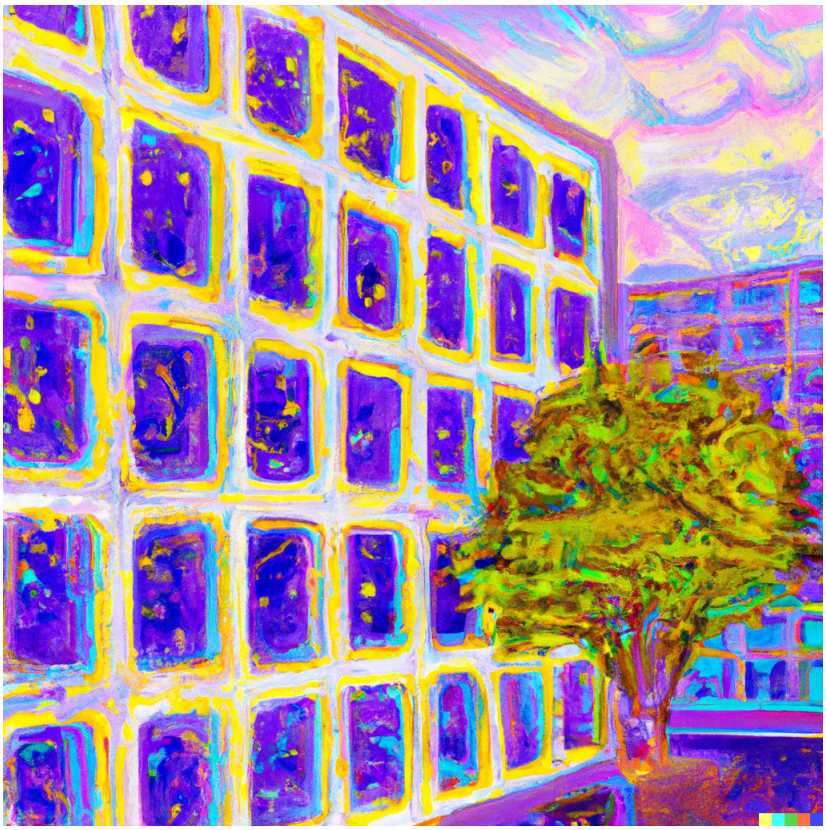
Advanced Machine Learning
Summary

Tim Launer

January 6, 2023



Contents

1 Machine Learning Fundamentals

1.1 Anomaly Detection with PCA and EM

We will introduce a method for anomaly detection based on principal component analysis (PCA), Gaussian mixture models (GMM) and the expectation maximization algorithm. Anomaly or outlier detection is relevant in a variety of use cases, such as detecting fraudulent transactions, detecting sick cells in tissue or identifying suspicious behaviour.

Problem Formalization - Anomaly Detection
Given a set $X = \{x_1, \dots, x_n\}$ from a normal class $N \subseteq \mathbb{R}^D$, compute a function $\phi : \mathbb{R}^D \to \{0, 1\}$ such that $\phi(x) = 1$ if and only if $x \notin N$.

Strategy
An anomaly is an unlikely event, thus we fit a model of a parametric family of distributions $\{p(\cdot|\theta) : \theta \in \Theta\}$ onto a projection in a low dimensional space.
It has been observed that linear projections of high dimensional distributions onto low dimensional spaces resemble Gaussian distributions. Thus, we will:

- project $X = \{x_1, \dots, x_n\}$ onto $\mathbb{R}^d$ with $\pi : \mathbb{R}^D \to \mathbb{R}^d$, where $d << D$.
- fit a GMM p_θ to projected data
- assign each point $x \in \mathbb{R}^D$ the anomaly score $-\log(p_\theta(\pi(x))$

In general, we must keep in mind that very often mathematically tractable solutions do not reflect reality. Thus, there will always be a trade-off between fidelity and tractability of our solutions. This is a recurring theme throughout the lecture.

1.1.1 Dimensionality Reduction with PCA

Principal component analysis tries to find, given $X = \{x_1, \dots, x_n\}$, a linear projection $\pi : \mathbb{R}^D \to \mathbb{R}^d$ with $d << D$ such that $\pi(X)$ has "sufficiently large" variance.

First, we want to deal with the simple case of $d = 1$. Thus, we want to find a projection of the data onto the real line $\pi : \mathbb{R}^D \to \mathbb{R}$. This means that

$$\pi(x) = u_1^T x$$

for some $u_1 \in \mathbb{R}^D$ with $||u_1|| = 1$. We can see that the mean of the projection data is $u_1^T \bar{X}$, where $\bar{X} = \frac{1}{n} \sum_{x \in X} x$ and that its variance is

$$\frac{1}{n} \sum_{x_i} \left(u_1^T \bar{X} - u_1^T x_i\right)^2 = u_1^T \left(\frac{1}{n} \sum_{x_i} (\bar{X} - x_i)(\bar{X} - x_i)^T \right) u_1 := u_1^T S u_1.$$

S is the **covariance matrix** of the data-set. We now want to compute u_1 to maximize the variance of the projection. This can be solved using the Lagrange method for optimization with constraints:
Finding the $\max_{u_1 \in \mathbb{R}^D} u_1^T S u_1$ s.t. $||u_1|| = 1$ is equivalent to optimizing

$$\begin{aligned} Ł(u_1) &= f(x) + \lambda g(x) = u_1^T S u_1 + \lambda(1 - u_1^T u_1) \\ &= u_1^T (S - \lambda \mathcal{I}) u_1 + \lambda \end{aligned}$$

Differentiating w.r.t u_1 and setting equal zero we can easily find:

$$0 \doteq \frac{\partial Ł}{\partial u_1} = (S - \lambda \mathcal{I}) u_1 \iff S u_1 = \lambda u_1$$

Thus, the optimal u_1^* is an eigenvalue of S.
The intuitive follow up question is then: which eigenvalue exactly?
Note:

$$u_1^{T*} S u_1^* = u_1^{T*}(\lambda u_1^*) = \lambda ||u_1^*|| = \lambda$$

Since we want to maximize this expression, λ must be the maximal eigenvalue of S.

Now we can generalize this result to the case $d \geq 1$. We proceed as follows:

- compute u_1^* from X as above.
- let $X_1 = x - proj_{u_1^*} x : x \in X$.
- compute u_2^* from X_1 from X_1 as above.
- let $X_2 = x - proj_{u_1^*} x : x \in X_1$.
- $\cdots$
- compute u_d^* from X_{d-1} as above.
- Define $\pi(x) = (x^T u_1^*, x^T u_2^*, \ldots, x^T u_d^*)$

Note that the definition of a linear projection subtraction is $x - proj_u(x) = x - u(u^T x)$.

We have now found a way to compute a linear projection of the data into a low dimensional space using PCA. Now, we must compute a Gaussian Mixture model on the projected data.

1.1.2 Gaussian Mixture Models with Maximum Likelihood

First, we introduce the model family.

Definition - GMM:
A GMM with K components consists of all distributions whose PDF is of the form

$$p(x) = \sum_{k \leq K} \pi_k \cdot \mathcal{N}(x|\mu_k, \Sigma_K)$$

where $\pi_1, \ldots, pi_k \in \mathbb{R}^+$ s.t. $\pi_1 + \cdots + \pi_k = 1$ are weights assigned to each Gaussian distribution.

Furthermore, let Z be a random variable with range in $1 \ldots, K$ and whos pdf is $Pr(Z = k) = \pi_k$.
The parameters we are concerned with are $\theta = \{\pi_k, \mu_k, \Sigma_k\}$.

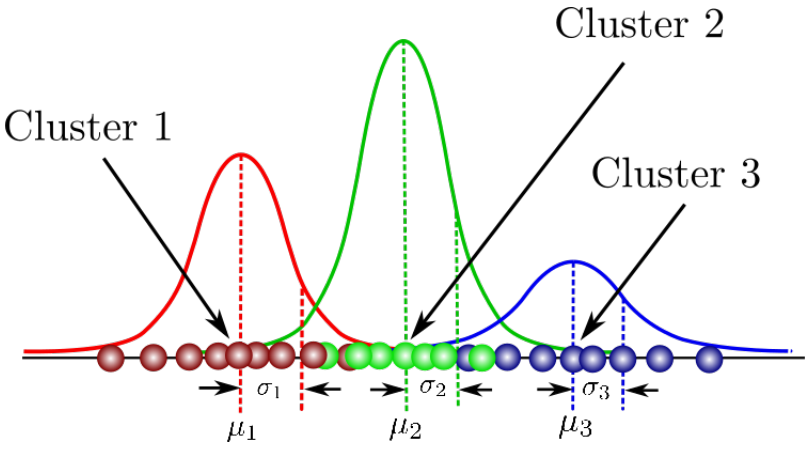


Figure 1: Schematic of a Gaussian Mixture Model in 1 dimension.

Maximum Likelihood Fitting:
Given a dataset $X = \{x_1, \ldots, x_n\}$ and a GMM with K components, we want to choose the parameters θ

such that the model p_θ maximizes the log-likelihood of the data under the model:

$$\log p_\theta(X) = \sum_{i \leq n} \log p_\theta(x_i) = \sum_{i \leq n} \log \left(\sum_k \pi_k \cdot \mathcal{N}\left(x_i | \mu_k, \Sigma_k\right) \right)$$

If we the $z_i \in \{1, \ldots, K\}$ denoting the Gaussian where x_i comes from, then the above expression for the log-likelihood would be much more mathematically tractable:

$$\log p_\theta(X, Z) = \log \left(\Pi_i p_\theta(x_i, z_i) \right) = \sum_i \log \pi_{z_i} + \log \mathcal{N}\left(x_i | \mu_{z_i}, \Sigma_{z_i}\right)$$

We will now look at a popular trick to decompose the intractable expression for $\log p_\theta(X)$ which we want to maximize w.r.t θ into more tractable terms. This will intuitively lead us directly to the expectation maximization (EM) algorithm.

Note that in the following, all expectations are taken over $Z \sim q$, where q can be any distribution for the variable $Z = \{z_1 \ldots, z_n\}$. Recall, that the random variable set Z specifies the cluster each data-point comes from, such that $z_i = k$ tells us that a data-point x_i came from the Gaussian cluster k with parameters μ_k, Σ_k.

$$\log p_\theta(X) = \mathbb{E}\left[\log p_\theta(X)\right] = \mathbb{E}\left[\log \left(\frac{p_\theta(X, Z)}{p_\theta(Z|X)} \frac{q(Z)}{q(Z)} \right)\right] = \mathbb{E}\left[\log p_\theta(X, Z)\right] - \mathbb{E}\left[\log q(Z)\right] + \mathbb{E}\left[\log \left(\frac{q(Z)}{p_\theta(Z|X)} \right)\right]$$

The first equality holds because q is not a distribution over X, the second by the rules for conditional probabilities, the third by properties of the logarithm. We now define the functions

$$M(q, \theta) := \mathbb{E}\left[\log p_\theta(X, Z)\right] - \mathbb{E}\left[\log q(Z)\right] = \mathbb{E}\left[\log \left(\frac{p_\theta(X, Z)}{q(Z)} \right)\right]$$

and

$$E(q, \theta) := \mathbb{E}\left[\log \left(\frac{q(Z)}{p_\theta(Z|X)} \right)\right]$$

Note that these functions are both probability distributions over the data X, taking on different shapes depending on the arguments. These have the properties (proven in exercises):

- 1) $E(q.\theta) \geq 0$ (by Jensen's Inequality)
- 2) $E(q^*, \theta) = 0$ for $q^* = \arg\min_q E(q, \theta) = p_\theta(Z|X)$
- 3) $\max_\theta M(q^*, \theta) \geq \log p_\theta(X)$

The properties and the expression we found directly lead us to an optimization procedure recall, we want to maximize $\log p_\theta(X)$): First, we choose an initial θ_0. By observation 1) and 2), we know that M is always less than $\log p_\theta(X)$, except when q is optimal, in which case $M(q^*, \theta) = \log p_\theta(X)$. This means that $M(q, \theta)(X)$ is an evidence lower bound on X. By observation 3), we also know that for q^*, if we find $\theta^* = \max_\theta M(q^*, \theta)$, we will get $p_{\theta^*}(X) \geq p_\theta(X)$. We can thus maximize $p_\theta(X)$ iteratively by first finding q for a given θ s.t. E is zero and then finding θ^* for a given q^*, which will yield a larger $p_{\theta^*}(X)$. Note here the greater or equal sign... convergence to a maximum $p_\theta(X)$ is guaranteed but the speed is not. We could get stuck.

Here is the algorithm more formally:

- Initialize θ_0
- For $t = 1, 2, \ldots$ do:
  Expectation Step: assign $q^* = \arg\min_q E(q, \theta_{t-1})$
  Maximization Step: compute $\theta_t = \arg\max_\theta M(q^*, \theta)$

There is another intuitive way of interpreting the algorithm: at each step, we first assign each point to its most likely cluster and then reassign the parameters of all the Gaussian clusters to better fit the assigned points.

- The expectation step assigns a distribution to Z. This is equivalent to assigning each point to a Gaussian distribution in the GMM (cluster).
- The maximization step calculates the parameters $\theta = \{\mu_k, \Sigma_k\}$ to make each Gaussian a better fit for the assigned points.

