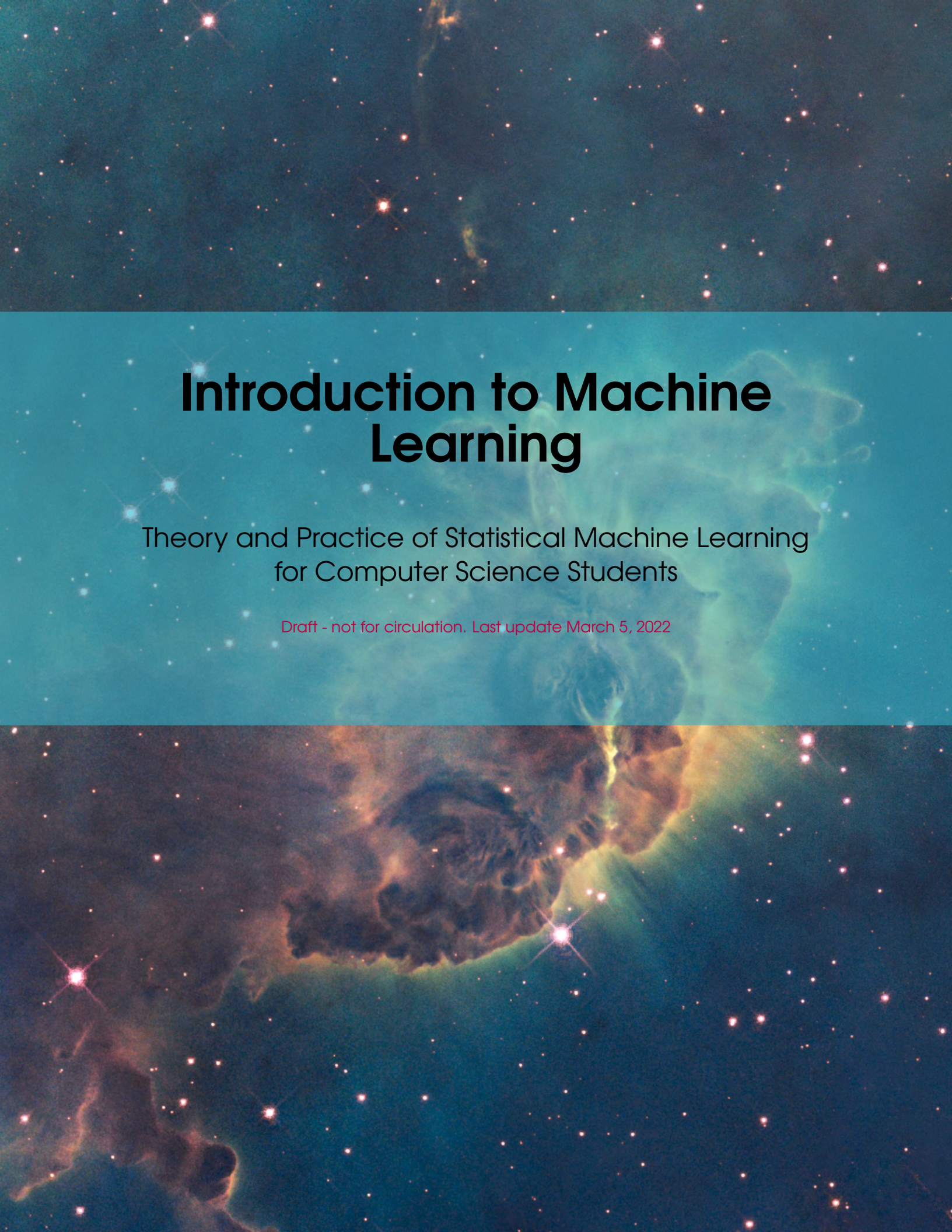


Introduction to Machine Learning

Theory and Practice of Statistical Machine Learning
for Computer Science Students

Draft - not for circulation. Last update March 5, 2022

INTRODUCTION TO MACHINE LEARNING

Matan Gavish and Gilad Green

Contents

1. Estimation Theory

"All models are wrong, but some are useful" — George E.P. Box, in *Science and Statistics*, 1976

In the study and practice of machine learning we try and define models in attempt to explain different observed phenomena, and into which we pour our understanding of the problem. These models, sometimes incorporating a probabilistic aspect, often depend on unknown parameters, whose values are to be inferred from given data.

Suppose for example we would like to know the current time. We do not have a means of telling the time ourselves, and so we ask people around us. We asked an individual and were told that the time is 13:15. Since people tend to approximate the time, it is possible that the true time is not 13:15 as told but rather 13:13. To get a better accurate assessment of the true time we decide to ask multiple individuals and record their answers. We organize the answers as *observations* denoted by $x_1,\ldots,x_m$, for m the number of individuals asked. Intuitively, we can now *estimate/approximate* the true time by averaging over the observations $\frac{1}{m}\sum x_i$.

It is helpful to consider the observations $x_1,\ldots,x_m$ as the *realization* of the *random variables* $X_1,\ldots,X_m$. These are called a *repeated sample* or simply a *sample*. By doing so, we emphasis the probabilistic nature of the problem and can devise a simplified probabilistic model for it. This model, will capture the essence of the problem by incorporating our prior knowledge and assumptions on the manner in which the observations behave (i.e. the underlying probability distribution).

Given a sample $x_1,\ldots,x_m$ for which we assume some underlying probability distribution $\mathcal{P}$ we say that $x_1,\ldots,x_m$ are *identically distributed* if they are all the realizations of random variables over the same probability distribution function, i.e. $X_1,\ldots,X_m \sim \mathcal{P}$. We further say that $x_1,\ldots,x_m$ are independent identically distributed (i.i.d) if they are all the realizations of random variables which are independent random variables and identically distributed. We denote this as $X_1,\ldots,X_m \overset{i.i.d}{\sim} \mathcal{P}$ for $X_1 = x_1,\ldots,X_m = x_m$ the realizations of $X_1,\ldots,X_m$. To simplify notation we often write $x_1,\ldots,x_m \overset{i.i.d}{\sim} \mathcal{P}$ (even though $x_1,\ldots,x_m$ are not random variables) in the sense that the corresponding random variables are i.i.d.

Probability versus Statistics

Throughout this chapter, and the rest of this book, we will be using concepts from both probability theory and from statistics. Though both are related fields of mathematics dealing with analyzing the chances of evens happening, there are some key fundamental differences in the manner in which they address the matter.

The field of probability is first and foremost a theoretical study, asking the outcome of mathematical definitions. It deals with *predicting* how likely are events to occur in a given setup. Statistics on the other hand, is primarily an applied study, that attempts to explain observed phenomena in the real world. It involves the *analysis of past* events.

In the context of the time estimation problem above

- If we are to think as a probabilist, we would notice the answers take a wide continuous range of values. We would then *assume* they follow some distribution, say Gaussian, and ask what is the probability of predicting a specific time.
- If we are to think as a statistician, though noticing the answers take a wide continuous range of values, we would be concerned with the question of "how do we assure these observations follow a Gaussian distribution?". We would then use these past observations to decide if they are indeed consistent with the Gaussian distribution assumption.

We will use both fields to tackle the different learning problems we will face.

1.1 Estimation of Distribution Parameters

Often the underlying probability distribution $\mathcal{P}$ is characterized by some set of parameters $\boldsymbol{\theta} \in \Theta$. $\boldsymbol{\theta}$ denotes a vector of parameters and Θ the set with all possible values for the paramerers. For example, a Poisson distribution $Poisson(\lambda)$ is characterized by the parameter λ, so $\boldsymbol{\theta} := \{\lambda\}$ and $\Theta := \mathbb{N}_0$. In the case of a Normal distribution $\mathcal{N}(\mu, \sigma^2)$, it is characterized by μ, σ^2, so $\boldsymbol{\theta} := \{\mu, \sigma^2\}$ and $\Theta := \mathbb{R} \times \mathbb{R}_+$.

In the case of (parametric) estimation theory, we assume some underlying probability distribution $\mathcal{P}(\boldsymbol{\theta})$ characterized by the parameters $\boldsymbol{\theta}$. Then, given a sample $x_1, \ldots, x_m$, drawn according to $\mathcal{P}(\boldsymbol{\theta})$ and for which we *do not* know the true value of $\boldsymbol{\theta}$, we wish to choose $\boldsymbol{\theta}^*$ "best" expressing the true value of $\boldsymbol{\theta}$. We formulate this problem by defining a decision function (or decision rule) $\delta_m : \mathbb{F}^m \to \Theta$ mapping the observations to the parameter space. For simplicity we shall omit the index and write $\delta(x_1, \ldots, x_m)$. Since we can devise many different decision functions, the problem of choosing $\boldsymbol{\theta}^* \in \Theta$ becomes a problem of choosing the *optimal* decision function δ from the set Δ:

$$\Delta := \{\delta(x_1, \ldots, x_m) : \mathbb{F}^m \to \Theta\} \tag{1.1}$$

where the notion of what does an optimal decision function means will be covered later. In the context of machine learning we refer to this set Δ by the term *hypothesis class* and each function in it as an *hypothesis*.

Definition 1.1.1 Let $\delta \in \Delta$ be a decision function. Then $\delta(X_1, \ldots, X_m)$ is called a point statistical estimator of a parameter $\boldsymbol{\theta}$, or simply an estimator.

Let us return to the example above of determining the current time. Given the answers of m different individuals $x_1, \ldots, x_m$ we stated that we could simply average the answers and approximate the true time as $\frac{1}{m}\sum x_i$. In fact, the function $f : \mathbb{R}^m \to \mathbb{R}$ defined by $f(x_1, \ldots, x_m) = \frac{1}{m}\sum x_i$ *is* a decision rule, and when using it over the random variables $X_1, \ldots, X_m$ (whose realizations are $x_1, \ldots, x_m$) we in fact defined an estimator.

1.1.1 Estimation of Univariate Gaussian Parameter

Recall the probability density function of the normal distribution.

Definition 1.1.2 — (Univariate) Normal Distribution. A random variable x has a ***normal distribution*** with expectation μ and variance σ^2 if it has a PDF of the form:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

In this case we write: $x \sim \mathcal{N}(\mu, \sigma^2)$

Consider the problem of estimating the expected value μ and variance σ^2 for a given sample drawn, i.i.d from a normal distribution characterized by μ, σ^2: $x_1, \ldots, x_m \overset{i.i.d}{\sim} \mathcal{N}\left(\mu, \sigma^2\right)$. To do so we should define decision rules that take the m observations and output an estimation of the expected value and variance. We define the estimators:

- ***Sample mean***: an estimator for the population mean

$$\hat{\mu}_X := \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1.2}$$

- ***Sample variance***: an estimator for the population variance

$$\hat{\sigma}_X^2 := \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \hat{\mu}_X)^2 \tag{1.3}$$

where the $\hat{}$ (hat) notation expressed that $\hat{\mu}_X, \hat{\sigma}_X^2$ are estimators of the true parameters μ, σ^2. Whenever context is clear, we will omit the subscript X.

R Note that the sample variance is somewhat different from the variance of the x_i's where we divide by $m-1$ rather than by m. The factor by which we divide (m or $m-1$) is determined by what is known as the number of *degrees of freedom*. Though beyond the scope of this book, an estimator's number of degrees of freedom is the number of independent observations used for estimation minus the number of parameters used for the estimation. In the case of the sample variance estimator (1.3) we provide m samples but use one parameter (being the estimator of the sample mean). Thus, we divide by $m-1$. The difference between the two options will be discussed shortly under bias of estimators (Definition 1.1.6).

1.1.2 Properties of Estimators

From the definition of the sample variance (1.3) we understand that we have a great deal of freedom when designing different estimators. Indeed, we might also consider the following functions are estimators of the sample variance:

$$\hat{\sigma} := \frac{1}{m}\sum(x_i - \hat{\mu}), \quad \hat{\sigma} := \frac{1}{m}\sum|x_i - \hat{\mu}|, \quad \hat{\sigma} := \frac{1}{m-1}\sum|x_i - \hat{\mu}|$$

Therefore, the question is how to choose the "right"/"best" estimator, what does it actually mean to be the "right"/"best" estimator, and does this decision depend on the given problem? Consider for example the following scenario. Suppose $x_1, \ldots, x_m \overset{i.i.d}{\sim} \mathcal{N}(\mu, 1)$ and the two following estimators for the expectation:

$$\hat{\mu}_1(x_1, \ldots, x_m) = \tfrac{1}{m}\sum x_i, \qquad \hat{\mu}_2(x_1, \ldots, x_m) = 1$$

Clearly, choosing $\hat{\mu}_2$ does not seem like a smart decision as for any true value of the distribution's expectation $\mu \neq 1$ our estimation will probably be wrong. However, if the true value is indeed $\mu = 1$ then the sample mean estimator $\hat{\mu}_1$ will be out-performed by $\hat{\mu}_2$.

And so, to try and answer the questions above we need to devise ways to evaluate estimators. To do so we define two types of objects:

- Properties of estimators - these will be desired properties that an estimator might fulfill, and that will help us choose an estimator.
- A loss function - A measure to compare between the estimated values obtained by the estimator and the true value of the parameters.

Then, using these objects we can define in what sense is one estimator better than another. Usually this will be done in the form of finding an estimator that minimizes some loss function out of the set of estimators fulfilling some property.

Throughout this book we will encounter different examples where we will switch from using one estimator with another depending on what we are trying to achieve.

1.1.2.1 Loss Functions

Definition 1.1.3 A *loss function* is a function that maps an event onto a real number: $L : \Omega \to \mathbb{R}$

Intuitively, a loss function represents the *cost* associated with the event. In the context of decision theory such an event will be the estimation outputted by an estimator and we will often want to find the estimator minimizing this cost. Here are a few examples for different loss functions that we will encounter throughout the book:

- The ℓ_{0-1} loss function defined as: $L(\hat{\theta},\theta) := \mathbb{1}_{\hat{\theta}=\theta}$.
- The ℓ_ε loss function defined as: $L_\varepsilon(\hat{\theta},\theta) := \mathbb{1}_{||\hat{\theta}-\theta||\leq\varepsilon}$ for some predetermined norm and $\varepsilon \geq 0$.
- The absolute-value (ℓ_1) loss function defined as: $L(\hat{\theta},\theta) := \left|\hat{\theta}-\theta\right|$.
- The quadratic or squared-error (ℓ_2) loss function defined as: $L(\hat{\theta},\theta) := \left(\hat{\theta}-\theta\right)^2$.

For different scenarios different loss functions are used. For example, in the case of the estimating the current time, perhaps estimating the exact time is less important than being close to it. If that is the case, we might prefer using the absolute-value or squared-error loss functions. Or perhaps we are not concerned with how far is our estimation as long as it is within a certain deviance from the actual time. In such case we might prefer using the ℓ_ε loss function.

Another term associated with the notion of loss functions is the *risk function* of a decision rule.

Definition 1.1.4 Let X be a set of observations and $\delta(X)$ a decision rule for parameter $\boldsymbol{\theta}$. The risk function of δ and $\boldsymbol{\theta}$ with respect to a loss function L is defined as the expected loss:

$$\mathcal{R}(\boldsymbol{\theta},\delta) := \mathbb{E}_X\left[L(\theta,\delta(X))\right]$$

One such risk function, which we will see in chapter 2, is the Mean Squared Error (MSE) risk function. This is the risk function with respect to the squared-error loss function $\mathcal{R}(\boldsymbol{\theta},\hat{\boldsymbol{\theta}}) := \mathbb{E}_X[(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}}(X))^2]$.

1.1.2.2 Bias, Variance & Consistency

Unbiasedness

One of the most desirable properties of an estimator is to be unbiased. That is, that on average our estimation equals to the true parameter estimated. Formally,

Definition 1.1.5 Let $\delta(x_1,\ldots,x_m)$ be an estimator for a parameter θ. The difference $d = \delta(x_1,\ldots,x_m) - \theta$ is called the *error* of the estimator δ.

Definition 1.1.6 Let $\delta(x_1,\ldots,x_m)$ be an estimator for a parameter θ. The quantity

$$Bias_\theta\left[\delta(x_1,\ldots,x_m)\right] := \mathbb{E}_{x_1,\ldots,x_m|\theta}\left[d\right] = \mathbb{E}_{x_1,\ldots,x_m|\theta}\left[\delta(x_1,\ldots,x_m)-\theta\right]$$

for $x_1,\ldots,x_m|\theta$ denoting the probability of sample $x_1\ldots,x_m$ from $\mathcal{P}(\theta)$, is called the *bias* (or systematic error) of the estimator δ.

Definition 1.1.7 Let $\delta(x_1,\ldots,x_m)$ be an estimator for a parameter θ. δ is said to be *unbiased* if

$$\forall\theta\in\Theta \quad Bias_\theta\left[\delta(x_1,\ldots,x_m)\right]=0 \text{ or equivelantly } \forall\theta\in\Theta \quad \mathbb{E}_{x_1\ldots,x_m|\theta}\left[\delta(x_1,\ldots,x_m)\right]=\theta$$

Let us revisit the previously defined sample mean (1.2) and sample variance (1.3) estimators and show that these are both unbiased estimators. Beginning with the sample mean estimator then

$$\mathbb{E}(\hat{\mu})=\mathbb{E}\left(\frac{1}{m}\sum_{i=1}^{m}x_i\right)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}(x_i)=\mathbb{E}(X)\frac{1}{m}\sum_{i=1}^{m}1=\mathbb{E}(X)=\mu \tag{1.4}$$

where we used the linearity property of the expectation and that the samples are i.i.d. Similarly we can show that the sample variance is unbiased:

$$\begin{aligned}\mathbb{E}\left(\hat{\sigma}^2\right) &= \frac{1}{m-1}\sum_{i=1}^{m}\mathbb{E}\left((x_i-\hat{\mu})^2\right)\\ &= \frac{1}{m-1}\sum_{i=1}^{m}\mathbb{E}\left(x_i^2-2x_i\frac{1}{m}\sum_{j=1}^{m}x_j+\frac{1}{m^2}\sum_{k,j=1}^{m}x_kx_j\right)\end{aligned}$$

The X_i's are i.i.d which implies that $\mathbb{E}(x_i)=\mathbb{E}(X)$ and $\mathbb{E}\left(x_i^2\right)=\mathbb{E}\left(X^2\right)$ for every i, as well as that $\mathbb{E}(x_kx_j)=\mathbb{E}^2(X)=\mu_X^2$ for every $k\neq j$. Substituting into the above sums we obtain that:

$$\mathbb{E}\left(\hat{\sigma}^2\right)=\mathbb{E}\left(X^2\right)-\mathbb{E}^2(X)=Var(X)=\sigma^2$$

In a similar manner we can show that the estimator of sample variance where we divide by m instead of $m-1$ is a *biased* estimator. Ex.2

Variance

Another useful property of an estimator, that provides us with an indication of how well is the estimator performing, is its variance. Since an estimator is a function of (a set of) random variables, it itself is a random variable. Therefore, the variance of an estimator is simply the variance of a random variable, where the probability space is defined over the events of obtaining a given set of samples.

Definition 1.1.8 Let $\delta(x_1,\ldots,x_m)$ be an estimator for a parameter θ. The *variance* of δ is defined as

$$Var(\delta):=\mathbb{E}_{x_1,\ldots,x_m|\theta}\left[\left(\delta(x_1,\ldots,x_m)-\mathbb{E}_{x_1,\ldots,x_m|\theta}\left[\delta(x_1,\ldots,x_m)\right]\right)^2\right]$$

where expectation is taken over the even of sampling $x_1,\ldots,x_m$ from $\mathcal{P}(\theta)$.

Let us compute the variance of the sample mean estimator (1.2). Let $x_1,\ldots,x_m$ be a set of independent random variables with identical variance σ^2. As the sample mean estimator is the mean of m random variables we could simply:

$$\begin{aligned}Var(\hat{\mu}) &= Var\left(\frac{1}{m}\sum x_i\right)\\ &= \frac{1}{m^2}Var\left(\sum x_i\right)\\ &\overset{(*)}{=} \frac{1}{m^2}\sum Var(x_i)\\ &\overset{(**)}{=} \frac{1}{m^2}\cdot m\cdot\sigma^2\\ &= \frac{\sigma^2}{m}\end{aligned}$$

where we used the assumption that the samples are independent $(*)$ and with identical variance $(**)$.

Consistency

Going back to the task of determining a given time where we defined our estimator as the sample mean, it seems intuitive that the more individuals asked the more accurate (i.e closer to the true value) our estimation should be. Let us formulate this as a property an estimator might fulfill.

Definition 1.1.9 A sequence $\{Z_n\}$ of random variables is said to *converge in probability* to a random variable Z if for any $\varepsilon > 0$

$$\lim_{n\to\infty} \mathbb{P}(|Z_n - Z| > \varepsilon) = 0$$

and is denoted by $Z_n \xrightarrow{p} Z$.

Definition 1.1.10 An estimator $\theta_n(x_1,\ldots,x_n)$ of parameter θ is said to be *consistent* if it converges in probability to the true value of θ:

$$\forall \varepsilon > 0 \ \lim_{n\to\infty} \mathbb{P}(|\theta_n - \theta| > \varepsilon) = 0$$

Let us understand these definitions as it is to be applied for an estimator. Since an estimator is a function of the random variables $x_1,\ldots,x_m \sim \mathcal{P}$ it is itself a random variable. Consider a set of estimators (random variables) $\{\hat{\theta}_n\}_{n=1}^{\mathbb{N}}$, each calculated over the first n samples. In addition, denote θ the constant random variable valuing as the true parameter estimated. Therefore, $\theta_n \xrightarrow{p} \theta$ means that as we increase the sample size n the probability of our estimation deviating from the true parameter by more than a fixed amount ε tends to zero. It can be shown that the sample mean has the property of being a consistent estimator.

In this section we have discussed the first and second moments of an estimator (first moment in the form of an estimator's bias). Being a random variable, we can also discuss the distribution of a random variable, as seen shortly in Lab 01 - Data Simulation and Sampling.

1.1.3 The Maximum Likelihood Estimator

And so, we can now define a criteria by which we define what does it mean for an estimator to be "optimal". For example, for a set of estimators Δ, we can decide that the optimal estimator is one that is *unbiased* and that achieves the minimal variance out of all estimators in Δ. This approach is known as the Minimum Variance Unbiased (MVU). This is a logical decision. Unbiased estimators are right "on average" and having a small as possible variance means we are often not far from this average. Notice also that by this definition there could be more than a single "optimal" estimator in Δ.

Another approach is to look for the *Maximum Likelihood Estimator*. This approach suggests choosing the estimator under which the observed data was *most likely*. Suppose we obtained the observations $x_1,\ldots,x_m \overset{i.i.d}{\sim} \mathcal{P}(\boldsymbol{\theta})$. For the probability distribution function $\mathcal{P}$ we define the likelihood function.

Definition 1.1.11 — Likelihood. Let X be a random variable following some probability distribution $\mathcal{P}$ with a density function f that depends on a parameter $\boldsymbol{\theta} \in \Theta$. The *likelihood* function is

$$\mathcal{L}(\boldsymbol{\theta}|x) := f_{\boldsymbol{\theta}}(x)$$

for x the realization of X.

For example, consider the case of the univariate Gaussian distibution seen above (1.1.2). Then, for $\boldsymbol{\theta} = \{\mu, \sigma^2\}$

the likelihood function of $\mathcal{P} = \mathcal{N}(\mu, \sigma^2)$ is:

$$\mathcal{L}(\boldsymbol{\theta}|x_i) = f_{\boldsymbol{\theta}}(x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right), \quad i \in [m] \tag{1.5}$$

where, for the sample $x_1, \ldots, x_m$ where the samples are drawn i.i.d the likelihood is:

$$\begin{array}{rcl} \mathcal{L}(\boldsymbol{\theta}|x_1,\ldots,x_m) & = & f_{\boldsymbol{\theta}}(x_1,\ldots,x_m) \\ & = & \prod_{i=1}^{m} f_{\boldsymbol{\theta}}(x_i) \\ & = & \frac{1}{\left(2\pi\sigma^2\right)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum(x_i-\mu)^2\right) \end{array} \tag{1.6}$$

Using the likelihood function derived for the univariate Gaussian distribution we can now provide each $\boldsymbol{\theta}$ with a quantity (the likelihood) of how likely is it to have generated the observed data.

■ **Example 1.1** Let $x_1, x_2, x_3, x_4 \overset{i.i.d}{\sim} \mathcal{N}(\mu, \sigma^2)$ where $x_1 = -1, x_2 = 0, x_3 = 0, x_4 = 1$ and $\sigma^2 = 1$. Let us calculate the likelihood for:

- $\mu = 0$:

$$\begin{array}{rcl} \mathcal{L}(\mu=0,\sigma^2=1|x_1,x_2,x_3,x_4) & = & \frac{1}{(2\pi)^2}\exp(-\frac{1}{2}\sum_{i=1}^{4}x_i^2) \\ & = & \frac{1}{4\pi^2}\exp(-\frac{2}{2}) \\ & \approx & 0.00931 \end{array}$$

- $\mu = 1$:

$$\begin{array}{rcl} \mathcal{L}(\mu=1,\sigma^2=1|x_1,x_2,x_3,x_4) & = & \frac{1}{(2\pi)^2}\exp(-\frac{1}{2}\sum_{i=1}^{4}(x_i-1)^2) \\ & = & \frac{1}{4\pi^2}\exp(-\frac{2^2+1+1}{2}) \\ & \approx & 0.00126 \end{array}$$

■

And so, given the likelihood function we define the Maximum Likelihood Estimator (MLE) as $\boldsymbol{\theta} \in \Theta$ that maximizes the likelihood function. Formally:

Definition 1.1.12 Let $\mathcal{L}$ be the likelihood function of some probability distribution $\mathcal{P}$ characterized by $\boldsymbol{\theta} \in \Theta$. Let $X \sim \mathcal{P}(\boldsymbol{\theta})$ be a random variable and x its realization. The *Maximum Likelihood Estimator* (MLE) for $\boldsymbol{\theta}$ is

$$\hat{\boldsymbol{\theta}}^{MLE} := \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmax}}\ \mathcal{L}(\boldsymbol{\theta}|x)$$

Let us derive the MLE for the expectation of a univariate Gaussian distribution when σ^2 is known. Let $x_1, \ldots, x_m \overset{i.i.d}{\sim} \mathcal{N}(\mu, \sigma^2)$. So:

$$\hat{\mu}^{MLE} = \underset{\mu\in\mathbb{R}}{\operatorname{argmax}}\ \mathcal{L}(\mu|x_1,\ldots,x_m,\sigma^2)$$

Since the logarithm function is a monotonous increasing transformation the maximizer of the likelihood

function is also the maximizer of the *log-likelihood*. Thus:

$$\begin{aligned}\hat{\mu}^{MLE} &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ \mathcal{L}(\mu|x_1,\ldots,x_m,\sigma^2)\\ &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ \log\mathcal{L}(\mu|x_1,\ldots,x_m,\sigma^2)\\ &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ \log\left(\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right)\\ &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ \log\left(\frac{1}{(2\pi\sigma^2)^{m/2}}\exp\left(-\frac{1}{2\sigma^2}\sum(x_i-\mu)^2\right)\right)\\ &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ \log\left(\exp\left(-\frac{1}{2\sigma^2}\sum(x_i-\mu)^2\right)\right)\\ &= \underset{\mu\in\mathbb{R}}{\text{argmax}}\ -\sum_{i=1}^{m}(x_i-\mu)^2\end{aligned} \tag{1.7}$$

And so, to find the maximizer, we calculate the derivative with respect to μ and equate to zero:

$$\begin{gathered}\frac{\partial}{\partial\mu}\left(-\sum_{i=1}^{m}(x_i-\mu)^2\right) = -\sum_{i=1}^{m}\frac{\partial(x_i-\mu)^2}{\partial\mu} = \sum_{i=1}^{m}2(x_i-\mu) = 0\\ \Downarrow\\ \hat{\mu}^{MLE} = \frac{1}{m}\sum_{i=1}^{m}x_i\end{gathered} \tag{1.8}$$

Notice that when we previously used the sample mean estimator (1.2) we have therefore actually used the maximum likelihood estimator for the expectation. Similarly, we can derive that the sample variance defined above is the MLE of the variance.

1.1.3.1 Beyond Maximum Likelihood - A Bayesian Approach

Suppose we are facing the task of estimating average human height. We observe $x_1,\ldots,x_m$ a set of i.i.d samples reflecting the heights of m individuals. In order to estimate the average human height $\hat{\theta}$, we must first specify a principle by which we select an estimator.

In the section above we defined the likelihood function $\mathcal{L}(\boldsymbol{\theta}|x)$ and the strategy of selecting the estimator that maximizes the likelihood function - the MLE. Let us assume that $x|\boldsymbol{\theta}$ follows a normal distribution. If our sample is $x_1 = 2m, x_2 = 2.03m, x_3 = 1.95m, x_4 = 2.1m, x_5 = 1.65m$ then the MLE is the sample mean: $\hat{\theta}^{MLE} = 1.94m$. This however seems to be an odd estimation. Even without looking at our observations it seems logical that over the general population the average human height should probably be lower than 1.9m and perhaps higher than 1.5m. This seems logical as we have some additional knowledge (a prior belief) on the problem. Therefore, the question is how could we include this additional knowledge in our estimation?

Let us go back to the likelihood function. We defined it as the PDF of an observation x under a probability distribution characterized by $\boldsymbol{\theta}$. We interpret $\mathcal{L}(\boldsymbol{\theta}|x)$ as "how likely is $\boldsymbol{\theta}$ "given" the observation x". The term "given" is itself interpreted as "while fixing the observation x" and not in the sense of probability theory. We could however further *assume* that both X and $\boldsymbol{\theta}$ are random variables which have some *joint probability distribution* function $f_{X,\Theta}(x,\boldsymbol{\theta})$. By doing so we can derive a new criteria for selecting an estimator. Under such assumption of a joint probability distribution $f_{X,\Theta}(x,\boldsymbol{\theta})$ the expression $x|\boldsymbol{\theta}$ can be understood through the conditional distribution:

$$f_{X|\Theta=\boldsymbol{\theta}}(x) = \frac{f_{X,\Theta}(x,\boldsymbol{\theta})}{f_{\Theta}(\boldsymbol{\theta})} \tag{1.9}$$

Same as before, we can choose the MLE as $\hat{\boldsymbol{\theta}}^{MLE}$ that maximizes $f_{X|\Theta=\boldsymbol{\theta}}(x)$. Now, by applying Bayes' theorem of conditional distributions we can express the conditional $\Theta|X$ as:

$$\overbrace{f_{\Theta|X=x}(\boldsymbol{\theta})}^{posterior} = \frac{\overbrace{f_{X|\Theta=\boldsymbol{\theta}}(x)}^{likelihood}\cdot\overbrace{f_{\Theta}(\boldsymbol{\theta})}^{prior}}{\underbrace{f_X(x)}_{evidence}} \tag{1.10}$$

where $f_{X|\Theta=\boldsymbol{\theta}}(x)$ is the likelihood function, $f_\Theta(\boldsymbol{\theta})$ is the marginal distribution of Θ and $f_X(x)$ the marginal distribution of X functioning as a normalization factor. The marginal f_Θ reflects our *belief* regarding the true value of $\boldsymbol{\theta}$ *before* (i.e *prior*) observing the data. Therefore, the *A-Posteriori* distribution $f_{\Theta|X=x}$, i.e the conditional of $\boldsymbol{\theta}$ *after* observing the data, is the weighting of the likelihood function by our prior belief and the evidence of the data.

By using the A-Posteriori distribution, we can derive the Maximum A-Posteriori estimator (MAP):

$$\hat{\boldsymbol{\theta}}^{MAP} := \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmax}}\, f_{\Theta|X=x}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmax}}\, \frac{f_{X|\Theta=\boldsymbol{\theta}}(x)\cdot f_\Theta(\boldsymbol{\theta})}{f_X(x)} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmax}}\, f_{X|\Theta=\boldsymbol{\theta}}(x)\cdot f_\Theta(\boldsymbol{\theta}) \tag{1.11}$$

Namely, the estimator that while takes into account both what is observed (the likelihood) and the prior belief.

Returning to the task of estimating human heights, let us further assume that the average human height follows a normal distribution centered around 1.7m with a variance of 0.01m. Therefore we assume that:

$$\theta \sim \mathcal{N}(1.7, 0.01) \quad (1)$$
$$x_i|\theta \overset{i.i.d}{\sim} \mathcal{N}(\theta, \sigma^2)\ \forall i \quad (2)$$

where we set σ^2 as $(\hat{\sigma}^2)^{MLE} \approx 0.17$. Under these assumptions and the sample above $x_1 = 2m, x_2 = 2.03m, x_3 = 1.95m, x_4 = 2.1m, x_5 = 1.65m$ we can now evaluate different values of θ:

- For $\theta = 1.94$ (the likelihood maximizer) we find that:

$$\log\left(f_{X|\theta=1.94,\sigma^2=0.17}(x_1,\ldots,x_5)\cdot f_\Theta(1.94)\right) = \log\left(f_\Theta(1.94)\cdot\prod_i f_{\theta=1.94,\sigma^2=0.17}(x_i)\right) = 1.264$$

- For $\theta = 1.7$ (the prior's expectation) we find that:

$$\log\left(f_{X|\theta=1.7,\sigma^2=0.17}(x_1,\ldots,x_5)\cdot f_\Theta(1.94)\right) = \log\left(f_\Theta(1.7)\cdot\prod_i f_{\theta=1.7,\sigma^2=0.17}(x_i)\right) = 1.442$$

Thus, under this prior we see that the MLE does not achieve the maximal posterior probability.

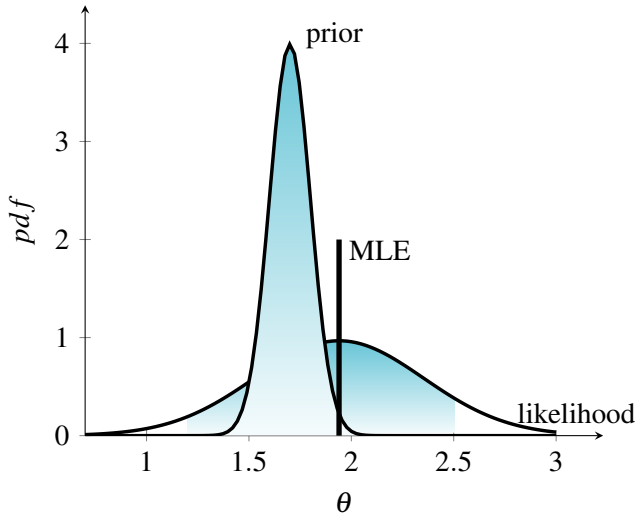


Figure 1.1: *Likelihood and prior distributions for human heights example*

R In general, the field of statistical learning splits between these two approaches: frequentistic methods which aim to express and maximize the likelihood function, and Bayesian methods which assume some prior distribution and derive estimators that use both the likelihood and the prior (one of these estimators is the MAP estimator seen above). It can be shown that when we correctly assume a prior distribution the Bayesian estimators are superior to the MLE. The challenge however with the Bayesian methods lies precisely in how to obtain such (estimation of ?) prior. Though this book focuses on frequentistic methods, we shortly discuss some Bayesian methods in section 3.5.

1.1.4 Measures of Concentration For Estimation Tasks

Occasionally, providing an estimation for some parameter of interest is not sufficient, and we are further interested in providing some bounds describing how "good"/"close" is our estimation to the true value. Consider for example the following scenario. Suppose we have a coin for which we want to estimate its bias, that is with what probability p flipping the coin will result with "Heads" being up and with probability $1-p$ "Tails" being up. Formally, we can think (i.e. model) a single coin flip as a Bernoulli random variable X which takes the value of 1 for "Heads" or 0 for "Tails".

We shall denote the probability distribution of X by $\mathcal{D}_p$ such that:

$$\mathcal{D}_p(X) = \begin{cases} p & X = 1 \\ 1-p & X = 0 \end{cases}, \quad p \in [0,1] \tag{1.12}$$

To estimate the value of p we take a sequence of m tosses of the coin $X_1, \ldots, X_m \overset{i.i.d}{\sim} Ber(p)$ and denote the results of the tosses (i.e. the samples) by $S = \{x_1, \ldots, x_m\}$ where x_i refers to the result of the random variable X_i. Lastly, we denote the probability distribution of the m coin tosses by $\mathcal{D}_p^m$ and therefore the probability of obtaining a specific sample S by $\mathcal{D}_p^m(S)$. To simplify notation we will omit the writing of p and write $\mathcal{D}^m(S)$.

Now, given such sample we would like to devise a *learning algorithm*, $\mathcal{A}$ to estimate/predict the true value of p. So, a coin prediction learning algorithm is a procedure which takes as input a sample $S = \{x_1, \ldots, x_m\}$, drawn according to $\mathcal{D}^m$, and outputs estimation of p. This estimation is denoted by $\mathcal{A}(S)$ or $\hat{p}(S)$ or simply $\hat{p}$. A straightforward strategy (i.e. algorithm) is to estimate p simply by calculating the empirical proportion of heads (ones):

$$\hat{p}(S) = \frac{1}{m}\sum x_i \tag{1.13}$$

It can be shown that this estimator is the MLE of the problem. Notice, that as we have shown earlier, the empirical mean in an unbiased estimator for the expectation (1.4). Namely, if we sample many set of samples $S_i = \left\{x_1^{(i)}, \ldots, x_m^{(i)}\right\}$, for each run our algorithm and output $\hat{p}_i$, then calculate the expected value over the p_is, then the result is p. Formally, $\mathbb{E}_S[\hat{p}(S)] = p$. This clearly seems like a desirable properly for our algorithm. However, since we seldomly have many different sets of samples and as a given sample S is finite, we do not expect our estimation of $\hat{p}$ to be exact. Instead we acknowledge that the algorithm might yield a $\hat{p}$ which satisfies that $|\hat{p} - p| \leq \varepsilon$, for some $\varepsilon \in (0,1)$ which is called the *accuracy* parameter. That is, our algorithm might not return exactly p, but it returns a result $\hat{p}$ which is "close enough".

Even then, unless p equals 1 or 0, there is always *some* chance that the drawn sample would be highly non-representative. We might for example, though unlikely, end up with a sample of all "Heads" or "Tails" regardless to the true value of p. So it is impossible to obtain a guarantee that $|\hat{p} - p| \leq \varepsilon$ holds with absolute certainty. Hence, we introduce a *confidence* parameter $\delta \in (0,1)$, and require that the event $\{S : |\hat{p} - p| > \varepsilon\}$ occurs with a probability of at most δ. In other words, we require our algorithm to be such that the probability of flipping the coin m times and obtaining a sequence S that causes it to produce an inaccurate estimation, $\hat{p}(S)$ ('inaccurate' meaning $|\hat{p} - p| > \varepsilon$) is smaller or equal to δ.

Intuitively, the larger the number of flips, the more information we have about the coin and the better chance

we have to satisfy the accuracy and confidence requirements. Since the accuracy and confidence parameters ε and δ are fixed, there should be some finite number of flips $m_{\mathcal{A}}$ (which depends on ε, δ and $\mathcal{A}$) such that for any sample of size $m \geq m_{\mathcal{A}}$ our algorithm satisfies the above accuracy and confidence requirements. The function that given ε, δ returns $m_{\mathcal{A}}$ such that that accuracy and confidence requirements are met is called the *sample complexity* function.

Measure Concentration Using Markov's Inequality

And so, we would like to ask "How good is our learning algorithm"? We already know that on average our learning algorithm will output the correct answer. In addition, as the empirical mean is a consistent estimator we know that the estimation will become more accurate as the number of samples m increases. What we do not know is how many samples do we need and can we somehow bound the probability of being inaccurate (i.e. the confidence) by some function depending on the number of samples. To do so we begin with using the most basic measure of concentration by applying Markov's inequality **??**.

