Content

1 Various Basics

1.1 Probability Notation

Technically, random variables, like X, are functions.

- $Val(X) =$ set of possible values
- $X: \Omega \to Val(X)$, means that the RV X maps each outcome $\omega \in \Omega$ to a value $X(\omega)$ in $Val(X)$

1.2 Discrete Variables

A random variable is *discrete* if its possible values are finite/countably infinite. For instance:

$$Val(X) = \{0,1\} \text{ or } Val(X) = \mathbb{N}$$

In this case, we denote the probability of the event that X has value x by $p(X = x)$.

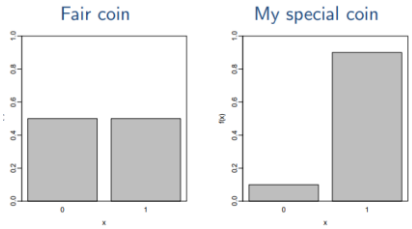
The *probability mass function (pmf)* f_X, is a function which *computes the probability of events* which correspond to setting the random variable to each of its possible values.

$$f_X: Val(X) \to [0, 1]$$

If X has a finite number of values, say K, the *pmf* can be represented as a list of K numbers, which we can plot as a histogram as in the distributions below.

1.2.1 Bernoulli Distribution $\mathbf{Ber}(\boldsymbol{\theta})$

A Bernoulli trial (or binomial trial) is a random experiment with exactly two possible outcomes, "success" and "failure", in which the probability of success is the same every time the experiment is conducted. For example, it models a coin flip with probability θ of heads.

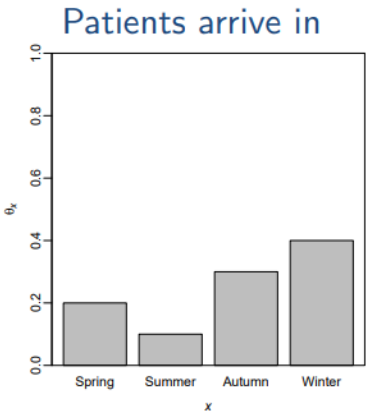


- $Val(X) = \{0,1\}$ ($\to$ *binary* RV)
- $f_X(1) = \theta$
- $f_X(0) = 1 - \theta$

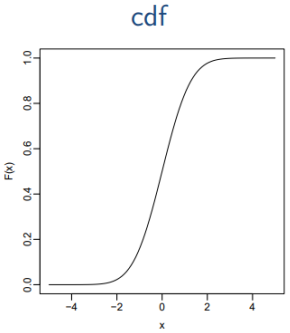
1.2.2 Categorical Distribution $\mathbf{Cat}(\boldsymbol{\theta})$

Generalization of Bernoulli distribution to $k > 0$ categories.

- The probabilities of categories are given by $\boldsymbol{\theta} = (\theta_1, \dots, \theta_k)$, summing to 1.
- Throwing a dice would be a categorical distribution over 6 categories, for a fair one with $\theta_k = \theta = \frac{1}{6}$.



1.3 Continuous Variables



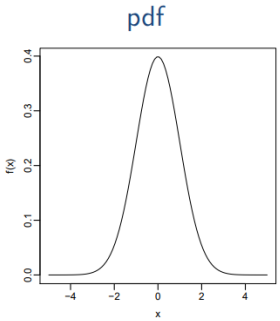
A random variable is *continuous* if its possible values uncountably infinite. For instance: $Val(X) = \mathbb{R}$. They are described by a *cumulative distribution function* (*cdf*).

$$F_X: Val(X) \rightarrow [0,1]$$

Note that we use a capital F to represent the *cdf*. Using this, we can compute the probability of being in any interval as follows:

$$p(a < Y \leq b) = F_X(b) - F_X(a)$$

Cdf's are monotonically non-decreasing functions.



The *probability density function* (*pdf*) f_X is defined as the derivative of the cdf:

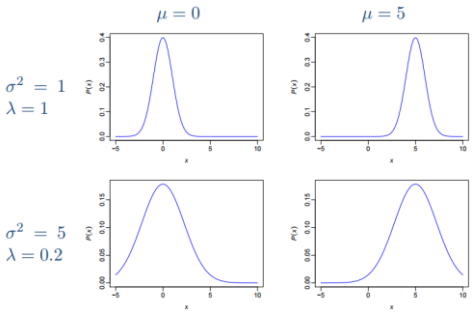
$$f_X(x) = \frac{d}{dx} F_X(x)$$

We can compute the probability of being in any interval as follows:

$$p(a \leq X \leq b) = \int_a^b f_X(x)\, dx$$

Which corresponds to the area below the curve.

1.3.1 Gaussian Distribution



Also called *Normal Distribution* $\mathcal{N}(\mu, \sigma^2)$.

- μ is the mean parameter
- σ^2 is a variance parameter

The *pdf* is given by:

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{1}{2\sigma^2}(x-\sigma)^2}$$

1.3.2 Mean, Variance, Covariance

The mean or *expected value* μ of a random variable is given by:

$$E[X] = \sum_{x \in Val(X)} x * f_X(x)$$

if X is discrete. This means we sum over the realizations x of the random variable and weigh them by their probability of occurring $f_X(x)$. Similarly:

$$E[X] = \int_x x * f_X(x)\ dx$$

if X is continuous.

Variance σ^2 is a measure of how much a single random variable varies from its mean.:

$$var[X] = E[(X - \mu)^2] = E[X^2] - E[X]^2$$

It provides a measure of the spread of a distribution along one dimension.

Covariance extends this concept to two dimensions by measuring how much two random variables change together. Covariance gives some sense of how much two values are linearly related to each other, as well as the scale of these variables:

$$Cov[X, Y] = E[(X - E[X]) * (Y - E[Y])]$$

It's the expected value of the product of their deviations from their respective means. If the covariance is positive, it means that the two variables tend to increase or decrease together; if it's negative, one tends to increase when the other decreases.

1.4 Likelihood

Suppose that you have a stochastic process that takes discrete values (e.g., outcomes of tossing a coin 10 times). In such cases, we can calculate the probability of observing a particular set of outcomes by making suitable assumptions about the underlying stochastic process (e.g., probability of coin landing heads is θ and that coin tosses are independent).

Denote the observed outcomes by $\mathcal{D}$ and the set of parameters that describe the stochastic process as θ. Thus, when we speak of probability, we want to calculate $p(\mathcal{D}|\theta)$. In other words, given specific values for θ, $p(\mathcal{D}|\theta)$ is the probability that we would observe the outcomes represented by $\mathcal{D}$.

However, when we model a real-life stochastic process, we often do not know θ. We simply observe $\mathcal{D}$ and the goal then is to arrive at an estimate for θ that would be a plausible choice given the observed outcomes $\mathcal{D}$. We know that given a value of θ the probability of observing $\mathcal{D}$ is $p(\mathcal{D}|\theta)$. Thus, a 'natural' estimation process is to choose that value of θ that would maximize the probability that we would actually observe $\mathcal{D}$. In other words, we find the parameter values θ that maximize the following function:

$$\mathcal{L}(\theta|\mathcal{D}) = p(\mathcal{D}|\theta)$$

$\mathcal{L}(\theta|\mathcal{D})$ is called the likelihood function. Notice that by definition the likelihood function is conditioned on the observed $\mathcal{D}$ and that it is a function of the unknown parameters θ.

1.5 Floating-Point Arithmetic

Suppose you're trying to multiply very large/small numbers:

$$300\,000\,000 * 0.00000015$$

This can easily be done by viewing them in their scientific notation:

$$3.0 * 10^8 * 1.5 * 10^{-7}$$
$$= 4.5 * 10^{8+(-7)}$$
$$= 4.5 * 10^1$$
$$= 45$$

1.5.1 Number Representation in Computers

Computers store numbers in base 2. This means that the number 10 is represented by 1010:

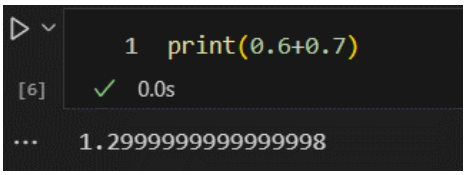
8	4	2	1	·	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
1	**0**	**1**	**0**	·	**0**	**0**	**0**	**0**

The same way the number $\frac{1}{3} = 0.\overline{33}$ in base 10 cannot be properly represented without the concept of recursion ($0.\overline{33}$ repeated infinitely), numbers such as $\frac{1}{10} = 0.1$ would have to be represented as:

2^3	2^2	2^1	2^0	·	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0	**0**	**0**	**0**	·	**0**	**0**	**0**	**1**	**1**

with the last four digits repeating indefinitely: $0000.0\overline{0011}$.

As these bits need to be stored in a finite way, 32-bit computers only store 23 digits, meaning that the repeating sequence ends afterwards, resulting in a small rounding error. Modern computers allow for a higher precision.



1.5.2 LogSumExp-Trick

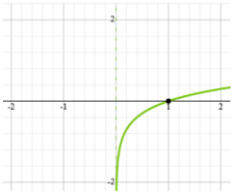
When working with really small numbers, e.g., multiplication of probabilities, we encounter numerical stability issues.

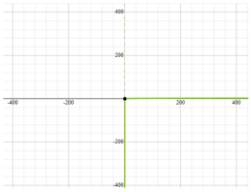
> *Underflow* occurs when numbers near zero are rounded to zero.
>
> *Overflow* occurs when numbers with large magnitude are approximated as ∞ or $-\infty$.

The logarithm drastically reduces the magnitude of a number.

$$\ln e^{500} = 500$$

$$\log_{10} 10^{-1000} = -1000$$





Suppose we want to calculate

$$\ln p^+ = \ln \sum_{i=1}^{n} e^{\ln p_i}$$

If $\ln p_i$ is very big, then $exp(\ln p_i)$ will over-/underflow. To reduce numerical stability issues, we can transform it using $c = \max_i \ln p_i$:

$$\begin{aligned}
\ln p^+ &= \ln \sum_{i=1}^{n} p_i \\
&= \ln \sum_{i=1}^{n} e^{\ln p_i} \\
&= \ln \sum_{i=1}^{n} e^{\ln(p_i)-c+c} \\
&= \ln \sum_{i=1}^{n} e^{\ln(p_i)-c} * e^{c} \\
&= \ln \left[e^{c} \sum_{i=1}^{n} e^{\ln p_i - c} \right] \\
&= \ln e^{c} + \ln \sum_{i=1}^{n} e^{\ln p_i - c} \\
&= c + \ln \sum_{i=1}^{n} e^{\ln p_i - c}
\end{aligned}$$

This allows us to perform operations in the log-space and then recovering the right solution without the problem of numerical instability.

1.6 Derivatives

Notation

If $f(x) = y$, then the following notation is the same: $f'(x) = y' = \frac{df}{dx} = \frac{dy}{dx} = \frac{d}{dx}[f(x)] = Df(x)$

For a derivate *evaluated at* $x = a$ this is the same: $f'(a) = y'|_{x=a} = \frac{df}{dx}|_{x=a} = \frac{dy}{dx}|_{x=a} = Df(a)$

1.6.1 Basic Properties and Formulas

If $f(x)$ and $g(x)$ are differentiable functions (meaning the derivative exists), and c and n are any real numbers.

If you have a **constant c multiplied by a function** $f(x)$, the derivative of this product is equal to the constant c times the derivative of the function $f'(x)$.

$$\big(cf(x)\big)' = c * f'(x)$$

The **derivative of a constant** c is zero.

$$\frac{d}{dx}(x) = 0$$

Sum Rule

$$\big(f(x) \pm g(x)\big)' = f'(x) \pm g'(x)$$

Product Rule

$$\big(f(x) * g(x)\big)' = f'(x) * g(x) + f(x) * g'(x)$$

Quotient Rule

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x) * g(x) - f(x) * g'(x)}{g(x)^2}$$

Power Rule

$$\frac{d}{dx}(x^n) = nx^{x-1}$$

Chain Rule

$$\frac{d}{dx}\big[f\big(g(x)\big)\big] = f'\big(g(x)\big) * g'(x)$$

1.6.2 Common Derivatives

$\frac{d}{dx}(x) = 1$

$\frac{d}{dx}(\sin x) = \cos x$

$\frac{d}{dx}(\cos x) = -\sin x$

$\frac{d}{dx}(a^x) = a^x * \ln a$

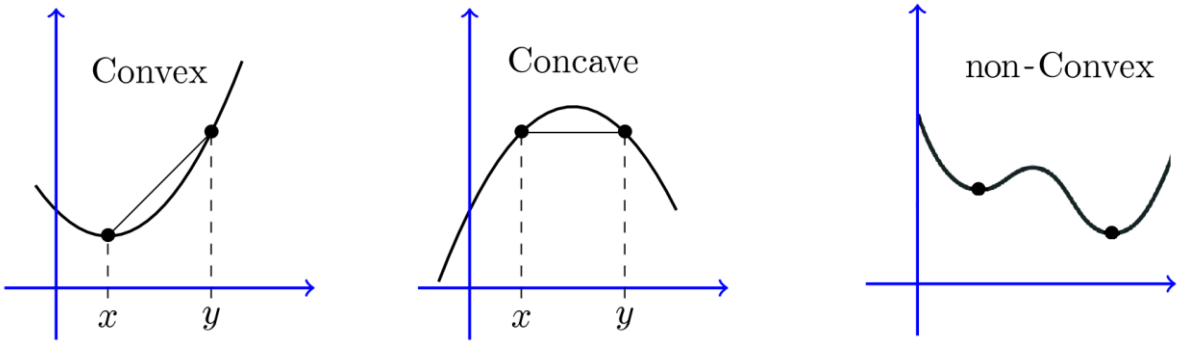
$\frac{d}{dx}(e^x) = e^x$

$\frac{d}{dx}(\ln x) = \frac{1}{x}, x > 0$

$\frac{d}{dx}(\log_a x) = \frac{1}{x * \ln a}, x > 0$

1.7 Non-Convex & Convex

📄 Convex (loss) functions have a one unique, global minimum. Convexity is a desirable property in optimization because it ensures that gradient-based optimization methods can reliably find the global minimum.



❗ In practice, we often encounter *non-convex* loss functions that have multiple, local minima. These local minima can be challenging to deal with, as traditional gradient-based optimization methods may get stuck in suboptimal local minima. Although finding a good local optimum, may in practice be good enough.

2 Introduction

2.1 Supervised Learning

In supervised learning, the model learns a *mapping* from inputs $\boldsymbol{x}$ to outputs y given a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ of training examples. N denotes the number of these examples.

The mapping may be a simple function $y = f(x)$ outputting a single value. It often takes different forms, for example, expressing the output in terms of confidence or degrees of belief. These *probabilistic models* give a distribution over outputs given a particular input $p(y|x)$. Other models learn a joint density which is a distribution over inputs and outputs $p(x, y)$

Each training input $\boldsymbol{x}_i$ is a D-dimensional vector of numbers, often called *features*, attributes or covariates. They are often stored in a $N \times D$ *design matrix* $\boldsymbol{X}$, with the corresponding *labels* (targets) as a N-dimensional vector $\boldsymbol{y}$.

	$\boldsymbol{X}$			$\boldsymbol{y}$	
$\boldsymbol{x}_1$	2	4	8	Good	y_1
$\boldsymbol{x}_2$	4	8	16	Good	y_2
$\boldsymbol{x}_3$	1	2	1	Bad	y_3

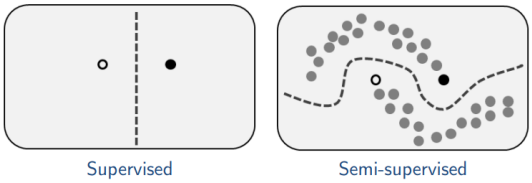
Types of Classification

- Binary Classification / Concept Learning
- Multiclass Classification
- Multi-Label Classification

2.2 Semi-Supervised Learning

In semi-supervised learning (SSL), we are given a labeled training set $\mathcal{D}_L = \{(\boldsymbol{x}_l, y_l)\}_{l=1}^L$ and an unlabeled training set $\mathcal{D}_U = \{\boldsymbol{x}_u\}_{u=1}^U$.

💡 An example of the influence of unlabeled data in semi-supervised learning. The left panel shows a decision boundary we might adopt after seeing only one positive and one negative example. The right panel shows a decision boundary we might adopt if, in addition to the two labeled examples, we were given a collection of unlabeled data.



Variants

- *Transductive learning*: infer labels of $\mathcal{D}_U$ only, no classifier ist learned
- *Inductive learning*: learn mapping from $\boldsymbol{x}$ to y

2.3 Unsupervised Learning

Finds "interesting" patterns in the data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$. For example, *density estimation*, which learns properties of the data distribution, or *clustering*, which divides data into groups.

Unsupervised *representation learning* learns useful representations or features of the data. For instance, mapping (potentially complex) data points into a low-dimensional *latent space* that retains or reveals the data's main structure (cf. Auto-Encoders).

2.4 Parametric vs. Non-Parametric Models

In a *parametric model*, the number of parameters is fixed. They can be ranging from a few parameters (e.g., linear/logistic regression on low-dimensional data) to billions of parameters (e.g., large language models).

❗ Esp. when few: Can be faster to use but may make stronger assumptions about nature of data.

In a *non-parametric model*, the number of parameters grows with the amount of training data. E.g., k-nearest neighbor classifier. This is more flexible but can be computationally intractable.

2.5 Curse of Dimensionality

Methods such as KNN may not work well with high-dimensional inputs as distance metrics become less effective and meaningful in high-dimensional spaces.

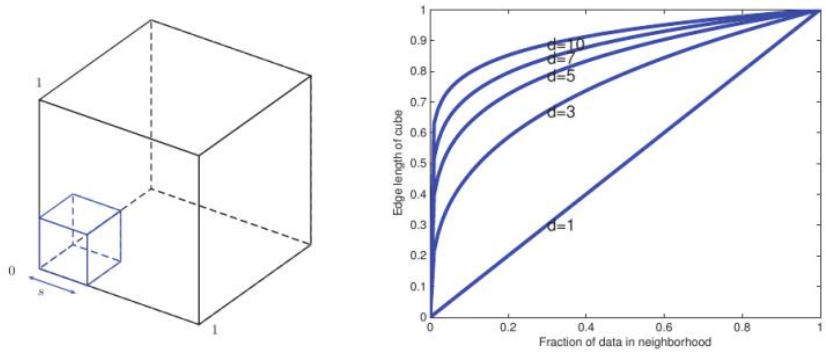


Figure 1.16 Illustration of the curse of dimensionality. (a) We embed a small cube of side s inside a larger unit cube. (b) We plot the edge length of a cube needed to cover a given volume of the unit cube as a function of the number of dimensions. Based on Figure 2.6 from (Hastie et al. 2009). Figure generated by `curseDimensionality`.

2.6 Linear Regression

The main way to combat this curse of dimensionality is to *make assumptions* about the distribution of data, e.g., by using a parametric model. Linear regression is such a regression model assuming that:

$$y(\boldsymbol{x}) = \sum_{j=1}^{D} w_j x_j + \epsilon$$

where $\{w_j\}_{j=1}^{D}$ are real-valued parameters and ϵ is the residual error. D represents the total number of input features in our dataset. For a specific data point, w_j represents the weight associated with the j-th feature and x_j represents the value of it.

> 💡 In this context, "linear" means that the relationship between the features and their combination is a straight line when plotted on a graph. It implies that a change in any one feature will have a proportional effect on the combined value, and the overall behavior is straightforward and easy to analyze.
>
> A linear combination is an expression constructed from a set of terms by multiplying each term by a constant and adding the results (e.g. a linear combination of x and y would be any expression of the form ax + by, where a and b are constants).

📄 The set of assumptions we place by choosing a specific model is known *as inductive bias*. This model makes quite strong assumptions, in particular that the output is a linear function of the input.

We can also replace $\boldsymbol{x}$ by a set of features $\{\phi_j(\boldsymbol{x})\}$ and assume $y(\boldsymbol{x}) = \sum_{j=1}^{D} w_j \phi_j(\boldsymbol{x}) + \epsilon$. This is known as *basis function expansion* or *feature engineering*

2.7 Model Selection

The chosen model class plays a big role in determining the model's *representational capacity*, essentially defining the scope of classifiers that can be feasibly constructed within the constraints of that specific model class. It outlines which types of patterns and relationships the model is inherently capable of capturing and which it cannot.

On the other hand, the learning algorithm, along with all its associated components and parameters, governs the model's *effective capacity*. This encompasses the model's ability to harness its *representational capacity* to adapt and generalize from the provided data.

> Golden Rule of Machine Learning
>
> The *test set* should not influence the learning process in any way, therefore *cannot* be used for model selection.

For model selection, we care about generalization error, which is the expected misclassification rate over future data. This can be approximated by computing misclassification rate on a sufficiently large, independent, and representative *test set*.

3 Inference and Decision

3.1 Generative & Discriminative Models

In a narrow sense, *training* refers to the process of fitting the model parameter on a data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$. In a broader sense it also includes model selection, hyperparameter tuning, etc.

Inference refers to the process of reasoning about the output y_{new} given a new input $\boldsymbol{x}_{new}$. This is often done by computing the posterior class probabilities $p(y_{new}|\boldsymbol{x}_{new})$. Depending on the on the model, this task can be trivial or very hard and computationally expensive. The term *inference* may also be used to reason about other unknown variables.

Decision means to predict a particular label $\hat{y}_{new}$ (in the classification sense). One may also refrain from a decision → *reject option*, for example, if the classifier is unsure.

3.1.1 Discriminative Functions

A discriminative function $f(\boldsymbol{x})$ is a function that directly maps each input $\boldsymbol{x}$ to a class label $\hat{y}$. There are no probabilities involved, hence the model cannot tell how confident it is. Inference and decision are merged. E.g., k-nearest neighbor with majority voting (no probabilistic interpretation) or SVMs.

3.1.2 Discriminative Models

Models the posterior class probability $p(y|\boldsymbol{x})$ directly. Given an input $\boldsymbol{x}$ they can reason about the output y, but know nothing per se about the distribution of the inputs themselves. E.g., k-nearest neighbor with probabilities.

> 💡 k-nearest neighbor with probabilities $N_K(x, \mathcal{D})$
>
> Views the number of votes for the respective class in a neighborhood as the confidence:
>
> $$p(y = c|\boldsymbol{x}, \mathcal{D}, K) = \frac{1}{K} \sum_{i \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_i = c)$$
>
> with the indicator function $\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{else} \end{cases}$
>
> This notation represents the probability that a particular data point $\boldsymbol{x}$ belongs to a specific class c given some dataset $\mathcal{D}$ within the context of a neighborhood defined by K. The term $N_K(\boldsymbol{x}, \mathcal{D})$ simply refers to the KNN classifier itself.

3.1.3 Generative Models

Models the joint distribution of inputs and outputs $p(\boldsymbol{x}, y)$. Often this done in two pieces, by:

- Modeling the *class-conditional densities* $p(\boldsymbol{x}|y)$ for each class y individually. This is the distribution of the input $\boldsymbol{x}$ for a particular class y.
- Modeling the *prior class probabilities* $p(y)$. This tells how likely it is to see the individual labels y, regardless of the input $\boldsymbol{x}$

to then combine those by the product rule $p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y)\, p(y)$. If this should be used for inference, simply apply the Bayes' theorem to get the *posterior class probability*:

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)\, p(y)}{p(\boldsymbol{x})}$$

By marginalizing over y (sum over all possible labels y') we can calculate $p(\boldsymbol{x}) = \sum_y p(\boldsymbol{x}, y')$. These models, for example Naïve Bayes, GANs, Restricted Bolzman Machines, can also generate data, e.g., given an output, they can generate suitable inputs.

Example:

$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^3$

$\boldsymbol{x}$		
x_1	x_2	y
Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

$p(y)$

State	p(State)
poor	2/3
rich	1/3

$p(\boldsymbol{x}|y)$

$p(\boldsymbol{x}\|$poor$)$	young	old
student	1/2	1/2
CEO	0	0

$p(\boldsymbol{x}\|$rich$)$	young	old
student	0	0
CEO	0	1

Using MLE to obtain relative frequencies for prior distribution $p(y)$ and class-conditional densities $p(\boldsymbol{x}|y)$.

The joint distribution is estimated as

$p(\boldsymbol{x}$, poor$)$	young	old
student	1/3	1/3
CEO	0	0

$p(\boldsymbol{x}$, rich$)$	young	old
student	0	0
CEO	0	1/3

3.1.4 Summary

Generative $p(\boldsymbol{x}, \boldsymbol{y})$	**Discriminative $p(\boldsymbol{y}|\boldsymbol{x})$**	**Function $f(\boldsymbol{x})$**
Modeling inputs and outputs jointly is *generally demanding*	Provides confidence values	Easy to use
Some models make very *strong assumptions*	Avoids modeling the input $\boldsymbol{x}$ which can be complex.	Risky because uncertainty about confidence/accuracy
If assumptions hold, they need *less training data* and are more accurate	Above also allows using a richer feature set, less stringent assumption	Hard to combine models
If not, $\boldsymbol{p(y|x)}$ may not be *well-calibrated*. NB may be over-confident.		
Can handle *missing data* in a principled way.		

3.2 Parameter Estimation

Suppose we are given a model class with parameters $\boldsymbol{\theta}$

- Generative $p(\boldsymbol{x}, y|\boldsymbol{\theta})$
- Discriminative $p(y|\boldsymbol{x}, \boldsymbol{\theta})$
- Discriminative function $f(\boldsymbol{x}, \boldsymbol{\theta})$

and some training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, we want to estimate the value of $\boldsymbol{\theta}$ using $\mathcal{D}$.

This task is called *training* or *learning*. We want to select parameters that fit the training data well and most importantly generalize well to unseen data. A specific choice of values for the parameters is called *point estimate* $\widehat{\boldsymbol{\theta}}$.

We *may* not even need $\widehat{\boldsymbol{\theta}}$ itself, as our prime goal is to make predictions. To do so, we need some information about $\boldsymbol{\theta}$, but not necessarily a point estimate. Instead, a *posterior* distribution $p(\boldsymbol{\theta}|\mathcal{D})$ tells us which choices of $\boldsymbol{\theta}$ are likely after having seen the data.

💡 These are two fundamentally different approaches to do parameter estimation and ultimately prediction.

3.2.1 Properties of Estimators

Consider the estimation of an of a single parameter $\theta \in \mathbb{R}$ in a *frequentist setting*.

- *True parameter* θ^* that determines the corresponding data distribution p^*
- Training data $\mathcal{D}$ is given by N iid. samples from p^*
- Estimator $\hat{\theta}$ computes a point estimate $\hat{\theta}(\mathcal{D})$ from the sampled training data set

An example for such an estimator could be the sample mean $\hat{\theta}(\mathcal{D}) = \frac{1}{N}\sum x_i$ for $\mathcal{D} = \{x_i\}_{i=1}^N$.

> 💡 Interpretations of Probabilities
>
> The *frequentist interpretation* views probabilities are frequencies of events. The probability of an event is the fraction of times the event occurs if repeated infinitely.
>
> Unknown parameters are often assumed to have a fixed but unknown value $\boldsymbol{\theta}^*$, which is the true and correct value → hence it would be inappropriate to treat this as a random variable. $\boldsymbol{\theta}^*$ determines the (true) data distribution p^*.
>
> The *Bayesian interpretation* views probabilities as degrees of belief that an event holds. This degree of belief depends on background knowledge and assumptions and is influenced by seeing data.
>
> Here the parameters are treated as random variables with associated probability distributions. Analysis is done w.r.t. posterior probabilities $p(\boldsymbol{\theta}|\mathcal{D})$.
>
> asks: "what is my degree of belief about the distribution of $\boldsymbol{\theta}$ after having seen the data?"

One property of an estimator in a frequentist setting is the *bias*:

$$bias[\hat{\theta}] \stackrel{\text{def}}{=} E[\hat{\theta} - \theta^*]$$

The bias of $\hat{\theta}$ is defined as the expected value of the difference between the estimate $\hat{\theta}$ we get from our sample and the true parameter value θ^*. This expectation is taken wrt. The N samples of the data distribution p^*, i.e., $E_{D\sim p^*}[\hat{\theta}(\mathcal{D}) - \theta^*]$. If $bias[\hat{\theta}] = 0$, then the estimator is *unbiased* (correct in expectation). Otherwise, it is biased.

The second property of an estimator $\hat{\theta}$ is the *variance*:

$$var[\hat{\theta}] = E\left[(\hat{\theta} - E[\hat{\theta}])^2\right]$$

which is the average squared deviation between the value of an estimator and its mean. The variance provides a measure of how much the estimator's values tend to vary around its mean value. To put it simply:

- $\hat{\theta}$ is the specific estimate you calculate from your data
- $E[\hat{\theta}]$ is the average or expected value of $\hat{\theta}$ if you were to repeat the estimation process multiple times with different data samples

The third property of an estimator $\hat{\theta}$ is the *mean squared error (MSE)*.

$$mse[\hat{\theta}] = E\left[(\hat{\theta} - \theta^*)^2\right]$$

which is an error measure for using $\hat{\theta}$ instead of the true value θ^*.

3.2.2 Bias-Variance Tradeoff

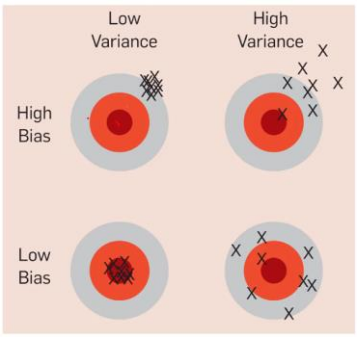
Bias-Variance Decomposition

$$mse[\hat{\theta}] = bias[\hat{\theta}]^2 + var[\hat{\theta}]$$

This gives rise to the *bias-variance tradeoff,* as we ultimately want to have a model where both properties are low. It also very roughly states that

- Simple models → high bias, low variance
- Complex models → low bias, high variance

Above formula is not computable as we do not know the true data distribution p^*.

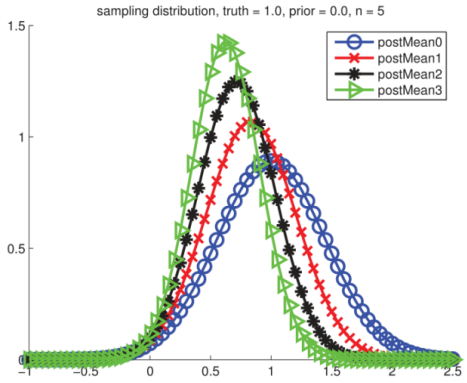


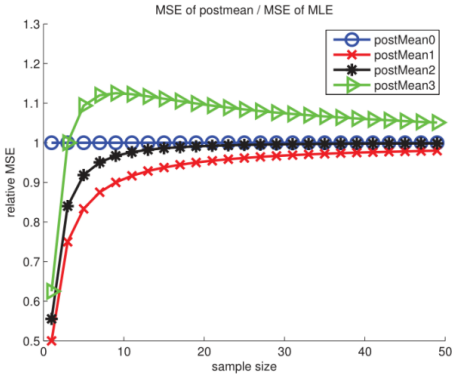
Top left: underfitting; bottom right: overfitting

Note that *mse* in this case is not the prediction error but the *parameter estimation error*.

For the task of predicting the mean of a normal distribution $\mathcal{N}(\theta^* = 1, \sigma^2)$, with the true mean being 1, we have different estimators.

- Blue is the *sample mean* → it has a low bias but a high variance
- Red is a downscaled sample mean by $\frac{N}{N+1}$ → is has a larger bias but less variance & MSE





By downscaling the sample mean by $\frac{N}{N+1}$, you are introducing a regularization term into your estimator. The downscaling *introduces a controlled amount of bias* because the estimator is no longer the exact sample mean, but rather a scaled-down version of it. When you downscale the sample mean by $\frac{N}{N+1}$, you are giving more weight to the sample mean itself and less weight to the individual data points. This means that the sample mean dominates the estimate, and the impact of each data point is reduced. As a result, the estimator becomes less sensitive to outliers or noise in the data.

3.2.3 Maximum Likelihood Estimation (MLE)

The likelihood of an observed data set and a parameter choice θ:

$$\mathcal{L}(\theta|\mathcal{D}) \overset{\text{def}}{=} \begin{cases} p(\mathcal{D}|\theta) & \text{for generative models} \\ p(\boldsymbol{y}|\boldsymbol{X},\theta) & \text{for discriminative models} \end{cases}$$

The larger the likelihood, the more consistent is the data with the parameter choice.