Designing the Anomaly detector
Two interesting things were mentioned. First, we define the anomaly detector as the function $\phi : \mathbb{R}^D \to \{0, 1\}$ which should output 1 if the anomaly score $-\log p_\theta(\pi(x)) > \tau$ for some threshold τ, and $\phi(x) = 0$ otherwise. When training, we want to use a validation metric that gives us both high precision $p = \frac{|C \cap A|}{|C|}$ and high recall $r = \frac{|C \cap A|}{|A|}$, where A is the set of anomalies and C the set of points our classifier marks as anomalies. Thus, we introduced the F_1 score:

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

1.2 Representation, Measurements and Data Types

I will write down here a thoughts from the introductory lecture by Prof. Buhmann that stand out to me.

In general, there are two ways of thinking:

- The deductive way: we start from axioms and deduce observable consequences, testing the axiomatic theory.
- The inductive way: we start from observations (data) and then try to identify general laws (or axioms).

Machine learning is obviously about inductive thinking.
Note the difficulty of defining intelligence (when thinking about what artificial intelligence should be). Prof. Buhmann considers the ability to do counterfactual reasoning and planning to be intelligence.

Algorithm run-time, memory etc have been studied widely, but their robustness and generalizations have not been studied. Learning algorithms are methods (statistical) to explore reality. A lot of work still needs to be done on their generalization and robustness. It is very difficult to talk about correctness of algorithms that process inputs with noise and compute random variables as outputs.

One of the essential questions of machine learning algorithms is validation - how can we know that our algorithms operating on huge data-sets are actually doing the correct thing? Note that verification is guaranteeing performance on seen data, validation aims to quantify how well an algorithm will do on unseen data.
Furthermore, we want to design algorithms that will perform well (whatever that means?) on problems that humans cannot solve - unsupervised learning.

When developing machine learning algorithms, you have to take a very puristic position about what exactly you want to guarantee with your methods. Indeed, in many high-complexity fields, many scientific results are wrong (50% of papers in psychology).
In machine learning pipelines, often preprocessing of the data makes the most significant difference. When we finally analyse the result of an algorithm pipeline in a high complexity problem where "correctness" is not defined, figuring out the sensitivity of each component in the pipeline is important for validation and improvement. It is important to put a lot of attention onto the weakest link of a pipeline.

A model where the signal does not fit into a humans memory cannot be understood. The simpler the signal for the model (an equation to describe the theory), the harder it often is to predict from it. For example, in weather forecasting, Navier Stokes equations are still used for prediction. However, much more data enters now into our boundary conditions, which enables more precise estimation of the solutions. For equations which have no analytical solutions, data on boundary conditions are very important. The struggle is that in high complexity fields, without tools the scientific method is hard.

Representations
A first though - tuples of numbers are not neccessarily vectors. Vector spaces require certain operations (addition and scaling) and sometimes these operations make no sense for certain tuples.
First, some definitions:
Measurement Space is the mathematical space in which the data are represented (numerical, boolean, categorical spaces).
Features are derived quantities or indirect observations which often significantly compress the information content of measurements (e.g. edges in images).
Note, The selection of a specific feature space predetermines the metric to compare data; this choice is the first significant design decision in a machine learning system.

The Learning Problem
Here is the basic learning problem:

- Representation of objects (data representation)
- Definition / modeling of structure and hypothesis classes
- Optimization - search for preferred structures / models
- Validation - are the stuctures indeed in the data or are they explained by fluctuations or method biases?

We are looking for a function $f \in \mathcal{C}$ out of the hypothesis class (or solution space) so that $f : \mathcal{X} \to \mathcal{Y}$.
The **Loss Function** $\mathcal{Q}(y, f(x))$ measures the deviation between dependent variables y and our prediction $f(x)$ - it is essentially a compact parameterization of a posterior. In choosing a loss function we implicitly make assumptions about the probabilistic law of the posterior.
The **Expected Risk** for a random variable X is:

$$R(f) = \mathbb{E}_X\left[R(f, X)\right] = \int_{\mathcal{X}} R(f, X)P(X)dX = \int_{\mathcal{Y}}\int_{\mathcal{X}} \mathcal{Q}(y, f(x))P(X, Y)dXdY$$

with

$$R(f, X) = \int_{\mathcal{Y}} \mathcal{Q}(y, f(x))P(X|Y)dY$$

The **Empirical Risk** is an approximation of the expected risk of our approximation. We split our data into training, validation and test data. Validation data is used as a guide to select an estimator (one with good generalization). Test data is used to evaluate a predictor (never go back to changing the estimator after using test data!).
Using the defintion of risk, we can now specify the training, validation and test error:

$$\hat{R}(\hat{f}, Z) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{Q}(Y_i, \hat{f}(X_i)),$$

where $Z = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ for train, validation and test data respectively. Note that the empirical risk (above on test data) is not the expected risk but only an approximation we use.

Measurements and Objects
We are given an object space $\mathcal{O}$. A measurement X will map an object set into a domain $\mathbf{K}$ (for example into $\mathbb{R}$) $X : \mathcal{O} \to \mathbf{K}$. For more examples see lecture 2B slides.
If you use polyadic data $\mathcal{O}_1 \times \mathcal{O}_2 \times \mathcal{O}_3$ such as {test persons}×{traits}×{behaviours} and you measure them as vectors, you might loose a lot of information.
If you map certain types of data into a specific representation, information about it might not be able to be easily learn-able. For example, sometimes data easily separable in 20 dimensions might be a mess when projected into 2 dimensions. Furthermore, different scales in data can introduce problems in algorithms, thus one should normalize data to get rid of various scales.

Invariances in data are important: if the measurements are invariant under a set of transformation then the mathematical definition of structure should obey the same invariances. Otherwise, our structure search procedure breaks the symmetry in an a priori (not data-dependent) way. In simpler terms, if the data has a certain scale with invariances (Kelvin, ratio scale), our solution structure (e.g. estimator) must obey the same invariances.

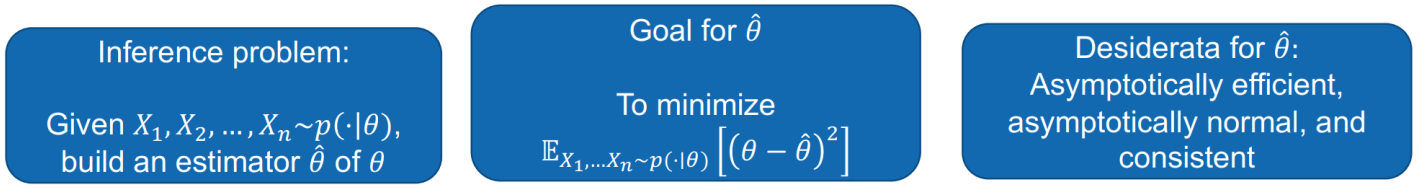
scale type	transformation invariances
nominal	$\mathcal{T} = \{f : \mathbb{R} \to \mathbb{R} \mid f \text{ bijective}\}$
ordinal	$\mathcal{T} = \{f : \mathbb{R} \to \mathbb{R} \mid f(x_1) < f(x_2),\ \forall x_1 < x_2\}$
interval	$\mathcal{T} = \{f : \mathbb{R} \to \mathbb{R} \mid f(x) = ax + c,\ a \in \mathbb{R}^+,\ c \in \mathbb{R}\}$
ratio	$\mathcal{T} = \{f : \mathbb{R} \to \mathbb{R} \mid f(x) = ax,\ a \in \mathbb{R}^+\}$
absolute	$\mathcal{T} = \{f : \mathbb{R} \to \mathbb{R} \mid f \text{ is identity map}\}$

Figure 2: Formal characterisation of different scale types and their invariance properties.

1.3 Density Estimation

Definition (from Wikipedia)
In statistics, probability density estimation or simply density estimation is the construction of an estimate, based on observed data, of an unobservable underlying probability density function.

Simply, as outlined in the figure below, the inference problem is as follows:
We are given a dataset $X = \{x_1, \dots, x_n\}$ following some parametric distribution $p_\theta(X)$ parameterized by an unknown parameter θ. We want to build an estimator that finds an approximation $\hat{\theta}$ for the true parameter θ explaining the data. Our goal is to minimize the expected difference between the estimation $\hat{\theta}$ and the ground truth θ. However, as we will see shortly, this is impossible. Therefore, we instead try to find asymptotically efficient, asymptotically normal and consistent approximations. We will see shortly what this means.



Frequentism	Bayesianism	Statistical learning	Non-parametric statistics
1. Pick a parametric model 2. Fit the model using MLE	1. Guess a prior 2. Pick a parametric model 3. Derive a posterior	1. Forget about distributions! 2. Define a loss function 3. Approximately minimize the expected loss	Later…
• Tractable. • Asymptotically unbiased. • Variance issues.	• Intractable. • Bias issues. • Low variance.	• Tractable. • Low bias and low variance with a proper model (model selection problem).	

Figure 3: Statistical Inference overview and standard approaches.

Recap: Parametric vs. Non-Parametric Inference
Recall our fundamental problem: "given $X = \{x_1, \dots, x_n\} \sim F$, find the distribution F". We now want to construct a statistical model $\mathcal{H}$, which is a set of distributions (or densities or regression functions), to solve this. A **Parametric model** is a set $\mathcal{H}$ of such functions which can be characterized by finitely many parameters

$$\mathcal{H} = \{p(x;\theta) \,:\, \theta \in \Theta\},$$

where Θ is the model parameter space. An example would be the statistical model set of Gaussian distributions characterized by parameters $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d\times d}$, thus $\Theta = \mathbb{R}^d \times \mathbb{R}^{d\times d}$. We implicitly assume that the distribution F lies in the parametric family $\mathcal{H}$.
In contrast, a **Non-Parametric Model** is a set $\mathcal{H}$ which cannot be characterized by finitely many parameters, such as for example $\mathcal{H} = \{\text{all CDFs}\}$. Once we have chosen a model, we must choose an estimator $p \in \mathcal{H}$ to best explain our data. In the following, we will concern ourselves with parametric inference. Note that when working with statistical models, choosing the correct model is key.

1.3.1 The Rao-Cramer Bound and Optimal Estimators

The **Rao-Cramer Bound** shows that in general, even unbiased estimators $\hat{\theta}$, s.t. $\mathbb{E}\left[\theta - \hat{\theta}\right] = 0$, are not optimal and thus cannot retrieve the original θ (never have variance 0):

$$\mathbb{E}_X\left[(\theta - \hat{\theta})^2\right] \geq \frac{1}{I_n(\theta)} = \left(\mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p(X|\theta\right)^2\right]\right)^{-1},$$

where $I_n(\theta)$ is the **Fisher Information**.

Proof.
First, we need to define the score:

$$\Lambda := \frac{\partial}{\partial\theta}\log p(X|\theta) = \frac{\frac{\partial}{\partial\theta}p(X|\theta)}{p(X|\theta)}.$$

The score has the following properties:

- 1) $\mathbb{E}_X[\Lambda] = \int p(X|\theta)\frac{\frac{\partial}{\partial\theta}p(X|\theta)}{p(X|\theta)} = \frac{\partial}{\partial\theta}1 = 0$
- 2) $\mathbb{E}_X\left[\Lambda\hat{\theta}\right] = \int p(X|\theta)\frac{\frac{\partial}{\partial\theta}p(X|\theta)}{p(X|\theta)}\hat{\theta}(X) = \frac{\partial}{\partial\theta}\mathbb{E}_X\left[\hat{\theta}\right] = \frac{\partial}{\partial\theta}\text{bias}(\hat{\theta} + 1$

Let $\hat{\theta}$ be unbiased. First, note:

$$\text{Cov}(\Lambda,\hat{\theta})^2 = \mathbb{E}\left[(\Lambda - \mathbb{E}[\Lambda])(\hat{\theta} - \theta)\right]^2 = (\mathbb{E}\left[\Lambda\hat{\theta} - \Lambda\theta\right])^2 = (\mathbb{E}\left[\Lambda\hat{\theta}\right] - \theta\mathbb{E}[\Lambda])^2 = 1$$

Furthermore:

$$\text{Cov}(\Lambda,\hat{\theta})^2 = \mathbb{E}\left[(\Lambda - \mathbb{E}[\Lambda])(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])\right]^2 \leq \mathbb{E}\left[\Lambda^2\right]\mathbb{E}\left[(\hat{\theta} - \theta)^2\right]$$

Thus, we have shown that $1 \leq \mathbb{E}\left[\Lambda^2\right]\mathbb{E}\left[(\hat{\theta} - \theta)^2\right]$ which implies:

$$\mathbb{E}\left[(\hat{\theta} - \theta)^2\right] \geq \frac{1}{\mathbb{E}[\Lambda^2]} = \frac{1}{I_n(\theta)}$$

We will not prove the last equality. □
We have shown that there is no hope to recover θ exactly. So, what can we aim for instead??

- **Asymptotic efficiency** for unbiased estimators: $\lim_{n\to\infty}\mathbb{E}\left[(\hat{\theta} - \theta)^2\right] I_n(\theta) = 1.$

- **Consistency** $\lim_{n\to\infty} Pr(|\hat{\theta} - \theta| > \epsilon) = 0$ for all $\epsilon > 0$.
- **Asymptotic Normality** (tractable confidence intervals) $\hat{\theta}_n \sim \mathcal{N}\left(\hat{\theta}_n|\theta,\, se^2\right)$

These are our desiderata for the estimator $\hat{\theta}$. We will now explore the popular approaches.

