrix}\begin{pmatrix} 8 \\ 4 \end{pmatrix} = \begin{pmatrix} 7/3 \\ 2/3 \end{pmatrix}.$$

And so on ...

7 The mathematical description of neural networks

7.1 The general idea

An artificial neuronal network (ANN or, short NN) is, motivated by biological neural networks, a weighted graph whose vertices are called neurons whose connections, imitating synapses, are the edges in the network. The weights display the strengths of the connections. A neuron can be active or not. In case it is active, its transmits information to the neurons it is connected with. This activation process is imitated by the so called activation function. Training a neural network means to adjust the weights such that the given task/data is approximated best possible. A detailed discussion of this process is the aim of these notes.

In the following we focus on so–called feedforward NNs whose graph is acyclic and directed. In case that cycles exist the NN is called a recurrent NN (RNN). However, the key ideas of training an RNN are related to training a feedforward NN so we concentrate on those.

7.2 The mathematical model

The ingredients of an NN are:

Layers: The NN is organised in layers, neurons from one layer are connected with the neurons of the next layer. The first layer, also called the input layer, takes the input values, the last layer, called the output layer, returns the output of the NN. The other layers are called hidden layers.

Here is one simplified picture of a feedforward NN:

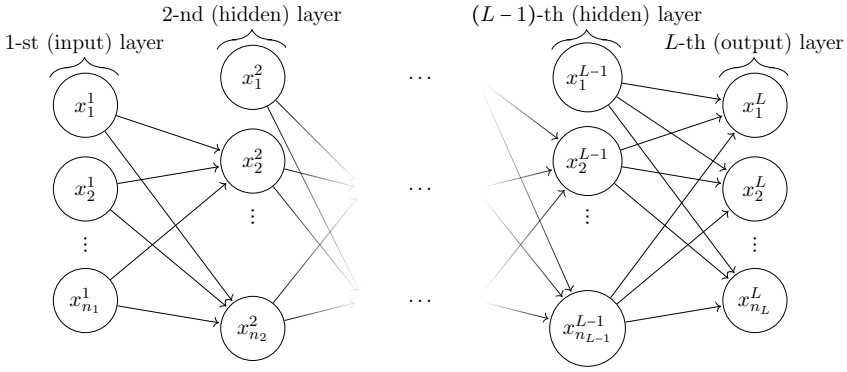


Figure 3: Network graph of a L-layer NN with n_1 input units and n_L output units. The l-th hidden layer contains n_l hidden units.

The values which are the output of the neuron j in layer k is denoted by $\boxed{x_j^l}$. Note in particular that x_j^1 stands for the j-th input and x_j^L for the j-th output.

Weights: For each layer l the outputs of the neuron x_j^l of the layer are multiplied by the weights assigned to each edge leaving the neuron. Each neuron, say the i-th of the next layer receives the sum of weighted outputs of the previous layers neurons connected to it. Additionally a so called bias is usually added as an extra (constant) input.

We write $\boxed{w_{ij}^l}$ for the weight assigned to the edge from the j-th neuron of the l-th layer to the ith neuron of the $(l+1)$-st layer and w_{i0}^l for the bias. Observe that the index of the taget neuron is first, then the index of the starting neuron – this notation will be justified later.

Using this notation, the total input at the i-th $(l+1)$-st layer neuron, denoted by net_i^{l+1} equals

$$net_i^{l+1} = \underbrace{w_{i0}^l}_{\text{bias}} + \underbrace{\sum_{j=1}^{n_l} w_{ij}^l x_j^l}_{\text{weighted inputs}} .$$

Adding the 0-th neuron to the NN in each layer and assign to it the constant output 1, the previous equation can be written more compactly.

$$\boxed{net_i^{l+1} = \sum_{j=0}^{n_l} w_{ij}^l x_j^l.}$$

Activation function: The input net_i^{l+1} of the i-th $(l+1)$-st layer neuron is finally the argument of the so called activation function f which mimics the biological activation potential. The output of f, i.e., $f(net_i^{l+1})$ is the output of the i-th $(l+1)$-st layer neuron, i.e. x_i^{l+1}.

There are several activation functions, they all display the biological idea – the neurons output is (essentially) zero (not active) or one (neuron fires). For computational purposes (see later), differentiable functions are used, one important activation function is the so called sigmoid function

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

Direct computation shows the identity $f'(x) = f(x)(1-f(x))$.

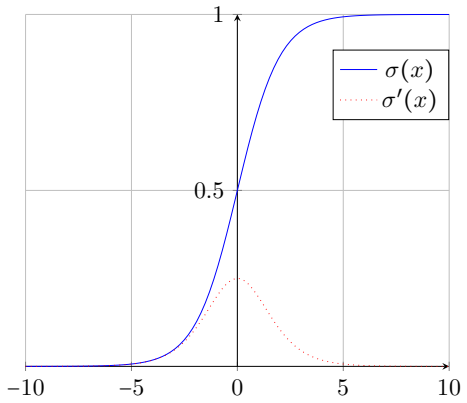


Figure 4: Plot of the sigmoid function σ and its derivative.

To sum up, a detailed picture of one neuron, say the i-th $(l+1)$-st layer neuron in some NN:

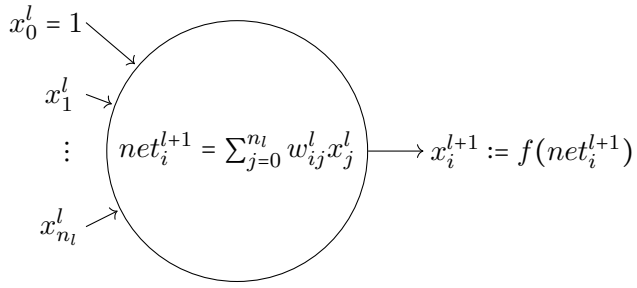


Figure 5: Single processing unit and its components. The activation function is denoted by f and applied on the actual input net_i^l of the unit to form its output $x_i^{l+1} := f(net_i^{l+1})$. $x_1^l, \ldots, x_{n_l}^l$ represent input from other units within the network.

Matrix representation: Observe that the flow of the input through the net can be denotes stepwise in vector/matrix form:

- 1st layer = Input:

$$\mathbf{x}^1 = \begin{pmatrix} x_0^1 = 1 \\ x_1^1 \\ x_2^1 \\ \vdots \\ x_{n_1}^1 \end{pmatrix}.$$

- Multiplied with first weights giving the inputs to the second layer:

$$\underbrace{\begin{pmatrix} net_1^2 \\ net_2^2 \\ \vdots \\ net_{n_2}^2 \end{pmatrix}}_{\mathbf{net^2}} = \underbrace{\begin{pmatrix} w_{10}^1 & w_{11}^1 & \ldots & w_{1n_1}^1 \\ w_{20}^1 & w_{21}^1 & \ldots & w_{2n_1}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_2 0}^1 & w_{n_2 1}^1 & \ldots & w_{n_2 n_1}^1 \end{pmatrix}}_{W^1} \cdot \underbrace{\begin{pmatrix} x_0^1 \\ x_1^1 \\ x_2^1 \\ \vdots \\ x_{n_1}^1 \end{pmatrix}}_{\mathbf{x}^1}.$$

  Here W^1 denotes the first layers weights matrix. We defined the indices of the weights to keep the usual matrix-index notation.

- Apply the activation function f to each of the n_2 second layer neurons

to obtain the second layers outputs x_i^2:

$$\underbrace{\begin{pmatrix} x_0^2 \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_{n_2}^2 \end{pmatrix}}_{\mathbf{x}^2} = \underbrace{\begin{pmatrix} 1 \\ f(net_1^2) \\ f(net_2^2) \\ \vdots \\ f(net_{n_2}^2) \end{pmatrix}}_{\mathbf{f(net^2)}}$$

- ... (this procedure repeats for each layer)

Finally we arrive at the output:

$$\boxed{\mathbf{x}^L = \mathbf{f}(W^{L-1} \ldots \underbrace{\mathbf{f}(W^2 \underbrace{\mathbf{f}(W^1 \mathbf{x}^1)}_{\mathbf{x}^2}) \ldots)}_{\mathbf{x}^3}}.$$

That is: An NN is the repeated alternating composition of linear functions and activation function applied to the input vector.

7.3 Backpropagation

7.3.1 Training to minimising the loss function

NNs are trained by given input and output data. The aim is to learn the effects of the input to the output, thus NNs are part of supervised learning algorithms. The NN stepwise updates the weights so that for a given input the difference (loss) between the output computed by the network and the real output is minimised. There are several loss functions $\mathcal{L}$ which quantify the loss, for now we focus on the M(ean) S(quared) Error.

More detailed: Assume the data set consists of N input vectors $\mathbf{x}(n) \in [0,1]^D$ and corresponding N output vectors $\mathbf{y}(n) \in [0,1]^E$. D and E are the number of inputs and outputs per datum, we assume them to be numerically and normalized between 0 and 1.

Remark: Normalising the data is before applying the NN is generally one important preprocessing step. It guarantees that input and output features

have comparable sizes and weight changes have comparable effects. How to normalise depends strongly on the activation function used in the NN. If the (nowadays preferred) tanh is used, whose range is in $[-1,1]$, then normalising in this range is common. Since we use the sigmoid function here, we assume normalisation w.r.t. its range, i.e. to $[0,1]$.

That is, if the L-th layer NN should approximate the given n-th output vector (given by the nth data point), i.e

$$\underbrace{\begin{pmatrix} x(n)_1^L \\ x(n)_2^L \\ \vdots \\ x(n)_{n_l}^L \end{pmatrix}}_{\mathbf{x}(n)^L} = NN_{\mathbf{w}} \underbrace{\left(\begin{pmatrix} x(n)_1 \\ x(n)_2 \\ \vdots \\ x(n)_{n_1} \end{pmatrix}\right)}_{NN(\mathbf{x}(n))},$$

where $n_1 = D$ and $n_L = E$ and $NN_{\mathbf{w}}(\mathbf{x})$ stands for the neuronal net with weights $\mathbf{w}$ applied to the input $\mathbf{x}$ should approximate the n-th target output

$$\underbrace{\begin{pmatrix} y(n)_1 \\ y(n)_2 \\ \vdots \\ y(n)_{n_l} \end{pmatrix}}_{\mathbf{y}(n)}.$$

Thus, we want to minimise the overall loss $L(\mathbf{w})$ (depending on the weights of the NN $\mathbf{w}$; in principle we minimise over all data available) – gradient descent is used to (approximatively) solve this minimisation task.