Imagine a coin flip with 4 iid. samples, each showing heads. The unknown parameter θ^* represents the probability of heads.

I) $\mathcal{L}(\theta = 0|n_H = 4) = 0$

II) $\mathcal{L}(\theta = 0.5|n_H = 4) = 0.5 * 0.5 * 0.5 * 0.5 = 0.0625$

III) $\mathcal{L}(\theta = 1|n_H = 4) = 1$

The *likelihood ratio* is the ratio of the likelihood of the data under two different hypotheses.

In this case, consider II) and III): $\frac{\mathcal{L}(\theta = 1|n_H = 4)}{\mathcal{L}(\theta = 0.5|n_H = 4)} = \frac{1}{0.5*0.5*0.5*0.5} = 16$

Meaning that the data is 16 times more likely under the hypothesis that $\theta = 1$ compared to the hypothesis that $\theta = 0.5$.

MLE chooses the value $\hat{\theta}$ (point est.) that maximizes the (conditional) likelihood of the data:

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}}\, \mathcal{L}(\theta|\mathcal{D})$$

This can only be used for *probabilistic models,* as discriminative functions, for example, do not provide probabilities and hence not likelihood.

MLE has good asymptotic properties, meaning when $N \to \infty$ with iid. samples:

- Consistent: it converges to the true value θ^* in probability (i.e., as N increases, the probability that the estimator converges to the true parameter value approaches 1)
- Efficient: no other estimator has lower asymptotic MSE
- Asymptotically normally distributed

❗ MLE is generally biased, but the bias tends to *decrease*, when the number of sample increases. Although, in practice we have limited data, hence the estimator tends to overfit to training data.

3.2.4 Risk Minimization

Often, we may not be interested in the parameters (θ) itself, but rather in the corresponding predictions. To quantify the quality of the models' predictions, we can use a *loss function.*

Suppose we are given such a non-negative, real valued *loss function* $L(\hat{y}, y)$ that measures how different prediction $\hat{y}$ is from the true answer y. For example:

- Classification → *0-1 loss*: $L(\hat{y}, y) = \mathbb{I}(y \neq \hat{y})$ with indicator fct. $\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{else} \end{cases}$
- Regression → *squared loss*: $L(\hat{y}, y) = (\hat{y} - y)^2$

3.2.4.1 True Risk $R(h)$

Assume that the data follows a distribution $p^*(\boldsymbol{x}, y)$.

The *risk* $R(h)$ associated with a *hypothesis* h is the expected loss over the data distribution:

$$R(h) = E_{(\boldsymbol{x},y)\sim p^*}[L(h(\boldsymbol{x}), y)] = \int \int L(h(\boldsymbol{x}), y)\, p^*(\boldsymbol{x}, y)\ \mathrm{d}\mathbf{x}\, \mathrm{d}y$$

with $h(\boldsymbol{x}) = \hat{y}$ being the models' prediction and $p^*(\boldsymbol{x}, y)$ being the likelihood of a data point.

However, as we do not know the true data distribution p^* we cannot calculate the risk. Instead, what we can compute though, is the *empirical risk.*

3.2.4.2 Empirical Risk $R_{emp}(h)$

The *empirical risk* $R_{emp}(h)$ is the average *loss* $L(\hat{y}, y)$ on the training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$:

$$R_{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} L(h(\boldsymbol{x_i}), y_i)$$

The Empirical Risk Minimization principle suggests choosing the hypothesis/estimator that minimizes $R_{emp}(h)$:

$$\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ R_{emp}(h)$$

As discussed, this typically results in overfitting to the training data if the *hypothesis space* is very large and can also be a difficult optimization problem.

3.2.4.3 Regularized Risk Minimization

Regularized risk minimization $R'_{emp}(h)$ tries to avoid overfitting by adding a *penalty term*:

$$R'_{emp}(h) = R_{emp}(h) + \lambda C(h)$$

where $C(h)$ measures the *complexity* of the model and λ controls the strength of the penalty. Ideally, $\lambda C(h)$ is chosen close to the generalization gap $R(h) - R_{emp}(h)$. This method can be used for discriminative functions as well as probabilistic models.

3.2.5 Prior & Posterior

Prior belief $p(\theta)$

- Incorporates prior knowledge, e.g., coin is likely to be fair or likely to show heads.
- Different priors may lead to different results → subjective aspect

Posterior belief $p(\theta|D)$

- Updated belief *after* seeing the data
- E.g., is coin fair after seeing 10 heads & 2 tails?
- Depends on prior and likelihood via Bayes' theorem

$$p(\theta|D) = \frac{p(D|\theta)\,p(\theta)}{p(D)} \propto p(D|\theta)\,p(\theta)$$

$$\boldsymbol{posterior \propto likelihood \times prior}$$

3.2.6 Maximum a posteriori (MAP) Estimation

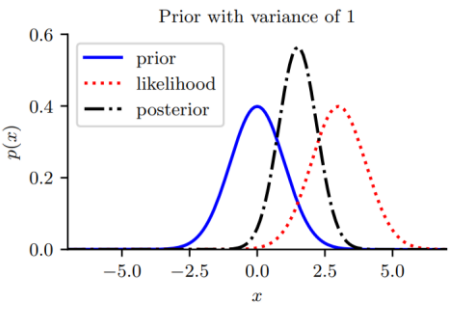
Recall: BMs reason about the belief that a parameter takes a single value. This belief can be captured in a probability distribution that assigns probability densities to possible parameter values. The two types of belief are:

- *Prior* belief $p(\theta)$ about a parameter
- *Posterior* belief $p(\theta|\mathcal{D})$ about a parameter *after* having seen some data

The *MAP estimate* $\hat{\theta}_{MAP}$ is the point estimate that maximizes the posterior

$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathcal{D}) = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\mathcal{D})\, p(\theta)$$

or, in other words, the *most probable parameter choice given data* $\mathcal{D}$ and prior $p(\theta)$. This prior weight can help to avoid overfitting.



3.2.7 Coin Flip Example: MLE vs. MAP

Suppose, we are perfoming n iid. coin flips and observe n_H heads. The parameter θ denotes the probability of showing heads.

Maximum Likelihood Estimation

We flip 10 and observe $n_H = 5$. MLE chooses the value $\hat{\theta}$ that maximizes the likelihood of the data:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\mathcal{D})$$

and for generative models:

$$\mathcal{L}(\theta|\mathcal{D}) = p(\mathcal{D}|\theta)$$

For a coin flip the liklihood is given by the binomial distribution:

$$p(\mathcal{D}|\theta) = \binom{10}{5} * \theta^5 * (1-\theta)^{(10-5)}$$

Now, to find the MLE for θ, we need to differentiate this $p(\mathcal{D}|\theta)$ with respect to θ, set the derivative equal to zero, and solve for θ.

> The binomial distribution is appropriate for modeling the number of successes (heads) in a fixed number of iid Bernoulli trials (coin flips). It is used when you have a series of trials, each with two possible outcomes (heads/tails), and you want to know the probability of getting a specific number of successes in those trials.

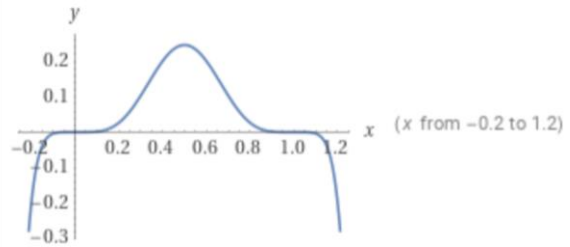
Input

$$p(\mathcal{D}|\theta) = \binom{10}{5} * \theta^5 * (1-\theta)^{(10-5)}$$

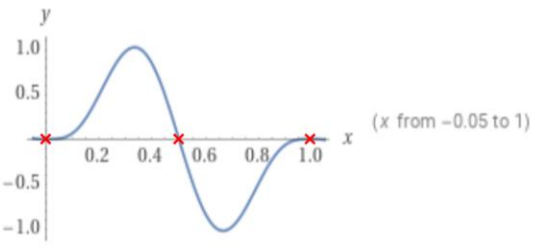
Derivative

$$\frac{d}{dx} p(\mathcal{D}|\theta) = -1260 * \theta^4 * (2\theta\text{-}1) * (1\text{-}\theta)^4$$

Plots



Plots



Solutions

$\theta = 0$ $\quad \theta = \frac{1}{2} = \hat{\theta}_{MLE}$ $\quad \theta = 1$

Maximum a posteriori Estimation

MAP estimate $\hat{\theta}_{MAP}$ is the point estimate that maximizes the posterior:

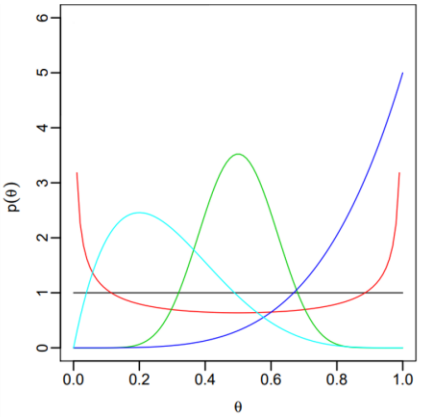
$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathcal{D}) = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\mathcal{D})\, p(\theta)$$

i.e., the most probable parameter choice given data $\mathcal{D}$ and prior $p(\theta)$. Since,

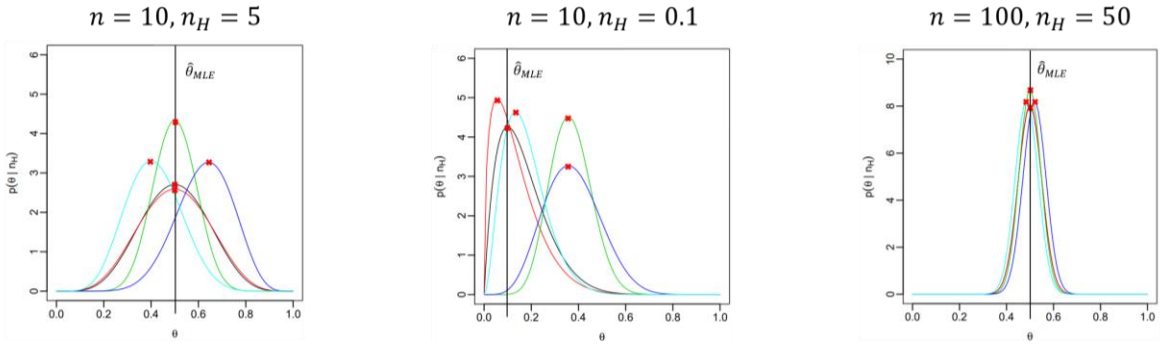
$$\mathcal{L}(\theta|\mathcal{D}) = p(\mathcal{D}|\theta) = \binom{10}{5} * \theta^5 * (1-\theta)^{(10-5)}$$

still holds (see above), we can multiply this by the different priors $p(\theta)$, differentiate this expression with respect to θ, set the derivative equal to zero, and solve for θ to find the MAP estimate.

Different *prior* beliefs $p(\theta)$



Depending on the prior belief, this results in:



3.2.8 Fully Bayesian Inference

💡 The fully Bayesian approach is to avoid parameter estimation (no MLE, no MAP) and instead reason about at all possible parameter choices using the *posterior predictive distribution*. It looks at the probability of observing new data $\mathcal{D}_{new}$ given the data already observed.

$$p(\mathcal{D}_{new}|\mathcal{D}) = \int p(\mathcal{D}_{new}|\theta)\, p(\theta|\mathcal{D})\ \mathrm{d}\theta = \int p(\mathcal{D}_{new}, \theta|\mathcal{D})\ \mathrm{d}\theta$$

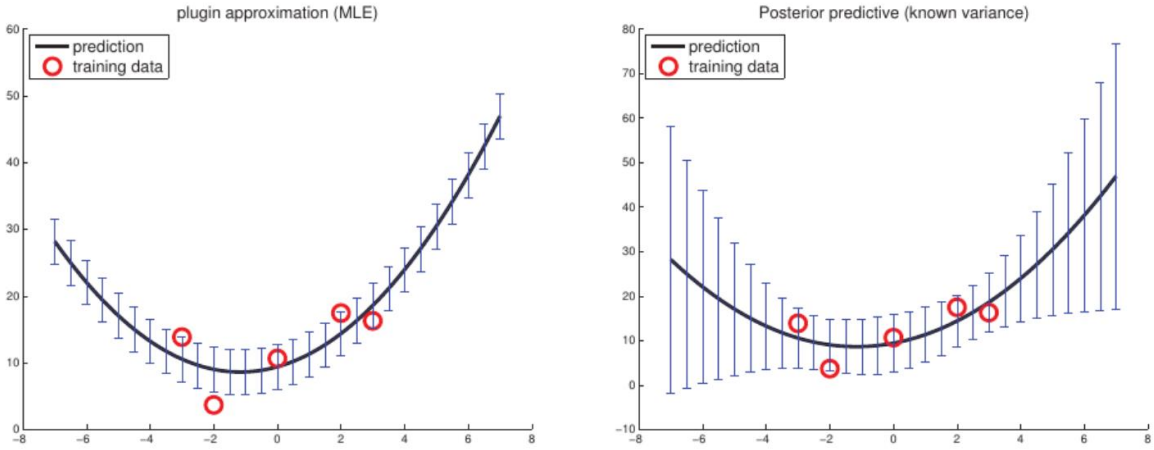
❗ It assumes that $\mathcal{D}_{new}$ is independent from $\mathcal{D}$, but ultimately stems from the same underlying distribution.

Intuitively, it combines predictions of every hypothesis $p(\mathcal{D}_{new}|\theta)$ weighted by its posterior probability, resulting in the joint distribution $p(\mathcal{D}_{new}, \theta|\mathcal{D})$. The integration marginalizes out θ.

$$\textbf{posterior predictive} = \int_\theta \textbf{new-data likelihood} \times \textbf{posterior}\ \mathrm{d}\theta$$

Likewise for discriminative models:

$$p(\boldsymbol{y}_{new}|\mathcal{D}, \boldsymbol{X}_{new}) = \int p(\boldsymbol{y}_{new}|\theta, \boldsymbol{X}_{new})\, p(\theta|\mathcal{D})\ \mathrm{d}\theta$$
$$\left(= E_{\theta \sim p(\theta|\mathcal{D})}[p(y|\theta, x)]\right)$$



This model is more certain around the area in which the observed data samples $\mathcal{D}$ are, and rather uncertain outside of this area. It allows to quantify the un-/certainty, which MLE cannot.

3.2.9 Discussion

- All methods (except Bayesian Inference) above obtain point estimates
  - Empirical risk minimization and MLE are prone to overfitting
  - Regularized risk minimization uses model complexity penalty to avoid this
  - MAP estimation uses prior to steer away from unlikely models
  - Estimates are obtained by solving an optimization problem
- fully Bayesian approach weights every hypothesis by its posterior probability
  - takes uncertainty in parameter estimates into account
  - estimates are obtained by performing probabilistic inference
  - can be quite expensive to use in practice

3.3 Bayesian Decision Theory

Once we know the *posterior predictive distribution* (cf. *3.2.8 Fully Bayesian* Inference) or the *posterior class probabilities* (cf. *3.1 Generative & Discriminative Models*), how should we decide?

In the frequentist setting risk is used, but in the Bayesian setting decision-making is often guided by the principles of decision theory → considering not only the likelihood of different outcomes but also our preferences and objectives.

We try to pick a suitable action $a \in \mathcal{A}$, in particular we want to derive a policy $\delta: \mathcal{X} \to \mathcal{A}$ to decide which action to take. We consider known data $\boldsymbol{x} \in \mathcal{X}$ and unknown data $y \in \mathcal{Y}$.

We can quantify how good an action a is if we know y by using a loss function $L(a, y)$. But since we do not know y, we consider the *expected loss*:

$$E[L(a, y)] = \sum_{y} L(a, y)\, p(y|\boldsymbol{x})$$

The *posterior predictive distribution* $p(y|\boldsymbol{x})$ tell us what the distribution of this unknown data y is, based on all data $\boldsymbol{x}$ we've seen so far. The *optimal policy* minimizes the expected loss:

$$\delta(\boldsymbol{x}) = \operatorname*{argmin}_{a \in \mathcal{A}} E[L(a, y)]$$

This decision is called the *bayes estimator*. This different than risk, as we're not using the true data distribution (as it is unknown) but rather the *posterior predictive distribution*.

Examples

- Classification with misclassification rate (0-1 loss)
  - Bayes estimator: pick most probable class

$$\hat{y} = \text{argmax}_y\, p(y|\boldsymbol{x})$$

  - Called Bayes optimal classifier
- Regression with MSE ($\ell 2$ loss)
  - Bayes estimator: pick posterior mean

$$\hat{y} = E[y|\boldsymbol{x}] = \int y * p(y|\boldsymbol{x})\ dy$$

4 Generative Models for Discrete Data

So far, we have discussed several ways to estimate parameters from data. However, these approaches ignore any uncertainty in the estimates. In this section, we use the *posterior distribution* $p(\boldsymbol{\theta}|D)$ to represent our uncertainty about $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta}|D) = \frac{p(D|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(D)} = \frac{p(D|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int p(D|\boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, \mathrm{d}\boldsymbol{\theta}'}$$

Now suppose we want to predict future observations. A very common approach is to first compute an estimate of the parameters based on training data, $\widehat{\boldsymbol{\theta}}(\mathcal{D})$, and then to plug that parameter back into the model and use $p\left(y|\widehat{\boldsymbol{\theta}}\right)$ to predict the future; this is called a *plug-in approximation*.

However, this can result in overfitting. Once we have computed the posterior $p(\boldsymbol{\theta}|D)$ over the parameters, we can compute the *posterior predictive distribution* over outputs given inputs by marginalizing out the unknown parameters:

$$p(\boldsymbol{y}|\boldsymbol{x}, D) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) * p(\boldsymbol{\theta}|D)\, d\boldsymbol{\theta}$$

This is the *Fully Bayesian Inference*, since we are making predictions using an infinite set of models/parameter values, each one weighted by how likely it is. This reduces the chance of overfitting, since we are not just using the single best model.

4.1 Beta-Binomial Model

The Beta-Binomial model is simple generative model for iid. coin flips.

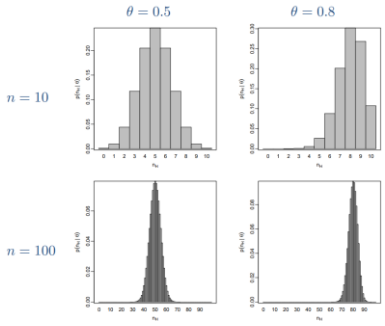
4.1.1 Binomial Distribution $\mathbf{Bin}(\boldsymbol{n}, \boldsymbol{\theta})$

The *binomial distribution* models the number of successes in a sequence of n independent experiments, each asking a yes–no question: success (with probability θ) or failure $(1-\theta)$.

For a coin flip, let the random variable $X \in \{0, \ldots, n\}$ be the number of heads observed. X follows the $\mathrm{Bin}(n, \theta)$.

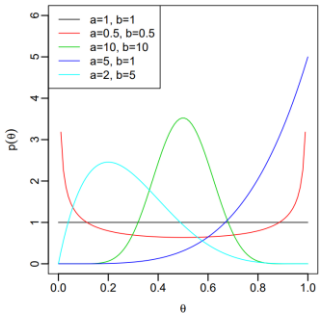
The *pmf* of observing n_H times heads for n trials is defined as:

$$\mathrm{Bin}(n_H|n, \theta) = \binom{n}{n_H} \theta^{n_H} * (1-\theta)^{n-n_H}$$



- Expected value: θn
- Likelihood: $\mathcal{L}(\theta|n_H) = p(n_H|\theta)$ is given by $\mathrm{Bin}(n_H|n, \theta)$
- Posterior: $p(\theta|n_H) = \frac{p(n_H|\theta)\, p(\theta)}{p(n_H)} \propto p(n_H|\theta)\, p(\theta)$

4.1.2 Beta Distribution Beta(α, β)



The Beta distribution is a distribution over a continuous random variable $\theta \in [0,1]$, which is often used to represent the probability for some binary event (e.g., the θ in the Bernoulli distribution). Another way to use it, is to *express the prior belief* $p(\theta)$ in the beta-binomial model.

The pdf of the Beta Distribution $Beta(\alpha, \beta)$ is defined as:

$$Beta(\theta|\alpha,\beta) = \frac{\theta^{\alpha-1} * (1-\theta)^{\beta-1}}{B(\alpha,\beta)}$$

While the similar looking *Binomial Distribution* $\text{Bin}(n_H|n,\theta)$ is a dist. on the number of heads, the Beta Distribution is a dist. on θ.

The hyperparameters $\alpha, \beta \in \mathbb{R}^+$ can be interpreted as *pseudo-counts,* where $\alpha - 1$ corresponds to the pseudo count for the successes and $\beta - 1$ for the failures. More generally, the higher alpha/beta, the more mass is put on the success/failure probability.

Mean

$$E[X] = \frac{\alpha}{\alpha+\beta}$$

Mode (point of largest density)

$$\frac{\alpha-1}{\alpha+\beta-2} \quad for\ \alpha,\beta > 1$$

$B(\alpha,\beta)$ is called the *beta function*. It does not depend on θ and acts as a normalizing constant to make sure $Beta(\theta|\alpha,\beta)$ integrates to 1:

$$B(\alpha,\beta) = \int_0^1 \theta^{\alpha-1} * (1-\theta)^{\beta-1} \ d\theta$$

4.1.3 Posterior Distribution (→ MAP)

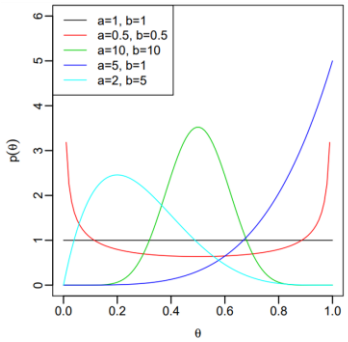
If we choose a Beta Distribution as a *prior* probability, the resulting *posterior* distribution $p(\theta|n_H)$ is also a Beta Distribution for a binomial likelihood (→ *conjugate prior*).

$$posterior \propto likelihood \times prior$$

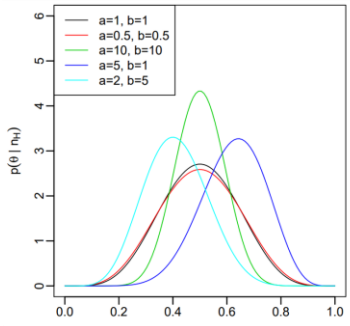
In fact, the *posterior* beta distribution it is:

$$p(\theta|n_H) = Beta(\theta|\alpha + n_H, \beta + n_T)$$

Different beta distribution *priors*



Resulting beta distribution *posteriors* after observing $n = 10, n_H = 5$



In this case, prior and posterior were from the same family of distributions. Prior that achieve this are called *conjugate priors* for the respective likelihood. So, the beta distribution is a *conjugate prior* for the binomial likelihood.

4.1.4 Posterior Predictive Distribution

Suppose we do another coin flip X_{n+1}. How likely is it to show heads, based on what we learned? For this, we seek the posterior predictive distribution:

$$p(X_{n+1}|n_H)$$

- We know the likelihood $p(X_{n+1} = \mathrm{H}|\theta) = \theta$, but we do not know θ.
- We know the posterior $p(\theta|n_H) = Beta(\theta|\alpha + n_H, \beta + n_T)$.

Derivation

As $X_{n+1} \perp n_H \mid \theta$, because the coin flips are iid., this holds:

$$p(X_{n+1} = \mathrm{H}|\theta, n_H) = p(X_{n+1} = \mathrm{H}|\theta) \tag{1}$$

Then, by using Posterior Distribution *4.1.3*, we do:

$$p(X_{n+1} = \mathrm{H}|\theta) * p(\theta|n_H) = \theta * Beta(\theta|\alpha + n_H, \beta + n_T) \tag{2}$$

using the definition of the *Beta Distribution* $\mathbf{Beta}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *4.1.2:*

$$\theta * Beta(\theta|\alpha + n_H, \beta + n_T) = \theta * \frac{\theta^{\alpha+n_H-1} * (1-\theta)^{\beta-1+n_T}}{B(\alpha + n_H, \beta + n_T)} = \frac{\theta^{\alpha+n_H} * (1-\theta)^{\beta-1+n_T}}{B(\alpha + n_H, \beta + n_T)} \tag{3}$$

This has a similar form as the posterior, where we simply "see" one head more.

Recall *3.2.8*, where **posterior predictive** $= \int_\theta$ **new-data likelihood** $\times$ **posterior** $\mathrm{d}\theta$

By (2) we also get the joint distribution $p(X_{n+1} = \mathrm{H}, \theta|n_H)$, which we use to marginalize out θ:

$$\begin{aligned} p(X_{n+1} = \mathrm{H}|n_H) &= \int_0^1 p(X_{n+1} = \mathrm{H}, \theta|n_H) \; d\theta \\ &= \frac{\int_0^1 \theta^{\alpha+n_H} * (1-\theta)^{\beta-1+n_T} \; d\theta}{B(\alpha + n_H, \beta + n_T)} \\ &= \frac{B(\alpha + 1 + n_H, \; \beta + n_T)}{B(\alpha + n_H, \; \beta + n_T)} \\ &= \frac{B(\alpha + n_H, \; \beta + n_T)}{B(\alpha + n_H, \; \beta + n_T)} * \frac{\alpha + n_H}{\alpha + \beta + n} = \frac{\alpha + n_H}{\alpha + \beta + n} \end{aligned} \tag{4}$$

which was further simplified by the property of the beta function: $B(x+1, y) = B(x, y) * \frac{x}{x+y}$.

Result

In this case, the *posterior predictive distribution* is:

$$p(X_{n+1} = \mathrm{H}|n_H) = \mathrm{Ber}\left(X_{n+1} \middle| \theta = \frac{n_H + \alpha}{n + \alpha + \beta}\right)$$

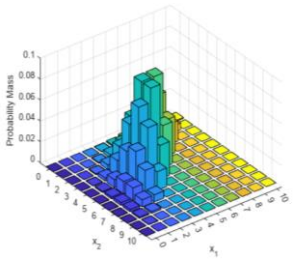
where $\frac{n_H+\alpha}{n+\alpha+\beta}$ is the posterior distributions mean.

❗ This assumes that new data is conditionally independent of old data, given the parameters, which is true for a coin flip.

4.2 Dirichlet-Multinomial Model

In this section, we generalize the results from the previous section from binary variables (e.g., coins) to *K-ary* variables (e.g., dice).

4.2.1 Multinomial Distribution $\mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, \boldsymbol{n})$



The Multinomial Distribution $\mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, n)$ models for a given 1.2.2Categorical Distribution $\mathbf{Cat}(\boldsymbol{\theta})$ $\boldsymbol{\theta} = (\theta_1 \quad \dots \quad \theta_K)^{\mathrm{T}}$, how often you are going to see each category $\boldsymbol{n} = (n_1 \quad \dots \quad n_K)^{\mathrm{T}}$ after observing n iid. trials. The *pmf,* or likelihood $\mathcal{L}(\boldsymbol{\theta}|n)$:

$$\mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, n) = \binom{n}{n_1 \cdots n_K} \prod_{k=1}^{K} {\theta_k}^{n_k}$$

with θ_k being the probability of each category and n_k the number of times we've seen them.

All of the different orders we can observe, are given by the *multinomial coefficient*:

$$\binom{n}{n_1 \cdots n_K} = \frac{n!}{n_1! * n_2! * \dots * n_k!}$$

Let the total number of examples we've seen be: $n = \sum_k n_k$.

Example

- suppose $\boldsymbol{\theta} = (0.2 \quad 0.1 \quad 0.3 \quad 0.4)^{\mathrm{T}}$
- Suppose $\boldsymbol{n} = (1 \quad 0 \quad 2 \quad 2)^{\mathrm{T}}$
- $n = \sum_k n_k = 1 + 0 + 2 + 2 = 5$

x_i
Spring
Autumn
Winter
Winter
Autumn

k	n_k
Spring	1
Summer	0
Autumn	2
Winter	2

We now want to know the *likelihood* of $\boldsymbol{\theta}$ being the true parameter given $\boldsymbol{n}$, or in other words, how likely the observations $\boldsymbol{n}$ are for a $\boldsymbol{\theta}$ given after n trials:

$$\mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, n) = \binom{5}{1,0,2,2} * 0.2^1 * 0.1^0 * 0.3^2 * 0.4^2$$
$$= 30 * 0.0028 = 0.0864$$

The likelihood $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{n}) = p(\boldsymbol{n}|\boldsymbol{\theta})$ of the parameter vector $\boldsymbol{\theta}$ given on the outcomes $\boldsymbol{n}$ is 8.64%.

4.2.2 MLE

Once we have the likelihood, we can perform MLE for the parameter vector $\boldsymbol{\theta}$. Recall that for MLE, we search $\widehat{\boldsymbol{\theta}}_{MLE} = \underset{\boldsymbol{\theta}}{\mathrm{argmax}}\, \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$, which in the Dirichlet-Multinomial-Model is:

$$\widehat{\boldsymbol{\theta}}_{MLE} = \underset{\boldsymbol{\theta} \in S_k}{\mathrm{argmax}}\, \mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, n) = \underset{\boldsymbol{\theta} \in S_k}{\mathrm{argmax}} \binom{n}{n_1 \cdots n_K} \prod_{k=1}^{K} {\theta_k}^{n_k} = \cdots$$

$$= \underset{\boldsymbol{\theta} \in S_k}{\mathrm{argmax}} \sum_{k=1}^{K} n_k \log \theta_k = \cdots = \frac{\boldsymbol{n}}{n} = \left(\frac{n_1}{n} \quad \dots \quad \frac{n_K}{n}\right)^{\mathrm{T}}$$

Therefore, the MLE estimate for the parameters $\boldsymbol{\theta}$ of a categorical distribution are simply the *relative frequencies* of the categories.

4.2.3 Dirichlet Distribution

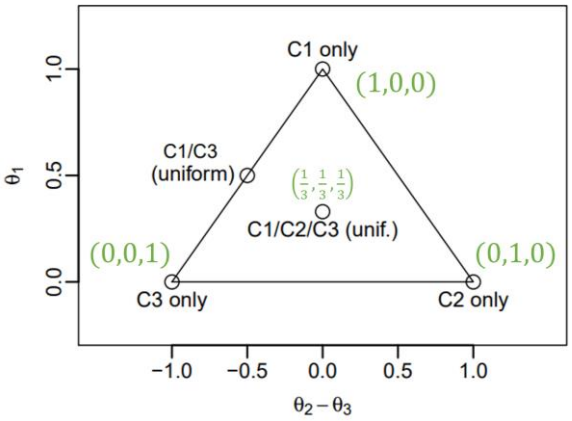
The Dirichlet distribution is parameterized by vector $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ with $\alpha_k > 0$, containing the real and strictly positive *concentration parameters* for each category of the original distribution.

$$\text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} \theta_k{}^{\alpha_k - 1}$$

Drawing a value from the Dirichlet distribution therefore results in a probability distribution $\boldsymbol{\theta}$ over the original categories of the Multinomial Distribution $\mathbf{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, \boldsymbol{n})$.

The Dirichlet Distribution is a distribution over probability vectors $\boldsymbol{\theta} \in S_K$.

Visualization of the case that $K = 3$, where each probability vector is an element of the 3-dimensional probability simplex S_3.



We know that any parameter is in the *probability simplex* with the following properties:

$$S_K = \left\{ \boldsymbol{\theta} \in \mathbb{R}^K, \quad 0 < \theta_k \leq 1, \quad \sum_k \theta_k = 1 \right\}$$

this is simply the set of all vectors of size K, such that they are probability vectors.