Notice that $|\hat{p} - p|$ is a non-negative random variable. Therefore, by fixing some accuracy level $\varepsilon \in (0,1)$ we can bound from above the probability of $\hat{p}$ deviating from p by more than ε:

$$\mathcal{D}^m \left[|\hat{p} - p| \geq \varepsilon \right] \leq \frac{\mathbb{E}\left[|\hat{p} - p| \right]}{\varepsilon}$$

where the expectation can be bounded from above by:

$$\begin{array}{rcccl} \mathbb{E}^2\left[|\hat{p} - p| \right] & \leq & \mathbb{E}\left[|\hat{p} - p|^2 \right] & = & \mathbb{E}\left[\hat{p}^2 \right] - p^2 \\ & = & Var\left(\hat{p} \right) & = & \frac{1}{m^2} Var\left(\sum_i x_i \right) \\ & = & \frac{p(1-p)}{m^2} & \leq & \frac{1}{4m^2} \end{array}$$
$$\Downarrow$$
$$\mathbb{E}\left[|\hat{p} - p| \right] \leq \frac{1}{\sqrt{4m}}$$

where we used the definition of variance $Var(A) = \mathbb{E}\left[A^2 \right] - \mathbb{E}^2\left[A \right]$. Put together:

$$\mathcal{D}^m \left[|\hat{p} - p| \geq \varepsilon \right] \leq \frac{\mathbb{E}\left[|\hat{p} - p| \right]}{\varepsilon} \leq \frac{1}{\sqrt{4m\varepsilon^2}}$$

Thus we have obtained a bound on the confidence, for a given accuracy, as a function of the number of samples. We can now express the above as follows. If we select m to be $m \geq \left\lceil \frac{1}{4\varepsilon^2} \cdot \frac{1}{\delta^2} \right\rceil$ then the right-hand side is smaller or equals to δ. So, for any $\varepsilon, \delta \in (0,1)$, if we sample $m_{\mathcal{A}}(\varepsilon, \delta) \geq \left\lceil \frac{1}{4\varepsilon^2} \cdot \frac{1}{\delta^2} \right\rceil$ samples then this learning algorithm achieves:

$$\mathcal{D}_p^m \left[|\hat{p}(S) - p| \geq \varepsilon \right] \leq \delta$$

Namely, the algorithm is accurate enough (as specified by ε) with a high enough confidence (as specified by $1 - \delta$).

Measure Concentration Using Chebyshev's Inequality

We can improve the upper bound seen in (1.1.4) (i.e find a sample complexity function that will need less samples) by using Chebyshev's inequality.

Theorem 1.1.1 — Chebyshev's Inequality. Let X be a random variable with a finite mean $\mathbb{E}[X]$ and variance $Var(X)$. Then, for every $\varepsilon > 0$:

$$\mathbb{P}\left[|X - \mathbb{E}[X]| \geq \varepsilon \right] \leq \frac{Var(X)}{\varepsilon^2}$$

Proof. Consider the random variable $Y = (X - \mathbb{E}[X])^2$. This is a non-negative random variable and as such we can apply Markov's inequality over it. We obtain that $\mathbb{P}\left[(X - \mathbb{E}[X])^2 \geq \varepsilon^2\right] \leq \frac{Var(X)}{\varepsilon^2}$. To conclude the proof, observe that $\mathbb{P}[|X - \mathbb{E}[X]| \geq \varepsilon] = \mathbb{P}\left[(X - \mathbb{E}[X])^2 \geq \varepsilon^2\right]$. ■

So, given a sample $x_1, \ldots, x_m \overset{i.i.d}{\sim} Ber(p)$ we can now bound the deviance of our estimator $\hat{p}$ from its expected value as follows:

$$\begin{array}{lll} \mathbb{P}[|\hat{p} - \mathbb{E}[\hat{p}]| \geq \varepsilon] & \overset{unbiased}{=} & \mathbb{P}[|\hat{p} - p| \geq \varepsilon] \overset{Chebyshev}{\leq} \frac{Var(\hat{p})}{\varepsilon^2} \\ & = & \frac{1}{\varepsilon^2} Var\left(\frac{1}{m}\sum x_i\right) \overset{i.i.d}{=} \frac{1}{\varepsilon^2 \cdot m^2}\sum Var(x_i) \\ & = & \frac{p(1-p)}{m\varepsilon^2} \end{array}$$

For which, since the variance of a Bernoulli random variable is $p(1-p) \leq 1/4$, we can simply conclude that $\mathbb{P}[|\hat{p} - \mathbb{E}[\hat{p}]| \geq \varepsilon] \leq \frac{1}{4m\varepsilon^2}$. The bound obtained using Chebyshev's inequality tends to zero as $\frac{1}{m}$ while the one obtained from Markov's inequality tends to zero as $\frac{1}{\sqrt{m}}$. By solving for m $\delta = \frac{1}{4m\varepsilon^2}$ then we derive a tighter bound for the sample complexity $m_{\mathcal{A}}(\varepsilon, \delta) \leq \left\lceil \frac{1}{4\delta\varepsilon^2} \right\rceil$.

Measure Concentration Using Hoeffding's Inequality

A natural question which arises is whether the obtained bound is optimal (tight). Indeed, we can further improve the bound by exploiting the fact that our random variable not only has a finite variance, but it is also bounded between 0 and 1. For this end we use Hoeffding's inequality for the average of independent and bounded random variables.

Theorem 1.1.2 — Hoeffding's inequality. Let $X_1, \ldots X_m$ be independent and bounded random variables with $a_i \leq X_i \leq b_i$. Let $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$. Then,

$$\mathbb{P}\left[\left|\overline{X} - \mathbb{E}\left[\overline{X}\right]\right| \geq \varepsilon\right] \leq 2\exp\left(\frac{-2m^2\varepsilon^2}{\sum_{i=1}^m (b_i - a_i)^2}\right)$$

Corollary 1.1.3 Let $X_1, \ldots X_m$ be a sequence of m i.i.d random variables, each with an expectation value $\mathbb{E}[X]$ and all of which are bounded: $a \leq X_i \leq b$. Denote $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$ then:

$$\mathbb{P}\left[\left|\overline{X} - \mathbb{E}[X]\right| \geq \varepsilon\right] \leq 2\exp\left(\frac{-2m\varepsilon^2}{(b-a)^2}\right)$$

Proof. Notice that as $X_1, \ldots, X_m$ all share the same expectation and bounds then

$$\mathbb{E}\left[\overline{X}\right] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}[X] = \mathbb{E}[X], \quad \sum_{i=1}^m (b_i - a_i)^2 = m \cdot (b-a)^2$$

By placing these expressions in Lemma 1.1.2 we conclude the inequality. ■

By applying Hoeffding's inequality to the case of the coin prediction problem, then for a sample of size m, we obtain that $\mathcal{D}_p^m[|\hat{p} - p| \geq \varepsilon] \leq 2\exp\left(-2m\varepsilon^2\right)$. Therefore, if using Hoeffding's inequality we are able to get a bound which converges exponentially in m. By taking $m \geq \left\lceil \frac{1}{2\varepsilon^2} \cdot \log\left(\frac{2}{\delta}\right) \right\rceil$ samples we obtain that this probability is bound above by δ as required.

1.2 Multivariate Distributions

Up to now, we only dealt with random variables taking a single value. However, we often face a situation where an observation consists of multiple properties. As different properties may (or may not) influence one another we wish to define how the set of properties jointly behaves.

■ **Example 1.2** Consider the question of describing human weight and height. It is possible to model (i.e. describe how the data behaves) each of these properties independently using some distribution. Suppose $w_1,\ldots,w_m$ are the weights (in kilograms) of m individuals and $h_1\ldots,h_m$ are the heights (in centimeters) of the same individuals. Let us assume that the weights and heights each follow some normal distribution:

$$w_1,\ldots,w_m \overset{i.i.d}{\sim} \mathcal{N}(75,3)$$
$$h_1,\ldots,h_m \overset{i.i.d}{\sim} \mathcal{N}(170,5)$$

However, we know that the properties of weight and height are linked. There exists (in the data) a connection between the two such that, in general, the taller an individual is the higher the weight. As such, we would prefer to model the data using a *join distribution* that describes the pairs of weights and heights $(w_1,h_1),\ldots,(w_m,h_m)$.
■

Definition 1.2.1 — Joint distribution. Given random variables $X_1,\ldots,X_d$, that are defined on a probability space, the joint probability distribution for $X_1,\ldots,X_d$ is a probability distribution that gives the probability that each of $X_1,\ldots,X_d$ falls in any particular range (for continuous RVs) or discrete set (for discrete RVs) of values specified for that variable.

Definition 1.2.2 — Joint PDF of a random vector. Two random variables X_1 and X_2 are ***jointly continuous*** if there exists a non-negative function $f_{X_1,X_2}:\mathbb{R}^2\to\mathbb{R}$, such that, for any set $A\in\mathbb{R}^2$, it holds that

$$\mathcal{D}\left((X,Y)\in A\right)=\int_A f_{X_1,X_2}(x_1,x_2)\,dx_1dx_2$$

The function f_{X_1,X_2} is called the ***joint probability density function (JPDF)*** of X_1 and X_2. Often, if the context is clear, one omits the subscripts of f and simply writes $f(x_1,x_2)$.

Definition 1.2.3 — Random vector. A (column) ***random vector***: $X\coloneqq(X_1,\ldots,X_d)^\top$, is a finite collection of random variables, denoted $X_1,\ldots,X_d$, defined on a common probability space $(\Omega,\mathcal{D})$.

Returning to the task of describing a distribution of human weights and heights we can now denote the observations using random vectors $\mathbf{x}_1,\ldots,\mathbf{x}_m\in\mathbb{R}^d$ for $d=2$. We could then consider how does a specific feature manifest in the sample. For the i'th property $\mathbf{x}_1(i),\ldots,\mathbf{x}_m(i)$ are the realizations random variables $X_1^{(i)},\ldots,X_m^{(i)}$. Another form of writing $\mathbf{x}_1(i),\ldots,\mathbf{x}_m(i)$ is by using two indices, the first standing for the sample's index and the second for the property $x_{1,i},\ldots,x_{m,i}$.

When dealing with more than a single random variable we can ask how do the different variables jointly vary. For two jointly distributed real-valued random variables X,Y with finite second moments, the covariance between X and Y as $cov(X,Y)=\mathbb{E}[X-\mathbb{E}[X]]\,\mathbb{E}[Y-\mathbb{E}[Y]]$. The covariance between a random variable and itself is $cov(X,X)=\mathbb{E}[X-\mathbb{E}[X]]^2=Var(X)$. We arrange the covariances between d different random variables in the form of a *covariance matrix*

Definition 1.2.4 — Covariance Matrix. Let $X\coloneqq(X_1,\ldots,X_d)^\top$ be a random vector. The *Covariance Matrix* Σ is the $d\times d$ matrix whose (i,j) entry is the covariance $\Sigma_{ij}\coloneqq\sigma(X_i,X_j)$:

$$\Sigma := \begin{pmatrix} \mathbb{E}\left[(X_1 - \mathbb{E}[X_1])(X_1 - \mathbb{E}[X_1])\right] & \dots & \mathbb{E}\left[(X_1 - \mathbb{E}[X_1])(X_d - \mathbb{E}[X_d])\right] \\ \vdots & \ddots & \vdots \\ \mathbb{E}\left[(X_d - \mathbb{E}[X_d])(X_1 - \mathbb{E}[X_1])\right] & \dots & \mathbb{E}\left[(X_d - \mathbb{E}[X_d])(X_d - \mathbb{E}[X_d])\right] \end{pmatrix}$$

- In matrix notation we can express the covariance matrix as:

$$\Sigma := \mathbb{E}\left[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top\right]$$

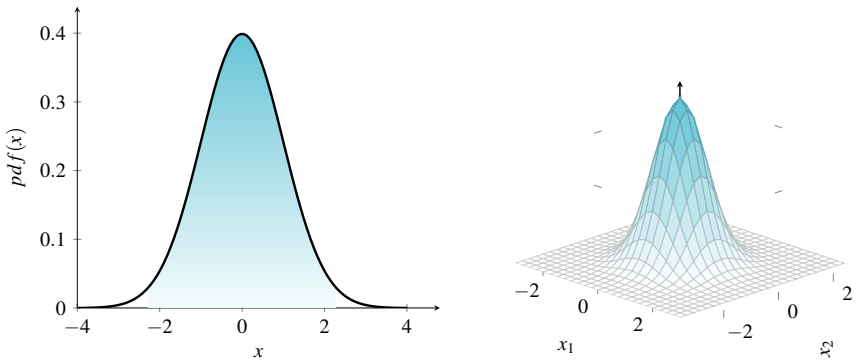
- The diagonal elements of Σ are $\sigma(X_i, X_i) \equiv \sigma^2_{X_i} \equiv Var(X_i)$. *Sigma* is a symmetric positive semi-definite matrix.

Now, using the notion of random vectors, joint distributions and covariance matrix we can define *multivariate normal distributions*.

Definition 1.2.5 A random vector $X := (X_1, \dots, X_d)^\top$ has a *multivariate normal distribution* with expectation μ and covariance matrix (see definition below) Σ if it has a joint PDF of the form:

$$f(X) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left\{-\frac{1}{2}(X - \mu)^\top \Sigma^{-1} (X - \mu)\right\}$$

In this case we write: $X \sim \mathcal{N}(\mu, \Sigma)$



(a) *Univariate Gaussian* $\mathcal{N}(0,1)$ ***(b)*** *Bivariate Gaussian* $\mathcal{N}(0, I_2)$

Figure 1.2: *Visualization of Uni- and Bivariate standard normal distributions*

Observe that Definition 1.2.5 is a generalization of Definition 1.1.2, i.e. when $d = 1$ both definitions are same. In the case of modeling human weight and height suppose the covariance between the properties is 0.2. We then denote the bi-variate normal distribution and the samples drawn from it by:

$$\mathbf{x}_1, \dots, \mathbf{x}_m \overset{i.i.d}{\sim} \mathcal{N}\left(\begin{bmatrix} 75 \\ 170 \end{bmatrix}, \begin{bmatrix} 3 & 0.2 \\ 0.2 & 5 \end{bmatrix}\right)$$

Sometimes, we are interested only in how a subset of the variates distributes, without the influence of the rest. For example what is the distribution of human height only. To do so, we would like to look at the *marginal distribution* of that subset.

Definition 1.2.6 The *marginal distribution* of a subset of coordinates of random variables with a joint probability distribution, is the probability distribution of the variables in the set:

$$f(\mathbf{x}) = \int_{\mathbf{y}} f(\mathbf{x},\mathbf{y})\, d\mathbf{y}$$

where the $\mathbf{y}$ integration is over all the random variables not in $\mathbf{x}$.

Let us find the marginal distribution of a bi-variate Gaussian. Let $\mathbf{x} \sim \mathcal{N}(\mu,\Sigma)$ where $\mathbf{x} \in \mathbb{R}^2$ and

$$\mu = (\mu_1,\mu_2)^\top, \quad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

To find the PDF of the marginal distribution of (w.l.o.g) x_1, Observe that we can write the PDF as follows:

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{\sqrt{(2\pi)^2|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right) \\ &= \frac{1}{\sqrt{(2\pi)^2\sigma_1^2\sigma_2^2}} \exp\left(-\frac{1}{2}\begin{bmatrix} x_1-\mu_1 & x_2-\mu_2 \end{bmatrix}\begin{bmatrix} \sigma_1^{-2} & 0 \\ 0 & \sigma_2^{-2} \end{bmatrix}\begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{bmatrix}\right) \\ &= \frac{1}{\sqrt{(2\pi)^2\sigma_1^2\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - \frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) \\ &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) \end{aligned}$$

Using the definition of the marginal distribution:

$$\begin{aligned} f(x_1) &= \int_{-\infty}^{\infty} f(x_1,x_2)\, dx_2 \\ &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) dx_2 \\ &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) dx_2 \end{aligned}$$

Notice that now we integrate a function of a uni-variate Gaussian for all values $x_2 \in (-\infty,+\infty)$. Therefore this integral equals to 1 and we are left with:

$$f(x_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right)$$

Which by definition 1.1.2 is a univariate Gaussian of the form $x_1 \sim \mathcal{N}(\mu_1,\sigma_1^2)$. Notice that this result is to be expected. The covariance between the two properties is zero, namely they are uncoordinated and as such they do not influence each others distribution.

1.2.1 Estimators of Multivariate Gaussian Distribution

Just as we were interested in estimating parameters in the univariate case so we are in the multivariate case. The sample mean estimator of a multivariate distribution is simply the univariate estimator in each of the coordinates:

$$\hat{\mu} := \begin{bmatrix} \vdots \\ \hat{\mu}_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{1}{m}\sum_i x_{i,j} \\ \vdots \end{bmatrix} \tag{1.14}$$

To define an estimator for the covariance matrix we must first define an estimator for the sample covariance between two random variables. The unbiased estimator of the *sample covariance* of the i'th and j'th random

variables is given by:

$$\hat{\sigma}(X_i, X_j) = \frac{1}{m-1}\sum_{k=1}^{m}(x_{k,i} - \hat{\mu}_i)(x_{k,j} - \hat{\mu}_j) \tag{1.15}$$

where $\hat{\mu}_i$ is the sample mean of the random variable X_i.
In particular, notice that for the case where $i = j$ we are left with the unbiased estimator previously seen. We can now define the sample covariance matrix $\hat{\Sigma}$:

Definition 1.2.7 — Sample Covariance Matrix. Let $X = (X_1, \ldots, X_d)^\top$ be a d-dimensional random vector. Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ be m i.i.d samples realizing X.. The ***sample covariance matrix*** is a square d-by-d matrix $\hat{\Sigma}$ such that $\hat{\Sigma}_{i,j} = \hat{\sigma}(X_i, X_j) \quad i, j = 1, \ldots, d$.

In matrix notation, for $\mathbf{X} \in \mathbb{R}^{m \times d}$ the matrix whose rows are the samples $\mathbf{x}_1, \ldots, \mathbf{x}_m$, the (biased) estimator for the sample covariance matrix is given by:

$$\hat{\Sigma} \coloneqq \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^\top = \frac{1}{m}\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}}$$

for $\widetilde{\mathbf{X}}$ being the centered matrix: $\widetilde{\mathbf{X}}_{\cdot,i} \coloneqq \mathbf{X}_{\cdot,i} - \hat{\mu}$. The unbiased estimator is given by:

$$\hat{\Sigma} \coloneqq \frac{1}{m-1}\sum_{i=1}^{m}(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^\top = \frac{1}{m-1}\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}}$$

■ **Example 1.3** Let $\mathbf{X} = \begin{pmatrix} 150 & 45 \\ 170 & 74 \\ 184 & 79 \end{pmatrix}$ be samples of height and weight of 3 different individuals. To calculate the sample covariance matrix we begin with centering the data. That is, subtract the empirical mean from each sample. The sample mean is: $\hat{\mu} = (168, 66)^\top$, so:

$$\mathbf{X}_{centered} = \mathbf{X} - \begin{pmatrix} 168 & 66 \\ 168 & 66 \\ 168 & 66 \end{pmatrix} = \begin{pmatrix} 150 & 45 \\ 170 & 74 \\ 184 & 79 \end{pmatrix} - \begin{pmatrix} 168 & 66 \\ 168 & 66 \\ 168 & 66 \end{pmatrix} = \begin{pmatrix} -18 & -21 \\ 2 & 8 \\ 16 & 13 \end{pmatrix}$$

Now, following the definition of the sample covariance matrix:

$$\hat{\Sigma} = \frac{1}{3-1}\mathbf{X}_{centered}^\top\mathbf{X}_{centered} = \begin{pmatrix} 292 & 301 \\ 301 & 337 \end{pmatrix}$$

■

1.3 Summary, Labs & Exercises

Exercises

Theoretical Questions

1. Let $x_1, x_2, \ldots \overset{i.i.d}{\sim} \mathcal{P}$ be a sample of infinity size drawn from some probability distribution function $\mathcal{P}$ with finite expectation and variance. Show that the sample mean estimator $\hat{\mu}_n = \frac{1}{n}\sum x_i$ calculated over the first n samples is a consistent estimator.

2. Consider the estimator of sample variance with *unknown* sample mean defined as $\hat{\sigma}^2 = \frac{1}{m}\sum(x_i - \hat{\mu})^2$, for $\hat{\mu}$ the sample mean estimator. Show that this is a biased estimator and find its systematic error.

3. Consider the estimator of sample variance with *known* sample mean defined as $\hat{\sigma}^2 := \frac{1}{m}\sum(x_i-\mu)^2$. Show that this is an unbiased estimator. Explain why the estimator in the question above is biased while the current estimator is unbiased.

4. Consider the estimator of sample variance with *unknown* sample mean defined as $\hat{\sigma}^2 = \frac{1}{m-1}\sum(x_i-\hat{\mu})^2$, for $\hat{\mu}$ the sample mean estimator. Show that this is an unbiased estimator.

5. Let $x_1,\ldots,x_m \overset{i.i.d}{\sim} \mathcal{N}\left(\mu,\sigma^2\right)$. Derive the MLE for the variance σ^2 and show that it is in fact the estimator seen in question Ex.2.

6. Let $\mathbf{x}_1,\ldots,\mathbf{x}_m \overset{i.i.d}{\sim} \mathcal{N}(\mu,\Sigma)$ be m observations sampled i.i.d from a multivariate Gaussian with expectation of $\mu \in \mathbb{R}^d$ and a covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. Derive the likelihood function of $\mathcal{N}(\mu,\Sigma)$. Hint: follow the approach used to derive the likelihood function for the univariate case (1.6).

7. Prove that multivariate Gaussian distributions are closed under affine transformations. That is, for $\mathbf{x} \sim \mathcal{N}(\mu,\Sigma)$, $\mu \in \mathbb{R}^d$, $\Sigma \in \mathbb{R}^{d\times d}$ and fixed $A \in \mathbb{R}^{m\times d}, b \in \mathbb{R}^m$ then $A\mathbf{x}+b$ follows a multivariate Gaussian distribution. What is the expectation and covariance matrix?

8. Let $X \sim \mathcal{N}(\mu,\Sigma)$ be a multivarite Gaussian distribution with $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d\times d}$. Denote $X = [X_1|X_2]^\top$ some partitioning of the coordinates of X. Prove that the conditional distribution $X_1|X_2$ is also a multivariate Gaussian and find its expectation and covariance matrix.

9. Let $X \sim \mathcal{N}(\mu,\Sigma)$ be a multivarite Gaussian distribution with $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d\times d}$. Denote $X = [X_1|X_2]^\top$ some partitioning of the coordinates of X. Prove that the marginal distribution X_1 is also a multivariate Gaussian and find its expectation and covariance matrix.

Practical Questions

Clone the IML.HUJI GitHub repository and setup a working python environment as explained on GitHub page.

1. Implement the `fit, pdf` and `log_likelihood` functions in class `UnivariateGaussian`, file `learners.GaussianEstimators.py`. Follow the details specified in class and function documentation.
   (a) Using `numpy.random.normal` draw 1000 samples $x_1,\ldots,x_{100} \overset{i.i.d}{\sim} \mathcal{N}(10,1)$ and fit a univariate Gaussian. What are the estimations of the expectation and variance?

   (b) Over previously drawn samples, fit a series of models of increasing samples size: 10, 20,...,100, 110,...1000. Plot the absolute distance between the estimated- and true value of the expectation, as a function of the sample size. What estimator property are we able to conclude for the sample mean estimator?

   (c) Compute the PDF of the previously drawn samples using the model fitted above (over all samples) and plot the empirical PDF distribution.

   (d) Over a sample $S = \{1,5,2,3,8,-4,-2,5,1,10,-10,4,5,2,7,1,1,3,2,-1,-3,1,-4,1,2,1\}$ which of the following models is more likely to have generated these samples: $\mathcal{N}(1,1)$ or $\mathcal{N}(10,1)$?

2. Implement the `fit, pdf` and `log_likelihood` functions in class `MultivariateGaussian`, file `learners.GaussianEstimators.py`. Follow the details specified in class and function documentation.

(a) Using `numpy.random.multivariate_normal` draw 1000 samples $\mathbf{x}_1, \ldots, \mathbf{x}_{1000} \overset{i.i.d}{\sim} \mathcal{N}(\mu, \Sigma)$

$$\mu = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 1 & 0.2 & 0 & 0.5 \\ 0.2 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}$$

Fit a multivariate Gaussian and evaluate the estimated expectation and covariance matrix with respect to the true parameters.

(b) Using the samples drawn in the question above calculate the log-likelihood for models with expectation $\mu = [f_1, 0, f_3, 0]^\top$ and the true covariance matrix defined above, where f_1, f_3 get values returned from `np.linspace(-10, 10, 200)`. Plot a heatmap of $f1$ values as rows, f_3 values as columns and color coded by the calculated log likelihood. What can be learned from the plot? What configuration of (f_1, f_3) values achieved maximal likelihood?

2. Linear Regression

2.1 Regression Models

The first class of learning problems we address is of regression problems. Imagine we work for an online store and would like to predict the "customer lifetime value", that is, the total future net revenue that an online customer will provide to the store. To do so, we choose different customer properties that we think might be relevant such as age, income, total spending in the website, average monthly visits, etc. The vector space defined over all possible values of these properties (commonly called features) is our *sample domain*. We denote it by $\mathcal{X}$. In the case of the online store $\mathcal{X} := \mathbb{R}^d$, for d the number of features. In addition, we also define the *response set* $\mathcal{Y}$ which in this case represents the customers' lifetime value. So in our case, $\mathcal{Y} = \mathbb{R}$.

Next, we collect these all these details for m customers. Each customer is represented as a *sample* – a pair $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathcal{X}$ is the column vector of features of the sample (also called data point) and the corresponding response $y \in \mathcal{Y}$. This set of *observations* is our *sample* which we will term as the *training dataset*. We would like to use our training dataset to find a way to *estimate* or *predict* the lifetime values of any new customer using solely their feature vector. This setup, for which each observation is characterized by a feature vector and a response value, is called *Batch Supervised Learning*. To try and estimate a customer's lifetime value we will build a *regression model*.

A regression model is a way to represent a functional relation between a set of explanatory variables (features) in $\mathcal{X}$ and a scalar response, also referred to as dependent variable, in $\mathcal{Y}$. So, we will *assume* that there exists some function $f : \mathcal{X} \to \mathcal{Y}$ that captures this relation for each sample $x \in \mathcal{X}$ and its response $y \in \mathcal{Y}$. This function f is unknown to us and we would like to find it. It may be deterministic or it may contain a random component.

We begin with assuming that the relation between $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ is *deterministic*. So we assume that there exists a function $f : \mathcal{X} \to \mathcal{Y}$ such that, each sample we observe, now or in the future, is of the form $(\mathbf{x}, y)$ with $y = f(\mathbf{x})$. In particular for our training set $y_i = f(\mathbf{x}_i)$ for every training sample $i = 1, \ldots, m$. Our goal is to *learn* f from a training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, so we can estimate or predict the value $f(\mathbf{x})$ for a new value $\mathbf{x}$. A sample we haven't seen in our training set – a new sample – is sometimes called a *test* sample. Using the

training sample S we will create a function that we hope is as similar as possible to the unknown function f. This function is the *prediction/decision rule* and we denote it by $\hat{f}$ or h_S. (The notation h_S emphasizes that our prediction rule depends on the training sample S).

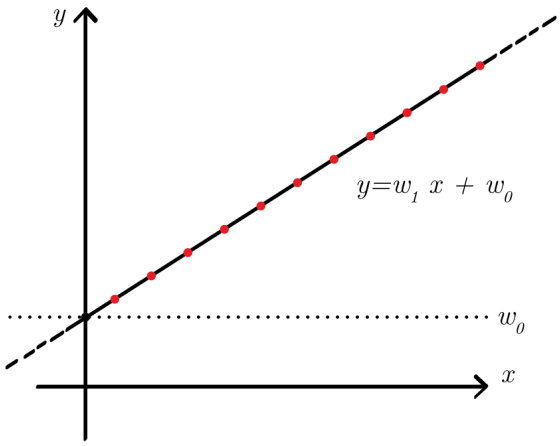


Figure 2.1: *Illustration of a regression model with $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$. Red dots are samples. The solid curve is the learned prediction rule $\hat{f}$.*

For reasons we discuss later (**??**), whenever we try to model a functional relation f, we restrict ourselves to a specific family of functions. Such a family is referred to as a *hypothesis class*. We decide on the hypothesis class before looking at the data, and the prediction rule we find must be in the chosen hypothesis class.

While we can build regression models over various domains $\mathcal{X}$, the simplest domain to consider is the Euclidean space $\mathbb{R}^d$ where each point x is a feature vector with d real numbers. In this chapter and in most of this book, we consider $\mathcal{X} \coloneqq \mathbb{R}^d$.

2.1.1 Linear Regression

Let us assume that the relation $\mathcal{X} \to \mathcal{Y}$ is *linear*. This is perhaps the simplest relation we can describe. Formally, we define the *linear model*, or the *linear hypothesis class*, as the set of linear functions from the domain set to the response set:

$$\mathcal{H}_{reg} \coloneqq \left\{ h(x_1,\ldots,x_d) = w_0 + \sum_{i=1}^{d} x_i w_i \;\middle|\; w_0, w_1, \ldots, w_d \in \mathbb{R} \right\} \tag{2.1}$$

In statistics, learning f from a training sample is known as *linear regression*[1]. Each function $h \in \mathcal{H}_{reg}$ is characterized by the *weights* (also known as regression coefficients) $w_1,\ldots,w_d$ representing the d features and an *intercept* w_0. To simplify the notation, for a given sample $\mathbf{x} = (x_1,\ldots,x_d)^\top \in \mathbb{R}^d$ we add a zero-th coordinate with the value of 1, and define $\mathbf{x} = (1,x_1,\ldots,x_d)^\top \in \mathbb{R}^{d+1}$. Using this notation each function in the linear hypothesis class can be written in the form $h(\mathbf{x}) \coloneqq \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^\top \mathbf{w}$. For the remainder of this chapter, we will assume that the intercept is already incorporated into the weights vectors, so we can define linear hypothesis class equivalently as

$$\mathcal{H}_{reg} \coloneqq \left\{ h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{2.2}$$

Note that by convention, the first coordinate of $\mathbf{w}$ is the intercept w_0.

[1] The name "regression" refers to a statistical phenomenon known as "regression to the mean".

So, given a training set S, we are looking for a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that $y_i = \mathbf{x}_i^\top \mathbf{w}$ for all $i \in [m]$. This setup should be familiar from **??**. However, as we will see, we may not be able to find a vector $\mathbf{w}$ for which all these equalities hold exactly.

The regression matrix

Let us arrange the training data $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ in matrix form. We define the *response vector* as the column vector $\mathbf{y} \in \mathbb{R}^m$ and the *regression matrix* (or *design matrix*) $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$ as follows.

$$\mathbf{X} = \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ & \vdots & \\ — & \mathbf{x}_m & — \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Note that m rows of $\mathbf{X}$ represent our m training samples and the $d+1$ columns of $\mathbf{X}$ represent the intercept and d features. In this notation, we are looking for a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies a system of m linear equations in the variable $\mathbf{w}$,

$$\mathbf{X}\mathbf{w} = \mathbf{y} \tag{2.3}$$

We must have enough training samples

At this point, we will assume that $m \geq d+1$, namely, that we have enough training samples so that the linear system (2.3) is not under-determined. In practical terms, this means that we have at least as many training samples as we have features. In our online store example, this means that we must collect data on $m \geq d+1$ customers before we start training our regression model, where d is the number of features we collect on each customer (e.g. age, income, total spending, number of monthly visits to the website, etc).

2.1.2 Designing A Learning Algorithm

2.1.2.1 Realizablity

Recall that to derive the problem of finding $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies (2.3) we have restricted ourselves to describing functional relations $\mathcal{X} \xrightarrow{f} \mathcal{Y}$ such that $f \in \mathcal{H}_{reg}$. The case where there exists a solution for (2.3) is called the *Realizable* case. Let $\hat{\mathbf{w}}$ be a solution for (2.3), then the prediction rule we choose is $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}$.

The case where there is no $f \in \mathcal{H}_{reg}$ that satisfies the system of equations (i.e there is no solution for the system) is called the *Non-Realizable* case. In this case, since we decided to choose a prediction rule in $\mathcal{H}_{reg}$, we must settle for finding $\hat{f} \in \mathcal{H}_{reg}$ which is *"most fitting"* for our purposes.

Our learning algorithm for linear regression must address both the realizable and non-realizable cases. In the realizable case, to find the rule f, all we need to do is solve the linear system (2.3) for $\mathbf{w}$. But what will we do in the non-realizable case, where $f \notin \mathcal{H}_{reg}$? How should we choose the prediction rule $\hat{f}$?

2.1.2.2 Empirical Risk Minimization

As we have seen in the previous chapter (subsubsection 1.1.2.1), one way to choose $\hat{f} \in \mathcal{H}_{reg}$ in the non-realizable case is to assign each $f \in \mathcal{H}_{reg}$ with some measure of quality, through the use of some *loss function*. This function will provide a measure of quality for the hypothesis by comparing between the true- and predicted values:

$$\sum_{i=1}^{m} L\left(f(\mathbf{x}_i), \hat{f}(\mathbf{x}_i)\right), \quad i = 1, \ldots, m$$

Two commonly used loss functions for regression problems are the *Absolute Value Loss* and *Squared Loss* functions.

$$L\left(y,\hat{f}\left(\mathbf{x}\right)\right) \coloneqq \left|y-\hat{f}\left(\mathbf{x}\right)\right|, \quad L\left(y,\hat{f}\left(\mathbf{x}\right)\right) \coloneqq \left(y-\hat{f}\left(\mathbf{x}\right)\right)^2$$

We focus on the linear regression setup when using the square loss function. As we are concerned for the performance of an estimator $\hat{f}$ on a new data point $\mathbf{x}$, we hope to minimize the *risk* (i.e. expected loss, 1.1.4) embodied in choosing $\hat{f}$ as our estimator, with respect to the squared loss:

$$\mathcal{R}\left(f,\hat{f}\right) \coloneqq \mathbb{E}_{\mathbf{x}}\left[L\left(f(\mathbf{x}),\hat{f}(\mathbf{x})\right)\right] = \mathbb{E}_{\mathbf{x}}\left[\left(f(\mathbf{x}),\hat{f}(\mathbf{x})\right)^2\right]$$

where the expectation is taken over the selection of the test sample $\mathbf{x}$. However, as we do not have access to the underlying distribution and only have the training set, we will evaluate this risk *empirically*, that is, over *the training data*. And so, using the sample mean as an estimator of the expectation, the *empirical risk* embodied in choosing $\hat{f}$ (with respect to the squared loss) is

$$\frac{1}{m}\sum_{i=1}^{m}\left(y_i-\hat{f}(\mathbf{x}_i)\right)^2$$

In the case of the square loss, the empirical risk of the linear function $\hat{f}\left(\mathbf{x}_i\right) = \mathbf{x}_i^\top \hat{\mathbf{w}}$ is given by:

$$\frac{1}{m}\sum_{i=1}^{m}\left(y_i-\mathbf{x}_i^\top\hat{\mathbf{w}}\right)^2 = \frac{1}{m}\left|\left|\mathbf{y}-\mathbf{X}\hat{\mathbf{w}}\right|\right|^2 = \frac{1}{m}\left(\mathbf{y}-\mathbf{X}\hat{\mathbf{w}}\right)^\top\left(\mathbf{y}-\mathbf{X}\hat{\mathbf{w}}\right) \tag{2.4}$$

We will then pick the estimator which *minimizes* the empirical risk. This strategy for choosing $\hat{f}$ is known as *Empirical Risk Minimization* (ERM). Since we are in search of the minimizer we often omit the constant value of $\frac{1}{m}$.

2.1.2.3 Least Squares Optimization Problem

Minimizing the empirical risk of (2.4) means minimizing the sum of squares of the deviations of the responses from a linear function. In other words, we choose the linear function in $\mathcal{H}_{reg}$ that is closest to the responses in terms of the squared error distance. The deviation $y_i - \mathbf{x}_i^\top \mathbf{w}$ is called the i-th *residual* and the total empirical risk in our case is called *Residual Sum of Squares* (or *RSS*):

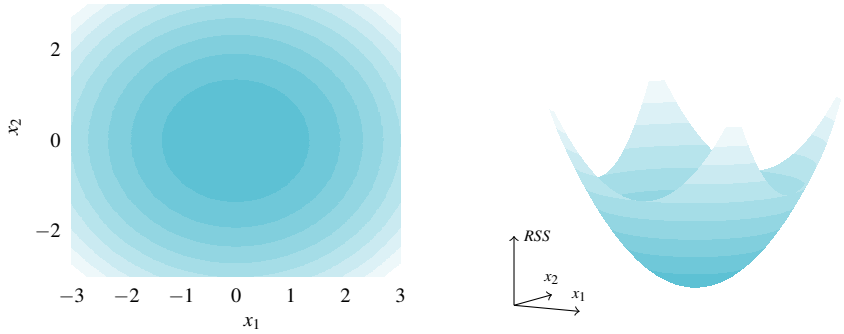
$$RSS_{\mathbf{X},\mathbf{y}}\left(\mathbf{w}\right) \coloneqq \left|\left|\mathbf{y}-\mathbf{X}\mathbf{w}\right|\right|^2$$

To simplify notation we often write $RSS\left(\mathbf{w}\right)$ keeping the dependence on $\mathbf{X},\mathbf{y}$ implicit. So to learn the linear function by empirical Risk minimization we want to find

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}}\ RSS\left(\mathbf{w}\right) = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}}\ \left|\left|\mathbf{y}-\mathbf{X}\mathbf{w}\right|\right|^2 \tag{2.5}$$

It is important to notice that the optimization problem (2.14) addresses both the realizable and non-realizable cases:

- In the realizable case, as $\mathbf{y} \in Im\left(\mathbf{X}\right)$ we know there exists at least one solution $\widehat{\mathbf{w}}$ such that $\mathbf{X}\widehat{\mathbf{w}} = \mathbf{y}$. Such a solution will achieve a value of zero. As the RSS function is bounded below by zero, such a solution is therefore a minimizer of the RSS.
- In the non-realizable case, as $\mathbf{y} \notin Im\left(\mathbf{X}\right)$ there is no solution $\widehat{\mathbf{w}}$ such that $\mathbf{X}\widehat{\mathbf{w}} = \mathbf{y}$. Therefore, no vector $\widehat{\mathbf{w}}$ will achieve a value of zero for the RSS objective. Instead, we decide to find a vector that is "good enough" in the sense of minimizing the squared loss.



(a) *Contour lines of the RSS function*

(b) *3D visualization of the paraboloid defined by the RSS*

Figure 2.2: *Visualization of RSS function*

A *necessary* condition for $\mathbf{w}$ to be a minimizer of the function $||\mathbf{y}-\mathbf{X}\mathbf{w}||^2$ is that all its partial derivative vanish at $\mathbf{w}$. Recalling the definition of the inner product, this condition can be written as:

$$\frac{\partial}{\partial w_j} RSS(\mathbf{w}) = -2\sum_{i=1}^{m} (\mathbf{x}_i)_j \cdot (y_i - \mathbf{x}_i\mathbf{w}) = 0 \tag{2.6}$$

for all $j = 0,\ldots,d$, where $(\mathbf{x}_i)_j$ is the j-th entry of $\mathbf{x}_i$. It is the $x_{j,i}$ element of the matrix $\mathbf{X}$. Notice that this constructs a system of $d+1$ linear equations in $\mathbf{w}$. We can organize (2.6) as such to get the form below. Recall that we have already derived this function in **??**.

$$\nabla RSS\,(\mathbf{w}) = -2\mathbf{X}^\top(\mathbf{y}-\mathbf{X}\mathbf{w}) = 0 \tag{2.7}$$

2.1.2.4 The Normal Equations

So a minimizer of (2.14) must also be a solution for the following linear system, known as the **Normal Equations**:

$$\mathbf{X}^\top(\mathbf{y}-\mathbf{X}\mathbf{w}) = 0 \quad \Longleftrightarrow \quad \mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\mathbf{w} \tag{2.8}$$

Geometric Interpretation

Let us derive a geometric interpretation of linear regression and gain a better understanding what the solution to (2.8) might be like. We usually think of $\mathbf{X} \in \mathbb{R}^{m\times(d+1)}$ as a matrix that consists of m rows, one for each training sample. Instead, we can equivalently think of $\mathbf{X}$ as a matrix that consists of $d+1$ columns, one for each feature (and the intercept). Define

$$\mathbf{X} := \begin{bmatrix} | & & | \\ \varphi_0 & \cdots & \varphi_d \\ | & & | \end{bmatrix}$$

and recall that the vector space spanned by the columns of $\mathbf{X}$ is:

$$span\,(\varphi_0,\ldots,\varphi_d) = Im\,(\mathbf{X}) \subset \mathbb{R}^m$$

Since we assume $m \geq d+1$, $Im\,(\mathbf{X})$ is a linear subspace of $\mathbb{R}^m$. If we have many more samples than features, $m \gg d+1$, then $Im\,(\mathbf{X})$ is just a small subspace of $\mathbb{R}^m$. If we have the minimal number of samples possible,

$m = d+1$, and the vectors $\varphi_0, \ldots, \varphi_d$ form an independent set, then the subspace fills the entire space: $Im(\mathbf{X}) = \mathbb{R}^m$.