Concretely in the case where the loss function is the averaged mean squared error, this means we want to find weights $\mathbf{w}$ in the neuronal net NN such that

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\|\mathbf{y}(n) - NN_{\mathbf{w}}(\mathbf{x}(n))\|^2 \\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{k=1}^{E}\left[\mathbf{y}(n)_k - NN_{\mathbf{w}}(\mathbf{x}(n))_k\right]^2. \end{aligned}$$

7.3.2 The algorithm

Backpropagation is a direct implementation of gradient descend for the function $\mathcal{L}(\mathbf{w})$ which makes use of the special shape of NNs.

To explain the algorithm using the matrix representation can be used. However, the compact notation might be in the way of understanding the steps. We will therefore stick with the "element-wise" computation. Once again the ingredients:

- w_{ij}^l The weight assigned to the edge from the j-th neuron of the l-th layer to the ith neuron of the $(l+1)$-st layer and w_{i0}^l for the bias. These are the variables w.r.t. which the loss function should be minimised.
- $\mathcal{L}(\mathbf{w})$ The loss function. We concretely use the MSE.
- x_i^l The output of the i-th neuron in the l-th layer.
- $net_i^{l+1} = \sum_{j=0}^{n_l} w_{ij}^l x_j^l$, the total input in the i-th neuron in the $(l+1)$-st layer
- $f(x)$ the activation function. We concretely use the sigmoid function.

Since we want to minimise $\mathcal{L}(\mathbf{w})$, we need the gradient $\nabla_{\mathbf{w}}(\mathcal{L})$. The key step is the chain rule, since the weights of the l-th layer are 'inner arguments" (w.r.t to composition of functions), 'inside"

1. the outermost loss function $\mathcal{L}$,
2. $L-l+1$ nested net functions and
3. $L-l+1$ nested activation functions.

Stepwise this gives:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \frac{\partial \mathcal{L}}{\partial x_i^{l+1}} \frac{\partial x_i^{l+1}}{\partial w_{ij}^l} = \frac{\partial \mathcal{L}}{\partial x_i^{l+1}} \frac{\partial x_i^{l+1}}{\partial net_i^{l+1}} \frac{\partial net_i^{l+1}}{\partial w_{ij}^l}$$

Now, looking at the three factors closely shows:

- $\frac{\partial net_i^{l+1}}{\partial w_{ij}^l}$: The definition $net_i^{l+1} = \sum_{j=0}^{n_l} w_{ij}^l x_j^l$, immediately gives

$$\frac{\partial net_i^{l+1}}{\partial w_{ij}^l} = x_j^l.$$

- $\frac{\partial x_i^{l+1}}{\partial net_i^{l+1}}$: Again, by definition $x_i^{l+1} = f(net_i^{l+1})$. Thus

$$\frac{\partial x_i^{l+1}}{\partial net_i^{l+1}} = f'(net_i^{l+1}).$$

  In particular, if f is the sigmoid function:

$$\frac{\partial x_i^{l+1}}{\partial net_i^{l+1}} = f(net_i^{l+1})(1 - f(net_i^{l+1})).$$

- $\frac{\partial \mathcal{L}}{\partial x_i^{l+1}}$: Recall how the layers outputs are composed:

$$\mathbf{x}^{l+2} = \mathbf{f}(\mathbf{net}^{l+2}) = \mathbf{f}(W^{l+1}\mathbf{x}^{l+1}).$$

  Therefore $\mathbf{x}^{l+2}$, the output of the $(l+2)$-nd layer is a function of the previous, the $(l+1)$-st layer. In other words the output of any layer is the inner function of the output of the next function. If N_i^{l+2} denotes the set of all neurons that receive the output x_i^{l+1} of the i-th $(l+1)$-st layer neuron, the chain rule leads to

$$\frac{\partial \mathcal{L}}{\partial x_i^{l+1}} = \sum_{k \in N_i^{l+2}} \frac{\partial \mathcal{L}}{\partial net_k^{l+2}} \frac{\partial net_k^{l+2}}{\partial x_i^{l+1}} = \sum_{k \in N_i^{l+2}} \frac{\partial \mathcal{L}}{\partial x_k^{l+2}} \frac{\partial x_k^{l+2}}{\partial net_k^{l+2}} \frac{\partial net_k^{l+2}}{\partial x_i^{l+1}}$$

  I.e. The derivative of $\mathcal{L}$ w.r.t. any layer can be expressed with the derivatives of the following layer. This recursive property is the key element of the backpropagation. Observe that the summands simplify to

$$\frac{\partial \mathcal{L}}{\partial x_i^{l+1}} = \sum_{k \in N_i^{l+2}} \frac{\partial \mathcal{L}}{\partial x_k^{l+2}} f'(net_k^{l+2}) w_{ki}^{l+1}$$

  Note that the derivative w.r.t the last layer, i.e., the output, is simply the derivative of the loss function.

In particular, if $\mathcal{L}$ is the MSE of all outputs (for one fixed training example) then

$$\frac{\partial \mathcal{L}}{\partial x_i^{l+1}} = 1/2 \frac{\partial (\sum_{k=1}^{n_L} (x_k^L - y_k)^2)}{\partial x_i^L} = x_i^L - y_i,$$

i.e. the error in the i-th output.

Summing up:

Using the common notation

$$\delta_i^l = \frac{\partial \mathcal{L}}{\partial x_i^{l+1}} \frac{\partial x_i^{l+1}}{\partial net_i^{l+1}}$$

(δ_i^l is called the error of the j-th neuron in the $l+1$-st layer) we have

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \delta_i^l x_j^l$$

where

$$\boxed{\delta_i^l = \begin{cases} \frac{\partial \mathcal{L}}{\partial x_i^L} f'(net_i^{l+1}) & \text{if } x_i^{l+1} \text{ an output neuron,} \\ \sum_{k \in N_i^{l+2}} \delta_k^{l+1} w_{ki}^{l+1} f'(net_i^{l+1}) & \text{if } x_i^{l+1} \text{ an inner neuron.} \end{cases}}$$

If $\mathcal{L}$ is in particular MSE and f is the sigmoid function the previous equation simplifies to

$$\delta_i^l = \begin{cases} (x_i^L - y_i) x_i^L (1 - x_i^L) & \text{if } x_i^{l+1} = x_i^L \text{ an output neuron,} \\ \sum_{k \in N_i^{l+2}} \delta_k^{l+1} w_{ki}^{l+1} x_i^{l+1} (1 - x_i^{l+1}) & \text{if } x_i^{l+1} \text{ an inner neuron.} \end{cases}$$

These rules allow to compute the gradient of $\mathcal{L}(\mathbf{w})$ tracking the layers backwards. As soon as the gradient is fully computed the weights are updated according to the principle of the gradient descent, i.e.

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla \mathcal{L}(\mathbf{w})(\mathbf{w}^{(old)}).$$

η denotes the so called learning rate, which is typically a number between 0 and 1.

A remark concerning δ_i^l

Focussing on the error just as a function of the input of the i-th $l+1$-st layer neuron, i.e. $\mathcal{L} = \mathcal{L}(net_i^{l+1})$ (all other variables are fixed). How much does changing this very input influence the error. According to the definition of the derivative

$$\mathcal{L}(net_i^{l+1} + h) \approx \mathcal{L}(net_i^{l+1}) + \underbrace{\frac{\partial \mathcal{L}}{\partial net_i^{l+1}}}_{=\delta_i^l} h.$$

Thus, if δ_i^l is a large positive or negative value changing the input by some (even small) h of opposite sign to δ_i^l reduces the error significantly. Conversely, if δ_i^l is small, it only big changes of the input can lower the error, i.e. the neuron is already contributing optimally to the overall outcome. Therefore δ_i^l is called the error of the neuron.

7.3.3 One worked Example

Assume the following NN is given:

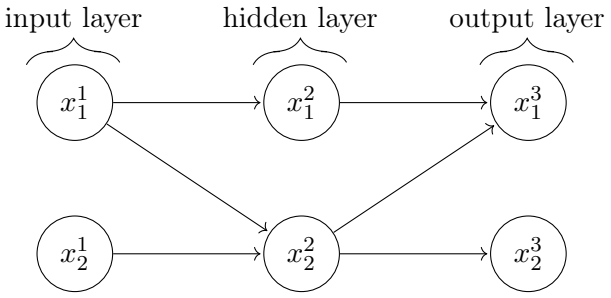


Figure 6: One Example

Assume given are

1. the input-vector $\mathbf{x} = (0.5, 0.25)$ and the target output vector $\mathbf{y} = (0.2, 0.7)$.

2. The starting weights

w_{11}^1	w_{21}^1	w_{22}^1	w_{11}^2	w_{12}^2	w_{22}^2
0.3	0.4	0.5	0.6	0.2	0.8

.

3. The activation function is the sigmoid function.

4. The loss function is MSE.

Feed-forward

Feed forward stands for computing the output of the NN for the given input. I.e, using matrix representation,

$$\begin{aligned}
\begin{pmatrix} net_1^2 \\ net_2^2 \end{pmatrix} &= \begin{pmatrix} w_{11}^1 = 0.3 & w_{12}^1 = 0 \\ w_{21}^1 = 0.4 & w_{22}^1 = 0.5 \end{pmatrix} \begin{pmatrix} x_1^1 = 0.5 \\ x_2^1 = 0.25 \end{pmatrix} = \begin{pmatrix} 0.15 \\ 0.325 \end{pmatrix} \\
\begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} &= \begin{pmatrix} f(net_1^2) \\ f(net_2^2) \end{pmatrix} = \begin{pmatrix} 0.5374298 \\ 0.5805423 \end{pmatrix} \\
\begin{pmatrix} net_1^3 \\ net_2^3 \end{pmatrix} &= \begin{pmatrix} w_{11}^2 = 0.6 & w_{12}^2 = 0.2 \\ w_{21}^2 = 0 & w_{22}^2 = 0.8 \end{pmatrix} \begin{pmatrix} x_1^2 = 0.5374298 \\ x_2^2 = 0.5805423x \end{pmatrix} = \begin{pmatrix} 0.4385664 \\ 0.4644338 \end{pmatrix} \\
\begin{pmatrix} x_1^3 \\ x_2^3 \end{pmatrix} &= \begin{pmatrix} f(net_1^3) \\ f(net_2^3) \end{pmatrix} = \begin{pmatrix} 0.6079174 \\ 0.6140655 \end{pmatrix}
\end{aligned}$$

Since the starting weights were assigned randomly it the first approximation cannot be seen as an approximation of the target output vector $\mathbf{y} = (0.2, 0.7)$.

Backpropagation

To adjust the weights based on the gradient descent to the error we use backwards the algorithm explained in the previous section.