Mean

$$E[\theta_k] = \frac{\alpha_k}{\sum_{k'} a_{k'}}$$

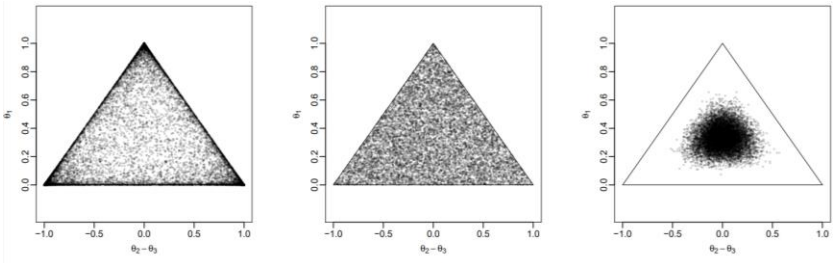
Mode (point of largest density)

$$\theta_k^{mode} = \frac{\alpha_k - 1}{\sum_{k'} a_{k'} - K}$$

unimodal $\boldsymbol{\theta}^{mode}$ for $\boldsymbol{\alpha} > 1$

Special Cases: Symmetric Dirichlet Distribution

- For $\alpha < 1$: multinomials concentrate around single category
- For $\alpha > 1$: multinomials spread uniformly over categories
- For $\alpha = 1$: uniform distribution over multinomials



$\alpha = .3$ $\alpha = 1$ $\alpha = 10$

4.2.4 MAP Estimation

By doing MAP estimation, we can avoid the *zero-count* and *overfitting* problem.

Recall that $\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{D}|\theta)$ as well as $posterior \propto likelihood \times prior$. The *posterior distribution* is:

$$\mathrm{Dir}(n_1 + \alpha_1, \dots, n_K + \alpha_K)$$

Therefore, the MAP estimate for the Dirichlet-Multinomial-Model is:

$$\begin{aligned}
\widehat{\boldsymbol{\theta}}_{MAP} &= \underset{\boldsymbol{\theta} \in S_k}{\operatorname{argmax}}\, \mathrm{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, n) * \mathrm{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) \\
&= \underset{\boldsymbol{\theta} \in S_k}{\operatorname{argmax}}\, \mathrm{Dir}(\boldsymbol{\theta}|n_1 + \alpha_1, \dots, n_K + \alpha_K) \\
&= \underset{\boldsymbol{\theta} \in S_k}{\operatorname{argmax}} \prod_{k=1}^{K} {\theta_k}^{n_k + \alpha_k - 1} \\
&= \frac{(n_1 + \alpha_1 - 1 \quad \cdots \quad n_K + \alpha_K - 1)^T}{n + \alpha_0 - K} \\
&= \frac{\boldsymbol{n} + \boldsymbol{\alpha} - 1}{n + \alpha_0 - K}
\end{aligned}$$

where $\alpha_0 = \sum_{k=1}^{K} \alpha_k$.

Example

Recall the example in *4.2.1 Multinomial Distribution* $\mathbf{Mu}(\boldsymbol{n}|\boldsymbol{\theta}, \boldsymbol{n})$, with a sym. Dirichlet ($\alpha_k = \alpha$):

- for $\alpha = 1$, a uniform prior, we obtain the MLE estimate
- for $\alpha = 2$, we obtain the *add-one smoothing*

1) With $\alpha = 2$, α_0 is $2 + 2 + 2 + 2 = 8$

2) With $\boldsymbol{n} = (1 \quad 0 \quad 2 \quad 2)^T$, n is $1 + 0 + 2 + 2 = 5$

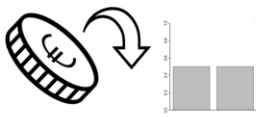
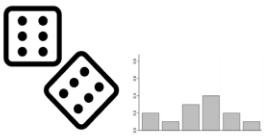
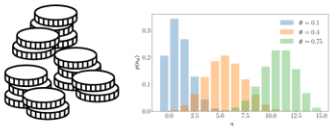
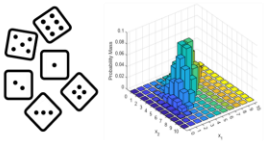
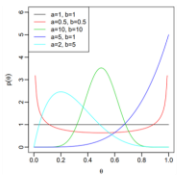
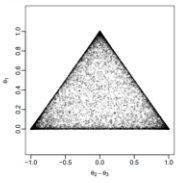
3) The number of categories is $K = 4$

4) Using $\widehat{\boldsymbol{\theta}}_{MAP} = \frac{\boldsymbol{n}+\boldsymbol{\alpha}-1}{n+\alpha_0-K}$, we obtain $\widehat{\boldsymbol{\theta}}_{MAP} = \left(\frac{2}{9} \quad \frac{1}{9} \quad \frac{3}{9} \quad \frac{3}{9}\right)^T$.

This model has a high complexity with $O(C * 2^D)$, with D binary features and C classes. This gets infeasible to even store for moderately large D. To combat this, well look at the Naïve Bayes.

To predict, we could then to plug that parameter $\widehat{\boldsymbol{\theta}}_{MAP}$ back into the model and use $p(y|\boldsymbol{x}, \widehat{\boldsymbol{\theta}}_{MAP})$ to predict the future; this is called a *plug-in approximation*.

4.3 Summary

Beta-Binomial Model		Dirichlet-Multinomial Model
2 possible outcomes  **Bernoulli Distribution** $\text{Ber}(\theta)$		K possible outcomes  **Categorical Distribution** $\text{Cat}(\boldsymbol{\theta})$
over n trials  **Binomial Distribution** $\text{Bin}(n_H \mid n, \theta)$	Used as Likelihood $p(\mathcal{D}\mid\theta)$	over n trials  **Multinomial Distribution** $\text{Mu}(\boldsymbol{n} \mid n, \boldsymbol{\theta})$
over outcome parameter θ  **Beta Distribution** $\text{Beta}(\theta \mid \alpha, \beta)$	Used as Prior $p(\theta)$ & Posterior $p(\theta\mid D)$	over outcome parameters $\boldsymbol{\theta}$  **Dirichlet Distribution** $\text{Dir}(\boldsymbol{\theta} \mid \boldsymbol{\alpha})$

❗ The problem with the techniques discussed so far usually arises with the *class-conditional densities* $p(\boldsymbol{x}|y)$, because the representation needed to store them exactly grows exponentially with the number of attributes. For D binary features, C classes:

$$O(C * 2^D)$$

E.g., for 2 binary features: age & position and 2 classes: poor & rich, we already have $2 * 2^2 = 8$ entries in the probability table.

4.4 Naïve Bayes (NB)

Naïve Bayes Classifiers significantly reduce the number of parameters and data that is stored by making the *naïve bayes assumption:*

$$p(\boldsymbol{x}|y) = \prod_{j=1}^{D} p(x_j|y)$$

that *features* $|\boldsymbol{x}| = D$ *are conditionally independent given the class label* y.

Example 1

Given the features $\boldsymbol{x} = (age, height)^{\mathrm{T}}$, and the labels $y = \{good\ vocab, bad\ vocab\}$. The Naïve Bayes Assumption states, that if we know somebody has a good/bad vocabulary, then knowing their age does not tell us anything about their height.

Say if $height \perp age \mid vocab$ holds, then $p(height|age, vocab) = P(height|vocab)$

This is quite a strong assumption that often does not hold in practice, but nonetheless the classifier may still achieve good results.

> Naïve Bayes is not *one* classifier but rather a family of classifiers. Anyone that makes the same, naïve assumption is part of it. They still differ in how they model class-conditional densities, whether or not they use priors, how they are trained and how they predict.

For D features with K possible values and C toal classes Naïve Bayes classifiers reduce the number of parameters for the *class-conditional densities* $p(\boldsymbol{x}|y)$ to:

$$O(CDK)$$

Example 2

Before

$p(\boldsymbol{x}|y = poor)$

x_j	young	old
student	1/2	1/2
CEO	0	0

$p(\boldsymbol{x}|y = rich)$

x_j	young	old
student	0	0
CEO	0	1

$O(CD^K) = 8$

After with NB-Assumption

x_j	$\hat{\boldsymbol{\theta}}_{poor,\,Age}$	x_j	$\hat{\boldsymbol{\theta}}_{poor,\,Pos}$
young	1/2	stud.	1
old	1/2	CEO	0

x_j	$\hat{\boldsymbol{\theta}}_{rich,\,Age}$	x_j	$\hat{\boldsymbol{\theta}}_{rich,\,Pos}$
young	0	stud.	0
old	1	CEO	1

$O(CDK) = 8$

This scales way better with increasing complexity.

4.4.1 NB & MLE for Categorical Data

The parameters we want to estimate are $\boldsymbol{\pi}$ and $\boldsymbol{\theta}_{cj}$:

- For the *prior class distributions* $p(y_i|\boldsymbol{\pi}) = \text{Cat}(y_i|\boldsymbol{\pi})$. These simply are the probabilities $\boldsymbol{\pi} = (\pi_1 \dots \pi_C)^T \in S_C$ (probability vector) of each class y_i.
- For the *class-conditional densities* $p(x_j|y = c, \boldsymbol{\theta}_{cj}) = \text{Cat}(x_j|\boldsymbol{\theta}_{cj})$. These are the probabilities $\boldsymbol{\theta}_{cj} \in S_K$ over the K (amount) possible input combinations of the features j and their classes c.

❗ Below, we sometimes denote both parameters $\boldsymbol{\pi}$ and $\boldsymbol{\theta}_{cj}$ as $\boldsymbol{\theta}$.

Single Sample

By applying the *naïve bayes assumption*, we obtain the likelihood by the product rule:

$$p(x, y|\theta) = p(x_i|y, \theta) * p(y|\theta)$$

Applied to our example:

$$p(\boldsymbol{x}_i, y_i|\boldsymbol{\theta}) = \text{Cat}(y_i|\boldsymbol{\pi}) \prod_{j=1}^{D} \text{Cat}(x_{ij}|y_i, \boldsymbol{\theta}_{y_i j})$$

Training Set

Using the iid. assumption, the likelihood of a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$ is given by the product over the likelihoods of each individual example in the training set:

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \prod_i p(\boldsymbol{x}_i, y_i|\boldsymbol{\theta}) = \prod_i \text{Cat}(y_i|\boldsymbol{\pi}) \prod_j^D \text{Cat}(x_{ij}|y_i, \boldsymbol{\theta}_{y_i j})$$

where D denotes the number of features.

Log Likelihood

When performing MLE, we often alternatively maximize the *log-likelihood*. This works due to the monotonic property of the log function.

$$\ell(\mathcal{D}|\boldsymbol{\theta}) \stackrel{\text{def}}{=} \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_{c=1}^{C} \sum_{i:\, y_i = c} \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:\, y_i = c} \log \left[\boldsymbol{\theta}_{cj}\right]_{x_{ij}}$$

This results in two parts, where the first one only accesses the prior class distribution and the later one only the class-conditional densities. These two parts then can be maximized separately. As π_c does not change for within the same class c, we can further simplify this sum:

$$\sum_{i:\, y_i = c} \log \pi_c = n_c * \log \pi_c$$

where n_c is the number of examples from class c. Similarly, we can simplify the right-hand side, where $n_{cjk} = \left|\{i: y_i = c, x_j = k\}\right|$ is the number of examples of class c for which feature j takes value k to:

$$\ell(\mathcal{D}|\boldsymbol{\theta}) \stackrel{\text{def}}{=} \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_{c=1}^{C} n_c * \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{k=1}^{K} n_{cjk} * \log \left[\boldsymbol{\theta}_{cj}\right]_k$$

This results in:

$$\hat{\pi}_c = \frac{n_c}{n} \qquad \left[\hat{\boldsymbol{\theta}}_{cj}\right]_k = \frac{n_{cjk}}{n_c}$$

Example

Data $\mathcal{D}$

Age	*Position*	*State*
young	student	poor
old	student	poor
old	CEO	rich

Prior Class Distribution

c	$\hat{\pi}_c$
poor	2/3
rich	1/2

Class Conditional Densities

	$j = Age$		$j = Position$	
$c = poor$	k	$\hat{\boldsymbol{\theta}}_{c,j}$	k	$\hat{\boldsymbol{\theta}}_{c,j}$
	young	1/2	stud.	2/2
	old	1/2	CEO	0
$c = rich$	k	$\hat{\boldsymbol{\theta}}_{c,j}$	k	$\hat{\boldsymbol{\theta}}_{c,j}$
	young	0	stud.	0
	old	1/1	CEO	1/1

$$\ell(\mathcal{D}|\boldsymbol{\theta}) = \sum_{c=1}^{C} n_c * \log \pi_c + \sum_{j=1}^{D}\sum_{c=1}^{C}\sum_{k=1}^{K} n_{cjk} * \log\left[\boldsymbol{\theta}_{cj}\right]_k$$

$$= \left(2 * \log\frac{1}{2} + 1 * \log 1\right) + \left(\begin{array}{l} 1 * \log\frac{1}{2} + 1 * \log\frac{1}{2} + 0 * \log 0 + 1 * \log 1 + \\ 2 * \log\frac{2}{2} + 0 * \log 0 + 0 * \log 0 + 1 * \log 1 \end{array}\right)$$

$$= -4 * \log 2 \;\Leftrightarrow\; \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \frac{1}{16}$$

Above estimations $\hat{\pi}_c$ and $\left[\hat{\boldsymbol{\theta}}_{cj}\right]_k$ maximize the likelihood given data set $\mathcal{D}$.

4.4.2 Fully Bayesian Naïve Bayes

For the posterior predictive we need to marginalize out all parameters. In our case:

$$p(y, \boldsymbol{x}|\mathcal{D}) = \int \mathrm{Cat}(y|\boldsymbol{\pi}) p(\boldsymbol{\pi}|\mathcal{D})\, d\boldsymbol{\pi} * \prod_{j=1}^{D}\left(\int \mathrm{Cat}\left(x_j|y, \boldsymbol{\theta}_{yj}\right) p\left(\boldsymbol{\theta}_{yj}|\mathcal{D}\right) d\boldsymbol{\theta}_{yj}\right)$$

When using a Dirichlet prior, we obtain *posterior means*:

$$\bar{\pi}_c = \frac{n_c + \alpha_c}{n + \sum_c \alpha_c} \qquad \left[\bar{\theta}_{cj}\right]_k = \frac{n_{cjk} + \alpha_{cjk}}{n_c + \sum_k \alpha_{cjk}}$$

and posterior predictive for a new example is:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) \propto \bar{\pi}_c \prod_{j=1}^{D}\left[\bar{\theta}_{cj}\right]_{x_j}$$

4.4.3 Discussion

- Hyperparameters can be estimated as part of model selection
- Add-one smoothing is also commonly used
  - Can be interpreted as MAP estimate with Dirichlet prior, $\alpha = 2$
  - Can be interpreted as fully Bayesian inference with uniform prior
- Simple model, cheap to compute, reduced overfitting, no zero-count problem
- But beware: Naive Bayes assumption is a strong assumption

5 Point Estimation / Optimization Methods

Recall that the *Maximum Likelihood Estimation* (MLE) chooses the value $\widehat{\boldsymbol{\theta}}$ such that it maximizes the likelihood of the data $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$:

$$\widehat{\boldsymbol{\theta}}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \begin{cases} p(\mathcal{D}|\boldsymbol{\theta}) & \text{for generative models} \\ p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) & \text{for discriminative models} \end{cases}$$

The larger the likelihood, the more consistent is the data with the parameter choice $\boldsymbol{\theta}$.

Parameter estimation often requires solving a nonlinear optimization problem of form:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, f(\boldsymbol{\theta})$$

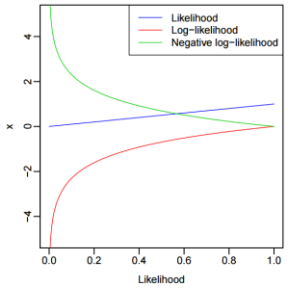
- For MLE, f is the neg. log-likelihood (NLL) of the training data: $-\ell(\boldsymbol{\theta}|\mathcal{D})$
- For ERM, f is the empirical risk: $R_{emp}(\boldsymbol{\theta})$
- For MAP, f is the neg. log posterior: $-\log p(\boldsymbol{\theta}|\mathcal{D})$
- For RRM, f is the regularized risk: $R'_{emp}(\boldsymbol{\theta})$

5.1 Log-Likelihood

💡 Now instead of maximizing the likelihood we can also:

- *maximize* the log-likelihood $\ell(\boldsymbol{\theta}|\mathcal{D}) \overset{\text{def}}{=} \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$
- *minimize* the negative log-likelihood $-\ell(\boldsymbol{\theta}|\mathcal{D})$

We can do this because the logarithm is a *monotonic transformation*. It is useful to avoid numerical instability due to underflow.



If training examples are iid. then the likelihood of an entire training set factors into the individual examples:

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$$

For the log-likelihood this product simplifies to a sum of logs per log-rules:

$$\ell(\boldsymbol{\theta}|\mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$$

where $p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$ is the classifier's probability for the ground truth associated with training example $\boldsymbol{x}_i$. This doesn't necessarily indicate the predicted (highest prob.) class by our classifier.

💡 This "summation form" makes it easier to optimize for gradient-based parameter estimation.

5.2 Loss Functions for ERM

❓ Is there any relationship between MLE and Empirical Risk $\boldsymbol{R_{emp}(h)}$? Recall that the ERM chooses the estimator that minimizes $R_{emp}(h)$:

$$\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\; R_{emp}(h) \qquad R_{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} L(h(\boldsymbol{x_i}), y_i)$$

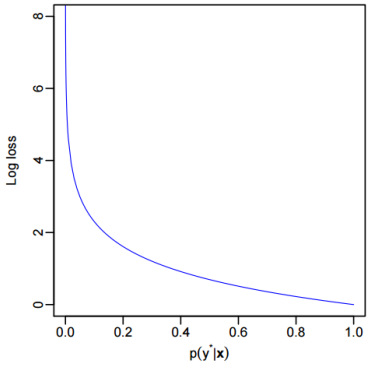
We will show for discriminative models, MLE and ERM with certain choices of loss functions (log loss, cross entropy loss, KL divergence loss, …) produce equivalent classifiers.

5.2.1 Log Loss

For probabilistic classifiers, the *log loss* is given by:

$$L_{log}(p(y|\boldsymbol{x}), y^*) = -\log p(y^*|\boldsymbol{x})$$

where y^* is the true label and $p(y|\boldsymbol{x})$ the probability distribution that the classifier outputs.



For $p(y^*|\boldsymbol{x}) = 1$ the log loss then is 0.

The empirical risk with log loss then is proportional to the neg. log-likelihood (cf. 5.1):

$$R_{emp}(\boldsymbol{\theta}) \propto -\ell(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})$$

For discriminative classifiers that perform ERM with log loss, an MLE also minimizes the empirical risk, and vice versa.

5.2.2 Entropy

$H(p)$ is called the Shannon Entropy of distribution p. It is a measure of uncertainty.

$$H(p) = \sum_x p(x) \log \frac{1}{p(x)}$$

$$= \mathrm{E}\left[\log \frac{1}{p(x)}\right] = \mathrm{E}[-\log p(x)]$$

where $p(x)$ is the probability for which x was sampled from p.

For a distribution that assigns $p(x) = 1$ for a particular value. The entropy is 0 as there is no uncertainty which value was sampled. The maximum value for $H(p) = \log K$, achieved by a uniform distribution over K classes.

5.2.3 Cross Entropy Loss

The cross entropy of *true* distribution q relative to *wrong* distribution p is:

$$H(p, q) = \sum_x p(x) \log \frac{1}{q(x)} = \mathrm{E}_{x\sim p}[-\log p(x)]$$

it quantifies the difference between p and q.

To use this as a loss function, we can compare the model distribution $q(y)$ of some classifier q, to the empirical distribution $\tilde{p}(y)$ that we as deem "correct" or "desired" distribution for our training data. The empirical distribution $\tilde{p}(y)$ therefore, serves as a reference when evaluating the performance of our model.

$$L_{CE}(q(y), y^*) = H(\tilde{p}, q)$$

5.2.4 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence, often denoted as $D_{KL}(p \parallel q)$, is a measure of the difference between two probability distributions, p and q. It quantifies how one distribution, q, diverges from another, p. The KL divergence is defined as:

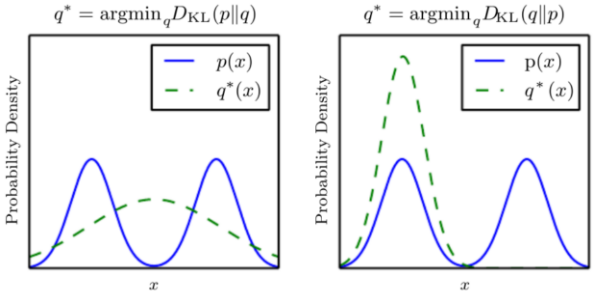
$$\begin{aligned} D_{KL}(p \parallel q) &= H(p,q) - H(p) \\ &= \sum_x p(x) \log \frac{p(x)}{q(x)} \\ &= \mathrm{E}_{x\sim p}[\log p(x) - \log q(x)] \end{aligned}$$

In contrast to the cross-entropy $H(p,q)$, the KL divergence measures the information lost when using distribution q to approximate distribution p.

It's not symmetric, meaning $D_{KL}(p \parallel q)$ is not the same as $D_{KL}(q \parallel p)$.

Consider the true distribution p that should be approximated by a normal distribution q.

The green line is the distribution that minimizes the respective KL divergence.



In the right picture, for $D_{KL}(q \parallel p)$, whenever q has high probability then p needs to have high probability, for a low KL divergence.

In the left picture, for $D_{KL}(p \parallel q)$, whenever p has high probability then q needs to have high probability, for a low KL divergence. This asymmetry leads to two different distributions

Similarly, to CE-Loss, we can also use KL divergence, since:

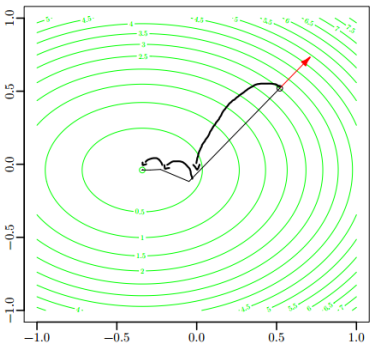
$$\operatorname*{argmin}_q H(\tilde{p}, q) = \operatorname*{argmin}_q [H(\tilde{p}, q) - H(\tilde{p})] = \operatorname*{argmin}_q D_{KL}(\tilde{p} \parallel q)$$

Meaning that a model q that minimizes the CE wrt. $\tilde{p}$, then that model also minimizes the KL divergence wrt. $\tilde{p}$, as the term $-H(\tilde{p})$ does not depend on q.

> Above concepts also generalize to other, non-categorical distributions.

5.3 Gradient Descent

Gradient-based methods aim to minimize an objective function, such as the negative log likelihood. They iteratively update the current estimate $\boldsymbol{x}_n$ until some stopping criterion is met. They use gradient information for this; also, the function that should be optimized needs to be differentiable.



To find the minimum $\boldsymbol{x}^*$ of a function f:

Pick starting point $\boldsymbol{x}_0$, repeat for t epochs:

1. Compute gradient $\nabla f(\boldsymbol{x}_0)$
2. Jump downhill to $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \epsilon_n \nabla f(\boldsymbol{x}_n)$

The direction of the negative gradient tells us how to move downhill and the norm how far to move. The learning rate ϵ_n further controls how far we move.

The gradient ∇ of the loss function L is a (row) vector:

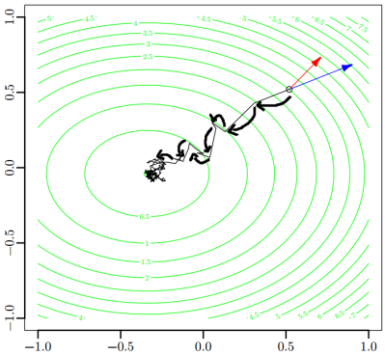
$$\nabla_{\mathbf{w}^\top} L \stackrel{\text{def}}{=} \left(\frac{\partial}{\partial w_1} L \quad \frac{\partial}{\partial w_2} L \quad \dots \quad \frac{\partial}{\partial w_j} L \right)$$

with each component representing the *partial derivative* of the *loss function* wrt. a certain *feature weight.* With j ranging from 1 to D, corresponding to the D number of features.

A single gradient computation and subsequent parameter update is called an *epoch*.

5.4 Stochastic Gradient Descent (SGD)

In SGD, we're approximating the gradient for N examples by using a *single, uniformly sampled training example* $Z \in \{1,2,\dots,N\}$, instead of all N training examples as in plain GD.



Repeat N times:

1. Pick random example z (w/wo replacement)
2. Approximate gradient: $\widehat{\nabla} f(\boldsymbol{x}_0)$
3. Jump approx. downhill: $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \epsilon_n \widehat{\nabla} f(\boldsymbol{x}_n)$

This finishes a **single**, SGD epoch.

The single gradient can be computed much quicker than computing an entire gradient over N examples. Furthermore, as we pick examples randomly, this helps escape local minima.

5.5 Comparison & Best Practices

💡 Empirically, for large datasets, SGD usually has faster convergence to vicinity of optimum.

Per epoch, assuming $O(D)$ gradient computation per example

	GD	SGD
Algorithm	Deterministic	Randomized
Gradient computations	1	N
Gradient types	Exact	Approximate
Parameter updates	1	N
Time	$O(DN)$	$O(DN)$
Space	$O(D)$	$O(D)$

❗ SGD can exploit repeated structure in training data. If some examples are equivalent or almost identical, then the approximated gradient becomes very close to the true gradient. Thus, in one epoch this allows us to move *N* times more steps with little deterioration in quality.

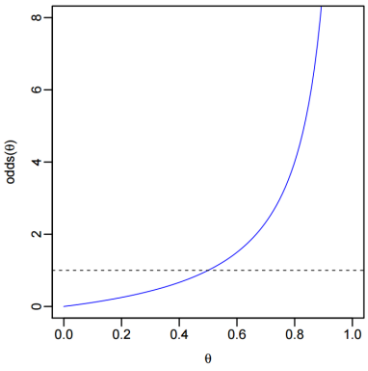
Also, step size (or learning rate) sequence $\{\epsilon_n\}$ needs to be chosen carefully; widely studied:

- Constant step size throughout
  - Large → good initially (move quickly), bad later on (juggle around optimum)
  - Small → slow convergence but will eventually stabilize around optimum
- *Bold driver heuristic*
  - Increase ϵ_n slightly when objective decreased (e.g., by 5%)
  - Decrease ϵ_n sharply when objective increased (e.g., by 50%)
- Line search

6 Classifiers for Continuous Data

6.1 Introduction / Basics

6.1.1 Odds

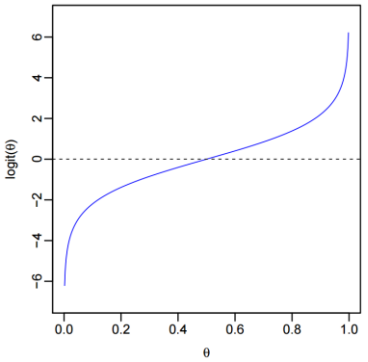


For the success probability θ of a Bernoulli Distribution $\mathbf{Ber}(\boldsymbol{\theta})$ $\mathrm{Ber}(\theta)$, we can define the *odds*:

$$\mathrm{odds}(\theta) = \frac{\theta}{1-\theta}$$

E.g., for $\theta = 0.1$ the odds are $\frac{0.1}{1-0.1} = \frac{1}{9}$, also written as $1:9$.

6.1.2 Logit Function



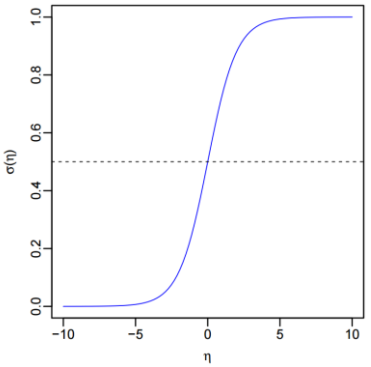
The *logit function* or *log odds* is defined as the logarithm of the odds function.

$$\mathrm{logit}(\theta) \overset{\text{def}}{=} \log \mathrm{odds}(\theta)$$

> Per logarithmic rules, this can be rewritten as:
>
> $$\mathrm{logit}(\theta) = \log \mathrm{odds}(\theta) = \log \theta - \log(1-\theta)$$

Now, given a $\mathrm{logit}(\theta) = \eta$, what is the corresponding success probability θ?

6.1.3 Logistic Sigmoid Function



The inverse of the logit function is called the *logistic function* $\sigma(\eta)$:

$$\sigma(\eta) \overset{\text{def}}{=} \frac{1}{1+\exp(-\eta)}$$

Therefore, $\sigma(\eta) = \sigma\big(\mathrm{logit}(\theta)\big) = \theta$.

Similar to the *tanh-function* for range $[-1,1]$, the logistic function can be used to squash an arbitrary large real value into the range of $[0,1]$.

> **Important Properties**
>
> $\sigma(\eta) = \frac{1}{1+\exp(-\eta)} = \frac{\exp(\eta)}{1+\exp(\eta)}$
>
> $\frac{\partial}{\partial \eta}\sigma(\eta) = \sigma(\eta) * \big(1-\sigma(\eta)\big)$
>
> $\sigma(-\eta) = 1-\sigma(\eta)$
>
> $\sigma^{-1}(\theta) = \log\left(\frac{\theta}{1-\theta}\right) = \mathrm{logit}(\theta)$

6.2 Logistic Regression (LR) Model

Logistic Regression is a *discriminative, binary classifier*. Meaning that it models $p(y|\boldsymbol{x})$, the variable $y \in \{0,1\}$ is binary and the input is real-valued $\boldsymbol{x} \in \mathbb{R}^D$.

Logistic regression assumes that you can model the log odds (logit function) of the binary outcome y as a linear function of the independent variables $\boldsymbol{x}$ (features). This allows us to make predictions about the probability of the event occurring $p(y = 1|\boldsymbol{x})$ based on the values of those independent variables. With every linear function being describable by:

$$\eta = w_0 + w_1 x_1 + \cdots + w_D x_D = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$$

> In the context of logistic regression, the *linear predictor*, denoted as η, represents the linear combination of the features that is used to calculate the log odds of the binary outcome.

This results in:

$$p(y = 1|\boldsymbol{x}, \{w\}_j) = \sigma\left(w_0 + \sum_j w_j x_j\right) = \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$$

$$p(y = 0|\boldsymbol{x}, \{w\}_j) = 1 - \sigma\left(w_0 + \sum_j w_j x_j\right) = 1 - \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$$

Notation/Implementation

We collect the *parameters* and *bias* into a *weight vector*:

$$\boldsymbol{w} = (w_0 \quad w_1 \quad \dots \quad w_D)^{\mathrm{T}}$$

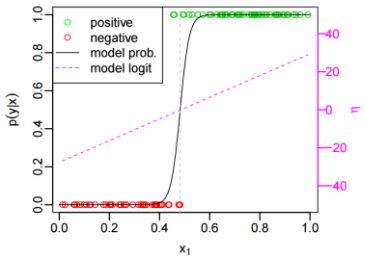
The features are collected in the respective feature vector *with implicit bias*:

$$\boldsymbol{x} = (1 \quad x_1 \quad \dots \quad x_D)^{\mathrm{T}}$$

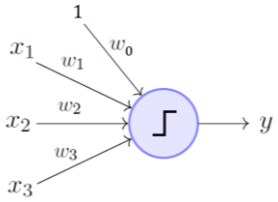
This allows us to calculate the *linear predictor* η by inner/dot product: $\eta = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$ and obtain:

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathrm{Ber}(y|\sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle))$$

To the right is a 1D-LR-example for the singe feature x_1. The decision boundary is at around 0.5.



6.2.1 Perceptron



A related ML model is called the *perceptron.* It is a *discriminative function* with a *step-function* decision rule:

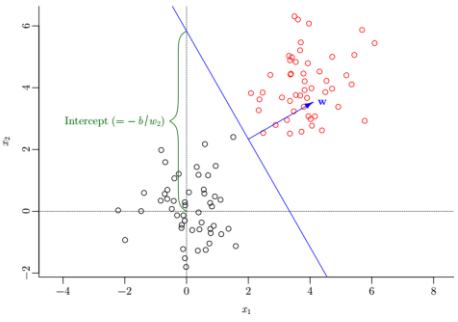
$$\hat{y} = \begin{cases} 0 & \langle \boldsymbol{w}, \boldsymbol{x} \rangle < 0 \\ 1 & \langle \boldsymbol{w}, \boldsymbol{x} \rangle > 0 \end{cases}$$

If we use LR and only predict the most likely class, we obtain the same decision rule. For both LR and the perceptron, the decision boundary is $\{\boldsymbol{x}: \langle \boldsymbol{w}, \boldsymbol{x} \rangle = 0\}$. Meaning that the log odds is 0, hence the success probability is 0.5.