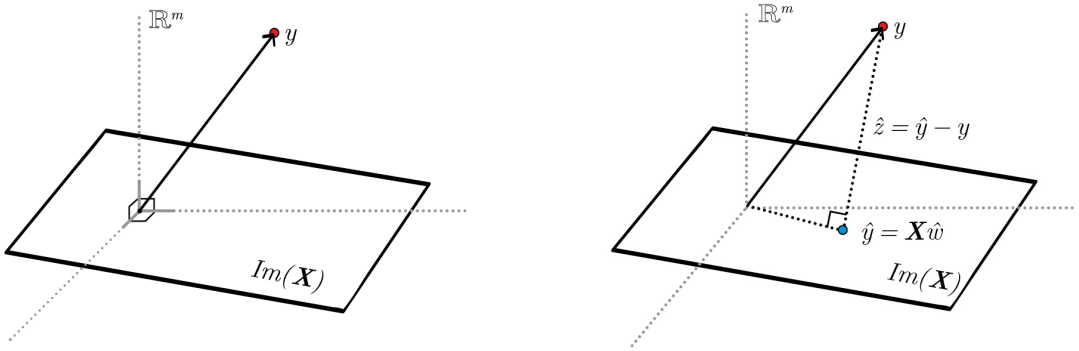
Now, consider the response vector $\mathbf{y} \in \mathbb{R}^m$:

- If $\mathbf{y} \in Im(\mathbf{X})$ then by definition $\mathbf{y}$ is a linear combination of $\varphi_0, \ldots, \varphi_d$ and there exists a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that $\mathbf{X}\mathbf{w} = \mathbf{y}$. This is the realizable case. We can now differentiate between two sub-cases:
  - If $\varphi_0, \ldots, \varphi_d$ are linearly independent, then $\mathbf{y}$ can be expressed as a *unique* linear combination of the columns of $\mathbf{X}$. In this case the linear system (2.8) has a unique solution.
  - If however $\varphi_0, \ldots, \varphi_d$ are in fact linearly dependent, then there are infinitely many ways to express $\mathbf{y}$ as a linear combination of the columns of $\mathbf{X}$. Any one of these ways is a valid solution for (2.8).
- If $\mathbf{y} \notin Im(\mathbf{X})$ then $\mathbf{y}$ is not a linear combination of $\varphi_0, \ldots, \varphi_d$. As such there is no vector $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies $\mathbf{X}\mathbf{w} = \mathbf{y}$. This is the non-realizable case. In this case we decided to choose the vector $\mathbf{w}$ for which $RSS(\mathbf{w}) = ||\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}||$ is minimal.

Now we are able to understand what is "normal" about the normal equations (2.8). Observe that the equations (2.6), from which we have derived the normal equations, can be equivalently written as

$$\langle \varphi_j, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle = 0 \quad j = 0, \ldots, d \tag{2.9}$$

We conclude that $\mathbf{w}$ is a solution to the normal equations if and only if $\mathbf{y} - \mathbf{X}\mathbf{w}$ is perpendicular to $\varphi_0, \ldots, \varphi_d$. Since these vectors span the subspace $Im(\mathbf{X})$, another way to write this is $\mathbf{y} - \mathbf{X}\mathbf{w} \in Im(\mathbf{X})^{\perp}$.



(a) *In the non-realizable case, the response vector* $\mathbf{y}$ *lies outside* $Im(\mathbf{X})$*, the subspace spanned by the columns of* $\mathbf{X}$*. In this case there is no solution for the system* $\mathbf{X}\mathbf{w} = \mathbf{y}$*.*

(b) *If* $\hat{\mathbf{w}}$ *is a solution to the normal equations, then* $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ *is an orthogonal projection of the response vector* $\mathbf{y}$ *onto* $Im(\mathbf{X})$*. The difference* $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ *is therefore perpendicular (normal) to* $Im(\mathbf{X})$*.*

Figure 2.3: Geometric interpretation of linear regression

Let $\hat{\mathbf{w}}$ be a solution to the normal equations and define $\hat{\mathbf{y}} := \mathbf{X}\hat{\mathbf{w}}$. Note that $\hat{\mathbf{y}}$, the vector where the i-th entry is the prediction on the i-th training sample $\mathbf{x}_i$, is $\hat{\mathbf{y}} \in Im(\mathbf{X})$. In this notation, when solving the normal equations, namely when seeking to minimize the RSS, we minimize $||\mathbf{y} - \hat{\mathbf{y}}||^2$. Define the *residual vector* $\hat{\mathbf{z}} := \mathbf{y} - \hat{\mathbf{y}}$. Note that from (2.9) we get that $\hat{\mathbf{z}} \in Im(\mathbf{X})^{\perp}$. In other words, if $\hat{\mathbf{w}}$ is a solution to the normal equations then $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ is the *orthogonal projection* of $\mathbf{y}$ on $Im(\mathbf{X})$ and $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ is a *normal* (a perpendicular vector) to $Im(\mathbf{X})$. Hence the name the "normal equations".

Solving The Normal Equations

As we have seen, from a geometric perspective, if $m \geq d+1$, solving the normal equations means finding a vector $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ is the orthogonal projection of $\mathbf{y}$ on $Im(\mathbf{X})$. We can deduce from this two important facts about the existence and uniqueness of a solution to the normal equations:

- **Existence:** As a linear system, the normal equations can have either (i) no solutions, (ii) a unique solution, or (iii) an infinite number of solutions that constitute an affine subspace. From the geometric interpretation we see that (i) is impossible. Indeed it can be shown that the normal equations must have at least one solution, so that they have a unique solution or an infinite number of solutions.
- **Uniqueness:**
  - If the columns of $\mathbf{X}$ form a linearly independent set (equivalently, if $dim(Ker(\mathbf{X})) = 0$) then the projection $\hat{\mathbf{y}}$ can be described uniquely as a linear combination of the columns $\varphi_0, \ldots, \varphi_d$, namely, there exists a unique $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$. This vector of coefficients $\hat{w}$ is a unique solution to the normal equations.
  - If the columns of $\mathbf{X}$ contain linear dependencies (equivalently, if $dim(Ker(\mathbf{X})) > 0$) then the projection $\hat{\mathbf{y}}$ can be described as infinitely many linear combinations of the columns $\varphi_0, \ldots, \varphi_d$. Any such linear combination will suffice and we simply need to find *one* vector $\hat{w}$ such that $\hat{y} = \mathbf{X}\hat{\mathbf{w}}$.

Case 1: Linearly Independent Feature Vectors

If the features are linearly independent then the kernel of $\mathbf{X}$ is trivial ($dim\left(Ker\left(\mathbf{X}\right)\right) = 0$). Now, consider the square and symmetric matrix $\mathbf{X}^\top\mathbf{X}$. As $Ker\left(\mathbf{X}\right) = Ker\left(\mathbf{X}^\top\mathbf{X}\right)$, then $\mathbf{X}^\top\mathbf{X}$ too has a trivial kernel. This means that $\mathbf{X}^\top\mathbf{X}$ is invertible and that a vector $\mathbf{w}$ satisfies $\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\mathbf{w}$ if and only if $\mathbf{w} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$. So in this case the unique solution to the normal equations is Ex.1

$$\hat{\mathbf{w}} := \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y} \tag{2.10}$$

Lastly, as the RSS function is a convex function (.26) we conclude that the unique solution $\hat{\mathbf{w}}$ is a minimizer of the function.

Notice when deriving a geometric interpretation of the normal equations (Figure 2.3) we argued that the minimizer will orthogonally project $\mathbf{y}$ onto the subspace spanned by the columns of $\mathbf{X}$. Looking at $\hat{\mathbf{y}} = \mathbf{X}\hat{w} = \mathbf{X}\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ we indeed find that $\mathbf{X}\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top$ is an orthogonal projection matrix onto the column space of $\mathbf{X}$. Ex.2

■ **Example 2.1** Let us find the estimator $\hat{\mathbf{w}}$ for the following scenario. Suppose we are interested in estimating the running times in a 100 meters long race, besed on an athlete's height and weight. We gathered the details of the 4 top ranking athletes in the 2016 Rio Olympics:

Athlete	Weight (kg)	Height (cm)	Running Time (sec)
Usain Bolt	94	195	9.81
Justin Gatlin	79	185	9.89
Andre de Grasse	70	176	9.91
Yohan Blake	80	180	9.93

So the features are the *weight, height* and the response is *running time*. To fit a linear regression model to the data we begin with arranging it in a matrix and adding the intercept:

$$\mathbf{X} := \begin{bmatrix} 1 & 94 & 195 \\ 1 & 79 & 185 \\ 1 & 70 & 176 \\ 1 & 80 & 180 \end{bmatrix}, \quad \mathbf{y} := \begin{bmatrix} 9.81 \\ 9.89 \\ 9.91 \\ 9.93 \end{bmatrix}$$

As we have proven above, the estimator is given by the closed form of $\hat{\mathbf{w}} := \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y}$. Over given data we obtain that $\hat{\mathbf{w}} \approx (11.38, 0.003, -0.009)^\top$ (up to rounding up numbers).

Next, let us use this estimator to estimate the running times of a new sample $\mathbf{x} = (1, 74, 176)^\top$:

$$\hat{y} = \mathbf{x}^\top \hat{\mathbf{w}} = \left\langle \begin{bmatrix} 1 \\ 74 \\ 176 \end{bmatrix}, \begin{bmatrix} 11.38 \\ 0.003 \\ -0.009 \end{bmatrix} \right\rangle = 10.018$$

■

Case 2: Linearly Dependent Feature Vectors

Next, let us consider the case of features that are linearly dependent. In this case the columns of $\mathbf{X}$ are linearly dependent and $dim\,(Ker\,(\mathbf{X})) > 0$. Therefore, there are infinitely many was to express the projection $\hat{\mathbf{y}}$ as a linear combination of the columns of $\mathbf{X}$. Since we need some way, it would be convenient if we could find a solution $\hat{\mathbf{w}}$ that is close to the origin in $\mathbb{R}^{d+1}$ (rather than a solution with very large norm, say). One way to do that is by using the SVD of $\mathbf{X}$.

Definition 2.1.1 Let $\mathbf{X} \in \mathbb{R}^{m\times(d+1)}$ and let $\mathbf{X} = U\Sigma V^\top$ be its SVD. The **Moore-Penrose pseudoinverse** of $\mathbf{X}$ is $\mathbf{X}^\dagger = V\Sigma^\dagger U^\top$ where $\Sigma^\dagger$ is a $(d+1)\times m$ diagonal matrix defined by:

$$\Sigma^\dagger_{i,i} = \begin{cases} 1/\Sigma_{i,i} & \Sigma_{i,i} \neq 0 \\ 0 & \Sigma_{i,i} = 0 \end{cases}$$

This is a generalization of the inverse matrix and indeed when the matrix $\mathbf{X}$ is invertible then then $\mathbf{X}^\dagger = \mathbf{X}^{-1}$. Using the definition above, let us find a minimizer of (2.14).

Claim 2.1.1 Let $\mathbf{X}, \mathbf{y}$ be a regression problem where $m \geq d+1$. If $dim(Ker(\mathbf{X})) = 0$ then $\hat{\mathbf{w}} = \mathbf{X}^\dagger \mathbf{y}$ is a minimizer of the RSS (2.14).

Proof. Denote the rank of $\mathbf{X}$ by r for which, since the kernel of $\mathbf{X}$ is non-trivial, $r\,[1, d+1)$. Let $X = U\Sigma V^\top$ be the SVD of $\mathbf{X}$ and $\sigma_1 \geq \ldots \geq \sigma_r > 0$. Recall the columns of U and V provide orthonormal bases for the four fundamental subspaces (**??**):

$$\begin{array}{llcl} U_\mathcal{R} \in \mathbb{R}^{m\times r} & \mathcal{R}(\mathbf{X}) & = & span\,\{\mathbf{u}_1, \ldots, \mathbf{u}_r\} \\ V_\mathcal{R} \in \mathbb{R}^{(d+1)\times r} & \mathcal{R}\left(\mathbf{X}^\top\right) & = & span\,\{\mathbf{v}_1, \ldots, \mathbf{v}_r\} \\ U_\mathcal{N} \in \mathbb{R}^{m\times(m-r)} & \mathcal{N}\left(\mathbf{X}^\top\right) & = & span\,\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\} \\ V_\mathcal{N} \in \mathbb{R}^{(d+1)\times(d+1-r)} & \mathcal{N}(\mathbf{X}) & = & span\,\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_{d+1}\} \end{array}$$

Denote $\mathcal{S} \in \mathbb{R}^{r\times r}$ the diagonal matrix with the r positive singular values on its main diagonal: $\mathcal{S} := diag\,(\sigma_1, \ldots, \sigma_r)$. Using these notations, recall the compact SVD form of $\mathbf{X}$ (36). So

$$\mathbf{X} := U\Sigma V^\top = \begin{bmatrix} U_\mathcal{R} & U_\mathcal{N} \end{bmatrix} \begin{bmatrix} \mathcal{S} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_\mathcal{R}^\top \\ V_\mathcal{N}^\top \end{bmatrix} = U_\mathcal{R} \mathcal{S} V_\mathcal{R}^\top = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^\top$$

Now we solve the normal equations for $\mathbf{w}$, substituting $\mathbf{X}$ with its compact SVD form:

$$\begin{array}{rcl} \mathbf{X}^\top \mathbf{y} & = & \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ \widetilde{V}\widetilde{\Sigma}^\top \widetilde{U}^\top \mathbf{y} & = & \widetilde{V}\widetilde{\Sigma}^\top \widetilde{U}^\top \widetilde{U}\widetilde{\Sigma}\widetilde{V}^\top \mathbf{w} \\ \widetilde{\Sigma}\widetilde{U}^\top \mathbf{y} & = & \widetilde{\Sigma}^2 \widetilde{V}^\top \mathbf{w} \end{array}$$

Since $\widetilde{\Sigma} \in \mathbb{R}^{r\times r}$ of full rank then $\widetilde{\Sigma}^{-1}$ exists and therefore:

$$\mathbf{w} = \widetilde{V}\widetilde{\Sigma}^{-1}\widetilde{U}^\top \mathbf{y}$$

Lastly, using the Moore-Perose pseudoinverse definition we "expand" the compact SVD form and conclude that:

$$\begin{aligned}\hat{\mathbf{w}} &= \widetilde{V}\widetilde{\Sigma}^{-}1\widetilde{U}^{\top}\mathbf{y} \\ &= V\Sigma^{\dagger}U^{\top}\mathbf{y} \\ &= \mathbf{X}^{\dagger}\mathbf{y}\end{aligned}$$

■

An important property of the pseudoinvers is that for a linear system of equations $A\mathbf{x} = \mathbf{b}$ with an infinite number of solutions, then $A^{\dagger}\mathbf{b}$ is a solution with minimal ℓ_2 norm, namely

$$A^{\dagger}\mathbf{b} = \text{argmin}\left\{||\mathbf{x}||_2 \;\middle|\; A\mathbf{x} = \mathbf{b}\right\} \tag{2.11}$$

and therefore $\hat{\mathbf{w}} := X^{\dagger}\mathbf{y}$ is the solution closest to the origin with respect ot the Euclidean norm.

It can be shown that when dealing with a matrix of linearly independent columns ($dim\,(Ker\,(\mathbf{X})) = 0$) then the previously found solution $\hat{\mathbf{w}} = \left[\mathbf{X}^{\top}\mathbf{X}\right]^{-1}\mathbf{X}^{\top}\mathbf{y}$ equals to $\mathbf{X}^{\dagger}\mathbf{y}$. We conclude that the formula $\mathbf{X}^{\dagger}\mathbf{y}$ *always* gives us a solution to the normal equations: the unique solution if the solution is unique, and the solution with minimal ℓ_2 norm if not. Ex.4

R In this chapter we have only delt with the case where $m \geq d+1$, namely that we are more samples than features, and were therefore able to find solutions to the linear system. If however $m < d+1$ we do not have enough data to learn a map $\mathcal{X} \xrightarrow{f} \mathcal{Y}$. Our hypothesis class is a $d+1$ dimensional linear subspace, so to learn we require at least $d+1$ samples. In general, the larger d, the more complicated the more complex our hypothesis class and therefore the more samples we will need in order to learn it. This intuition will be further formulated in **??**.

2.1.3 Numerical Considerations When Implementing

So far we have designed the learning algorithm. Now we want to *implement* it, namely, write efficient code that implements the algorithm that we have designed. The field of *numerical linear algebra* assists in addressing this challenge as the implementation of every machine learning algorithm is eventually reduced to performing linear algebra computations (e.g. matrix-vector or matrix-matrix products, matrix inverses and matrix decompositions). In the case of linear regression, as we have seen, to write software that trains a linear regression model, we need to be able to calculate a matrix SVD.

In your basic linear algebra courses you worked with mathematical objects over real and complex vector spaces. Likely, you did not stop to wonder how to compute (say) the inverse of a matrix on a computer. This is not as simple as it may sound. Computers do not calculate over $\mathbb{R}$, they use bits and more specifically floating-point arithmetics with finite precision. There is an entire field in the intersection of mathematics and computer science, known as numerical linear algebra, that studies the accuracy and complexity of algorithms for computing linear algebraic quantities and matrix decompositions. As a machine learning expert, you must be as knowledgable as possible regarding the numerical implementation of your learning algorithms. You should care *deeply* about how your algorithms are implemented and when they break numerically.

Let's see a simple example for a numerical consideration in our case of linear regression. (We will discover that the SVD is even more useful than we thought.) Recall that if $dim(Ker(\mathbf{X})) = 0$, and equivalently if $\mathbf{X}^{\top}\mathbf{X}$ is not singular (invertible) then we have a simple formula for training our linear regression model. But what happens if $\mathbf{X}^{\top}\mathbf{X}$ is "almost singular"?

Sometimes $\mathbf{X}^{\top}\mathbf{X}$ is formally invertible but *close to singular*. This happens if columns of $\mathbf{X}$ are almost co-linear or if one column of $\mathbf{X}$ is almost spanned by other columns. When this happens, if we are not careful we will run into numerical trouble. For example:

- Suppose we use the formula $\hat{\mathbf{w}} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ and try to compute $[\mathbf{X}^\top\mathbf{X}]^{-1}$ using (say) Gauss elimination, we'll find that Gauss elimination may yield wildly incorrect results.
- Suppose we use the pseudoinverse formula and compute $\mathbf{X}^\dagger$. When $\mathbf{X}^\top\mathbf{X}$ is close to singular, we'll discover that the smallest singular values σ_i of $\mathbf{X}$ are very very small; when we try to compute $1/\sigma_i$ for the pseudoinverse with floating-point arithmetics, $1/\sigma_i$ will not be precise.

There is a simple practical solution for this problem: we choose a "numerical precision threshold" $\varepsilon > 0$ in advance. We can choose, say, $\varepsilon := 10^{-8}$. We then change the definition of the pseudoinverse slightly and define

$$\Sigma^{\dagger,\varepsilon}_{i,i} = \begin{cases} 1/\sigma_i & \sigma_i > \varepsilon \\ 0 & \sigma_i \leq \varepsilon \end{cases}.$$

This ensures that even if the columns of $\mathbf{X}$ are close to being linearly dependent our implementation will be numerically stable.

2.1.4 A Statistical Model - Adding Noise

So far we assumed that the response y was a deterministic function of the sample $\mathbf{x}$, and that there was some deterministic function $f : \mathcal{X} \to \mathcal{Y}$ that underlies the relation $\mathcal{X} \to \mathcal{Y}$. This is an unrealistic assumption - in reality, measurements always contain randomness. In our online store example, we may consider that the revenue y measured for a customer is the sum of a deterministic component $\mathbf{x}^\top\mathbf{w}$ (where $\mathbf{x}$ is the customer's feature vector) and some random component z. This means that our dataset will not look like Figure 2.1 even if it is well described by the linear model. Instead is more likely to look like Figure 2.4.

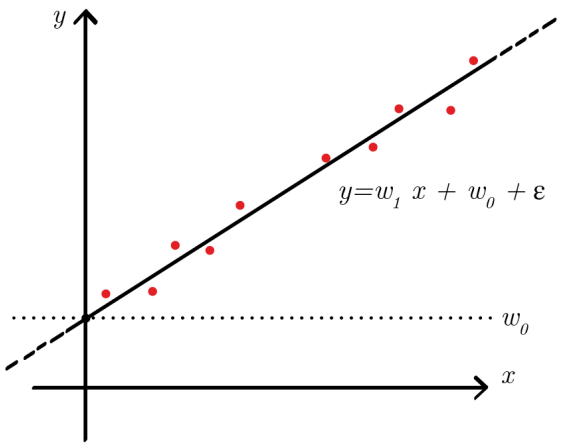


Figure 2.4: *Illustration of a linear regression model where training data is noisy*

To address this problem we describe a probabilistic model of the data. Let us assume, as before, that the relation $\mathcal{X} \to \mathcal{Y}$ linear, but with an additional factor capturing randomness in the relation. Suppose now that there exists a function $f : \mathcal{X} \to \mathcal{Y}$ such that the response for sample $\mathbf{x}$ is $y = f(\mathbf{x}) + \varepsilon$ where ε is some random variable. We assume that the noise ε in a sample is identically distributed and independent of the noise in any other sample. In particular, our training sample S is

$$\left(x_i, f(\mathbf{x}_i) + \varepsilon_i\right) \qquad i = 1, \ldots, m$$

with $\varepsilon_1, \ldots \varepsilon_m$ being i.i.d: independent and identically distributed. Let us adapt the learning algorithm we designed for the deterministic case to the probabilistic (noisy) case. Let us choose the linear hypothesis class $\mathcal{H}_{reg}$ as before, so that our learning algorithm will output a linear prediction rule. We also assume that we

have enough training data to learn, namely that $m \geq d+1$. We assume that there is a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that for every sample vector $\mathbf{x}_i$ in our data follows the model:

$$y_i = \mathbf{x}_i^\top \mathbf{w} + \varepsilon_i$$

Denoting the noise vector $\boldsymbol{\varepsilon} := (\varepsilon_1, \ldots, \varepsilon_m)^\top$ we have in matrix notation

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$$

Note that the vector $\mathbf{y}$ will typically not be in $Im(\mathbf{X})$, so that system $\mathbf{y} = \mathbf{X}\mathbf{w}$ has no solutions. As before, using the square loss function, and learning by the empirical risk minimization principal then:

$$\hat{\mathbf{w}} := \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \; ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$$

This means that our learning algorithm remains the same. We learn by solving the normal equation. As mentioned, $\mathbf{y} \notin Im(\mathbf{X})$ since the noise "pushed" $\mathbf{y}$ out of $Im(\mathbf{X})$. As we have seen, solving the normal equations is equivalent to projecting $\mathbf{y}$ back onto $Im(\mathbf{X})$, so our algorithms effectively attempts to remove the noise and recover the original prediction rule f.

2.1.4.1 Solving By Maximum Likelihood

Another approach to solving the problem of linear regression with noise is as follows. Suppose we assume further that the noise is Gaussian $\varepsilon_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$. This means that the i-th observation is independently distributed $y_i \sim \mathcal{N}(\mathbf{x}_i^\top \mathbf{w}, \sigma^2)$. In vector notation, we are assuming that the responses in our training sample follow a multivariate Gaussian distribution:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{X}\mathbf{w}, \sigma^2 I_m\right) \tag{2.12}$$

Now, suppose we *knew* the weight vector $\mathbf{w}$, we could then ask the following question: Given a fixed design matrix $\mathbf{X}$ and a known coefficients vector $\mathbf{w}$, what is the probability of observing the response vector $\mathbf{y}$? As each sample is independent to the others, the probability density is the product of the Gaussian densities of each sample

$$p(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{m} \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\mathbf{x}_i^\top \mathbf{w} - y)^2}{2\sigma^2} \right) \right] \tag{2.13}$$

This is a question in probability: We know $\mathbf{w}$ and ask for the chance to observe $\mathbf{y}$? However, when we design a learning algorithm, we are actually interested in the reverse question. We have the training sample, including the response vector $\mathbf{y}$. We are interested in a way to choose a linear prediction rule in $\mathcal{H}_{reg}$ and, equivalently, a vector $\mathbf{w}$. We can ask: what is the most "likely" value of $\mathbf{w}$ given the response vector that we observed. This is the Maximum Likelihood approach (subsection 1.1.3) where we choose $\mathbf{w}$ for which the probability density of getting the observed $\mathbf{y}$ is maximal.

As we assumed Gaussian noise, we could express the likelihood function (**??**) of $\mathbf{w}$ using the density function of the Gaussian distribution (1.2.5).

$$\begin{aligned}
\mathcal{L}(\mathbf{w}|X,\mathbf{y}) &= \mathbb{P}\left(y_1 = \mathbf{x}_1^\top \mathbf{w}, \ldots, y_m = \mathbf{x}_m^\top \mathbf{w} | \mathbf{X}, \mathbf{w}\right) \\
&= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2}{2\sigma^2} \right) \\
&= \frac{1}{\left(2\pi\sigma^2\right)^{m/2}} \prod_{i=1}^{m} \exp\left(-\frac{\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2}{2\sigma^2} \right) \\
&= \frac{1}{\left(2\pi\sigma^2\right)^{m/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2 \right)
\end{aligned}$$

Now we can use the likelihood function to derive the MLE for our linear regression model (2.12):

$$\begin{aligned}\hat{\mathbf{w}}^{MLE} &= \underset{\mathbf{w}}{\text{argmax}}\ \mathcal{L}(\mathbf{w}|\mathbf{y}) \\ &= \underset{\mathbf{w}}{\text{argmax}}\ \log \mathcal{L}(\mathbf{w}|\mathbf{y}) \\ &= \underset{\mathbf{w}}{\text{argmax}}\ \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2\right) \\ &= \underset{\mathbf{w}}{\text{argmin}}\ \sum_{i=1}^{m}\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2\end{aligned}$$

We therefore conclude that the MLE (assuming i.i.d Gaussian noise) is identical to the Least Squares estimator obtained from a completely different principle - that of empirical risk minimization.

2.1.5 Categorical Variables

to be added

2.2 Coefficient of Determination - R^2

When fitting a linear regression model we search for $\mathbf{w} \in \mathbb{R}^{d+1}$ which minimizes the RSS

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}}\ \text{RSS}(\mathbf{w}) = \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}}\ ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 \tag{2.14}$$

Notice however, that for a given model $\mathbf{w}$ and a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the quantity RSS($\mathbf{w}$) is not very informative as it has an arbitrary range, determined by the scaling of the data. As such, we cannot tell if an RSS of $10,000$ is good or bad. If the scaling of the data is in 0.1 this is an enormous error, but if the scaling of the data is in the millions then this error is not too bad. We would therefore like to derive some quantity with an intuitively meaningful range of values, which is insensitive to the scaling of the data.

The R-squared (R^2) provides a measure of the goodness of fit of a model. It measures how good are the model predictions in approximating the real data and is defined as the fraction of variance of $\mathbf{y}$ explained by the model

$$R^2 := 1 - \frac{\text{Unexplained Variation}}{\text{Totla Variation}} \tag{2.15}$$

The unexplained variation is the sum of the squared errors (SSE), while the total variation is total sum of squares (SST)

$$\text{SSE} := \sum (y_i - \hat{\mathbf{y}}_i)^2, \quad \text{SST} := \sum (y_i - \bar{y})^2 \tag{2.16}$$

It holds that SSE $\leq$ SST and thus $R^2 \in [0,1]$, providing an intuitive measure of how good is the fit. If $R^2 = 1$ it means that the amount of unexplained variation in y is zero, namely a perfect fit. If $R^2 = 0$ it means our model does not explain any of the observed variation in y.

2.2.1 Connection With Correlation Coefficient

An interesting observation regarding the R^2 is its connection to the Pearson correlation coefficient. The Pearson correlation coefficient is defined as follows:

$$\rho_{A,B} := \frac{cov(A,B)}{\sigma_A \sigma_B} \tag{2.17}$$

This measure, denoted also as r, captures the strength and direction of the *linear* relation between the two random variables A and B. Its values range between $[-1,1]$ where a value of 1 indicates a perfect linear relation between A and B, a value of 0 indicates no linear relation between A and B, and a value of -1 indicates a perfect linear anti-relation between A and B (i.e. $A = -B$).

It holds, that in the case of a linear LS regression with an intercept and a *single* variate then $R^2 = \rho_{y,x}^2$. In the multivariate case then $R^2 = \rho_{y,\hat{y}}^2$.

2.3 Polynomial fitting

Next, we would like to address the question of what type of functions we can learn using the linear regression model. Could we, for example, use the tools developed above, to learn the function $p(x)$ seen in Figure 2.5? And then, given some new value of x predict the value of $p(x)$?

Clearly this function is not a linear function. However, if we look closely at this polynomial $y = x^3 - 3x^2 + \frac{1}{2}x + 2$ and think of it not as a function of x but rather as a function of the coefficients of the polynomial, we realize that this is indeed a linear function. This function simply does not take the original values of x but instead some transformation of the original values: $x \mapsto \left(1, x, x^2, x^3\right)^\top$.

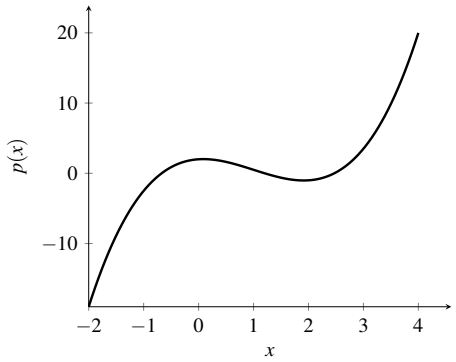


Figure 2.5: *A univariate polynomial* $p(x) = x^3 - 3x^2 + \frac{1}{2}x + 2$

To formulate the above understanding let $\psi_1, \ldots, \psi_k : \mathbb{R}^d \to \mathbb{R}$ be a set of functions such that each ψ_j receives a sample $\mathbf{x} \in \mathbb{R}^d$ and outputs a single value - namely, each ψ_j computes a single feature. Now, using these functions, that are referred to as *basis functions*, we can describe a relation that is *linear* in the parameters $\mathbf{w}$ but could be non-linear in the original input data:

$$y = \sum_{j=1}^{k} \psi_j(\mathbf{x}) \cdot w_j = \boldsymbol{\psi}(\mathbf{x})^\top \mathbf{w}, \qquad \boldsymbol{\psi}(\mathbf{x}) \coloneqq \left(\psi_1(\mathbf{x}), \ldots, \psi_k(\mathbf{x})\right)^\top \tag{2.18}$$

For the specific case of (univariate) polynomial fitting we would like to describe a polynomial relation between $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$ of degree at most $k \in \mathbb{N}$, but linear in the coefficients. The hypothesis class fitting this relation is:

$$\mathcal{H}^k_{poly} = \left\{ x \mapsto p_{\mathbf{w}}(x) \,|\, \mathbf{w} \in \mathbb{R}^{k+1} \right\} \tag{2.19}$$

where $p_{\mathbf{w}}(x) = \sum_{i=0}^{k} w_i x^i$. Thus, the set of basis functions is $\psi_j(x) = x^j$ for any $j \in \{0, \ldots, k\}$. As before, given a training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ we would like to choose a coefficients vector $\mathbf{w}$, best describing the coefficients of the polynomial. To do so we solve, as before, the LS problem:

$$\hat{\mathbf{w}} \coloneqq \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{k+1}} \frac{1}{m} \sum \left(y_i - p_{\mathbf{w}}(x_i)\right)^2 = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{k+1}} \frac{1}{m} \sum \left(y_i - \boldsymbol{\psi}(\mathbf{x})^\top \mathbf{w}\right)^2 \tag{2.20}$$

Notice that the design matrix $\mathbf{X}$, defined over the transformation $\boldsymbol{\psi}(\mathbf{x})$, is the Vandermonde matrix:

$$\mathbf{X} \coloneqq \begin{bmatrix} — & h(x_1) & — \\ — & h(x_2) & — \\ & \vdots & \\ — & h(x_m) & — \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^k \end{bmatrix}$$

Since we assume that the x_i's are different from one another, the design matrix $\mathbf{X}$ is of full rank. This means that solving this linear (in $\mathbf{w}$) system of equations can be done as we have seen for the non-singular case above. After finding $\hat{\mathbf{w}}$, we can predict the value of the unknown function p at a new point x using the value $p_{\hat{\mathbf{w}}}(x)$.

R Here we discuss polynomial fitting where $\mathcal{X} = \mathbb{R}$. With very little adaptation, we could also allow the input data to be $\mathcal{X} = \mathbb{R}^d$, $d > 1$. In such cases the defined polynomial could include terms of multiplication of two (or more) features. We will encounter such an example in **??**.

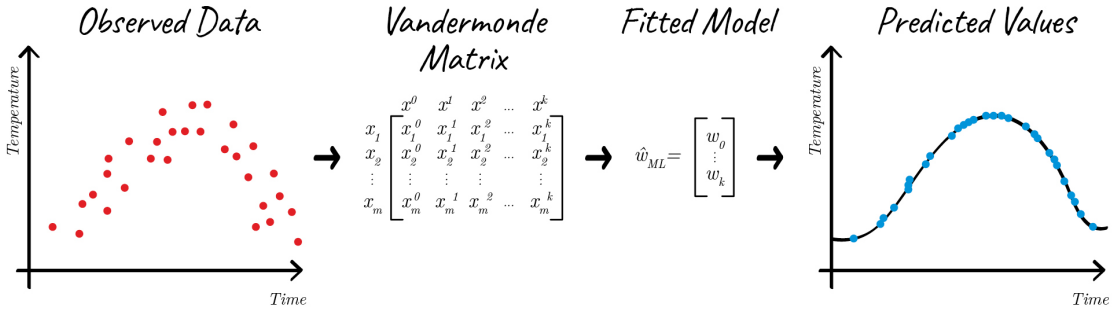


Figure 2.6: Scheme of Polynomial Fitting

2.4 Statistical Properties Least Squares Estimator - Bias & Variance

In the sections above we have seen that given a regression problem $\mathbf{X}, \mathbf{y}$, where $\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$, $\varepsilon \sim \mathcal{N}\left(0, \sigma^2 I_m\right)$, the estimator $\hat{\mathbf{w}} \in \mathbb{R}^d$ minimizing $||\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}||$ is given by $\hat{\mathbf{w}} := \mathbf{X}^{\dagger}\mathbf{y}$. It is important to notice that as $\mathbf{y}$ is a random variable, the LS estimator is too a random variable. Therefore, let us look at different properties of this estimator, specifically the *bias* (1.1.6) and *variance* (1.1.8) properties:

$$Bias\left(\hat{\theta}\right) = \mathbb{E}\left[\hat{\theta}\right] - \theta, \quad Var\left(\hat{\theta}\right) = \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right)^2\right]$$

Both these properties involve calculating the expectation of $\hat{\theta}$, but over what is the expectation calculated? An estimator is a decision function over a sample, used to estimate some parameter. Therefore, the expectation is over the selection of the samples:

$$Bias\left(\hat{\theta}\right) = \mathbb{E}_S\left[\hat{\theta}\right] - \theta, \quad Var\left(\hat{\theta}\right) = \mathbb{E}_S\left[\left(\hat{\theta} - \mathbb{E}_S\left[\hat{\theta}\right]\right)^2\right]$$

for $S \overset{i.i.d}{\sim} (\mathcal{X} \times \mathcal{Y})^m$. As we are not assuming any probability distribution over $\mathcal{X}$ this is equivalent to calculating the expectation over the sampling of $\varepsilon \sim \mathcal{N}\left(0, \sigma^2 I_m\right)$ and obtaining $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $y_i = \mathbf{x}_i^{\top}\mathbf{w} + \varepsilon_i$. And so, going back to the LS estimator, it can be shown that this is an *unbiased* estimator with variance of $\sigma^2\left[\mathbf{X}^{\top}\mathbf{X}\right]^{-1}$.

To understand what do the bias and variance properties imply for the quality of our estimation, let us revisit polynomial fitting. Consider for example the polynomial

$$Y = X^4 - 2X^3 - 0.5X^2 + 1 + \varepsilon \quad \varepsilon \sim \mathcal{N}(0, 2) \tag{2.21}$$

and let $x_1, \ldots, x_m$ be a set of samples where $x_i \in [-2, 2]$. For these observations let us create 10 different datasets, generated according to the model above. For each dataset we use $x_1, \ldots, x_m$ and generate the response value $y_1, \ldots, y_m$ with the addition of the noise. Figure Figure 2.7 shows the different datasets generated by the model above and the fitted polynomial of degree 1. Black, red and blue points represent the true model, the observed data-points (with the sample noise) and the fitted model over the observed data-points. Notice how the different datasets yield different predicted models. This is the randomness of the prediction, driven by the randomness of the trainset. Over these datasets we can now ask, for each value of x, what is the average prediction and its variance:

- In green is the average prediction of y for a given x across all datasets. The difference between the green and black lines capture the concept of the bias.
- In grey is the area of $\mathbb{E}[\widehat{y}] \pm 2 \cdot Var(\widehat{y})$ for a given x, also known as the confidence interval. The wider this area is, the more out prediction of $\widehat{y}$ varies for different samples. This area captures the concept of the variance.

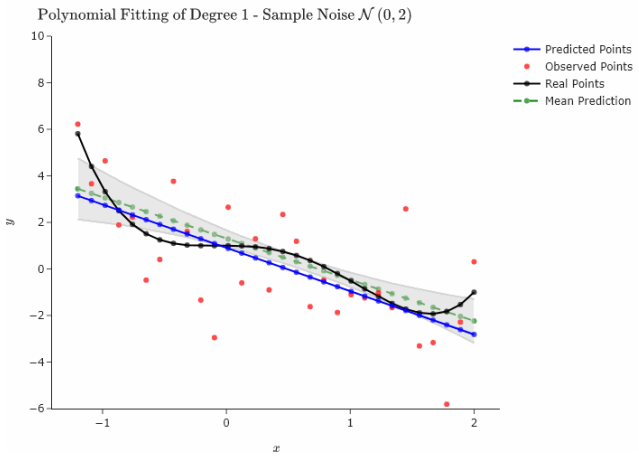


Figure 2.7: Polynomial Fitting: *Fitted polynomial of degree* 1 *over different datasets differing only in values of added sample noise. Chapter 2 Examples - Source Code*

Two phenomena are visible. The first is that the average distance of the fitted model (in green) and the true model (in black) is large. This means that our hypothesis class doesn't have sufficient expressive power to learn the true model. As such, we conclude that the *bias* of our estimator is high. The second is that the fitted models over different datasets do not differ by much. As such, we conclude that the *variance* of our estimator is low.

Next, consider the same setup as before but with the fitting of a polynomial of degree 8 (Figure 2.8). This time the different between the average prediction at each x and the true value of x is lower, while the differences between the fitted models (as indicated by the confidence intervals) is much higher. So the **bias** is low and the **variance** is high. As we enable more "flexible" (i.e. complex) models we are able to fit a model better to our given sample. However, as seen in Figure 2.8, if the model is too complex we might actually be fitting a model to the noise, rather than the actual true signal.

R It is important to note that what is seen in the figures are not the bias and variance of $\widehat{\mathbf{w}}^{LS}$ themselves but how these manifest over the shown datasets.