Starting with the output layer:

The error is measured with the loss function

$$\mathcal{L} = 1/2(x_1^3 - y_1)^2 + 1/2(x_2^3 - y_2)^2$$

Recall that, e.g. $x_1^3 = f(w_{11}^2 x_1^2 + w_{12}^2 x_2^2)$ and therefore, by the chain rule,

$$\frac{\partial \mathcal{L}}{\partial w_{11}^2} = (x_1^3 - y_1) f'(w_{11}^2 x_1^2 + w_{12}^2 x_2^2) x_1^2 = (x_1^3 - y_1)(x_1^3(1 - x_1^3) x_1^2.$$

In our example this gives $\frac{\partial \mathcal{L}}{\partial w_{11}^2} = 0.05225359$ and $\delta_1^2 = (x_1^3 - y_1)(x_1^3(1 - x_1^3) = 0.09722867$. Analogously one obtains $\frac{\partial \mathcal{L}}{\partial w_{12}^2} = (x_1^3 - y_1)(x_1^3(1 - x_1^3) x_2^2 = 0.056445369$ and $\frac{\partial \mathcal{L}}{\partial w_{22}^2} = (x_2^3 - y_2)(x_2^3(1 - x_2^3) x_2^2 = -0.01182306$ as well as $\delta_2^2 = -0.02036554$.

Next we derive the weights leading to the hidden layer: In order to illustrate the formula derived in the previous section we will proceed stepwise. Since x_2^2 is connected to x_1^3 as well as to x_2^3 the derivates $\frac{\partial \mathcal{L}}{\partial w_{21}^1}$ and $\frac{\partial \mathcal{L}}{\partial w_{22}^1}$ are the most interesting ones.

Let us look at $\frac{\partial \mathcal{L}}{\partial w_{21}^1}$: w_{21}^1 is an argument of

1. $net_2^2 = w_{21}^1 x_1^1 + w_{22}^1 x_2^1$ and consequently of
2. $x_2^2 = f(net_2^2)$ and consequently of
3. $net_1^3 = w_{11}^2 x_1^2 + w_{12}^2 x_2^2$ as well as $net_2^3 = w_{21}^2 x_1^2 + w_{22}^2 x_2^2$ and consequently of
4. $x_1^3 = f(net_1^3)$ as well as $x_2^3 = f(net_2^3)$.

Therefore $\frac{\partial \mathcal{L}}{\partial w_{21}^1}$ must be derived according to the chain rule, i.e.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{21}^1} &= \frac{\partial 1/2(x_1^3 - y_1)^2 + 1/2(x_2^3 - y_2)^2}{\partial w_{21}^1} \\ &= \underbrace{\frac{\partial \mathcal{L}}{\partial x_1^3}\frac{\partial x_1^3}{\partial net_1^3}}_{=\delta_1^2}\frac{\partial net_1^3}{\partial w_{21}^1} + \underbrace{\frac{\partial \mathcal{L}}{\partial x_2^3}\frac{\partial x_2^3}{\partial net_2^3}}_{=\delta_2^2}\frac{\partial net_2^3}{\partial w_{21}^1}\end{aligned}$$

Moreover

$$\frac{\partial net_1^3}{\partial w_{21}^1} = \frac{\partial net_1^3}{\partial x_2^2}\frac{\partial x_2^2}{\partial net_2^2}\frac{\partial net_2^2}{\partial w_{21}^1} = w_{12}^2 x_2^2(1 - x_2^2)x_1^1$$

and, analogously

$$\frac{\partial net_2^3}{\partial w_{21}^1} = \frac{\partial net_2^3}{\partial x_2^2}\frac{\partial x_2^2}{\partial net_2^2}\frac{\partial net_2^2}{\partial w_{21}^1} = w_{22}^2 x_2^2(1 - x_2^2)x_1^1$$

Summing up, we arrive at

$$\frac{\partial \mathcal{L}}{\partial w_{21}^1} = \delta_1^2 w_{12}^2 x_2^2(1 - x_2^2)x_1^1 + \delta_2^2 w_{22}^2 x_2^2(1 - x_2^2)x_1^1.$$

Numerically this means that $\frac{\partial \mathcal{L}}{\partial w_{21}^1} = 0.0003839348$. In the same fashion

$$\frac{\partial \mathcal{L}}{\partial w_{22}^1} = \delta_1^2 w_{12}^2 x_2^2(1 - x_2^2)x_2^1 + \delta_2^2 w_{22}^2 x_2^2(1 - x_2^2)x_2^1 = 0.0001919674$$

and

$$\frac{\partial \mathcal{L}}{\partial w_{11}^1} = \delta_1^2 w_{11}^2 x_1^2 (1 - x_1^2) x_1^1 = 0.007251285$$

If we have a learning rate of $\eta = 0.8$, say, then the updated weights are

$$\begin{array}{c|l} w_{11}^1 & 0.3 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{11}^1} = 0.294199 \\ w_{21}^1 & 0.4 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{21}^1} = 0.3996929 \\ w_{22}^1 & 0.4 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{22}^1} = 0.4998464 \\ w_{11}^2 & 0.6 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{11}^2} = 0.5581971 \\ w_{12}^2 & 0.2 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{12}^2} = 0.1548437 \\ w_{22}^2 & 0.8 - 0.8 \frac{\partial \mathcal{L}}{\partial w_{22}^2} = 0.8094584 \end{array}$$

Feed-forward again

Evaluating the NN again, similar to before but with the new weights and the original input gives the new output vector

$$\begin{pmatrix} x_1^3 \\ x_2^3 \end{pmatrix} = \begin{pmatrix} 0.59615644 \\ 0.615357 \end{pmatrix}$$

This vector is closer to the given target vector, due to the small gradients the convergence is slow.

Next, Backpropagation is applied again ....

7.4 Additional remarks and concepts

7.4.1 Batch, Mini Batch and Stochastic gradient descent

So far we trained the NN only with one data point – in reality the NN should learn a whole data set. Using the whole data set at once is called batch gradient descent - the loss function is the average error over all training data, i.e.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2n} \sum_{\mathbf{x}} \mathcal{L}_{\mathbf{x}}(\mathbf{w})$$

The gradient of this loss function takes all data points into consideration, it is sometimes called the true gradient. Algorithmically, due to linearity of the gradient, i.e., since

$$\nabla_{\mathbf{w}}(\mathcal{L}_{\mathbf{x}}(\mathbf{w}) + \mathcal{L}_{\mathbf{y}}(\mathbf{w})) = \nabla_{\mathbf{w}}(\mathcal{L}_{\mathbf{x}}(\mathbf{w})) + \nabla_{\mathbf{w}}(\mathcal{L}_{\mathbf{y}}(\mathbf{w})),$$

the true gradient can be computed by computing the gradient for each training example separately using backpropagation and averaging those gradients afterwards.

However, if the data set is big, this computation can still be slow. To accelerate the computation one partitions the data set into smaller subsets, called mini-batches, and updates the weights w.r.t. the gradients of the loss function based on these (randomly chosen but approximately equal sized) mini batches. Each loop over all mini batches is called an epoch. I.e., after one epoch each data point is used exactly once. If the mini batches have size 1 one calls the procedure stochastic gradient descent. One randomly picks one data point and runs back propagation once to update the weights, picks another data point and repeats the process.

7.5 Neural nets and classification

In case the neural net is used for classification the last activation function is typically the so-called softmax function and the loss function is the cross entropy discussed in Section 1.15.1. We discuss them shortly:

7.5.1 Softmax

In case of classification one uses typically hot-encoding for the different classes occurring in the given problem. I.e., if there are n classes, then the binary vector of lengths n which has a 1 at position k and 0s everywhere else encoded the k-th class. Thus, when modelling a n-class classification problem one presents the (target) output as such hot encoding binary vectors.

Hat encodings can be seen as probability distribution – the probability that a certain input $\mathbf{x}$ leads to class k. In the training set, the probability that the i-th input $\mathbf{x}_i$ belongs to its class k_i is 1, which is exactly the hot encoding of the output.

Thus, the aim of a classification NN is to learn the probabilities of each class – the softmax function as the last layers activation function, i.e., directly before the output, has several features useful for this goal.

The standard (unit) softmax function $\sigma : \mathbb{R}^K \to [0,1]^K$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

Note that $\sigma(\mathbf{z})_i$ stands for the i-th component of this function, i.e. K is the total output dimension, i.e., - in the setting of one hot classification the number of classes.

We list properties of the softmax function:

1. The K output values of the softmax function add up to one, i.e. the values given by softmax function can be seen as a probability distribution.

2. The softmax function can be seen as a smooth approximation of hot encoding of the arg max function whose value is which index has the maximum value of a given list of numbers.

3. The derivative of the softmax function is simple, which is important for computing the gradient:

$$\frac{\partial \sigma(z_1, \dots, z_K)_i}{\partial z_k} = \sigma(z_1, \dots, z_K)_i (\delta_{ik} - \sigma(z_1, \dots, z_K)_k),$$

   where the indices i and k denote the component of σ and δ_{ik} is the so-called Kronecker delta which is 1 if $i = k$ and 0 otherwise.

7.5.2 Cross entropy as loss function

In classification models the loss function is usually the cross entropy. Recall that cross entropy is motivated by the usual entropy function and can be used to measure the distance between two probability distributions.

The cross entropy of a probability distribution $q(x)$ over another probability distribution $p(x)$ is given by

$$H(p, q) = -\sum_x p(x) \log(q(x)).$$

Interpreting the hot encoded outcome of the training set as the probability distribution to be approximated and the output of the NN (the softmax function) as distribution q the loss can, by the just made argument, be measured by the cross entropy of q over p – this the the applied loss function.

7.6 The universal approximation theorem

Any NN can be seen as a function which maps an D dimensional input vector to an E dimensional output. A data set can be seen as a table of values of a fixed function $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^E$. The aim of a NN is to approximate the unknown function best possible for every possible input $\mathbf{x} \in \mathbb{R}^D$.

The universal approximation theorem is a striking theoretical result which makes the strength of NNs even more visible. It says that a neural network with only one hidden layer (but possibly many neurons contained in this layer) can approximate any such function $\mathbf{f}$ arbitrarily well. Unfortunately, these theorems are typically non-constructive.

8 Transformers

The Transformer architecture is a type of deep learning model introduced in the paper "Attention is All You Need" by Vaswani et al. It's primarily used for various natural language processing (NLP) tasks, but it has also been adapted for other sequence-based tasks in machine learning.

A transformer model transforms a set of vectors into a new set of vectors of the same dimension which carry more information. E.g. if those vectors represent some words (word vectorization) the transformed vectors contain information which words are semantically close to each other, are like to appear next to each other, ...