> The *decision boundary* of a classifier is the set of data points for which the classifier is unsure what to predict. Here, multiple classes achieve the same, highest probability or score.
>
> I.e., a data point $\boldsymbol{x}$ is on the decision boundary, if there exist two different classes $c_1 \neq c_2$, s.t.: $$p(y = c_1|x) = p(y = c_2|x) = \max_c p(y = c|\boldsymbol{x})$$
>
> For which the classifier gives the same probability, and no other class has a higher one.

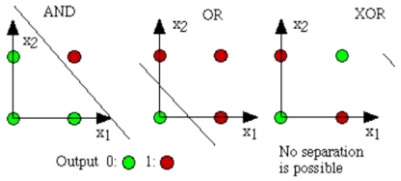
6.2.2 Decision Boundary

Using the geometric interpretation of the dot product $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$, this means, that the angle between the two vectors is 90°.



The decision boundary for logistic regression, and other linear classifiers, is the *set of points that are orthogonal to the weight vector* $\boldsymbol{w}$, forming a hyperplane with normal vector $\boldsymbol{w}$.

Only for LR – not perceptron, the *magnitude* of the weight vector tells how certain/confident it is near the decision boundary. So, for a large weight vector, the area of uncertainty is very small.



This hyperplane-decision-boundary in turn means that those classifiers are not able to learn non-linear relationships, such as the XOR function for example.

6.2.3 Gradient & Derivatives of LR

Intuition

For logistic regression with *negative log-likelihood loss*, we update a single feature weight w_j by the $\frac{\partial}{\partial w_j} L$ entry in the gradient, where $L = -\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$:

$$\frac{\partial}{\partial w_j} - \ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = \sum_i^N -x_{ij} \left[y_i - p_{i1} \right]$$

Above equation is the component of the gradient wrt. to a single feature weight w_j. It is the result of summing over all training examples i and multiplying the value of the j-th feature $-x_{ij}$ by the error term $[y_i - p_{i1}]$. In the error term, p_{i1} is the predicted probability that the i-th example belongs to class 1 and y_i the true class label; recall that $p_{i1} = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)$

The entire term represents how a small change in the feature weight w_j would affect the loss function $-\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$. If the derivative is positive, then increasing w_j would also increase $-\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$, if the derivative is negative, this relationship is inverted.

Note that the gradient is the accumulation of these multiplicative contributions across all training examples, which guides the optimization process to update the weights for better model performance.

Vectorized Notation & Matrix Calculus

To further simplify, we denote the error on a single example as $e_i = y_i - p_{i1}$ and store all errors in the error vector $\boldsymbol{e} = (e_1 \quad e_2 \quad \dots \quad e_N)^\top$. Then above equation simplifies to:

$$\frac{\partial}{\partial w_j} - \ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = -\sum_i^N e_i x_{ij} = -\boldsymbol{e}^\top \boldsymbol{x}_j$$

where $-\boldsymbol{e}^\top \boldsymbol{x}_j$ is the dot product operation. Furthermore, the *entire NLL gradient* becomes:

$$\nabla_{\boldsymbol{w}^\top} - \ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = -\boldsymbol{e}^\top \boldsymbol{X}$$

where $-\boldsymbol{e}^\top \boldsymbol{X}$ represents the matrix-vector product between the transpose of the *error vector* $\boldsymbol{e}^\top$ and the design matrix $\boldsymbol{X}$. This operation results in a vector, where each component of the vector corresponds to the gradient of the loss function with respect to a specific weight parameter.

The *design matrix* $\boldsymbol{X}$ has rows representing examples (or data points) and columns representing their corresponding features.

6.2.4 Stochastic Gradient of LR

The *approximated NLL gradient* for LR is:

$$\hat{\nabla}_{\boldsymbol{w}^\top} - \ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = -N e_Z \boldsymbol{x}_Z^\top$$

where $Z \in \{1,2, \dots, N\}$ is a *single training example* chosen *uniformly* and at *random* from the N examples in the training set.

6.2.5 Discussion

- LR can be extended to handle non-linear decision boundaries → *kernel LR*
- Under mild conditions, LR can handle imbalanced classes in the data to some extent.
- Logistic regression with NLL loss and decreasing step-size ϵ_n converges to the MLE estimator $\hat{\boldsymbol{w}}_{MLE}$. This is because NLL of LR is convex (cf. *1.7*)

6.2.6 Linearly Dependent Features

❗ If the features of LR are *linearly dependent*, the MLE for the weights are underdetermined. In other words, you have insufficient data to uniquely determine the values of the model parameters (weights).

- The logistic regression model becomes unstable, and small changes in the data can lead to significant changes in the estimated weights.
- You might have weight configurations that fit the data equally well
- The model may overfit the training data

Example

x_1	x_2	y
1	1	0
1	1	1

Features are linearly dependent

$$\boldsymbol{w}_1 = \begin{pmatrix} 1 & -1 \end{pmatrix}^\top$$
$$\boldsymbol{w}_2 = \begin{pmatrix} 0 & 0 \end{pmatrix}^\top$$
$$\boldsymbol{w}_3 = \begin{pmatrix} -1 & 1 \end{pmatrix}^\top$$

Possible MLE estimates

$$\boldsymbol{x}_{\text{new}} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$$
$$\sigma(\boldsymbol{w}_1^\top \boldsymbol{x}_{\text{new}}) \approx 0.73$$
$$\sigma(\boldsymbol{w}_2^\top \boldsymbol{x}_{\text{new}}) = 0.5$$
$$\sigma(\boldsymbol{w}_3^\top \boldsymbol{x}_{\text{new}}) \approx 0.27$$

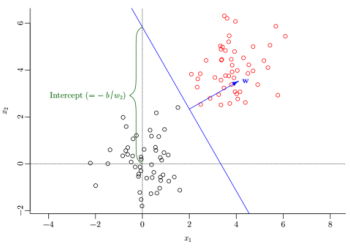
They give different estimates for new data

In general, MLE cannot be used in such cases. Feature selection, or priors/regularization need to be used.

6.2.7 Linearly Separable Examples

❗ When examples are *linearly separable*, which technically is really good for the classifier, training via MLE won't work.

Example



Say we have linearly separable data. Recall that for LR, the *magnitude* of the weight vector $\|\boldsymbol{w}\|$ tells how certain/confident it is near the decision boundary.

If we have found weights $\boldsymbol{w}$ that perfectly separate all training examples, there will *always* be another weight vector $\boldsymbol{w}'$ that assigns higher certainty to its predictions, simply by scaling its features $\boldsymbol{w}' = c * \boldsymbol{w}$ by a constant $c > 1$, increasing the magnitude and corresponding training-data likelihood.

We can repeat this process ad infinitum and end up with weights that are arbitrarily large and an ever-increasing likelihood for the training data. These pose problems related to model stability and overfitting.

6.3 MAP Estimation

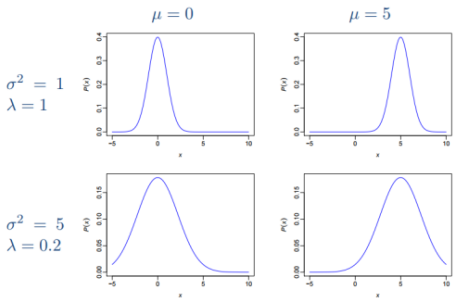
To mitigate the problems discussed above, we can make use of a prior. Recall normal/gaussian distributions (cf. 1.3.1).

We can express the variance also as precision with:

$$\lambda = \frac{1}{\sigma^2}$$

Meaning that a precision $\lambda = 0.2$ corresponds to $\sigma^2 = 5$.
Or, more generally:

- high precision $\lambda \rightarrow$ low spread
- low precision $\lambda \rightarrow$ high spread



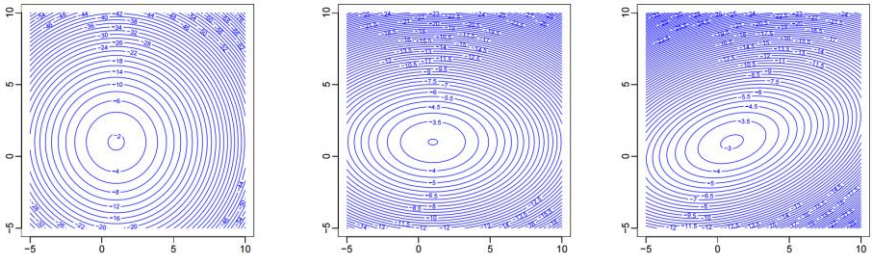
6.3.1 Multivariate Gaussian Distribution

The multivariate gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is parametrized by mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$ (or precision $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$).

Its PDF is defined if $\boldsymbol{\Sigma}$ is positive definite. This is the generalization that the variance needs to be positive. Let $|\boldsymbol{\Sigma}|$ be the determinant of $\boldsymbol{\Sigma}$:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right]$$

where $(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$ is the squared *Mahalanobis distance*. It's a single quantity that measures how far $\boldsymbol{x}$ is away from $\boldsymbol{\mu}$. The density of point decreases exponentially in this distance.



$\boldsymbol{\mu} = \begin{pmatrix}1\\1\end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix} = \sigma^2 \boldsymbol{I}$ (spherical)

$\boldsymbol{\mu} = \begin{pmatrix}1\\1\end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix}5 & 0\\0 & 2\end{pmatrix}$ (diagonal)

$\boldsymbol{\mu} = \begin{pmatrix}1\\1\end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix}5 & 1\\1 & 2\end{pmatrix}$

6.3.2 Spherical Gaussian Prior

The spherical gaussian prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ states that the weight vector $\boldsymbol{w}$ is normally distributed with $\mu = 0$ and some variance $\sigma^2 > 1$. Where $\boldsymbol{I}$ is the *Identity Matrix* and has 1's along main diagonal apart from 0's elsewhere.

If the is the multivariate gaussian is spherical or diagonal, which is the case for $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, then the individual components are independent and follow $w_j \sim \mathcal{N}(0, \sigma^2)$.

Intuitively, this prior keeps each weight close to zero unless data suggest otherwise.

6.3.3 MAP Estimate

The point estimate that maximizes the posterior wrt. LR likelihood and spherical gaussian prior:

$$\widehat{\boldsymbol{w}}_{MAP} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})\, \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \sigma^2 \boldsymbol{I})$$

which is the most likely weight given data and prior. Taking logs, we obtain:

$$\begin{aligned}\widehat{\boldsymbol{w}}_{MAP} &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \left[\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) + \log \prod_j \mathcal{N}\left(w_j|0, \sigma^2\right) \right] \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \left[\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) + \log \sum_j \frac{1}{2\sigma^2} w_j^2 \right] \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \left[\ell(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) - \frac{\lambda}{2} \|w\|^2 \right]\end{aligned}$$

💡 The likelihood is decreased more, the bigger the magnitude of the weight vector $\|w\|$ is.

For $\lambda = 0$ we obtain the MLE.

With MLE, shifting or changing the scale of parameters does not affect the predicted probabilities. E.g., when we scale a parameter by factor 10 and its weight by factor 1/10, we obtain the same predictions,

❗ With MAP, this does not hold; rescaled weight vector has higher prior density and thus a lower penalty resulting in a changed MAP objective.

6.3.4 Regularized Risk Minimization

Intuition

Recall that for iid. data, MLE estimates for classifiers are related to as empirical risk minimization with log loss (cf. Loss Functions for ERM).

$$\widehat{\boldsymbol{w}}_{MLE} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=1}^{N} -\log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) \right]$$

📄 Likewise, MAP estimation with a spherical gaussian prior is related to *Regularized Risk Minimization* with ℓ_2 penalty function.

$$\widehat{\boldsymbol{w}}_{MAP} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=1}^{N} -\log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) + \frac{\lambda'}{2} \|\boldsymbol{\theta}\|^2 \right]$$

For $\lambda' = \frac{1}{N\sigma^2}$ MAP estimation with ℓ_2 *regularization* produces the same results as regularized risk minimization.

Effect on Gradient Descent

ℓ_2 regularization leads to *plain GD* learning rules with *weight decay*:

$$\begin{aligned}\boldsymbol{\theta}_{n+1} &\leftarrow \boldsymbol{\theta}_n - \epsilon_n \nabla R_{emp}(\boldsymbol{\theta}_n) - \epsilon_n \lambda' \boldsymbol{\theta}_n \\ &= (1 - \epsilon_n \lambda')\boldsymbol{\theta}_n - \epsilon_n \nabla R_{emp}(\boldsymbol{\theta}_n)\end{aligned}$$

Likewise, for *SGD*, with loss L and example z, we obtain:

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \epsilon_n \nabla L(p(Y|\boldsymbol{x}_z, \boldsymbol{\theta}_n), y_z) - \epsilon_n \lambda' \boldsymbol{\theta}_n$$

E.g., for penalized logistic regression: $\boldsymbol{w}_{n+1} \leftarrow \boldsymbol{w}_n - \epsilon_n \mathrm{e}_z \boldsymbol{x}_z - \epsilon_n \lambda' \boldsymbol{w}_n$

6.3.5 Further Notes

- ℓ_2 regularization / weight decay is generally applicable (model does not need to be probabilistic). Whenever it is used, we perform weight decay in the learning rule.
- For iid data, MAP estimation with diagonal (*not spherical*) Gaussian prior can be expressed as regularized risk minimization with *weighted* ℓ_2 regularization (has different weight for each parameter).
- Using an arbitrary gaussian prior as RRM results in *Tikhonov regularization* (penalty is $\|\boldsymbol{\Lambda}'\boldsymbol{\theta}\|^2$)

6.4 SoftMax Regression

SoftMax Regression (or *Multinomial Logistic Regression*) is an extension of binary LR to C classes. Instead of using a single linear predictor η, we use C linear predictors (one per class):

$$\eta_1 = \langle \boldsymbol{w}_1, \boldsymbol{x} \rangle \quad \dots \quad \eta_C = \langle \boldsymbol{w}_C, \boldsymbol{x} \rangle$$

Further, instead of using the logistic function $\sigma(\eta)$, we now use the SoftMax function. Let $\boldsymbol{W} = (\boldsymbol{w}_1\ \boldsymbol{w}_2 \dots \boldsymbol{w}_C)$. For C classes, SoftMax regression uses the model:

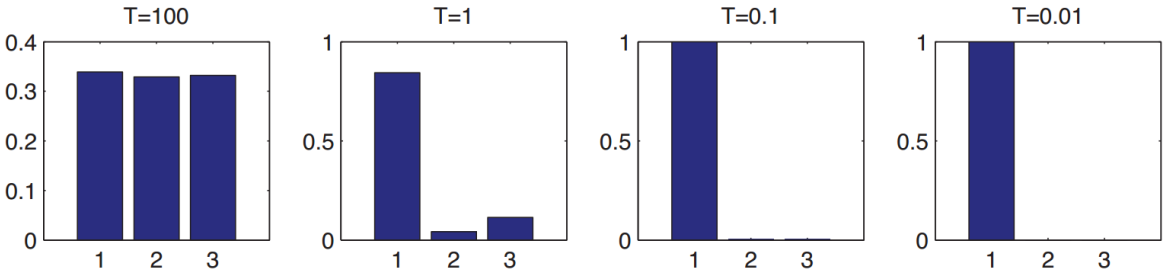
$$p(y = c|\boldsymbol{x}, \boldsymbol{W}) = S(\langle \boldsymbol{w}_1, \boldsymbol{x} \rangle, \dots, \langle \boldsymbol{w}_C, \boldsymbol{x} \rangle)_c = S\left(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{x}\right)_c$$

6.4.1 SoftMax Function

The SoftMax function takes as input a vector $\boldsymbol{\eta} = (\eta_1, \dots, \eta_C)^{\mathrm{T}} \in \mathbb{R}^C$ consisting of C real numbers and transforms these real values into a *probability vector* $S(\boldsymbol{\eta})$.

$$S(\boldsymbol{\eta})_c = \frac{\exp(\eta_c)}{\sum_{c'=1}^{C} \exp(\eta_{c'})}$$

It exaggerates differences in the input vector. Below is the result for $\boldsymbol{\eta} = (3, 0, 1)^{\mathrm{T}}$ with $S\left(\frac{\boldsymbol{\eta}}{T}\right)$ at different *temperatures* T.



When the temperature T is high (left), the distribution is rather uniform, whereas when the temperature T is low (right), the distribution is "spiky", with all its mass on the largest element.

6.4.2 MLE

To obtain the probabilities for all C classes simultaneously given an example i, we calculate:

$$\boldsymbol{p}_i = S\left(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{x}_i\right) \in \mathcal{S}_C$$

The likelihood of a particular choice of *weight vectors* is given by multiplying up the probabilities that the model assigns to the correct class for each training example.

$$\mathcal{L}(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{y}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{W}) = \prod_{i=1}^{N} S\left(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{x}_i\right) = \prod_{i=1}^{N} p_{iy_i}$$

The Gradient of the neg. log-likelihood wrt. one of the weight vectors $\boldsymbol{w}_c$ is:

$$\nabla_{\boldsymbol{w}_c^{\mathrm{T}}} - \ell(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{y}) = \sum_i \left(p_{ic} - \mathbb{I}(y_i = c)\right) \boldsymbol{x}_i^{\mathrm{T}}$$

by weighting $\boldsymbol{x}_i^{\mathrm{T}}$ with the error $p_{ic} - \mathbb{I}(y_i = c)$ wrt. to class c and summing over all N training examples.

7 Linear Algebra Recap

7.1 Matrices & Transformations

Matrix multiplication is an operation that *transforms* vectors in some way.

7.1.1 Identity Matrix *I*

- A square matrix with 1's on the main diagonal.
- It corresponds to no transformation.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 7 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$$

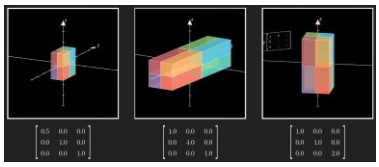
7.1.2 Scalar Matrices

- A square matrix with constant values on the main diagonal
- It scales vectors uniformly by the constant value *k*

$$\begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

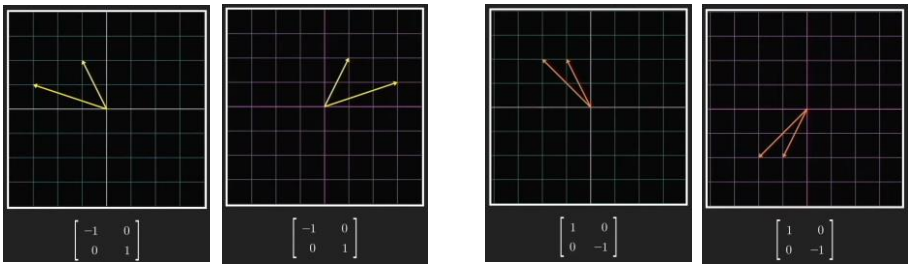
7.1.3 "Off one" Matrices

- Similar to ***I***, but one value is other than 1
- Scales vectors along certain axis based on value *k*



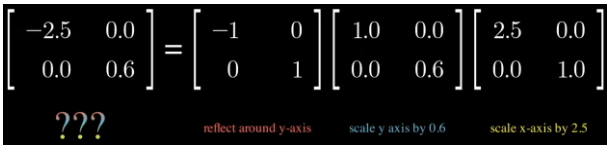
7.1.4 Identity with Negative Values

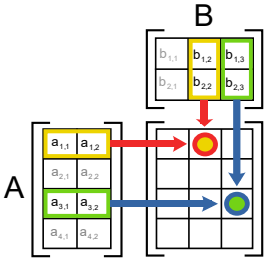
- Mirror vectors along an axis based on negative entry



7.1.5 Matrix Multiplication

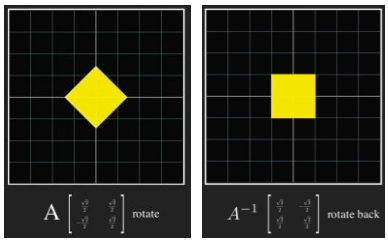
Matrix multiplication simply combines the effects of separate transformations by multiple matrices, into one single transformation.





Each entry is computed by taking the dot product of the corresponding row of the first matrix and the column of the second matrix.

7.1.6 Inverse Matrices A^{-1}

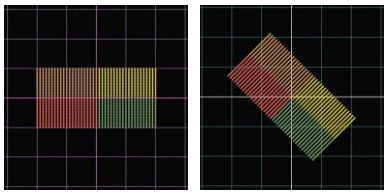


- Reverses the transformation of another matrix
- Not all matrices are invertible!

→ Therefore $\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$ holds.

💡 For all matrices $\boldsymbol{A}$ where $\det([\cdot]) \neq 0$, there exists $\boldsymbol{A}^{-1}$

7.1.7 Orthogonal Matrices



- A square matrix
- All col. vectors are unit vectors (magnitude of 1)
- All col. vectors are orthogonal (90°, dot product is 0)

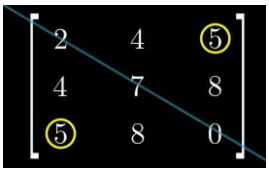
They always produce a perfect *rotation* transformation.

💡 Transposing an orthogonal matrix also creates its inverse:

$$\boldsymbol{A}^{\mathsf{T}} = \boldsymbol{A}^{-1}$$

which is another orthogonal matrix.

7.1.8 Symmetric Matrices



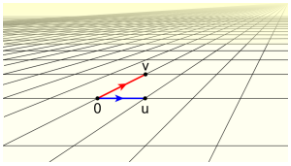
- Have the same numbers on each side of the main diagonal
- Rectangular (non-square) Matrices are never symmetrical

Transposing a symmetric matrix doesn't change anything:

$$\boldsymbol{A}^{\mathsf{T}} = \boldsymbol{A}$$

💡 Symmetric matrices have orthogonal eigenvectors.

7.2 Span



📄 The *linear span* (also called the *linear hull* or just *span*) of a set S of vectors, is defined as the *set of all linear combinations* of the vectors in S. For example, two linearly independent vectors $\vec{u}$ and $\vec{v}$ span a plane.

7.3 Rank

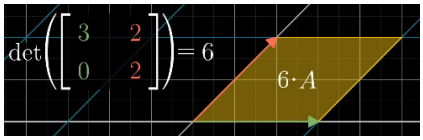
📄 The *rank* of a matrix $\boldsymbol{A}$ is the dimension/size of the vector space generated (or spanned) by its columns. In other words, it's the number of dimensions in the output of a transformation.

This corresponds to the maximal number of *linearly independent columns* of $\boldsymbol{A}$.

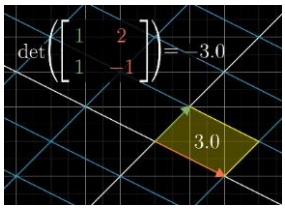
- $rank(\boldsymbol{A}) = 1$ corresponds to the transformation to a line by $\boldsymbol{A}$
- $rank(\boldsymbol{A}) = 2$ corresponds to the transformation to a plane by $\boldsymbol{A}$

💡 The set of all possible outputs of a transformation, is called the *column space* of $\boldsymbol{A}$.

7.4 Determinant



Measures how much the area A (*or volume in 3D*) in a given region is stretched/squished by the transformation of any matrix. What happens to one region then can be generalized to all other regions.



If $\det([\cdot]) = 0$, the area of any region gets reduced to a single line (or even point), more generally, squishing it into a *smaller dimension*. If this case, the columns of the matrix are linearly dependent.

If $\det([\cdot]) < 0$, the orientation of space has been inverted by the matrix.

💡 Also, $\det(M_1 M_2) = \det(M_1)\det(M_2)$ holds, which is intuitive as matrix multiplication simply combines the transformations of both matrices into one.

7.5 Column, Row and Null Space

📄 The *column space* of a matrix is a set of all possible linear combinations of its column vectors. It's a vector space itself. The rank of a matrix essentially tells you the dimension of the column space. The rank is a scalar quantity, while the column space is a vector space.

It's called that way, because the columns of a matrix tell us where the basis vectors land after applying the transformation. In other words, the column space is the span of the matrix's columns.

📄 The zero vector $\begin{bmatrix}0\\0\end{bmatrix}$ will always be included in the column space since linear transformations must keep the origin in place. For full rank transformations (no loss of dimensions) only the zero vector lands on $\begin{bmatrix}0\\0\end{bmatrix}$, but for matrices that lose dimension, a whole set of vectors can fall into the origin. This set of vectors is called the *null space* or *kernel* of the matrix. Depending on the number of dimensions lost it can be a line (-1 dim), a plane (-2 dim), and so on.

💡 Similarly, to the column space, the row space of a matrix is the set of all possible linear combinations of its row vectors. It's a vector space that is formed by these rows. The row rank and the column of any matrix are always equal.

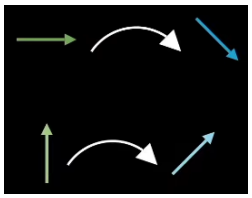
7.6 Eigenvectors & Eigenvalues

📄 Vectors for which the transformation by a matrix $\boldsymbol{A}$ does not change their direction, at most their magnitude, are called the *eigenvectors* of this matrix. All vectors along their span are also eigenvectors of this matrix.

📄 The corresponding *eigenvalue* λ of a eigenvector states the amount by which this vector gets scaled during the transformation by the matrix $\boldsymbol{A}$:

$$\boldsymbol{A}\vec{v} = \lambda\vec{v}$$

7.7 Spectral-/Eigendecomposition



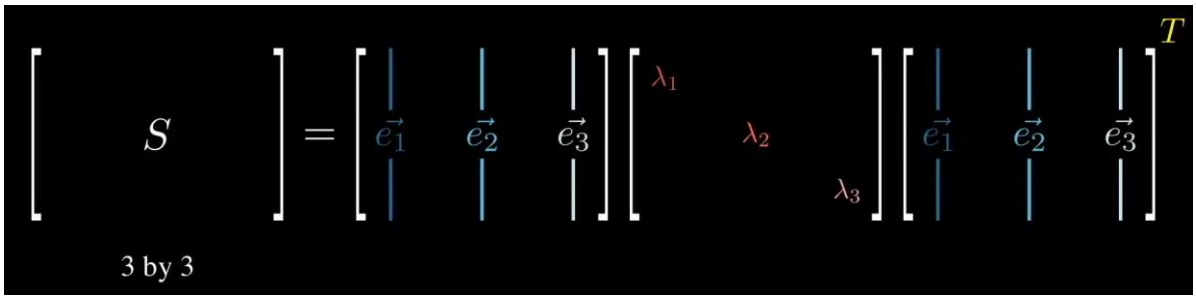
💡 For any symmetric matrix $\boldsymbol{S}$, there always exists an orthogonal matrix $\boldsymbol{Q}$, that rotates the standard basis vectors ($\begin{bmatrix}1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$) onto the eigenvectors of $\boldsymbol{S}$. Similarly, $\boldsymbol{Q^{-1}}$ rotates $\boldsymbol{S}$'s eigenvectors onto the axes. This is because the *eigenvectors of a symmetric matrix are orthogonal* and thus can form a basis for the vector space.

Theorem

For a symmetric matrix $\boldsymbol{S}$ you can always decompose it into a sequence of three matrices:

$$\boldsymbol{S} = \boldsymbol{Q}\,\boldsymbol{\Lambda}\,\boldsymbol{Q}^{\mathrm{T}}$$

where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{\Lambda}$ a diagonal matrix. For this, the columns of $\boldsymbol{Q}$ have to be the *normalized eigenvectors* of $\boldsymbol{S}$, on $\boldsymbol{\Lambda}$'s diagonal has to have the eigenvalues of $\boldsymbol{S}$.



7.8 Vector Norm

📝 The *norm* of a vector $\boldsymbol{v} = (v_1 \ldots v_n)^{\mathrm{T}}$ defines its magnitude. Generally, many different norms can be described by:

$$\|\boldsymbol{v}\|_p == \left(\sum_{i=1}^{n} |v_i|^p\right)^{\frac{1}{p}}$$

💡 The ℓ_p norms never increase as p increases, i.e., $\|v\|_{p+a} \leq \|v\|_p$ for $a \geq 0$

Examples

The *Euclidean norm* (or ℓ_2 norm, (*bird-fly distance* from origin) is defined as:

$$\|\boldsymbol{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2} = \left(\sum_{i=1}^{n} v_i^2\right)^{\frac{1}{2}}$$

The ℓ_1 norm is simply the *sum of absolute values* (or *Manhattan distance* from origin):

$$\|\boldsymbol{v}\|_1 = \sum_{i=1}^{n} |v_i|$$

The infinity norm (ℓ_∞) gives the maximum absolute value of a vector.

8 Dimensionality Reduction

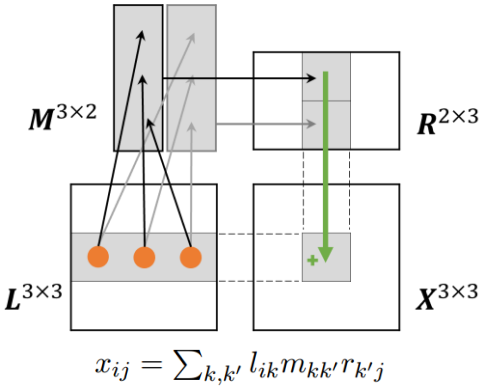
8.1 Matrix Decomposition

A matrix decomposition of a data matrix $\boldsymbol{X}$ is given by three factor matrices $\boldsymbol{L}, \boldsymbol{M}, \boldsymbol{R}$ such that:

$$\boldsymbol{X} = \boldsymbol{LMR}$$

where:

- $\mathbf{X}$ is a $m \times n$ data matrix
- $\boldsymbol{L}$ is a $m \times r_1$ data matrix
- $\boldsymbol{M}$ is a $r_1 \times r_2$ data matrix
- $\boldsymbol{R}$ is a $r_2 \times n$ data matrix
- r_1, r_2 are integers ≥ 1



We can think of $\boldsymbol{L}$ and $\boldsymbol{R}$ as compact representations of $\boldsymbol{X}$, given that their dimensionalities are smaller. To make decompositions useful, the factor matrices need to satisfy certain properties. In practice we rather use approximations, such that: $\boldsymbol{X} \approx \boldsymbol{LMR} = \boldsymbol{X}$

Effect

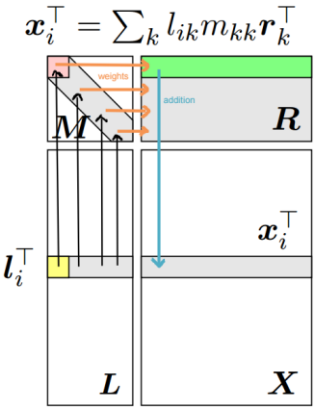
- the approximate decomposition can be much smaller than exact (small r)
- the reconstruction $\widehat{\boldsymbol{X}}$ can be viewed as a denoised version of $\boldsymbol{X}$
- It can lead to more insightful/interpretable decompositions.
- More efficient to compute

The notion of closeness ($\approx$) can be defined via a *reconstruction error* $E(\boldsymbol{X}, \widehat{\boldsymbol{X}})$ function.

8.2 Factor Interpretation

Assume $\boldsymbol{M}$ is a diagonal matrix with $r \times r$.

- **Rows** of $\boldsymbol{R}$ (parts) can be viewed as *latent objects/factors.*
- **Entries** of $\boldsymbol{M}$ are *weights* of corresponding factors.
- Rows of $\boldsymbol{MR}$ we call *weighted parts*.
- **Rows** of $\boldsymbol{L}$ are representations of objects constructed via weighted parts, called *embedding, code, scores, ...*
- *Size* r controls compactness of decomposition

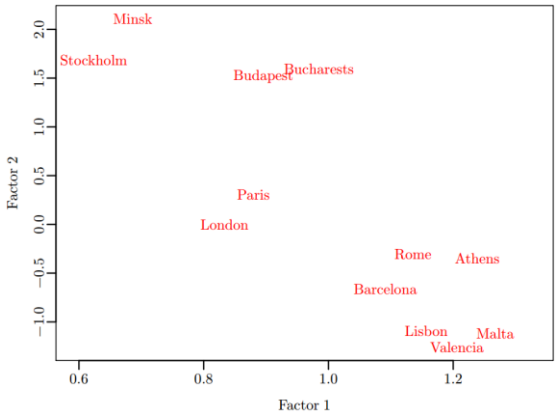


Each feature $\boldsymbol{X}_J$ can *also* be viewed as a combination of r *weighted latent features* $\boldsymbol{l}_j^\top$.