Interestingly, these two properties of bias and variance are linked. Let $\hat{\mathbf{y}} = \hat{\mathbf{y}}(S)$ denote the estimator of $\mathbf{y}$

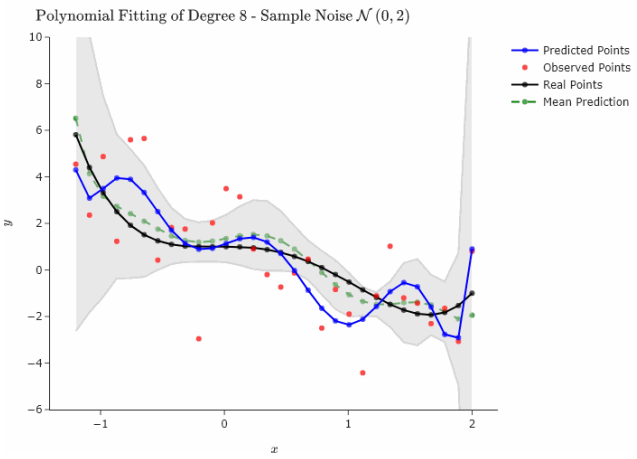


Figure 2.8: Polynomial Fitting: *Fitted polynomial of degree* 8 *over different datasets differing only in values of added sample noise.* *Chapter 2 Examples - Source Code*

when using $\widehat{\mathbf{w}}^{LS}$, and $\mathbf{y}^*$ the true $\mathbf{y}$ values. When solving the regression problem we wanted to minimize the mean square error between $\widehat{\mathbf{y}}$ and $\mathbf{y}^*$. What would be the expected MSE value?

Denote $\overline{\mathbf{y}} = \mathbb{E}[\hat{\mathbf{y}}]$ so:

$$\begin{aligned}\mathbb{E}\left[(\hat{\mathbf{y}}-\mathbf{y}^*)^2\right] &= \mathbb{E}\left[(\hat{\mathbf{y}}-\overline{\mathbf{y}}+\overline{\mathbf{y}}-\mathbf{y}^*)^2\right] \\ &= \mathbb{E}\left[(\hat{\mathbf{y}}-\overline{\mathbf{y}})^2\right]+2(\hat{\mathbf{y}}-\overline{\mathbf{y}})\mathbb{E}[\hat{\mathbf{y}}-\overline{\mathbf{y}}]+(\overline{\mathbf{y}}-\mathbf{y}^*)^2 \\ &= \mathbb{E}\left[(\hat{\mathbf{y}}-\overline{\mathbf{y}})^2\right]+(\overline{\mathbf{y}}-\mathbf{y}^*)^2 \\ &= Var(\hat{\mathbf{y}})+Bias^2(\hat{\mathbf{y}})^2\end{aligned} \tag{2.22}$$

Namely, we could *decompose* the generalization error (expected square loss between prediction and true value) into a variance component and a (squared) bias component.

$$MSE(\hat{\mathbf{y}}) = Var(\hat{\mathbf{y}})+Bias^2(\hat{\mathbf{y}}) \tag{2.23}$$

This means, that whenever we devise some estimator over our training data, the generalization error is influenced by both these factors. This is called the *Bias-Variance Trade-off*.

2.5 Summary and Exercises

Exercises

Theoretical Questions

1. Let $\mathbf{A}$ be some matrix. Show that $Ker(\mathbf{A}) = Ker(\mathbf{A}^\top\mathbf{A})$.
2. Let $\mathbf{X}$ be a design matrix with independent columns. Prove that $\mathbf{X}^\top\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top$ is an orthogonal projection matrix onto the column space of $\mathbf{X}$.
3. Let $\mathbf{A}$ be an invertible matrix. Show that $\mathbf{A}^\dagger = \mathbf{A}^{-1}$.
4. Let $\mathbf{X},\mathbf{y}$ be a linear regression problem where the columns of $\mathbf{X}$ are linearly independent. Show that $\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top = \mathbf{X}^{\dagger\top}\mathbf{y}$.
5. Let $\mathbf{X},\mathbf{y}$ be a regression problem such that $\mathbf{y} = \mathbf{X}\mathbf{w}+\varepsilon,\ \varepsilon\sim\mathcal{N}(0,\sigma^2 I_m)$ and $\hat{\mathbf{w}} := \mathbf{X}^\dagger\mathbf{y}$ the LS estimator. Show that $\hat{\mathbf{w}}$ is an unbiased estimator.
6. Let $\mathbf{X},\mathbf{y}$ be a regression problem such that $\mathbf{y} = \mathbf{X}\mathbf{w}+\varepsilon,\ \varepsilon\sim\mathcal{N}(0,\sigma^2 I_m)$ and $\hat{\mathbf{w}} := \mathbf{X}^\dagger\mathbf{y}$ the LS estimator. Prove that the variance of $\hat{\mathbf{w}}$ is $\sigma^2\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}$.

Practical Questions

Fitting A Linear Regression Model

In this part you will fit a linear regression model to the House Sales dataset.

1. Implement the `mean_square_error` function in the `metrics.loss_functions.py` file as described in the function documentation.

2. Implement the `LinearRegression` class in the `learners.regressors.linear_regression.py` file as described in the documentation. When implementing the `_loss` function be sure to call the `mean_square_error` function previously implemented.

3. Implement the `load_data` function in the `house_sales_prediction.py` and call the function. The function receives the path to the house sales dataset and returns a design matrix and response vector after performing any necessary preprocessing:
   - Addressing missing/invalid values for different features.
   - Treating categorical features.
   - Adding additional features that might be beneficial for predicting the price of the house.
   - Remove irrelevant features.

   Elaborate on the data exploration and decision making process that lead to the final preprocessing code.

4. Implement the `split_train_test` function in the `utils.utils.py` file as described in the function documentation. Call the function in the `house_sales_prediction.py` file, splitting the loaded dataset into a train set (75%) and test set (25%).

5. After splitting the dataset, fit a linear regression model (using your implementation of the `LinearRegression` class) over increasing percentages of the *train set* and evaluate performance over the *test set*. Do so in the following manner:
   - Iterate for every value $p = 10\%, 11\%, \ldots, 100\%$ of the train set samples.
   - Sample p of the train set, fit a model over these samples and evaluate performance over the test set.
   - Repeat the procedure of sampling, fitting and evaluating 10 times for every value of p.

   Plot the average loss, as well as a confidence interval of $\text{mean}(loss) \pm 2 \cdot \text{std}(loss)$, as a function of $p\%$. Explain the results seen in the plot.

6. Repeat the procedure of splitting the train- and test-sets and of fitting the model for different sample sizes (i.e questions 4,5) multiple times. Plot the average of loss averages as well as the confidence interval as a function of $p\%$. Explain what is the difference between the current evaluation approach and the previous.

Performing Polynomial Fitting

to be added

3. Classification

3.1 Classification Overview

In the previous chapter we discussed learning a regression problem where the response is a continuous value $\mathcal{Y} = \mathbb{R}$. When the response set $\mathcal{Y}$ is a finite set, this is a **classification** problem. We distinguish between classification problems where $|\mathcal{Y}| = 2$ (such as $\mathcal{Y} = \{\pm 1\}$ or $\mathcal{Y} = \{0,1\}$) and multi-classification problems where $\mathcal{Y} = \{1,\ldots,k\}$. In the binary classification problem (or just "classification"), we provide a "yes"/"no" prediction. In a multi-class classification, we predict one of $k > 2$ classes. For most, we restrict out discussion only to binary classification problems, though all methods below can be generalized to k classes. Also, we will only deal with the Euclidean sample space $\mathcal{X} = \mathbb{R}^d$, namely, each sample has d **features**. Therefore out setup is as follows:

$$\mathcal{X} := \mathbb{R}^d, \mathcal{Y} := \{\pm 1\}$$

Classification Problems Examples

- Predict whether a patient will develop a certain medical condition, or not.
- Predict whether a user will like a new product, or not.
- Determine if a given network traffic pattern is one of a cyber attack or not.
- Determine whether an art work is an original or forged.
- Determine whether a given email is spam or not.
- Detect fraud on credit card transactions.
- Predict whether a loan applicant will default on the loan.
- (Multi-class) What are the objects seen in a given picture.

■ **Example 3.1** Seen in Figure 3.1 are samples of the "South Africa Heart Disease" dataset. Given the parameters of blood pressure, smoking, family history, etc., could we predict who has/will have coronary heart disease (chd)? Notice that some of the features are numerical (e.g. tabacco, ldl, etc.) while some are categorical (e.g famhist). ■

Visualizing The Feature Space

When given a learning problem it is important to try and get intuition into "what the data looks like". In the case of a training sample for binary classification, we can plot the different axes and color by the label

	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
0	160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
1	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
2	118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
3	170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
4	134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1
...	...	...	...	...	...	...	...	...	...	...
457	214	0.40	5.98	31.72	Absent	64	28.45	0.00	58	0
458	182	4.20	4.41	32.10	Absent	52	28.61	18.72	52	1
459	108	3.00	1.59	15.23	Absent	40	20.09	26.64	55	0
460	118	5.40	11.61	30.79	Absent	64	27.35	23.97	40	0
461	132	0.00	4.82	33.41	Present	62	14.70	0.00	46	1

Figure 3.1: Example classification dataset: *South African Heart Data from Elements of Statistical Learning*

(Figure 3.2). This task is more difficult for data of higher dimensions, but attempting to imagine it in such cases will help understand what models might fit better to the specific task.

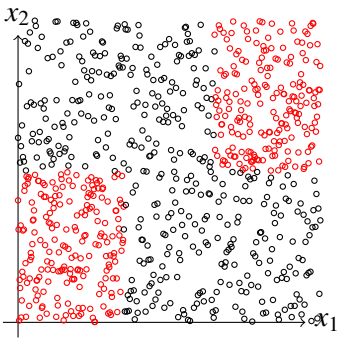


Figure 3.2: Classification training sample in $\mathbb{R}^2$***:*** *Where samples are positioned in space according to the values of their features and color coded by their label.*

3.1.1 Type-I and Type-II Errors

When we discussed regression problems we decided to measure the performance of a given hypothesis using the square loss (and mentioned that we could also use the absolute loss). In the case of classification our hypothesis outputs a label $\{\pm 1\}$ which we want to compare with the true labels also in $\{\pm 1\}$. It therefore makes less sense to measure "how far" is the prediction from the true value (as we do using the squared loss). Instead we would like to measure if we were correct or not. A very straight forward way to evaluate the performance of a classification predictor is to simply count the number of correctly classified samples. That is, given a prediction rule $h : \mathcal{X} \to \{\pm 1\}$ and a labeled sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the *misclassification* loss of h on this sample is:

$$L_S(h) := \sum_{i=1}^{m} \mathbb{1}_{y_i \neq h(\mathbf{x}_i)} = |\{i | y_i \neq h(\mathbf{x}_i)\}| \tag{3.1}$$

Can there be any problems or issues with the misclassification loss? After all, it just counts the number of times h was wrong - the number of times h misclassified a sample. In practice, there are two kinds of errors the

classifier can make, and making each kind of error might have very different implications, or costs. Therefore, simply counting the total number of errors may not be a useful performance measure.

■ **Example 3.2 — Credit Decisions.** Suppose we are building a classifier that predicts whether a bank customer seeking a loan is credit-worthy and will return a given loan or not. We choose the labels such that -1 means "not credit worth - deny loan", and 1 means "credit worthy - approve loan". Denote y_i the true label and $\widehat{y_i}$ the classifier-predicted label of sample i. The two errors this classifier might make have very different consequences:

- If $y_i = -1$ and $\hat{y}_i = 1$, the classifier predicted that a non-credit-worthy customer will return the loan. If we act on this prediction, and the customer defaults on the loan, the bank loses all the loan sum.
- On the other hand, if $y_i = 1$ and $\hat{y}_i = -1$, the classifier predicted that a credit-worthy customer, which would have paid the interest and returned the loan in full, is not credit-worthy and should be denied the loan. If we act on this recommendation, the bank loses the interest it would have earned on the loan.

Which of the two errors is more serious? Which of the two errors cost more for the bank? If we could choose which error should we avoid "at all costs" and which error could we "allow to happen", what would we choose? ■

■ **Example 3.3 — Drug safety.** Let us look at a more extreme example to help illustrate this point. We are creating a classifier to predicts whether a certain drug **is safe** to use for a particular person, or **unsafe/** deadly/ dangerous to use. We choose the labels such that -1 means "unsafe drug - do not use" and 1 means "safe drug - ok to use". Similar to before our errors are:

- If $y_i = -1$ and $\hat{y}_i$, the classifier recommends to give a drug which is actually potentially deadly.
- If $y_i = 1$ and $\hat{y}_i$, the classifier recommends that the patient should avoid a drug which is actually safe to use.

■

Therefore, we see that depending on the context of the classification problem, the two kinds of errors can have very different costs. We name the first error, the one we would like to avoid at all costs, the *Type-I error* and the second error as *Type-II error*. By choosing what label is "negative" and what label is "positive" we essentially defined what error is the Type-I error. As such, given a classification problem we try to choose the "negative" and "positive" labels such that the error we are more concern of (and therefore would like to avoid more) is the Type-I Error. That is, the error of misclassifying a negative sample by predicting it as a positive sample.

Returning to the drug safety example 3.3, we can assign the following meaning to the labels: $y = -1$ (negative) means the new drug is safe to use and $y = 1$ (positive) means the new drug is dangerous. In this case the Type-I error means that we decided not to offer a safe drug. If however we reverse the meaning such that $y = -1$ (negative) means the drug is dangerous and $y = 1$ (positive) means the new drug is safe, then the Type-I error means that we have decided to offer a dangerous drug. In this case this assignment of labels is the more serious of the two kinds of errors we can make.

For the classification problem of "is this email spam or not" how would you choose the labels? What are the two errors a spam detector can make? which one is the one we really want to avoid? So, which of the labels "spam email" and "not spam, valid email" would you label "negative" and which is "positive"?

3.1.2 Measurements of performance

With the decision on "positive" and "negative" labels, we define four basic terms: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative(FN). These terms refer to the prediction made for a sample with respect to its true label. So suppose a sample's true label is $y = -1$ (negative). If a classifier predicts:

- $\hat{y} = -1$ we term this as true negative.

- $\hat{y} = 1$ we term this as false positive.

Now, suppose a sample's true label is $y = 1$ (positive) then if a classifier predicts:

- $\hat{y} = -1$ we term this as false negative.
- $\hat{y} = 1$ we term as true positive.

Therefore, the false negative and false positive cases are the misclassification errors. False positive is what we referred to as Type-I error and false negative is what we called Type-II error. These four options are shown in **??**.

Using these four basic groups we can devise more domain-specific measurements. Denote by P the number of positive samples and N the number of negative samples then:

- The *Error Rate* is the number of misclassification out of all predictions: $(FP+FN)/(P+N)$.
- The *Accuracy* is the number of correct classification out of all predictions: $(TP+TN)/(P+N) = 1-ErrorRate$.
- The *Recall/ Sensitivity/ True-Positive-Rate (TPR)* is the number of truthfully positive predictions out of all positive samples: TP/P.
- The *False-Positive-Rate (FPR)* is the number of of falsely positive predictions out of all negative samples: FP/N.

There are many more measurements that can be defined from the four basic ones presented with different fields using different measurements. In Computer Science we often encounter the TPR and FPR for reasons described below.

3.1.3 Decision Boundaries

Let h be a binary classification rule in $\mathbb{R}^d$. (Suppose, for example, that we used a training sample to select h from some hypothesis class $\mathcal{H}$). We can feed any point $\mathbf{x} \in \mathbb{R}^d$ into h and get one of two classes. This means that we can view $\mathbb{R}^d$ as disjoint union of two sets:

$$\mathbb{R}^d = \{\mathbf{x}|h(\mathbf{x}) = 1\} \biguplus \{\mathbf{x}|h(\mathbf{x}) = 0\}$$

These sets can be very simple (two half-spaces) or very complicated. The boundary between these two sets is called the *decision boundary*: a test sample on one side of the boundary will be classified to one class by h, and a test sample on the other side of the boundary will be classified to the other class.

Different classifies, derived from different hypothesis classes, will generate different decision boundaries (Figure 3.3). Observing these over different data scenarios is helpful to understand is modeled by the different classifiers. It can also help get a qualitative assessment of the bias and variance of the classifiers.

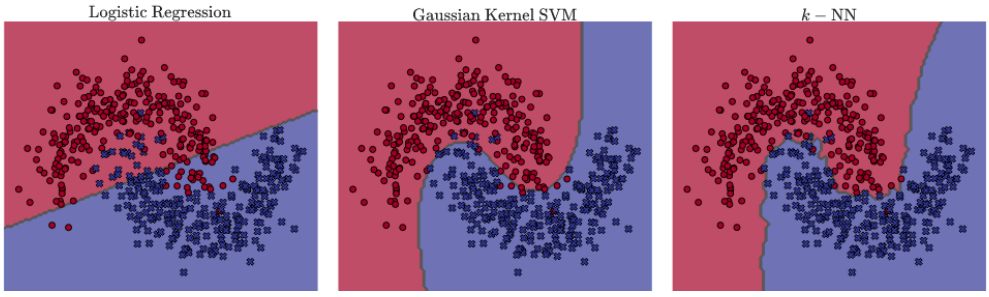


Figure 3.3: Decision Boundaries *of classifiers fitted over moons dataset. Chapter 3 Examples - Source Code*

3.1.4 Studying A New Classifier

For the rest of this chapter we will discuss different sorts of classifiers. As there are numerous types of classifiers, each for tailored for specific data scenarios, it is important to understand how to read about a new classifier. Therefore, when going over the classifiers below keep in mind the following guiding questions:

- How does it model the classification problem? and what are the assumptions made on the data?
- What is the hypothesis class defined? and how does the decision boundary looks like?
- What is the learning principle we use? and how does the algorithm match the learning principle?
- How can the learning principle can be implemented computationally? What is the time complexity of the algorithm and are there any considerations of numeric stability?
- What is done in the training step? and how, given a trained model, to predict for new samples?
- Is the model interpretable? Are we provided with estimations of class probabilities?
- Are we facing a single model or rather a family of models with some parameters for choosing specific models from this family? How do these parameters affect the bias-variance tradeoff?
- When will we decide to use this learning algorithm? What are its advantages and disadvantages?

3.2 Half-Space Classifier

Similar to linear regression, one of the simplest families of classifiers is that of linear classifiers. In these, we are interested in separating a given dataset into two classes using a linear separator function, as seen in Figure 3.4. It will be convenient to work with the class labels of $\mathcal{Y} := \{\pm 1\}$.

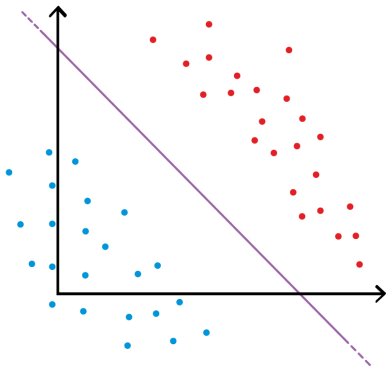


Figure 3.4: Half-space Classification Illustration: *For a domain-set $\mathcal{X} \in \mathbb{R}^2$ the two classes, coded as red and blue colors, are linearly separable .*

Similar to the definition used in linear regression (2.2), the family of linear functions can be described as the set of functions of the form $\mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w} + b$, $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. The linearity refers to the functions being linear in the parameters $\mathbf{w}$. Unlike in the regression setup, here we are interested in a mapping to a discrete response value.

Definition 3.2.1 Let $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The hyperplane defined by $(\mathbf{w}, b)$ is the set

$$\left\{ \mathbf{x} \middle| \langle \mathbf{w}, \mathbf{x} \rangle = b, \mathbf{x} \in \mathbb{R}^d \right\}$$

Definition 3.2.2 Let $(\mathbf{w}, b)$ be an hyperplane, so the half-space of $(\mathbf{w}, b)$ is defined as the set

$$\left\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle \geq b, \mathbf{x} \in \mathbb{R}^d\right\}$$

or equivalently as $\left\{\mathbf{x} | sign\left(\langle \mathbf{w}, \mathbf{x} \rangle - b\right) \geq 0, \mathbf{x} \in \mathbb{R}^d\right\}$.

Let us define the hypothesis class of half-spaces in $\mathbb{R}^d$. These functions can be thought of as the composition of the *sign* function over the linear functions:

$$\mathcal{H}_{half} := \left\{h_{\mathbf{w},b}(\mathbf{x}) = sign\left(\mathbf{x}^\top \mathbf{w} + b\right) \mid \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\right\} \tag{3.2}$$

So why are functions in the form seen in (3.2) are half-space classifiers? Let us assume at first that $b = 0$. We can express the domain set as a disjoint union of the following:

$$\mathbb{R}^d = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} > 0\right\} \biguplus \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} = 0\right\} \biguplus \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} < 0\right\} \tag{3.3}$$

These sets correspond to the open half spaces on either side of the hyperplane $\mathbf{w}^\perp = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} = 0\right\}$ and points on the hyper-plane itself. As such, each vector $\mathbf{w} \in \mathbb{R}^d$ defines a hyper-plane $\mathbf{w}^\perp$ that divides $\mathbb{R}^d$ into two half-spaces.

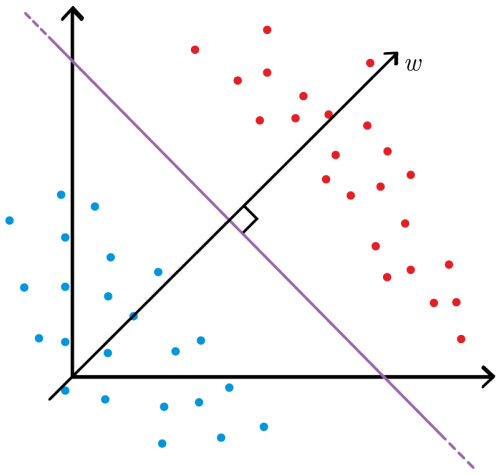


Figure 3.5: *Corresponding Hyperplane to* $\mathbf{w}^\perp$

The case where $b = 0$ is called the *homogeneous* case, as the hyperplane $\mathbf{w}^\perp$ is a linear subspace going through the origin. When $b \neq 0$ the hyperplane does not go through the origin and is called the non-homogeneous case. Recall that we have seen how we could transition from the non-homogeneous to the homogeneous case in the linear regression chapter.

Given a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, we would like to find an hypothesis $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ such that all data points in S that are labeled 1 are on the one side of the hyper-plane and all those labeled -1 are on the other side. To find such an hypothesis we must first make the assumption that the dataset is *linearly separable*. That is, there exists a hyper-plane such that samples of opposing labels are on opposite sides. Mathematically, we assume that

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad s.t \quad \forall i \in [m] \quad y_i \cdot sign\left(\langle \mathbf{x}, \mathbf{w} \rangle + b\right) = 1$$

or equivalently since the inner product will be negative for all samples with $y_i < 0$ and positive for all samples with $y_i > 0$:

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad s.t \quad \forall i \in [m] \quad y_i \cdot (\langle \mathbf{x}, \mathbf{w} \rangle + b) > 0 \tag{3.4}$$

Note, that assuming that a given training set is linearly separable is a *realizability assumption*. Namely, the labels are generated by a function in our hypothesis class $\mathcal{H}_{half}$. For simplicity, let us consider the homogeneous case where $b = 0$, since in the non-homoneneous case we can always shift the data by some fixed vector such that the separating hyperplane goes through the origin. So the hypothesis class of linear separators is of the form:

$$\mathcal{H}_{half} := \left\{ h_{\mathbf{w}}(\mathbf{x}) = sign\left(\mathbf{x}^\top \mathbf{w}\right) \;\middle|\; \mathbf{w} \in \mathbb{R}^d \right\} \tag{3.5}$$

3.2.1 Learning Linearly Separable Data Via ERM

To train a model over the defined hypothesis class of homogenous half-spaces ($\mathbf{w} \in \mathbb{R}^d, b = 0$) observe the following: for any hypothesis $h_{\mathbf{w}} \in \mathcal{H}_{half}$, the misclassified training samples are exactly those where $y_i \cdot sign\left(\mathbf{x}^\top \mathbf{w}\right) = -1$ or equivalently $y_i \cdot \mathbf{x}^\top \mathbf{w} < 0$. So defining the loss of a given hypothesis over S is:

$$L_S(h_{\mathbf{w}}) := \sum_{i=1}^{m} \mathbb{1}_{y_i \cdot \mathbf{x}^\top \mathbf{w} < 0} \tag{3.6}$$

Since we are assuming realizability (i.e. that S is linearly separable), we would like to find $h_{\mathbf{w}} \in \mathcal{H}_{half}$ that perfectly separates the training set. Such an hypothesis will be one that achieves $L_S(h_{\mathbf{w}}) = 0$. In other words, we are applying the ERM principle and seeking for any separating hyperplane $\mathbf{w}^\perp$, corresponding to an hypothesis $h_{\mathbf{w}}$ that minimizes the empirical risk $L_S(h_{\mathbf{w}})$.

So the next task is finding a compulationally efficient algorithm to find the desired hypothesis. As we are applying the ERM principle we would like to efficiently compute

$$\hat{\mathbf{w}} := \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \; L_S(h_{\mathbf{w}}) \tag{3.7}$$

where since assuming realizability, we know there exists a vector $\mathbf{w}^* \in \mathbb{R}^d$ such that $y_i \langle \mathbf{x}_i, \mathbf{w}^* \rangle > 0 \quad i = 1, \ldots, m$ (3.4). Notice, that if we define $\bar{\mathbf{w}} = \frac{\mathbf{w}^*}{\gamma}$ for $\gamma = \min_i (y_i \langle \mathbf{x}, \mathbf{w}^* \rangle)$ we have that

$$y_i \langle \mathbf{x}_i, \bar{\mathbf{w}} \rangle = \frac{1}{\gamma} y_i \langle \mathbf{x}_i, \mathbf{w}^* \rangle \geq 1 \quad i = 1, \ldots, m \tag{3.8}$$

Thus, it is enough to search for a vector $\hat{\mathbf{w}}$ that satisfies

$$y_i \cdot sign\left(\mathbf{x}^\top \hat{\mathbf{w}}\right) \geq 1 \quad \forall i = 1, \ldots, m \tag{3.9}$$

Convex Optimization

In (3.9) we therefore understand that the problem of finding a linear separator is in fact a a problem of finding a vector that satisfies m linear constraints. As these constraints are all convex constraints this is an exaple of what is known as a *convex optimization problem*. Specifically it is a case of a linear program.

Definition 3.2.3 — Optimization Problem. An optimization problem over $\mathbb{R}^d$ has the general form:

$$\begin{array}{ll} \text{minimize} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \leq b_i \quad i = 1, \ldots, n \end{array}$$

where $\mathbf{x}$ is the optimization variable, $f_0 : \mathbb{R}^d \to \mathbb{R}$ is the objective function and $f_i : \mathbb{R}^d \to \mathbb{R}$ are the constraint functions. It is implicitly implied that the optimization problem happens over $dom(f_0) \subset \mathbb{R}^d$,

the domain of f_0.

Then, naturally, a convex optimization problem is an optimization problem as above in which $f_0, f_1, \ldots, f_n$ are all convex functions. When these functions are all linear, this is a linear programming problem

Definition 3.2.4 — Linear Program. An optimization problem is called a Linear Program (LP) if it can be written in the following form:

$$\begin{array}{ll} \min\limits_{\mathbf{x}\in\mathbb{R}^n} & c^\top \mathbf{x} \\ \text{such that} & A\mathbf{x} \leq \mathbf{b} \end{array}$$

where $A \in \mathbb{R}^{m\times n}, \mathbf{c} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$ are fixed vectors and matrices.

In general, optimization problems are hard to solve computationally. We take special interest in *convex optimization* problems since they have a unique solution, and that solution can be found in computationally tractable ways. A great deal is known about *convex optimization algorithms*, which are iterative numerical algorithms that converge to the solution of a convex optimization problem. There are general solvers, which will solve a convex problem in the general form above, and there are specialized solvers for specific types, or families, of convex optimization problems. A specialized solver is typically preferred, as it leverages some particular structure of the problem to solve it more efficiently, using less space, etc.

Why is convex optimization interesting for machine learning? In supervised learning, we would like to choose a hypothesis $h \in \mathcal{H}$ from our selected hypothesis class, based on some learning principle (such as ERM). Many learning principles are formulated as optimization problems, namely, the h chosen by the learning algorithm is given as the minimizer of some quantity. So implementation of the learning algorithm needs to solve an optimization problem.

Sometimes, our hypothesis class is equivalent to a Euclidean space. When this happens, our learning principle reduces to solving an optimization problem, namely, the hypothesis we choose $h \in \mathcal{H}$ is found as a minimum over $\mathbb{R}^d$ or a subset of $\mathbb{R}^d$ of some objective function, usually a loss function. When this objective is convex, we can use convex optimization algorithms to implement our learning algorithm efficiently.

3.2.2 Solving ERM for Half-Spaces

Returning to the problem of the half-spaces classifier, we have seen that a hyperplane $\mathbf{w}^\perp$ minimizing the empirical risk is in fact a solution to the following linear program:

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & y_i \cdot \langle \mathbf{x}, \mathbf{w} \rangle \geq 1 \quad i = 1, \ldots, m \end{array} \tag{3.10}$$

Such an optimization problem, where a trivial objective, it is a *feasibility* problem. That is, we are looking for any vector which satisfies the constraints. To solve this we can apply some generic solver for linear programs.

3.2.2.1 The Perceptron Algorithm

Another way for finding a separating hyperplane using the ERM principle is by using the Perceptron algorithm, suggested by Frank Rosenblatt in 1958. This is an iterative algorithm that constructs a series of vectors $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$, where each vector is derived from the vector preceeding it. At each iteration t we search for a sample i which is misclassified by $w^{(t)}$. Then, we update $\mathbf{w}^{(t)}$ by moving it in the direction of thhe misclassified sample $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$.
Our goal when using the Perceptron algorithm is to find a vector $\mathbf{w}$ such that $y_i \cdot \mathbf{x}_i^\top \mathbf{w} > 0 \quad i = 1, \ldots, m$.
Notice that as

$$y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle + ||\mathbf{x}_i||^2$$

Algorithm 1 Batch-Perceptron

procedure PERCEPTRON($S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$)
  $\mathbf{w}^{(1)} \leftarrow 0$ ▷ Initialize parameters
  for $t = 1, 2, \ldots$ **do**
    if $\exists i \; s.t. \; y_i \left\langle \mathbf{w}^{(t)}, \mathbf{x}_i \right\rangle \leq 0$ **then** ▷ If there exists a misclassified sample
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      return $\mathbf{w}^{(t)}$
    end if
  end for
end procedure

the update rule of the Perceptron iteratively adjusts the hyperplane to be "more correct" on the i'th sample. It can be shown that in the realizable case the algorithm is guaranteed to terminate, returning a solution that correctly classifies all the samples.

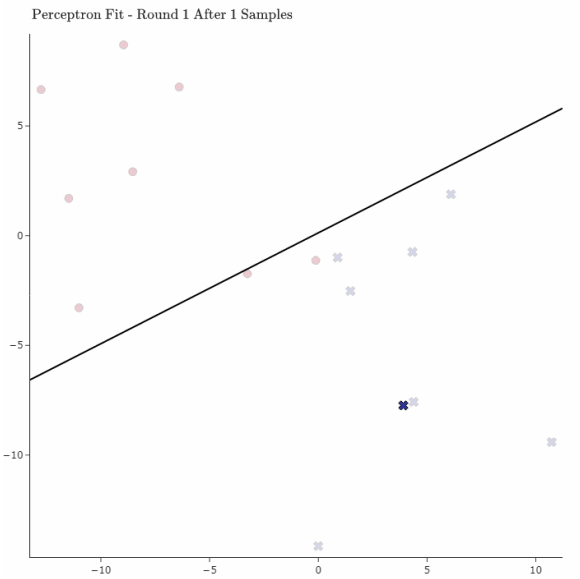


Figure 3.6: Perceptron Fitting: *Fit a separating hyperplane using Perceptron algorithm. Chapter 3 Examples - Source Code*

R The Perceptron algorithm is in fact a simple case of the more general algorithm of Subgradient Descent covered in **??**.

3.2.3 Learner ID Card

- **Hypothesis class:** the class of linear separators (3.5)
- **Learning principle used for training:** ERM for misclassification loss
- **Computational implementation:** Linear programming or Perceptron
- **Interpretability:** We do not have any specific insight into why a solution was chosen besides it simply satisfying the conditions.

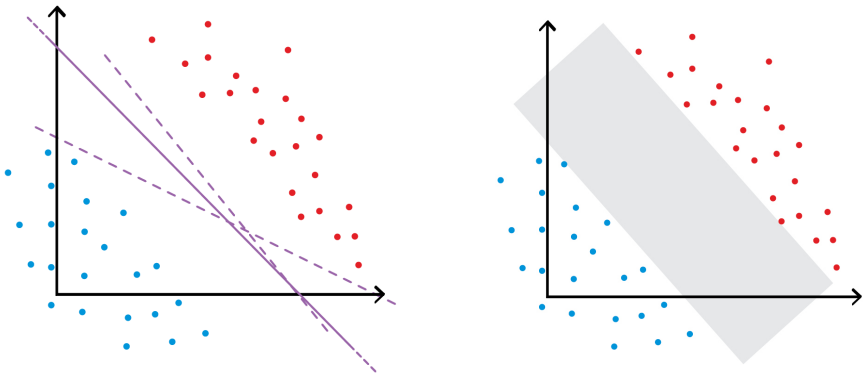
- **Family of models:** No.
- **Storing fitted model:** Fitted model is the vector $\mathbf{w}$ perpendicular to the hyperplane defining the half-space. To store the model we simply store the d coefficients of the vector. In the case of non-homogeneous halfspace we also store the intercept coordinate
- **Prediction of new sample:** $\hat{y}_{new} := sign\left(\mathbf{x}_{new}^{\top}\mathbf{w}+b\right)$
- **When to use:** Since realizability assumption rarely holds this classifier is only used as a simple baseline

3.3 Support Vector Machines (SVM)

When using the half-space classifier seen above, we encounter two problems:

- Solution Uniqueness: When we are searching for a separating half-space the solution is not unique. That is, there could be more than a single vector satisfying the constraints of (3.10) and achieving the minimal empirical loss (of zero when assuming realizability or any other positive number if not). As such we are faced with the problem of which one to choose. Figure 3.7 illustrates the existance of multiple separating hyper-planes.

- Realizability: A more severe problem rises when we chose to work with the ERM learning principle for selecting the hypothesis, but the data is not linearly separable (non-realizible case). In this case the optimization problem described in 3.10 is computationally hard.



(a) *Algorithm selected hyper-plane (solid line) and additional optional separating hyperplanes (dashed lines)*

(b) *All hyper-planes linearly separating the data*

Figure 3.7: *Illustration of the existance of multiple separating hyper-planes*

Returning to the hypothesis class of non-homogeneous separating half-spaces $\mathcal{H}_{half}$ (3.2), we would like to describe a different learning principle that will be able to cope with both problems above: finds a unique hyperplane and that can be implemented computationally efficiently even when data is not linearly separable (i.e with polynomial running time given the input).

3.3.1 Maximum Margin Learning Principle

This learning principle is the one of maximum margin.

Definition 3.3.1 Let $(\mathbf{w},b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane and $u \in \mathbb{R}^d$. Define the distance between $(\mathbf{w},b)$

and u by:

$$d\left((\mathbf{w},b),u\right) := \min_{v:\langle v,w\rangle+b=0} ||u-v||$$

(namely, the Euclidean distance between u and the closest point on the hyperplane)

Definition 3.3.2 — Margin. Let $(\mathbf{w},b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane and $S = u_1,\ldots,u_m \in \mathbb{R}^d$ a set of points. The **margin** of $(\mathbf{w},b)$ and S is the smallest distance between the hyperplane and any point:

$$M\left((\mathbf{w},b),S\right) := \min_{i\in[m]} d\left((\mathbf{w},b),u_i\right)$$

The margin of a given hyperplane with respect to S is therefore the minimal distance between the hyperplane and a sample in S. It seems logical that a hyperplane with a larger margin is more likely to still satisfy all separability constraints even if S is slightly different.

So, the new learning principle is: choose $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ that has the **largest margin** with respect to our training data S. Figure 3.8 illustrates the margins of two different potential hyperplanes. Based on these, we would prefer selecting the hyper-plane that is in the center of the grey area. The vectors closest to the hyperplane determine the margin. They are called **support vectors** and hence this learner's name.

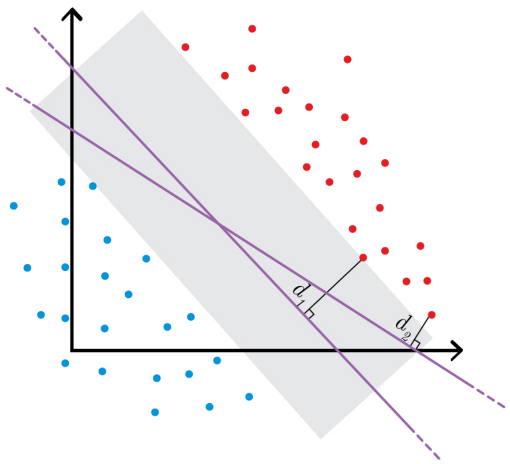


Figure 3.8: *Margin of specified hyper-planes*

3.3.2 Hard-SVM

Let us start with the **realizable case**. To implement our learning principle of maximal margin, we need to search, among all the separating hyperplanes of S, for the hyperplane with maximum margin. Namely, the hypothesis $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ our learner will choose is the solution to the following optimization problem:

$$\begin{array}{ll} \text{maximize} & M\left((\mathbf{w},b),S\right) \\ \text{subject to} & y_i \cdot \left(\langle \mathbf{x}_i,\mathbf{w}\rangle + b\right) > 0 \quad i = 1,\ldots,m \end{array} \tag{3.11}$$

The optimization variables are $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. Comparing with the linear program of half-spaces (3.10) we see that the constraints are kept, which ensure the hyperplane chosen separates the training sample, but instead of a trivial objective, we seek to minimize the margin.

3.3.2.1 Solving Hard-SVM

So is the Hard-SVM a convex optimization problem? Recall, that by our optimization problem (3.11), we are searching of a separating hyperplane that maximizes the margin from all points. As for any $c > 0$ it holds that $(\mathbf{w}, b) = (c\mathbf{w}, cb)$ we can w.l.o.g constraint ourselves to $||\mathbf{w}|| = 1$. This way, each hyperlane has a unique vector $\mathbf{w}$ that corresponds to it.

In order to calculate the margin $M((\mathbf{w}, b), S)$ we first must define what does it mean to measure a distance between a point in space and a set of point. For the purpose of the SVM classifier we define the distance as follows. Let $\mathbf{x} \in \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$. The distance from $\mathbf{x}$ to B: is $\inf_{\mathbf{v} \in B} ||\mathbf{x} - \mathbf{v}||$.

Lemma 3.3.1 Let $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane where $||w|| = 1$ and $\mathbf{x} \in \mathbb{R}^d$ then the distance between $\mathbf{x}$ and the hyperplane $(\mathbf{w}, b)$ is $|\langle \mathbf{x}, \mathbf{w} \rangle + b|$.

* We saw that adding a 1 coordinate to the feature vector allows us to express the bias term. Therefore, for brevity, we would neglect it and only consider $|\langle \mathbf{x}, \mathbf{w} \rangle|$ instead of $|\langle \mathbf{x}, \mathbf{w} \rangle + b|$.

Proof. Since we define the distance between $\mathbf{x}$ and the hyperplane as the minimal distance between $\mathbf{x}$ and a vector in the hyperplane we are in fact looking that the orthogonal projection of $\mathbf{x}$ onto the heyperplane.