The key innovation of the Transformer architecture is its attention mechanism, which allows the model to focus on different parts of the input sequence when generating an output sequence. This attention mechanism enables the model to capture long-range dependencies more effectively.

8.1 Transformer Layer

Let $(\mathbf{x}_n)$ be a set of N input vectors from $\mathbb{R}^d$ called tokens whose components are called features. These vectors are usually combined row-wise in a $N \times d$ matrix X. As indicated in the introduction, the key idea of a transformer is to transform X into some other matrix $\tilde{X}$ carrying more (learned) information. I.e., we will now describe a mechanism

$$\tilde{X} = \text{TRANSFORMER}(X).$$

8.1.1 Self attention

The first step is to self-relate the tokens. The idea underlying this step is to

1. compute new vectors as linear combination of the given ones with
2. weights depending on the similarity between tow vectors.

This can be archived by defining

$$\mathbf{y}_m = \sum_{n=1}^{N} a_{mn}\mathbf{x}_n,$$

i.e. $Y = AX$, where the coefficients a_{mn} are computed using the inner product (measuring similarity) and the softmax function (for normalization)

$$a_{mn} = \frac{e^{\mathbf{x}_m \cdot \mathbf{x}_n}}{\sum_{i=1}^{N} e^{\mathbf{x}_m \cdot \mathbf{x}_i}}.$$

Indeed, this definition guarantees that

1. the entries of A satisfy $a_{mn} \geq 0$ and
2. $\sum_{n=1}^{N} a_{mn} = 1$.

In short, we defined

$$Y = \text{Softmax}(XX^T)X.$$

However, this approach has some disadvantages:

1. it only measures how the tokens are related to itself (that is why this approach is called self attention),
2. all feature values play an equal role
3. there are no learnable weights (which measure the importance of certain features or the relation between tokens).

8.1.2 Attention

Motivated by the just mentioned facts one firstly applies a linear transformation of (learnable) weights to X, i.e. $X' = XU$. XX^T becomes then XUU^TX^T, which is still a symmetric matrix. In order to additionally get rid of the symmetry of $\text{Softmax}(XUU^TX^T)XU$ (some tokens refer to others only in one direction, e.g. in language processing 'Monday' is a day of the week but not every day of the week is Monday), one therefore defines three

matrices depending on different learnable weight matrices:

$$\begin{aligned} Q &= XW^Q \quad (\text{query, where }), W^Q \in \mathbb{R}^{d\times d'}, \\ K &= XW^K \quad (\text{query, where }), W^K \in \mathbb{R}^{d\times d'}, \\ V &= XW^V \quad (\text{query, where }), W^V \in \mathbb{R}^{d\times d''}. \end{aligned}$$

Usually one sets $D' = D'' = D$. With these matrices one defines the scaled attention head

$$Y = \text{ATTENTION}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d'}}\right)V.$$

The denominator $\sqrt{d'}$ serves as a scaling factor which helps with the issue that the gradient of the softmax function is small for high input values.

8.1.3 Multi head attention

To enable the model to learn more features one multiplies the concept of the attention mechanism to obtain the so called multi-head attention:

Assume there are H attention heads, i.e.

$$H_h = \text{ATTENTION}(Q_h, K_h, V_h), \quad h = 1, \ldots, H,$$

where $Q_h = XW_h^Q$, $V_h = XW_h^V$ and $K_h = XW_h^K$, i.e. each H_h has dimension $N \times d''$ Then set

$$Y = \text{Concatenate}(H_1, H_2, \ldots, H_H)W^m,$$

where W^m is a weight matrix in $\mathbb{R}^{Hd''\times d}$ such that Y has the same dimension $N \times d$ as the input X. The following sketch illustrates the model.

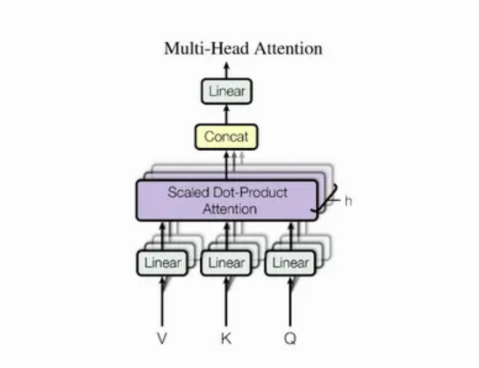


Figure 7: Attention head

8.1.4 Final steps

The multi head attention blocks are the main ingredients to any transformer model. To stack several attention heads on top of each other and still maintain a good training speed on includes residual connections combined with a normalisation of the result, i.e.

$$Z(X) = \text{LayerNorm}(Y(X) + X).$$

LayerNorm(A) denotes here the procedure of normalizing along each feature, i.e. computing column-wise mean μ_j and standard deviation σ_j and then normalising $a_{ij} = \frac{a_{ij}-\mu_j}{\sigma_j}$.

Additionally, to further increase the flexibility and the non-linearity of the model one finally includes a fully connected neural net, to finally obtain

$$\tilde{X} = \text{TRANSFORMER}(X) = \text{LayerNorm}(NN(Z) + Z).$$

8.2 Further remarks

8.3 Embeddings and positional encodings

As discussed in the previous section the idea of a transformer model is to transform tokes, i.e., sequences, to sequences with more structure. Transformers are used in sequential data (like language) where different tokens

do not occur independently but as a sequence themselves (so a sequence of tokes (e.g. words) where context and position in the sequence matter.

To firstly translate a given input data to sequences one often uses so called embeddings. These are usually the weights of a certain layer of small NNs where the network learns self supervised to "fill in gaps" and thus learn context of the input data.

To also be able to record at which position each token occurs in a given input sequence one uses so called positional encoding. Here the position, say n, of the n–th input token $\mathbf{x}_n$ is encoded by so called position vector ($\mathbf{r}_n$ for position n) which is then added to $\mathbf{x}_n$, i.e. one defines new token vectors $\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n$ which also depend on the position of each token.

8.4 Different Transformer based architectures

8.4.1 Decoder Transformers

The idea of decoder transformers is to compute a autoregressive model of the form

$$p(x_n|x_1, \ldots, x_{n-1}).$$

This can be done by using several transformer layers that take a sequence of tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and transform them to tokens $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N$, both of the same dimension D. To obtain the targeted probability distribution one finally adds a softmax layer, i.e.

$$Y = \text{Softmax}(\tilde{X}W)$$

where W is some additional weight matrix and Softmax is again

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Remark: G(enerative) P(retrained) T(ransformers) is a decoder transformer type used in NLP. These models are trained unsupervised, the models learns to predict the next word in a given text corpus.

8.4.2 Encoder Transformers

Encoder transformer take sequences as inputs and returns fixed length vectors as output (classes e.g.). The architecture is again combination stacked transformer layers together with a softmax function. The key difference to decoder transformers is the fact that the encoder model is able to look ahead (e.g. it sees the whole sentence at one) whilst the decoder always predicts the next element in a sequence (e.g. of words). The fact that the model can "look in both directions" gave one established architecture of this type its name B(idirectional) E(ncoder) R(epresentation of) T(ransformers).

8.4.3 Sequence-to-Sequence Transformers

For the sake of completeness we mention the combination of the endcoder-decoder architecture which is, e.g., used for translation tasks.

9 Further topics in Optimisation

9.1 Constrained Optimisation: First Order Conditions

9.1.1 Definitions

The typical constraint optimisation problem has the following shape

$$\begin{array}{ll} \text{maximize} & f(x_1,\dots,x_n) \\ \text{subject to} & g_1(x_1,\dots,x_n) \le b_1,\dots,\ g_k(x_1,\dots,x_n) \le b_k \\ & h_1(x_1,\dots,x_n) \le c_1,\dots,\ h_m(x_1,\dots,x_n) \le c_m. \end{array}$$

The function f is called objective function, the functions g_i and h_i are called constraint functions. The g_i define inequality constraints, the h_i equality constraints. Common constraints are non-negativity constraints, i.e. $x_1,\dots,x_n \ge 0$.

9.1.2 Equality constraints

The two dimensional case

Let f and h be real valued functions on the $\mathbb{R}^2$. Let $c \in \mathbb{R}$. Suppose we want to solve the problem

$$\text{maximize } f(x,y) \quad \text{subject to} \quad h(x,y) = c. \tag{9}$$

A geometrical approach: For $l \in \mathbb{R}$, the set of points (x,y) such that $f(x,y) = l$ is a level set of the function $f(x,y)$. If (x^*,y^*) is a solution of (9) and $f(x^*,y^*) = l^*$ then the level set given by $f(x,y) = l^*$ touches the level set given by $h(x,y) = c$ at (x^*,y^*) but it does not intersect it since otherwise on one side of the intersection would lie higher values of $f(x,y)$ subject to $h(x,y) = c$. But touching means that the level sets have parallel tangents and hence also parallel normal vectors. Recall that the normal vector to $f(x,y) = l^*$ at (x^*,y^*) is given by $\nabla f(x^*,y^*)$ and to $h(x,y) = c$ at (x^*,y^*) is given by $\nabla h(x^*,y^*)$. So, parallelity translates to $\nabla f(x^*,y^*) = \mu \nabla h(x^*,y^*)$.

Summing up, a solution (x^*, y^*) satisfies

$$\begin{aligned} f_x(x^*, y^*) - \mu h_x(x^*, y^*) &= 0, \\ f_y(x^*, y^*) - \mu h_y(x^*, y^*) &= 0, \\ h(x^*, y^*) - c &= 0 \end{aligned}$$

This can elegantly be rewritten in the following way: Let the Lagrangian of the problem (9) be the function

$$L(x, y, \mu) = f(x, y) - \mu(h(x, y) - c).$$

Then any solution of (9) is necessarily a critical point of the Lagrangian $L(x, y, \mu)$.

The general equality case

Let $f, h_1, \ldots, h_m : \mathbb{R}^n \to \mathbb{R}$ be real valued functions on the $\mathbb{R}^N$ of class $\mathcal{C}^1$. Let $a_1, \ldots a_m \in \mathbb{R}$. We want solve the problem

$$\text{find the extreme values of } f(\mathbf{x}) \quad \text{subject to} \quad h_i(\mathbf{x}) = a_i,\ i = 1, \ldots, m. \tag{10}$$

Let $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$ be the function whose components are the functions h_i and $J_{\mathbf{h}}(\mathbf{x})$ its Jacobian at the point $\mathbf{x}$.