Example: Weather Data

Factor 1 generally describes the spectrum of rather cold/warm cities, while factor 2 then describes pronounced extremes in summer/winter. Example for factor 2: Valencia deviates less from its yearly mean ($\pm$8°C) compared to Minsk ($\pm$ 10°C).

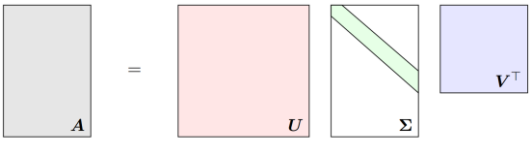
1.00			9.05	16.55	26.73	18.75	17.81
	1.00		-4.14	0.27	2.32	-0.89	-0.69
			Jan	Apr	Jul	Oct	Year
0.62	1.69	Stockholm	-1.34	10.79	20.59	10.20	9.95
0.69	2.11	Minsk	-2.52	11.94	23.23	10.99	10.77
0.83	0.00	London	7.54	13.80	22.28	15.63	14.85
0.90	1.52	Budapest	1.82	15.24	27.46	15.45	14.91
0.88	0.30	Paris	6.71	14.65	24.22	16.23	15.46
0.98	1.59	Bucharest	2.31	16.74	30.02	17.05	16.44
1.09	-0.66	Barcelona	12.61	17.88	27.64	21.05	19.90
1.14	-0.31	Rome	11.55	18.71	29.64	21.57	20.44
1.16	-1.09	Lisbon	15.00	18.85	28.39	22.67	21.36
1.24	-0.35	Athens	12.65	20.41	32.31	23.54	22.31
1.21	-1.26	Valencia	16.14	19.62	29.31	23.74	22.36
1.27	-1.12	Malta	16.10	20.67	31.29	24.76	23.35
							$\hat{X}$



Reconstruction $\hat{\boldsymbol{X}}$ of weather data $\boldsymbol{X}$ with $r = 2$ & $RMSE = 0.66$. Plot uses $\boldsymbol{L}$.

8.3 Singular Value Decomposition (SVD)

For each $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, there are orthogonal matrices $\boldsymbol{U}_{m \times m}$, $\boldsymbol{V}_{n \times n}$, and a diagonal matrix $\boldsymbol{\Sigma}_{m \times n}$ with values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$ on the main diagonal such that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$.



- $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ is called the *singular value decomposition* of $\boldsymbol{A}$
- σ_i are the *singular values* of $\boldsymbol{A}$
- Columns of $\boldsymbol{U}$ are the *left singular vectors* $\vec{u_j}$ of $\boldsymbol{A}$
- Columns of $\boldsymbol{V}$ are the *right singular vectors* $\vec{v_j}$ of $\boldsymbol{A}$

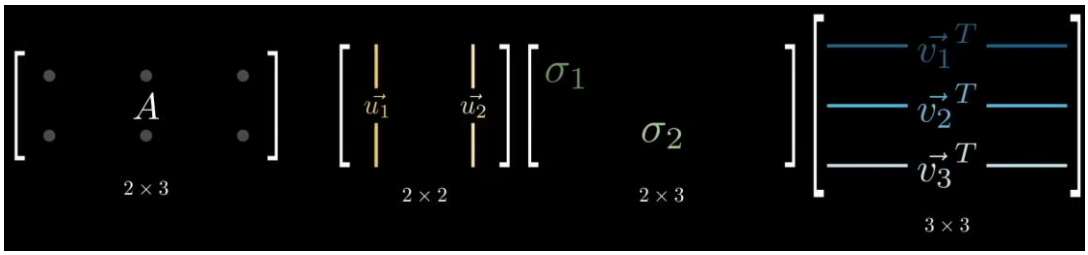
❗ For all decompositions of $\boldsymbol{A}$, the matrix $\boldsymbol{\Sigma}$ is fixed and unique within the matrix.

When you multiply any matrix $\boldsymbol{A}$ by its transpose $\boldsymbol{A}^\top$ and vice versa, the resulting matrix $\boldsymbol{A}\boldsymbol{A}^\top$ is always symmetric and square, regardless of the shape of $\boldsymbol{A}$.

8.3.1 Example

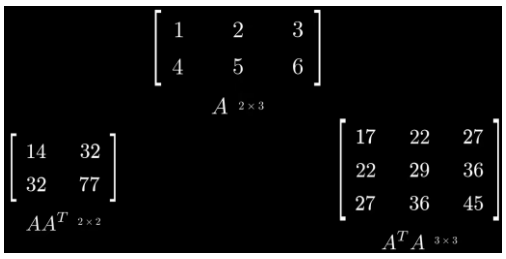
The primary appeal of SVD is that it generalizes to rectangular matrices. So, for any matrix $\boldsymbol{A}$, regardless of Symmetry, Dimension or Rank, can be decomposed into three factor matrices:

$$\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^\top$$



Consider the example on the right. Let's denote $\boldsymbol{A}\boldsymbol{A}^\top \overset{\text{def}}{=} \boldsymbol{S}_L$ and $\boldsymbol{A}^\top\boldsymbol{A} \overset{\text{def}}{=} \boldsymbol{S}_R$.

As they're symmetric matrices, they have orthogonal eigenvectors. $\boldsymbol{S}_L$ has two $\vec{u}_1, \vec{u}_2$ and $\boldsymbol{S}_R$ has three $\vec{v}_1, \vec{v}_2, \vec{v}_3$. They are called the *left/right singular vectors* of matrix $\boldsymbol{A}$.



The eigenvalue to each corresponding eigenvector $\lambda_i \geq 0$ is positive. Again, $\boldsymbol{S}_L$ has two λ_1, λ_2 and $\boldsymbol{S}_R$ has three $\lambda_1, \lambda_2, \lambda_3$. The biggest values of the two vectors are always equal: $\lambda_1^u = \lambda_1^v$, $\lambda_2^u = \lambda_2^v$. The leftover eigenvalue λ_3 is 0.

If we take their square root , we receive the singular values σ_1, σ_2 of matrix $\boldsymbol{A}$:

$$\sqrt{\lambda_1} = \sigma_1 \qquad \sqrt{\lambda_2} = \sigma_2$$

With this, we can construct the SVD.

8.3.2 Moore-Penrose Pseudo-Inverse

Consider the problem of least squares:

Given $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$, find $\boldsymbol{x} \in \mathbb{R}^n$ that minimizes $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. For example, $\boldsymbol{A}$ may correspond to the design matrix, $\boldsymbol{b}$ to the labels and $\boldsymbol{x}$ to the weight vector of linear regression.

To solve $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = 0$, we could:

1. rearrange to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$
2. multiply both sides with the inverse $\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{b}$
3. as $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{I}$ this simplifies to $\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{b}$

given that $\boldsymbol{A}$ is invertible. If not, we could use a *pseudo-inverse* $\boldsymbol{A}^+$, that has similar properties**.**

The *Moore-Penrose pseudo-inverse* of $\boldsymbol{A}^{m \times n}$ is a matrix $\boldsymbol{A}^+ \in \mathbb{R}^{n \times m}$ satisfying these criteria:

- $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A}$, note: $\boldsymbol{A}\boldsymbol{A}^+ \neq \boldsymbol{I}$ is possible
- $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^*$
- $(\boldsymbol{A}\boldsymbol{A}^+)^\top = \boldsymbol{A}\boldsymbol{A}^+ \rightarrow \boldsymbol{A}\boldsymbol{A}^+$ is symmetric
- $(\boldsymbol{A}^+\boldsymbol{A})^\top = \boldsymbol{A}^+\boldsymbol{A}$

As $\boldsymbol{A}^+ \in \mathbb{R}^{n \times m}$, this allows inverses of any matrices, as opposed to square-only before.

Example

For diagonal matrices (they only have non-zero values on their main diagonal), the *Moore-Penrose pseudo-inverse* takes a very simple form:

Given $\boldsymbol{A} = \begin{bmatrix} 5 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}$, its pseudo-inverse are the reciprocals (x^{-1} or $\frac{1}{x}$): $\boldsymbol{A}^+ = \begin{bmatrix} \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}$.

Relation to SVD

If the SVD of $\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^\top$, then $\boldsymbol{A}$'s pseudo inverse can be obtained by $\boldsymbol{A}^+ = \boldsymbol{V}\,\boldsymbol{\Sigma}^+\boldsymbol{U}^\top$.

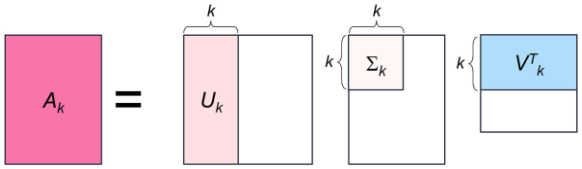
Factor matrix $\boldsymbol{\Sigma}^+$ then simply replaces each $\sigma_i > 0$ by its reciprocal $\frac{1}{\sigma_i}$ and then transposes, as $\boldsymbol{\Sigma}$ is a diagonal matrix.

For the *least squares problem*, an optimum solution is $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b}$.

8.4 Truncated SVD

The *size-k truncated SVD* is obtained by only taking the first k columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ and the main $k \times k$ submatrix of $\boldsymbol{\Sigma}$.

$$\boldsymbol{A}_k = \sum_{j=1}^{k} \sigma_j \, \boldsymbol{u}_j \, \boldsymbol{v}_j^\top = \boldsymbol{U}_k \, \boldsymbol{\Sigma}_k \, \boldsymbol{V}_k^\top$$



As *zero-valued singular values* $\sigma_j = 0$ neutralize the corresponding components' $\boldsymbol{u}_j, \boldsymbol{v}_j^\top$ impact, they do not matter for reconstruction. The truncated SVD now removes these parts we do not need when wanting to reconstruct data. The rank of a matrix is exactly the number of its *non-zero singular values* σ_j.

Properties

- $rank(\boldsymbol{A}_k) = k$ if $\sigma_k > 0$. So, the truncation can only be smaller than $\boldsymbol{A}$, never lager
- $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$ are not orthogonal anymore, but only *column-orthogonal*

Terminology

- If we choose $k = \min\{m, n\}$, then reconstruction is exact ($\boldsymbol{A}_k = \boldsymbol{A}$), and called *thin SVD*
- If we choose $k = rank(\boldsymbol{A})$, then reconstruction is also exact, and is called *compact SVD*
- If we choose $k < rank(\boldsymbol{A})$, then $\boldsymbol{A}_k$ is a *low-rank approximation* of $\boldsymbol{A}$

8.4.1 Frobenius Norm

The general *Frobenius norm* of a matrix $\boldsymbol{A}$ is the vector-ℓ_2 norm applied to the elements of a matrix a_{ij}, treating it like a vector: $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$. It is a measure of matrix difference.

In terms of SVD, the *squared Frobenius norm* is equal to $\|\boldsymbol{A}\|_F^2 = \sum_{i=1}^{\min\{m,n\}} \sigma_i^2$. This is because the singular values capture the 'size' of the matrix in their respective dimensions.

Consequently, the squared Frobenius norm of the truncated SVD is: $\|\boldsymbol{A}_k\|_F^2 = \sum_{i=1}^{k} \sigma_i^2$

The approximation error for the truncation is: $\|\boldsymbol{A} - \boldsymbol{A}_k\|_F^2 = \sum_{i=k+1}^{\min\{m,n\}} \sigma_i^2$. This is the sum of the squared singular values that we did not include. To obtain the MSE, we can divide this by the number of entries in our original matrix.

8.4.2 Spectral Norm

The *spectral norm* of $\boldsymbol{A}$, denoted $\|\boldsymbol{A}\|_2$, is the largest singular value σ_1:

$$\|\boldsymbol{A}\|_2 = \sigma_1$$

Recall the descending ordering of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$. This norm measures the maximum stretching factor of the matrix.

Therefore, $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F \leq \sqrt{n}\, \|\boldsymbol{A}\|_2$. Meaning that the spectral norm is always less than or equal to the Frobenius norm, and the Frobenius norm is $\leq \sqrt{\text{n}}$-times the spectral norm.

8.4.3 Eckart-Young Theorem

Let $\boldsymbol{A}_k = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^\top$ be the size-k truncated SVD of A with $k \leq rank(\boldsymbol{A})$. Then $\boldsymbol{A}_k$ is best rank-k approximation to $\boldsymbol{A}$ in terms of Frobenius norm.

$$\|\boldsymbol{A} - \boldsymbol{A}_k\|_F \leq \|\boldsymbol{A} - \boldsymbol{B}_k\|_F \qquad \text{for all rank-}k\text{ matrices } \boldsymbol{B}$$

💡 In summary, the Eckart-Young Theorem justifies the use of truncated SVD as the optimal way (in terms of Frobenius norm) to approximate a matrix by another matrix of lower rank.

8.5 Interpreting the SVD

If two *rows* of $\boldsymbol{U}$ have similar values in one of the columns, then these *examples* are similar wrt. this particular *latent feature*.

If two *columns* of $\boldsymbol{V}^\top$ have similar values in one of the rows, then these *latent attributes* are somehow similar.

In both cases, the first entries (leftmost $\boldsymbol{U}$, uppermost $\boldsymbol{V}^\top$) have the largest singular value and matter most.

Σ_2

147.54	
	20.09

$V_2^\top$

0.22	0.40	0.64	0.45	0.43
-0.85	0.06	0.47	-0.18	-0.14

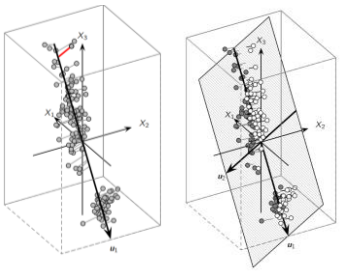
U_2

0.18	0.41
0.19	0.51
0.24	0.00
0.25	0.37
0.25	0.07
0.28	0.39
0.31	-0.16
0.32	-0.07
0.33	-0.27
0.35	-0.08
0.34	-0.31
0.36	-0.27

X

	Jan	Apr	Jul	Oct	Year
Stockholm	-0.70	8.60	21.90	9.90	10.00
Minsk	-2.10	12.20	23.60	10.20	10.60
London	7.90	13.30	22.80	15.20	14.80
Budapest	1.20	16.30	26.50	16.10	15.00
Paris	6.90	14.70	24.40	15.80	15.50
Bucharest	1.50	18.00	28.80	18.00	16.50
Barcelona	12.40	17.60	27.50	21.50	20.00
Rome	11.90	17.70	30.30	21.40	20.40
Lisbon	14.80	19.80	27.90	22.50	21.50
Athens	12.90	20.30	32.60	23.10	22.30
Valencia	16.10	20.20	29.10	23.60	22.30
Malta	16.10	20.00	31.50	25.20	23.20

8.5.1 Geometric Interpretation



One can think of the *right singular vectors* $\overrightarrow{v_1}$ as giving directions in which the data varies a lot. The first right singular vector has the largest variation "along its axis". The *first left singular vector* $\overrightarrow{\boldsymbol{u}_1}$ tells us where each data point is on the line *spanned* by the *first right singular vector*.

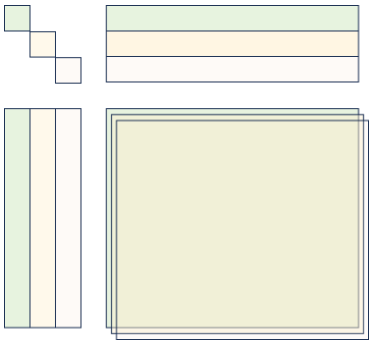
Scaling by the singular values: $\overrightarrow{\boldsymbol{u}_1}\sigma_1$ and looking at an entry for a single data point, gives the orthogonal projection of the data point onto the line (red residual shown in plot) spanned by the first right singular vector. So $\overrightarrow{\boldsymbol{v}_1^\top}$ covers a lot of variation and $\overrightarrow{\boldsymbol{u}_1}\sigma_1$ tell us along this direction of variation, where is each data point placed.

This reasoning can further be applied to each subsequent singular vector (second, third, etc.) is orthogonal to the previous ones and captures variation along a new axis. This orthogonality ensures that each singular vector identifies a unique direction of variance in the data.

8.5.2 Component Interpretation

Recall that we can write:

$$\boldsymbol{X} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^\top = \sum_i^r \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top = \sum_i^r \boldsymbol{X}_{(i)}$$



The components of the SVD are $\boldsymbol{X}_{(i)}$, so the combination of the i-th column of $\boldsymbol{U}$ and the i-th row of $\boldsymbol{V}^\top$, scaled by the i-th singular value. These components form layers, and their sum is the final reconstruction.

- The first layer explains the most
- The second layer corrects by adding/removing small values
- The third layer corrects with even smaller values

8.6 Determining Size k

❓ How to select size k of truncated SVD?

- To small: all subtlety is lost.
- Too big: all smoothing is lost

Guttman-Kaiser Criterion

📄 Select k such that we include all components with $\sigma_i \geq 1$ and exclude those with $\sigma_i < 1$.

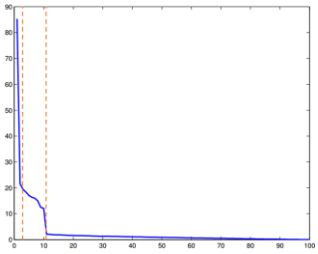
Squared Singular Values

📄 Select k such that $\sum_{i=1}^{k} \sigma_i^2 \geq 90\%$ of the total sum of squared singular values. The motivation is that the resulting matrix "explains" 90% of the squared *Frobenius Norm* $\|\boldsymbol{A}_k\|_F^2$ of the matrix.

❗ Both of these methods are based on arbitrary thresholds and don't consider the data's shape.

8.6.1 Cattell's Scree Test

The *scree plot* shows squared singular values in decreasing order. It should look like debris (scree) in front of a hill.



The scree test is a subjective decision of the size based on the shape of the plot. Size k should be set to a point where there is a clear drop in magnitude of the values; or the values start to even out.

❗ Data that does not look like this becomes hard to truncate.

8.6.2 Entropy-based Method

📄 Considers the relative contribution of each singular value to the overall sq. Frobenius Norm.

$$\text{Relative contribution of } \sigma_k \text{ is } f_k = \sigma_k^2 \,/\, \textstyle\sum_i \sigma_i^2$$

We can treat these (f_k) as probabilities and define the normalized *entropy* of the sing. values:

$$E = -\frac{1}{\log(\min\{n,m\})} \sum_{i}^{\min\{m,n\}} f_i \log f_i$$

We normalize the entropy of these values by the maximum possible value $\frac{1}{\log(\min\{n,m\})}$.

The size is selected to be the smallest k such that:

$$\sum_{i=1}^{k} f_i \geq E$$

💡 Low entropy (close to 0) means that the first singular values has almost all mass, while high entropy (close to 1) means that the singular values are almost equal.

8.6.3 Random Flip of Signs

We multiply every element of the matrix $\boldsymbol{A}$ randomly with either 1 or -1 to obtain $\widetilde{\boldsymbol{A}}$.

- *Frobenius Norm* does not change: $\|\boldsymbol{A}\|_F = \|\widetilde{\boldsymbol{A}}\|_F$
- *Spectral Norm* does change: $\|\boldsymbol{A}\|_2 \neq \|\widetilde{\boldsymbol{A}}\|_2$

💡 In matrices with a lot of structure the difference in spectral norm is large. For matrices without much structure this alteration does not really matter, hence the difference is low.

📄 Idea: select k such that the *residual matrix* $\boldsymbol{X}_{-k}$ contains only noise

- $\boldsymbol{X}_{-k} = \boldsymbol{X} - \boldsymbol{X}_k$ is the *residual matrix* left after truncation with size-k
- Construct $\widetilde{\boldsymbol{X}}_{-k}$ from $\boldsymbol{X}_{-k}$ by randomly flipping signs, then select k to be such that

$$\frac{\|\boldsymbol{X}_{-k}\|_2 - \|\widetilde{\boldsymbol{X}}_{-k}\|_2}{\|\boldsymbol{X}_{-k}\|_F}$$

is small. This essentially checks if the residual matrix $\boldsymbol{X}_{-k}$ still contains structure/information and if yes, we want to increase k.

8.7 Data Preprocessing for SVD

Consider *data normalization* before applying SVD, as SVD is sensitive to data scaling. Attributes with bigger values and scales (height [m] vs. weight [g]) carry more importance if left unnormalized.

Further, *data needs to be centered* (by subtracting the column mean). If, for example, all data is positive, the first singular vector will just explain where in the positive quadrant the data is.

uncentered data *centered data*

💡 Computing *z-scores* (per column) does both:

$$\boldsymbol{z} = \frac{\boldsymbol{x} - \mu}{\sigma}$$

where σ is the standard deviation and μ the mean. Note that z-scores assume normal distribution, hence the division by σ actually normalizes the data. Further it is assumed that all attributes are actually equally important, as they are brought into the same scale.

8.8 Relationship to PCA

💡 The *truncated SVD* (and PCA) can be used to battle the curse of dimensionality. If we choose $k \ll n$, we perform *dimensionality reduction*.

SVD is closely related to *principal component analysis* (PCA). The key difference is the centering of the data.

$$PCA \approx SVD\ on\ centered\ data$$

Curse of Dimensionality

For very high dimensional spaces, all data points are very far from each other. High dimensionality slows down data mining algorithms.

8.9 Other Uses of SVD

Denoising with SVD

Another common application of SVD is to remove noise from the data. Perturbations with (gaussian, iid.) random noise does not significantly affect low-rank approximations. Assume:

$$\boldsymbol{X} = \boldsymbol{A} + \boldsymbol{E}$$

where $\boldsymbol{A}$ is low-rank data and $\boldsymbol{E}$ is noise. If noise is iid. and not too large, we can approximately recover $\boldsymbol{A}$ by taking the truncated SVD of $\boldsymbol{X}$. As before, the key problem is to select k.

Latent Semantic Analysis

Consider a document-term matrix $\boldsymbol{X}$, usually with *tf-idf values* (frequency of a term in a document divided by global term frequency)**.** The truncated SVD $\boldsymbol{X}_k = \boldsymbol{U}_k \, \boldsymbol{\Sigma}_k \, \boldsymbol{V}_k^\top$ is computed**:**

- Matrix $\boldsymbol{U}_k$ associates documents to *topics*
- Matrix $\boldsymbol{V}_k$ associates *topics* to terms

💡 If two rows of $\boldsymbol{U}_k$ are similar, the corresponding documents talk about the *same topics*.

Also: **Visualization** (see *8.2 Factor Interpretation*) […].

8.10 Latent Linear Models

📄 *Latent Variable Models* (LVM) are models that assume that the data is explained by a set of *unobserved latent variables*.

Consider the weather data in *8.2 Factor Interpretation*: one latent variable z_1 may correspond to mean temperature (cold/warm) and another latent variable z_2 may correspond to temperature variance (balanced/hot summers vs. mild/icy winters). LVMs then assume that a city's temperature data ($\boldsymbol{x}_i$) can be explained by the values of its latent variables $\boldsymbol{z}_i = (z_1 \quad z_2)^\top$.

LVMs are conceptually very general. Latent variables can be used to model dependencies (cf. graphical models), e.g., between features or between examples (non-iid data).

📄 *Latent Linear Models* (LLM) model linear relationships between the example $\boldsymbol{x}$ and the respective latent variable $\boldsymbol{z}$:

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$$

where, $\boldsymbol{x} \in \mathbb{R}^D$ refers to a data point and $\boldsymbol{z} \in \mathbb{R}^L$ refers to latent variables for this data point. Parameter $\boldsymbol{\mu} \in \mathbb{R}^D$ is a bias term (usually the mean) and the *factor loading matrix* $\boldsymbol{W} \in \mathbb{R}^{D \times L}$ describes how to transform $\boldsymbol{z}$**.** Lastly, $\boldsymbol{\epsilon} \in \mathbb{R}^D$ is a noise term (independent, mean 0).

D is *number of features* (observed variables) *in each data point* $\boldsymbol{x}$ and L represents the dimensionality of the *latent variable space*. Each latent variable $\boldsymbol{z}$ is an L-dimensional vector. Typically, $L < D$, indicating that the latent space is a lower-dimensional representation of the observed data. Therefore, *factor loading matrix* $\boldsymbol{W}$ maps the latent variables in the lower-dimensional space to the higher-dimensional observed data space. This matrix is crucial for understanding how changes in the latent space affect the observed data.

💡 We can view SVD and PCA as LLMs with a particular choice of variables and parameters.

Recall that in SVD, a single example can be obtained by $\boldsymbol{x}_i = \boldsymbol{V}\,\boldsymbol{\Sigma}^\top \boldsymbol{u}_i$, which we can bring in the form of a latent variable model:

- $\boldsymbol{\mu} = 0$; $\boldsymbol{W} = \boldsymbol{V}$; $\boldsymbol{z}_i = \boldsymbol{\Sigma}^\top \boldsymbol{u}_i$, where $\boldsymbol{u}_i = \boldsymbol{U}_{i:}^\top$

Similarly, for PCA, where a single example can be obtained by $\boldsymbol{x}_i = \boldsymbol{Q}\mathbf{z_i} + \boldsymbol{\mu}$, this gives:

- $\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{x}_i$; $\boldsymbol{W} = \boldsymbol{Q}$; $\boldsymbol{z}_i = \mathbf{Q}^\top(\boldsymbol{x}_i - \boldsymbol{\mu})$

❗ Neither model is generative, as we do not model the distribution of the latent variables (yet).

8.10.1 Factor Analysis Model

Factor analysis is an LLM, where latent variables $\boldsymbol{z}$ and noise is assumed to be Gaussian.

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

The latent variable follows a standard normal distribution, $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, where $\mathbf{0}$ is the mean vector and $\boldsymbol{I}$ is the identity matrix (indicating no correlation among the components of $\boldsymbol{z}$).

$$\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

The conditional distribution of $\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}$ is also Gaussian. Variable $\boldsymbol{x}$ represents the observed data. The mean of this distribution is $\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}$, where $\boldsymbol{W}$ is a weight matrix and $\boldsymbol{\mu}$ is a mean vector. The variance/noise around the mean $\boldsymbol{\mu}$, is given by a diagonal covariance matrix $\boldsymbol{\Psi} \in \mathbb{R}^{D\times D}$.

It is diagonal, so the individual components of the noise are independent (more/less noise across different features) so $\boldsymbol{z}_i$ explains the correlation in the data, not the noise term. Parameters are $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\Psi}\}$.

💡 One can show that the *marginal distribution* is Gaussian, too: $p(\boldsymbol{x}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{W}\boldsymbol{W}^\top + \boldsymbol{\Psi})$.

Essence

Factor analysis begins with a latent variable $\boldsymbol{z}$ that has a simple Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This latent variable $\boldsymbol{z}$ represents underlying, unobserved factors. This latent variable is then transformed using a set of parameters (like the weight matrix $\boldsymbol{W}$, mean vector $\boldsymbol{\mu}$, and covariance matrix $\boldsymbol{\Psi}$) into something that resembles real-world data (the observed variable $\boldsymbol{x}$).

The transformation typically involves linear combinations (as in $\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}$) and other operations that introduce complexity and variability, making the resulting data more representative of real-world observations.

These parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\Psi}\}$ need to be learned from the real-world data. They are not known a priori and are estimated based on the observed data.

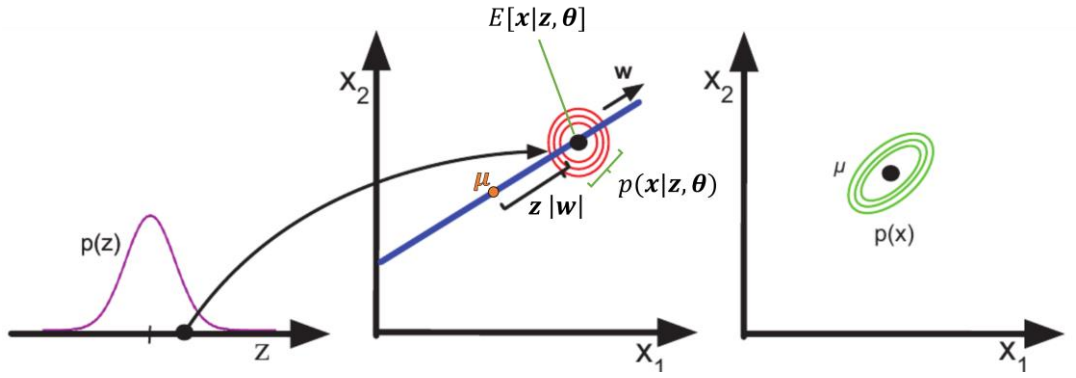
💡 Essentially, we're learning how to generate structure given random input data.

8.10.2 Probabilistic PCA (PPCA)

When the covariance matrix $\boldsymbol{\Psi} = \sigma^2 \boldsymbol{I}$ is isotropic Gaussian Noise, meaning each feature of $\boldsymbol{x}$ has the same amount of noise, then the resulting model is called *probabilistic PCA*.

In PPCA, data is generated by

- sampling from a standard multivariate Gaussian to obtain latent variables $\boldsymbol{z}$
- using this $\boldsymbol{z}$, we map it to $\boldsymbol{Wz} + \boldsymbol{\mu}$ functioning as our mean
- And finally add Gaussian Noise $\mathcal{N}(0, \boldsymbol{\Psi})$ to obtain example $\boldsymbol{x}$



The blue line represents the mapping from the *latent space* (purple) to the *data space* by applying $\boldsymbol{Wz} + \boldsymbol{\mu}$. This transformation is linear, involving scaling by $\boldsymbol{W}$ and translation by $\boldsymbol{\mu}$.

The black arrow labeled with $\boldsymbol{w}$ in the graphic represents the direction of the weight vector. It shows the primary direction of variation in the data space that is captured by the PPCA model. The latent variable $\boldsymbol{z}$ is projected onto this direction and scaled by the magnitude of $\boldsymbol{w}$. $\boldsymbol{z}|\boldsymbol{w}|$ represent this distance from the mean $\boldsymbol{\mu}$ along the direction of the weight vector $\boldsymbol{w}$ in the data space. The red ellipses represent the Gaussian noise $\mathcal{N}(0, \boldsymbol{\Psi})$ added at the end.

In the last frame, $\boldsymbol{\mu}$ is the mean of the observed data distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ in the high-dimensional space. The green ellipses $p(\boldsymbol{x})$ show where we are most likely to find data points $\boldsymbol{x}$ without considering any specific value of $\boldsymbol{z}$. It is not isotropic anymore. The spread of the green ellipse indicates the variance of the observed data.

8.10.3 Why Factor Analysis?

We assume that data $\boldsymbol{X}$ follows a multivariate Gaussian (MVN): $p(\boldsymbol{x}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{WW}^\top + \boldsymbol{\Psi})$.

Why not directly estimate the mean vector and covariance matrix of this Gaussian?

- PPCA requires less parameters (D for mean, DL for $\mathbf{W}$) than MVN (D for mean, D^2 for $\boldsymbol{\Sigma}$)
- We hope that latent variables $\boldsymbol{z}$ reveal interesting properties

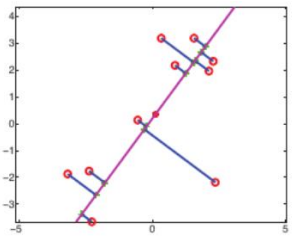
$$\boldsymbol{\Sigma} = (\boldsymbol{I} + \boldsymbol{W}^{-1}\boldsymbol{\Psi}^{-1}\boldsymbol{W})^{-1} \qquad \boldsymbol{m} = \boldsymbol{\Sigma}\boldsymbol{W}^\top\boldsymbol{\Psi}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$$

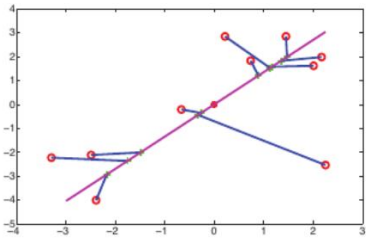
We call $\boldsymbol{m}$ *scores*, that refer to mean of the latent variables given the observed data. They represent the estimated position of each observed data point in the latent variable space, providing a lower-dimensional representation of the original data.

Latent variables serve as a compressed representation of the data, especially when the number of latent dimensions L is *less than* the original data dimensions D. This reduction in dimensions, with $\boldsymbol{z}$ and m existing in $\mathbb{R}^L$, makes $\boldsymbol{z}$ a *bottleneck* through which the data is compressed.