Similar to the way we have dealt with the intercept term in linear regression, let us assume that $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{d+1}$, $x_0 = 1$ and w_0 represents the intercept. Let $P = \mathbf{w}\mathbf{w}^\top$ be the projection matrix onto span$(\mathbf{w})$ and consider the matrix $(I - P)$. Firstly, this is a projection matrix as

$$(I-P)^2 = I^2 - 2P + P^2 = I - 2P + P = I - P$$

Secondly, it holds that $Im(I - P)$ is the hyperplane defined by $(\mathbf{w}, b)$. Let $\mathbf{u} \in \mathbb{R}^d$ then :

$$\begin{aligned} \langle (I-P)\mathbf{u}, \mathbf{w} \rangle &= \langle \mathbf{u} - P\mathbf{u}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle - \langle P\mathbf{u}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle - \langle \mathbf{u}, P^\top \mathbf{w} \rangle \\ &= \langle \mathbf{u}, \mathbf{w} \rangle - \langle \mathbf{u}, \mathbf{w}\mathbf{w}^\top \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle - \langle \mathbf{u}, \mathbf{w} \rangle = 0 \end{aligned}$$

Thus $(I - P)$ is an orthogonal projection matrix onto the hyperplane $(\mathbf{w}, b)$ and $(I - P)\mathbf{x}$ is the orthogonal projection of $\mathbf{x}$ onto $(\mathbf{w}, b)$. Therefore, the distance between $\mathbf{x}$ and the hyperplane is given by:

$$||\mathbf{x} - (I-P)\mathbf{x}|| = ||\mathbf{x} - \mathbf{x} + P\mathbf{x}|| = \left|\left|\mathbf{w}\mathbf{w}^\top \mathbf{x}\right|\right| = ||\langle \mathbf{x}, \mathbf{w} \rangle \cdot \mathbf{w}|| = |\langle \mathbf{x}, \mathbf{w} \rangle| \cdot ||\mathbf{w}|| = |\langle \mathbf{x}, \mathbf{w} \rangle|$$

■

So, as the margin between a given hyperplane $(\mathbf{w}, b)$ and a set of points S is the minimal distance between the hyperplane and any point in the set, we derive that our optimization problem is in fact of the form:

$$\begin{aligned} \underset{(\mathbf{w},b):||\mathbf{w}||=1}{\text{argmax}} \quad & \min_{i \in [m]} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| \\ \text{subject to} \quad & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0 \quad i = 1, \ldots, m \end{aligned} \tag{3.12}$$

While the constraints enforce $\mathbf{w}$ to define a separating hyperplane, the objective will make us choose a separating hyperplane with the maximal margin. We further simplify the optimization problem. Consider a *feasible* solution $\mathbf{w}$ to the problem (i.e. that satisfies all constraints). It holds that $|\langle \mathbf{x}_i, \mathbf{w} \rangle + b| = y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b)$. Hence, we can rewrite (3.12) as:

$$\begin{aligned} \underset{(\mathbf{w},b):||\mathbf{w}||=1}{\text{argmax}} \quad & \min_{i \in [m]} y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \\ \text{subject to} \quad & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0 \quad i = 1, \ldots, m \end{aligned} \tag{3.13}$$

Notice that the constraints are now redundant. If $\mathbf{w}$ is infeasible then $\min_i y_i\left(\mathbf{x}_i^\top \mathbf{w}+b\right)<0$, achieving a lower objective than any feasible solution. Therefore, we can re-write the problem as:

$$\underset{(\mathbf{w},b):||\mathbf{w}||=1}{\operatorname{argmax}} \quad \min_{i\in[m]} y_i\left(\langle \mathbf{x}_i,\mathbf{w}\rangle+b\right) \tag{3.14}$$

And lastly, we notice that we can represent (3.14) as a norm minimization problem instead of margin maximization problem:

Claim 3.3.2 Consider the following optimization problem:

$$\underset{(\mathbf{w},b)}{\operatorname{argmin}} \ ||\mathbf{w}||^2 \qquad \text{subject to} \quad y_i\left(\langle \mathbf{x}_i,\mathbf{w}\rangle+b\right)\geq 1 \quad i=1,\ldots,m \tag{3.15}$$

If $(\mathbf{w}^*,b^*)$ is an optimal solution to (3.15) then $\left(\hat{\mathbf{w}},\hat{b}\right)$ is an optimal solution to (3.14), where $\hat{\mathbf{w}}=\frac{\mathbf{w}^*}{||\mathbf{w}^*||}$, $\hat{b}=\frac{b^*}{||\mathbf{w}^*||}$.

Proof. Let $(\mathbf{w},b)$ be a feasible solution to (3.14) and denote the margin achieved by $(\mathbf{w},b)$ by γ then:

$$y_i\left(\langle \mathbf{x}_i,\mathbf{w}\rangle+b\right)\geq\gamma \quad i=1,\ldots,m$$

Since $\gamma=\min_i y_i\left(\langle \mathbf{x}_i,\mathbf{w}\rangle+b\right)$ it holds that:

$$y_i\left(\left\langle \mathbf{x}_i,\frac{\mathbf{w}}{\gamma}\right\rangle+\frac{b}{\gamma}\right)\geq 1 \quad i=1,\ldots,m$$

meaning that $(\frac{\mathbf{w}}{\gamma},\frac{b}{\gamma})$ is a feasible solution to (3.15). Let $(\mathbf{w}^*,b^*)$ be an optimal solution for (3.15). As such, it achieves the minimal norm out of all feasible solutions and specifically it means that:

$$||\mathbf{w}^*||\leq\left|\left|\frac{\mathbf{w}}{\gamma}\right|\right|=\frac{1}{\gamma}||\mathbf{w}||=\frac{1}{\gamma}$$

where the last equality is due to $(\mathbf{w},b)$ being a feasible solution to (3.14) and therefore $\mathbf{w}$ a unit vector. Consider $(\hat{\mathbf{w}},\hat{b})$ achieved from $(\mathbf{w}^*,b^*)$. It follows that for all $i\in[m]$:

$$y_i\left(\langle \mathbf{x}_i,\hat{\mathbf{w}}\rangle+\hat{b}\right)=y_i\left(\left\langle \mathbf{x}_i,\frac{\mathbf{w}^*}{||\mathbf{w}^*||}\right\rangle+\frac{b^*}{||\mathbf{w}^*||}\right)=\frac{1}{||\mathbf{w}^*||}y_i\left(\langle \mathbf{x}_i,\mathbf{w}^*\rangle+b^*\right)\geq\frac{1}{||\mathbf{w}^*||}\geq\gamma$$

with $\hat{\mathbf{w}}$ being a unit vector. Thus, $(\hat{\mathbf{w}},\hat{b})$ achieves a higher or equal objective to (3.14) from any feasible solution, concluding optimality. ■

This means in fact that maximizing the margin is equivalent to minimizing the size of the hyperplane.

Definition 3.3.3 — Quadratic Program. An optimization problem is called a Quadratic Program (QP) if it can be written in the following form:

$$\begin{aligned}\min_{\mathbf{w}\in\mathbb{R}^n} \quad & \tfrac{1}{2}\mathbf{w}^\top Q\mathbf{w}+\mathbf{a}^\top\mathbf{w}\\ \text{such that} \quad & A\mathbf{w}\leq\mathbf{d}\end{aligned}$$

where $Q\in\mathbb{R}^{n\times n},A\in\mathbb{R}^{m\times n},\mathbf{a}\in\mathbb{R}^n,\mathbf{d}\in\mathbb{R}^m$ are fixed vectors and matrices.

The optimization problem written in (3.15) is a **Quadratic Program (QP)** for which there exist efficient solvers. By using them to solve problem (3.15) we can obtain an optimal solution for the Hard-SVM optimization problem.

R But so how is it that minimizing $||\mathbf{w}||^2$ is equivalent to maximizing the margin? Let us denote the width of the total margin (i.e. the sum of margin from both sides) by l, and let x_+ and x_- be the positive- and negative support vectors . To calculate the value of l we will project the vector $x_+ - x_-$ onto the normalized normal $\mathbf{w}$:

$$\begin{aligned} l &= \left\langle x_+ - x_-, \frac{\mathbf{w}}{||\mathbf{w}||} \right\rangle \\ &= (\langle x_+, \mathbf{w} \rangle - \langle x_-, \mathbf{w} \rangle) \, ||\mathbf{w}|| \\ &= (1 - b - (-1 - b)) \, / \, ||\mathbf{w}|| \\ &= 2 / ||\mathbf{w}|| \end{aligned}$$

where support vectors satisfy $y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right)$ and that for positive samples $y_i = 1$ and negative samples $y_i = -1$. This shows how minimizing $||\mathbf{w}||$ maximizes l.

3.3.3 Soft-SVM

The basic assumption of Hard-SVM is that the training sample is *linearly separable*, that is, that the realizability assumption holds. If that is not the case then the optimization problem has no solutions as for any candidate $(\mathbf{w}, b)$ at least one of the constraints $y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b \right) \geq 1$ cannot be satisfied.

However, what if the training sample is *almost* linearly separable? That is, what if most of the samples are linearly separable with only a few violating the constraints by "not too much"? Recall that if $y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b \right) < 0$ then sample $\mathbf{x}_i$ is on the "wrong side" of the hyperplane. This means that:

$$\exists \xi_i > 0 \quad s.t. \quad y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b \right) \geq 1 - \xi_i$$

Therefore, sample $\mathbf{x}_i$ is on the "wrong" side of the **margin** by an amount proportional to ξ_i (Figure 3.9). To allow training samples to violate the constraints "a little", we modify the optimization problem to:

$$\begin{aligned} &\text{minimize} \quad ||\mathbf{w}||^2 \\ &\text{subject to} \quad \begin{cases} y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b \right) \geq 1 - \xi_i & i = 1, \ldots, m \\ \xi_i \geq 0 \quad \wedge \quad \frac{1}{m} \sum_{i=1}^m \xi_i \leq C \end{cases} \end{aligned} \tag{3.16}$$

where $C > 0$ is a constant we specify. The variables $\xi_1, \ldots, \xi_m$ are new auxiliary variables we introduce (sometimes known as slack variables). Notice that the larger we choose C to be, the more violations of margin we allow. On the one hand, we want to allow "noisy" samples to violate the margin, so the hyperplane will ignore them. On the other hand, if we allow too many violations, we lose touch with the training sample and its structure. This is exactly the bias-variance trade-off: the larger C, the more freedom the learner has to "chase after the training sample".

Instead of specifying C directly, we often prefer working with a slightly different optimization problem, where instead of constraining the value of $\frac{1}{m} \sum \xi_i$ we jointly minimize the norm of $\mathbf{w}$ (related to the margin) and the average of ξ_i (corresponding margin violations).

$$\begin{aligned} &\underset{\mathbf{w}, b, \{\xi_i\}}{\text{minimize}} \quad \lambda \, ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \\ &\text{subject to} \quad y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b \right) \geq 1 - \xi_i, \; \xi_i \geq 0 \quad i = 1, \ldots, m \\ &\qquad\qquad \lambda \geq 0 \end{aligned} \tag{3.17}$$

To simplify the above optimization problem let use define the **hinge** loss function:

$$\ell^{hinge}(a) := \max \{0, 1 - a\}, \; a \in \mathbb{R} \tag{3.18}$$

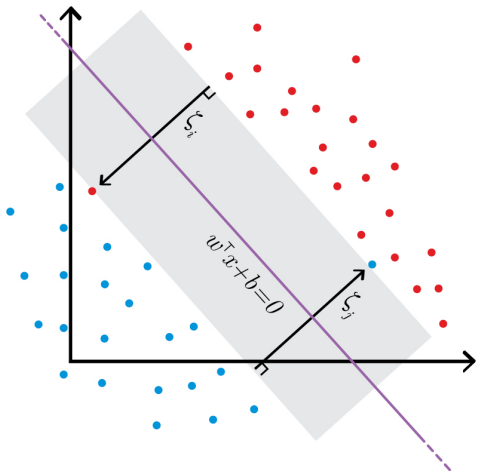


Figure 3.9: *Slack variables of data-points that are on the "wrong" side of the hyper-plane.*

Claim 3.3.3 Given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and hyperplane $(\mathbf{w}, b)$, the Soft-SVM optimization problem (**??**) is equivalent to

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \left(\lambda \, ||\mathbf{w}||^2 + L_S^{hinge}\left((\mathbf{w}, b)\right) \right)$$

where $L_S^{hinge}\left((\mathbf{w}, b)\right) := \frac{1}{m} \sum \ell^{hinge}\left(y_i \cdot \mathbf{x}_i^\top \mathbf{w}\right)$

Proof. Given a specific hyperplane $(\mathbf{w}, b)$ consider the minimization over $\xi_1, \ldots, \xi_m$. Since we defined the auxiliary variables to be nonnegative, the optimal assignment of ξ_i is

$$\xi_i := \begin{cases} 0 & y_i\left(\mathbf{x}_i^\top \mathbf{w} + b\right) \\ 1 - y_i\left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) > 1 & otherwise \end{cases}$$

Thus $\xi_i = \ell^{hinge}\left(y_i\left(\mathbf{x}_i^\top \mathbf{w} + b\right)\right)$ ■

The hyper-parameter λ controls the trade-off between the norm of $\mathbf{w}$ and the violations of margin.

- The larger λ, the less sensitive the solution will be to the term $\frac{1}{m}\sum_{i=1}^m \xi_i$, and will allow more violations.
- The smaller λ, the more sensitive and will allow less violations.

Therefore when we consider the parameter λ in (3.17) (or equivalently C in (3.16)), we are in fact considering different learners within a **family of learners**, where each member of the family is specified by a specific value of λ (or C). These different family members can be placed somewhere along the hypothesis complexity axis. Thus, changing the value of λ (or C) moves us along the bias-variance tradeoff. λ is known as a **regularization parameter**. This topic is covered in **??**.

3.3.4 Learner ID Card

- **Hypothesis class:** the class of non-homogeneous linear separators (3.2)
- **Learning principle used for training:** Maximal margin
- **Computational implementation:** Quadratic Program
- **Interpretability:** Retrieved solution does not provide meaningful insights regarding predictions
- **Family of models:** The Soft-SVM learner provides us with a set of models, indexed by the regularization parameter $\lambda \in [0, \infty)$

- **Storing fitted model:** Fitted model is the vector $\mathbf{w}$ perpendicular to the hyperplane defining the half-space as well as the intercept coordinate
- **Prediction of new sample:** $\hat{y}_{new} := sign\left(\mathbf{x}_{new}^{\top}\mathbf{w}+b\right)$
- **When to use:** By itself this learner should be used as a simple baseline. However, after embedding the data in some high-dimensional space (kernelization) this becomes a powerful learner (**??**)

3.4 Logistic Regression

3.4.1 A Probabilistic Model For Noisy Labels

Let us revisit the model of linear regression. Recall that when assuming Gaussian errors (2.1.4) we modeled the relation $\mathcal{X} \to \mathcal{Y}$ as $\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$ and $\varepsilon \sim \mathcal{N}\left(0, \sigma^2 I_m\right)$. Notice that as ε is a random variable, $\mathbf{y}$ too is a random variable distributing as a multi-variate Gaussian:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{X}\mathbf{w}, \sigma^2 I_m\right) \tag{3.19}$$

Focusing on a pair $(\mathbf{x}_i, y_i)$, we can think of the above as the **conditional probability** of y_i given $\mathbf{x}_i$:

$$p\left(y_i|\mathbf{x}_i, \mathbf{w}\right) = \mathcal{N}\left(y_i|\phi_{\mathbf{w}}\left(\mathbf{x}_i\right), \sigma^2\right) \quad \text{where} \quad \phi_{\mathbf{w}}\left(\mathbf{x}\right) = \mathbf{x}^{\top}\mathbf{w} \tag{3.20}$$

where the notation of $\mathcal{N}\left(y_i|\mathbf{x}_i, \mathbf{w}\right)$ means the probability of observing the response y_i for the feature vector of $\mathbf{x}_i$ and coefficients vector $\mathbf{w}$. We also condition on $\mathbf{w}$ (though is not a random variable) to explicitly state the dependence on the model parameters. In other words, we assumed that each sample $(\mathbf{x}, y)$ is such that the **expected value** of the label y is linear in $\mathbf{x}$. As we are dealing with a regression model and $y_i \in \mathbb{R}$, the support of the random variable $y_i|\mathbf{x}_i, \mathbf{w}$ is $\mathbb{R}$.

Let us adapt the model above to fit for classification problems. We would like to assume that y_i distributes **Bernoulli** with a probability $p\left(\mathbf{x}_i\right)$ that somehow relates with $\mathbf{x}_i$, and captures how likely is y_i of being 1:

$$p\left(y_i|\mathbf{x}_i\right) = Ber\left(y_i|p\left(\mathbf{x}_i\right)\right) \tag{3.21}$$

How shall $p\left(\mathbf{x}_i\right)$ relate with $\mathbf{x}_i$? Unlike the linear regression model, we cannot assume a linear function $\phi_{\mathbf{w}}\left(\mathbf{x}\right) = \mathbf{x}^{\top}\mathbf{w}$ as $\phi_{\mathbf{w}} \in \mathbb{R}$ while $p\left(\mathbf{x}_i\right)$ is restricted to $[0,1]$. Instead, we would like to choose some **link** function $\phi_{\mathbf{w}} : \mathbb{R} \to [0,1]$ that is monotone increasing and maps $(-\infty, \infty)$ bijectively to $(0,1)$. Define the relation to be:

$$p\left(y_i|\mathbf{x}_i, \mathbf{w}\right) = Ber\left(y_i|\phi_{\mathbf{w}}\left(\mathbf{x}_i\right)\right), \quad \phi_{\mathbf{w}} := sigm\left(\mathbf{x}^{\top}\mathbf{w}\right) \tag{3.22}$$

where $sigm$ is the **sigmoid** function, also known as the **logit** function:

$$sigm\left(\mathbf{a}\right) := \frac{e^{\mathbf{a}}}{e^{\mathbf{a}}+1} \tag{3.23}$$

This function is indeed monotone increasing and maps $(-\infty, \infty)$ bijectively to $(0,1)$:

- As $\mathbf{x}^{\top}\mathbf{w} \to -\infty$ then $sigm\left(\mathbf{x}^{\top}\mathbf{w}\right) \to 0$. This means that it is "very unlikely" that the label is 1: $p\left(y_i = 1|\mathbf{x}_i, \mathbf{w}\right) \to 0$.
- As $\mathbf{x}^{\top}\mathbf{w} \to \infty$ then $sigm\left(\mathbf{x}^{\top}\mathbf{w}\right) \to 1$. This means that it is "very likely" that the label is 1: $p\left(y_i = 1|\mathbf{x}_i, \mathbf{w}\right) \to 1$.

R In (3.22) we modeled the logistic regression model for binary classification problems. Notice that the Bernoulli distribution can be seen as a private case of the Multinomial distribution $Multinomial\left(p_1, \ldots, p_K\right)$, $\sum_p i = 1, 0 \leq p_i \leq 1$. We can expand the above logistic regression model to fit multi-classification problems by extending the sigmoidal function to what is known as the softmax function $\sigma\left(\mathbf{a}\right) = e^{\mathbf{a}_i} / \sum_{j=1}^{K} e^{\mathbf{a}_j}$

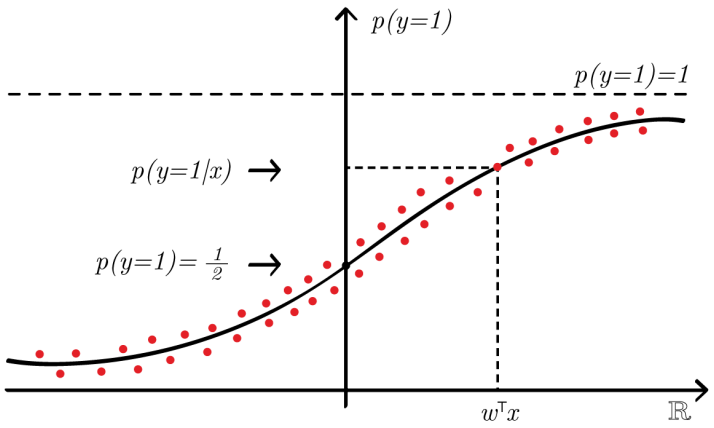


Figure 3.10: Illustration of fitted logit function *for values corresponding to* $\mathbf{x}^\top \mathbf{w}$.

3.4.1.1 The Hypothesis Class

So we would like to define the hypothesis class of logistic regression as:

$$\mathcal{H}_{logistic} := \left\{ h_{\mathbf{w}}(\mathbf{x}) = sigm\left(\mathbf{x}^\top \mathbf{w}\right) | \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{3.24}$$

where $\mathbf{w} \in \mathbb{R}^{d+1}$ since we incorporate the intercept variable into $\mathbf{w}$ (and a zeroth coordinate of 1 to $\mathbf{x}$) similar to the way we did in the linear regression hypothesis class. Notice that the hypotheses are defined $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to [0,1]$ and not $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to \{0,1\}$ as required for classification problems. Since $\{0,1\} \subset [0,1]$, we can use the training sample to select a function in $\mathcal{H}_{logistic}$. This means we will be able to train a model, but how will we predict over new samples? Suppose our learner chose some $h_{\mathbf{w}} \in \mathcal{H}_{logistic}$. As we think of $h_{\mathbf{w}}(\mathbf{x})$ as an estimate of the probability that the label corresponding to $\mathbf{x}$ is 1, we can use it for classification. If $h_{\mathbf{w}}(\mathbf{x})$ is low the label is likely to be 0. If $h_{\mathbf{w}}(\mathbf{x})$ is high, the label is likely to be 1. Choosing some **cutoff** value $\alpha \in [0,1]$, our class prediction will be: $\hat{y} := \mathbb{1}_{h_{\mathbf{w}}(\mathbf{x}) > \alpha}$. To choose a fitting value for α we can calculate the Type-I and Type-II errors (subsection 3.1.1) of the classifier and plot its ROC curve (3.4.4)

3.4.1.2 Learning Via Maximum Likelihood

Once we have defined the logistic regression model (3.22) and hypothesis class (3.24), we would like to come up with a learner. To do so we will use the *maximum likelihood principle*. Recall, that by the maximum likelihood principle, we estimate the parameters (the desired hypothesis) as those that have the highest probability, given the data.

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be our sample of independent observations, assuming that $\mathbf{y}_i \sim Ber(\phi_{\mathbf{w}}(\mathbf{x}))$ where ϕ is the logistic function. Therefore, the likelihood of $\mathbf{w} \in \mathbb{R}^{d+1}$ is:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}|\mathbf{X},\mathbf{y}) &= \mathbb{P}(y_1,\ldots,y_m|\mathbf{X},\mathbf{w}) \\
&= \prod \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \\
&= \prod_{i:y_i=1} \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \cdot \prod_{i:y_i=0} \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \\
&= \prod_{i:y_i=1} p_i(\mathbf{w}) \cdot \prod_{i:y_i=0} (1-p_i(\mathbf{w})) \\
&= \prod p_i(\mathbf{w})^{y_i} (1-p_i(\mathbf{w}))^{1-y_i}
\end{aligned} \tag{3.25}$$

where $p_i(\mathbf{w}) = \phi_{\mathbf{w}}(\mathbf{x}_i)$. Since the log function is monotone increasing we can maximize the log-likelihood $\ell(\mathbf{w}) := \log \mathcal{L}(\mathbf{w})$ instead:

$$\begin{aligned} \ell(\mathbf{w}|\mathbf{X},\mathbf{y}) &= \sum_{i=1}^{m} \left[y_i \log\left(p_i(\mathbf{w})\right) + (1-y_i)\log\left(1-p_i(\mathbf{w})\right) \right] \\ &= \sum_{i=1}^{m} \left[y_i \log\left(\frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1+e^{\mathbf{x}_i^\top \mathbf{w}}}\right) + (1-y_i)\log\left(\frac{1}{1+e^{\mathbf{x}_i^\top \mathbf{w}}}\right) \right] \\ &= \sum_{i=1}^{m} \left[y_i \cdot \mathbf{x}_i^\top \mathbf{w} - \log\left(1+e^{\mathbf{x}_i^\top \mathbf{w}}\right) \right] \end{aligned} \tag{3.26}$$

And therefore, choosing the function $h \in \mathcal{H}_{logistic}$ by applying the maximum likelihood principle means that:

$$\hat{\mathbf{w}} := \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\operatorname{argmax}} \sum_{i=1}^{m} \left[y_i \cdot \mathbf{w}^\top \mathbf{x}_i - \log\left(1+e^{\mathbf{w}^\top \mathbf{x}_i}\right) \right] \tag{3.27}$$

R Instead of deriving the learner using the maximum likelihood principle, we could derive it using the ERM principle with the following loss function: $\ell(h_{\mathbf{w}}) := \log\left(1+\exp\left(-y\left\langle \mathbf{w},\mathbf{x}\right\rangle\right)\right)$. We would have reached the same optimization expression.

3.4.2 Computational Implementation

Now that we have defined the hypothesis class and an optimization problem to find the desired hypothesis, the next step is finding an efficient algorithm to solve it. By working with the logistic function, the resulting log-likelihood expression is **concave** function of the optimization variable $\mathbf{w}$. This means, that instead of solving the maximization problem (3.27) we can solve the minimization of minus the log-likelihood, which is convex. As such, there are general algorithms for finding the minima of such functions.

While there is no closed form for the maximizer $\hat{\mathbf{w}}$, as logistic regression is a very useful learner, there is a custom iterative algorithm that usually converges quickly to $\hat{\mathbf{w}}$. This algorithm is based on **Newton-Raphson** iterations.

3.4.3 Interpretability

One important properly of the logistic regression learner is interpretability both in the sense of which features were important for the model and why was a certain prediction given. When working with $\mathcal{X} = \mathbb{R}^d$, we gather many feature, and might think that some of them are important for prediction while others less. Similar to linear regression, we are able to ask "which features were important" for the model by simply investigating the entries of the fitted coefficients vector $\mathbf{w}$. Features corresponding to coefficients of values close to zero have a small impact on prediction and therefore these features are less important for the model. Features corresponding to coefficients of values far from zero have a large impact on prediction and are therefore important for the model.

Then, for a given sample $\mathbf{x}$, by looking at entries corresponding to important features we can understand why the model predicted $\hat{y}$. If in entries of $\mathbf{x}$ corresponding important features there are large (positive or negative) values, they will have much influence the outcome. If these values are of same sign as of the coefficients then the expression $\mathbf{x}^\top \mathbf{w}$ will be larger, increasing the likelihood of the prediction being 1. If these values are of opposite signs to the coefficients then the expression $\mathbf{x}^\top \mathbf{w}$ will become smaller, decreasing the likelihood of the prediction being 1.

3.4.4 Predictions Over New Samples & The ROC Curve

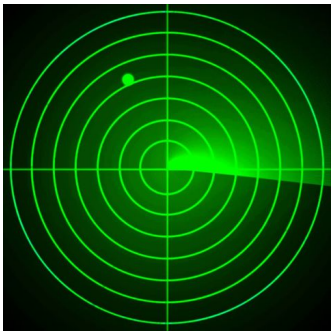
As we will encounter later in this chapter, in many classification scenarios we face the following question: Suppose we trained some classifier $h \in \mathcal{H}$ and derive classifications by the following rule: for some cutoff

value $\alpha \in [0,1]$

$$\widehat{y} := \begin{cases} 1 & h(\mathbf{x}) > \alpha \\ 0 & h(\mathbf{x}) \leq \alpha \end{cases}$$

How do we choose α? There is an important **tradeoff** in the selection of α. If we set α to be very high we are mainly going to predict 0. By doing so we are **less** likely to have false-positives (which is the error we try to avoid at all cost), for which we are pleased. However, at the same time, we are bf more likely to gave false-negatives. So by setting α too high we might have low a FPR but "miss" (misclassify) most of the positive samples. On the other extreme, if we set α to be very low we are mainly going to predict 1. So we will be **more** likely to have false-positives, but at the same time will be **less** likely to have false-negatives. So if we set α too low, we might have a high FPR but "catch" (correctly classify) most of the positive samples. Therefore, we see that changing $\alpha \in [0,1]$ governs some trade-off between the chances of making a Type-I error (false-positives) and correctly classifying positive samples.

This trade-off was first studied during World War II, when radar was invented. The designer of the radar had to choose when to put a green dot on the radar, indicating a target detected there. Sometime radar waves would bounce off back from clouds or birds, and the designer had to choose a **threshold** α. If the radar pulse returning is stronger than α, the radar screen would show a green dot. If weaker than α, no dot. Now, if α is set too low (say $\alpha = 0.1$), the screen would be full of a thousand green dots - since any bird or cloud (with, say, $h(\mathbf{x} = 0.2)$) would be classified as **positive**, a target. So that the radar will be full of false positives, false targets, and will be useless. On the other hand, if α is set too high (say $\alpha = 0.9$) then enemy airplanes (with, say, $h(\mathbf{x} = 0.8)$) will not appear on the screen, since they will be classified by mistake as birds, and the radar again would be useless.



The radar engineers developed a way to visualize this tradeoff, which is still used today in machine learning. After training some linear model (choosing some hypothesis $h \in \mathcal{H}$) we make a grid of values of $\alpha \in [0,1]$. For each value of α we create a classifier by thresholding h at α, and calculate the number of Type-I and Type-II errors the classifier makes over a test sample that was not used for training. We plot a parametric curve of TPR (true positive rate) against FPR (false positive rate) when α is the parameter. This curve is called the **Receiver Operating Characteristic (ROC)** curve. It is continuous, increasing and goes from $(0,0)$ in the FPR-TPR plane (for $\alpha = 0$ we classify everything as negative, so no false positives and not true positives) to $(1,1)$ (for $\alpha = 1$ we classify everything as positive, so false positive rate is 1 - we make every possible Type-I error - and also true positive rate is 1 - we “catch“ all the positive samples).
Convince yourself that if the ROC curve is a linear line from $(0,0)$ to $(1,1)$, the classifier is just a random guess. If a classifier has an ROC curve that is closed to this linear line, it’s a poor classifier. Now convince yourself that if the ROC curve rises sharply from $(0,0)$, for example makes a “jump” to $(FPR = 0.1, TPR = 0.9)$, it’s a good classifier - we are able to correctly detect 0.9 of the positive samples at the price of 0.1 false positive rate.

Plotting the ROC curve of a classifier has a few different uses:

- **Tuning α:** It allows us to see the tradeoff, provided by the classifier, between Type-I errors and correct detection of positive samples, so we can choose the tuning of α we would like to work with for the

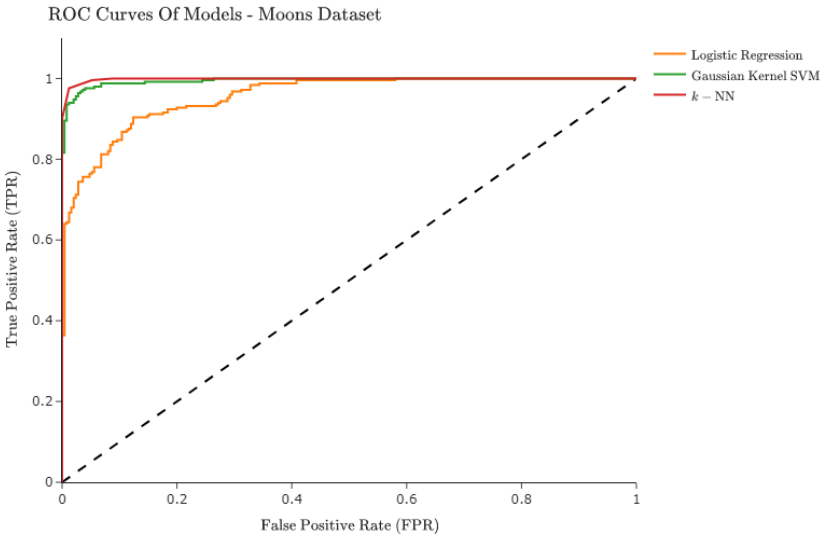


Figure 3.11: ROC Curve *of classifiers fitted over moons dataset.* *Chapter 3 Examples - Source Code*

actual prediction.

- **AUC - Area Under Curve:** A performance measure for the tradeoff itself. This performance measure evaluates the prediction rule h we chose without having to decide on α - it measures the quality of the **tradeoff** provided by h, a tradeoff from which we must choose a specific point in order to actually classify new samples. AUC is simply the **definite integral** of the ROC curve on the segment $[0,1]$ - the area under the AUC curve. As mentioned above, AUC around $1/2$ means that h is poor - more specifically, that the **tradeoff** provided by h is poor. AUC is bounded from above by 1, so an AUC close to 1 means h offers an excellent trade-off, and in this case we expect to be able to find a cutoff α that gives a classifier with very few false-positives and very high detection rate (true positive rate).
- **Comparing candidate rules:** Suppose we have a couple of candidate rules h_1, h_2 (or more). For example, maybe we trained some classifier on the same training sample with different features, or maybe we trained two different types of classifiers over the same data, and we are wondering which one to use. Now we have a problem - we can't turn h_i into an actual classification rule without choosing a cutoff α_i, but would like to compare h_1 to h_2 without committing to a cutoff - to compare the tradeoff offered by h_1 to that offered by h_2. It is very useful here to plot the two ROC curves of h_1 and h_2 on a single axis - and visually compare the tradeoffs they offer.

3.4.5 Multiclass Logistic Regression

To be added

3.4.6 Learner ID Card

- **Hypothesis class:** The composite of the sigmoidal function over the linear functions (3.24)
- **Learning principle used for training:** Maximum likelihood
- **Computational implementation:** Specialized iterative method based on Netwon-Raphson iterations, or a general convex solver. When using a general convex solver we must pay attention to the effective rank of the regression matrix, similar to linear regression. Near-singular regression matrices will lead to numerical instabilities
- **Interpretability:** Given a fitted model we can interprate which features drive the classification as well as understand why a given sample was predicted as it was
- **Family of models:** As seen in **??** it is possible to add regularization terms to control the bias-variance properties of the fitted model

- **Storing fitted model:** Store the regression coefficients vector $\mathbf{w}$
- **Prediction of new sample:** To perform predictions we must specify a thresholding parameter $\alpha \in (0,1)$. Once we chose a value of α then prediction is performed by $\widehat{y}_{new} := \mathbb{1}_{sigm\left(\mathbf{x}_{new}^{\top}\mathbf{w}+b\right)\geq\alpha}$
- **When to use:** The logistic regression learner, especially when adding regularization terms, is a powerful learner. It is always good to try it, especially when classes are more or less balanced.

For the logistic regression learner we have adapted the hypothesis class of linear regression by composing it with the sigmoid function:

$$\mathcal{H}_{logistic} := \left\{ \mathbf{x} \mapsto sigm\left(\mathbf{x}^{\top}\mathbf{w}\right) \;\middle|\; \mathbf{w} \in \mathbb{R}^{d} \right\}$$

Then, we have derived an optimization problem using the maximum likelihood principle (3.27). To computationally implement the learner there are specialized iterative methods based on Newton-Raphson iterations, or general convex solvers. It is important to note that just like linear regression we must pay attention to the effective rank of the regression matrix. Near-singular matrices will cause numerical problems.

Given trained model, we have to specify a threshold parameter α, which will provide some good tradeoff between the FPR and TPR. Once we have specified α prediction of a new sample is given by $\hat{y} := \mathbb{1}_{sigm\left(\mathbf{x}_{new}^{\top}\mathbf{w}\right)\geq\alpha}$.

3.5 Bayes Classifiers

When deriving the logistic regression model, we assumed a probability distribution over the response set $\mathcal{Y}$ and treated the observations as deterministic values influencing the distribution of the response. We could however *assume* that both the response set and the domain set follow some *joint probability distribution* $\mathcal{F}$ over $\mathcal{X} \times \mathcal{Y}$. Under such assumption we are now able to look at the data from two different perspectives. Given the sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, we could first consider $\mathbf{x}_1, \ldots, \mathbf{x}_m$ as fixed and learn $y_i|\mathbf{x}_i$. That is, the distribution of the response given the observation. This is the perspective used in the logistic regression, as well as the linear regression, model. For the second perspective we consider $y_1, \ldots, y_m$ as fixed and ask how do the observations depend on the responses $\mathbf{x}_i|y_i$.

$$\overbrace{f_{Y|X=\mathbf{x}}(y)\cdot f_X(\mathbf{x})}^{\text{Perspective I}} = \overbrace{f_{X,Y}(\mathbf{x},y)}^{\substack{\text{Joint Probability}\\ \text{Distribution}}} = \overbrace{f_{X|Y=y}(\mathbf{x})\cdot f_Y(y)}^{\text{Perspective II}} \tag{3.28}$$

■ **Example 3.4** Consider the classification task of separating images of cats and dogs. Let the domain space be RBG images of 1024-by-1024 pixels and the response set be $\mathcal{Y} := \{\text{cat}, \text{dog}\}$. Further assume there exists some joint probability distribution $\mathcal{F}$ over $\mathcal{X} \times \mathcal{Y}$. Considering the first perspective where we wish to learn the conditional distribution $y_i|\mathbf{x}_i$. By doing so we try to discriminate a given picture being either of cat or of dog. That is, we simply try to understand how to differentiate between these two possibilities. If however we consider the second perspective, we wish to learn the conditional distribution of $\mathbf{x}_i|y_i$. This probability distribution describes what sort of observations might be seen for a given response. That is, what do pictures of cats or of dogs look like. ■

Besides providing insights into the manner in which different samples behave - how do cat pictures look? how do dog pictures look? - what benefit do we get from considering this second perspective? How can it be used for predicting the response of a given sample? To answer this question we use the Bayes' Law of conditional probability. Given a joint probability distribution function $f_{X,Y}$ over the observation $\mathbf{x}$ and response y we can express the conditional distribution $y|\mathbf{x}$ using the conditional distribution $\mathbf{x}|y$:

$$f_{Y|X=\mathbf{x}}(y) = \frac{f_{X|Y=y}(\mathbf{x})\cdot f_Y(y)}{f_X(\mathbf{x})}$$

From this relation we are able to derive the Bayes Optimal classifier.

Definition 3.5.1 Let $f_{\mathcal{D}}$ be a joint probability distribution function over $\mathcal{D} := \mathcal{X} \times \mathcal{Y}$. The *Bayes Optimal Classifier* is defined as

$$h^{Bayes}(\mathbf{x}) := \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, f_{Y|X=\mathbf{x}}(y)$$

That is, we predict the response that (given the observation) achieves the highest probability. Since we are searching for a maximizer, we can use Bayes' Law to express the Bayes Optimal classifier as

$$h^{Bayes}(\mathbf{x}) := \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, f_{Y|X=\mathbf{x}}(y) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, \frac{f_{X|Y=y}(\mathbf{x}) f_Y(y)}{f_X(\mathbf{x})} \overset{(*)}{=} \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, f_{X|Y=y}(\mathbf{x}) f_Y(y)$$

where $(*)$ is because given $\mathbf{x}$, $f_X(\mathbf{x})$ is constant over the different values of y. Notice that we have already previously seen this sort of algorithm (in the context of regression) when mentioning Bayesian statistics (subsubsection 1.1.3.1). The conditional $f_{X|y=y}$ is the *likelihood function* whose maximizer is the maximum likelihood estimator. It assesses the probability of observing $\mathbf{x}$ given that the response was y. The marginal distribution $f_Y(y)$ is the *prior* distribution and it assesses the *a-priori* probability (i.e belief) of receiving a sample with such a response. Therefore, the Bayes Optimal classifier weights the MLE according to the different class probabilities. The conditional distribution $f_{Y|X=x}$ is called the *a-posteriori* distribution - the probability of the response *after* (i.e give) observing $\mathbf{x}$. This estimator is called the *Maximum A-Posteriori Estimator* (MAP) and is often denoted as $\hat{y}^{MAP}$.

What is left to explain where does the "Optimal" in "Bayes Optimal" comes from. For that, let us consider the misclassification loss function. It therefore holds that the Bayes Optimal classifier achieves the minimal misclassification risk out of all possible classifiers.

Theorem 3.5.1 Let $f_{X,Y}$ be a joint probability distribution function over $\mathcal{X} \times \mathcal{Y}$ for $\mathcal{Y} = [k], k \in \mathbb{N}$. The Bayes optimal classifier $h^{Bayes}(\mathbf{x}) = \operatorname{argmax}_{y \in \mathcal{Y}} f_{Y|X=\mathbf{x}}(y)$ is the optimal classifier with respect to the misclassification error. Namely that for any hypothesis $h : \mathcal{X} \to \mathcal{Y}$ it holds that $L_{\mathcal{D}}(h^{Bayes}) \leq L_{\mathcal{D}}(h)$.

Proof. Let $h : \mathcal{X} \to \mathcal{Y}$ be an hypothesis. The probability of h misclassifying a sample $\mathbf{x}$ is given by:

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}[h(\mathbf{x}) \neq y] = \sum_{y \neq h(\mathbf{x})} f_{Y|X=\mathbf{x}}(y) \cdot f_X(\mathbf{x})$$

Thus, for the misclassification risk of h it holds that:

$$\begin{aligned} L_{\mathcal{D}}(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[h(\mathbf{x}) \neq y] &= \int_{\mathcal{X}} \sum_{y \neq h(\mathbf{x})} f_{Y|X=\mathbf{x}}(y) \cdot f_X(\mathbf{x}) d\mathbf{x} \\ &\geq \int_{\mathcal{X}} \sum_{y \neq h^{Bayes}(\mathbf{x})} f_{Y|X=\mathbf{x}}(y) \cdot f_X(\mathbf{x}) d\mathbf{x} = L_{\mathcal{D}}(h^{Bayes}) \end{aligned}$$

where the inequality holds since the Bayes optimal classifier is defined to select the label y that maximizes the conditional probability $y|\mathbf{x}$. ■

It is important to note that in reality we do not know the underlying distribution $\mathcal{D}$, to whom all we have is a mere window in the form of the dataset. Therefore, we cannot program an algorithm that would find the maximizer of the posterior distribution. As such, we must think of the Bayes optimal classifier as an Oracle - a "black box" entity capable of solving the problem of finding the maximizer of the posterior.