Fact:

(Lagrange multiplier Theorem) Let the problem (10) be given. Assume $\mathbf{x}^*$ solves (10) and

$$rank(J_{\mathbf{h}}(\mathbf{x}^*)) = m. \tag{11}$$

Then there exists $\boldsymbol{\mu}^* = (\mu_1^*, \ldots, \mu_m^*)$ such that $(\mathbf{x}^*, \boldsymbol{\mu}^*)^t$ is a critical point of the Lagrangian

$$L(\mathbf{x}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{l=1}^{m} \mu_l (h_l(\mathbf{x}) - a_l).$$

Remarks:

1. The condition 11 is called (non-degenerate or linear) constraint qualification. It guarantees the linear independence of $\nabla h_l(\mathbf{x}^*)$.

2. According to the Theorem the first order conditions hold, i.e.,

$$\frac{\partial L}{\partial x_k}(\mathbf{x}^*, \boldsymbol{\mu}^*) = \frac{\partial f}{\partial x_k}(\mathbf{x}^*) - \sum_{l=1}^{m} \mu_l \frac{\partial h_l}{\partial x_k}(\mathbf{x}^*) = 0 \qquad \text{for all } k = 1, 2, \ldots, N$$

$$\frac{\partial L}{\partial \mu_i}(\mathbf{x}^*, \boldsymbol{\mu}^*) = h_i(\mathbf{x}^*) - a_i = 0 \qquad \text{for all } i = 1, 2, \ldots, m.$$

   These equalities $n+m$ equalities in $n+m$ variables are used to compute the candidates solving the maximisation problem.

3. A totally similar result holds for minimisation problems.

To understand the Langrange multiplier theorem observe that if $\mathbf{x}^*$ is a max of f on the level set $\mathbf{h}^{-1}(\mathbf{a})$ then

$$\nabla f \cdot \mathbf{v} = 0.$$

for every C^1 curve $\mathbf{r}(t) \in \mathbf{h}^{-1}(\mathbf{a})$ such that $\mathbf{r}(0) = \mathbf{x}^*$ and $\mathbf{r}'(0) = \mathbf{v}$. Thus, every $\mathbf{v} \in Th(\mathbf{x}^*) = nullDh(\mathbf{x}^*)$ satisfies $\nabla f \cdot \mathbf{v} = 0$. So, if $\mathbf{v}$ satisfies the linear equation system

$$\underbrace{\begin{pmatrix} \nabla h_1(\mathbf{x}^*) \\ \nabla h_2(\mathbf{x}^*) \\ \vdots \\ \nabla h_m(\mathbf{x}^*) \end{pmatrix}}_{=A} \mathbf{v} = \mathbf{0}$$

$\mathbf{v}$ also solves

$$\underbrace{\begin{pmatrix} \nabla f(\mathbf{x}^*) \\ \nabla h_1(\mathbf{x}^*) \\ \nabla h_2(\mathbf{x}^*) \\ \vdots \\ \nabla h_m(\mathbf{x}^*) \end{pmatrix}}_{=B} \mathbf{v} = \mathbf{0},$$

meaning that the set of solutions of the first system is a subset of the set of solutions of the second set. Since $\mathbf{v}$ solving homogenous systems of linear equations are by definition in the kernel, $dim(kerA) \leq dim(kerB)$. On the other hand, since B has more rows, clearly any solution of the second equation is one of the first, i.e. $dim(kerA) \geq dim(kerB)$. Summing up $dim(kerA) = dim(kerB)$ and therefore

$$rankA = n - dim(kerA) = n - dim(kerB) = rankB.$$

Consequently $\nabla f(\mathbf{x}^*)$ is linearly dependent of $\nabla h_i l(\mathbf{x}^*)$. By linear independence of the vectors $\nabla h_l(\mathbf{x}^*)$ there are unique coefficients to write

$$\nabla f(\mathbf{x}^*) = \mu_1 \nabla h_1(\mathbf{x}^*) + \ldots + \mu_m \nabla h_m(\mathbf{x}^*).$$

9.1.3 The inequality case

The two dimensional case:

Let f and h be real valued functions on the $\mathbb{R}^2$ of class $\mathcal{C}^n$, $n \geq 1$. Let $c \in \mathbb{R}$ Suppose we want solve the problem

$$\text{maximize } f(x,y) \quad \text{subject to} \quad g(x,y) \leq c. \tag{12}$$

Again a geometrical approach: Dealing with an inequality implies that we must distinguish the following two cases:

- First case: If (x^*, y^*) is a solution of (12) and $f(x^*, y^*) = l^*$ and (x^*, y^*) lies on the boundary of the area $g(x,y) \leq c$ then – as in the equality case – the level set given by $f(x,y) = l^*$ touches level set given by $g(x,y) = c$ at (x^*, y^*). Moreover the gradients are not only parallel, but they point in the same direction, namely out of the area $g(x,y) \leq c$, since both functions increase when leaving the constraint area. Thus necessarily $\nabla f(x^*, y^*) = \lambda \nabla g(x^*, y^*)$ and $\lambda > 0$. In this case we call the constraint **binding**.

- Second case: If (x^*, y^*) is a solution of (12) and (x^*, y^*) lies in the interior of the area $g(x,y) \leq c$ then the constraint $g(x,y) \leq c$ can be neglected (it is **not binding**), since (x^*, y^*) is in fact a local maximum of f. Thus, one can write $\nabla f(x^*, y^*) = \lambda \nabla g(x^*, y^*)$ and $\lambda = 0$.

The two cases can be summed up using the Lagrangian $L(x,y,\lambda) = f(x,y) - \lambda(h(x,y) - c)$ as before: Any solution of (12) satisfies the following (also called first order–) conditions:

$$\begin{aligned} L_x(x,y,\lambda) &= f_x - \lambda h_x = 0, \\ L_y(x,y,\lambda) &= f_y - \lambda h_y = 0, \\ \lambda(g(x,y) - c) &= 0, \\ \lambda &\geq 0, \\ g(x,y) &\leq c. \end{aligned}$$

Observe that the last three (in)equalities sum up both cases. The third equality is called complementary slackness condition since it guarantees, that at least one of the latter two inequalities is in fact an equality.

The general inequality case

Let $f, g_1, \ldots, g_k : \mathbb{R}^n \to \mathbb{R}$ be real valued functions on $\mathbb{R}^n$ of class $\mathcal{C}^1$. Let $b_1, \ldots b_k \in \mathbb{R}$. We want solve the problem

$$\text{find the maximal values of } f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \leq b_i,\ i = 1, \ldots, k. \tag{13}$$

Fact: (Inequality case Lagrange method) Let the problem (13) be given. Assume $\mathbf{x}^*$ solves (13). Assume for $\mathbf{x}^*$ are r constraints binding (say the first k_0, i.e., $g_i(\mathbf{x}) = b_i$ for $i = 1, \ldots, k_0$) and the remaining ones are not binding (i.e., $g_i(\mathbf{x}) < b_i$ for $i = k_0 + 1, \ldots, k$). Assume $rank(J_{\mathbf{g}}(\mathbf{x}^*)) = k_0$, where $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{k_0}$ is the function whose components are the functions g_i, for $i = 1, \ldots, k_0$, and $J_{\mathbf{g}}(\mathbf{x})$ its Jacobian at the point $\mathbf{x}$. Let the Lagrangian of the problem be defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{l=1}^{k} \lambda_l (g_l(\mathbf{x}) - b_l).$$

Then there exists $\boldsymbol{\lambda}^* = (\lambda_1^*, \ldots, \lambda_k^*)$ such that $(\mathbf{x}^*, \boldsymbol{\lambda}^*)^t$ satisfies the following first order conditions:

$$\begin{aligned}
\text{For all } j = 1, 2, \ldots, n \quad &: \quad \frac{\partial L}{\partial x_j}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \frac{\partial f}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{k} \lambda_l^* \frac{\partial g_l}{\partial x_j}(\mathbf{x}^*) = 0, \\
\text{For all } l = 1, 2, \ldots, k \quad &: \quad \lambda_l^* (g_l(\mathbf{x}^*) - b_l) = 0, \\
&\quad\ \ \lambda_l^* \geq 0, \\
&\quad\ \ g_l(\mathbf{x}^*) \leq b_l.
\end{aligned}$$

Assume that the variables x_i are ordered in such a way, that

$$Det\left(\frac{\partial(g_1, .., g_{k_0})}{\partial(x_1, .., x_{k_0})}(\mathbf{x}^*)\right) \neq 0.$$

For the binding inequality constraints the Lagrange multiplier theorem can be applied to find (uniquely) multipliers $\lambda_1^*, \ldots, \lambda_k^*$ such that,

$$\frac{\partial L}{\partial x_j}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \frac{\partial f}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{k} \lambda_l^* \frac{\partial g_l}{\partial x_j}(\mathbf{x}^*) = 0,$$

definitely holds for all $j = 1, 2, \ldots, n$ if the remaining λ_l^*'s are fixed = 0 f§r $l = k_0 + 1, \ldots, m$.

Therefore it only remains to show that $\lambda_i \geq 0$ for $i = 1, \ldots, k_0$ – to make the idea of the argument transparent, consider the following observations:

1. We can approximate f close to $\mathbf{x}^*$ by

   $$f(\mathbf{x}^* + \mathbf{h}) \approx f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*) \cdot \mathbf{h},$$

   and since

   $$\nabla f(\mathbf{x}^*) = \lambda_1 \nabla g_1(\mathbf{x}^*) + \ldots + \lambda_{k_0} \nabla g_{k_0}(\mathbf{x}^*),$$

   we arrive at

   $$f(\mathbf{x}^* + \mathbf{h}) \approx f(\mathbf{x}^*) + \lambda_1 \nabla g_1(\mathbf{x}^*) \cdot \mathbf{h} + \ldots + \lambda_{k_0} \nabla g_{k_0}(\mathbf{x}^*) \cdot \mathbf{h}.$$

2. The inequality constraints $g_i(\mathbf{x}) \leq b_i$ are assumed to be binding for $i = 1, \ldots, k_0$ and $\nabla g_i(\mathbf{x}^*)$ points in the direction where g_i is increasing. Therefore, if a vector $\mathbf{h}$ satisfies $\nabla g_i(\mathbf{x}^*) \cdot \mathbf{h} < 0$ then one can move a bit in direction $\mathbf{h}$ while the constraint $g_i(\mathbf{x}) \leq b_i$ still holds.