1.3.2 Frequentism

Frequentism follows two steps: first, we choose a parametric model $\mathcal{H} = \{p(x;\theta) : \theta \in \Theta\}$. Then, we choose the estimator $p_\theta \in \mathcal{H}$ which maximizes the likelihood of the data.
We define the likelihood function as $\mathcal{L}_n(\theta) = \prod_{i=1}^n p_\theta(X_i)$. Then, the **Maximum Likelihood Estimator (MLE)** is the $\hat{\theta}_n$ which maximizes the likelihood function $\hat{\theta}_n = \arg\max_\theta \mathcal{L}_n(\theta)$. It can be shown that MLE is unbiased and fulfills the desiderata. However, the frequentist approach often suffers from unnecessarily large variance of the estimator.
Recall the **Bias-Variance Decomposition**:

$$\begin{aligned}\mathbb{E}\left[(\hat{\theta} - \theta)^2\right] &= \mathbb{E}\left[(\theta - \mathbb{E}\left[\hat{\theta}_n\right] + \mathbb{E}\left[\hat{\theta}_n\right] - \hat{\theta}_n)^2\right] = \left(\theta - \mathbb{E}\left[\hat{\theta}_n\right]\right)^2 + \mathbb{E}\left[(\mathbb{E}\left[\hat{\theta}_n\right] - \hat{\theta}_n)^2\right] \\ &= \text{bias}(\hat{\theta}_n)^2 + \text{var}(\hat{\theta}_n)\end{aligned}$$

Thus, we can trade bias for variance! This leads us to Bayesian estimation.

1.3.3 Bayesian Inference

Bayesianism follows the following approach:

- 1) Impose a prior on the parameter θ by choosing a distribution (the prior) $q(\theta)$ over Θ.
- 2) Choose a parametric model $\mathcal{H}$ to define the likelihood $p(X|\theta) \in \mathcal{H}$.
- 3) Update the distribution $q(\theta)$ to find the posterior $p(\theta|X) = \frac{p(X|\theta)q(\theta)}{p(X)} = \frac{p(X|\theta)q(\theta)}{\int q(\theta)p(X|\theta)d\theta}$.
- 4) Estimate and event of interest ϕ - $p(\phi|X) = \int p(\theta|X)p(\phi|\theta)d\theta$.

Note that the integral in step 3) and 4) is often intractable. This means that the likelihood and prior need to be chosen carefully for this approach to work. It is useful to introduce **Conjugate Priors**:
Let $\mathcal{H} = \{p(x;\theta) : \theta \in \Theta\}$ and $\mathcal{Q} = \{q(\,\cdot\,;\omega) : \omega \in \Omega\}$ be two parametric families of distributions. $\mathcal{Q}$ is a conjugate prior for $\mathcal{H}$ if the posterior induced by any two $p \in \mathcal{H}$ and $q \in \mathcal{Q}$ is also in $\mathcal{Q}$.
Conjugate priors are very attractive for Bayesian Inference because we get tractable posteriors.
Note that the Bayesian approach might not fulfill the desiderata.

1.3.4 Statistical Learning

Statistical learning is yet another approach to density estimation. It provides mathematically tractable solutions with low bias and low variance... for a proper model.
In statistical learning, we forget about distributions. First, we must define a model (not a probability distribution) which is a set of functions $\mathcal{H}$ for the task. For example, this could be $\mathcal{H} \subseteq \{f|f : \mathbb{R}^d \to [0,1]\}$ for a classification task.
Second, we must define a loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ which takes the ground truth and the prediction and returns a loss score. The strategy is then to choose the function $f \in \mathcal{H}$ which minimizes the expected (approximated by the empirical) risk:

$$\min_{f\in\mathcal{H}} \mathbb{E}_{X,Y}\left[\mathcal{L}(Y, f(X))\right] \approx \min_{f\in\mathcal{H}} \frac{1}{n}\sum_{i\leq n} \mathcal{L}(y_i, f(x_i))$$

In statistical learning, we are given the freedom to choose the model and the loss function. However, with that freedom comes danger - choosing a good model is difficult. In general, statistical learning does not fulfill the desiderata, it strongly depends on the model and loss function.

When choosing the model, it is important to make sure that the functions are mathematically tractable (such as e.g. $\mathcal{H} \subseteq L_2$). When choosing the loss function, there are many options used commonly in the literature (see slide 52, Lecture 3). One common example is the cross entropy $\mathcal{L} = -\log p_{f(x)(y)}$. When $f(x) \in [0,1]$, f induces a Bernoulli distribution and we find the empirical loss $\frac{1}{n}\sum_{i\leq n}\left(-y_i \log(\sigma(w^T x)) - (1-y_i)\log(1-\sigma(w^T x)\right)$.

When dealing with a range of variables (features), the model selection becomes the appropriate selection of a subset of "meaningful" variables. Furthermore, a good loss function must be selected. There are a number of different approaches for variable (model) selection, including the Akaike information criterion, cross validation or the **Bayesian Information Criterion** (BIC) (for loss function derivation). When imposing a Gaussian prior on the vector of weights (coefficients). It can then be shown that the log likelihood of your data becomes:

$$\log p(X,Y) \approx \text{const} + \log(P(X,Y|w*) - \frac{|S|}{2}\log(n),$$

where $S \subseteq \{1,\dots,d\}$ is the set of variables used and $\log(P(X,Y|w*)$ is the maximized likelihood function. The BIC is then given by:

$$\text{BIC} = -\log p(X,Y|w*) + \frac{|S|}{2}\log(n).$$

We want to select a model with a high BIC (in Lecture, low BIC in Wikipedia???). One way to do this is by forward selection, where we train models with more and more variables and find an optimal cutoff.

1.4 Regression and the Bias-Variance Trade-off

We talk about regression problems and continue to think about optimal estimators and the bias-variance tradeoff. Remember to look at Steins Paradox...

1.4.1 The Regression Problem

The regression problem objective is to find, given $(X,Y) = \{(X_1,Y_1),\dots,(X_n,Y_n)\}$, the (noisy) functional relationship $Y(X) = f(X) + \epsilon(X)$. The optimal solution of the regression problem is the minimum of the expected conditional risk

$$f^*(X) = \arg\min_f \mathbb{E}_{Y|X}\left[(Y-f(X))^2\right] = \mathbb{E}\left[Y|X=x\right].$$

For a derivation, see lecture 4 Notes Thursday. If we assume a Gaussian noise distribution $\epsilon \sim \mathcal{N}\left(\epsilon|Y-f^*(X),\,\sigma^2\right)$, we find

$$f^*(X) = \mathbb{E}_{Y|X}\left[\left(\frac{Y-f(X)}{\sigma}\right)^2\right].$$

Note that since we have limited observations, we always make implicit smoothness assumptions, i.e. we always bet on a simple relationship. In general, the statistical learning approach and the MLE often leads to the same solution.

Recap - Linear Regression
We choose as our statistical model the linear functions $f : \mathbb{R}^d \to \mathbb{R}$ s.t., given vector of inputs $X^T = \{X_1,\dots,X_d\}$:

$$Y = \beta_0 + \sum_{j\leq d} X_j\beta,$$

where β_0 is referred to as the bias in machine learning or the intercept in statistics. Equivalently, we can assume a constant coordinate X_0 s.t. $X \in \mathbb{R}^{d+1}$, which makes the model simply $Y = X^T\beta$ (often also $Y = w^T X$).

To solve the linear regression model, we introduce the **Residual Sum of Squares (RSS)** estimator:
We seek to minimize, given the data $D = \{(x_i, y_i)|i = 1, \dots, n\} \subset \mathbb{R}^{d+1} \times \mathbb{R}$, the sum:

$$RSS(\beta) = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 = (Y - X\beta)^T(Y - X\beta).$$

The minimum condition is $0 \doteq \nabla_\beta RSS(\beta) = X^T(Y - X\beta)$, giving the solution for non-singular X^TX: $\hat{\beta} = (X^TX)^{-1}X^TY$. We can make predictions directly with $y = x^T\hat{\beta} = x^T(X^TX)^{-1}X^TY$.
If we go back to the statistical Gaussian noise assumption, we find that the predictions should follow the normal distribution $\beta \sim \mathcal{N}\left(\beta|\hat{\beta}, (X^TX)^{-1}\sigma^2\right)$.

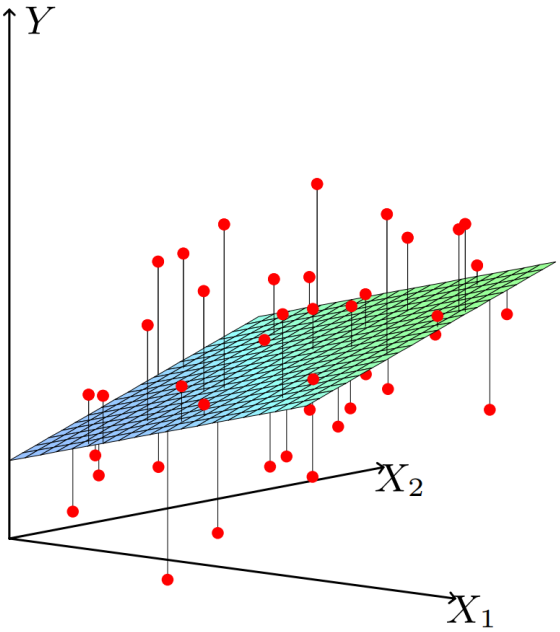


Figure 4: Linear Least Squares Fitting - We seek the linear function of X that minimizes the sum of squared residuals from Y.

Optimality of the Least Squares Estimate
We showed that the least squares estimate of the parameter β has the smallest variance of all linear unbiased estimates!
Consider the problem of estimating the value of of a linear combination $\theta = a^T\beta$, e.g. the $f(a)$ at a new location $a \in \mathbb{R}^{d+1}$. Look at slides 7 and 8 of Lecture 4 and repeat the calculations to show that the least squares estimate $\hat{\theta} = a^T\hat{\beta} = a^T(X^TX)^{-1}X^TY$ is unbiased (i.e. $\mathbb{E}\left[a^T\hat{\beta}\right] = a^T\beta$ and that its variance is:

$$\mathbb{V}\left[a^T\hat{\beta}\right] = \sigma^2 a^T(X^TX)^{-1}a.$$

Furthermore, any alternative estimator $\tilde{\theta} = c^TY = a^T\hat{\beta} + a^TDY$ is also unbiased if $a^TDX = 0$.
Gauss Markov Theorem
For any linear estimate $\tilde{\theta} = c^TY$ that is unbiased for $a^T\beta$ will have larger variance than $\hat{\beta}$:

$$\mathbb{V}\left[a^T\hat{\beta}\right] \leq \mathbb{V}\left[c^TY\right].$$

Proof.
Let $c^TY = a^T((X^TX)^{-1}X^T + D)Y$ be any unbiased estimate of the function $f(a) = a^T\beta$. It follows that $a^TDX = 0$ (see slides).

$$\begin{aligned}\mathbb{V}\left[c^TY\right] &= \mathbb{E}\left[(c^TY)^2\right] - \mathbb{E}\left[c^TY\right]^2 = c^T(\mathbb{E}\left[[]\, YY^T\right] - \mathbb{E}\left[Y\right]\mathbb{E}\left[Y\right]^T)c = \sigma^2 c^Tc \\ &= \cdots \text{(see slides, do yourself)} \\ &= \mathbb{V}\left[a^T\hat{\beta}\right] + \sigma^2 a^TDD^Ta \geq \mathbb{V}\left[a^T\hat{\beta}\right]. \qquad \square\end{aligned}$$

The question becomes, is this now the best we can do at all?
We found that $\hat{f}(x) = x^T(X^TX)^{-1}X^TY$ is the best unbiased estimate we can find. However, over-fitting can be a problem. But now we must remember the bias-variance tradeoff.

1.4.2 The Bias-Variance Tradeoff

Remember that the mean squared error of our prediction can be split into different kinds of errors:

$$\text{MSE} = \text{bias}^2 + \text{variance} + \text{noise variance}.$$

More technically, for a dataset $D = \{(x_i, y_i)|i = 1, \ldots, n\} \subset \mathbb{R}^{d+1} \times \mathbb{R}$ we can see:

$$\mathbb{E}_D\left[\mathbb{E}_{Y|X=x}\left[(\hat{f}(x) - Y)^2\right]\right] = \mathbb{E}_D\left[\hat{f}(x) - \mathbb{E}\left[Y|X=x\right]\right]^2 + \mathbb{E}_D\left[\left(\hat{f}(x) - \mathbb{E}_D\left[\hat{f}(x)\right]\right)^2\right] + \mathbb{E}\left[(Y - \mathbb{E}\left[Y|X=x\right])^2\right].$$

In general, we are facing two main issues:

- We only have limited data to construct an estimate.
- We don't know the hypothesis (model) class $\mathcal{C}$ - complexity is unknown.