If however implementing the Bayes optimal classifier is not feasible is it of any practical use? Though we do not know $\mathcal{D}$ we might still have some prior insights into the manner by which the data behaves. For example we might have some prior knowledge regarding the prevalences of different labels or we might assume that for different responses different feature values are more likely to be observed. We can incorporate these insights into the derived model. Then, if these assumptions hold, we might be able to provide a realization of the Bayes optimal classifier. Below are two examples for such realizations.

3.5.1 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) algorithm is a realization of the Bayes Optimal classifier. In this model we assume that samples of different labels have different Gaussian distributions. Explicitly, we assume the following generative model:

1. Each sample selects a label $y_i = Ber(\pi)$, $\pi \in [0,1]$.
2. Then, the sample itself is drawn from the conditional probability of $X|Y$ where X denotes the random variable of sampling some samples $X = \mathbf{x}$ and Y denotes the random variable of $Y = y$, $y \in \{0,1\}$. The distribution used to model $X|Y$ is a Gaussian distribution where each label is characterized by a different mean vector $\mu_0, \mu_1 \in \mathbb{R}^d$ but the same covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$.

Namely, for any $i \in 1,\ldots,m$ we assume that:

$$\begin{aligned} y_i &\sim Ber(\pi) \\ \mathbf{x}_i|y_i &\sim \mathcal{N}(\mu_{y_i}, \Sigma) \end{aligned} \tag{3.29}$$

Under these assumptions, predicting the class of a new sample is done by simply using the Bayes Law over the Bayes Optimal classifier to get:

$$\hat{y}(\mathbf{x}) := \underset{y}{\text{argmax}}\ \mathbb{P}(y|\mathbf{x}) = \underset{y}{\text{argmax}}\ \frac{\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)}{\mathbb{P}(x)}$$

Claim 3.5.2 Let $\mathcal{D}$ be a joint distribution over $\mathcal{X}\times\mathcal{Y}$ with the conditional distribution

$$y|x \sim \mathcal{N}(\mu_k, \Sigma) \quad k \in \{0,1\}$$

Denote $\mathbb{P}(y=k) = \pi_k$ where $\pi_i \leq 0$, $\sum \pi_i = 1$. Then the Bayes Optimal classifier is given by:

$$\widehat{y}(\mathbf{x}) := \underset{k\in\{0,1\}}{\text{argmax}}\ a_k^\top \mathbf{x} + b_k, \quad a_k := \Sigma^{-1}\mu_k, b_k := -\frac{1}{2}\mu_k\Sigma^{-1}\mu_k$$

Proof. We begin with expressing $\mathbb{P}(y=k|\mathbf{x})$. By applying Bayes Law then:

$$\mathbb{P}(y=k|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y=k)\mathbb{P}(y=k)}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{x}|y=k)\mathbb{P}(y=k)}{\sum_{k'\in\{0,1\}}\mathbb{P}(\mathbf{x}|y=k')\mathbb{P}(y=k)} = \frac{\pi_k\mathcal{N}(\mathbf{x}|\mu_k,\Sigma)}{\sum_{k'\in\{0,1\}}\pi_{k'}\mathcal{N}(\mathbf{x}|\mu_k',\Sigma)}$$

Next, from the PDF of a multivatiate Gaussian (**??**) then:

$$\begin{aligned} \mathbb{P}(y=k|\mathbf{x}) &= \frac{\pi_k \cdot \frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^\top\Sigma^{-1}(\mathbf{x}-\mu_k)\right)}{\sum_{k'}\pi_{k'}\cdot\frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{k'})^\top\Sigma^{-1}(\mathbf{x}-\mu_{k'})\right)} \\ &= \frac{\pi_k \cancel{\frac{1}{Z}}\exp\left(\cancel{-\frac{1}{2}\mathbf{x}^\top\Sigma^{-1}\mathbf{x}} + \mathbf{x}^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k\right)}{\sum_{k'}\pi_{k'}\cancel{\frac{1}{Z}}\exp\left(\cancel{-\frac{1}{2}\mathbf{x}^\top\Sigma^{-1}\mathbf{x}} + \mathbf{x}^\top\Sigma^{-1}\mu_{k'} - \frac{1}{2}\mu_{k'}\Sigma^{-1}\mu_{k'}\right)} \\ &= \frac{\pi_k\exp\left(\mathbf{x}^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k\right)}{\sum_{k'}\pi_{k'}\exp\left(\mathbf{x}^\top\Sigma^{-1}\mu_{k'} - \frac{1}{2}\mu_{k'}\Sigma^{-1}\mu_{k'}\right)} \end{aligned}$$

for $Z := \sqrt{(2\pi)^d|\Sigma|}$ the Gaussians' normalization factor. Notice, that as we assume the classes are generated from Gaussians with the same covariance matrix $\frac{1}{Z}$ does not depend on k. Denote $a_k := \Sigma^{-1}\mu_k, b_k := -\frac{1}{2}\mu_k\Sigma^{-1}\mu_k$ and we conclude that:

$$\mathbb{P}(y=k|\mathbf{x}) \propto \exp\left(a_k^\top\mathbf{x} + b_k\right) \quad \Longrightarrow \quad \hat{y}(\mathbf{x}) = \underset{k\in\{0,1\}}{\text{argmax}}\ a_k^\top\mathbf{x} + b_k$$

■

The claim above, besides showing that under the LDA assumptions we are dealing with a Bayes classifier, also tells us something about the classifier learned. Looking at the expression derived from the assumptions, we see that the classification is in fact done by some **linear separator/discriminant**.

It can be shown, by taking the $\log - likelihood$ ratio between the likelihood for being classified for a class divided the likelihood for being classified for the other class, that the decision boundary between the classes is linear, similar to Figure 3.12.

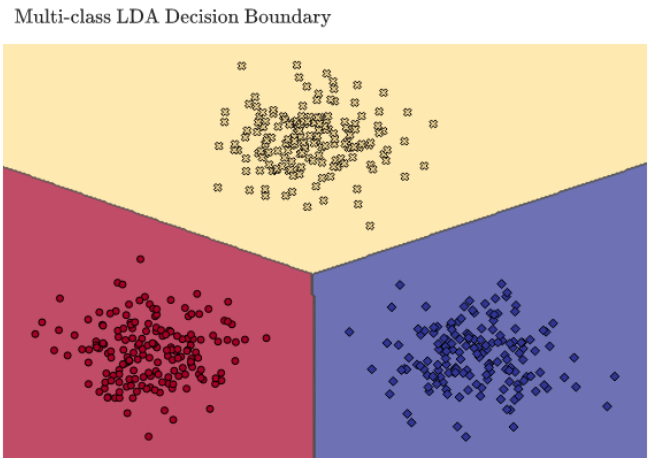


Figure 3.12: LDA Decision Boundaries *for a multiclass setup of three Gaussians. Chapter 3 Examples - Source Code*

Learning A LDA Classifier

So, in order to predict using an LDA classifier we need to know the class probabilities π, the Gaussian centers $\{\mu_i\}$ and the covariance matrixmeans Σ. To do so we derive the maximum likelihood estimators. Let us generalize the binary LDA model (i.e. of classification) to a multiclassification LDA model.

Definition 3.5.2 Let $\Omega = \{1,\ldots,K\}$ for $K \in \mathbb{N}$ be a sample space. A random variable $X : \Omega \to [0,1]$ follows a *Multinomial* distribution with parameter $\pi \in [0,1]^K, \sum \pi_i = 1$ if $\mathbb{P}(X = j) = \pi_j,\ j \in [K]$. We denote $X \sim \text{Mult}(\pi)$.

Then, the LDA model assumptions are:

$$\begin{aligned} y &\sim \text{Mult}(\pi) \\ x|y = k &\sim \mathcal{N}(\mu_k, \Sigma) \end{aligned} \tag{3.30}$$

Given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ then the likelihood is given by:

$$\begin{aligned} \mathcal{L}(\Theta|\mathbf{X},\mathbf{y}) &= f_{X,Y|\Theta}\left(\{(\mathbf{x}_i,y_i)\}_{i=1}^m\right) \\ &\overset{iid}{=} \prod_{i=1}^m f_{X,Y|\Theta}(\mathbf{x}_i,y_i) \\ &= \prod_{i=1}^m f_{X|Y=y_i}(\mathbf{x}_i) \cdot f_{Y|\Theta}(y_i) \\ &= \prod_{i=1}^m \mathcal{N}(\mathbf{x}_i|\mu_{y_i},\Sigma) \cdot \text{Mult}(y_i|\pi) \end{aligned}$$

Since the log transformation is monotonous increasing finding the maximizer of the likelihood is equivalent to

finding the maximizer of the log-likelihood.

$$\begin{aligned}
\ell(\Theta|\mathbf{X},\mathbf{y}) &= \log\left(\textstyle\prod_i \mathcal{N}(\mathbf{x}_i|\mu_{y_i},\Sigma)\cdot \text{Mult}(y_i|\pi)\right) \\
&= \textstyle\sum_i \log\left(\mathcal{N}(\mathbf{x}_i|\mu_{y_i},\Sigma)\right) + \log\left(\text{Mult}(y_i|\pi)\right) \\
&= \textstyle\sum_i \log\left(\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_{y_i})^\top\Sigma^{-1}(\mathbf{x}_i-\mu_{y_i})\right)\right)+\log(\pi_{y_i}) \\
&= \textstyle\sum_i \log(\pi_{y_i}) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\mathbf{x}_i-\mu_{y_i})^\top\Sigma^{-1}(\mathbf{x}_i-\mu_{y_i}) \\
&= \textstyle\sum_k \left[n_k\cdot\log(\pi_k) - \frac{1}{2}\sum_i(\mathbf{x}_i-\mu_k)^\top\Sigma^{-1}(\mathbf{x}_i-\mu_k)\right] - \frac{md}{2}\log(2\pi) - \frac{m}{2}\log|\Sigma|
\end{aligned}$$

for $n_k = \sum_i \mathbb{1}_{y_i=k}$. To find the maximizers we derive with respect to the different parameters $\{\pi_k\},\{\mu_k\},\Sigma$ and equate to zero. However, before doing so recall the constraint on π: $\pi\in[0,1]^K$, $\sum_k \pi_k = 1$. To solve the optimization problem with the constraint we use the Lagrange Multipliers method. Since the constraint is $\sum_k \pi_k = 1 \iff \sum_k \pi_k - 1 = 0$, we define the function $g(\pi) = \sum_k \pi_k - 1$ and the Lagrangian

$$\mathcal{L} = \ell(\Theta|\mathbf{X},\mathbf{y}) - \lambda g(\pi)$$

Now, we derive with respect to each of the parameters including λ. Beginning with the class probabilities then:

$$\frac{\partial\mathcal{L}}{\partial\pi_k} = \frac{\partial}{\partial\pi_k}\ell(\Theta|\mathbf{X},\mathbf{y}) - \lambda\frac{\partial}{\partial\pi_k}g(\pi) = \frac{n_k}{\pi_k} - \lambda = 0 \quad\Rightarrow\quad \pi_k = \frac{n_k}{\lambda} \tag{3.31}$$

To find the value of λ we replace π in the constraint with the expression found in (3.31).

$$1 = \sum_k \pi_k = \sum_k \frac{n_k}{\lambda} \quad\iff\quad \lambda = m$$

and therefore, the MLE of the class probabilities are $\hat{\pi}_k^{MLE} = \frac{n_k}{m}$. To find the MLE of the Gaussian parameters notice that the log-likelihood above is identical to the one derived of a single Gaussian while considering only samples sharing the same label. As such

$$\hat{\mu}_k^{MLE} = \frac{1}{n_k}\sum_i \mathbb{1}_{y_i=k}\mathbf{x}_i, \qquad \hat{\Sigma}_k^{MLE} = \frac{1}{n_k}\sum_i \mathbb{1}_{y_i=k}\left(\mathbf{x}_i-\hat{\mu}_k^{MLE}\right)\left(\mathbf{x}_i-\hat{\mu}_k^{MLE}\right)^\top$$

Looking closely at the expressions derived we in fact realize that the MLE predicts the values proportional to what is found in the training set.

The covariance estimator described in 3.33 is the one derived when equating the derivative with respect to Σ to zero. This is a *biased* estimator. The unbiased estimator for the covariance matrix is given by multiplying by $\frac{1}{m-2}$ instead of $\frac{1}{m}$:

$$\hat{\Sigma}^{MLE} := \frac{1}{m-2}\sum_y\sum_{i:y_i=y}\left(\mathbf{x}_i-\hat{\mu}_y^{MLE}\right)\left(\mathbf{x}_i-\hat{\mu}_y^{MLE}\right)^\top$$

This covariance matrix expression is known as the *pooled covariance* and it accounts for the use of an estimator of $\hat{\mu}_j$ rather the true parameter μ_j. In the general case of multi-classification the multiplication factor of the unbiased estimator is $\frac{1}{m-K}$ where K is the number of classes.

3.5.2 Quadratic Discriminant Analysis

In the LDA algorithm we assumed the data is generated from a set of Gaussians, differing in their mean but sharing the same covariance matrix (3.29). The Quadratic Discriminant Analysis algorithm allows different covariance matrices. That is, for any $i \in 1,\ldots,m$ we assume that:

$$\begin{aligned} y_i &\sim Ber(\pi) \\ x_i|y_i &\sim \mathcal{N}(\mu_{y_i}, \Sigma_{y_i}) \end{aligned} \tag{3.32}$$

By enabling different covariance matrices the quadratic expression (in $\mathbf{x}$) of $\mathbb{P}(y=k|\mathbf{x})$ does not cancel out. This in turn causes the decision boundaries between classes to be quadratic rather than linear. In both Figure 3.12 and Figure 3.13 the same data was used to fit either the LDA or the QDA models. We can see that while the decision boundaries of the LDA fit (Figure 3.12) are linear, in the case of QDA (Figure 3.13) we get curved (quadratic) boundaries.

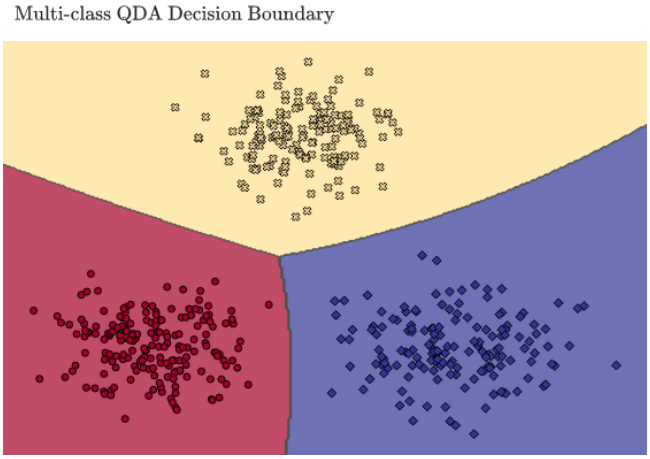


Figure 3.13: QDA Decision Boundaries *for a multiclass setup of three Gaussians. Chapter 3 Examples - Source Code*

Learning A QDA Classifier

Fitting a QDA classifier is very similar to the process of fitting a LDA classifier. The difference is in the estimation of the covariance matrices. In this case we fit a different covariance matrix for each class based on the samples of the class:

$$\hat{\Sigma}_y^{MLE} := \frac{1}{m_y} \sum_{i:y_i=y} \left(\mathbf{x}_i - \hat{\mu}_y^{MLE}\right)\left(\mathbf{x}_i - \hat{\mu}_y^{MLE}\right)^\top \tag{3.33}$$

3.6 Nearest Neighbors

Nearest Neighbors classifiers are a popular, simple and effective learner in which we predict a sample's response based on a set of "nearest" training samples. This classifier is **not** based on the paradigm of a hypothesis class and learning principle. It is a *model free* learner and has no training stage. Instead, when given a training set, we store it in some manner. Then, when we are given a new sample to predict its response we "simply" find the subset of training samples nearest to the new sample, with respect to some measure of distance, and make a prediction based on the responses of those neighbor samples.

R This family of leaners are part of a wider *graph-based approach* for learning. In this approach we first define some graph structure over the samples - forming the nodes of the graph. Then, using the constructed graph we perform training and prediction. The different learners differ in how to define edges in the graph; are they weighted or not? how to transition between nodes; and how is this structure used for training and prediction. Another graphed-based approach that will be seen later is of Spectral Clustering **??**.

3.6.1 Prediction Using k-NN

Let us begin with the simplest form of k-NN. The first step is to determine two hyper-parameters required by the algorithm: an integer k (the number of neighbors to use) and a distance function $\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$. We can decide for example to use the square Euclidean norm $\rho(\mathbf{x}_1, \mathbf{x}_2) := ||\mathbf{x}_1 - \mathbf{x}_2||^2$ or a weighted square norm giving different importance levels to different features $\rho(\mathbf{x}_1, \mathbf{x}_2) := \sum \omega_i((\mathbf{x}_1)_j - (\mathbf{x}_2)_j)^2$.

Then, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ the prediction is done as follows:

Algorithm 2 k-NN

procedure k-NN(k, ρ, S,$\mathbf{x}'$)▷ Where k, ρ are the pre-determined hyper-parameters, S the training set and $\mathbf{x}'$ the sample to predict for

Compute distance from $\mathbf{x}'$ with respect to ρ: $\forall \mathbf{x} \in S \quad d_{\mathbf{x}} := \rho(\mathbf{x}', \mathbf{x})$.

Denote $\pi = (\pi_1, \ldots, \pi_m)$ the permutation of $(1, \ldots, m)$ such that $d_{\mathbf{x}_{\pi_1}} \leq \ldots \leq d_{\mathbf{x}_{\pi_m}}$

Select k nearest samples $\mathbf{x}_{\pi_1}, \ldots, \mathbf{x}_{\pi_k}$ and predict by majority vote:

$$\hat{y} := \underset{y \in \{0,1\}}{\operatorname{argmax}} \sum_{i=1}^{k} \mathbb{1}_{y_{\pi_i} = y}$$

return $\hat{y}$

end procedure

3.6.2 Selecting Value of k Hyper-Parameter

A very important aspect in k-NN is the chosen value of k. Though methods for determining the "right" k will be discussed in future chapters, let us dwell on a few cases:

- $k = 1$**:** The test point is given the label of the single nearest neighbor in the training set. Such classifier has a very low bias but very high variances.
- $k = m$**:** The classifier predicts a single label for any given test sample, regardless to its values. It will predict the majority vote of the training set labels. In this case the bias is very high and the variance is zero.

As we change k we change the bias-variance tradeoff, with larger values of k creating simpler models while smaller values of k creating more complex models (Figure 3.14).

3.6.3 Computational Implementation

Implementing a k-nearest-neighbors classifier is very easy on small datasets, but becomes computationally challenging (either in terms of execution time or in terms of space) when d and/or m are large. There are generally three types of implementation approaches:

- **Brute force implementation:** We keep the entire training sample S in storage during the entire prediction process. For each new test sample $\mathbf{x} \in \mathbb{R}^d$ we calculate $\rho(\mathbf{x}, \mathbf{x}_i)$ $i \in [m]$ and partially sort to find the k smallest distances.

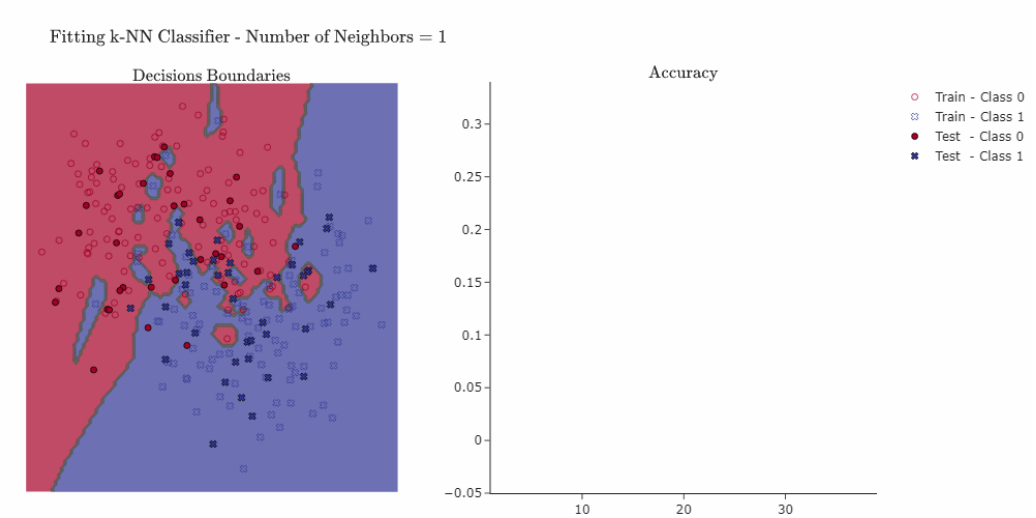


Figure 3.14: Decision Boundaries of k-NN: *Fitting model over dataset for different values of k.*
Chapter 3 Examples - Source Code

Suppose ρ is the Euclidean distance. What are the computational costs of prediction? As the sample space is $\mathbb{R}^d$ computing the distance between two points is $\mathcal{O}(d)$. Doing so for all points in the dataset is $\mathcal{O}(dm)$. Next we want to retrieve the k nearest train samples. If $k \ll m$ we can retrieve k times the sample of minimal distance (without repeating previously selected samples) in a time complexity of $\mathcal{O}(km)$. However, if $k \approx ....$ then it is more computationally efficient to sort all distances and then select the k minimal. Lastly, summation over selected samples is done in $\mathcal{O}(k)$. All together the time complexity of such approach is $\mathcal{O}(dm+km)$. In terms of space complexity we must store distances of all training samples and thus $\mathcal{O}(m)$.

- **Exact nearest neighbors search with preprocessed data structure:** Depending on the selection of ρ, we could pre-process the training sample and construct a special data structure. After doing so in the training step, we can use this data structure to quickly locate the k nearest neighbors of a given test sample. In the case of ρ being the Euclidean distance we could you an algorithm such as *kd-tree*.

- **Fast randomized nearest neighbors search:** Beyond the scope of this course.

3.6.4 Learner ID Card

- **Hypothesis class:** This is a "model free" learner and therefore has no hypothesis class
- **Learning principle used for training:** There is no training phase for this learner
- **Computational implementation:** Different strategies for calculating the nearest neighbors. Simplest method is by brute force nearest neighbor search
- **Interpretability:** We do not know why s given sample was predicted as it was. We can only explain near what other training samples it is
- **Family of models:** Indexed according to the hyper-parameter k
- **Storing fitted model:** Must store the entire training sample or some preprocessed data structure
- **Prediction of new sample:** Find the k samples in the training set closest (with respect to the used metric) to the given sample. Predict based on the majority vote of these samples
- **When to use:** When implementation is computationally feasible try this model.

3.7 Decision Trees

Decision Trees are classification and regression methods by which we partition the sample space into disjoint parts. Then, given such a partition, the response of our classification (or regression) is computed based on the training samples in the partition of the observation in question. These methods are very intuitive and yet capture many interesting aspects of learning. We will discuss some of these aspects in details in the following chapter.

To this day, one of the more powerful classification and regression algorithms is what is called: Classification And Regression Trees (CART) Random Forest. It uses the power of committee decisions over the basic decision trees to achieve very good performances. Some of the aspects of this algorithm will be discussed in later chapters.

3.7.1 Axis-Parallel Partitioning of $\mathbb{R}^d$

Earlier in this chapter we have discussed two classifiers that use piecewise-constant prediction rules: half-spaces and SVM. For both, the hypothesis class consisted of half-spaces where prediction was determined by position of sample with respect to the hyper-plane. For decision trees we will describe a more complicated piecewise-constant prediction rule (more complex hypotheses). Let us consider a rule that partitions the sample space $\mathbb{R}^d$ into **axis-parallel boxes, or "hyper-rectangles"** where each box is associated with labels 1 or -1. The learner's task would be to use the training sample to "chop" the samples space $\mathcal{X}$ int a disjoint union of axis-parallel boxes, and to assign a class prediction to each box.

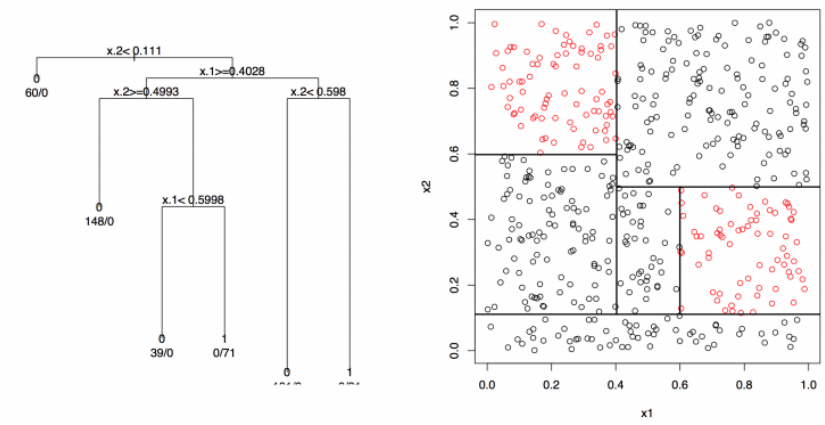


Figure 3.15: *Decision tree and induced partitioning of $\mathbb{R}^2$ sample space*

To make out hypothesis $\mathcal{H}_{CT}$ class smaller and simpler, we will focus on disjoint unions of boxes that are obtained by iteratively splitting an existing box into two smaller boxes along one of the axes:

- We start with the whole sample space $\mathbb{R}^d$.
- By selecting some coordinate $i_1 \in [d]$ and some value $t_1 \in \mathbb{R}$ we split R^d into two axis-parallel "boxes" (half-spaces). We obtain:

$$B_1^+ = \left\{ \mathbf{x} \in \mathbb{R}^d | \mathbf{x}_{i_1} > t_1 \right\}, \quad B_1^- = \left\{ \mathbf{x} \in \mathbb{R}^d | \mathbf{x}_{i_1} \leq t_i \right\}$$

- Next, by focusing of some previously split "box" B_j^s for $s \in \{-,+\}$, we can again select some coordinate $i_{j+1} \in [d]$ and some splitting value $t_{j+1} \in \mathbb{R}^d$ to obtain B_{j+1}^-, B_{j+1}^+. Notice that B_{j+1}^- and B_{j+1}^+ are disjoint, and if following this procedure then by induction they are also disjoint from any other obtained box.

Note that the partitions obtained this way are special - most partitions of $\mathbb{R}^d$ into axis-aligned boxes are not Tree Partitions. Namely, cannot be constructed by such a top-down iterative chopping procedure.

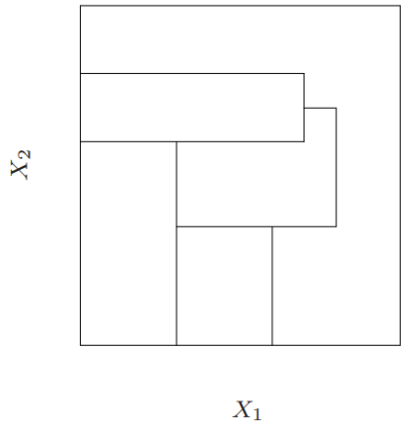


Figure 3.16: *Partitioning $\mathbb{R}^2$ into axis-aligned boxes not describing a tree partition*

3.7.2 Classification & Regression Trees

The hypothesis class $\mathcal{H}_{CT}$ we will consider consists of piecewise-constant functions, that assign a class prediction (1 or 0) to each box in a Tree Partition. Unless we restrict it somehow, the class contains all piecewise-constant functions supported on all Tree Partitions of $\mathbb{R}^d$ (to any number of boxes). Formally, for a Tree Partition $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$ of $\mathbb{R}^d$ into N boxes, and label assignments $c_j \in \{0,1\}$ $(j = 1,\ldots,N)$ assigning label c_j to box B_j, the hypothesis $h \in \mathcal{H}_{CT}$ is a function $h : \mathbb{R}^d \to \{0,1\}$ defined by

$$h(\mathbf{x}) \coloneqq \sum_{j=1}^{N} c_j \mathbb{1}_{\mathbf{x}\in B_j}$$

■ **Example 3.5** Consider the following scenario: suppose someone comes into a hispital emergency room. The first step of triage is to determine - fast - whether they are in a life-threatening medical emergency, or else they can wait in line and receive treatment in a little while. The triage uses a sequence of yes/no questions, such as: Is the patient conscious yes/no?

- If not conscious: classify as **emergency**
- If patient is conscious: is the patient's pulse < 40 beats per minute?
  - If yes (pulse < 40): classify as textbfemergency
  - If no, (pulse ≥ 40): is the patient's pulse > 130 beats per minute?
    - If yes (pulse > 130): is the patient's systolic blood pressure < 80 mmHg?
      - If yes classify as **emergency**
      - If not, is the patient's systolic blood pressure > 140 mmHg? If yes, classify as **emergency**. Otherwise, classify as **no emergency**
    - If not classify as **no emergency**

This is a decision tree that uses three features: conscious (a binary categorical feature), pulse (a numerical feature) and blood pressure (also a numerical feature). See if you can we write a diagram for this decision tree in the shape of a tree, where every node is a question, and every leaf is a decision / classification. The root of the tree is the first question ("conscious yes/no?"). Now observe that every function in our Classification Trees hypothesis class $\mathcal{H}_{CT}$ is equivalent to a decision tree. In the notations of the generic example above, the first question is: "$x_{i_1} > t_1$-yes/no?". If yes, we ask the second question "$x_{i_{2,1}} > t_{2,1}$ - yes/no?". If not, we ask

the second question: "$x_{i_{2,2}} > t_{2,2}$ - yes/no?". And so on, until there are no more splits and we have reached a box over which the function in $\mathcal{H}_{CT}$ is constant. If the constant value is 1, we classify / predict class 1. If the constant value is 0, we predict class 0. This is why our hypothesis class is called - classification trees ■

3.7.3 Growing a Classification Tree

Having defined our hypothesis class, the next question is what learning principle to use. That is, how shall be select $h_s \in \mathcal{H}_{CT}$ based on the training sample S. Suppose we have already obtained a Tree Partition of $\mathbb{R}^d$ is some manner, that consists of N disjoint boxes $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$. Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ be our taining set and denote the predicted label assigned to box B_j by $\hat{y}(B_j) \in \{0,1\}$. As such, the number of misclassification errors that are incurred by the training sample that fall inside B_j is

$$\sum_{\mathbf{x}_i \in B_j} \mathbb{1}\left[y_i \neq \hat{y}(B_j)\right]$$

Let us begin learning by applying the ERM principle. Denote the fraction of misclassified samples with label $y \in \{0,1\}$ in some box B by:

$$\ell_S(B,y) \coloneqq \frac{1}{|B|} \sum_{\mathbf{x}_i \in B} \mathbb{1}\left[y_i \neq y\right]$$

The label that will minimize the empirical risk for those training samples in box B is the **majority vote** over the labels. So, for any sample falling in box B we would predict

$$\hat{y}_S(B) \coloneqq \underset{y \in \{0,1\}}{\operatorname{argmin}} \ \ell_S(B,y)$$

Applying over the entire Tree Partition, minimizing the empirical risk is achieved by labeling box B_j with $\hat{y}_S(B_j)$, $j \in [N]$. Therefore, for a given training set S, every Tree Partition corresponds with a unique label assignment, and as such, a unique classification tree $h \in \mathcal{H}_{CT}$ that minimizes the empirical risk. It seems therefore, that finding the desired ERM tree is done by solving

$$h^* \coloneqq \underset{h \in \mathcal{H}_{CT}}{\operatorname{argmin}} \ L_S(h)$$

where $L_S(h)$ is the misclassification error of h over S. Looking back at the described procedure is it therefore possible to describe which tree would minimize the empirical risk and therefore be selected (for any training set S)? Consider the tree where the number of leaves is $|S|$ and each sample is in a box containing only itself. Following the prediction rule we devised, such a tree would achieve $L_S(h) = 0$. Though we achieved the lowest possible empirical risk, this tree will fail to generalize to new samples. To cope with this problem we should limit the number of levels in the classification tree (equivalent for limiting number of leaves). Denote $\mathcal{H}_{CT}^k$ the hypothesis class of all tree partitions with at most k levels. Now, we will choose k and then using the ERM principle return

$$h^* \coloneqq \underset{h \in \mathcal{H}_{CT}^k}{\operatorname{argmin}} \ L_S(h) \tag{3.34}$$

Selecting A Value For k:

Note that by adding the hyper-parameter k we now have *a family of hypothesis classes*, one for each value of k. The value of k controls the size of the hypothesis class and therefore controls the bias-variance tradeoff (Figure 3.17):

- For small values of k, the hypothesis class is smaller, containing trees of smaller sizes. Therefore the ERM learner will have a **higher** bias as it can only select simple Tree Partitions. It will also have a **lower** variance: as the boxes are very large, the labels assigned to each box are based on a majority vote of typically many training samples. Therefore changing a few training samples will barely change the selected hypothesis.
- For large values of k, the hypothesis class is much more complex, with more "specialized" trees in it. Therefore the ERM learner will have a **lower** bias and **higher** variance.

Later in the course we will introduce a few methods for selecting the value of k.

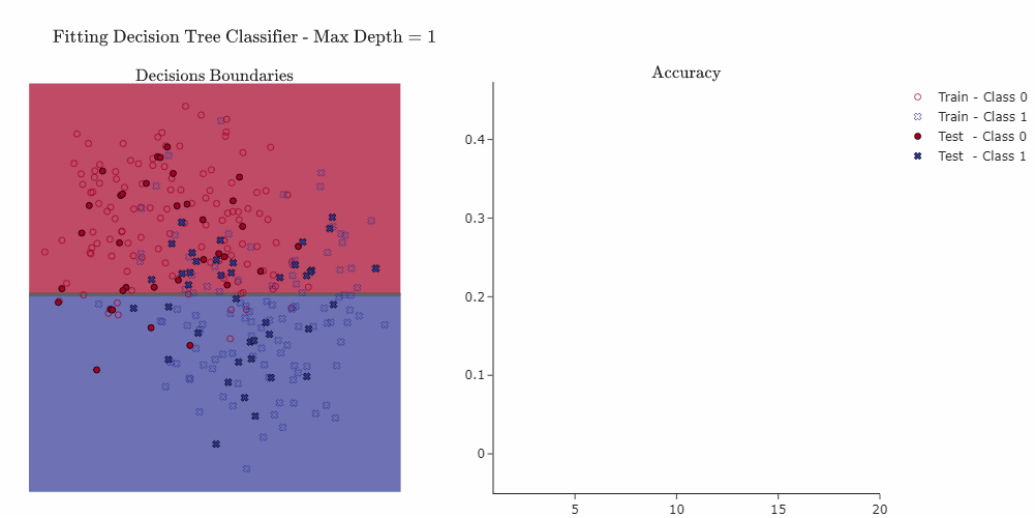


Figure 3.17: Decision Boundaries of Decision Trees: *Fitting model over dataset for different values of max depth k.* *Chapter 3 Examples - Source Code*

3.7.4 CART Heuristic For Growing Trees

The next challenge is how to find the minimizer of (3.34) computationally? So far, all the ERM learners we encountered were **computationally tractable**:

- The linear regression optimization problem was based on ERM. We were able to find aclosed form expression for the minimizer.
- The Half-space classifier was based on ERM and lead to a simple convex optimization problem.

In contrast to those examples the search space over $\mathcal{H}_{CT}^k$ is exponentially large and has no Euclidean or other structure to be used. Finding an ERM solution would mean to use brute-force search, which is infeasible. In fact, it has been proven that implementing ERM on $\mathcal{H}_{CT}^k$ is an NP-Hard problem with respect to the training sample size[1] .

This is our first encounter with the bitter truth that though the ERM principle is nice, it is often impossible to implement efficiently, especially when the hypothesis class has no Euclidean structure. Therefore, we must resort to defining and using **heuristics**: an approach to solving the optimization problem that does not guaranteed to be optimal, but is still sufficient for finding a solution. While the definition of decision trees, the hypothesis class and prediction assignment to boxes in a tree partition are all canonical, there are several different heuristic approaches to the way we "grow a decision tree', namely to the way we choose an hypothesis in practice.

One common approach, coming out of the statistical learning community, is called **Classification and Regression Trees** (CART). This heuristic cinsists of two stages: **growing** the tree, resulting in a tree that is a little too large, and then **pruning** it to bring it down to the most effective size.

Suppose we have chosen k to be the maximal tree depth. The heuristic of growing a full decision tree with at most k levels will proceed top-down, starting from $\mathbb{R}^d$ and progressively chopping each box into two boxes. A given box is **not chopped** if either:

- The maximum number of levels k has been reached.

[1] By reduction from "three dimensional matching", see Hyafil and Rovest, "Constructing Optimal Binary Decision Trees is NP-Complete", Information Processing Letters 5(1), 1976

- The box has reached a pre-determined minimal number of training samples. At the very least we would not split a box if it consists of only a single training sample.

Chopping is done by finding the **best** coordinate, at the **best** value to chop, and whenever you chop given each half-box the **best** class assignment, in the sense of minimizing misclassification error over the training sample. Formally, the pseudocode of the CART heuristic is seen in Algorithm 3.

Algorithm 3 CART - For Growing Classification Tree

procedure CART($S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, k, m_{min})
  $Tree - Partition \leftarrow \emptyset$
  $Boxes \leftarrow \left\{\mathbb{R}^d\right\}$ ▷ Entire sample space as initial box
  while $Boxes \neq \emptyset$ **do**
    $B \leftarrow \text{pop}(Boxes)$
    if $|B| \leq m_{min}$ or depth at B reached k **then**
      $Tree - Partition \leftarrow Tree - Partition \cup \{B\}$
      continue
    end if
    for all feature $i \in [d]$ **do** ▷ Scan all features
      for all threshold value $t \in \mathbb{R}$ **do** ▷ Scan thresholds for features
        Split B along coordinate i at value t:

$$B_{i,t}^+ := \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i > t\right\}, \quad B_{i,t}^- := \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i \leq t\right\}$$

        Let $\hat{y}(B_{i,t}^{\pm})$ denote the class assignment for boxes $B_{i,t}^{\pm}$.
        Let $\ell_S(B_{i,t}^{\pm}, \hat{y}(B_{i,t}^{\pm}))$ denote the empirical risk incurred by $\hat{y}(B_{i,t}^{\pm})$.
        Define $g_i(t) := \ell_S(B_{i,t}^+, \hat{y}(B_{i,t}^+)) + \ell_S(B_{i,t}^-, \hat{y}(B_{i,t}^-))$
      end for
      Set the **best** splitting point: $t_i \leftarrow \text{argmin}_{t \in \mathbb{R}}\, g_i(t)$
    end for
    Select **best** feature to split by: $i^* \leftarrow \text{argmin}_{i \in [d]}\, g_i(t_i)$
    $Boxes \leftarrow Boxes \cup \left\{B_{i^*,t_{i^*}}^+, B_{i^*,t_{i^*}}^-\right\}$ ▷ Split box by best feature and threshold
  end while
  return *Boxes*
end procedure

Time Complexity Analysis

Before we introduced the CART heuristic we described that solving the ERM principle over this hypothesis class is NP-Complete and therefore cannot be done in polynomial time. Let us see that the CAR heuristic can indeed be computed efficiently.

The algorithm iteratively splits boxes into two by finding a coordinate $i \in [d]$ and a value $t \in \mathbb{R}$. It scans all d coordinates and for each scans all possible values of t. Notice, that even though $t \in \mathbb{R}$ we do not need to try all values and it suffices to check only the values in the i'th coordinate of the training sample: $\{\mathbf{x}_i | \mathbf{x} \in S\}$. As we have m samples we only need to evaluate for at most m values, giving each step a time complexity of $\mathcal{O}(md)$.