3. If there was an $\lambda_i < 0$ and one can construct such a vector $\mathbf{h}$ such that $\nabla g_i(\mathbf{x}^*) \cdot \mathbf{h} < 0$ whilst $\nabla g_j(\mathbf{x}^*) \cdot \mathbf{h} = 0$ for all $j \neq i$ then $\mathbf{x}^* + \mathbf{h}$ would satisfy the constraints but

   $$f(\mathbf{x}^* + \mathbf{h}) \approx f(\mathbf{x}^*) + \lambda_1 \nabla g_1(\mathbf{x}^*) \cdot \mathbf{h} + \ldots + \lambda_{k_0} \nabla g_{k_0}(\mathbf{x}^*) \cdot \mathbf{h} \tag{14}$$
   $$= f(\mathbf{x}^*) \lambda_i \nabla g_i(fx^*) \cdot \mathbf{h} > f(\mathbf{x}^*) \tag{15}$$

   contradicting the maximality of $\mathbf{x}^*$.

4. To construct such a vector, the implicit function theorem can be used:

Assuming that $\lambda_i < 0$ construct the following function $\mathbf{F} : \mathbb{R}^{n+1} \to \mathbb{R}^n$

$$\begin{array}{lcll}
F_1(\mathbf{x},t) & = & g_1(\mathbf{x}) & -b_1, \\
F_2(\mathbf{x},t) & = & g_2(\mathbf{x}) & -b_2, \\
 & \vdots & & \\
F_{i-1}(\mathbf{x},t) & = & g_{i-1}(\mathbf{x}) & -b_{i-1}, \\
F_i(\mathbf{x},t) & = & g_i(\mathbf{x}) & -b_i - t, \quad \longleftarrow (!) \\
F_{i+1}(\mathbf{x},t) & = & g_{i+1}(\mathbf{x}) & -b_{i+1}, \\
 & \vdots & & \\
F_{k_0}(\mathbf{x},t) & = & g_{k_0}(\mathbf{x}) & -b_{k_0}, \\
F_{k_0+1}(\mathbf{x},t) & = & x_{k_0+1} & -x^*_{k_0+1}, \\
 & \vdots & & \\
F_n(\mathbf{x},t) & = & x_n & -x^*_n.
\end{array}$$

Then

$$DF(\mathbf{x}^*,0) = \begin{pmatrix}
\frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_k\,0} & \frac{\partial g_1}{\partial x_{k_0+1}} & \cdots & \frac{\partial g_1}{\partial x_n} \\
\frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_2}{\partial x_k\,0} & \frac{\partial g_2}{\partial x_{k_0+1}} & \cdots & \frac{\partial g_2}{\partial x_n} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial g_{k_0}}{\partial x_1} & \cdots & \frac{\partial g_{k_0}}{\partial x_{k_0}} & \frac{\partial g_{k_0}}{\partial x_{k_0+1}} & \cdots & \frac{\partial g_{k_0}}{\partial x_n} \\
0 & \ldots & 0 & 1 & \ldots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \ldots & 0 & 0 & \ldots & 1
\end{pmatrix}.$$

In particular

$$Det(DF(\mathbf{x}^*,0)) = Det\left(\frac{\partial(g_1,..,g_{k_0})}{\partial(x_1,..,x_{k_0})}(\mathbf{x}^*)\right) \neq 0,$$

so by the Implicit function theorem there exists a C^1 function $\mathbf{x}(t)$ such that $\mathbf{x}(0) = \mathbf{x}^*$ and $\mathbf{F}(\mathbf{x}(t),t) = \mathbf{0}$. $\mathbf{F}(\mathbf{x}(t),t) = \mathbf{0}$ implies that

1. $g_i(\mathbf{x}(t)) = b_i$ for $i \neq j$ and $i = 1,\ldots,k_0$,
2. $g_j(\mathbf{x}(t)) = b_j - t$,
3. $x(t)_l = x^*_l$ for $l = k_0+1,\ldots,n$.

Therefore, computed directly, $g_i'(\mathbf{x}(t)) = 0$ for $i \neq j$ and $i = 1,\ldots,k_0$, and $g_j'(\mathbf{x}(t)) = -1$. By the chain rule

$$(g_i(\mathbf{x}(t)))'\big|_{t=0} = \frac{\partial g_i}{\partial x_1}(\mathbf{x}^*)\cdot x_1'(0) + \ldots + \frac{\partial g_j}{\partial x_i} n(\mathbf{x}^*)\cdot x_n'(0).$$

Therefore, writing $\mathbf{h} = \mathbf{x}'(0)$ according to 14 above, we have

$$\nabla g_i(\mathbf{x}^*) \cdot \mathbf{h} = \begin{cases} -1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

If $\mathbf{x}^*$ is a max, then $f(\mathbf{x}^*) > f(\mathbf{x}(t))$ and therefore

$$0 \geq f'(\mathbf{x}(t))\big|_{t=0} = \nabla f(\mathbf{x}^*) \cdot \mathbf{h} = \lambda_1 \nabla g_1(\mathbf{x}^*) \cdot \mathbf{h} + \ldots + \lambda_{k_0} \nabla g_{k_0}(\mathbf{x}^*) \cdot \mathbf{h} = -\lambda_i,$$

contradicting the original assumption $\lambda_i < 0$.

9.1.4 Mixed constraints –SB 18.4

The following sums up what we discussed so far:

Suppose $\mathbf{x}^*$ is a solution of the problem

$$\begin{array}{ll} \text{maximize} & f(x_1, \ldots, x_n) \\ \text{subject to} & g_1(x_1, \ldots, x_n) \leq b_1, \ldots, \ g_k(x_1, \ldots, x_n) \leq b_k \\ & h_1(x_1, \ldots, x_n) = c_1, \ldots, \ h_m(x_1, \ldots, x_n) = c_m. \end{array}$$

and suppose the first k_0 of the inequality constraints are binding. If

$$\begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{k_0}}{\partial x_1} & \cdots & \frac{\partial g_{k_0}}{\partial x_n} \\ \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_m} \end{pmatrix}$$

satisfies the constraint qualification, i.e. its rank is $k_0 + m$ then there exist $\lambda_1^*, \ldots, \lambda_k^*$ and $\mu_1^*, \ldots, \mu_m^*$ such that

- For all $j = 1, 2, \ldots, n$

$$\frac{\partial L}{\partial x_j}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \frac{\partial f}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{k} \lambda_l^* \frac{\partial g_l}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{m} \mu_l^* \frac{\partial h_l}{\partial x_j}(\mathbf{x}^*) = 0$$

- For all $l = 1, 2, \ldots, m$

$$h_i(\mathbf{x}^*) = c_i,$$

- For all $l = 1, 2, \ldots, k$

$$\lambda_l^*(g_l(\mathbf{x}^*) - b_l) = 0, \quad \lambda_l^* \geq 0, \quad \text{and } g_l(\mathbf{x}^*) \leq b_l,$$

where

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) &- \lambda_1(g_1(\mathbf{x}) - b_1) - \ldots - \lambda_k(g_k(\mathbf{x}) - b_k) \\ &- \mu_1(h_1(\mathbf{x}) - c_1) - \ldots - \mu_m(h_m(\mathbf{x}) - c_m) \end{aligned}$$

denotes the Lagrangian of the problem.

9.1.5 Constrained minimisation problem

Looking through the ideas of the discussion so far the following necessary condition for constrained minimisation problems can be derived.

Suppose $\mathbf{x}^*$ is a solution of the problem

$$\begin{array}{rl} \text{minimize} & f(x_1, \ldots, x_n) \\ \text{subject to} & g_1(x_1, \ldots, x_n) \geq b_1, \ldots, \ g_k(x_1, \ldots, x_n) \geq b_k \\ & h_1(x_1, \ldots, x_n) = c_1, \ldots, \ h_m(x_1, \ldots, x_n) = c_m. \end{array}$$

and suppose the first k_0 of the inequality constraints are binding. If

$$\begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{k_0}}{\partial x_1} & \cdots & \frac{\partial g_{k_0}}{\partial x_n} \\ \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_m} \end{pmatrix}$$

satisfies the constraint qualification, i.e. its rank is $k_0 + m$ then there exist $\lambda_1^*, \ldots, \lambda_k^*$ and $\mu_1^*, \ldots, \mu_m^*$ such that

- For all $j = 1, 2, \ldots, n$

$$\frac{\partial L}{\partial x_j}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \frac{\partial f}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{k} \lambda_l^* \frac{\partial g_l}{\partial x_j}(\mathbf{x}^*) - \sum_{l=1}^{m} \mu_l^* \frac{\partial h_l}{\partial x_j}(\mathbf{x}^*) = 0$$

- For all $l = 1, 2, \ldots, m$

$$h_i(\mathbf{x}^*) = c_i,$$

- For all $l = 1, 2, \ldots, k$

$$\lambda_l^*(g_l(\mathbf{x}^*) - b_l) = 0, \quad \lambda_l^* \geq 0, \quad \text{and } g_l(\mathbf{x}^*) \geq b_l,$$

where

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) &- \lambda_1(g_1(\mathbf{x}) - b_1) - \ldots - \lambda_k(g_k(\mathbf{x}) - b_k) \\ &- \mu_1(h_1(\mathbf{x}) - c_1) - \ldots - \mu_m(h_m(\mathbf{x}) - c_m) \end{aligned}$$

denotes the Lagrangian of the problem.

Remark:
It is equivalent to formulate the minimisation problem with the constraint inequality signs as in the max-type problem, i.e. $g_l(\mathbf{x}) \leq b_l$ but alternatively require for the inequality multipliers $\lambda_l^* \leq 0$ for $i = 1, \ldots, k$.

9.1.6 Kuhn Tucker formulation

A special type of inequality constraint problems is one with non-negativity constraints:

$$\begin{aligned} \text{maximize} \quad & f(x_1, \ldots, x_n) && (16) \\ \text{subject to} \quad & g_1(x_1, \ldots, x_n) \leq b_1, \ldots, \; g_k(x_1, \ldots, x_n) \leq b_k && (17) \\ & x_1 \geq 0, \ldots, x_n \geq 0 && (18) \end{aligned}$$

.