If we choose a very complex hypothesis class $\mathcal{C}$, we might quickly run into overfitting. However, using a $\mathcal{C}$ that is too simple will not allow us to captures functional relationship - we under-fit. Usually, minimizing both the bias and variance is impossible. The optimal tradeoff between bias and variance is achieved when we avoid both underfitting (large bias) and overfitting (large variance). For small datasets and a large $\mathcal{C}$ we usually have large variance and small bias. For large datasets and a small $\mathcal{C}$ we usually have small variance and large bias.
Outlook: Ensemble methods seem to avoid the bias/variance tradeoff since they lower variance while keeping the bias fixed. Note: The Rao-Cramer inequality defines a lower bound for variance reduction by ensemble averaging (no free lunch).

Regularization
There are some tricks we can employ to try to find a good tradeoff:

- Regularization - adding a complexity term to the loss function $\arg\min_\theta \sum_D \mathcal{L}(f(x_i, \theta), y_i) + R(\theta)$, which is often equivalent to choosing priors and using maximum a posteriori (MAP) estimators in a Bayesian Framework.
- Model selection based on generalization error estimate (e.g. by cross-validation).
- Ensembles of classifiers (later...).

We will now focus on regularization of linear regression. There are two very common approaches - Ridge and LASSO (Least absolute shrinkage and selection operator) are outlined in the figure below.

Ridge Regression

Cost function: $RSS(\beta;\lambda) = (\mathbf{y}-\mathbf{X}\beta)^\mathsf{T}(\mathbf{y}-\mathbf{X}\beta) + \lambda\beta^\mathsf{T}\beta.$

Bayesian view: $Y|(X,\beta) \sim \mathcal{N}\left(x^\mathsf{T}\beta, \sigma^2\mathbf{I}\right)$, prior on β: $\beta \sim \mathcal{N}(0, \sigma^2/\lambda\mathbb{I})$.

Solution: $\hat{\beta}^{\text{ridge}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$

Tikhonov regularization $R(\beta) = \lambda\beta^\mathsf{T}\beta$ is also called weight decay in Neural Networks Literature.

LASSO

Cost function: $RSS(\beta;\lambda) = (\mathbf{y}-\mathbf{X}\beta)^\mathsf{T}(\mathbf{y}-\mathbf{X}\beta) + \lambda\|\beta\|_1.$

Bayesian view: $Y|(X,\beta) \sim \mathcal{N}\left(x^\mathsf{T}\beta, \sigma^2\mathbf{I}\right)$, prior on β_i: Laplace: $p(\beta_i) = \frac{\lambda}{4\sigma^2}\exp(-|\beta|\frac{\lambda}{2\sigma^2})$.

Solution: By efficient optimization techniques (e.g. LARS). Note: $\|\beta\|_1 = \sum_{j=0}^{d}|\beta_j|$ is not differentiable.

Figure 5: Equivalence of common regularization methods and Bayesian MAP estimation. For model selection, the complexity parameters λ or s are chosen by estimates of the generalization error, e.g. cross-validation.

If we look at the ridge regression estimate $X\hat{\beta}_{ridge} = X(X^TX + \lambda\mathcal{I})^1X^TY$ and take the singular value decomposition $X = UDV^T$, we find:

$$X\hat{\beta}_{ridge} = UD(D^2 + \lambda\mathcal{I})^1U^TY = \sum_{j=1}^{d} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T y,$$

where d_j are the singular values. Note that the shrinkage factor is small for small singular values and approaches 1 for large ones - built in model selection.
Equivalently, LASSO can be defined as the least squares loss function subject to the constraint $\sum_{j=1}^{d}|\beta_j| < s$ for some shrinkage factor s.
LASSO estimates are known to be sparse with few coefficients non-vanishing. This is because the least squares error (LSE) surface often hits the corners of the constraint surface (see fig 3.12 of Hastie et al.). This is illustrated in the figure below, along with the generalized version of ridge regression.

Generalized Ridge Regression

$$\hat{\beta} = \arg\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{d} x_{i,j}\beta_j\right)^2 + \lambda\sum_{j=1}^{d}|\beta_j|^q\right\}.$$

This cost function models the shrinkage of the coefficients!

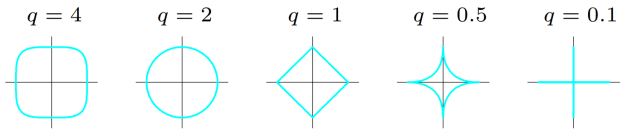


Figure 6: General Ridge Regression Illustration - the least squares error (LSE) surface often hits the corners of the constraint surface for $q = 1$, which leads to sparse solution. However, other choices will not give nice sparse kernels. For $q < 1$, cost functions may be non-convex.

Non Linear Regression
To do non linear regression with the same methods, we can transform X non linearly $\phi(X)$ and then do linear regression in the new feature space. E.g. use transformations $h_m(X) : \mathbb{R}^d \to \mathbb{R}$, $1 \le m \le M$, then the model

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

will be linear in β and but nonlinear in X. In the lecture, examples such as smoothing wavelet and cubic splines transformations are presented.

2 Gaussian Processes

2.1 Kernel Methods - A Summary

Sometimes there are algorithms for well established problems (such as linear regression), that do not model a sufficiently complex hypothesis class $\mathcal{C}$, but we nevertheless wish to apply these algorithms to model a more complex $\mathcal{C}$.
One approach we have already seen is manual feature engineering, where we design a nonlinear feature map $\phi : \mathbb{R}^n \to \mathbb{R}^m$, where n is the number of features of the data and m the nonlinear features we wish to work with (e.g. polynomial features to model polynomial functions with linear regression).
Kernel Methods allow us to work in highly complex, infinite dimensional feature spaces and model complex functions. They can be applied to any algorithm, where the training data enters only in the form of an inner product in $\langle x, y\rangle = x^T y$. We can then rewrite the scalar product in an infinite dimensional feature space using the **Kernel Function**

$$k(x, y) = \langle \phi(x), \phi(y)\rangle = \phi(x)^T \phi(y).$$

$k(x, x')$ is a valid kernel function if and only if it fulfills the definition of a scalar product in some space $\mathcal{V}$. More technically, the function must fulfill Mercer's condition for all sequences of points $(x_i, \dots, x_n)$ and coefficients $(c_1, \dots, c_n)$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} k(x_i, x_j)c_i c_j \geq 0$$

Equivalently, its $n \times n$ **Gram Matrix** $K = \phi(X)^T\phi(X)$ (for $X = \{x_1, \dots, x_n\}$) must be positive *semi definite*, which means hermitian (symmetric) $K^\dagger = K$ with non-negative eigenvalues or equivalently $x^T Kx \geq 0$ for all non-zero column vectors x;

$$K_{ij} = \phi(x_i)^T \phi(x_j) = k(x_i, x_j).$$

The advantage of such a formulation is that it allows us to *implicitly* work in an infinite dimensional feature space without defining $\phi(x)$ *explicitly*.

Some classical kernels on $\mathbb{R}^d$:

Linear kernel:	$k(x,x') = x^\mathsf{T}x'$
Polynomial kernel:	$k(x,x') = (x^\mathsf{T}x' + 1)^p$, for $p \in \mathbb{N}$
Gaussian (RBF) kernel:	$k(x,x') = \exp\left(-\|x - x'\|_2^2/h^2\right)$
Sigmoid (tanh) kernel:	$k(x,x') = \tanh \kappa x^\mathsf{T}x' - b$

Figure 7: Classical examples of kernel functions. Different kernels have different invariance properties, such as under rotation or translation, which can bring advantages for certain data types.

Example. The loss function of **Linear Ridge Regression** is

$$\mathcal{L}(x) = \frac{1}{2}||w^T\phi(X) - y||^2 + \lambda w^T w$$

Using the gram matrix K, we can rewrite this equivalently in the infinite dimensional feature space:

$$\mathcal{L}(\alpha) = \frac{1}{2}\alpha^T KK\alpha - \alpha^T Ky + \frac{1}{2}y^T y + \lambda\alpha^T K\alpha.$$

Discussion. Recall that technically, a function that defines a scalar product must be associated to a positive definite (not semi-definite matrix). I am actually unsure exactly why the weaker condition holds, but I believe it must be the content of the proof of Mercer's theorem. I assume that a slightly weaker (not complete) scalar product suffices as a kernel function. In the lecture, the conditions required are symmetry and positive semi-definiteness: $\int_\Omega k(x, x')f(x)f(x')dxdx' \geq 0$, where $f \in L_2$, $\Omega \subset \mathbb{R}^d$.
An interesting comment made by Prof. Buhmann about the structure of mathematics: hierarchy matters, meaning that the most elemental thing we can define in mathematics is a Set of objects. The first operation

we can think about the testing whether and element is already in a set. The next best thing we can hope for is a similarity measure, which is often very difficult for dissimilar objects (inter-species genomes etc.). Only then can we talk about distance measures. Kernels are powerful because they can be similarity measures for many objects. Note here that kernel functions can be defined beyond $\mathbb{R}^d$ (such as sequence kernels, graph kernels, etc.) which measure similarity on different data structures.

Given two kernel functions $k_1(x, x')$ and $k_2(x, x')$ defined on the same data space,
new kernel functions $k(x, x')$ can be constructed by the following rules:

Addition:	$k(x, x') = k_1(x, x') + k_2(x, x')$
Multiplication:	$k(x, x') = k_1(x, x') \cdot k_2(x, x')$
Scaling:	$k(x, x') = c \cdot k_1(x, x')$ for $c > 0$
Composition:	$k(x, x') = f(k_1(x, x'))$ where f is a polynomial with positive coefficients or the exponential function.

Figure 8: Kernel Engineering is important in many domains. It allows us to flexibly model machine learning solutions for various data types like strings, graphs, trees, lists, etc., when the algorithm depends only on scalar products.

2.2 Definition, Learning and Prediction with $\mathcal{GP}$s

Gaussian Processes can be derived intuitively from Bayesian linear regression. This is very useful for understanding, but involves some long calculations, which I skipped here. I recommend it though, it is nicely documented in the PAI script or the slides.

Definition. A Gassian process ($\mathcal{GP}$) is an infinite set of random variables such that any finite subset of which has a joint Gaussian distribution.

We use $\mathcal{X}$ do denote an infinite set of random variables. A Gaussian Process $\mathcal{GP}(\mu, k)$ is characterized by its *mean function* $\mu : \mathcal{X} \to \mathbb{R}$ and a *covariance function* (also called *kernel function*) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. For any subset $A := \{x_1, \dots, x_m\} \subset \mathcal{X}$, we have:

$$f_A = [f_{x_1}, \dots, f_{x_m}] \sim \mathcal{N}\left(f | \mu_A, K_{AA}\right),$$

where

$$f_A = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{bmatrix} \text{ and } K_{AA} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \dots & k(x_m, x_m) \end{bmatrix}.$$

Learning
Now that we have defined Gaussian Processes, how do we learn with them from data?
Recall the regression problem $y_i = f(x_i) + \epsilon$, with $\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ at the observations $A = \{x_1, \dots, x_m\}$. Given a prior $f \sim \mathcal{GP}(\mu, k)$, we can show (good exercise) that the posterior will be $f | x_{1:n}, y_{1:n} \sim \mathcal{GP}(\mu', k')$, where for $k_{x,A} = [k(x, x_1) \cdots k(x, x_n)] \in \mathbb{R}^m$ we have:

$$\mu'(x) = \mu(x) + k_{x,A}^T (K_{AA} + \sigma^2 \mathcal{I})^{-1} (y_A - \mu_A)$$
$$k'(x, x') = k(x, x') - k_{x,A}^T (K_{AA} + \sigma^2 \mathcal{I})^{-1} k_{x',A}.$$

This posterior can now be used for prediction. Note that we can only decrease the posterior covariance by conditioning on additional data.

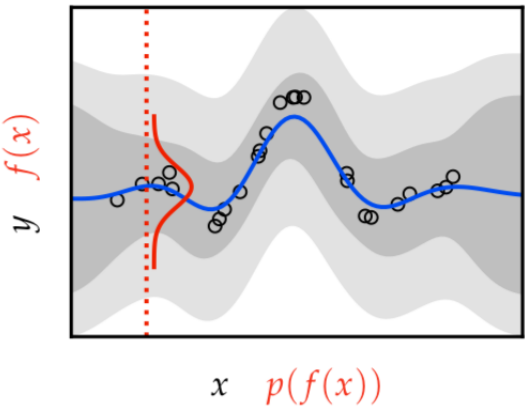


Figure 9: A Gaussian Process can be interpreted as a Gaussian distribution over functions. At any point x, we get a Gaussian distribution over values $f(x)$. The blue function is the MAP estimate given the data (i.e. the mean function of the posterior $\mathcal{GP}$). The dark grey region is the epistemic uncertainty, the light grey the additional aleatoric uncertainty (label noise).

Prediction
From the above posterior, we can derive a conditional distribution for the noise free predictions $f^*|x^*, x_{1:n}, y_{1:n} \sim \mathcal{N}(\mu^*, k^*)$ of a new data point x^* (important exercise with conditional Gaussians., see PAI script):

$$\mu^* = \mu(x^*) - k_{x^*,A}^T (K_{AA} + \sigma^2 \mathcal{I})^{-1} (y_A - \mu_A)$$
$$k^* = k(x^*, x^*) - k_{x^*,A}^T (K_{AA} + \sigma^2 \mathcal{I})^{-1} k_{x^*,A}.$$

From this, we find the predictive posterior distribution of the labels y, assuming homoscedastic noise:

$$y^*|x^*, x_{1:n}, y_{1:n} \sim \mathcal{N}\left(\mu^*, k^* + \sigma^2\right).$$

Examples and Comments from the Lecture
For model selection in $\mathcal{GP}$s, we have two steps. First, we must choose a kernel for the problem (critical). Our kernel will come with hyperparameters, which we must optimize in the next step. Model validation (kernel validation) is essential, otherwise we build prejudices into the model. This can be done using e.g. leave-one-out cross validation.