8.10.4 Reconstruction of PCA vs. PPCA

💡 The key takeaway is that PCA provides a deterministic linear projection of data onto lower dimensions, while PPCA provides a probabilistic model that accounts for uncertainty and noise in the data, leading to different projected points and potentially more robust representations, especially when dealing with noisy or incomplete data.





8.10.5 Unidentifiability

📄 Parameters of FA $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\Psi}\}$ are *unidentifiable*, i.e., multiple different parameter choices correspond to the same data distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$. This implies that the *true* parameters (assuming they exist) cannot be found, even with infinite data. This does not affect predictive performance but does affect the interpretation of the factors.

Common solutions to avoid this are

- Force $\boldsymbol{W}$'s columns to be orthogonal and order by their norm (as in PCA)
- Force $\boldsymbol{W}$'s to be lower triangular (only numbers on & below main diagonal)
- Use a (sparsity-promoting) prior on weights
- Use a non-Gaussian prior on the latent factors (e.g., ICA)

8.10.6 Discussion

- Parameters of FA $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\Psi}\}$ can be fit using the EM algorithm.
- For PPCA, the MLE estimate (assuming centered data) is:

$$\widehat{\boldsymbol{W}}_{MLE} = \boldsymbol{Q}_L(\boldsymbol{\Lambda}_L - \sigma^2\boldsymbol{I})^{\frac{1}{2}} \qquad \hat{\sigma}^2_{MLE} = \frac{1}{D-L}\sum_{j=L+1}^{D}\lambda_j$$

- When noise variance $\sigma^2 \to 0$ (no noise) in PPCA, we obtain PCA

9 Expectation-Maximization (EM) Algorithm

The *EM algorithm* is a *framework* to estimate model parameters with missing data. It is useful when observed data and missing data can be modeled jointly.

❗ The algorithm makes the following assumptions:

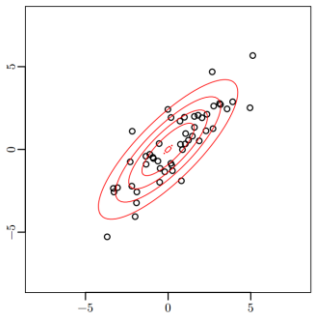
- Parameter estimation would be easy if all data was known (M step)
- Handling missing values would be easy when all parameters were known (E step)

💡 It is an iterative method that alternates between E step, using current parameter estimates, and M step, using filled in data.

9.1 Motivational Example: MLE for MVN

Consider a generative model $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ that models given data using an MVN. With N iid. observations, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, the ML estimate is obtained by the sample mean $\hat{\boldsymbol{\mu}}_{MLE}$ and the sample covariance $\hat{\boldsymbol{\Sigma}}_{MLE}$:

$$\hat{\boldsymbol{\mu}}_{MLE} = \frac{1}{N}\sum_i \boldsymbol{x}_i \qquad \hat{\boldsymbol{\Sigma}}_{MLE} = \frac{1}{N}\sum_i (\boldsymbol{x} - \hat{\boldsymbol{\mu}})(\boldsymbol{x} - \hat{\boldsymbol{\mu}})^\top$$



MLE is successfully used to estimate the parameters $\hat{\boldsymbol{\mu}}_{MLE}$ and $\hat{\boldsymbol{\Sigma}}_{MLE}$ of the underlying Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ that generated the observed data.

So far, we assume all variables are observed and available, e.g., in supervised learning $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ during training, and $\boldsymbol{x}_{test}$ during prediction. But how can we fit parameters if data is missing?

9.1.1 Missing Data Mechanisms

Reasons for Missing Data

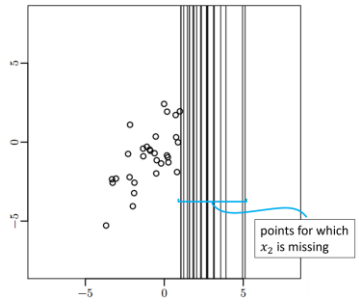
- High cost of complete data acquisition
- Errors in data acquisition
- Non-response in surveys
- Latent Variable Models

❗ The reason why its missing is important → *missing data mechanisms* describe the relationship between the complete data and the event that a data item is missing.

Missing Completely At Random MCAR	**Missing At Random** MAR	**Missing Not At Random** MNAR
The event that a data item is missing is independent of observed and missing data. There is no systematic reason for why it is missing. **Example**: some questions only asked to random subset of persons in survey.	Event that a data item is missing depends on observed data but not on missing data. Reason is systematic but can be explained by observed data. **Example**: only the students who passed assignments write the exam (missing for some).	Event that data item is missing can depend on observed and missing data. ❗ non-ignorable **Example**: persons with high-income tend not answer questions about income

The exact mechanism often cannot be determined. In this course, we are mainly interested in latent variable models (→ MCAR). Generally, we subsequently assume: MCAR or MAR ❗

9.1.2 Example (1): MVN with Missing Values

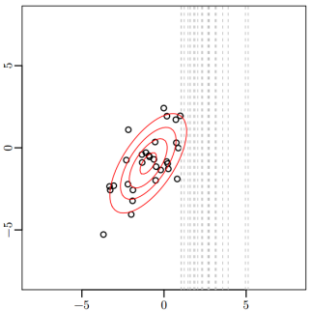


💡 Recall the example from *section 9.1*.

Let's assume a MAR mechanism, where in each example (x_1, x_2), variable $x_2$is likely to be missing when $x_1 > 1$.

How can we obtain MLE estimates for the parameters $\hat{\boldsymbol{\mu}}_{MLE}$ and $\hat{\boldsymbol{\Sigma}}_{MLE}$ of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ that generated the data?

One way to obtain the MLE estimate for $\hat{\boldsymbol{\mu}}_{MLE}$ and $\hat{\boldsymbol{\Sigma}}_{MLE}$, is to use *Complete Case Analysis* (or *listwise deletion*). The idea is to ignore all missing data to perform parameter estimation.



❗ This approach can lead to biased estimates when the mechanism is not MCAR. It does not use all available data and is not useful at all for LVMs!

We see, the resulting estimate does not match the data very well.

9.1.3 Statistical Properties of MVN

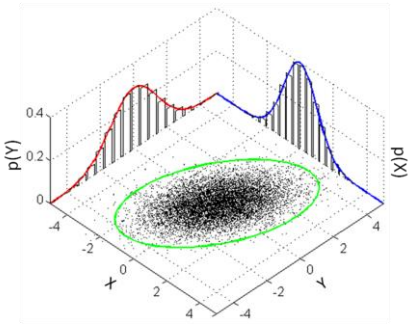
Recall that we can express variance σ^2 as precision λ:

$$\lambda = \frac{1}{\sigma^2} = \sigma^{2^{-1}}$$

Similarly, for the multivariate case, the precision matrix $\boldsymbol{\Lambda}$ is equal the inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$:

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$$

In a sense, covariance can be thought of as a multidimensional extension of variance (cf. *1.3.2*).



Sample points from a MVN distribution with $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ *and* $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 3/5 \\ 3/5 & 2 \end{bmatrix}$ *shown along with the* 3σ*-ellipse, the two marginal distributions, and the two histograms.*

💡 Imagine you are studying the environment and are collecting various data points. Your vector $\boldsymbol{x}$ consists of:

- Temperature
- Humidity
- Precipitation
- Air Quality Index
- Wind Speed
- Solar Radiation

Now, suppose you want to study the relationship between meteorological factors and pollution levels separately. You can partition your vector $\boldsymbol{x}$ into two subvectors: $\boldsymbol{x}_1$ for meteorological data and $\boldsymbol{x}_2$ for pollution data:

- $\boldsymbol{x}_1$ includes Temperature, Humidity, Precipitation, and Wind Speed
- $\boldsymbol{x}_2$ includes Air Quality Index (AQI) and Solar Radiation

This partitioning allows you to simplify the analysis and focus on specific relationships within a large set of environmental variables. By analyzing the joint Gaussian distribution of these variables, you could then look at the marginal distribution of just the meteorological factors $(\boldsymbol{x}_1)$ to understand the typical weather patterns without considering pollution levels $(\boldsymbol{x}_2)$ and

vice versa. Furthermore, you might be interested in the conditional distributions, such as how pollution levels (in $\boldsymbol{x}_2$) behave given certain meteorological conditions ($\boldsymbol{x}_1$), which could be crucial for predicting pollution levels and issuing health advisories.

9.1.4 Theorem: Marginals and Conditionals in Gaussian Models

This theorem captures the relationships between variables. The theorem states that if we have a vector $\boldsymbol{x}$, which in our case contains both meteorological and pollution measurements, and this vector is jointly Gaussian with parameters (mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$), then we can analyze the marginals and conditionals of this vector in a structured way.

Theorem

Suppose $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ *is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}.$$

Then the marginals are given by

$$p(\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \quad p(\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}),$$

and the conditional distribution by

$$\begin{aligned} p(\boldsymbol{x}_1 | \boldsymbol{x}_2) &= \mathcal{N}(\boldsymbol{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned}$$

Determine Marginal Distributions

The theorem provides the methodology to calculate the *marginal distribution* of each subset of variables (meteorological $p(\boldsymbol{x}_1)$ and pollution $p(\boldsymbol{x}_2)$). This means we can describe the behavior of weather variables without considering pollution variables and vice versa.

Calculate Conditional Distributions

Perhaps more powerfully, the theorem gives us the conditional distributions, which tell us how one set of variables behaves given specific values of the other set. In our case, this could answer questions like "Given a high level of solar radiation, what is the distribution of temperature and humidity?" $p(\boldsymbol{x}_2|\boldsymbol{x}_1)$).

Adjust Predictions Based on New Information

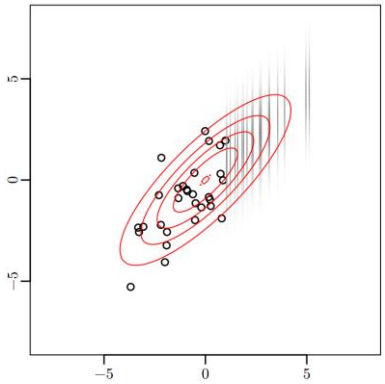
The conditional mean $\boldsymbol{\mu}_{1|2}$ and covariance formulas $\boldsymbol{\Sigma}_{1|2}$ provided by the theorem enable us to update our predictions of one set of variables (like weather conditions $\boldsymbol{x}_1$) based on new information about the other set (like pollution levels $\boldsymbol{x}_2$). In practical terms, if you know the outcome of $\boldsymbol{x}_2$, the conditional mean gives you the best estimate, in the mean square error sense, of $\boldsymbol{x}_1$ considering that information.

9.1.5 Example (2): MVN with Missing Values

Even though part of the data is missing, the assumption is that the missing data also comes from the same multivariate Gaussian distribution as the observed data.

If we would know the model parameters ($\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$), we can use this result to determine the distribution of missing data. E.g., for our 2D example with $\boldsymbol{\mu} = \begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}$ and $\boldsymbol{\Sigma} = \begin{bmatrix}\sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22}\end{bmatrix}$, we obtain:

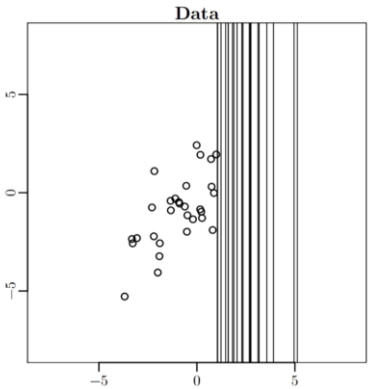
$$p(x_2|x_1) = N(x_2 \mid \mu_2 + \sigma_{21}\sigma_{11}^{-1}(x_1 - \mu_1),\ \sigma_{22} - \sigma_{21}\sigma_{11}^{-1}\sigma_{12})$$



In practice, we often do not know the true parameters of the distribution, especially in the presence of missing data. The EM algorithm helps to estimate these parameters. Once we have an estimate of these parameters, we can use them to make inferences about the missing data, as shown above.

9.2 EM Algorithm: Intuition

Given the model parameters, we could determine the distribution of the missing data. But we do not know the model parameters. Given the distribution of missing data, we could determine the model parameters. But we do not know the distribution of missing data.



EM Algorithm

1. Start with initial parameter estimate (e.g., random)
2. Repeat until *stopping criterion* is satisfied:
   a. *E-Step*: Determine distribution of missing data based on current parameter estimate.
   b. *M-Step*: Estimate parameters based on distribution of missing data.

9.3 Notation & Terminology

We split all available data into:

- Observed data $\boldsymbol{x}$
- Missing data $\boldsymbol{z}$
- Complete data $\boldsymbol{d} = (\boldsymbol{x}, \boldsymbol{z})$

Generally, $\boldsymbol{x}$, $\boldsymbol{z}$ and $\boldsymbol{d}$ are sets of variables.

E.g., for iid. data with missing values for each data point:

- Observed values for each example: $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^N$
- Missing values for each example: $\boldsymbol{z} = \{\boldsymbol{z}_i\}_{i=1}^N$
- $\boldsymbol{d} = \{\boldsymbol{d}_i\}_{i=1}^N$ with $\boldsymbol{d}_i = (\boldsymbol{x}_i, \boldsymbol{z}_i)$

The set of observed $\boldsymbol{x}_i$ and missing values $\boldsymbol{z}_i$ may be different for each example. Meaning, on day 1 ($i = 1$), we might have observed temperature and humidity ($\boldsymbol{x}_1$), but we're missing AQI and solar radiation ($\boldsymbol{z}_1$). On Day 2 ($i = 2$), perhaps the humidity sensor failed, and now we're missing humidity and solar radiation ($\boldsymbol{z}_2$), but we have temperature and AQI ($\boldsymbol{x}_2$). And so on.

Explanation

Observed data $\boldsymbol{x}$ represents the data we have successfully measured. For instance, in a given time period, we may have accurate readings for temperature and humidity. For our N data points (e.g., N days of measurement), this could be expressed as $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^{N}$, where each $\boldsymbol{x}_i$ is the set of observed measurements on day i ($\boldsymbol{x}_i$ could include temperature and humidity for the i-th day).

Missing Data $\boldsymbol{z}$ is the data that we were unable to measure. Continuing the example, for the same time period, we might be missing the AQI and solar radiation measurements. The missing data is represented as $\boldsymbol{z} = \{\boldsymbol{z}_i\}_{i=1}^{N}$, where each $\boldsymbol{z}_i$ represents the missing environmental measurements on day i (e.g., $\boldsymbol{z}_i$ could include AQI and solar radiation for the i-th day).

🧠 We focus on determining the ML estimate: $\widehat{\boldsymbol{\theta}}_{MLE} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$. Once we know $\boldsymbol{\theta}$, this distribution / generative model describes the relationship between $\boldsymbol{x}$ and $\boldsymbol{z}$.

But, as we do not know the values of $\boldsymbol{z}$, we cannot compute likelihood $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$. The EM algorithm addresses this problem by iteratively estimating the missing values . Note, that in the context of Gaussian distributions, $\boldsymbol{\theta}$ typically includes the means $\boldsymbol{\mu}$ and covariances $\boldsymbol{\Sigma}$.

💡 The EM algorithm is also applicable for MAP estimation $\widehat{\boldsymbol{\theta}}_{MAP} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})$; and for discriminative models of form $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x})$ → if explanatory variables $\boldsymbol{x}$ are fully observed.

9.4 EM in Detail

9.4.1 Observed-Data Log-Likelihood

In the context of determining the ML estimate: $\widehat{\boldsymbol{\theta}}_{MLE} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$, we define our maximization target, the *complete-data likelihood*, as:

$$\ell_c(\boldsymbol{\theta}) \overset{\text{def}}{=} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = \ell_c(\theta|x, z)$$

❗ This involves the *actual/true* joint probability of both observed data $\boldsymbol{x}$ and latent data $\boldsymbol{z}$, given the parameters $\boldsymbol{\theta}$. Because $\boldsymbol{z}$ is unknown, we cannot directly compute this likelihood.

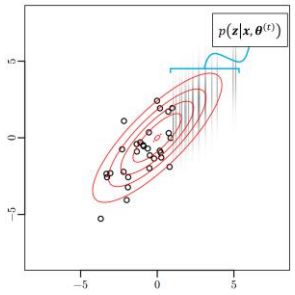
📄 Instead, we maximize the *observed-data log-likelihood* by marginalizing/integrating out $\boldsymbol{z}$:

$$\ell_o(\boldsymbol{\theta}) \overset{\text{def}}{=} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \log \int_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z} = \ell(\theta|x)$$
$$= \log \int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z}|\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z}$$

When we integrate over the latent variable $\boldsymbol{z}$, we are essentially considering all possible values of $\boldsymbol{z}$ and their contribution to the probability of observing $\boldsymbol{x}$, given the parameter $\boldsymbol{\theta}$.

To optimize this, we may use a gradient-based optimizer, but the integral can be tricky to evaluate. EM is often much simpler, but not always faster.

9.4.2 Missing-Data Distribution



Given the current parameter estimate $\boldsymbol{\theta}^{(t)}$ we can determine the distribution of the missing data $\boldsymbol{z}$:

$$p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)}) = \frac{p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta}^{(t)})}{p(\boldsymbol{x}|\boldsymbol{\theta}^{(t)})}$$

This a step where, based on what is currently known $(\boldsymbol{x},\boldsymbol{\theta}^{(t)})$, we make an educated guess about the distribution of the missing data.

We can calculate $p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta}^{(t)})$ as we assume that our model with parameters $\boldsymbol{\theta}$ defines the distribution of both the observed and latent variables. The "model" is a statistical model that we are using to describe our data. It generally involves a set of assumptions about how data is generated or behaves. For example, an MVN.

9.4.3 Expectation-Step

In the E-step, the EM algorithm "computes" the auxiliary *Q function*:

$$\begin{aligned} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &\overset{\text{def}}{=} E_{\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)}}[\ell_c(\boldsymbol{\theta})] \\ &= \int_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)}) \log p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z} \end{aligned}$$

- the *current* estimate $\boldsymbol{\theta}^{(t)}$ determines the *missing-data distribution* $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$
- "new" parameters $\boldsymbol{\theta}$ determine the *complete-data log-likelihood* $p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})$

The term $E_{\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)}}$ results in $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$, while $\log p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})$ comes from $\ell_c(\boldsymbol{\theta})$. The Q-function corresponds to *expected complete-data log-likelihood* for a *fixed* missing-data distribution.

The Q function helps us to answer this question: "Given our current guess about the model parameters $\boldsymbol{\theta}^{(t)}$, if we were to fill in the missing data in all possible ways $\int_{\boldsymbol{z}}$ it could reasonably be filled in, how likely would the resulting, complete datasets be?"

Interpretation and Context

During the E-step, we use our current estimates $\boldsymbol{\theta}^{(t)}$ to estimate the distribution of the missing data $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$. Then, we would use this estimated distribution $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$ to compute the Q function $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$, which tells us what the *expected complete-data log-likelihood* is for a particular $\boldsymbol{\theta}$.

For our estimate of the missing-data distribution $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$, Q tells us what the expected log-likelihood of observing the entire dataset (both observed and missing data) is. This expectation is wrt. $p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$, meaning we're considering all the ways that the missing data could have been realized and how likely each of those ways is according to our current model.

Q gives us an expectation of what the log-likelihood for the complete data would be if we knew the missing data, under the current model parameters $\boldsymbol{\theta}^{(t)}$.

This Q function then guides the M-step, where we adjust our parameters to maximize this expected log-likelihood. This process results in a new set of parameters $\boldsymbol{\theta}^{(t+1)}$.

9.4.4 Maximization-Step

In the M-step, the EM algorithm computes new parameter values by maximizing the Q function:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$$

After the M-step, the estimated missing-data distribution $p\left(\boldsymbol{z}\middle|\boldsymbol{x},\boldsymbol{\theta}^{(t)}\right)$ changes since the parameter changed $\boldsymbol{\theta}^{(t)} \rightarrow \boldsymbol{\theta}^{(t+1)}$. By iterating through these E and M steps, the EM algorithm converges to a set of parameters that best explain both the observed and missing data.

9.4.5 Monotonic Convergence Theorem

Recall, the observed-data log-likelihood $\ell_o(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \log p(\boldsymbol{x}|\boldsymbol{\theta})$. One can show that the EM algorithm monotonically increases the observed-data log-likelihood, meaning that with each iteration, the log-likelihood of the observed data under the current parameter estimates does not decrease. In simpler terms, the algorithm is designed to either improve or maintain the fit of the model to the observed data at each step.

Theorem

The EM algorithm monotonically increases the observed-data log-likelihood in that

$$\ell_o(\boldsymbol{\theta}^{(t+1)}) \geq \ell_o(\boldsymbol{\theta}^{(t)})$$

and

$$\ell_o(\boldsymbol{\theta}^{(t+1)}) > \ell_o(\boldsymbol{\theta}^{(t)}) \qquad \text{if } \boldsymbol{\theta}^{(t)} \text{ is not a stationary point of } \ell_o.$$

Consequently, our parameter estimates $\boldsymbol{\theta}^{(t)}$ continuously improve over time. At some point the algorithm will converge to a stationary point. This could be a local optimum, a saddle point (where the function has a slope of zero but is not a local optimum), or part of a flat region. The distinction is important, as reaching a stationary point does not always mean that the best possible solution (global optimum) has been found.

9.4.6 Proof

Why does above theorem hold? We're not maximizing $\ell_o(\boldsymbol{\theta})$ but instead the *expected complete-data log-likelihood* $Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$ under a "wrong" missing-data distribution. One can show that the observed-data log-likelihood is:

$$\ell_o(\boldsymbol{\theta}) = Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right) + H\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$$

The cross entropy $H\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$ tells us how different the *actual* missing-data distribution is to the one we use in the E-step. Expressing the improvement in terms of Q and H, shows:

$$\begin{aligned}\ell_o\left(\boldsymbol{\theta}^{(t+1)}\right) - \ell_o\left(\boldsymbol{\theta}^{(t)}\right) &= Q\left(\boldsymbol{\theta}^{(t+1)}\middle|\boldsymbol{\theta}^{(t)}\right) + H\left(\boldsymbol{\theta}^{(t+1)}\middle|\boldsymbol{\theta}^{(t)}\right) - \left(Q\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right) + H\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right)\right) \\ &= Q\left(\boldsymbol{\theta}^{(t+1)}\middle|\boldsymbol{\theta}^{(t)}\right) - Q\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right) + H\left(\boldsymbol{\theta}^{(t+1)}\middle|\boldsymbol{\theta}^{(t)}\right) - H\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right)\end{aligned}$$

1. The M step proves that $Q\left(\boldsymbol{\theta}^{(t+1)}\middle|\boldsymbol{\theta}^{(t)}\right) - Q\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right) \geq 0$
2. The cross-entropy states $H\left(\boldsymbol{\theta}^{(t)}\middle|\boldsymbol{\theta}^{(t)}\right) = H\left(\boldsymbol{\theta}^{(t)}\right)$ and $H\left(\boldsymbol{\theta}^{(t)}\right) \leq H\left(\boldsymbol{q}\middle|\boldsymbol{\theta}^{(t)}\right)$ for any $\boldsymbol{q}$.

Therefore $\ell_o\left(\boldsymbol{\theta}^{(t+1)}\right) - \ell_o\left(\boldsymbol{\theta}^{(t)}\right) \geq 0$ holds.

9.5 Discussion

Convergence to local maximum or stationary point of $\ell_o(\boldsymbol{\theta})$ is guaranteed

- We may not find global maximum
- Should run EM multiple times with different initializations (e.g., random restarts)

In practice, use a stopping criterion such as:

- small improvement in observed-data log-likelihood or Q function
- small change of parameters
- limit of total number of iterations or time budget

Many variants exist: better convergence, support for complex models, parallelizability

- Generalized EM (don't maximize but improve Q)
- Stochastic or batch EM (use subset of examples)
- Monte-Carlo EM (approximate E step)
- Hard EM (impute missing values)

Depending on model, E and M steps may still be complicated.

9.6 Mixture Models

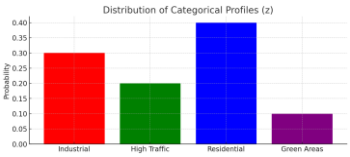
A *mixture model* (MM) is a LVM with a *single categorical* latent variable z per example $\boldsymbol{x}$:

$$z \sim \text{Cat}(\boldsymbol{\pi}) \qquad \boldsymbol{x} \sim p(\boldsymbol{x}|z, \boldsymbol{\theta})$$

where $p(\boldsymbol{x}|z, \boldsymbol{\theta})$ is some distribution. As it depends on z, the observed data $\boldsymbol{x}$ may have a different distribution for each of the individual categories $\boldsymbol{\pi}$.

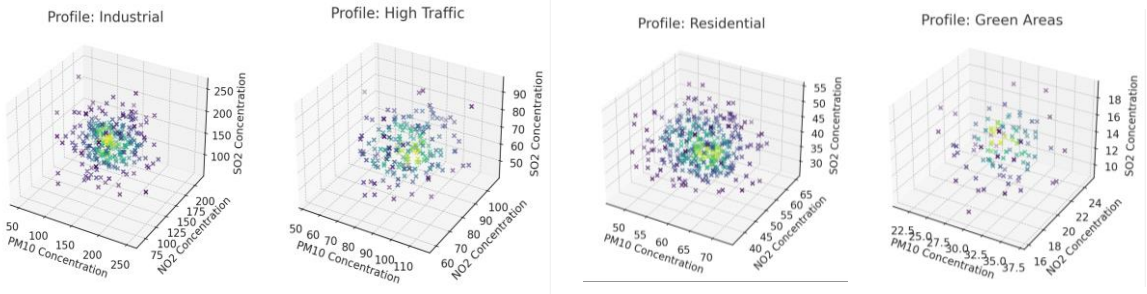
Example

Imagine a city-wide air quality monitoring system that collects data $\boldsymbol{x}$ on various pollutants like particulate matter `PM10`, nitrogen dioxide `NO2`, and sulfur dioxide `SO2`. The data is collected from multiple sensors spread across different parts of the city.



Each observation $\mathbf{x}$ (e.g., a set of pollutant levels like `PM10`, `NO2`, etc., at a given time and location) is analyzed in the context of profiles $z \sim \text{Cat}(\boldsymbol{\pi})$.

These categorical profiles represent different environmental conditions: 'industrial', 'high traffic', 'residential', and 'green areas'. Hence $\boldsymbol{\pi} = [0.3, 0.2, 0.4, 0.1]$. While $z \sim \text{Cat}(\boldsymbol{\pi})$ is fixed, the distribution over the observations $\boldsymbol{x} \sim p(\boldsymbol{x}|z, \boldsymbol{\theta})$ may look different based on the value of z.



Base Distributions $p(\boldsymbol{x}|z = k, \boldsymbol{\theta})$ with $z \sim Cat(\boldsymbol{\pi} = [0.3, 0.2, 0.4, 0.1])$.

The *observed-data likelihood* is obtained by marginalizing out z:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x}|\boldsymbol{\theta})$$

where $\sum_k \pi_k = 1$ as well as $0 \leq \pi_k \leq 1$. In the formula π_k are called the *mixing weights*, where:

$$\pi_k = p(z = k|\boldsymbol{\theta})$$

Data $\boldsymbol{x}$ is modeled as a mixture of K *base distribution* p_k (mixture components), where the outcome of the categorical is fixed:

$$p_k(\boldsymbol{x}|\boldsymbol{\theta}) \stackrel{\text{def}}{=} p(\boldsymbol{x}|z = k, \boldsymbol{\theta})$$

with which we can also generate data → *generative model*.

Intuition

This means that each data point is generated from one base distribution, which one is described by latent variable z. In practice, we may not know what this latent variable has actually been but try to reason about it.

9.6.1 GMM and K-Means

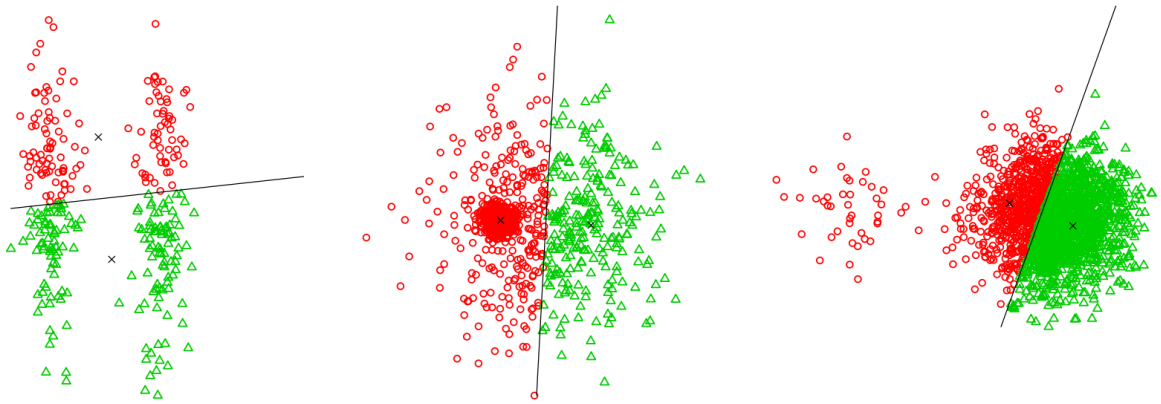
MMs and in particular GMMs are closely related to K-Means.

K-Means Clustering makes a set of assumptions that do not necessarily hold in practice:

- Clusters are spherical
- Clusters are non-overlapping
- Clusters have similar sizes

The examples on the right illustrate why these assumptions may be problematic, even with optimal K-Means clustering.

All of these three problems can be solved by (Gaussian) Mixture Models.

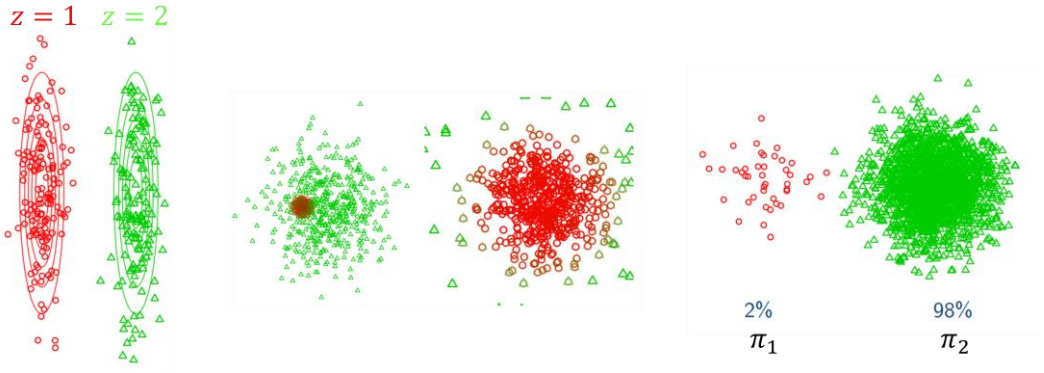


9.6.2 Gaussian Mixture Models (GMMs)

A GMM is a generative type of mixture model in which the base distributions p_k are MVNs.

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

although p_k are MVNs, the full distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ is generally not Gaussian, as it, for example, may have multiple modes. A GMM is described as having K components, meaning takes on values in the set $\{1,2,\dots,K\}$, representing each individual Gaussian distribution within the mixture.



These models can be viewed as a *soft clustering variant* to K-Means. Meaning we do not assign each data point to a cluster, but instead to multiple clusters. For each point we therefore calculate probability $p(z = k|\boldsymbol{x}, \boldsymbol{\theta})$ of belonging to cluster $K \rightarrow$ *cluster membership probability*.

- z corresponds to the cluster identifier (as in K-Means)
- π_k models the size of the cluster, since $\pi_k = p(z = k|\boldsymbol{\pi})$
- individual cluster shape and size can be read of the GMM parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ after fitting the data
- all above also applies for non-gaussian MMs

9.7 EM for GMMs

Consider the following setting. We have N data points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ that are sampled independently from a data distribution. The model parameters are $\boldsymbol{\theta} = \bigcup_k \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}$, where k refers to the index of a particular Gaussian component within the mixture model.

9.7.1 Likelihood & Responsibility

The *likelihood* $f_k(\boldsymbol{x}_i)$ of data point $\boldsymbol{x}_i$ being in the k-th cluster is if we knew $\boldsymbol{\theta}$:

$$f_k(\boldsymbol{x}_i) \stackrel{\text{def}}{=} p(\boldsymbol{x}_i|z_i = k, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Remember, latent variable z_i indicates cluster membership and $k \in \{1,2,\dots K\}$ is a cluster.