The Lagrangian of this problem is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) - \lambda_1(g_1(\mathbf{x}) - b_1) - \ldots - \lambda_k(g_k(\mathbf{x}) - b_k) + \nu_1 x_1 + \ldots \nu_n x_n.$$

This gives the first order conditions

1. (First order w.r.t. $\mathbf{x}$) For $l = 1, \ldots, n$:

$$\frac{\partial L}{\partial x_l} = \frac{\partial f}{\partial x_l} - \lambda_1 \frac{\partial g_1}{\partial x_l} - \ldots - \lambda_k \frac{\partial g_k}{\partial x_l} + \nu_l = 0. \tag{19}$$

2. (First order w.r.t. $\boldsymbol{\lambda}$ and complementary slackness) For $l = 1, \ldots, k$:

$$\lambda_l (g_l(\mathbf{x}) - b_l) = -\lambda_l \frac{\partial L}{\partial \lambda_l} = 0 \tag{20}$$

3. (complementary slackness) For $l = 1, \ldots, n$:

$$\nu_l x_l = 0 \tag{21}$$

4. (non-negativity) For $l = 1, \ldots, n$:

$$\nu_l \geq 0. \tag{22}$$

5. (non-negativity) For $l = 1, \ldots, k$:

$$\lambda_l \geq 0. \tag{23}$$

6. (constraints) For $l = 1, \ldots, k$:

$$g_l(\mathbf{x}) \leq b_l \tag{24}$$

7. (constraints) For $l = 1, \ldots, n$:

$$x_l \geq 0. \tag{25}$$

Defining the Kuhn Tucker Lagrangian (ignoring the non-negativity constraints 25) as

$$\tilde{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \lambda_1 (g_1(\mathbf{x}) - b_1) - \ldots - \lambda_k (g_k(\mathbf{x}) - b_k), \tag{26}$$

i.e., $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \tilde{L}(\mathbf{x}, \boldsymbol{\lambda}) + \nu_1 x_1 + \ldots \nu_n x_n$, we can rewrite:

1. 19 as $\frac{\partial L}{\partial x_l} = \frac{\partial \tilde{L}}{\partial x_l} + \nu_l = 0$, i.e.

$$\frac{\partial \tilde{L}}{\partial x_l} = -\nu_l \tag{27}$$

2. Combining equations27 and 22 as well as 27 and 21 gives

$$\frac{\partial \tilde{L}}{\partial x_l} \leq 0 \quad \text{and} \quad x_j \frac{\partial \tilde{L}}{\partial x_j} = 0. \tag{28}$$

3. Combing 20 with the definition of $\tilde{L}$ gives

$$\frac{\partial \tilde{L}}{\partial \lambda_l} = \frac{\partial L}{\partial \lambda_l} = b_l - g_l(\mathbf{x}) \geq 0. \tag{29}$$

Summing up, we arrive at the following first order conditions for the Kuhn Tucker Lagrangian:

$$\boxed{\frac{\partial \tilde{L}}{\partial x_j} \leq 0, \quad x_j \frac{\partial \tilde{L}}{\partial x_j} = 0, \quad \frac{\partial \tilde{L}}{\partial \lambda_l} \geq 0, \quad \lambda_l \frac{\partial \tilde{L}}{\partial \lambda_l} = 0} \tag{30}$$

holds for all $j = 1, \ldots, n$ and all $l = 1, \ldots, k$.

Fact:

Let $\mathbf{x}^*$ be a solution of the maximisation problem 16 and assume that

$$\frac{\partial(g_1, .., g_{l_0})}{\partial(x_1, .., x_{j_0})}(\mathbf{x}^*)$$

has rank l_0 where the g_l, $1, \ldots, l_0$, are assumed to be the binding constraints at $\mathbf{x}^*$ and x_j, $1, \ldots, j_0$, are assumed to be those coordinates of $\mathbf{x}^*$ which are > 0, so the non-binding non-negativity constraints. Then there are nonnegative $\lambda_1^*, \ldots, \lambda_k^*$ such that the Kuhn Tucker Lagrangian 26 satisfies the first order conditions 30.

9.2 Three applications of constrained optimisation

9.2.1 Principal component analysis

To address the curse of dimensionality as mentioned in Section 3.2.5 one can reduce the data set to a lower dimensional subspace which contains the most relevant features of the data. One well established approach in this direction

is the so called P(rincipal) C(omponent) A(nalysis). PCA is an unsupervised algorithm.

The idea: Given data $M = \{\mathbf{x}_i : 1 \leq i \leq n\}$ where each given data vector is of dimension D, i.e. there are D features. The aim is to find a direction $\mathbf{u} \in \mathbb{R}^d$ (i.e. a unit vector) such that M projected on $\mathbf{u}$ contains as much information as possible.

In the context of PCA, the information that is preserved in direction $\mathbf{u}$ is simply the variance of the projected (one-dimensional) data.

We have therefore established the following constrained maximization problem:

$$\max_{\mathbf{u}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{u}^T (\mathbf{x}_i - \bar{\mathbf{x}})^2 \mathbf{u},$$
$$\text{subject to } \mathbf{u}^T \mathbf{u} = 1.$$

Observe that

$$\frac{1}{n} \sum_{i=1}^{n} \mathbf{u}^T (\mathbf{x}_i - \bar{\mathbf{x}})^2 \mathbf{u} = \mathbf{u}^T \left(\frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})^2 \right) \mathbf{u} = \mathbf{u}^T S \mathbf{u},$$

where S is the covariance matrix of the data M.

Using the Lagrange multiplier λ this translates to (already the derivative of the Lagrangian):

$$2S\mathbf{u} = 2\lambda\mathbf{u}$$
$$\mathbf{u}^T \mathbf{u} = 1.$$

The first equation says, that λ is one of the Eigenvalues and $\mathbf{u}$ one of the Eigenvectors of S, it is only open which one. However, multiplying the first equation by $\mathbf{u}^T$ from left and using the second equation for the right hand side gives

$$\mathbf{u}^T S \mathbf{u} = \lambda.$$

Thus, we arrived on the original objective function on the left hand side. Summing up: The solution to the original problem is the Eigenvector of the covariance matrix S of the data corresponding to the largest Eigenvalue of S. Moreover: Since S is symmetric, there exists and orthonormal collection

of D Eigenvectors of S and it can be shown that the Eigenvectors to the next largest Eigenvalues are the next "most important directions".

Summing up: for PCA one picks the set of $d < D$ Eigenvectors $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ of the d largest Eigenvalues of the covariance matrix S of the data M and projects the data on the d dimensional subspace generated by U, i.e. the new features are the coordinates w.r.t. U which are obtained by $U^T M$.

9.2.2 Max Entropy again

Assume you have some information on a finite probability distribution and you want to find the distribution with minimal assumptions. Then the Maximum Entropy principle says that the distribution whose entropy is maximal is to be favoured. Recall that the entropy of a distribution was defined as

$$-\sum_{i=1}^{6} p_i \log p_i$$

and measured the amount of surprise/information stored in the distribution. I.e. the higher the entropy, the more surprising are the outcomes, i.e. the less information we assume on the distribution.

As a concrete example assume you (only) know the expectation (estimated by the average of the rolled numbers) of a given dice, say m (if the dice was fair, m was 3.5). What is the most plausible distribution of this dice? Using the max entropy principle this translates to the optimisation problem

$$\begin{aligned} &\text{maximise } -\sum_{i=1}^{6} p_i \log p_i, \\ &\text{constraint to } \sum_{i=1}^{6} p_i = 1 \text{ and } \sum_{i=1}^{6} i p_i = m. \end{aligned}$$

The Lagrangian for this problem is

$$L(p_i, \lambda, \mu) = -\sum_{i=1}^{6} p_i \log p_i - \lambda \left(\sum_{i=1}^{6} p_i - 1 \right) - \mu \left(\sum_{i=1}^{6} i p_i - m \right).$$

Its FOC give the equations

$$p_i = e^{-1-\lambda-i\mu}, \tag{31}$$

$$\sum_{i=1}^{6} p_i = 1, \tag{32}$$

$$\sum_{i=1}^{6} i p_i = m. \tag{33}$$

Plugging the first equation (31) in the second (32) gives

$$\sum_{i=1}^{6} e^{-i\mu} = e^{1+\lambda}$$

Dividing the latter two equations (33) and (32) and plugging in the first equation (31) gives

$$m = \frac{\sum_{i=1}^{6} i p_i}{\sum_{i=1}^{6} p_i} = \frac{\sum_{i=1}^{6} i e^{-i\mu}}{\sum_{i=1}^{6} e^{-i\mu}},$$

and thus

$$m \sum_{i=1}^{6} e^{-i\mu} = \sum_{i=1}^{6} i e^{-i\mu}$$

which, unfortunately, cannot be solved explicitly. Numerically one obtains for fixed m the following values for λ and μ which consequently determine p_i:

For $m = 3.5$ (the expectation of the fair dice) the max entropy distribution is the uniform distribution, i.e. $p_i = 1/6$, as expected. For $m = 4$, the result is

$$0.1030650, 0.1227304, 0.1461479, 0.1740337, 0.2072402, 0.2467827.$$

For $m = 5$, the result is

$$0.02053167, 0.03853431, 0.07232208, 0.13573574, 0.25475194.$$

and, for $m = 5.9$, the result is

$$0.000000, 0.000100, 0.000700, 0.007500, 0.082600, 0.909100.$$

I.e., the distribution gets more and more skewed to the left, the more m exceed 3.5.

9.2.3 Support vector machines

S(upport) V(ector) M(achines) are used to classify binary labeled data which are - in best case - linearly separable. It can be shown that the computation of the separating hyperplane can be done by solving the constrained optimisation problem

Minimize $\|\mathbf{w}\|$ subject to $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$ for $i = 1, \ldots, n$.

10 Concepts of dynamic programming

10.1 The general idea

The basic idea of dynamic programming is the so called principle of optimality:

A problem can be solved optimally by breaking it into sub-problems and recursively finding solutions to these sub problems.

This principle is applied in many areas, esp. in computer science. Problems like finding a shortest path in a weighted graph or sequence alignment can be solved using this approach.

10.2 The deterministic framework

The general idea is the following: Given are

1. X a set of possible states. x_t denotes the state of at time t. If the time can be measured stepwise (i.e. in $\mathbb{N}$), then we speak of a discrete time setting. Otherwise, i.e. if $t \in \mathbb{R}$, we speak of a continuous tie problem. We start at some state x_0.

2. In each state we can perform one action out of a certain amount of actions. The set of possible actions in state x_t is denoted by $\Gamma(x_t)$. One action is denoted by $a_t \in \Gamma(x_t)$.

3. $T(x, a)$ denotes the state which is reached from state x after taking action a.

4. $F(x, a)$ denotes the instant payoff from state x and taking action a.

5. $\beta < 1$ is the so called discount factor modelling impatience - the payoff of future steps counts less than the current payoff.

Under these assumptions, an infinite-horizon decision problem takes the following form:

$$V(x_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, a_t),$$

subject to the constraints

$$a_t \in \Gamma(x_t),\ x_{t+1} = T(x_t, a_t),\ \forall t = 0, 1, 2, \ldots.$$

The Bellman equation

Using the recursive property of this definition one directly obtains

$$V(x_0) = \max_{a_0} \Big\{ F(x_0, a_0) + \beta \left[\max_{\{a_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} F(x_t, a_t) : a_t \in \Gamma(x_t),\ x_{t+1} = T(x_t, a_t),\ \forall t \geq 1 \right] \Big\}.$$

subject to the constraints $a_0 \in \Gamma(x_0),\ x_1 = T(x_0, a_0)$.