3 Convex Optimization and Support Vector Machines (SVMs)

We talk about the basics of convex optimization and duality in the context of support vector machines. First, we briefly recap Lagrange's method for optimization. Then, we will specify the problem of support vector machines in the context of convex optimization and finally, we will solve it.

3.1 Super Short Recap on Lagrange's Method

We want to find an extremum of the function $f(x) : \mathbb{R}^n \to \mathbb{R}$, given some constraint functions $g(x) : \mathbb{R}^n \to \mathbb{R}$ and $h(x) : \mathbb{R}^n \to \mathbb{R}$, such that $g(x) = 0$ and $h(x) \leq 0$.
With some geometric considerations, we notice that at the optimal solution, all the gradients of the function f and the two constraints g,h must be co-planar. Thus, at the extremum x^*:

$$\nabla f(x^*) + \lambda \nabla g(x^*) + \alpha \nabla h(x^*) = \nabla (f(x^*) + \lambda g(x^*) + \alpha h(x^*) = 0,$$

where $\alpha \geq 0$ and $\alpha h(x^*) \leq 0$. We therefore find that to optimize f under constraints we must find the stationary points of the Lagrange function:

$$\mathcal{L}(x, \lambda, \alpha) = f(x) + \lambda g(x) + \alpha h(x).$$

This function can be generalized for multiple constraints $g_i(x) = 0$ and $h_j(x) \leq 0$:

$$\mathcal{L}(x, \lambda, \alpha) = f(x) + \sum_i g_i(x) + \sum_j h_j(x).$$

3.2 Duality

In mathematical optimization, the duality principle is the idea that an optimization problem may be viewed from two different perspectives: the **primal** problem and the **dual** problem. If the primal problem is a minimization problem, then the dual problem is a maximization problem and vice versa.
Any feasible solution to the primal (minimization) is at least as large as any feasible solution to the dual (maximization). Therefore, the solution to the primal is an upper bound to the solution of the dual and the solution to the dual is a lower bound to the solution of the primal. In general, the solutions to the dual and the primal don't have to be equal but may differ with some non-zero *duality gap*. This is called **weak duality**.
For convex optimization problems, the duality gap is zero if **Slater's condition** holds. This is then called **Strong duality**.

The derivation of the Dual Problem Recall that the primal is to find $\min_w f(w)$ s.t. $g_i(w) = 0$ and $h_j(w) \leq 0$.
Let w^* be the optimal of the primal and let $\lambda = (\lambda_1, \ldots)$ and $\alpha = (\alpha_1, \ldots)$ with $\alpha_j \geq 0$. We find for the Lagrange function:

$$\mathcal{L}(w^*, \lambda, \alpha) = f(w^*) + \sum_i g_i(w^*) + \sum_j h_j(w^*) \leq f(w^*).$$

We thus define the **Dual** to find $\max_{\lambda,\alpha} \theta(\lambda, \alpha) = \max_{\lambda,\alpha} \inf_{w \in \mathbb{R}^d} \mathcal{L}(w, \lambda, \alpha)$.

In the case of weak duality, we have that $\theta(\lambda, \alpha) = \inf_{w \in \mathbb{R}^d} \mathcal{L}(w, \lambda, \alpha) \leq \mathcal{L}(w^*, \lambda, \alpha) \leq f(w^*) = \min_w f(w)$ s.t. $g_i(w) = 0$ and $h_j(w) \leq 0$.

Slater's Condition
Is a sufficient condition for strong duality:
If there exists a $w \in \mathbb{R}^d$ s.t. $g_i(w) = 0$, $\forall i$ and $h_j(w) \leq 0$, $\forall j$, thus if at least one w exists that fulfills all the constraints, then strong duality holds.

In the case of strong duality, we have that $\theta(\lambda^*, \alpha^*) = f(w^*)$. Furthermore, strong duality implies the following:

- $w^* = \arg\min_w \mathcal{L}(w, \lambda, \alpha)$ The primal solution is the minimizer of the Lagrangian.
- $\alpha_j^* h_j(w^*) = 0, \forall j$ Complementary Slackness.

Discussion. Note that the objective is to find the minimal value taken by some function f (the primal) subject to constraints. Instead, we can find the maximum value taken by some other function θ without constraints as a lower bound on the minimum value of f. Given such an optimum, we can easily then retrieve the parameter $w^* = \arg\min_w f(w)$, if needed. In the case of strong duality, the maximum of the function θ is already the minimum of f exactly. Furthermore, the value of w at which this occurs is exactly the minimizer of the Lagrangian as well.

3.3 Hard Margin SVMs

Lets look at convex optimization in the case of support vector machines. The problem setting is the following: Given the data $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathbb{R}^d \times \{-1, +1\}$, we want to find the hyperplane $w^T x + w_0 = 0$ which separates the dataset $y_i(w_i^T x_i + w_0) > 0$ and simultaneously maximizes the margin between support vectors $2m = \frac{|w^T x^+ - w^T x^-|}{||w||}$. This is shown in the figure below. The primal can be constructed by using

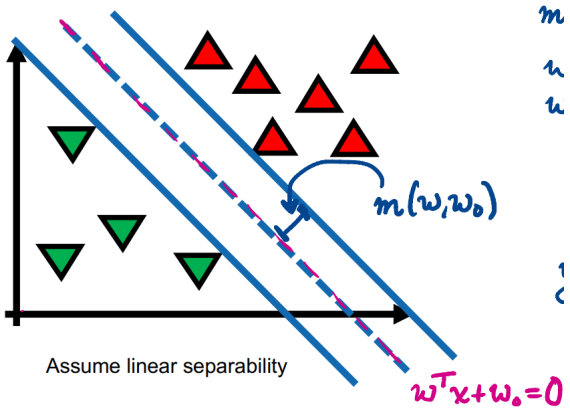


Figure 10: Problem setting of SVMs: finding the hyperplane that maximizes the margin between the support vectors.

a normalization trick shown in the lecture (slide 33, lecture 4), whereby we observe that $m(w, w_0)$ does not depend on the norm $||w||$. We can thus choose the normalization of w s.t. for the support vectors $(w^*)^T x^{\pm} + w_0^* = \pm 1$, in which case we find that $2m(w, w_0) = \frac{2}{||w||}$. We want to choose w as small as possible, such that the labels are still classified correctly.
Thus in the case of binary classification $y_i \in \{-1, +1\}$, the **Primal** becomes:

$$\min_{w, w_0} \frac{1}{2}||w||^2, \text{ such that } y_i(w^T x_i + w_0) \geq 1, \ \forall i \leq n.$$

If we assume linear seperability of the data, then Slater's condition is fulfilled and we have **strong duality**. The Lagrangian of this problem is:

$$\mathcal{L}(w, w_0, \alpha) = \frac{1}{2} w^T w + \sum_{i \leq n} \alpha_i (1 - y_i(w^T x_i + w_0)).$$

We now want to minimize $\mathcal{L}$ in w, w_0. Using the conditions $0 \doteq \frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial w_0}$, we find that the minimum value is at $w^* = \sum_{i \leq n} \alpha_i y_i x_i$. Thus, we can construct the **Dual** problem:

$$\max_{\alpha} \sum_{i \leq n} \alpha_i - \frac{1}{2} \sum_{i,j \leq n} \alpha_i \alpha_j y_i y_j x_i^T x_j, \text{ such that } \alpha_i \geq 0, \sum_{i \leq n} \alpha_i y_i = 0.$$

This is solved by quadratic optimization on a simplex.
Note that the data x_i only enter into the algorithm as a scalar product, which implies that SVMs can be kernelized.
Complementary slackness shows us that furthermore, $\alpha_i^*(1-y_i((w^*)^T x_i+w_0)=0$, which means that $\alpha_j \neq 0$ if and only if x_j is a support vector, which is rare. Therefore, the vector of coefficients α is sparse, and by extension the optimal parameter $w^* = \sum_{i\leq n} \alpha_i^* y_i x_i$ is a sparse linear combination of support vectors. The optimal w_0 can be found with $w_0^* = -\frac{1}{2}(w^*)^T x^+ + (w^*)^T x^-)$.

3.4 Soft Margin and Multiclass SVMs

So far, we have seen the case where the dataset is linearly separable. Of course, this may not always be the case. Furthermore, even if it is the case, a hard margin may in some cases not bring the otimal classifier with the lowest variance (i.e. might not generalize well) because of points "misrepresented" due to noise or perturbations. In such cases, allowing some misclassification or neglect of training examples will provide a more robust solution.

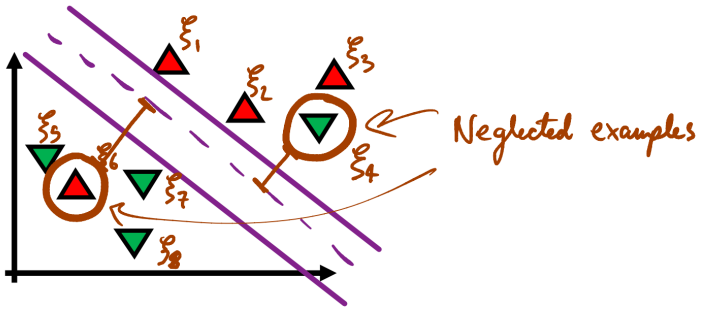


Figure 11: Soft Margin SVMs: we allow some examples to be neglected to get a more robust separation.

Soft Margin SVMs
We construct a new formalization for the prior, allowing for some neglect:

$$\min_{w,w_o,\xi} \frac{1}{2}||w||^2 + C\sum_{i\leq n}\xi_i, \text{ such that } y_i(w^T x_i + w_0) \geq 1-\xi_i \ \forall i$$

where $C \in \mathbb{R}$ is a new hyperparameter we must choose. Note that is C is very large, the $\xi_i \in \mathbb{R}$ will go to zero and we will recover the hard margin primal. If C is small however, an optimal solution can be found where some ξ_i are large, allowing for w to not satisfy the original constraint in some examples (those examples are neglected). In a way, we give each datapoint some help to make it across the boundary. The corresponding dual problem becomes (derivation as exercise):

$$\max_{\alpha} \sum_{i\leq n}\alpha_i - \frac{1}{2}\sum_{i,j\leq n}\alpha_i\alpha_j y_i y_j x_i^T x_j, \text{ such that } C \geq \alpha_i \geq 0, \sum_{i\leq n}\alpha_i y_i = 0.$$

Note that this dual is exactly the same as for the hard margin primal, except that $0 \leq \alpha \leq C$, such that α is now upper bounded by C. The optimal w^* and ξ_i^* are:

$$w^* = \sum_{i\in\{\text{supp.vec}\}} \alpha_i^* y_i x_i$$
$$\xi_i^* = \max\{0, 1-y_i((w^*)^T x_i + w_0^*)\}.$$

Kernelization
As noted above, SVMs are kernelizable. This is another approach to get linear separability back (instead of soft margins), by transforming the dataset into a space where it is hopefully separable. Since the data enters

into the dual during training only as an inner product, we can rewrite the dual in terms of some non-linear coordinate transformation $\phi : \mathbb{R}^d \to \mathbb{R}^m$:

$$\max_{\alpha} \sum_{i \leq n} \alpha_i - \frac{1}{2} \sum_{i,j \leq n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j), \text{ such that } \alpha_i \geq 0, \sum_{i \leq n} \alpha_i y_i = 0.$$

For prediction of a new example x_{n+1}, we find:

$$w^* \phi(x_{n+1}) = \sum_{i \leq n} \alpha_i^* y_i k(x_i, x_{n+1}),$$

where we write as usual $k(x, x') = \phi(x)^T \phi(x')$ for the kernel function. We saw in the lecture that the RBF kernel is indeed an inner product in an infinite dimensional euclidian space. This is interesting, but not relevant here. We also saw kernelized least squares regression, which is a good exercise.

Multiclass SVMS
The formalization above can very easily be generalized. Assume we are given k classes and $y_i \in \{1, \ldots, k\}$. We introduce weight vectors for every class $w^T = (w_1^T, w_2^T, \ldots, w_k^T)$, each defining a separating hyperplane. We define the margin as the maximal m that satisfies $(w_{y_i}^T x_i + w_{y_i}^0) - \max_{y \neq y_i} (w_y^T x_i + w_y^0) \geq m$. Note that blindly maximizing the margin does not work, since we can arbitrarily increase w. Thus we instead set the scale $m = 1$. We then find the generalization of the primal:

$$\min_{all\, w_y} \frac{1}{2} \sum_{y \leq k} w_y^T w_y, \text{ such that } (w_{y_i}^T x_i + w_{y_i}^0) - \max_{y \neq y_i} (w_y^T x_i + w_y^0) \geq 1.$$

The constraint can be though of as making sure that the correct class is sufficiently separated from the second best class prediction.
For the soft margin formalization, we simply add again the term $+C \sum_{i \leq n} \xi_i$ and subtract ξ_i on the rhs of the constraint.