With that, we can compute the *complete-data likelihood* of $(\boldsymbol{x}_i, z_i)$:

$$\begin{aligned} p(\boldsymbol{x}_i, z_i = k|, \boldsymbol{\theta}) &= p(z_i = k|\boldsymbol{\theta}) \, p(\boldsymbol{x}_i|z_i = k, \boldsymbol{\theta}) \\ &= \pi_k \, f_k(\boldsymbol{x}_i) \end{aligned}$$

If we knew a parameter choice $\boldsymbol{\theta}$ we could tell how likely this is that $\boldsymbol{x}_i$ comes from a particular cluster z_i. This w_{ik} is the *cluster-membership probability* of $\boldsymbol{x}_i$ being in cluster k:

$$w_{ik} \stackrel{\text{def}}{=} p(z_i = k|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{p(\boldsymbol{x}_i, z_i = k|\boldsymbol{\theta})}{p(\boldsymbol{x}_i|\boldsymbol{\theta})} = \frac{\pi_k \, f_k(\boldsymbol{x}_i)}{\sum_{k'} \pi_{k'} \, f_{k'}(\boldsymbol{x}_i)}$$

where $p(\boldsymbol{x}_i|\boldsymbol{\theta})$ is marginal likelihood of the observed data $\boldsymbol{x}_i$. We think of w_{ik} as the weight of cluster k for example i. We also refer to w_{ik} as the *responsibility* of the k-th mixture component for data point $\boldsymbol{x}_i$.

9.7.2 E-Step

We compute *cluster membership probabilities* $\left\{w_{ik}^{(t)}\right\}$ based on current estimate $\boldsymbol{\theta}^{(t)}$ by:

$$w_{ik}^{(t)} \leftarrow p\left(z_i = k\middle|\boldsymbol{x}_i, \boldsymbol{\theta}^{(t)}\right)$$

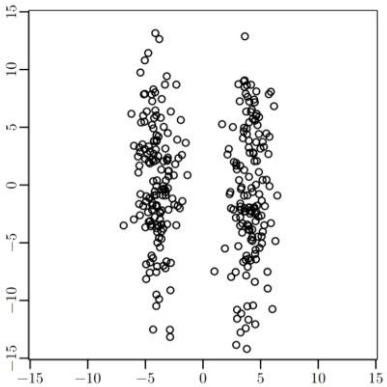
The cluster membership probabilities will change after every M-step, because $\boldsymbol{\theta}^{(t)}$ will change. They essentially tell us the distribution of the missing data under the parameter estimate $\boldsymbol{\theta}^{(t)}$.

Let's gather all individual clusters $\boldsymbol{z} = (z_1 \quad \cdots \quad z_N)^\top$, then the Q function is:

$$\begin{aligned} Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right) &= E_{\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}^{(t)}}[\log p(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{\theta})] \\ &= \sum_i \sum_k w_{ik}^{(t)} \log[\pi_k \, f_k(\boldsymbol{x}_i)] \\ &= \sum_i \sum_k w_{ik}^{(t)} \log \pi_k + \sum_i \sum_k w_{ik}^{(t)} \log f_k(\boldsymbol{x}_i) \end{aligned}$$

The Q-function uses old cluster memberships probabilities $w_{ik}^{(t)} \rightarrow$ which we compute in the E-step. The terms π_k and $f_k(\boldsymbol{x}_i)$ do not depend on the old parameterization $\boldsymbol{\theta}^{(t)}$ but do on $\boldsymbol{\theta}$.

9.7.3 M-Step



The M step finds $\boldsymbol{\theta}$ that maximizes $Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\theta}{\operatorname{argmax}} \, Q\left(\boldsymbol{\theta}\middle|\boldsymbol{\theta}^{(t)}\right)$$

for GMMs this can be done in *closed form*, such that:

$$\pi_k^{(t+1)} \leftarrow \frac{1}{N} \sum_i w_{ik}^{(t)}$$

$$\boldsymbol{\mu}_k^{(t+1)} \leftarrow \frac{1}{\sum_i w_{ik}^{(t)}} \sum_i w_{ik}^{(t)} \, \boldsymbol{x}_i$$

$$\boldsymbol{\Sigma}_k^{(t+1)} \leftarrow \frac{1}{\sum_i w_{ik}^{(t)}} \sum_i w_{ik}^{(t)} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(t+1)}\right)\left(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(t+1)}\right)^\top$$

These are similar to ML estimates for MVNs, but data points are now weighted by cluster membership probabilities $w_{ik}^{(t)}$.

9.7.4 GMMs and K-Means

💡 *Lloyd's algorithm* is popular method for performing clustering with *K-means objective*:

1. Start with initial centroids, e.g., random, farthest point clustering / K-means++
2. Assign each data point to its closest centroid (*in GMMs → membership-probs.*)
3. Set each centroid to mean of the data points assigned to it (*in GMMs → weighted by* $w_{ik}^{(t)}$)
4. Repeat steps 2 & 3 until some stopping criterion is met

💡 Equivalent to certain GMM with *hard* EM, i.e.:

- use (a priori) equally likely clusters ($\pi_k = \frac{1}{K}$)
- use spherical Gaussians ($\boldsymbol{\Sigma}_k = \boldsymbol{I}$)
- use hard cluster assignments in E step ($w_{ik} = 1$ for most likely cluster; zero for other)
- only centroids $\boldsymbol{\mu}_k$ need to be computed in M step

9.8 GMM Discussion

💡 EM for GMMs determines cluster membership probabilities $w_{ik}^{(t)}$, but no hard clustering. At the very end, we may assign each point to its most likely cluster.

❗ GMMs can have large number of parameters when data high-dimensional (D)

- Covariance matrices have $O(D^2)$ parameters → $O(KD^2)$ parameters in total
- Can be reduced, for example, by using diagonal covariances matrices or by combination with factor analysis
- Overfitting (e.g., fits singularities) may arise → use MAP (prior on parameters)

📄 Discriminative variant of MMs is called *mixture of experts:*

$$p(y|x,\theta) = \sum_k p(z = k|\boldsymbol{x},\boldsymbol{\theta})\, p(y|\boldsymbol{x}, z = k, \boldsymbol{\theta})$$

- Each component model $p(y|\boldsymbol{x}, z = k, \boldsymbol{\theta})$ is considered an *expert*
- *Gating function* $p(z_i = k|\boldsymbol{x}_i, \boldsymbol{\theta})$ decides which expert to use.

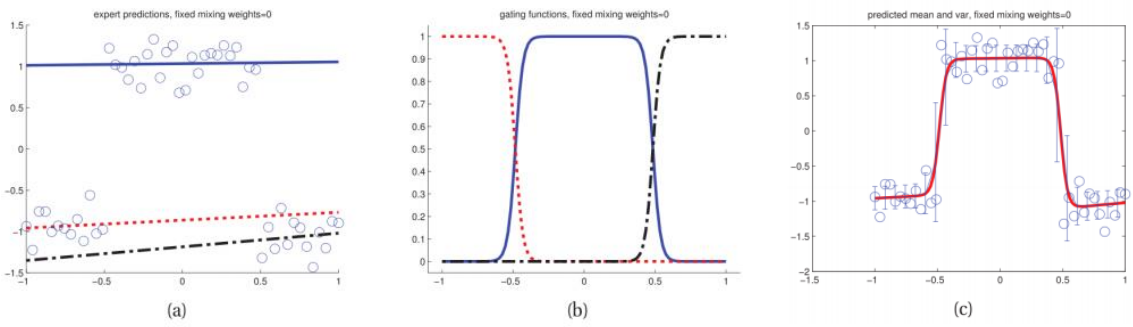


Figure 11.6 (a) Some data fit with three separate regression lines. (b) Gating functions for three different "experts". (c) The conditionally weighted average of the three expert predictions. Figure generated by `mixexpDemo`.

10 Kernels & Vector Machines

So far, we represented model inputs by a fixed-size feature vector $\boldsymbol{x}_i \in \mathbb{R}^D$. For some inputs, its not clear how to do this, e.g., text documents, protein sequences or graphs. The Kernel approach uses similarity instead of features. For this they use a *kernel function* $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ that can be interpreted as similarity between objects $\boldsymbol{x}$ and $\boldsymbol{x}'$. In general, this is also useful when data *can* be represented as fixed-size feature vectors. The *Kernel Trick* modifies existing learning algorithms (LR, SVM, KNN, …) to work solely with kernel functions (κ-values).

10.1 Kernel Functions

A *kernel function* is a real-valued function $\kappa(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}$ with two inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$.

- Typically, we use symmetric kernels, where $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \kappa(\boldsymbol{x}', \boldsymbol{x})$ holds.
- Sometimes, we use non-negative kernels, where $\kappa(\boldsymbol{x}, \boldsymbol{x}') \geq 0$.

If both properties hold, then we may interpret it as a measure of similarity.

Sometimes, the term *kernel* is used to explicitly refer to *Mercer kernels*. Their kernel function $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ can be expressed as an inner/dot product on "transformed" inputs. Kernelization of learning algorithms using the kernel trick requires $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ to be a mercer kernel.

10.1.1 Gaussian Kernels

The Gaussian Kernel is given by:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right) \propto \mathcal{N}(\boldsymbol{x}|\boldsymbol{x}', \boldsymbol{\Sigma})$$

it takes values in [0,1] and in particular $\kappa(\boldsymbol{x}, \boldsymbol{x}) = 1$. The value of the kernel decreases exponentially in squared Mahalanobis distance.

A special case for diagonal covariance, is the *ARD kernel* or *squared exponential kernel*:

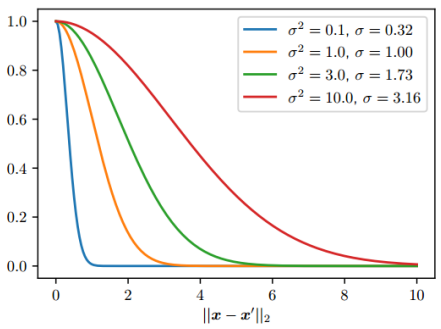
$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{1}{\sigma_j^2}(x_j - x_j')^2\right)$$

where σ_j^2 can be interpreted as a *length scale* of dimension j. For very small σ_j^2, we scale dimension j up. When $\sigma_j^2 \to \infty$, dimension j is ignored.

Another special case is the *isotropic Gaussian kernel*:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2\right)$$

where σ^2 is known as *bandwidth* of dimension j. It's a measure of how far apart two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ can be in the input space for the kernel to consider them as similar. A larger bandwidth (σ^2) means the kernel considers points that are farther apart as similar.



10.1.2 Other Kernels

- Linear kernel: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$
- Linear kernel with basis function expansion: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x}^\top)\, \phi(\boldsymbol{x}')$
- Polynomial kernel: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\gamma \boldsymbol{x}^\top \boldsymbol{x}' + r)^M$ with $\gamma, r \geq 0$
- Cosine similarity kernel: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \frac{\boldsymbol{x}^\top \boldsymbol{x}'}{\|\boldsymbol{x}\| \|\boldsymbol{x}'\|}$

10.1.3 Mercer Kernel

A kernel is a *mercer kernel* if the *Gram matrix* $\boldsymbol{K}$ (kernel matrix) is positive (≥ 0) semi-definite (angle between original & new vector $\leq 90°$) for all inputs $\{\boldsymbol{x}_i\}_{i=1}^N$:

$$\boldsymbol{K} = \begin{pmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

The *Gram matrix* $\boldsymbol{K}$ contains the value of the kernel function for each pair of inputs $(\boldsymbol{x}, \boldsymbol{x}')$.

Above implies that there exists an eigendecomposition and all $\lambda_i \geq 0$:

$$K = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top = \left(\boldsymbol{Q}\boldsymbol{\Lambda}^{\frac{1}{2}}\right)\left(\boldsymbol{Q}\boldsymbol{\Lambda}^{\frac{1}{2}}\right)^\top$$

If we set $\phi(\boldsymbol{x}_i) = \left[\boldsymbol{Q}\boldsymbol{\Lambda}^{\frac{1}{2}}\right]_{i:}^\top$, then we can compute $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^\top\, \phi(\boldsymbol{x}_j)$. So instead of computing the kernel matrix, we could also transform each of the inputs separately using ϕ and then take the matrix product.

10.1.4 Mercer's theorem

κ is positive semidefinite iff. there exists a "feature map" $\phi_k: \mathcal{X} \to \mathbb{R}^D$ s.t. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top\, \phi(\boldsymbol{x}')$ for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$. D may be a finite (e.g., polynomial kernel) or infinite (e.g., gaussian kernel).

Example

Consider the polynomial kernel : $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^2$. For each $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^2$ we have:

$$(1 + \boldsymbol{x}^\top \boldsymbol{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2 = 1 + 2x_1 x_1' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2'$$

which can be written as $\phi(\boldsymbol{x})^\top\, \phi(\boldsymbol{x}')$ with:

$$\phi(\boldsymbol{x}) = \begin{pmatrix} 1 & \sqrt{2} * x_1 & \sqrt{2} * x_2 & x_1^2 & x_2^2 & \sqrt{2} * x_1 x_2 \end{pmatrix}^\top \in \mathbb{R}^6$$

Using this kernel is equivalent to working in the 6-dimensional feature space.

10.1.5 Discussion

- Mercer kernels define an implicit feature map. If $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ is a mercer kernel, there exists a function ϕ s.t. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top\, \phi(\boldsymbol{x}')$. Thus κ computes inner product of mapped features. But κ is not an inner product on $\mathcal{X}$ (e.g., $\kappa(a\boldsymbol{x}, \boldsymbol{x}')$ may not equal to $a\kappa(\boldsymbol{x}, \boldsymbol{x}')$).
- If $\kappa(\boldsymbol{x}, \boldsymbol{x}') = c$, it is a Mercer kernel for any $c \in \mathbb{R}_{\geq 0}$. Sums/Prdoucts of Mercer Kernels are also Mercer kernels. As is mult. by non-neg. sca lar and exponentiation of kernel.

10.2 Kernel Machines

A *kernel machine* is a generalized linear model which uses a *kernelized feature vector* as input:

$$\phi_{km}(\boldsymbol{x}) = \big(\kappa(\boldsymbol{x}, \boldsymbol{\mu}_1) \ \ldots \ \kappa(\boldsymbol{x}, \boldsymbol{\mu}_K)\big)^\top$$

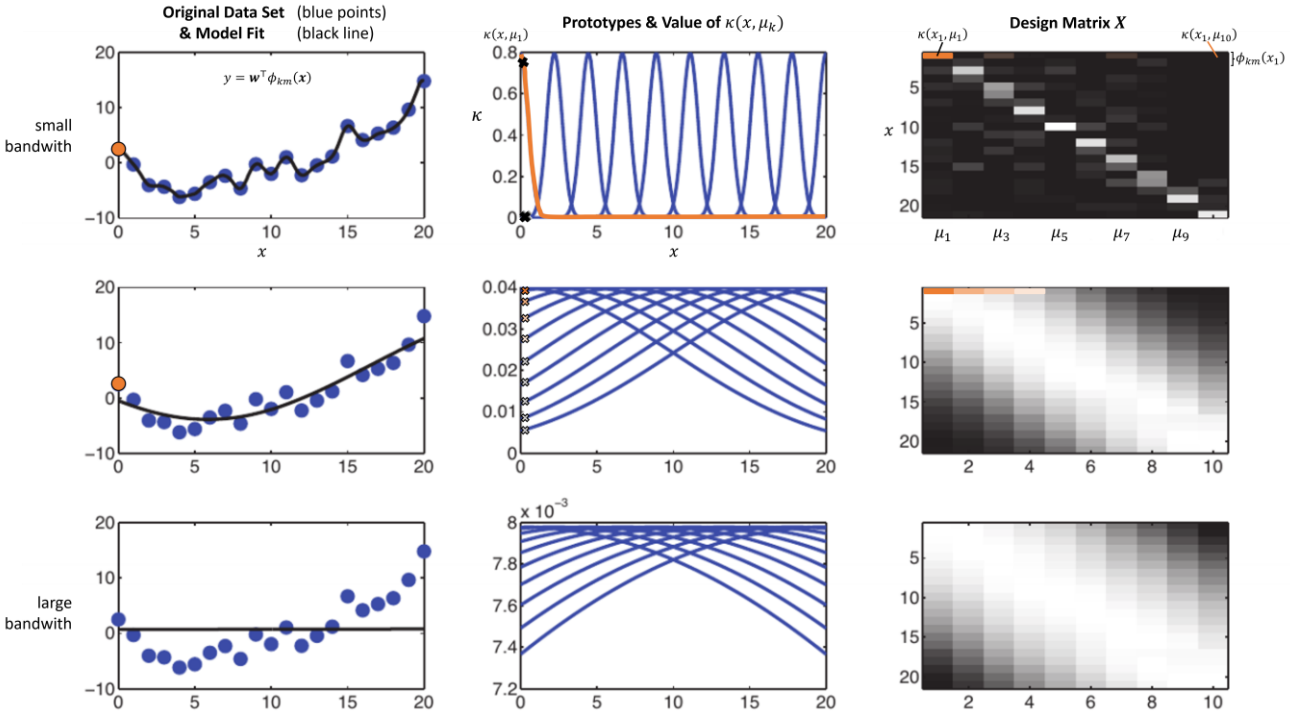
The $\boldsymbol{\mu}_K \in \mathcal{X}$ are a set of K *centroids/prototypes;* they look like data points. Then, a concrete example is constructed by the kernelized feature vector that consists of the values of the kernel functions $\kappa(\boldsymbol{x}, \boldsymbol{\mu}_K)$ between the original data point $\boldsymbol{x}$ and the centroid $\boldsymbol{\mu}_K$. If κ is an RBF kernel, the resulting kernel machine is called an *RBF network*.

The linear predictor then is specified by the weight vector $\boldsymbol{w} \in \mathbb{R}^K$:

$$\eta(\boldsymbol{x}) = \boldsymbol{w}^\top \phi_{km}(\boldsymbol{x}) = \sum_k w_k \, \kappa(\boldsymbol{x}, \boldsymbol{\mu}_k)$$

Consider a centroid $\boldsymbol{\mu}_K$ and suppose κ measures similarity. The contribution of a particular centroid $\boldsymbol{\mu}_K$ to our $\boldsymbol{x}$ is larger the more similar $\boldsymbol{x}$ is to $\boldsymbol{\mu}_K$. Further, the contribution is larger, the higher the centroids weight $|w_k|$ is. The contribution can be positive $w_k > 0$ or negative $w_k < 0$.

Examples



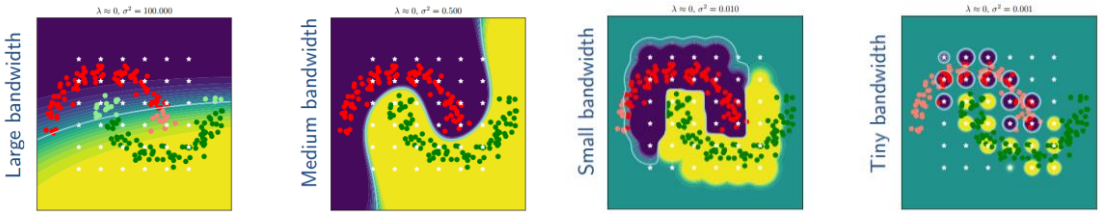
In the design matrix $\boldsymbol{X}$, each row represents the feature vector $\phi_{km}(x)$ for a particular data point x after being transformed by the RBF kernel.

Each curve $\kappa(x, \mu_k)$ in the middle column corresponds to one RBF kernel centered at a different prototype μ_k, and it shows how the influence of that prototype extends over the input space x.

The final fit then is a linear combination of the ten kernel functions $\kappa(x, \mu_k)$, where their respective weights $\boldsymbol{w}$ is what we fit in the linear regression model.

The plot illustrates the effect of bandwidth σ^2 as for too low, the resulting model overfits, as no data point is similar to more than one prototype. While for too high σ^2, every data point is similar, and we learn nothing.

The stars in the plot for **Logistic Regression** correspond to the centroids/prototypes $\boldsymbol{\mu}_K$.



Choosing Centroids

- On low-dimensional inputs, we may use a grid (as above), but this breaks down on high dimensionality.
- If $\boldsymbol{\mu}_k \in \mathbb{R}^D$ (real vector) we may use numerical optimization. But kernels are most useful for structured input spaces.
- Cluster data and use cluster centers? These centers are generally dense regions in input space but may not be useful for predicting outputs.

10.3 Vector Machines

📄 Vector Machines use all of the data points as centroids

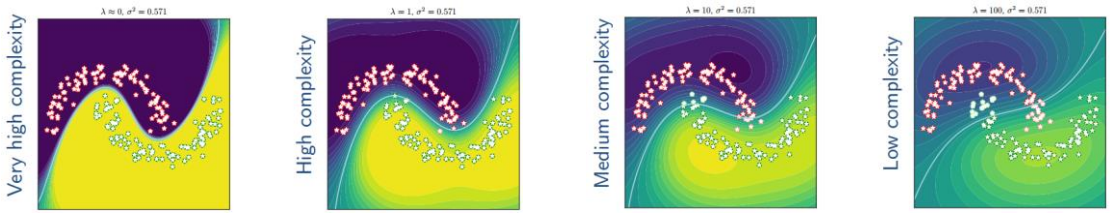
$$\eta(\boldsymbol{x}) = \boldsymbol{\alpha}^\top \phi_{vm}(\boldsymbol{x}) = \sum_i \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$$

The model is parameterized by a weight α_i for each data point. To predict, the model needs access to all data points $\boldsymbol{x}_i$ (assuming all $\alpha_i \neq 0$). We have N weights, N data points, the cost to predict is $O(N) \rightarrow$ expensive for large datasets.

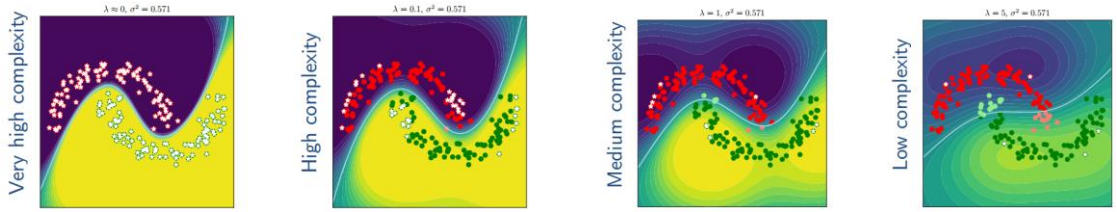
💡 Various approaches exist to select a subset of the data points as centroids. These methods "encourage" choices of $\alpha_i = 0 \rightarrow$ corresponding data points ignored.

- Can use a sparsity-promoting prior, e.g., ℓ_1 regularization (L1VM)
- Can use customized loss function → support vector machines (SVM)
- Pro: lower computational cost, lower space consumption
- Cons: also affects model complexity (and thus generalization performance)

LR with RBF and L2 regularization: all data points are centroids/prototypes.



Choosing L1 regularization results in fewer centroids that matter (stars have $\alpha_i \neq 0$).



💡 Regularization: L1 (Lasso) vs. L2 (Ridge)

L1 regularization adds the absolute value of the coefficients (ℓ_1 norm) as penalty to the loss function. This promotes sparsity in the model coefficients, meaning that it encourages the model to set a larger number of weight coefficients (α_i) to zero.

L2 regularization adds the squared magnitude of the coefficients as penalty to the loss function. Unlike L1, L2 does not encourage sparsity in the weights; instead, it tends to shrink the values of the weights, but does not set them to zero. This is because the penalty becomes progressively larger as the weight values move away from zero.

This distinction makes L1 more useful for situations where feature selection is important, while L2 is often used for models where all features are assumed to be relevant, but overfitting needs to be controlled.

10.4 The Kernel Trick

📄 Rather than working with kernelized features as before, the kernel trick modifies the learning algorithm directly. The key idea is to express the learning algorithm in a way that accesses the data *only* in terms of inner products of form $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$ and then replace all these inner products by calls to the kernel function $\kappa(\boldsymbol{x}, \boldsymbol{x}')$.

This works well if the kernel is a *mercer kernel*. Then the approach is equivalent to working on the implicit feature representation ϕ_k corresponding to κ.

❗ Among other things, this changes the way regularization works. While we were previously learning weights $\boldsymbol{w}$ for each of the kernelized features $\kappa(\boldsymbol{x}, \boldsymbol{x}_N)$, were now leaning the weights in the ϕ_k representation, resulting in a different set of weights.

Example: Kernelized KNN

Recall, that in KNN we make predictions by applying neighborhood statistics $N_K(\boldsymbol{x}, \mathcal{D})$:

$$p(y = c|\boldsymbol{x}, \mathcal{D}, K) = \frac{1}{K} \sum_{i \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_i = c)$$

where $\mathbb{I}(e)$ is the indicator function: $\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{else} \end{cases}$

In KNN were accessing data only in $N_K(\boldsymbol{x}, \mathcal{D})$, here we need a distance measure, which can be sq. Euclidian distance $d(\boldsymbol{x}_i, \boldsymbol{x}) = \|\boldsymbol{x}_i - \boldsymbol{x}\|_2^2$ which we can express as inner products by:

$$\|\boldsymbol{x}_i - \boldsymbol{x}\|_2^2 = \langle \boldsymbol{x}_i - \boldsymbol{x}, \boldsymbol{x}_i - \boldsymbol{x} \rangle = \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle + \langle \boldsymbol{x}, \boldsymbol{x} \rangle - 2\langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$$

After replacing inner products by kernel function calls, we obtain:

$$d(\boldsymbol{x}_i, \boldsymbol{x}) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) + \kappa(\boldsymbol{x}, \boldsymbol{x}) - 2\kappa(\boldsymbol{x}_i, \boldsymbol{x})$$

With a Gaussian RBF this even simplifies to:

$$d_{rbf}(\boldsymbol{x}_i, \boldsymbol{x}) = 2 - 2\kappa(\boldsymbol{x}_i, \boldsymbol{x})$$

So, the distance $d_{rbf}(\boldsymbol{x}_i, \boldsymbol{x})$ is smaller, the more similar $\boldsymbol{x}$ and $\boldsymbol{x}_i$ are. For this choice of kernel, we obtain the same neighbors as KNN an hence the same classifier.

10.5 Sparse Vector Machines

Recall that VMs use a linear predictor: $\eta(\boldsymbol{x}) = \boldsymbol{\alpha}^\top \phi_{vm}(\boldsymbol{x}) = w_0 + \sum_i \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$

💡 In sparse VMs, only a few weights α_i are non-zero → fewer data points matter. This reduces overfitting and computational cost. This is important, especially when data set / N is large.

Sparse VMs can be obtained by using:

- non-kernelized methods, where $\kappa(\boldsymbol{x}_i, \boldsymbol{x})$ is a direct input to the ML algorithm. In this case, we're directly learning w_0 and the values of α_i. Hence, we can put a prior/regularization on α to encourage sparsity
- kernelized methods, where we implicitly use transformed features $\phi_k(\boldsymbol{x})$, e.g., by using mercer kernel and applying the kernel trick. Hence, the feature space is determined by the kernel κ not the input data. Regularization/Priors are implicitly on weights for transformed features. In this setting, sparsity can be achieved via a modified loss → SVM.

10.5.1 Non-Kernelized Case

10.5.1.1 Sparsity via ℓ_0 Regularization

In the *Regularized Risk Minimization* framework, we may directly encourage sparsity of the model parameters $\boldsymbol{\theta}$ using the ℓ_0 *pseudo*-norm:

$$J_0(\boldsymbol{\theta}) = R_{emp}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_0$$

- Recall that $\|\boldsymbol{\theta}\|_0$ simply counts the non-zero entries in $\boldsymbol{\theta}$.
- $R_{emp}(\boldsymbol{\theta})$ measures how good our model with parameter $\boldsymbol{\theta}$ fits the training data.
- Coefficient λ trades off fit (small λ) and sparsity (large λ)

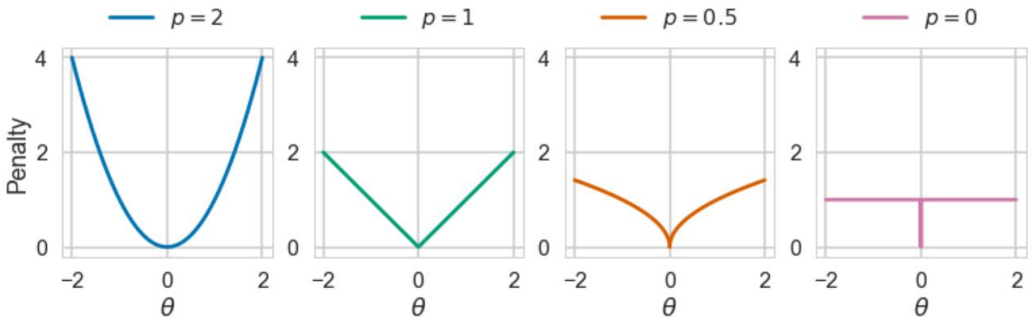
❗ This is a very hard optimization problem and therefore not often used in practice.

10.5.1.2 Sparsity via ℓ_1 Regularization

We may use ℓ_1 regularization to encourage sparsity instead:

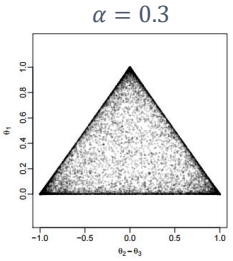
$$J_1(\boldsymbol{\theta}) = R_{emp}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

- Recall that $\|\boldsymbol{\theta}\|_1$ is the *sum of absolute values* in $\boldsymbol{\theta}$.
- This leads to *shrinkage* (model is encouraged to choose smaller parameter estimates)
- Easier to optimize, e.g., via (sub-) gradient-based methods



L_p norm penalties for a parameter $\boldsymbol{\theta}$ according to different values of p. It is easily observed that $p > 0$, impose shrinkage for large values of $\boldsymbol{\theta}$. By gradually allowing decreasing p we observe that the shrinkage is reduced and for $p = 0$ we observe that the penalty is a constant for $\boldsymbol{\theta} \neq 0$.

10.5.1.3 Sparsity via Prior



Consider a probabilistic model $p(\cdot\,|\boldsymbol{\theta})$ and a prior $p(\boldsymbol{\theta})$. A prior is *sparsity-promoting* if substantial probability mass/density is at regions where $\boldsymbol{\theta}$ contains zeros. This encourages sparse MAP estimates.

An example of such a prior is the Dirichlet prior ($\alpha \ll 1$). Note that this introduces an inductive bias.

10.5.2 Support Vector Machines

We're now considering the kernelized case. By applying the kernel trick, we're now working in the *implicit feature space* of the kernel. We cannot simply use ℓ_2 regularization as this would regularize the implicit weights $\boldsymbol{w}$ and not the model parameters $\boldsymbol{\alpha}$.

Support Vector Machines modify the loss function to encourage sparsity. In a nutshell:

$$SVM \approx linear\ predictor + kernel\ trick + sparsity/large\ margin$$

The linear predictor $\eta(\boldsymbol{x}) = w_0 + \sum_i \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$ is the same as before. As only few α_i's are nonzero, the predictions are more efficient to perform. The corresponding data points $\boldsymbol{x}_i$ are called *support vectors*. Therefore, we only need to store these $\boldsymbol{x}_i$'s and α_i's to perform predictions → representation of the model.

❗ Note that there is no natural probabilistic interpretation and no estimate of (un-) certainty.

10.5.2.1 ϵ-insensitive Loss

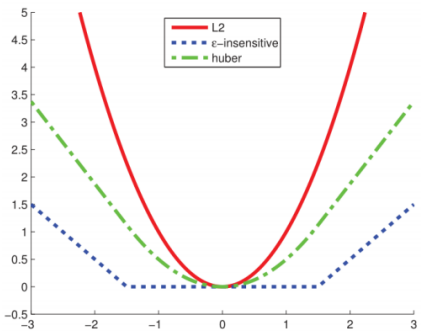
💡 The Key Idea of SVMs: examples which are predicted *well enough* do not incur any loss. These examples do not affect the cost at the solution - when the optimal hyperplane is found, and they do not affect predictions ($\alpha_i = 0$) → no support vectors.