Here we are choosing a_0, knowing that our choice will cause the time 1 state to be $x_1 = T(x_0, a_0)$. That new state will then affect the decision problem from time 1 on. The whole future decision problem appears inside the square brackets on the right.

So far it seems we have only made the problem uglier by separating today's decision from future decisions. But we can simplify by noticing that what is inside the square brackets on the right is the value of the time 1 decision problem, starting from state $x_1 = T(x_0, a_0)$.

Therefore, we can rewrite the problem as a recursive definition of the value function:

$$V(x_0) = \max_{a_0} \{ F(x_0, a_0) + \beta V(x_1) \},$$

subject to the constraints:$a_0 \in \Gamma(x_0),\ x_1 = T(x_0, a_0)$. This is the Bellman equation. It can be simplified even further if we drop time subscripts and plug in the value of the next state:

$$V(x) = \max_{a \in \Gamma(x)} \{ F(x, a) + \beta V(T(x, a)) \}.$$

The Bellman equation is classified as a functional equation, because solving it means finding the unknown function V, which is the value function. Recall that the value function describes the best possible value of the objective, as a function of the state x. By calculating the value function, we will also find the function $a(x)$ that describes the optimal action as a function of the state; this is called the policy function.

10.3 Markov decision processes

The fundamental idea presented in the previous section carries over to so called Markov decision problems. Here the setting is (motivated by the last section):

1. A (finite) set of states, now denoted by s, and for each state a set of actions $A(s)$, denoted by a which lead from one state to another state.

2. A transition probability, where only the current step is relevant (Markov property), i.e. given is the probability to reach state s' if the state s and the actions a are known, notation $\mathbf{P}(s'|s, a)$.

3. A reward function $R(s, a, s')$ which assigns to the action a leading from state s to state s' a (positive or negative) reward. Negative rewards make sense to teach the model to reach a predefined final state.
   Sometimes the rewards are also only depending on s, the current state, i.e. $R = R(s)$. This changes the following algorithms depending on which approach is chosen.

4. A discount factor $\gamma \leq 1$ which describes that future rewards are less and less important.

With this vocabulary we can formulate the goal of a MDP:

In an MDP the outcome of the behaviour of an agent is measured by the overall (discounted) rewards earned for the single actions taken by the agent. We want to maximise these rewards, a solution to the problem must specify which action maximises this overall reward. Any solution which specifies the action in every state is called a policy, typically denoted with π. I.e. $\pi(s)$ denotes what action the agent should take in state s. An optimal policy, typically denoted by π^*, is a policy which maximises the overall reward.

Finally we must clarify what we mean with the overall reward which we aim to maximise: We focus on infinite horizon, stationary policies. Infinite horizon means that in principle the agent can choose actions infinitely many times, i.e. there is no fixed time limit in our MDP. Stationary means that the optimal action does not depend on the time but only on the current state. We will moreover focus on additive, discounted rewards. The overall reward of a sequence of states s_i with action a_i taken in state s_i is then given by

$$\begin{aligned} & U(s_0, a_0, s_1, a_1, \dots) \\ &= R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \dots \\ &= \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}). \end{aligned}$$

This value U, i.e. the sum of discounted rewards of a given history is called utility. Since the each action is a random experiment, the utility of a given policy π is not determined. In order to compare policies we therefore use the expectation, i.e. we define the utility of a given policy π when starting in state s by

$$U^{\pi}(s) = E\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right),$$

where $s_0 = s$.

Further we set

$$U(s) = \max_{\pi} U^{\pi}(s)$$

and

$$\pi^*(s) = \arg\max_{\pi} U^{\pi}(s)$$

and call $U(s)$ is the true utility of the MDP in state s and $\pi^*(s)$ the optimal policy starting in s. I.e., $U^{\pi^*}(s)$ is the true utility of the MDP in state s.

Analogously to section 10.2 we use the recursive property to derive the Bellman equation.

The Bellman equation for MDP

For any state s let again $U(s) = \max_{\pi} U^{\pi}(s)$. Then it can be shown (takes some steps, since the expectation needs to be controlled), that also for MDP a Bellman type equation holds, i.e., we have

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|a, s) U(s').$$

This functional equation can now be applied to construct U (and consequently π^*) iteratively. The mathematical background for the convergence of the algorithm presented in the next section is Banach's fixed point theorem.

10.4 Value iteration

As announced we present an algorithm which iteratively approximates the utility function - this algorithm is called value iteration. The algorithm can be extended to directly approximate the optimal policy π^* (policy iteration algorithm), we omit further details here and refer to the literature.

Given is a MDP, i.e. a set of states and actions on these states, transition probabilities, and rewards as well as a discount γ.

Then the algorithm reads as follows:

- Set $U(s) = 0$ for all states s, $t \leftarrow 1$
- While $t \neq 0$
  - $U \leftarrow U'$, $\delta \leftarrow 0$
  - for each state s do
    - $U'(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|a,s) U(s')$
    - if $|U'(s) - U(s)| > \delta$ then $\delta \leftarrow |U'(s) - U(s)|$
  - If $\delta < \frac{\varepsilon(1-\gamma)}{\gamma}$ then $t \leftarrow 0$

  return(U)

Remark: If the reward is depending on the action and the reached state the update rule changes to

$$U_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s,s') \left(R_a(s,s') + \gamma U_i(s') \right) \right\}.$$

10.5 Q-learning

General reinforcement learning faces the additional challenge that the model does not have information about its environment. I.e., compared to the setting discussed in the previous section now the set of states and also the transition probabilities are unknown. In principle there are two possible approaches:

1. Exploration: Gather as much information about its environment and finally learning the environment.

2. Exploitation: Follow a already found policy and greedily maximise the profit obtained by this policy.

Typical algorithms try to balance these two approaches – with sometimes more emphasis on exploration in the beginning and on exploitation later on.

Q-learning is a so called temporal difference (TD) method (where - put simply - the current model of the world is stepwise updated if new information is different to the one learned so far). The Q (quality) function is a so-called action-utility function, i.e. $Q(s,a)$ is a function not only depending on the current state s but also on the taken action a. As with utilities such a Q function must also satisfy the Bellman equilibrium when the Q values are correct, i.e.

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|a,s) \max_{a'} Q(s',a').$$

This equation can be used to define the TD difference update equation

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

which is calculated whenever action a is executed in state s leading to state s'. α is the learning rate. If α decreases the more numbers a state has been visited, then this updating rule will converge to the $Q(s,a)$ satisfying the Bellman equilibrium. The idea behind this update rule is to stick with the already found Q value a fraction of $1-\alpha$ and update a fraction of α as indicated. n the first paragraph of this subsection.

Observe that this TD - update rule is model-free, it does not contain the transition probabilities.

Finally set $U(s) = \max_a Q(s,a)$.

11 The final project

The final project consists of two parts

1. Some applied Python stuff,
2. Some computations 'by hand".

Please present your findings as two .pdf files (use a pdf-merger if necessary), one for each part. The python stuff related file should be a "report-like" .pdf which explains the outcomes of your tasks and how you obtained them, an explanation of the used methods, ... The solution of the "by hand" example is a scan of the handwritten calculations of the given task.

What to do in detail?

11.1 Python part

The goal: Use a group-dependent subset of the well known mnist data-set to train and compare two models which do a binary classification.

To get the group-specific data set you need two numbers

1. the group number, which is the 6 digit number obtained by concatenating the last three digits of the group members student IDs and use this group number.
2. the group digit which is the third digit of your group number.

To obtain the data set take 10000 random-samples of the Mnist data set according to your group number - enter your group number in the python code where the data set is generated. Then your two models should learn to classify your group digit correctly.

The tasks:

1. Work through the provided code and explain the model. In particular explain what the SGD-Classifier does/ how it works? If helpful, extend the code to clarify what is the concept of the model, what it predicts, etc. Finally explain the model evaluation.

2. Extend the provided notebook and apply a Convolutional Neural Network (CNN) to the same binary classification task. There are many online tutorials for the Mnist dataset with CNNs - pick any of those and adapt them to your groups data set and the binary classification task. Experiment a bit with some of the hyper-parameters of the used model, explain what your model does (indicate how CNNs work, no details are necessary but feel free to add them) and what hyper-parameters you have picked.

3. Compare both of your models? Which one is better, which one would you recommend?

Present the key steps and the findings of your data set in a nicely written report. Include python code, graphs, ..

11.2 By hand part

To individualize your network first fix the following numbers. Let

- α be the median of the digits of the group number,
- β = 2 + remainder of the group number divided by 7,
- γ group digit

Given is the simple neural network:

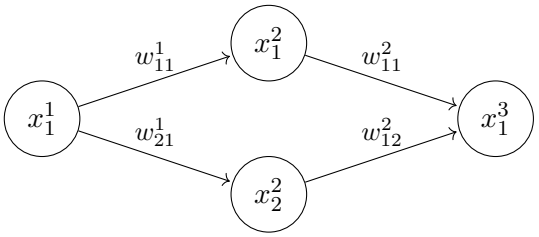


Figure 8: Neural network

The starting weights are

$$w_{11}^1 = \alpha/20, \quad w_{21}^1 = (\alpha + \beta)/20, \quad w_{11}^2 = 0.1, \quad w_{12}^2 = 0.3.$$

and the training data has the two points $(0.2, \beta/10)$ and $(0.6, \gamma/10)$.

Let the activation function be the sigmoid function and the loss function MSE. Compute the gradient of the loss function w.r.t. the weights. Highlight the recursive character of the derivation (as far as possible in this simple neural net) and compute two training epochs of backpropagation by hand using stochastic gradient descent.

12 Some References

1. Deep Learning; Bishop and Bishop
2. Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow; Geron.
3. Data analysis – a Bayesian tutorial; Sivia and Skilling.
4. Probabilistic graphical models; Koller and Friedman.
5. Artificial Intelligence; Russell and Norvig.
6. Probability theory: The logic of science; Jaynes.
7. Probability, Frequency and Reasonable Expectation; Cox.
8. Hornik et al: Multilayer Feedforward Networks are Universal Approximator

   `https://cognitivemedium.com/magic_paper/assets/Hornik.pdf`
9. A visual proof that neural nets can compute any function

   `http://neuralnetworksanddeeplearning.com/chap4.html`