The next question is how many steps will the algorithm perform? We know that the algorithm will terminate after growing a tree with at most k levels. Such a tree will have at most $2^k - 1$ nodes and leaves. Though this

seems exponentinal in the given input (notice that the hyper-parameter k is also part of the input) we can upper bound this value. As we do not allow empty boxes (and in fact any box with less than some minimal number of samples) the number of nodes (including leaves) in the tree is at most m. We therefore conclude that the time complexity of the CART heuristic is $\mathcal{O}\left(m^2 d\right)$

Pruning a Decision Tree

Pruning a tree means cutting off unnecessary branches. The tree obtained when we are done with the "growing" stage of CART may be too large. A tree too large means some of the boxes are too small, so we are not in an optimal point on the bias-variance tradeoff. It could help reduce the generalization error to merge some of the boxes together, so that the majority votes to determine the box label assignment would be based on larger sets of training samples. Merging two boxes is equivalent, from the decision tree perspective, to merging to leaves together and removing the node between them. Hence, "pruning". We will complete the CART heuristic when we discuss regularization (**??**).

3.7.5 Interpretability

One of the great advantages of a classification tree is that it is very interpretable. To understand which features where important in the classification process, we just look at the nodes (the splits) and see which features the classification tree algorithm chose to split on. A feature that never appeared in any split has not been useful for classification of the training sample. A feature that appears once or more (remember that the algorithm can choose to split on some feature again and again in different areas of R^d) has been useful. To understand why a new sample was classified the way it was classified, we just follow the tree from top to bottom, and see how each answer to each question went.

3.7.6 Learner ID Card

- **Hypothesis class:** The piecewise-constant functions induced by Tree Partitions (axis-aligned rectangles) of depth at most k: $\mathcal{H}_{CT}^k$
- **Learning principle used for training:** ERM
- **Computational implementation:** Implementation of ERM is NP-Hard. Therefore we use heuristics such as the top-down greedy heuristic of CART
- **Interpretability:** A very interpretable learner. Simply reading the tree structure
- **Family of models:** Indexed by k the maximal tree depth
- **Storing fitted model:** To store this model we must store for each node the split information: the coordinate i by which to split and the threshold value t. In addition we need to store the label assignment at the leafs of the tree
- **Prediction of new sample:** Navigate top-down along the tree until reaching a leaf
- **When to use:** This classifier is used as a simple baseline or to get a highly interpretable rule that is easy to explain and plot. Otherwise, classification trees are used to construct "random forests" which are covered in **??**

Appendices

A Linear Algebra

A.1 Norms & Inner Products

A *metric* (or distance function) is a function defined over an arbitrary set X that associates each pair of items in the set with a non-negative scalar quantity. Formally a function $d : X \times X \to \mathbb{R}$ is called a *metric* if and only if for all $x, y, z \in X$ the following properties hold:

- Identity of indiscernibles: $d(x,y) = 0 \iff x = y$
- Symmetry: $d(x,y) = d(y,x)$
- Triangle inequality $d(v,u) \leq d(v,w) + d(w,u)$.

These conditions imply that a metric is a non-negative function returning values in $[0,\infty)$. Some common metric functions are the Euclidean distance, graph distance and string edit distance.

■ **Example .6** Consider the vector space $\mathbb{R}^d$ let us show that the absolute distance, defined as the sum of absolute element-wise subtraction between the vectors $d(\mathbf{v},\mathbf{u}) \coloneqq \sum_{i=1}^{d} |v_i - u_i|$, is a metric function. Firstly, notice that from the properties of the absolute value over $\mathbb{R}$, for any two scalars $a, b \in \mathbb{R}$ it holds that $|a-b| = 0$ if and only if $a = b$. Therefore d, being a sum of non-negative elements equals zero if and only if all summed elements are zero. This takes place if and only if $\mathbf{v} = \mathbf{u}$. Next, symmetry of d is achieved through symmetry of the absolute value function. Lastly, let $\mathbf{v}, \mathbf{u}, \mathbf{w} \in \mathbb{R}^k$ then:

$$d(\mathbf{v},\mathbf{u}) = \sum |v_i - u_i| = \sum |v_i - w_i + w_i - u_i| \leq \sum |v_i - w_i| + \sum |w_i - u_i| = d(\mathbf{v},\mathbf{w}) + d(\mathbf{w},\mathbf{u})$$

and so also the triangle inequality property holds ■

A *norm* on vector space $\mathbb{R}^d$ is a function $||\cdot|| : \mathbb{R}^d \to \mathbb{R}_+$ that satisfies that for all $\alpha \in \mathbb{R}$ and for all $\mathbf{v}, \mathbf{u} \in \mathbb{R}^d$ the following properties hold:

- Positive definiteness: $||\mathbf{v}|| \geq 0$ and $||\mathbf{v}|| = 0$ if and only if $\mathbf{v}$ is the zero vector
- Absolute homogeneity: $||\alpha\mathbf{v}|| = |\alpha| \cdot ||\mathbf{v}||$
- Triangle inequality: $||\mathbf{v} + \mathbf{u}|| \leq ||\mathbf{v}|| + ||\mathbf{u}||$

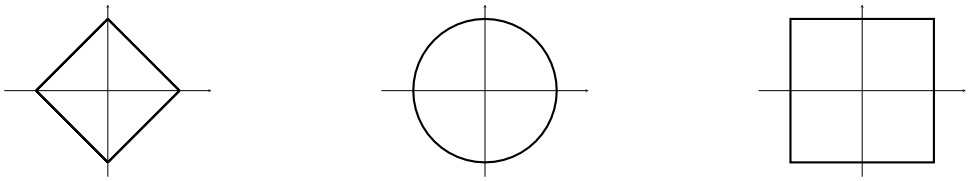
It is helpful to think of a norm as the distance of a vector from the origin. A few commonly used norms are:

- Absolute norm (ℓ_1): $||\mathbf{v}||_1 \coloneqq \sum |v_i|$.

- Euclidean norm (ℓ_2): $||\mathbf{v}||_2 \coloneqq \sqrt{\sum v_i^2}$.
- Infinity norm (ℓ_∞): $||\mathbf{v}||_\infty \coloneqq \max_i |v_i|$.

R These norms are part of a wider family of norms called the L_p norms, defined as $||\mathbf{v}||_p \coloneqq \left(\sum |v_i^p|\right)^{1/p}$ for $p \in \mathbb{N}$. The infinity norm is obtained by taking the limit as $p \to \infty$.

Given a norm on a vector space, i.e. a normed vector space $(V, ||\cdot||)$ we specify the *unit ball* of $||\cdot||$ as the set of vectors such that: $B_{||\cdot||} = \{\mathbf{v} \in V : ||\mathbf{v}|| \leq 1\}$. The use of different norms, and thus different shapes of their unit ball influence the outcome of optimization algorithms using them (**??**).



(a) $||\cdot||_1$ ***(b)*** $||\cdot||_2$ ***(c)*** $||\cdot||_\infty$

Figure 18: *Unit balls of norms on* $\mathbb{R}^2$

Similar to a metric, an *inner product* provides a scalar quantity associated with two given vectors. An inner product space is a vector space V over $\mathbb{R}$ together with a map $\langle\cdot,\cdot\rangle : V \times V \to \mathbb{R}$ satisfying that $\forall \mathbf{v}, \mathbf{u}, \mathbf{w} \in V, \alpha \in \mathbb{R}$ the following properties hold:

- Symmetry: $\langle \mathbf{v}, \mathbf{u} \rangle = \langle \mathbf{u}, \mathbf{v} \rangle$
- Linearity: $\langle \alpha\mathbf{v} + \mathbf{w}, \mathbf{u} \rangle = \alpha \langle \mathbf{v}, \mathbf{u} \rangle + \langle \mathbf{w}, \mathbf{u} \rangle$
- Non-negativity : $\langle \mathbf{v}, \mathbf{v} \rangle \geq 0$ and $\langle \mathbf{v}, \mathbf{v} \rangle = 0 \iff \mathbf{v} = \mathbf{0}$

An example for an inner product is the *standard inner product* $\langle \mathbf{v}, \mathbf{u} \rangle \coloneqq \sum_i v_i u_i$. These definitions of a norm and an inner product are very similar. In fact, given an inner product space, we are also given a norm over this space. Given an inner product space H, the function $||\cdot|| : H \to \mathbb{R}_+$ that is defined by $||\mathbf{v}|| = \langle \mathbf{v}, \mathbf{v} \rangle^{\frac{1}{2}}$ for all $\mathbf{v} \in H$ is a norm on H. It is called the *induced norm*.

Using inner products wee can further formulate other intuitive geometrical notions. We can describe the angle between vectors as $\cos\theta = \langle \mathbf{v}, \mathbf{u} \rangle / ||\mathbf{v}|| \cdot ||\mathbf{u}||$. Consider the norm of the vector $\mathbf{v} - \mathbf{u}$. Being the induced norm of some inner product then:

$$||\mathbf{v} - \mathbf{u}||^2 = \langle \mathbf{v} - \mathbf{u}, \mathbf{v} - \mathbf{u} \rangle = \langle \mathbf{v}, \mathbf{v} \rangle - 2\langle \mathbf{v}, \mathbf{u} \rangle + \langle \mathbf{u}, \mathbf{u} \rangle = ||\mathbf{v}||^2 + ||\mathbf{u}||^2 - 2\langle \mathbf{v}, \mathbf{u} \rangle$$

On the other hand, by the Law of Cosines for the triangle defined by $\mathbf{v}, \mathbf{u}, \mathbf{v} - \mathbf{u}$ then

$$||\mathbf{v} - \mathbf{u}||^2 = ||\mathbf{v}||^2 + ||\mathbf{u}||^2 - 2||\mathbf{v}|| \cdot ||\mathbf{u}|| \cdot \cos\theta$$

Put together we obtain that:

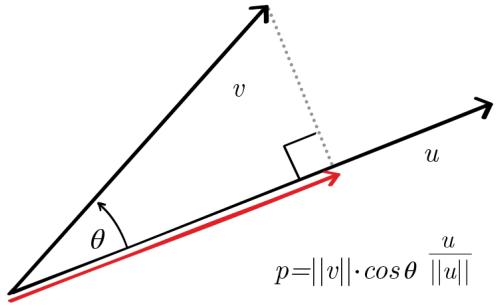
$$\langle \mathbf{v}, \mathbf{u} \rangle = ||\mathbf{v}|| \cdot ||\mathbf{u}|| \cdot \cos\theta \quad \Rightarrow \quad \cos\theta = \langle \mathbf{v}, \mathbf{u} \rangle / ||\mathbf{v}|| \cdot ||\mathbf{u}||$$

A special case if when the angle between two vectors is of 90° for which the result of the inner product between the vectors equals zero: $\theta = 90° \iff \langle \mathbf{v}, \mathbf{u} \rangle = 0$. In this case we say the vectors are *orthogonal* to each other and denote it as $\mathbf{v} \perp \mathbf{u}$.

Another geometric notion is that of *vector projection*. The vector projection of $\mathbf{v}$ onto $\mathbf{u}$ is the orthogonal

projection of $\mathbf{v}$ onto a line parallel to the vector $\mathbf{u}$. It is defined as $\mathbf{p} \coloneqq p \cdot \hat{\mathbf{u}}$ for $p \coloneqq \langle \mathbf{v}, \hat{\mathbf{u}} \rangle \hat{\mathbf{u}}$ and $\hat{\mathbf{u}} \coloneqq \mathbf{u}/||\mathbf{u}||$. p is called the *scalar projection* of $\mathbf{v}$ onto $\mathbf{u}$, and $\hat{\mathbf{u}}$ is the unit vector in the direction of $\mathbf{u}$. The vector $\mathbf{v} - \mathbf{p}$ that is perpendicular to $\mathbf{u}$ and completes a right-angle triangle is called the *vector rejection* of $\mathbf{v}$ from $\mathbf{u}$. Using the angle θ between the two vectors, we can write the vector projection as:

$$\mathbf{p} = \langle \mathbf{v}, \hat{\mathbf{u}} \rangle \hat{\mathbf{u}} = \langle \mathbf{v}, \mathbf{u} \rangle \cdot \frac{\mathbf{u}}{||\mathbf{u}||^2} = ||\mathbf{v}|| \, ||\mathbf{u}|| \cos\theta \cdot \frac{\mathbf{u}}{||\mathbf{u}||^2} = ||\mathbf{v}|| \cos\theta \cdot \hat{\mathbf{u}}$$



A.2 Matrices of Linear Transformations

Given two vector spaces V and W over a field $\mathbb{F}$, a *linear transformation* $T : V \to W$ is a function satisfying that $\forall \mathbf{v}, \mathbf{u} \in V$ and $\forall \alpha \in \mathbb{F}$:

- Additivity: $T(\mathbf{u} + \mathbf{v}) = T(\mathbf{u}) + T(\mathbf{u})$
- Scalar multiplication: $T(\alpha \mathbf{v}) = \alpha \cdot T(\mathbf{v})$

In the case where V, W are of finite dimensions d and m, any linear map $T : V \to W$ is of the form $T(\mathbf{v}) = A\mathbf{v}$ for $\mathbf{v} \in V$ and some matrix $A \in \mathbb{R}^{m \times d}$. The matrix A is called is called the *representing matrix* of T. Extending linear transformations, an *affine transformation* is a transformation of the form $T(\mathbf{v}) = A\mathbf{v} + \mathbf{w}$ for $\mathbf{v} \in V, \mathbf{w} \in W$ and A the matrix associated with a linear transformation from V to W. Notice that an affine transformation is not a linear transformation as it does not map $\mathbf{0}_V$ to $\mathbf{0}_W$.

Using the representing matrix A of a linear transformation $T : V \to W$ we define four *fundamental subspaces*:

- Kernel- (or null-) space of A as $Ker(A) \coloneqq \{\mathbf{x} \in V | A\mathbf{x} = 0\}$. Also denotes as $\mathcal{N}(A)$.
- Image- (or column/range-) space of A as $Im(A) \coloneqq \{\mathbf{w} \in W | \mathbf{w} = A\mathbf{x}, \mathbf{x} \in V\}$. Also denotes as $Col(A)$ or $\mathcal{R}(A)$.
- Row space of A as $Im\left(A^\top\right) \coloneqq \left\{\mathbf{x} \in V | \mathbf{x} = A^\top \mathbf{w}, \mathbf{w} \in W\right\}$. Equivalently it can be defined as the column space of $A^\top$ and therefore denoted as $Col\left(A^\top\right)$.
- Null space of $A^\top$ as $Ker\left(A^\top\right) \coloneqq \left\{\mathbf{x} \in W | A^\top \mathbf{x} = 0\right\}$. This space is also referred to as the left null space of A.

Note that by definition, $Ker(A), Row(A) \subseteq V$ and $Im(A) \subseteq W$. Another quantity associated with matrices is the *rank* of a matrix. For $A \in \mathbb{R}^{m \times d}$ the rank of A is the maximum number of linearly independent rows of A and denoted by $rank(A)$. It holds that the rank of A equals both the dimension of the columns space and of the row space of A. As such, we refer to A being of *full rank* if and only if $rank(A) = \min\{m, d\}$. Otherwise we say that A is *rank deficient*.

For the case of a square matrix we define the notion of invertability. For $A \in \mathbb{R}^{d \times d}$ a square matrix, we say that A is invertible (or non-singular) if there exists a matrix $B \in \mathbb{R}^{d \times d}$ such that $AB = I_d = BA$. We denote the inverse by A^{-1}. Then, the following are equivalent:

- A is invertible (non-singular)
- A is full-rank
- $Det(A) \neq 0$
- $Im(A) = \mathbb{R}^m$ (i.e. the image is the whole space)
- $ker(A) = \vec{0}$ (i.e. the kernel is trivial)

A.2.1 Orthogonal Projection Matrices

We can extend the notion of vector projections to projecting a given vector onto a subspace of arbitrary dimension.

Definition A.1 Let V be a k-dimensional subspace of $\mathbb{R}^d$, and let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be an orthonormal basis of V. The matrix $P := \sum_{i=1}^k \mathbf{v}_i \otimes \mathbf{v}_i = \sum_{i=1}^k \mathbf{v}_i \mathbf{v}_i^\top$ is an *orthogonal projection matrix* onto the subspace V.

where the operation performed on the vectors is the *outer product*. For two vectors $\mathbf{v} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m$, the outer product of $\mathbf{v}$ and $\mathbf{u}$, which is denoted by $\mathbf{v} \otimes \mathbf{u}$ or $\mathbf{v}\mathbf{u}^\top$ is an $n \times m$ matrix with entries:

$$[\mathbf{v} \otimes \mathbf{u}]_{ij} = v_i \cdot u_j, \quad \mathbf{v} \otimes \mathbf{u} = \begin{bmatrix} v_1 u_1 & v_1 u_2 & \cdots & v_1 u_m \\ \vdots & \vdots & \ddots & \vdots \\ v_n u_1 & v_n u_2 & \cdots & v_n u_m \end{bmatrix}$$

That is, P is expressed as the sum of k one dimenional matrices. Another way to write P is as $P = AA^\top$ where the columns of A are $\mathbf{v}_1, \ldots, \mathbf{v}_k$ an orthonormal basis of V. The following lemma summarizes some useful properties of orthogonal projection matrices.

Lemma A.1 Let P be an orthogonal projection matrix then P has the following properties:

- P is symmetric
- $P^2 = P$
- The eigenvalues of P are either 0 or 1. $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are the eigenvectors of P which correspond to the eigenvalue 1.
- $(I - P)P = 0$
- $\forall \mathbf{x} \in \mathbb{R}^d$ and $\forall \mathbf{u} \in V$, $||\mathbf{x} - \mathbf{u}|| \geq ||\mathbf{x} - P\mathbf{x}||$
- $\mathbf{x} \in V \Rightarrow P\mathbf{x} = \mathbf{x}$

The outer product of two non-zero vectors defines a matrix with rank of 1. Notice, that the orthogonal projection matrix, being a sum of such matrices, is in fact a sum of rank 1 matrices.

A.2.2 Positive (Semi-) Definiteness

For square symmetric matrices we define the notion of definiteness and semi-definiteness.

Definition A.2 Let $A \in \mathbb{R}^{d \times d}$ be a symmetric matrix. A is a *positive semi-definite* (PSD) matrix if and only if

$$\forall \mathbf{x} \in \mathbb{R}^d \quad \mathbf{x}^\top A \mathbf{x} \geq 0$$

and denote so by $A \succeq 0$. Further, A is *positive definite* (PD) if and only if

$$\forall \mathbf{x} \in \mathbb{R}^d \quad \mathbf{x}^\top A \mathbf{x} > 0$$

and denote so by $A \succ 0$.

In addition, for a symmetric matrix A the following are equivalent:

- A is a PSD matrix
- For all $\mathbf{x} \in \mathbb{R}^d$ then $\mathbf{x}^\top A \mathbf{x} \geq 0$

- For λ an eigenvalue of A then $\lambda \geq 0$
- A can be written as the product $A = B^\top B$ for some matrix $B \in \mathbb{R}^{k\times d}$

Similarly, these conditions can be defined for PD matrices restricting that $\mathbf{x}^\top A\mathbf{x} > 0$, strictly positive eigenvalues and $A = B^* B$ for B^* being the conjugate transpose of B. There are many useful properties for PSD matrices such as that for $\alpha \geq 0$ also αA is a PSD matrix; the sum of PSD matrices is a PSD matrix; and for M, N two PSD matrices also the products MNM and NMN are PSD matrices.

A.3 Matrix Factorizations

Matrix factorization/decomposition is a strong tool with many theoretical as well as practical usages. The core idea is to find a representation of a given matrix as the product of several different matrices which have certain desired properties. For example, in the case of a PSD matrix $A \in \mathbb{R}^{d\times d}$ we have seen that there exists a matrix $B \in \mathbb{R}^{k\times d}$ such that $A = B^\top B$. Here we were able to decompose A into the product of the matrices $B^\top$ and B. We could now try and find if there exists a matrix B with some property. For example, if B is a lower triangular matrix, i.e. $B_{ij} = 0\ \forall j > i$, it is called the *Cholesky decomposition*.

A.3.1 Eigenvalue Decomposition

Let A be a square matrix. We say that a vector $\mathbf{0} \neq \mathbf{v} \in \mathbb{R}^d$ is an *eigenvector* of A corresponding to an *eigenvalue* $\lambda \in \mathbb{R}$ if and only if $A\mathbf{v} = \lambda\mathbf{v}$. To find the eigenvectors and eigenvalues of A we therefore wish to solve the linear system of $(A - \lambda I)\,\mathbf{v} = \mathbf{0}$. This system has a non-zero solution if and only if $\det(A - \lambda I) = 0$. These solutions if exist, are the roots of the *characteristic polynomial* of A: $p_A(\lambda) := |A - \lambda I|$. From the fundamental theorem of algebra we learn that the characteristic polynomial can be factored as the product

$$|A - \lambda I| = (\lambda_1 - \lambda) \cdot \ldots \cdot (\lambda_d - \lambda)$$

for λ_i the roots of the characteristic polynomial and the eigenvalues of A. Given λ and eigenvalue of A we define the set of eigenvectors corresponding to λ as $E_\lambda := \{\mathbf{v} \mid (A - \lambda I)\,\mathbf{v} = \mathbf{0}\}$. This linear subspace is the *eigenspace* of A associated with the eigenvalue λ. The dimension of E is termed the *geometric multiplicity* of λ.

The idea of matrix decomposition relates to that of *matrix diagonalization*. For a square matrix $A \in \mathbb{R}^{d\times d}$, we say that A is diagonalizable if there exists an invertible matrix P such that $P^{-1}AP$ is diagonal. In the case of a symmetric matrix, the Eigenvalue Decomposition (EVD) provides a diagonalization of a matrix using its eigenvalues and eigenvectors.

Theorem A.2 Let $A \in \mathbb{R}^{d\times d}$ be a real symmetric matrix. Then, there exist an orthogonal matrix $U \in \mathbb{R}^{d\times d}$ and a diagonal matrix $D \in \mathbb{R}^{d\times d}$ such that $A = UDU^\top$. Furthermore, it holds that

- The diagonal entries of D are the eigenvalues of A (with their multiplicities).
- The columns of U are eigenvectors for A corresponding the eigenvalues on the diagonal of D. These form an orthonormal basis of $\mathbb{R}^d$.

This decomposition is called the *Eigenvalue Decomposition* (EVD) or the *Spectral Decomposition* of A.

Namely, $A\mathbf{u}_i = D_{ii}\mathbf{u}_i,\ i \in [d]$. Without loss of generality, we arrange the elements of D (and respectively, the columns of U) such that the eigenvalues are in decreasing order $D_{ii} \geq D_{i+1,i+1}$.

■ **Example .7** Consider the following symmetric matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

To find the EVD of A we solve its eigenvalue problem by finding λ such that $\det(A - \lambda I) = 0$, yielding the equation $(2-\lambda)^2 - 1 = 0$ with the solutions $\lambda = 3, 1$. Now, to find the eigenvectors we search for $\mathbf{v} \neq 0$ such that $(A - \lambda I)\mathbf{v} = 0$ for $\lambda = 3, 1$. We find that the eigenvector corresponding eigenvalues $\lambda = 3$ is $\mathbf{v}_3 = (1,1)^\top$

and the eigenvector corresponding eigenvalue $\lambda = 1$ is $\mathbf{v}_1 = (1,-1)^\top$. We normalize the eigenvectors and obtain the EVD of A:

$$A = UDU^\top \quad U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}, D = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

By multiplying $UDU^\top$ we validate that indeed we achieve the matrix A and the correctness of the decomposition. ∎

To gain a deeper understanding into the EVD, consider the linear transformations represented by the matrix A and the matrices U and D. It holds that for a square matrix, the eigenspace corresponding eigenvalue zero spans the kernel space of A, while the eigenspace corresponding non-zero eigenvalues spans the range of A. Thus, for $rank(A) = k \leq d$:

$$\mathcal{R}(A) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}, \quad \mathcal{N}(A) = \text{span}\{\mathbf{u}_{k+1}, \ldots, \mathbf{u}_d\}$$

As these vectors are orthogonal to each other, we have an orthonormal basis of eigenvectors of A to $\mathbb{R}^d$. Now consider a unit vector $\mathbf{x} \in \mathbb{R}^d$, $||\mathbf{x}|| = 1$. The image of $\mathbf{x}$ under A is

$$A\mathbf{x} = (UDU^\top)\mathbf{x} = UD \begin{bmatrix} \langle \mathbf{x}, \mathbf{u}_1 \rangle \\ \vdots \\ \langle \mathbf{x}, \mathbf{u}_d \rangle \end{bmatrix} = U \begin{bmatrix} \lambda_1 \langle \mathbf{x}, \mathbf{u}_1 \rangle \\ \vdots \\ \lambda_d \langle \mathbf{x}, \mathbf{u}_d \rangle \end{bmatrix} = \sum_{i=1}^{d} \lambda_i \langle \mathbf{x}, \mathbf{u}_i \rangle \mathbf{u}_i$$

Namely, A provides a representation of $\mathbf{x}$ in terms of the orthonormal basis $\mathbf{u}_1, \ldots, \mathbf{u}_d$, and dialites or contracts components (i.e directions in space) according to the magnitude of the eigenvalues. If the matrix is not of full rank, in some of the directions (those corresponding the null space) we are left with the zero vector.

The spectral decomposition has many useful properties. Just by observing the decomposition itself we are able to "read" the spectrum of A - the unique values along the diagonal of D - with their algebraic multiplicity, and the range- and null-spaces of A. In addition, the determinant and trace of A are obtained from D by:

$$det(A) = \prod_i D_{ii}, \quad tr(A) = \sum_i D_{ii}$$

Furthermore, taking A to the power is simply done by raising each of the eigenvalues to that power. And in the case of an invertible matrix A, the inverse is simply

$$A^{-1} = (UDU^\top)^{-1} = UD^{-1}U^\top, \quad D_{ii}^{-1} = (D_{ii})^{-1}$$

A.3.2 Singular-Values Decomposition

The *Singular-Values Decomposition* is a decomposition related to the EVD for an arbitrary real matrix $A \in \mathbb{R}^{m \times d}$. We say that $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^d$ are left- and right singular values of A, corresponding singular value $\sigma \in \mathbb{R}_+$ if and only if $A\mathbf{v} = \sigma\mathbf{u}$.

Theorem A.3 Let $A \in \mathbb{R}^{m \times d}$ be a real matrix. Then, there exists:

- An orthogonal matrix $U \in \mathbb{R}^{m \times m}$ whose columns are the left singular vectors of A.
- An othrogonal matrix $V \in \mathbb{R}^{d \times d}$ whose columns are the right singular vectors of A.
- A diagonal matrix $\Sigma \in \mathbb{R}^{m \times d}$ whose diagonal entries are the non-negatve singular values of A such at $A\mathbf{v}_i = \Sigma_{ii}\mathbf{u}_i$.

$$A = U\Sigma V^\top = \overbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_m \\ | & & | \end{bmatrix}}^{m \times m} \overbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix}}^{m \times d} \overbrace{\begin{bmatrix} - & \mathbf{v}_1^\top & - \\ & \vdots & \\ - & \mathbf{v}_d^\top & - \end{bmatrix}}^{d \times d}$$

This is called the *Singular-Value Decomposition* of A.

Without loss of generality, we arrange the elements of Σ (and respectively, the columns of U, V) such that the singular values are in decreasing order $\Sigma_{ii} \geq \Sigma_{i+1,i+1}$.

Geometric Interpretation

As we have done for the EVD, let us consider the linear transformations represented by A. In this case the transformation takes $\mathbb{R}^d$ to a different space $\mathbb{R}^m$. When represented using the decomposition, similar to the EVD, the nature of the transformation becomes clear. For V, U whose columns are orthonormal basis of the domain $\mathbb{R}^d$ and range $\mathbb{R}^m$ of A it can be shown that Ex.7

$$\begin{aligned} \mathcal{R}(A) &= \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}, \quad \mathcal{N}(A^\top) = \text{span}\{\mathbf{u}_{k+1}, \ldots, \mathbf{u}_m\} \\ \mathcal{R}(A^\top) &= \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}, \quad \mathcal{N}(A) = \text{span}\{\mathbf{v}_{k+1}, \ldots, \mathbf{v}_d\} \end{aligned}$$

where $rank(A) = k \leq \min\{m, d\}$. Then, for a general element in $\mathbb{R}^d$, represented in this basis $\mathbf{x} = \sum_i \langle \mathbf{x}, \mathbf{v}_i \rangle \mathbf{v}_i$ we see that:

$$A\mathbf{x} = A \sum_i \langle \mathbf{x}, \mathbf{v}_i \rangle \mathbf{v}_i = \sum_i \langle \mathbf{x}, \mathbf{v}_i \rangle U \Sigma V^\top \mathbf{v}_i = \sum_i \langle \mathbf{x}, \mathbf{v}_i \rangle U \Sigma \mathbf{e}_i = \sum_i \sigma_i \langle \mathbf{x}, \mathbf{v}_i \rangle U \mathbf{e}_i = \sum_i \sigma_i \langle \mathbf{x}, \mathbf{v}_i \rangle \mathbf{u}_i$$

That is, the SVD simply scales (expands or contracts) some of the components according to the magnitude of the singular values, and discards components corresponding to the null-spaces. Now, to understand the manner in which A deforms the space consider its action on the unit sphere in $\mathbb{R}^d$. Suppose $\mathbf{x}$ is on the unit sphere, i.e $\mathbf{x} = x_1\mathbf{v}_1 + \ldots + x_d\mathbf{v}_d$, $||\mathbf{x}||_2^2 = \sum x_i^2 = 1$. The image of the unit sphere under A is therefore $A\mathbf{x} = y_1\mathbf{u}_1 + \ldots + y_k\mathbf{u}_k$ for $y_i := \sigma_i x_i$ where

$$||A\mathbf{x}||_2^2 = \frac{y_1^2}{\sigma_1^2} + \ldots + \frac{y_k^2}{\sigma_k^2} = \sum_{i=1}^{k} x_i^2 \leq 1$$

Namely, A maps the unit k-sphere in $\mathbb{R}^d$ into an ellipsoid with axes in the directions of $\mathbf{u}_i$ and with magnitudes σ_i (collapsing $d-k$ dimensions of the domain) and then embeds the ellipsoid in $\mathbb{R}^m$.

The connection between SVD and EVD

The natural question at this point is how to choose the basis $\{\mathbf{v}_1, \ldots, v_d\}$ and $\{\mathbf{u}_1 \ldots, \mathbf{u}_m\}$. Since $A\mathbf{v}_i = \sigma_i\mathbf{u}_i$ it is simple to obtain the diagonal representation of A. Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ be some orthonormal basis of $\mathbb{R}^d$ such that the first k elements span the row space of A while the remaining $d-k$ elements span the null space of A. We can now obtain $\mathbf{u}_i$ as a unit vector parallel to $A\mathbf{v}_i$, and extend this to a basis of $\mathbb{R}^m$. Relative to these basis we have achieved a diagonalization of A:

$$U^\top AV = U^\top U \Sigma V^\top V = \Sigma \tag{35}$$

In general however, even for orthonormal v's there is no guarantee for orthogonality to be preserved under A. Therefore the key point is in finding the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ for which $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are orthonormal. We find such a basis using the EVD of the $d \times d$ symmetric matrix $A^\top A$. Let $A^\top A = VDV^\top$ be the EVD of $A^\top A$, where the columns of V, $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$, (i.e the eigenvectors of $A^\top A$) are an orthonormal basis of $\mathbb{R}^d$. Then

$$\langle A\mathbf{v}_i, A\mathbf{v}_j \rangle = (A\mathbf{v}_i)^\top (A\mathbf{v}_j) = \mathbf{v}_i^\top \left(A^\top A\mathbf{v}_j\right) = \mathbf{v}_i^\top (\lambda_j \mathbf{v}_j) = \lambda_j \mathbf{v}_i^\top \mathbf{v}_j = 0$$

Thus, the image set of applying A over the basis of eigenvectors of $A^\top A$, $\{A\mathbf{v}_1, \ldots, A\mathbf{v}_d\}$, is orthogonal with the nonzero vectors forming a basis for the range of A. Namely, the images under A of the eigenvectors of $A^\top A$ provide an orthogonal bases allowing the diagonalization of A as seen in (35). By normalizing the non-zero vectors we obtain $\{u_1, \ldots, u_k\}$

$$\mathbf{u}_i := \frac{A\mathbf{v}_i}{||A\mathbf{v}_i||} = \frac{A\mathbf{v}_i}{\sqrt{\lambda_i \mathbf{v}_i^\top \mathbf{v}_i}} = \frac{1}{\sqrt{\lambda_i}} A\mathbf{v}_i, \quad i \in [k]$$

completing the construction of the orthonormal bases for $\mathbb{R}^d$ and $\mathbb{R}^m$. By setting $\sigma_i = \sqrt{\lambda_i}$ we also achieve that $A\mathbf{v}_i = \sigma_i \mathbf{u}_i$ or in matrix notation $A = U\Sigma V^\top$.

Conversely, given the SVD of $A = U\Sigma V^\top$ we can recover the EVD of $A^\top A$ or of $AA^\top$:

$$A^\top A = \left(U\Sigma V^\top\right)^\top \left(U\Sigma V^\top\right) = V\Sigma^\top U^\top U\Sigma V^\top = V\Sigma^\top\Sigma V^\top$$
$$AA^\top = \left(U\Sigma V^\top\right)\left(U\Sigma V^\top\right)^\top = U\Sigma V^\top V\Sigma^\top U^\top = U\Sigma\Sigma^\top U^\top$$

where $\Sigma^\top\Sigma$ and $\Sigma\Sigma^\top$ are both square matrices whose first k diagonal entries are σ_i^2. We therefore conclude that the left singular values of A are the eigenvectors of $AA^\top$; the right singular values of A are the eigenvectors of $A^\top A$; and the singular values of A are the square root of the eigenvalues of $AA^\top$ and $A^\top A$. Furthermore, it can be shown that up to orthogonal transformations of $A^\top A$ and $AA^\top$ the SVD of A is uniquely determined.

Compact SVD Form

When writing the SVD of a matrix A, with rank $rank(A) = k < min\{m,d\}$, it becomes evident that we can express the SVD in a more compact way. Without loss of generality, suppose $d \leq m$. Notice that Σ consists of zero rows and columns as $\sigma_{k+1},\ldots,\sigma_d$ are all zero.

$$A = \left[\begin{array}{ccc|ccc} \mathbf{u}_1 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} & \cdots & \mathbf{u}_m \end{array}\right] \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & \mathbf{0} \\ & & \sigma_k & \\ \hline & \mathbf{0} & & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_k^\top \\ \hline \mathbf{v}_{k+1}^\top \\ \vdots \\ \mathbf{v}_d^\top \end{array}\right]$$

When we partition the multiplication as follows it becomes evident that the left and right singular vectors $\mathbf{u}_{k+1},\ldots,\mathbf{u}_m$ and $\mathbf{v}_{k+1},\ldots,\mathbf{v}_d$ do not make any contribution to A. Their purpose is to expand the leading left and right singular vectors into orthonormal bases of $\mathbb{R}^m$ and $\mathbb{R}^d$ respectively.

$$A = \left[\begin{array}{ccc} \mathbf{u}_1 & \cdots & \mathbf{u}_k \end{array}\right] \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_k^\top \end{array}\right] + \left[\begin{array}{ccc} \mathbf{u}_{k+1} & \cdots & \mathbf{u}_m \end{array}\right] \left[\begin{array}{c} \mathbf{0} \end{array}\right] \left[\begin{array}{c} \mathbf{v}_{k+1}^\top \\ \vdots \\ \mathbf{v}_d^\top \end{array}\right]$$

Thus, we eliminate left and right singular vectors corresponding zero singular values to obtain the *Compact SVD* form.

$$A = \overbrace{\left[\begin{array}{ccc} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ | & & | \end{array}\right]}^{m\times r} \overbrace{\left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array}\right]}^{k\times k} \overbrace{\left[\begin{array}{ccc} - & \mathbf{v}_1^\top & - \\ & \vdots & \\ - & \mathbf{v}_k^\top & - \end{array}\right]}^{k\times d} \tag{36}$$

A.4 Exercises

Theoretical Questions

1. Let $\mathbf{v}_1,\ldots,\mathbf{v}_k \in V \subset \mathbb{R}^d$ be an orthonormal basis of V. Let A be the matrix whose columns are $\mathbf{v}_1,\ldots,\mathbf{v}_k$. Prove that $\sum \mathbf{v}_i\mathbf{v}_i^\top = AA^\top$.

2. Let $V \subset \mathbb{R}^d$ be a vector space of dimension k and P an orthogonal projection matrix onto V. Prove the properties of an orthogonal projection matrix described in Lemma A.1.

3. Let $A \in \mathbb{R}^{d\times d}$ be a square matrix of rank $k \leq d$. Let $\mathbf{u}_1,\ldots,\mathbf{u}_k$ be eigenvectors of A corresponding non-zero eigenvalues and $\mathbf{u}_{k+1},\ldots,\mathbf{u}_d$ be eigenvectors of A corresponding eigenvalue zero. Show that $\mathcal{R}(A) = \text{span}\{\mathbf{u}_1,\ldots,\mathbf{u}_k\}$ and that $\mathcal{N}(A) = \text{span}\{\mathbf{u}_{k+1},\ldots,\mathbf{u}_d\}$.

4. Let $A \in \mathbb{R}^{d\times d}$ be a symmetric matrix.
   - Show that $det(A) = \prod D_{ii}$.
   - Show that $tr(A) = \sum D_{ii}$.

5. Let $A = UDU^\top$ be the EVD of A. Provide an expression for A^k for $k \in \mathbb{N}$, using the eigenvalues of A.

6. Let $U \in \mathbb{R}^{d\times d}$ be an orthogonal matrix. Show that U is isometric, that is $||U\mathbf{x}||_2 = ||\mathbf{x}||_2 \ \forall \mathbf{x} \in \mathbb{R}^d$.

7. Let $A \in \mathbb{R}^{m\times d}$ be a real matrix of rank k and $A = U\Sigma V^\top$ an SVD of A. Show that the four fundamental subspaces of A are given by:

$$\begin{array}{lcllcl} \mathcal{N}(A) & = & \text{span}\{\mathbf{v}_{k+1},\ldots,\mathbf{v}_d\}, & \mathcal{R}(A) & = & \text{span}\{\mathbf{u}_1,\ldots,\mathbf{u}_k\} \\ \mathcal{N}(A^\top) & = & \text{span}\{\mathbf{u}_{k+1},\ldots,\mathbf{u}_m\}, & \mathcal{R}(A^\top) & = & \text{span}\{\mathbf{v}_1,\ldots,\mathbf{v}_k\} \end{array}$$

B Calculus

B.1 Gradients, Jacobians & Hessian

Derivatives The *derivative* of a function measures the degree in which the function changes in a small area around a point of interest. For a scalar function $f : \mathbb{R} \to \mathbb{R}$, the derivative of f at point $x \in \mathbb{R}$ is defined as

$$\frac{d}{dx} f(x) := \lim_{a\to 0} \frac{f(x+a) - f(x)}{a}$$

■ **Example .8** Consider the **Re**ctified **L**inear **U**nit function defined as the positive part of its argument: $f(x) = \max(0,x)$. The derivative if this function is:

$$\frac{\partial}{\partial x} f(x) = \mathbb{1}_{x>1}$$

Note that at $x = 0$ the derivative of ReLU is undefined. To deal with such cases we will later define sub-gradients. ■

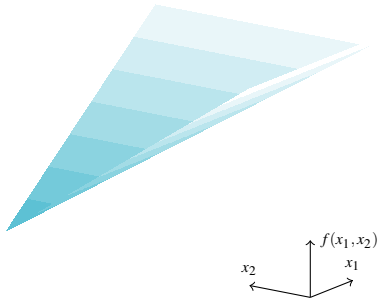
In the case of multivariate functions where $f : \mathbb{R}^d \to \mathbb{R}$ the notion of derivative is generalized to the degree in which a function changes in each of the input coordinates. The *partial derivative* of f at point $\mathbf{x} \in \mathbb{R}^d$ with respect to x_i is defined as

$$\begin{aligned} \frac{\partial}{\partial x_i} f(\mathbf{x}) &:= \lim_{a\to 0} \frac{f(\mathbf{x}+a\mathbf{e}_i) - f(\mathbf{x})}{a} \\ &= \lim_{a\to 0} \frac{f(x_1,\ldots,x_i+a,\ldots,x_d) - f(x_1,\ldots,x_i,\ldots,x_d)}{a} \end{aligned}$$

where $\mathbf{e}_i$ is the i-th standard basis vector. Namely, a partial derivative of a function is its derivative with respect to one of its variables, while all others are kept constant.