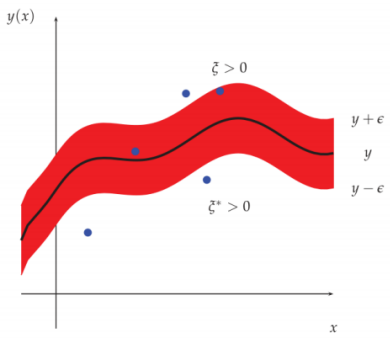
For Regression, ϵ-insensitive Loss is defined as:

$$L_\epsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{else} \end{cases}$$
$$= (|y - \hat{y}| - \epsilon)_+$$

where $(x)_+ = \max(0, x)$. Meaning if the model is closer than ϵ to the true value, then $L_\epsilon = 0$.

This forms a ϵ-tube around the true values, wherein predictions have no loss.





10.5.2.2 Support Vector Regression (SVR)

Using the ϵ-insenstive loss with a linear predictor, we obtain *Support Vector Regression*.

$$J = C\sum_i L_\epsilon\big(y_i, \hat{y}(\boldsymbol{x}_i)\big) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

where the predictor $\hat{y}(\boldsymbol{x}_i)$ is a linear function in the kernel's feature space. Similarly, the ℓ_2 regularization/penalty is *with respect to the weights* $\boldsymbol{w}$ in the feature space of the kernel.

- If the kernel is linear, then the resulting SVM is called *linear SVR*.
- $C = \frac{1}{\lambda}$ is a regularization constant → hyperparameter to prevent overfitting
- ϵ → hyperparameter

This is a convex problem, meaning that it has a one unique, global minimum (cf. 1.7).

10.5.2.3 Hinge Loss

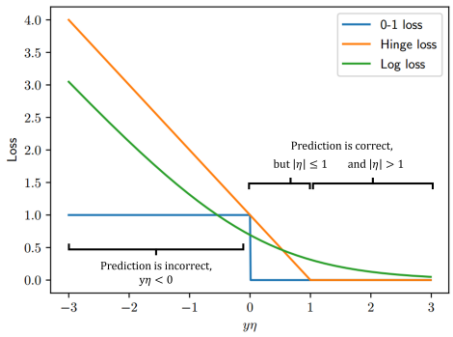
Let $y \in \{-1,1\}$ then the *hinge loss* is defined as:

$$L_{\text{hinge}}(y,\eta) = \begin{cases} 0 & \text{if } y\eta > 1 \\ 1 - y\eta & \text{else} \end{cases} = (1 - y\eta)_+$$

where $y\eta$ is called the *margin*. Iff the *prediction is correct* $\big(y = \text{sgn}(\eta)\big)$ and the predictor is confident (meaning the score is large: $|\eta| > 1$), then $y\eta > 1$, and the hinge loss is zero.

- $\eta(\boldsymbol{x})$ is a score/predictor produced by the classifier
- $\text{sgn}(x)$ is the sign-function, defined as: $\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$

The important point is that a data point that is correct with a large enough score, does not incur any loss, regardless of the score η.



Even though a larger η means that the model is "more confident", there is no probabilistic interpretation, when using the hinge loss (or 0-1 loss).

10.5.2.4 Support Vector Classification (SVC)

Using the *hinge loss* with a linear predictor η, we obtain *Support Vector Classification*.

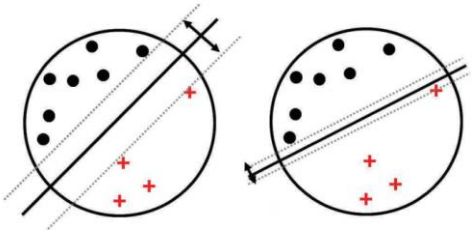
$$J = C\sum_i L_{\text{hinge}}\big(y_i, \eta(\boldsymbol{x}_i)\big) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

where the predictor $\eta(\boldsymbol{x}_i)$ is a linear function in the kernel's feature space. Similarly, the ℓ_2 regularization/penalty is *with respect to the weights* $\boldsymbol{w}$ in the feature space of the kernel.

- If the kernel is linear, then the resulting SVM is called *linear SVC*.
- $C = \frac{1}{\lambda}$ is a regularization constant → hyperparameter to prevent overfitting

This is a convex problem, meaning that it has a one unique, global minimum (cf. 1.7).

10.5.3 Large-Margin Classifiers



The *large-margin principle* is to select a decision boundary with the largest distance to the closest training point.

The goal is to reduce generalization error.

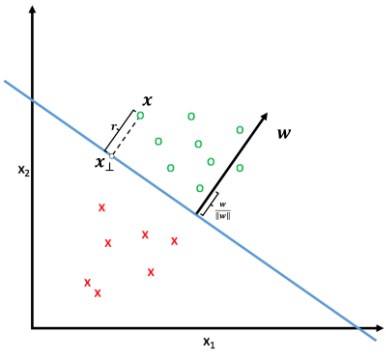
Consider a perfect classifier (e.g., data is linearly separable). All data points within the margin of any other training point, will be classified just as this one. The larger the margin, the more consistent the output of the classifier is around the training data.

This expresses an inductive bias/assumption → data points that are close get the same label.

SVCs are large-margin classifiers. Recall that the score function is: $\eta(\boldsymbol{x}) = w_o + \boldsymbol{w}^\top \boldsymbol{x}$.

The distance of a point $\boldsymbol{x}$ to the decision boundary is:

$$r = \frac{\eta(\boldsymbol{x})}{\|\boldsymbol{w}\|} \quad \text{resp.} \quad \eta(\boldsymbol{x}) = r\|\boldsymbol{w}\|$$



The weight vector $\boldsymbol{w}$ is orthogonal to the decision boundary; this is a characteristic feature of linear models. Therefore:

$$\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

as dividing a vector $\boldsymbol{w}$ by its norm v$\|\boldsymbol{w}\|$ results in the same vector $\boldsymbol{w}'$ that has been scaled to unit length: $\|\boldsymbol{w}'\| = 1$.

Note that $r > 0$ for positive prediction and $r > 0$ for negative predictions.

For linearly separable data, the *maximum margin classifier* is given by:

$$\underset{\boldsymbol{w}, w_0}{\text{argmax}} \min_i y_i r_i = \underset{\boldsymbol{w}, w_0}{\text{argmax}} \min_i y_i \frac{\eta(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

meaning we're computing the margin with $\min_i y_i r_i$ and then choosing $\boldsymbol{w}$ s.t. it maximizes this margin. The label y_i corresponds to data point $\boldsymbol{x}_i$. At the final solution all points are correctly classified ($y_i r_i > 0$) and margin ($y_i r_i = |r_i|$) is as large as possible.

As scaling the weight vector by factor c is possible without changing the resulting solution (decision boundary is the same, as also gets scaled by c), we obtain multiple solutions to above maximization problem. To circumvent this, we can add a constraint that each margin $y_i \eta_i \geq 1$.

This results in a reformulated objective with *hard-margin constraints*:

$$\underset{\boldsymbol{w}, w_0}{\text{argmin}} \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{s.t.} \quad y_i \eta_i \geq 1 \quad \text{for all } i = 1, \ldots, N$$

But what if the data is not linearly separable? There would be no solution for which $y_i \eta_i \geq 1$ $\forall i = 1, \ldots, N$ holds. This is because $y_i \eta_i$ is negative for wrongly classified data points, which will happen if it is not linearly seperable.

10.5.4 Soft-Margin Classifiers

By introducing *slack variables* ξ_i we obtain *soft-margin constraints*:

$$\underset{\boldsymbol{w}, w_0}{\operatorname{argmin}}\ C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad \xi_i \geq 0;\ \ y_i\eta_i \geq 1 - \xi_i$$

these slack variables are $\xi_i \geq 0$ for each data point.

- $\frac{1}{2}\|\boldsymbol{w}\|^2$ is for large margin
- $\sum_i \xi_i$ is for few errors (too small margin or misclassified)
- C is regularization-hyperparameter that controls trade-off

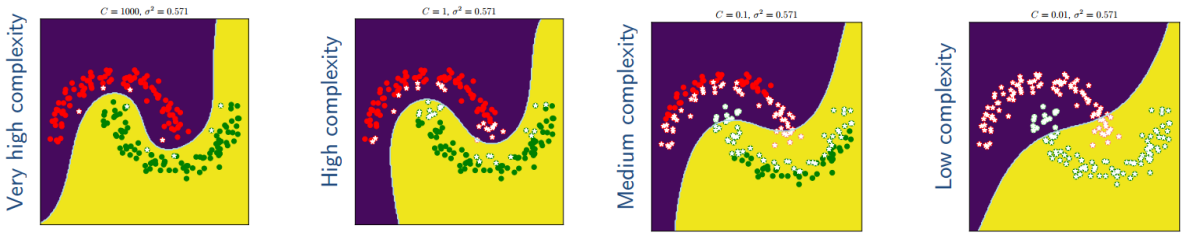
Further:

- If correctly classified and $margin \geq 1$, then: $\xi_i = 0$
- If correctly classified but $0 < margin < 1$, then: $0 < \xi_i < 1$
- If incorrectly classified, then: $\xi_i \geq 1$

This optimization is equivalent to using *hinge loss,* and obtain the SVM objective (cf. *10.5.2.4*):

$$\underset{\boldsymbol{w}, w_0}{\operatorname{argmin}}\ C\sum_i L_{\text{hinge}}(y_i, \eta_i) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

- C controls allowed training errors and affects generalization error
- Sometimes parameterized using $C = \frac{1}{\nu N}$, where ν roughly corresponds to the fraction of misclassified training examples



Note that the margin is computed in the feature space of the kernel, not in data space.

10.5.5 Discussion

Key ingredients to SVM:

- Kernel trick prevents underfitting with linear classifier
- Sparsity/large margin prevents overfitting

Key ingredients to L1VM:

- Uses kernel (solely) to generate explicit features ϕ_{vm}
- $\ell 1$ regularization for sparsity

L1VMs are probabilistic, SVMs are not. L1VMs fast to train, SVMs slow (unless linear). L1VMs any kernel, SVMs Mercer kernel. L1VMs handle multiclass classification naturally, SVMs with difficulties. Both: kernel parameters / regularization weights need tuning

Both are discriminative kernel methods that often perform similar performance in practice.

11 Hyperparameter Optimization (HPO)

Hyperparameters (HP) are fed into, for example, a learning algorithm, while regular *parameters* are the output of that algorithm; they are what the algorithm learns. HP heavily influence the learning process of an algorithm. We distinguish between:

- discrete hyperparameters (e.g., K in KNN, number of units in hidden layer, …)
- continuous hyperparameters (e.g., learning rate, regularization weight, …)
- categorical hyperparameters (e.g., activation function, early stopping, …)

The goal of HP Optimization (HPO) is to automatically select the values for the HPs; but:

- The HP configuration space is complex and high-dimensional
- Model training/evaluation may be expensive
- We cannot directly optimize generalization performance due to limited training data
- Often no gradient / useful properties (e.g., convexity) in HP spaces (often multimodal)

11.1 The HPO Problem

Consider a case with L hyperparameters where the l-th hyperparameter has *domain* Λ_l. This *domain* Λ_l could be binary, categorical or a continuous interval. The cartesian product (all possible combinations of choices) of all HPs give us the *hyperparameter configuration space* $\boldsymbol{\Lambda}$:

$$\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_L$$

This space may also contain *conditional HPs* that are only active for certain choices of other HPs.

Definition: Hyperparameter Optimization

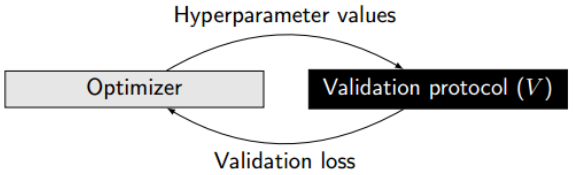
Given data $\mathcal{D}$, the hyperparameter optimization problem is to find: $\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} V(\boldsymbol{\lambda}, \mathcal{D})$

$V(\cdot)$ is a *validation protocol/loss* that measures the performance of using a specific hyperparameter configuration $\boldsymbol{\lambda}$ for a particular task based on data $\mathcal{D}$. Note that in practice we need to approximate $V(\cdot)$ since our dataset is finite. For example, cross validation with a user-defined loss (e.g., misclassification rate).

11.2 Blackbox Optimization

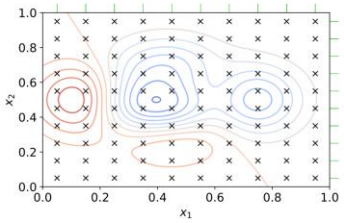
The goal of *black box optimization* (BBO) is to solve above HPO problem. We can only evaluate configurations $V(\boldsymbol{\lambda})$ but have no knowledge about what happens inside of $V(\boldsymbol{\lambda})$. We also do not have gradient information such as $\nabla_{\boldsymbol{\lambda}} V(\boldsymbol{\lambda}, \mathcal{D})$.

Generally, what BBO methods try to do, is to use the *history* $\{(\boldsymbol{\lambda}_i, V(\boldsymbol{\lambda}_i))\}_{i=1}^{n}$ to decide where to evaluate the next configuration $\boldsymbol{\lambda}_{n+1}$. The explored HP values are referred to as *trials*.



Approaches to BBO are *model-free methods* that do not use history (e.g., grid/random search), *stochastic search* methods that maintain distribution $p(\boldsymbol{\lambda})$ to sample new HP values to evaluate, and *global optimization methods*, that maintain an approximation of the validation loss $\hat{V}(\boldsymbol{\lambda})$.

11.2.1 Grid Search



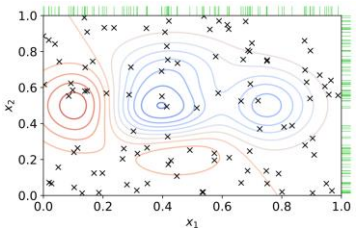
Specifies a finite set of choices for each hyperparameter and the evaluates all combination of these choices.

For example, two real-valued hyperparameters on a 10×10 grid. Blue valleys indicate good results (low loss) while red hills indicate bad results (high loss).

❗ It suffers from the curse of dimensionality, as number of trials grows exponentially with dimensionality of configuration space. E.g., 30 HP with 4 choices each → 10^{18} trials.

❗ The choice of grid points may be non-trivial and also may significantly affect/bias results.

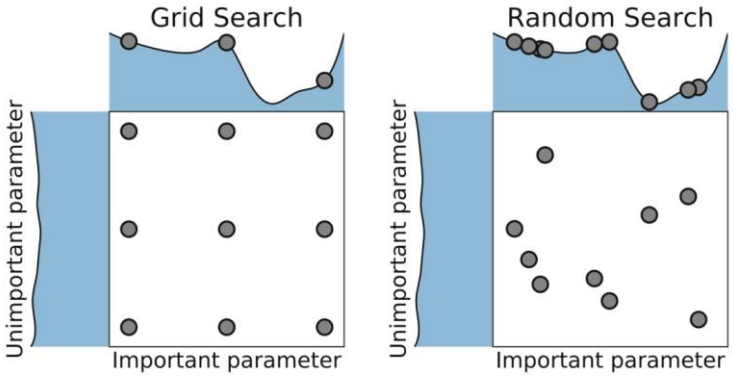
11.2.2 Random Search



Randomly samples configurations from $\boldsymbol{\Lambda}$. For example, two real-valued hyperparameters with 100 trials using uniform sampling. To avoid clumping (dense Λ clusters) in practice, individual components x_1, x_2 should not be sampled uniformly.

💡 Random search explores $\boldsymbol{\Lambda}$ better, especially if some hyperparameters are unimportant.

- Both methods are easily parallelizable across a set of machines
- Random size is easier to shrink/scale



❗ Both methods may take far longer than a *guided* search method.

11.2.3 Population-based methods

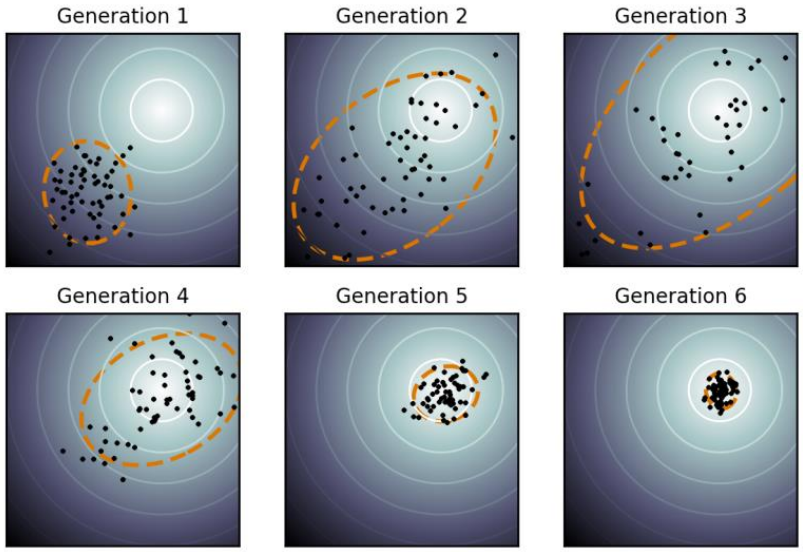
📄 Population-based methods belong to the category of *stochastic search methods*. They maintain a *population* of configurations. The population is used to determine which trials to perform next. Afterwards the population is updated based on the results → *mutation*, *crossover*.

Evolution Strategy (ES)

We maintain an isotropic Gaussian $\mathcal{N}\left(\boldsymbol{\theta}^{(t)}, \sigma^2 \boldsymbol{I}\right)$ that describes where to look for new configurations. The mean $\boldsymbol{\theta}^{(t)}$ is essentially the center of the exploration space. The scope of the search is determined by σ^2.

In the i-th iteration:

1. Sample λ configurations from $\mathcal{N}(\boldsymbol{\theta}^{(t)}, \sigma^2 \boldsymbol{I})$ as offspring and evaluate
2. Keep best μ results
3. Use their mean as $\boldsymbol{\theta}^{(t+1)}$



Variations include:

- (μ, λ)-ES, where we select and keep best results from offspring only
- $(\mu + \lambda)$-ES, where we include prior trials (parents) when selecting offspring

11.2.4 Bayesian Optimization

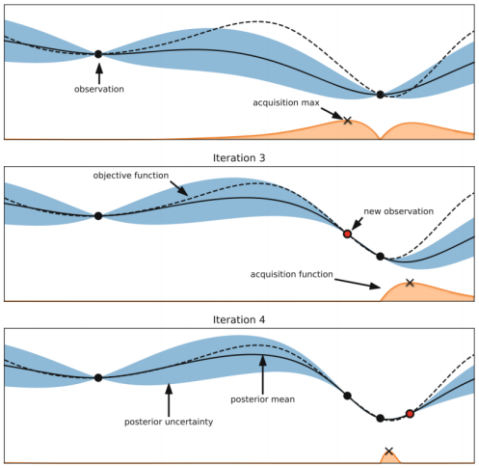
Bayesian Optimization belongs to the category of *global optimization* methods. Approach:

1. Maintain a *probabilistic surrogate model* of $V(\boldsymbol{\lambda})$

2. Optimize *acquisition function* using surrogate model to select next configuration

3. Update surrogate model and repeat

The acquisition function trades off exploration and exploitation. It determines the utility of evaluating each configuration. It is very important to consider the *uncertainty* provided by the surrogate model.



11.2.4.1 Surrogate Models

Surrogate Models are used to model current knowledge about $V(\boldsymbol{\lambda})$.

At heart, this is a regression problem where our input is $\boldsymbol{x} = \boldsymbol{\lambda}$ and output $y = V(\boldsymbol{\lambda})$ corresponds to the validation loss. Data is given by the history: $\mathcal{D}\{(\boldsymbol{\lambda}_i, V(\boldsymbol{\lambda}_i))\}_{i=1}^{n}$

Hence, we could use any parametric model to derive the posterior predictive: $p(\hat{V}|\boldsymbol{\lambda}, \mathcal{D})$
E.g., this could be *Bayesian linear regression*: $p(\hat{V}|\boldsymbol{\lambda}, \mathcal{D}, \sigma^2) = \mathcal{N}(\boldsymbol{\theta}^\top \boldsymbol{\lambda}, \sigma^2)$

11.2.4.2 Gaussian Process Regression

A *Gaussian Process* $\mathrm{GP}(\mu_0, \kappa)$ represents a distribution over possible validation functions:

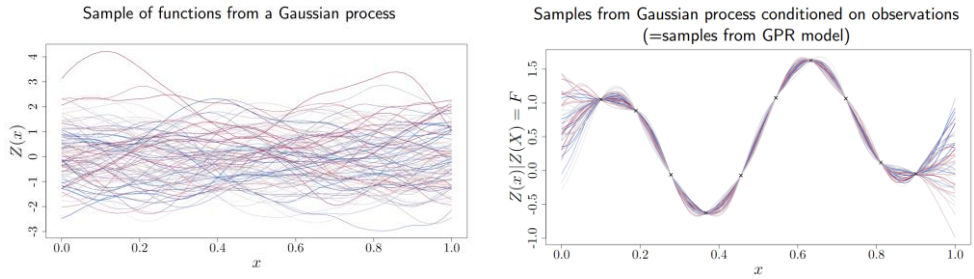
$$f: \boldsymbol{\Lambda} \to \mathbb{R}$$

Gaussian Process Regression (GPR) is a non-parametric model Bayesian approach that uses a GP as a prior to build a regression model. For this it uses the history $\mathcal{D}\{(\boldsymbol{\lambda}_i, V(\boldsymbol{\lambda}_i))\}_{i=1}^{n}$ and some adequately chose hyperparameters.:

- Kernel κ
- Prior mean function $\mu_0: \boldsymbol{\Lambda} \to \mathbb{R}$

We obtain a GP by kernelizing Bayesian linear regression. For this we use kernel $\kappa(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ to measure similarity between configurations. We replace the design matrix $\boldsymbol{X} = (\boldsymbol{\lambda}_i^T)_{i=1}^{n}$ with a kernel matrix $\boldsymbol{K}$, where $\boldsymbol{K}_{ij} = \kappa(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j)$.

Intuitively, prior ensures smoothness of functions for validation loss, i.e., that similar HP-configuration inputs (according to kernel) produce similar outputs (validation loss).



Assumptions

Let $f_i = f(\boldsymbol{\lambda}_i)$ be unknown function values of an arbitrary set of configurations $\{\boldsymbol{\lambda}_i\}_{i=1}^{n}$. we assume the distribution of these function values to be a jointly gaussian distribution:

$$p(\boldsymbol{f}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{m}, \boldsymbol{K})$$

where the mean of this distribution $m_i = \mu_0(\boldsymbol{\lambda}_i)$ is given by the *prior mean function.* The covariance structure is given by the kernel matrix $\boldsymbol{K}$, meaning that the covariance depends on the similarity of each of the configurations $\boldsymbol{\lambda}_i$.

The GP serves as a prior within Bayesian Optimization, meaning that we treat the observations as noisy observations $\boldsymbol{y}$ of an unknown function $\boldsymbol{f}$ (with GP as prior). The final model is as follows:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2) \quad \text{where} \quad \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K})$$

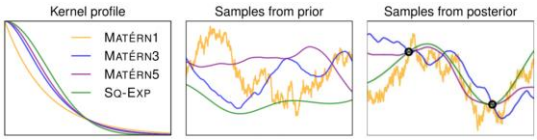
Acquisition Function

Consider a new input $\boldsymbol{\lambda}_{new}$. We get the prediction from the posterior mean $E[f_{new}]$ and the uncertainty from the posterior variance $\mathrm{var}[f_{new}]$. The cost is $O(n^3)$ to fit and then $O(n^2)$ per prediction → expensive for large datasets.

Matérn Kernels

A common choice of kernel classes for this are Matérn kernels. These are stationary kernels where the value only depends on the difference of inputs $\kappa(\lambda_1, \lambda_2) = f(\lambda_1 - \lambda_2)$. It has a smoothness parameter ν because the kernel is differentiable $\lfloor \nu - 1 \rfloor$ times.

- $\nu \to \infty$ equivalent to Gaussian kernel with diagonal covariance
- $\nu \to \frac{1}{2}$ equivalent to exponential kernel



11.2.5 Discussion

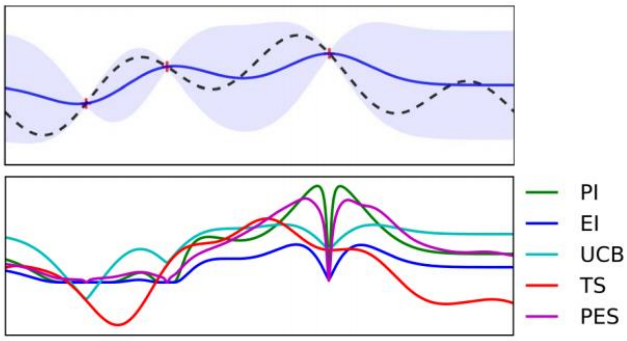
💡 GP in Bayesian Optimization are very expressive, smooth, give well-calibrated uncertainty estimates and are easy to use due to closed-form predictive distribution

❗ However, they are very expensive and face problems with complex configuration spaces. They also are harder to parallelize.

11.2.6 Acquisition Function

📄 The *acquisition function* α scores each of the possible hyperparameter configurations.

- *Probability of Improvement* (PI): $\alpha_{PI}(\boldsymbol{\lambda}) = p\big(\hat{V}(\boldsymbol{\lambda}) < \tau\big)$. States that the score of a configuration is given by the probability the surrogate model assigns to λ being better than target τ. This can work well when τ close to min.; else may exploit too aggressively
- *Expected Improvement* (EI): $\alpha_{EI}(\boldsymbol{\lambda}) = E\left[\left(\tau - \hat{V}(\boldsymbol{\lambda})\right)^{+}\right]$. Uses amount of improvement.
- *Upper Confidence Bound* (UCB) E.g., set $\alpha_{UCB}(\boldsymbol{\lambda})$ such that $p\left(\hat{V}(\boldsymbol{\lambda}) < \alpha_{UCB}(\boldsymbol{\lambda})\right) = 0.05$
- Information-based method use posterior distribution $p\big(\boldsymbol{\lambda}^{*}\big|\hat{V}\big)$ of best configuration



11.2.7 Practical Considerations

In practice, for BBO we need to have a description of the configuration space $\boldsymbol{\Lambda}$ in some way. For continuous values we often make use of box constraints (upper/lower bounds).

In practice, we often have additional constraints: memory consumption, training time, prediction time, energy usage, etc. E.g., a single slow configuration should not consume all of the available time budget. Such can usually only be observed afterwards (e.g., training time).

Simplest approach: add penalty term to $V(\boldsymbol{\lambda})$ for constraint violations

❗ BBO can be expensive, especially for increasing dataset sizes (cf. BERT training took 4 days).

11.3 Multi-Fidelity Optimization

💡 The idea of Multi-Fidelity Optimization (MFO) is instead of always training a model to completion, we can probe a hyperparameter configuration in a less costly manner using low-fidelity approximations. We tradeoff quality of approximation versus runtime → *fidelity*

These approximations can be done by using data subsets, reduced inputs (e.g., image resolution), only training for a few iterations, etc. This allows us to explore larger parts of HP configuration space within a given computational budget.

> 🧠 For expensive tasks, runtime gains often outperform approximation error in practice.

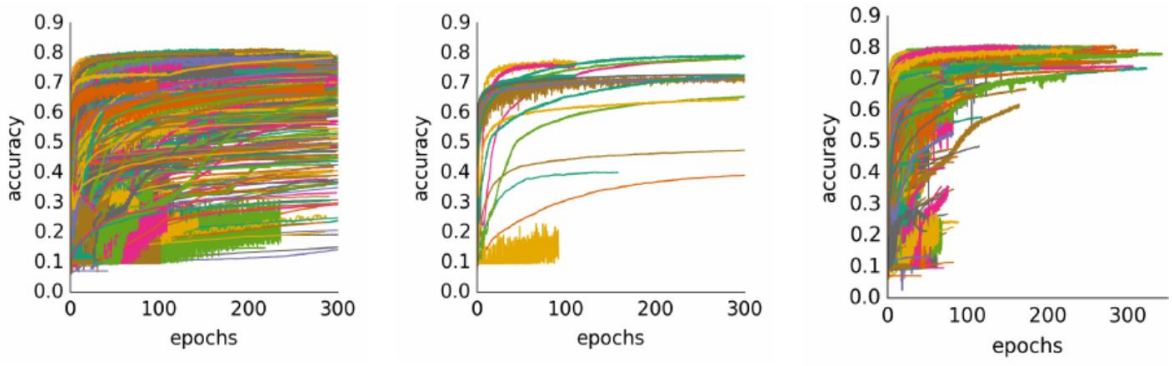
We distinguish three major kinds of MFO methods:

1. *Predictive termination* methods – stop when increasing fidelity is not further beneficial
2. Selection with *static fidelities* – use fixed schedule of fidelities (low → high) for mult. $\boldsymbol{\lambda}$
3. Selection with *adaptive fidelities* – methods try to actively choose fidelities for $\boldsymbol{\lambda}$

11.3.1 Predictive Termination

📄 We obtain a *partial learning curve* (e.g., validation loss, or accuracy samples over epochs) and then use a model to estimate whether we're *likely to fall behind* the best model so far. We can use any HPO method, e.g., random search, for *iterative* ML algorithms but we modify the validation protocol to stop early. This implies we need to validate regularly during training.

💡 The key goal is to identify bad configurations quickly while only having a few observations per curve → uncertainty modeling!



(a) Without predictive termination (b) Random subset of Figure 4a (c) With predictive termination

11.3.2 Multi-Armed Bandits (MAB)

❗ Compared to multi-armed bandits, HPO is a *pure exploration* problem, as the goal is not to maximize "payoff" but rather to estimate the best configuration. But the number of "arms" (configurations) is much larger than the number of available trials; esp. for continuous HP.

We can model MFP as a MAB, where arms are certain HP configurations $\boldsymbol{\lambda}$:

- the first pull of an arm i → train low-fidelity model
- subsequent pulls of arm i → continue training to obtain higher-fidelity model

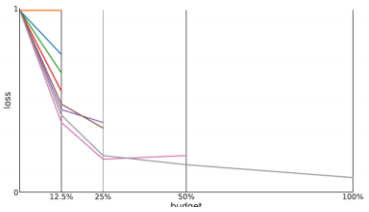
11.3.3 Successive Halving

Input is a set of n arms and an overall budget B. Output should be the best arm. The budget is distributed over $R = \lceil \log_2(n+1) \rceil \approx \frac{B}{R} per\ round$ where R is the number of rounds.

Start with all arms active, then in each round:

- Pull each active arm equally often (≈ doubles each round)
- Remove the bottom half of active arms based on loss

After R rounds only one arm survives → estimated best arm



Example: *"You have a budget of 8 runs over the complete dataset. How could this be realized?"*

First run with all 256 configs (2^8) but only using $\frac{1}{256}$ of the dataset to train. Then, the second run with the 128 best configs and $\frac{1}{128}$ of the training data. Continue until only 2 configs (8^{th} run) are left and pick the better one.

Discussion

- Under certain conditions, provably better than random search. Generally, not much worse than random search. Better anytime performance than random search
- How many arms (n) should we use? ("n $vs.$ B/n problem")

11.3.4 Hyperband

Runs successive halving repeatedly from scratch with a fixed budget but different number of configurations per re-run. It is parameterized by a max-per-configuration budget R and a factor η (we only keep $1/\eta$) for successive "halving". We perform $\lfloor \log_\eta R + 1 \rfloor$ *brackets*:

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Table 1: The values of n_i and r_i for the brackets of HYPERBAND corresponding to various values of s, when $R = 81$ and $\eta = 3$.

In practice, hyperband is outperformed by successive halving with "right" number of configurations.

11.4 HPO in Practice

💡 The choice of HPO method is difficult and application dependent → no golden bullet.

Some criteria for choosing an HPO method:

- Structure of search space → small/large, real-valued, conditional?
- Number of possible evaluations → 10s, 100s, 1000s?
- Degree of possible parallelization
- Expertise and domain knowledge to set HPs?
- Multiple fidelities possible?
- Extensibility (extendable budget) and/or anytime properties (quick results) needed?

❗ HPO methods have hyperparameters themselves (size of grid, sampling distribution, …). These are hopefully easier to set by domain experts.

❗ HP tuned on validation loss may be overfit to validation data. To circumvent this, we can vary training and validation splits across function evaluations. Further, we could prefer stable optima (flat in vicinity of configuration). Also, using ensembles/Bayesian model selection.