3.5 SVM Regression and Structured Support Vector Machines

SVM Regression
We introduce a loss function

$$L^\epsilon(x, y, f) = |y - f(x)|_\epsilon = \max(0, |y - f(x)| - \epsilon).$$

$(w^T x_i + w_0) - y_i \leq \epsilon + \xi_i$ The primal problem is:

$$\min_{w,\xi} \frac{1}{2} w^T w + C \sum_{i \leq n} (\xi_i + \hat{\xi}_i), \text{ such that } (w^T x_i + w_0) - y_i \leq \epsilon + \xi_i \text{ and } y_i - (w^T x_i + w_0) \leq \epsilon + \hat{\xi}_i,$$

where $\xi_i, \hat{\xi}_i \geq 0$ are the slacks for being below or above the regression line and outside the sensitivity band with width ϵ, respectively.
The dual formulation is then:

$$\max_{\alpha_i, \hat{\alpha}_i} \sum_{i \leq n} (\hat{\alpha}_i - \alpha_i) y_i - \epsilon \sum_{i \leq n} (\hat{\alpha}_i + \alpha_i) - \frac{1}{2} \sum_{i,j \leq n} (\hat{\alpha}_j - \alpha_i)(\hat{\alpha}_i - \alpha_j) x_i x_j, \text{ such that } C \geq \alpha_i, \hat{\alpha}_i \geq 0, \sum_{i \leq n} (\hat{\alpha}_i - \alpha_i) = 0,$$

where $\alpha_i \hat{\alpha}_i$ are the variables associated to the constraints of being above or below the regression line and outside the sensitivity bound.

Structured SVMs
Since kernels can be constructed for a variety of different mathematical objects to introduce a similarity or distance measure, we can generalize the concepts learned from support vector machines to very complex data

structures which are not easily represented in $\mathbb{R}^d$. Examples include sequences (in NLP), bipartite graphs, trees (Bioinformatics) or sequences of actions (in Robotics).

There are some essential problems we must overcome with SSVMs. First, the output space may not be easily representable (e.g. too many classes w.r.t training data). Prediction may also be very hard (e.g. classes not feasible to enumerate) or errors during prediction may be hard to quantify (e.g trees that are "almost correct" should be better than "completely wrong"). Lastly, we may always run into training efficiently problems.

We need to define some functions for formalize structural SVMs:

- a joint feature map $\Psi(y, x)$ based on class labels y and inputs x, which gives us features for each example based on which class it would be assigned to. This allows us to control the complexity of the output space. It should capture the domain expertise (i.e. what are the essential features).
- a scoring function $f_w(y, x) = w^T \Psi(y, x)$, which tells us how well an example fits to a class, based on the features and trainable weights w.
- classification $\hat{y} = h(x) = \arg\max_y f_w(y, x)$.

We define the margin m as the maximum that satisfies $w^T \Psi(y_i, x_i) - \max_{y \neq y_i} w^T \Psi(y_i, x_i) \geq m, \forall i$. Setting the scale with $m = 1$, this yields the primal optimization problem for hard functional margin SSVM:

$$\min_w \frac{1}{2} w^T w \text{ such that } w^T \Psi(y_i, x_i) - \max_{y \neq y_i} w^T \Psi(y_i, x_i) \geq 1, \ \forall i.$$

To get prediction errors, we define a loss function $\Delta : \mathbf{K} \times \mathbf{K} \to \mathbb{R}$ for the output space $\mathbf{K}$, where $\Delta(y', y)$ is the loss of predicting y' when y is correct. Note that the margin is also a score function. We can now reformulate the problem with introduced slacks $\xi_i \geq 0$:

$$\min_w \frac{1}{2} w^T w + C \sum_{i \leq n} \xi_i \text{ such that } w^T \Psi(y_i, x_i) - \max_{y \neq y_i} [\Delta(y, y_i) + w^T \Psi(y_i, x_i)] \geq -\xi_i, \ \forall i.$$

Note that we want the example to be better by at least some slack compared to the second best prediction enhanced by the corresponding error. Furthermore, we cannot use standard quadratic programming solvers. Instead, we must refer to more complex algorithms such as a *cutting plane* algorithm. Convergence (i.e. solving the problem) is not longer guaranteed.

4 Ensemble Methods

The idea of ensemble estimators is nicely illustrated by a story: In 1906, Sir Francis Galton asked 787 people to estimate the weight of an ox. No one got the right answer. However, the average prediction was precise within one pound. This phenomenon is called the *wisdom of the crowd*. Note that it allows us to shift the effort from the "smart" design choice side to the computation side. These methods are still very popular in areas where deep learning has not yet been successful, such as medical biology.

Key Idea
Weak, but better than random classifiers are easy to train! Data randomization in the spirit of Bootstrap captures statistical dependencies between alternative hypotheses. Thus, we train an ensemble of weak estimators on Bootstrapped or weighted alternative datasets to get an aggregate solution with (hopefully) low bias and variance.

Strategy
Given an ensemble of estimators $h_1(x), \dots, h_B(x)$, want to infer a combined classifier with weights α_i:

$$\hat{h}(x) = \sum_{i=1}^{B} \alpha_i h_i(x).$$

For this to work, co-variances between estimators need to be small, otherwise, averaging will not help. This can be achieved by using different subsets of data or features for each estimators, or by shifting hyperparamters or even models for the estimators.
Ensemble methods are similar in spirit to the Bayesian method, where we integrate over a set of models to find a MAP estimate.

4.1 Bagging

Bagging stands for **B**ootstrap **Agg**regation. The idea is to train a number of classifiers on Bootstrapped (sampling with replacement) subsets of the original dataset $\{(x_1, y_1) \dots, (x_n, y_n)\}$ and then average their prediction:

$$\hat{h}(x) = \sum_{i=1}^{B} h_i(x).$$

In the case of classification, we choose the majority vote label as the final classifier.
For **Classifier Selection**, we first train both types of classifiers on the bootstrapped samples and then choose the classifier which is "more often correct" on the bootstrapped subsets.

Discussion. Why does Bagging work?
It reduces the variance by the wisdom of the crowd by a factor $\frac{1}{B}$, where B is the number of bootstrapped subsets. Furthermore, covariances are kept small by training each estimator on a different subset. The aggregate bias is also only weakly affected by this. Thus, Bagging reduces expected risk by reducing variance.
Random Forests are a strategy for Bagging decision trees, whereby we first from a tree sufficiently deep to reduce bias and then average over many such trees trained on bootstrapped subsets.

4.2 Boosting

Boosting is analogous to forward stage-wise additive empirical risk minimization. An adaptive combination of poor learners with sufficient diversity leads to and excellent and complex classifier.

AdaBoost
AdaBoost is a sequence of classifiers, each trained on a subset of data weighted $w_i^(b)$ more heavily on the examples the previous classifier got wrong. The algorithm is shown in the figure below. ϵ_b is the performance of the classifier b. It is the average of the weighted misclassifications, normalized by the total introduced

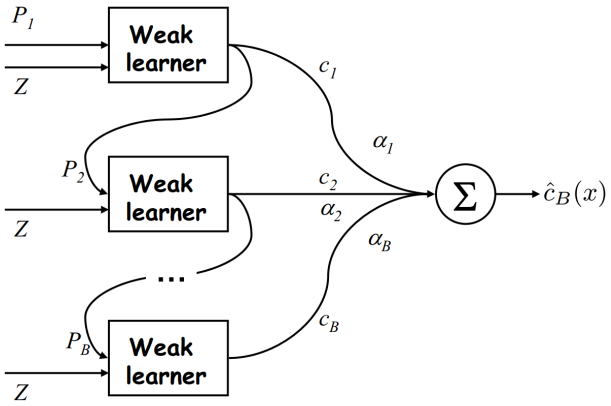


Figure 12: AdaBoost will train a chain of classifiers $c_i(x)$, each one is trained on a dataset weighted heavily on examples which the previous one misclassified.

weights. Each classifier is then weighted in the final prediction by α_b, which is based on the performance ϵ_b.

AdaBoost.M1

Input: data $(x_1, y_1), \ldots, (x_n, y_n)$

Output: classifier $\hat{c}_B(x)$

initialize observation weights $w_i = 1/n$
for $b = 1$ to B **do**
  Fit a classifier $c_b(x)$ to the training data using weights w_i;
  Compute $\epsilon_b \leftarrow \sum_{i=1}^{n} w_i^{(b)} \mathbb{I}_{\{c_b(x_i) \neq y_i\}} / \sum_{i=1}^{n} w_i^{(b)}$;
  Compute $\alpha_b \leftarrow \log \frac{1-\epsilon_b}{\epsilon_b}$;
  Set $\forall i: \; w_i \leftarrow w_i \exp\left(\alpha_b \mathbb{I}_{\{y_i \neq c_b(x_i)\}}\right)$;
end for
return $\hat{c}_B(x) = \text{sgn}\left(\sum_{b=1}^{B} \alpha_b \, c_b(x)\right)$

Discussion. Why does AdaBoost work?
It was shown in the lecture that AdaBoost.M1 is equivalent to forward stage-wise modeling using the exponential loss $\mathcal{L}(F(x), y) = e^{-yF(x)}$. The minimizer of the exponential loss function (w.r.t. true distribution) can be shown to be the log-odds of the class probabilities. This implies that AdaBoost is an additive logistic regression model (with Newton-like updates for minimizing $\mathbb{E}\left[e^{-yF(x)}\right]$).

5 Deep Learning

To start the discussion on deep learning, we look at an interesting and open phenomenon.

Double Descent Learning Curves
Sometimes, in the case where we have over-parameterized our models (more parameters than data), we notice double descent learning curves. This cannot be explained. In the case of the limit where we have more data

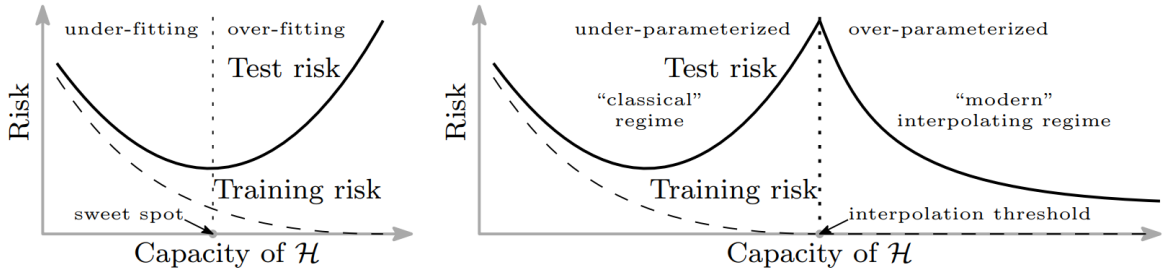


Figure 13: On the left, the underparametrized case where we have more data than parameters (the asymptotic limit) in statistics. On the right, the case where we have more parameters than data, where unexplained things happen.

than model parameters, we always find that generalization to a test set starts to decrease at some point (overfitting). In the case where we have more model parameters than data, we can often observe double descent curves on the test set. This is the result of randomization of excess degrees of freedom, but it cannot be explained. Buhmann argues that it is important to think about these problems because the limit where we have much more data than parameters can in reality often not be guaranteed.

It is interesting to note that regularization methods imply that we do not believe in the correctness of our model, since we don't want it's optimum solution (we consider it overfitting). This is a way to compensate for the fact that our models are deterministic and cannot model randomness, which is fundamental to our reality. In fact, at the very foundation, quantum mechanics its limit to classical mechanics tells us that determinism only arises from averaging....

5.1 Basics of Feedforward Neural Networks

I will be following the notation of the book *Deep Learning* by Goodfellow, Benigo and Courville.
Deep feedforward neural networks, also called simply feed forward neural networks or **Multilayer Perceptrons** (MLPs) are the quintessential neural networks. The goad of a neural network is always to approximate some function f^*, e.g. a classifier $y = f^*(x)$ that output a category y for some input x. A neural network defines a parametric mapping $y = f(x;\theta)$ and learns the parameters θ which gives the best function approximation.
They are called networks because they are typically represented by composing many different functions, called **layers**, together to form $f(x) = f^{(3)}(f^{(2)}(f_{(1)}(x)$. Here, $f^{(1)}(x)$ is the first layer (also called input layer), $f^{(3)}(a)$ is the output layers and $f^{(2)}(a)$ is often referred to as a hidden layer.
Furthermore, the neural networks are called neural because of the choice of the functions $f^{(i)}(a)$, which are loosely guided by neuroscience to consist of many separate units (neurons).

5.1.1 Fully Connected Feedforward Neural Networks

I will now briefly introduce the mathematical formulation of feedforward neural networks composed of only fully connected (dense) layers $f^{(i)}(a)$. This is the classic formulation, represented frequently by sketches such as the one seen in the figure below. let the index $l = 1, \dots$ denote the layer of the neural. Note that each neuron takes as input all the outputs from all the neurons of the previous layer and then computed a single scalar. This is loosely inspired by biology, where neurons often synthesise signals from many other neurons.