■ **Example .9** For $f(\mathbf{x}) = \max_i(x_1,\ldots,x_d)$ the partial derivatives of f at x_i are:

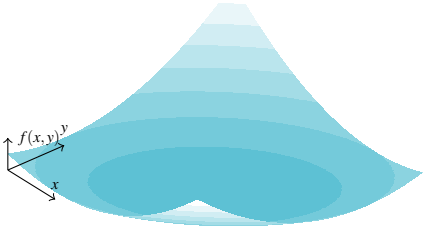
$$\frac{\partial}{\partial x_i} f(\mathbf{x}) = \mathbb{1}_{i=\text{argmax}\,(x_1,\ldots,x_d)}$$



■

■ **Example .10** For $f(x,y) = x^2 + xy + y^2$ the partial derivatives of f at (x_0, y_0) are

$$\frac{\partial}{\partial x} f(x_0,y_0) = 2x_0 + y_0, \quad \frac{\partial}{\partial y} f(x_0,y_0) = 2y_0 + x_0$$



■

Gradients Then the notion of *gradient* of $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x}$ is simply the vector of all partial derivatives. It is as a convention that we define the gradient as a column vector.

$$\nabla f(\mathbf{x}) := \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_d} \right)^\top$$

■ **Example .11** Let $\mathbf{w} \in \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ be a linear functional defined by $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$. To calculate the gradient of f at point $\mathbf{x}$ we derive the partial derivatives of f. From linearity of the derivative:

$$\frac{\partial}{\partial x_j} f(\mathbf{x}) = \sum_i \frac{\partial}{\partial x_j} [f(\mathbf{x})]_i = \sum_i \frac{\partial}{\partial x_j} \mathbf{w}_i \mathbf{x}_i = \mathbf{w}_j$$

Therefore, in vector notation $\nabla f(\mathbf{x}) = \mathbf{w}$. ■

■ **Example .12** Let $f : \mathbb{R}^d \to \mathbb{R}$ be defined by $f(\mathbf{x}) = ||\mathbf{x}||^2$. Using linearity of the derivative then

$$\frac{\partial}{\partial x_j} f(x) = \sum_i \frac{\partial}{\partial x_j} x_i^2 = 2x_j$$

which in vector notation can be written as $\nabla f(\mathbf{x}) = 2\mathbf{x}$. ■

Jacobians The Jacobian is the generalization of the gradient for multivariate vector-valued functions, that is, functions that receive and output vectors. So for $\mathbf{f}:\mathbb{R}^d \to \mathbb{R}^m$ where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}),\ldots,f_m(\mathbf{x}))^\top$, the *Jacobian* of f is the $m \times d$ matrix of all partial derivatives:

$$J_{\mathbf{x}}(\mathbf{f}) \coloneqq \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_d} \\ \vdots & & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

■ **Example .13** Let $f:\mathbb{R}^2 \to \mathbb{R}$ be defined as $f(\mathbf{x}) = x_1^2 + x_2^2$. The Jacobian of f is:

$$J_x(\mathbf{f}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \end{bmatrix} = [2x_1, 2x_2] = \nabla f(\mathbf{x})^\top$$

■

Notice that for any function where $k = 1$ the Jacobian is in fact the transposed gradient vector: $J_{\mathbf{x}}(f) = \nabla f(\mathbf{x})^\top$.

■ **Example .14** Let $f:\mathbb{R}^d \to \mathbb{R}^m$ defined as $f(\mathbf{x}) = A\mathbf{x}$ where $A \in \mathbb{R}^{m\times d}$. To find the Jacobian of of f, $J_{\mathbf{x}}(f)$), define the set of functions computing each coordinate in the output vector $\forall i \in [m] \quad f_i(\mathbf{x}) = \langle A_i, \mathbf{x}\rangle$. Then the Jacobian of f is comprised of the gradients of $f_1,\ldots,f_m$ as rows. Notice that we have already computed the gradient of linear functionals in Example .11 so:

$$J_{\mathbf{x}}(f) = \begin{bmatrix} \nabla f_1(\mathbf{x})^\top \\ \vdots \\ \nabla f_d(\mathbf{x})^\top \end{bmatrix} = \begin{bmatrix} -A_1- \\ \vdots \\ -A_m- \end{bmatrix} = A$$

■

Hessians Similar to the gradient we can consider the second derivative of a function with respect to each of its partial first derivatives. Let $f:\mathbb{R}^d \to \mathbb{R}$ be a twice differential function. The *Hessian* matrix H of f is the square matrix of second derivative:

$$H[f] \coloneqq \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_d} \end{bmatrix}$$

■ **Example .15** Let us calculate the Hessian of the polynomial function $f(x_1,x_2) = x_1^2 + x_1x_2 + x_2^2$. We begin with calculating the first partial derivatives of f: $\frac{\partial f(x_1,x_2)}{\partial x_i} = 2x_i + x_j$ for $i \in \{1,2\}$ and $j \neq i$. Next, we calculate the derivative a second time with respect to each of the parameters:

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_i \partial x_j} = \begin{cases} 2 & x_i = x_j \\ 1 & x_i \neq x_i \end{cases}$$

We therefore conclude that the Hessian of f is :

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

■

B.2 Chain Rules

Many times a given function of interest is the composition of other functions. In that case, when we calulate the derivatives we apply the chain rule. In the case of a scalar function, let $f:\mathbb{R}\to\mathbb{R}$ and $g:\mathbb{R}\to\mathbb{R}$ be two differential functions, then the derivative of the composite $f\circ g$ is:

$$(f\circ g)' \coloneqq \left(f'\circ g\right)\cdot g'$$

Namely, for $h(x)=f(g(x))$ then $\forall x\in\mathbb{R}\ h'(x)=f'(g(x))\cdot g'(x)$. We can extend this to the multivariate case where the composite receives a vector and outputs another.

Theorem B.1 Let $f:\mathbb{R}^d\to\mathbb{R}^m$ and $g:\mathbb{R}^k\to\mathbb{R}^d$. The Jacobian of the composition $(f\circ g):\mathbb{R}^k\to\mathbb{R}^m$ at $\mathbf{x}$ is

$$J_{\mathbf{x}}(f\circ g)=J_{g(\mathbf{x})}(f)J_{\mathbf{x}}(g)\coloneqq\begin{bmatrix}\frac{\partial f_1(g(\mathbf{x}))}{\partial g_1(\mathbf{x})} & \cdots & \frac{\partial f_1(g(\mathbf{x}))}{\partial g_d(\mathbf{x})}\\ \vdots & & \vdots\\ \frac{\partial f_m(g(\mathbf{x}))}{\partial g_1(\mathbf{x})} & \cdots & \frac{\partial f_m(g(\mathbf{x}))}{\partial g_d(\mathbf{x})}\end{bmatrix}=\begin{bmatrix}\frac{\partial g_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\mathbf{x})}{\partial x_k}\\ \vdots & & \vdots\\ \frac{\partial g_d(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_d(\mathbf{x})}{\partial x_k}\end{bmatrix}$$

In element-wise notation:

$$J_{\mathbf{x}}(f\circ g)_{i,j}\coloneqq\sum_l\frac{\partial f_i(g(\mathbf{x}))}{\partial g_l(\mathbf{x})}\frac{\partial g_l(\mathbf{x})}{\partial x_j}$$

■ **Example .16** Let $f(\mathbf{x})=||\mathbf{x}||^2$ and $g(\mathbf{x})=A\mathbf{x}$ for some matrix $A\in\mathbb{R}^{m\times d}$. Using the chain rule, Jacobian of $f\circ g$ is

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

As g a linear transformation we have see in Example .14 that $J_{\mathbf{x}}(g)=A$. Notice, that as $Im(f)\subseteq\mathbb{R}$, the Jacobian of f equals to the transpose of its gradient. As seen in Example .12, $J_{g(\mathbf{x})}(f)=(2g(\mathbf{x}))^\top$. Therefore

$$J_{\mathbf{x}}(f\circ g)=2\mathbf{x}^\top A^\top A$$

■

■ **Example .17** Next, let us calculate the gradient of the following function: $f(\mathbf{x})=\frac{1}{2}||A\mathbf{x}-\mathbf{y}||^2$. Applying the chain rule then:

$$\begin{aligned}\nabla f(\mathbf{x}) &= \tfrac{\partial}{\partial\mathbf{x}}\tfrac{1}{2}||A\mathbf{x}-\mathbf{y}||^2\\ &= \tfrac{1}{2}\tfrac{\partial}{\partial\mathbf{x}}\left(\mathbf{x}^\top A^\top A\mathbf{x}-2\mathbf{y}^\top A\mathbf{x}+\mathbf{y}^\top y\right)\\ &= A^\top A\mathbf{x}-A^\top y\\ &= A^\top(A\mathbf{x}-\mathbf{y})\end{aligned}$$

■

■ **Example .18** The softmax function defined over $S:\mathbb{R}^d\to[0,1]^d$ returns a vector that its coordinates sum up to 1. It is defined by

$$S(\mathbf{a})_j=\frac{e^{a_j}}{\sum_{k=1}^d e^{a_k}}$$

Let us calculate the derivative of the softmax function. Denote $g_i(\mathbf{a})\coloneqq e^{a_i}$ and $h(\mathbf{a})\coloneqq\sum_k g_k(\mathbf{a})$. So:

$$\frac{\partial S_i}{\partial a_j}=\frac{\partial}{\partial a_j}\frac{e^{a_i}}{\sum_d e^{a_k}}=\frac{\partial}{\partial a_j}\frac{g_i}{h}$$

Note that for any a_j the derivative of h is e^{a_j}. In the case of g_i, when deriving with respect to a_j we get that the derivative is e^{a_j} only if $i=j$. Otherwise, the derivative is 0. Therefore, the derivative of S_i in the case where $i=j$ is:

$$\frac{\partial}{\partial a_j}\frac{e^{a_i}}{\sum_k e^{a_k}}=\frac{e^{a_i}(\sum_k e^{a_k})-e^{a_i}e^{a_j}}{(\sum_k e^{a_k})^2}=\frac{e^{a_i}}{(\sum_k e^{a_k})}\cdot\frac{(\sum_k e^{a_k})-e^{a_j}}{(\sum_k e^{a_k})}=S_i(1-S_j)$$

It is left to show the derivative in the case where $i \neq j$. Intuitively, the softmax function is a "soft" version of the argmax function. Instead of just selecting one maximal element, the softmax breaks the vector into segments with the maximal input element getting a proportionally larger portion, but the other elements getting some of it as well. ■

B.3 Function Approximations

First Order Approximation Consider a function $f : \mathbb{R} \to \mathbb{R}$ and recall the definition of the Taylor series:

$$T(x_0+x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} x^n = f(x_0) + f'(x_0)x + \frac{1}{2}f''(x_0)x^2 + \dots$$

A linear approximation (or first order approximation) is an approximation of a general function using a linear function. For a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$, Taylor's theorem implies that for small x

$$f(x_0+x) \approx f(x_0) + f'(x_0)x$$

We can now extend this theorem to define linear approximations of multivariate functions.

Definition B.1 — Linear Approximation. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$. The linear approximation of f for every $\mathbf{x}' \in \mathbb{R}$ near $\mathbf{x}$ is defined as

$$f(\mathbf{x}') = f(\mathbf{x}) + \left\langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}' \right\rangle$$

Equivelantly, for $\mathbf{x}'$ denoting the deviation from $\mathbf{x}$, then the linear approximation of f near $\mathbf{x}$ is

$$f(\mathbf{x}+\mathbf{x}') = f(\mathbf{x}) + \left\langle \nabla f(\mathbf{x}), \mathbf{x}' \right\rangle$$

So if for example, we consider a bivariate function $f : \mathbb{R}^2 \to \mathbb{R}$, the linear approximation of f near the point (x_0, y_0) is:

$$f(x_0+x, y_0+y) \approx f(x_0,y_0) + x \cdot \frac{\partial f(x_0,y_0)}{\partial x} + y \cdot \frac{\partial f(x_0,y_0)}{\partial y}$$

Now, if f is a linear function, intuition dictates that the linear approximation of f would be the function itself. Let $\mathbf{b} \in \mathbb{R}^d$ and let $f : \mathbb{R}^d \to \mathbb{R}$ be defined by $f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x}$. Then, the linear approximation of f near $\mathbf{x}$ is

$$\begin{aligned} f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}' \rangle &= \mathbf{b}^\top \mathbf{x} + \langle \mathbf{b}, \mathbf{x} - \mathbf{x}' \rangle \\ &= \mathbf{b}^\top (\mathbf{x} + \mathbf{x}' - \mathbf{x}) = f(\mathbf{x}') \end{aligned}$$

■ **Example .19** Let $f : R^2 \to \mathbb{R}$ be defined by $f(x,y) = \sqrt{x^2+y^2}$. Let us calculate the linear approximation of f near $(3,4)$. We begin with expressing the gradient of f. So, the partial derivative of f with respect to first argument at point x is:

$$\frac{\partial}{\partial x} = f(x_0,y_0) = 2x_0 \cdots \frac{1}{2\sqrt{x_0^2+y_0^2}}$$

Therefore the gradient of f is $\nabla f(\mathbf{x}) = \left(\frac{x_0}{\sqrt{x_0^2+y_0^2}}, \frac{y_0}{\sqrt{x_0^2+y_0^2}} \right)^\top$. So for a point (x,y) in the vicinity of $(3,4)$ the linear approximation is:

$$f(3+x, 4+y) \approx 5 + \frac{3}{5}x + \frac{4}{5}y$$

If for example $x = 0.1, y = 0.2$ then $f(3+0.1, 4+0.2) = 5.2201.. \approx 5.22 = 5 + 3/5 \cdot 0.1 + 4/5 \cdot 0.2$ ■

Second Order Approximation Using the gradient of a function we are able to provide a first order (linear) approximation of a function. Similarly, we can also provide a second order approximation of a function. The second order Taylor expansion of f near $\mathbf{x}$ is given by:

$$f\left(\mathbf{x}+\mathbf{x}'\right) \approx f\left(\mathbf{x}\right)+\nabla f\left(\mathbf{x}\right)^{\top}\mathbf{x}'+\frac{1}{2}\mathbf{x}'^{\top}H\left[f\left(\mathbf{x}\right)\right]\mathbf{x}' \tag{37}$$

■ **Example .20** Returning to the second order polynomial function defined in Example .15, let us find the first- and second-order approximations of f near point $\mathbf{x}_0=(x_0,y_0)^{\top}$. The first order approximation is given by:

$$\begin{aligned} f\left(\mathbf{x}_0+\mathbf{x}\right) &\approx f\left(\mathbf{x}_0\right)+\nabla f\left(\mathbf{x}_0\right)^{\top}\mathbf{x} \\ &= x_0^2+x_0y_0+y_0^2+\begin{bmatrix}2x_0+y_0\\2y_0+x_0\end{bmatrix}^{\top}\begin{bmatrix}x\\y\end{bmatrix} \\ &= x_0^2+x_0y_0+y_0^2+2x_0x+y_0x+2y_0y+x_0y \end{aligned}$$

The second order approximation is given by

$$\begin{aligned} f\left(\mathbf{x}_0+\mathbf{x}\right) &\approx x_0^2+x_0y_0+y_0^2+2x_0x+y_0x+2y_0y+x_0y+\tfrac{1}{2}\begin{bmatrix}x\\y\end{bmatrix}^{\top}\begin{bmatrix}2&1\\1&2\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} \\ &= x_0^2+x_0y_0+y_0^2+2x_0x+y_0x+2y_0y+x_0y++x^2+yx+y^2 \\ &= \left(x_0+x\right)^2+\left(x_0+x\right)\left(y_0+y\right)+\left(y_0+y\right)^2 \end{aligned}$$

Notice that since f is a second order polynomial then the calculated value is the exact value of the function at $\mathbf{x}_0+\mathbf{x}$. ■

B.4 Convexity

Convex Sets A set $C\subseteq\mathbb{R}^d$ is called a *convex set* if and only if $\forall\mathbf{v},\mathbf{u}\in C,\forall\alpha\in[0,1]$ then $\alpha\mathbf{v}+(1-\alpha)\mathbf{u}\in C$. That is, if the line segment between any two vectors $\mathbf{v},\mathbf{u}\in C$, is contained in C. The vector $\alpha\mathbf{v}+(1-\alpha)\mathbf{u}$ is called a *convex combination* of $\mathbf{v}$ and $\mathbf{u}$.

■ **Example .21** Let V be a vector space and $U\subseteq V$. U is a convex set as for every $\mathbf{v},\mathbf{u}\in U$ and $\alpha\in[0,1]$ $\alpha\mathbf{v}+(1-\alpha)\mathbf{u}$ is a linear combination of vectors in U and therefore is also in U. ■

■ **Example .22** Consider the unit ball of some norm $B_{||\cdot||}=\{\mathbf{v}\in V:||\mathbf{v}||\leq 1\}$. This is a convex set as for any $\mathbf{v},\mathbf{u}\in B$ and $\alpha\in[0,1]$, the triangle inequality imples that:

$$\begin{aligned} \|\alpha\mathbf{v}+(1-\alpha)\mathbf{u}\| &\leq \|\alpha\mathbf{v}\|+\|(1-\alpha)\mathbf{u}\| \\ &= \alpha\|\mathbf{v}\|+(1-\alpha)\|\mathbf{u}\| \\ &\leq \alpha+1-\alpha=1 \end{aligned}$$

■

■ **Example .23** Let $(\mathbf{w},b)$ be a hyperplane for $\mathbf{w}\in\mathbb{R}^d$ and $b\in\mathbb{R}$. The closed halfspace $W:=\{\mathbf{v}:\mathbf{w}^{\top}\mathbf{v}\leq b\}$ is a convex set. For any $\mathbf{v},\mathbf{u}\in W$ and $\alpha\in[0,1]$ it holds that:

$$\langle\mathbf{w},\alpha\mathbf{v}+(1-\alpha)\mathbf{u}\rangle=\alpha\langle\mathbf{w},\mathbf{v}\rangle+(1-\alpha)\langle\mathbf{w},\mathbf{u}\rangle\leq\alpha b+(1-\alpha)b=b$$

■

Convexity is preserved under several operations. The claim below demonstrates a few such operations.

Claim B.2 Convexity is preserved under the following operations:

1. The intersection $C:=\bigcap_{i\in I}C_i$ for $\{C_i:i\in I\}$ a collection of convex sets.
2. The vector sum $C_1+C_2:=\{c_1+c_2:c_1\in C_1,c_2\in C_2\}$ of two convex sets.

3. The set $\lambda C \coloneqq \{\lambda c : c \in C\}$ is convex, for any convex set C, and every scalar λ.

Proof. Proving directly from definition:

1. Let $\mathbf{v}, \mathbf{u} \in C$, and let $\alpha \in [0,1]$. As C is the intersection of $\{C_i\}$ it holds that $\mathbf{v}, \mathbf{u} \in C_i$ for any $i \in I$. Since $\{C_i\}$ are convex sets then for any $\alpha \in [0,1]$ it holds that $\alpha\mathbf{v} + (1-\alpha)\mathbf{u} \in C_i$. Thus $\alpha\mathbf{v} + (1-\alpha)\mathbf{u} \in \bigcap_{i \in I} C_i = C$.

2. Let $\mathbf{v}, \mathbf{u} \in C_1 + C_2$, then there exists $\mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{u}_1, \mathbf{u}_2$ for which

$$\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2 \text{ s.t. } \mathbf{v}_1 \in C_1, \mathbf{v}_2 \in C_2, \quad \mathbf{u} = \mathbf{u}_1 + \mathbf{u}_2 \text{ s.t. } \mathbf{u}_1 \in C_1, \mathbf{u}_2 \in C_2$$

   Let $\alpha \in [0,1]$ then:

$$\begin{aligned} \alpha\mathbf{v} + (1-\alpha)\mathbf{u} &= \alpha(\mathbf{v}_1 + \mathbf{v}_2) + (1-\alpha)(\mathbf{u}_1 + \mathbf{u}_2) \\ &= [\alpha\mathbf{v}_1 + (1-\alpha)\mathbf{u}_1] + [\alpha\mathbf{v}_2 + (1-\alpha)\mathbf{u}_2] \end{aligned}$$

   where, from convexity of C_1, C_2 it holds that:

$$\alpha\mathbf{v}_1 + (1-\alpha)\mathbf{u}_1 \in C_1, \quad \alpha\mathbf{v}_2 + (1-\alpha)\mathbf{u}_2 \in C_2$$

   and therefore $\alpha\mathbf{v} + (1-\alpha)\mathbf{u} \in C_1 + C_2$

3. Let $\mathbf{v}, \mathbf{u} \in \lambda C$, then there exists $\mathbf{x}, \mathbf{y} \in C$ such that $\mathbf{v} = \lambda\mathbf{x}$ and $\mathbf{u} = \lambda\mathbf{y}$. Let let $\alpha \in [0,1]$ them:

$$\alpha\mathbf{v} + (1-\alpha)\mathbf{u} = \lambda(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \in \lambda C.$$

■

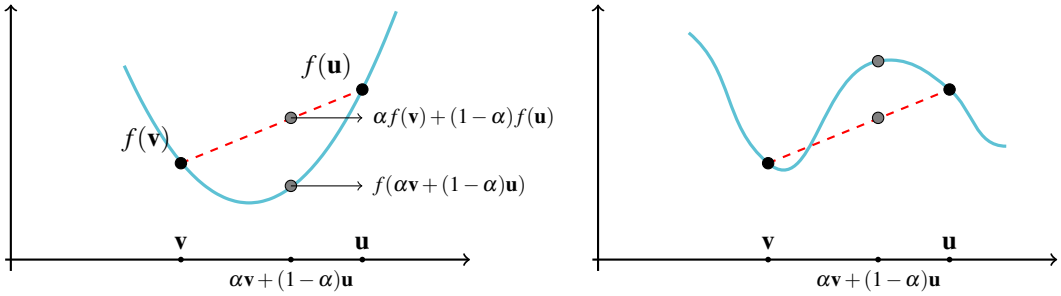
In a similar manner we can show that the vector sum of a finite set of convex sets is convex, giving a convex combination of arbitrary length.

(a) *convex sets examples* ***(b)*** *Non-convex sets examples*

Figure 19: *Examples of convex- and non-convex sets*

Convex Functions Given a convex set, we can define the notion of a convex function. A function $f : C \to \mathbb{R}$ is *convex* if and only if $C \equiv \text{dom} f$ is convex and for every $\mathbf{u}, \mathbf{v} \in C$ and every $\alpha \in [0,1]$ then $f(\alpha\mathbf{v} + (1-\alpha)\mathbf{u}) \leq \alpha f(\mathbf{v}) + (1-\alpha) f(\mathbf{u})$.

This means that the line segment connecting any two points on the curve of a convex function f lies fully above that curve. In other words, the value of a convex function f at any convex combination $\mathbf{v}$ and $\mathbf{u}$ is always smaller than the convex combination of the values of f at $\mathbf{v}$ and $\mathbf{u}$.



(a) *Convex function* ***(b)*** *Non-convex function*

Figure 20: *Illustrating convex vs. non-convex function*

■ **Example .24** Let $||\cdot|| : \mathbb{R}^d \to \mathbb{R}$ be a norm, $\mathbf{v},\mathbf{u} \in \mathbb{R}^d$ and $\alpha \in [0,1]$. From, the triangle inequality it holds that:

$$\|\alpha\mathbf{v}+(1-\alpha)\mathbf{u}\| \leq \|\alpha\mathbf{v}\| + \|(1-\alpha)\mathbf{u}\| = \alpha\|\mathbf{v}\| + (1-\alpha)\|\mathbf{u}\|$$

■

■ **Example .25** Let $f : \mathbb{R}^d \to \mathbb{R}$ be an affine transformation, that is $f(\mathbf{x}) = \langle \mathbf{w},\mathbf{x}\rangle + b$ for $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Let $\mathbf{v},\mathbf{u} \in \mathbb{R}^d$ and $\alpha \in [0,1]$ then:

$$\begin{aligned} f(\alpha\mathbf{v}+(1-\alpha)\mathbf{u}) &= \langle \mathbf{w}, \alpha\mathbf{v}+(1-\alpha)\mathbf{u}\rangle + b \\ &= \alpha(\langle \mathbf{w},\mathbf{v}\rangle + b) + (1-\alpha)(\langle \mathbf{w},\mathbf{u}\rangle + b) \\ &= \alpha f(\mathbf{v}) + (1-\alpha) f(\mathbf{u}) \end{aligned}$$

■

■ **Example .26** Let $f(\mathbf{w}) \coloneqq ||\mathbf{X}\mathbf{w}-\mathbf{y}||^2$ for $\mathbf{X} \in \mathbb{R}^{m\times d}, \mathbf{y} \in \mathbb{R}^m$ and $\mathbf{w} \in \mathbb{R}^d$. Let us show that f is convex in $\mathbf{w}$. Let $\mathbf{v},\mathbf{u} \in \mathbb{R}^d$ and $\alpha \in [0,1]$ then:

$$\begin{aligned} f(\alpha\mathbf{v}+(1-\alpha)\mathbf{u}) &= ||\mathbf{X}[\alpha\mathbf{v}+(1-\alpha)\mathbf{u}]-\mathbf{y}||^2 \\ &= ||\alpha(\mathbf{X}\mathbf{v}-\mathbf{y})+(1-\alpha)(\mathbf{X}\mathbf{v}-\mathbf{y})||^2 \\ &\leq \alpha||\mathbf{X}\mathbf{v}-\mathbf{y}||^2 + (1-\alpha)||\mathbf{X}\mathbf{u}-\mathbf{y}||^2 \\ &= \alpha f(\mathbf{v}) + (1-\alpha) f(\mathbf{u}) \end{aligned}$$

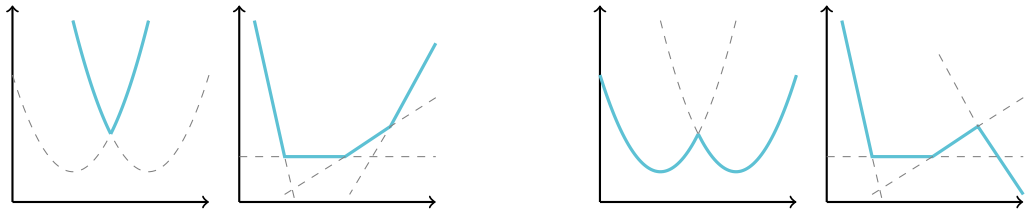
■

Some commonly used convex functions are the exponent $x \to e^{ax}, \forall a$, $x \to x^a, \forall a \notin (0,1)$ and negative logarithm $x \to -\log(x)$. Furthermore, as can be seen from Example .26 affine transformations $\mathbf{x} \to \mathbf{w}^\top\mathbf{x}+\mathbf{b}$ are convex as well as quadratic tarnsformations are convex $\mathbf{x} \to \mathbf{x}^\top A\mathbf{x}+\mathbf{w}^\top\mathbf{x}+\alpha$ for $A \succcurlyeq 0$. The family of ℓ_p norms (Example A.1) defined as $||\mathbf{x}||_p \coloneqq \left(\sum x_i^p\right)^{1/p}$ is convex for $p \geq 1$.

On the basis of some known convex functions, we can define a set of closure properties of convexity. The following are a few of such operations that preserve convexity:

- Non-negative linear combinations preserve convexity. That is, for $g(\mathbf{x}) = \sum \alpha_i f_i(\mathbf{x})$, then g is convex if f_i are convex functions are a_i are non-negative.
- The function $g(\mathbf{x}) = \sup_i f_i(\mathbf{x})$ is convex if f_i are convex functions.
- Partial minimization of a function preserves convexity. That is, for a convex function $g : \mathbb{R}^{d+k} \to \mathbb{R}$ defined by $(\mathbf{x}_1|\mathbf{x}_2) \to g(\mathbf{x}_1|\mathbf{x}_2)$ for $\mathbf{x}_1 \in \mathbb{R}^d, \mathbf{x}_2 \in \mathbb{R}^k$, then the partial minimization function $h : \mathbb{R}^d \to \mathbb{R}$ defined by $h(\mathbf{x}_1) = \min_{\mathbf{x}_2 \in C} g(\mathbf{x}_1,\mathbf{x}_2)$ (where $C \subset \mathbb{R}^k$ is a convex set) is a convex function.

- For $g : \mathbb{R}^d \to \mathbb{R}$ convex function and $h : \mathbb{R} \to \mathbb{R}$ convex and nondecreasing, the composition $h \circ g$ is a convex function.
- For $h : \mathbb{R}^k \to \mathbb{R}$ convex and nondecreasing in each of its coordinates and $g_1, \ldots, g_k : \mathbb{R}^d \to \mathbb{R}$ convex functions, then $f(\mathbf{x}) = h(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$ is a convex function.



(a) *Convex functions* ***(b)*** *Non-convex functions*

Figure 21: *Examples of convex- and non-convex functions illustrating closure properties*

Properties of Convex Functions Convex functions have many useful properties, making them simple to optimize, i.e. finding the input that minimizes them.

> **Claim B.3 — First order characterization.** Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differential function, then f is convex if and only if $\mathrm{dom} f \subseteq \mathbb{R}^d$ is a convex set and $f(\mathbf{u}) \geq f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w}) \rangle \ \ \forall \mathbf{u}, \mathbf{w} \in \mathrm{dom} f$.

Proof. Without loss of generality, assume $\mathbf{w} = \mathbf{0}$ since otherwise we could simply shift the axis. Therefore, it is enough to show that for any $\mathbf{u}$ $f(\mathbf{u}) \geq f(\mathbf{0}) + \langle \nabla f(\mathbf{0}), \mathbf{u} \rangle$. As f is convex it holds that

$$\begin{aligned} f(\alpha \mathbf{u}) \ &\leq \ (1-\alpha) f(\mathbf{0}) + \alpha f(\mathbf{u}) \\ &\Downarrow \\ \frac{f(\alpha \mathbf{u}) - f(\mathbf{0})}{\alpha} \ &\leq \ f(\mathbf{u}) - f(\mathbf{0}) \end{aligned}$$

This holds for any $\alpha \in [0,1]$ and thus taking the limit of both sides, with $\alpha \to 0$, and recalling that $\lim_{\alpha \to 0} \frac{f(\alpha \mathbf{u}) - f(\mathbf{0})}{\alpha} = \langle \mathbf{u}, \nabla f(\mathbf{0}) \rangle$ concluding the proof. ■

Namely, at any point on the graph of a convex (and differential) function f, the tangents lie below the graph of the function.

■ **Example .27** Let $f : \mathbb{R}^d \to \mathbb{R}$ be a linear functional defined by $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ for $\mathbf{w} \in \mathbb{R}^d$. Let us show that f is convex using the characterization above. For any $\mathbf{x} \in \mathbb{R}^d$ it holds that $\nabla f(\mathbf{x}) = \mathbf{w}$. Therefore, for a point $\mathbf{y} \in \mathbb{R}^d$ it holds that

$$f(\mathbf{y}) = \langle \mathbf{w}, \mathbf{y} \rangle = \langle \mathbf{w}, \mathbf{x} \rangle + \langle \mathbf{y} - \mathbf{x}, \mathbf{w} \rangle = f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle$$

■

Notice that the affine transformation of $\mathbf{u}$ that is given by $f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w}) \rangle$ is the first-order Taylor approximation of f near $\mathbf{x}$. Suppose we wish to estimate (i.e approximate) the value of f near a point of interest. We therefore learn that for a convex f the first-order Taylor approximation is a *global underestimator*. Furthermore it shows that in the case of a convex function, the *local* information about the function (i.e the gradient at $f(\mathbf{w})$) allows us to derive *global* information about it. An example for such information is seen in the following claim.

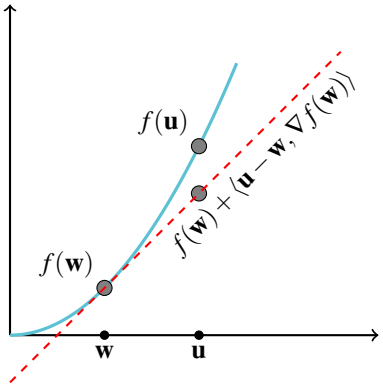


Figure 22: *Tangent of Convex Function*

Claim B.4 Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. If $f(\mathbf{u})$ is a local minimum then it is a global minimum.

Proof. Let $f : C \to \mathbb{R}$. and denote $B(\mathbf{u}, r)$ the intersection of C and a sphere of radius r around the point $\mathbf{u}$:

$$B(\mathbf{u}, r) \coloneqq \{\mathbf{v} : \mathbf{v} \in C, \|\mathbf{v} - \mathbf{u}\| \leq r\}$$

Let $\mathbf{u}$ be a local minimizer of f (i.e. $f(\mathbf{u})$ is a local minimum). Therefore, there exists $r > 0$ such that for any $\mathbf{v} \in B(\mathbf{u}, r)$ it holds that $f(\mathbf{u}) \leq f(\mathbf{v})$. Note that since f is a convex function, its domain, C, is a convex set - otherwise, the convex combination of two vectors in C may not belong to C and the convexity condition in definition would not be well-defined. Let $\mathbf{v} \in C$ (not necessarily in B) and take $\alpha > 0$ close enough to 1 such that $\alpha\mathbf{u} + (1-\alpha)\mathbf{v}$ is inside B. Since C is convex, $\alpha\mathbf{u} + (1-\alpha)\mathbf{v}$ is in C. It follows that to ensure that $\alpha\mathbf{u} + (1-\alpha)\mathbf{v}$ is in B it is enough to choose an α close enough to 1 so that the distance between $\alpha\mathbf{u} + (1-\alpha)\mathbf{v}$ and $\mathbf{u}$ is smaller than r. Since $\alpha\mathbf{u} + (1-\alpha)\mathbf{v}$ is inside B and by definition, one has

$$f(\mathbf{u}) \leq f(\alpha\mathbf{u} + (1-\alpha)\mathbf{v}) \;\leq\; \alpha f(\mathbf{u}) + (1-\alpha) f(\mathbf{v}) \,.$$

Hence, $(1-\alpha)f(\mathbf{u}) \;\leq\; (1-\alpha)f(\mathbf{v})$, that is, $f(\mathbf{u}) \;\leq\; f(\mathbf{v})$. This holds for every $\mathbf{v}$, hence $f(\mathbf{u})$ is also a global minimum of f. ■

Claim B.5 — Second order characterization. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a twice differentiable function, then f is convex if and only of $\mathrm{dom} f \subset \mathbb{R}^d$ is a convex set and $\nabla^2 f(\mathbf{x}) \succcurlyeq 0$, $\forall \mathbf{x} \in \mathrm{dom} f$

■ **Example .28** Let $A \in \mathbb{R}^{d \times d}$ be a symmetric matrix and let $f : \mathbb{R}^d \to \mathbb{R}$ defined as $f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$. Let us show that f is convex using the characterization above. To do so we find the first and second derivatives of f. Denote $g(\mathbf{x}) = A\mathbf{x}$ and $h(\mathbf{x}) = \mathbf{x}$ so we can write f as $f \equiv h^\top g$. Using the product rule then:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial h(\mathbf{x})^\top}{\partial \mathbf{x}} \cdot g(\mathbf{x}) + \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} \cdot h(\mathbf{x})$$

As the derivative of g by $\mathbf{x}$ is A then:

$$\nabla f(\mathbf{x}) = A^\top \mathbf{x} + A\mathbf{x} = \left(A + A^\top\right)\mathbf{x}$$

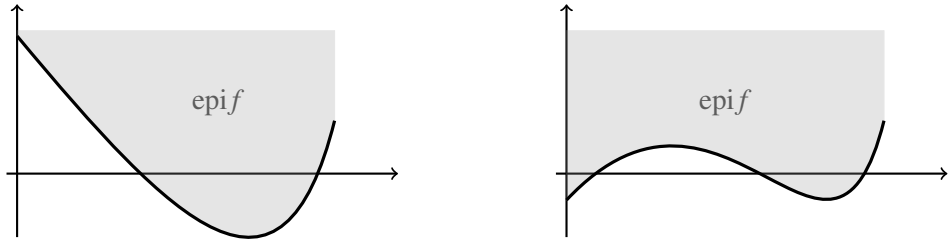
Since A is symmetric then $\nabla f(\mathbf{x}) = 2A\mathbf{x}$. Next, let us compute the second derivative:

$$\nabla^2 f(\mathbf{x}) = \frac{\partial^2 f}{\partial^2 \mathbf{x}} = \frac{\partial 2A\mathbf{x}}{\partial \mathbf{x}} = 2A$$

Thus, by the characterization above f is convex if and only if $A \succeq 0$. That is, if and only if A is a PSD matrix.
■

R Notice that the requirement of $\text{dom}f$ to be a convex set is necessary in both the first- and second-order characterizations.

Given a function $f : \mathbb{R}^d \to \mathbb{R}$, the *graph* of the function is defined as the set $\{(\mathbf{x}, f(\mathbf{x})) \,|\, \mathbf{x} \in \text{dom}f\} \subseteq \mathbb{R}^{d+1}$. The *epigraph* of f is the set of all points above f $\text{epi}f \coloneqq \{(\mathbf{x}, \beta) \,|\, \mathbf{x} \in \text{dom}f, f(\mathbf{x}) \leq \beta\}$. The connection between convex sets and convex functions is through the epigraph of the function. A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if *epi*f is a convex set.



(a) *Convex function & convex epigraph*

(b) *Non-convex function & non-convex epigraph*

B.5 Exercises

Theoretical Questions

1. Show that the set of real PSD matrices $V \coloneqq \{A \in \mathbb{R}^{d \times d} | A \in PSD\}$is a convex set.

2. Show that the image and pre-image of an affine transformation are convex sets.

3. Show that the image and pre-image of the linear-fraction function are convex sets. The linear-fraction function is defined as $f(\mathbf{x}) = \frac{A\mathbf{x}+\mathbf{b}}{\mathbf{c}^\top \mathbf{x}+\mathbf{d}}$

4. Show that quadratic functions $\mathbf{x} \to \mathbf{x}^\top A\mathbf{x} + \mathbf{w}^\top \mathbf{x} + \alpha$ are convex if and only if $A \succcurlyeq 0$.

5. Show that for any $p \geq 1$ the norm $||\mathbf{x}||_p \coloneqq \left(\sum x_i^p\right)^{1/p}$ is a convex function.

6. Show that the log-sum-exp function, defined as $f(\mathbf{x}) = \log\left(\sum_{i=1}^k \exp\left(\mathbf{w}_i^\top \mathbf{x} + b_i\right)\right)$ is a convex function. Hint: use the fact that affine compositions preserve convexity and the second order characterization.

7. Show that the following function is a convex function: $f(\mathbf{w}) \coloneqq \frac{1}{m}\sum_i \log(1 + \exp(-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle))$ for $\mathbf{w}, \mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. This function is the loss function used in the logistic regression optimization problem (3.27).

8. Let $f(\mathbf{w})$ be a differentiable convex function. Show that, beside the tangent at $\mathbf{w}$, no other line passing through the point $(\mathbf{w}, f(\mathbf{w}))$ lies fully below $f(\mathbf{w})$. In other words, show that if $\forall \mathbf{u} \in \mathbb{R}^d$, $f(\mathbf{u}) \geq f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \mathbf{a} \rangle$ then $\mathbf{a} = \nabla f(\mathbf{w})$.

9. Show that $f(\mathbf{w}) \coloneqq \min_{i\in[m]} \left|\mathbf{w}^\top \mathbf{x}_i\right|$ for $\mathbf{x}_1, \ldots, \mathbf{x}_m$ is a concave function. Namely, that $-f$ is a convex function. This function is the margin which maximized in the Hard-SVM optimization problem (3.12).

10. Show that in both the first- and second-order characterization of a convex function, the requirement that $dom(f)$ is a convex set is necessary. That is, describe non-convex functions f whose domain is not a convex set and that $\nabla f \geq 0$ or $\nabla^2 f \succcurlyeq 0$.