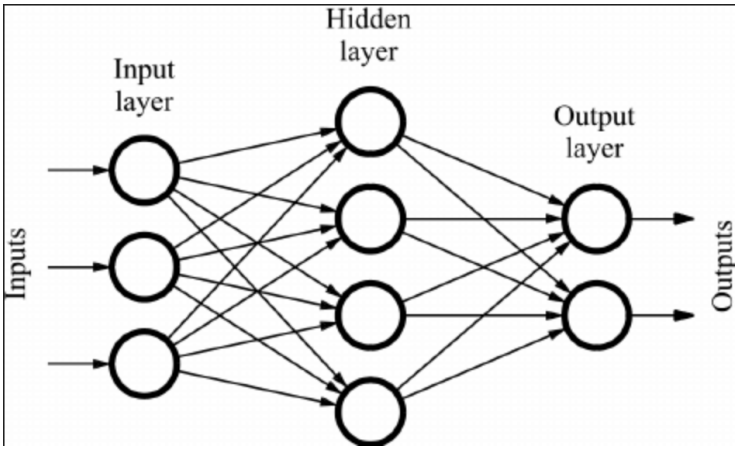


Figure 14: Classic representation of a feedforward neural network with one fully connected (dense) hidden layer.

Let $a^{(l-1)} \in \mathbb{R}^d$ be the vector of outputs, called **activations**, from the previous layer $l-1$ (d is the width of the layer $l-1$). The activations passed to the first layer $a^{(0)} = x$ are simply the input vectors.
Then each neuron i in the layer l will do the following operations:
First, a linear function is applied to the activations:

$$z_i^{(l)} = (w_i^{(l)})^T a^{(l-1)} + b_i^{(l)} = \langle w_i^{(l)}, a^{(l-1)} \rangle + b_i^{(l)},$$

where $w_i^{(l)} \in \mathbb{R}^d$ and $b_i^{(l)} \in \mathbb{R}$ are the weightvector and bias associated with the neuron i in layer l, respectively.
Second, a nonlinear **activation function** $g(z)$ is applied to the output $z_i^{(l)}$.
Thus, at each neuron we do the following operation to compute its activation:

$$a_i^{(l)} = g(z_i^{(l)}) = g\left((w_i^{(l)})^T a^{(l-1)} + b_i^{(l)}\right).$$

Now we can stack all the transposed weight vectors into a matrix and compute directly the activations for all neurons by matrix-vector mutiplication and a subsequent nonlinear function. This gives us the function $f^{(l)} : \mathbb{R}^d \to \mathbb{R}^m$ that defines the layer l. Note that m is the width (dimension) of the layer l and d the width of the layer $l-1$. We find:

$$a^{(n)} = f^{(l)}(a^{(l-1)}) = g^{(l)}(z^{(l-1)}) = g\left((W^{(l)})a^{(l-1)} + b^{(l)}\right),$$

where

$$W^{(l)} = \begin{bmatrix} \cdots (w_1^{(l)})^T \cdots \\ \vdots \\ \cdots (w_m^{(l)})^T \cdots \end{bmatrix} \in \mathbb{R}^{m \times d}, \; b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \vdots \\ b_m^{(l)} \end{bmatrix} \in \mathbb{R}^m,$$

are now the weight matrix and bias vector associated with the layer $f^{(l)} : \mathbb{R}^d \to \mathbb{R}^m$, respectively. The non linear activation function is simply applied to each entry of the activation vector separately.
Note that it is more frequent to use column vector notation here. However, I like row vectors better so all vectors are assumed to be row vectors. Of course, the calculation is exactly the same no matter what.

5.1.2 The fitting power of Neural Networks

The question is, what kinds of functions we can approximate with our nested set of linear functions followed by nonliearities we call a neural network??

The answer is, a lot! There are two fundamental theorems here.
First, **the universal approximation theorem** showed that any Borel measurable function from one finite dimensional space to another can be uniformly approximated using a feed forward neural network with at least one hidden layer and a sigmoid or ReLU activation function, provided the network has "sufficiently many" hidden units (neurons).
This certainly includes any real valued continuous function between finite dimensional spaces.
However, this may quickly become infeasible to compute. In fact, the second theorem by **Montufar** showed that a neural network with more hidden layers can approximate functions using a limited number of neurons that would require exponentially many more neurons when only one hidden layer is provided.
In practice, poor generalization may be expected when we use insufficiently deep and wide architectures (modeling problem). On the other hand, if the model architecture to big w.r.t the data complexity, we get overfitting networks.

Keep in mind that just because a huge range of functions can be approximated by our network does not mean that they can be learned efficiently!!

5.1.3 Backpropagation and Learning

Calculations for backpropagation are done in the exercises. Here, I will only give some comments.

We use a feedforward neural network to produce an output $\hat{y}$ for some input x. The input propagates through the network to produce the output, which is called **forward propagation**. During training, the forward pass will result in a scalar cost $J(\theta)$. This cost function assigns a loss to the output of the network $\hat{y}$ when compared to the data y. The **backpropagation** algorithm (often called **backprop**) allows the information from the cost function to flow backward thorough the network (by the chain rule) to compute the gradient w.r.t to the weights. Then, the gradient can be used by a learning algorithm (Gradient Descent, Adam, etc.) to update the weights. This is the training / learning process of a neural network.

Note that gradient based learning lacks biological plausibility. Recently, Geoffrey Hinton has proposed an alternative training algorithm to update the weights of neural networks...

Gradient based learning (backprop + update rule) is not very different in neural networks than in other machine learning models. Computing the gradients with backprop is just a little bit more complicated than with less complex models.

Stochastic and Minibatch Gradient Descent (SGD)
Grandient descent uses the gradients for the entire dataset batch to update the parmeters. Stochastic gradient descent uses individual examples for updates. In minibatch gradient descent, we instead use a minibtach $m \subset \mathcal{X}$ chosen from the entire dataset $X = \{x_1, \dots, x_n\}$ with corresponding targets y_i. We compute the gradient estimate:

$$\hat{g} \leftarrow +\frac{1}{|m|}\nabla_\theta \sum_{x_i \in m} \mathcal{L}(f(x_i;\theta), y_i),$$

where $\mathcal{L}(f(x_i;\theta), y_i)$ is the loss associated to the predictions w.r.t the labels. Given a **learning rate** α_k, the update rule is simply:

$$\theta \leftarrow \theta - \alpha\hat{g}.$$

To guarantee convergence of SGD, we require that $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$ hold for the learning rates over the training iterations. In SGD, it is very important that the examples are randomized to avoid dependencies of the gradients for certain batches.
There are many more sophisticated update rules, but all are derived from SGD. We can for example add momentum to the update rules or make the learning rate adaptive. The most popular algorithms are SGD, SGD with momentum, RMSProp, RMSprop with momentum, AdaDelta and Adam. The choice of algorithm, at this point, seems to depend largely on the users familiarity with the algorithm (for ease of hyperparameter tuning).

5.1.4 Thoughts on Activation Functions, Output Units and Cost Functions

The choice of cost function is a very important aspect of designing deep neural networks. Fortunately, neural networks are not that different from other parametric models, such as linear models.
In most cases, our parametric model defines some distribution $p(y|x;\theta) = p_\theta(y|x)$ and we simply use the principle of maximum likelihood. This means that we use the cross entropy as our cost function, which is simply the log likelihood:

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}}\left[\log p_{model}(y|x)\right] = -\mathbb{E}_{x,y\sim\hat{p}_{data}}\left[\log p_\theta(y|x)\right].$$

This is the definition of the cross entropy, which is a similarity measure between the distribution defined by our model and the data distribution. It defines a score on the log likelihood to observe the collected data, given the model distribution. The total cost function we use for training is often the cross entropy combined with regularization terms.

Sometimes, we do not wish to learn a full distribution but only some statistic of the distribution. For example, given a regression problem $y = f(x;\theta) + \epsilon$ with the assumption of Gaussian noise $\epsilon \sim \mathcal{N}\left(0,\, \sigma^2\right)$, we find that

$$y \sim \mathcal{N}\left(f(x;\theta),\, \sigma^2\right).$$

We can now let our model function $f(x;\theta)$ simply approximate the mean output for y. In the case of such a setting, we find that the cross entropy cost function actually becomes the mean squared error (MSE):

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}}\left[||y - f(x;\theta)||^2\right] + const.$$

For regression problems, we most often use a linear output function.

In the case where we directly approximate some distribution, such as in binary classification $y \in \{0, 1\}$, we must carefully choose our output function. For example, we can model a binomial distribution, where a probability $\hat{y} \in [0, 1]$ for being in class 1 is assigned to the input. This can be achieved using a **sigmoid** activation function:

$$\sigma(z^{(l)}) = \frac{1}{1 + e^{-z^{(l)}}}.$$

In this case, the cross entropy loss function takes the form:

$$J(\theta) = -\sum_{i=1}^{N} \hat{p}_i \log(p_\theta(y_i|x_i)) = \mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{N} \Big(y_i \log(\sigma(z^{(L)})) + (1 - y_i)\log(1 - \sigma(z^{(L)}))\Big),$$

where N is the number of training examples and $z^{(L)} \in \mathbb{R}$ are the linear activations of the output layer.
The multinomial version of the sigmoid output activation is called the **softmax** function:

$$s(z^{(L)})_i = \frac{\exp z_i^{(L)}}{\sum_j \exp z_j^{(L)}}.$$

Note that the gradient of both these function saturate to zero when the linear activation $z^{(L)}$ gets large. This is another key reason why the cross entropy is often the best choice: we want large and predictable (stable) gradients to update the weights. The log in the cross entropy undoes the exponential of the outputs and significantly stabilizes the gradients. It is important to keep these considerations in mind when choosing outputs and activation functions.
As a general strategy, the principle of maximum likelihood is great: first, choos the parametric model our network should learn, and then use the negative log-likelihood (cross entropy) of the model as a cost function.

Hidden Unit Activation Functions
There is no "best" choice here. ReLU and its variations such as LeakyReLU or ELU are the standard choice and perform very well.

$$\text{ReLU}(z) = \max\{0,\, z\}.$$

5.1.5 The Robbins-Monro Method

Prof. Buhmann presented a method for finding the root x_0 of a monotonically rising, continuous function from noisy samples (x_i, y_i). He argues that essentially, all update rules originate from this type of method to find a optimum of a function, which is essentially finding the zero-point of the gradient of a (hopefully) convex function.

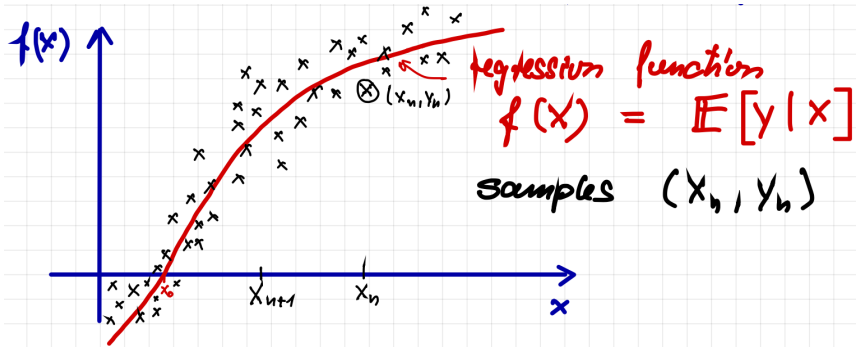


Figure 15: The Robbins-Monro Problem setup.

The update rule proposed is:

$$x_{n+1} = x_n - \alpha_n y_n.$$

The conditions on the step-size α_n is convergence to zero $\lim_{n\to\infty} \alpha_n = 0$, but "slow enough" s.t. $\sum_{n=1}^{\infty} \alpha_n = \infty$. Lastly, the variance must be bounded $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$.

Does this converge? Propose:

$$\lim_{n\to\infty} P(x_n = x_0) = 1 \iff \lim_{n\to\infty} \mathbb{E}\left[(x_n - x_0)^2\right] = 0 \iff \lim_{n\to\infty} x_n = x_0.$$

Proof.
First, write

$$\begin{aligned} x_{n+1} &= x_n - \alpha_n f(x_n) - \alpha_n \gamma_n \\ \gamma_n &= y_n - f(x_n), \end{aligned}$$

Where γ_n is the unbiased ($\mathbb{E}[\gamma_n | y_m]$) noise. We assume $\max_j \mathbb{E}\left[\gamma_j^2\right] \leq \sigma^2$ and $\max_j \mathbb{E}\left[f(x_j)^2\right] \leq b$. We now analyse the expression:

$$\begin{aligned} \mathbb{E}\left[(x_{n+1} - x_0)^2\right] &= \mathbb{E}\left[(x_n - x_0)^2\right] - 2\alpha_n \mathbb{E}\left[(x_n - x_0)(f(x_n) - \gamma_n)\right] + \alpha^2 \mathbb{E}\left[f(x_n)^2 - 2f(x_n)\gamma_n + \gamma_n^2\right] \\ &= \mathbb{E}\left[(x_n - x_0)^2\right] + \alpha^2 \left(\mathbb{E}\left[f(x_n)^2\right] + \mathbb{E}\left[\gamma_n^2\right]\right) - 2\alpha_n \mathbb{E}\left[(x_n - x_0) f(x_n)\right] \end{aligned}$$

We notice that the terms linear in γ_n vanish in expectation, and that the quadratic terms in γ_n and $f(x_n)$ are bounded. We can continue the expansion to reduce the term and find:

$$\mathbb{E}\left[(x_{n+1} - x_0)^2\right] - \mathbb{E}\left[(x_1 - x_0)^2\right] \leq (b + \sigma^2) \sum i = 1^{n-1} \alpha_i^2 - 2\alpha_n \mathbb{E}\left[(x_i - x_0) f(x_i)\right].$$

If the term $\mathbb{E}[(x_i - x_0) f(x_i)]$ is greater than zero for all i, then the rhs of the inequality will to to negative infinity $-\infty$. However, the terms on the rhs are finite, thus the rhs is finitely bound from below. Therefore, the term $\mathbb{E}[(x_i - x_0) f(x_i)]$ must vanish, which gives $\lim_{n\to\infty} P(x_n = x_0) = 1$. □

To make this applicable to neural networks, we observe that the same type of update rule can be used to find the minimum of a convex (loss) function.

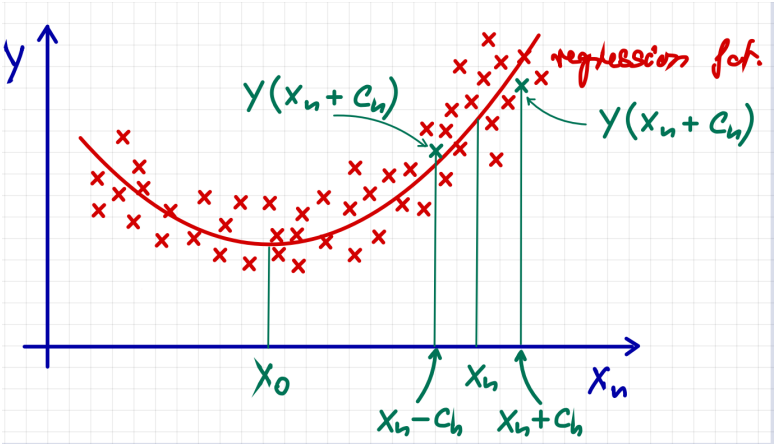


Figure 16: Finding the minimum x_0 of a convex function.

We now use the update rule:

$$x_{n+1} = x_n - \alpha_n \frac{f(x_n + c_n) - f(x_n - c_n)}{2c_n},$$

and for some constraints $\lim_{n\to\infty} \alpha_n = 0$, $\lim_{n\to\infty} c_n = 0$, $\sum_{i=1}^{n} \alpha_n = \infty$ and $\sum_{i=1}^{n} \left(\frac{\alpha_n}{c_n}\right)^2 < \infty$, then we will get convergence again $\lim_{n\to\infty} P(x_n = x_0) = 1$. We can indeed in this case use the Taylor expansion of $f(x_n)$:

$$f(x_n + \Delta x) = f(x_n) + \nabla f(x_n)\Delta x + \frac{1}{2}\Delta x^T H \Delta x,$$

where H is the Hessian, to derive an optimal step size $\alpha = \frac{\nabla f^T \nabla f}{\nabla f^T H \nabla f}$ or to directly derive Newton's rule. For this, we assume the gradient descent update rule $\Delta x = x_{n+1} - x_n = -\alpha_n \nabla f(x_n)$ and optimize the Taylor expansion w.r.t α_n or Δx, respectively. This is a good exercise. Prof. Buhmann then used these updates to illustrate stochastic and minibatch gradient descent updates and momentum (Nesterov). These are explained above already. Note that very often, choosing the update rule is not understood and more like alchemy. Indeed, especially the tradeoff between stochasticity and optimal solutions is poorly explored and very often, the models that perform best are not strictly optimal.

5.2 Deep Generative Modeling

Generative modeling is useful in many applications, such as generatic realistic new samples for artwork, super-resolution, colorization, or to generate time series data for simulations, which is imporant in reinforcement learning. Furthermore, we can learn latent representations and compressions of data types. Ultimately, it is an approach to teach machines "how the world works".
Generative modeling is in a way density estimation. It is unsupervised, but also not in the sense that you can compare generated samples individually to samples from reality. Purely unsupervised learning is trying to learn features from data without the ability to compare it to anything.

In general, the problem is as follows: given the data $\mathcal{X}$, we want to learn $P_{model}(\mathcal{X})$ as close as possible to $P_{data}(\mathcal{X})$. There are two strategies we can follow. We can either explicitly define and solve for $P_{data}(\mathcal{X})$, or we can learn a sampler from $P_{data}(\mathcal{X})$ without explicitly defining it. We will look at the explicit strategy in variational autoencoders, and the explicit strategy in GANs.

5.2.1 Autoencoders

The idea of autoencoders is to take an input and learn and encoding of the data in a latent feature space. We then learn to decode points in the latent feature points back to the original format. Nowadays, both

the encoder and decoder are most often large neural networks. An illustration is shown in the figure below. Such architectures can be used to solve a wide range of problems. For example, we can train an autoencoder

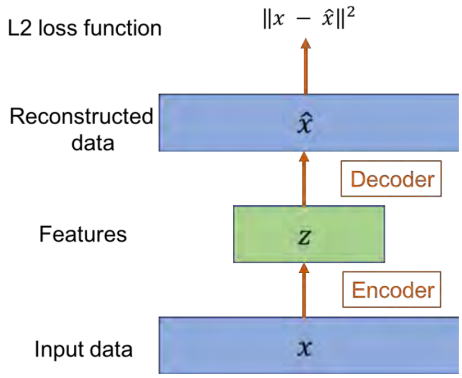


Figure 17: The structure of autoencoders. The encoder and decoder blocks are most often neural networks. The loss is distributed across the weight of both during training.

to learn a compact feature space representation of our data, and then use these features to efficiently train a more simple classifier.

Denoising Autoencoders
In order to learn robust representations of data, denoising autoencoders were introduced. In effect, we want to model to learn invariances in the data and be able to ingore them for encoding. Good representations should be recoverable from noisy or partial inputs. Therefore, we can make use of data augmentation (rotations, blurring etc. for images and things like pitch or speech rate for audio) to generate more data and to make the representations more robust. We can also implement invariances directly into the networks structure, e.g. by using CNNs for images.

Variational Autoencoders (VAEs)
Due to overfitting, the latent space of an autoencoder can be extremely irregular (close points in latent space can give very different decoded data, some point of the latent space can give meaningless content once decoded, ...) and, so, we can't really define a generative process that simply consists to sample a point from the latent space and make it go through the decoder to get a new data. This can be addressed with variational autoencoders.

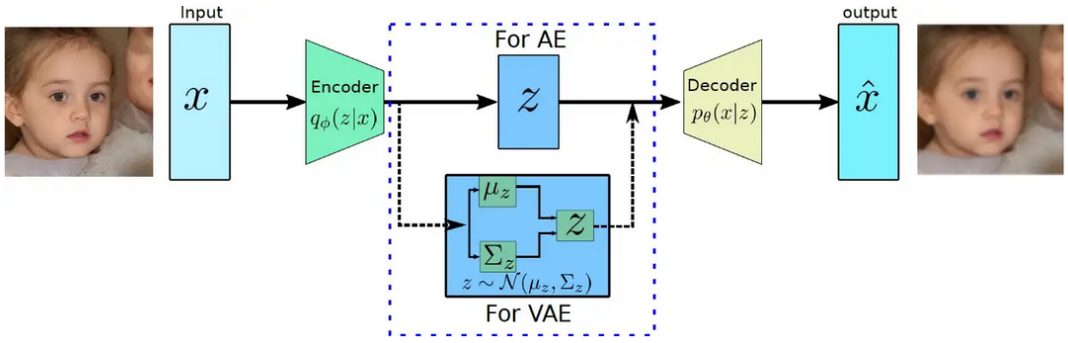


Figure 18: Variational Autoencoders.

Variational autoencoders introduce a probabilistic element to autoencoder architecture. We assume that the training data $X = \{x_1, \ldots, x_n\}$ is generated from an unobserved (latent) representation $\mathcal{Z}$, e.g. an image x is generated from some latent features z. In the latent space, we have the true prior $p(z)$ and the decoder

should represent the true conditional $p_\theta(x|z)$. We want to choose a simple prior, e.g. a Gaussian and the conditional to be complex (and then represent it by a neural network). The problem is that the integral

$$p_\theta(x) = \int p(z)p_\theta(x|z)dz,$$

since it is taken over the latent space. Thus, in addition to the decoder, we add an encoder network that learn an approximation $q_\phi(z|x) \approx p(z)$ parameterized by ϕ. This distribution is chosen to be Gaussian as well, such that $\phi = (\mu_z, \Sigma_z)$. This is useful because it allows us to estimate a lower bound on the data likelihood, which is tractable and can be optimized:

$$\begin{aligned}\log(p_\theta(x)) &= \mathbb{E}_{z\sim q_\phi}\left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x)||p(z)\right) + D_{KL}\left(q\phi(z|x)||p_\theta(z|x)\right)\\ &\geq \mathbb{E}_{z\sim q_\phi}\left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x)||p(z)\right) := \mathcal{L}(x,\theta,\phi),\end{aligned}$$

where we used the fact that the KL divergence is always positive. This derivation is a good exercise (slide 52, lecture 9). The first term is the log likelihood of the data given the latent variable z and therefore represent the reconstruction loss. It will be learned by the decoder. The second term is the KL divergence between the parameterized distribution of the decoder and the prior on z. Both are simple Gaussians and thus the KL divergence has a nice closed form solution. We choose $z \sim p(z) = \mathcal{N}\left(0, \sigma^2\mathcal{I}\right)$. This is OK as long as the decoder is expressive enough (i.e. has enough non-linearities). This choice can be interpreted as a kind of regularization to ensure a more structured latent space. We want to make the decoder very expressive, i.e. able to decode the latent variables, while still making sure that the distribution of points across the latent space stays compact and close to our prior. If we do not do this, then the network might place points simply far apart in the latent space and run into overfitting. This tradeoff is natural for Bayesian inference problem and express the balance that needs to be found between the confidence we have in the data and the confidence we have in the prior.

The Kullback-Leibler Divergence. The KL divergence D_{KL} is a similarity measure between probability distributions (just like the cross entropy). It can be interpreted as the expected excess cost we incur from using a distribution Q when the true distribution is P

$$D_{KL}(P||Q) := \mathbb{E}_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right] = \int_{\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)}dx \text{ or } \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)}.$$

One problem we may face when working with the KL divergence is that it can take values in $(0,\infty)$. I can therefore make cost functions that include it unstable. In the following, we will see what methods can be used to avoid the KL divergence as a cost.

Hierarchical VAEs (HVAEs)
Some long range dependencies (e.g. across images) can be hard to encode with a VAE. HVAEs aim to improve the quality of generated samples by stacking more layers of representations. Hierarchical VAEs propose to insert intermediate representation states in the network.

Lets look at the case with two representation layers z_1 and z_2. The decoder takes z_1, produces a new representation z_2 and only then produces the output x. Similarly for the encoder: $x \to z_1 \to z_2 \to z_1 \to \hat{x}$. We can similarly derive an ELBO for the data likelihood in that case:

$$\log p(x) \geq \mathbb{E}\left[\log p(x|z_1\right] - \mathbb{E}\left[\log\frac{q(z_1|x}{q(z_1|z_2}\right] - \mathbb{E}\left[\log\frac{q(z_2|z_1}{p(z_2)}\right],$$

where we again use a simple Gaussian prior on $p(z_2)$. Note that the second and third term are KL divergences, which can lead to even more unstable training. How can we keep the distributions from becoming too disjoint during training?
We introduce a *top-down shared model*. The decoding is done as usual: $z_2 \sim \mathcal{N}\left(0, \sigma^2\mathcal{I}\right)$ and $z_1 \sim \mathcal{N}\left(\mu_{\theta_2}(z_2), \Sigma_{\theta_2}(z_2)\right)$ and $x \sim \mathcal{N}\left(\mu_{\theta_1}(z_1), \Sigma_{\theta_1}(z_1)\right)$, which are all our simple priors. The covariances are diagonal matrices (sometimes even just one parameter).

The encoding is done as follows: instead of producing a full mean and covariances at each step, the encoder neural network at each step produces a relative mean and variance $\Delta\mu_{\theta_i}$ and $\Delta\Sigma_{\theta_i}$. We then draw at each representation from $z_i \sim \mathcal{N}\left(\mu_{\theta_i}(z_i) + \Delta\mu_{\theta_i},\, \Sigma_{\theta_i}(z_i) \cdot \Delta\Sigma_{\theta_i}\right)$. The trick is then to keep the changes small. This can be extended to n intermediate steps.

Instead of having to always sample at the representations during the encoding, we go straight down to the deepest encoding and then sample straight back up to the image. The KL divergences will look different (see slides 0-16 lecture 11A). Changes in the decoder distribution propagate directly to the encoder, which avoids sudden changes in the KL. For Gaussians, the KL divergence will have a very simple form. For visualizations, look at the slides.

5.2.2 Diffusion Models

Diffusion models aims to represent a very complex high dimensional distribution. We again use a structure similar to the HVAE, by chaining together representations, where the deepest representation is a isotropic and simple Gaussian. At each intermediary step, the representation becomes slightly more complex. So we learn a step by step transformation to a highly complex distribution.
The encoding is drastically more simple and works as follows: at each step, we add a little bit more Gaussian noise $x \to z_1 \to \ldots \to z_n$, where

$$\begin{aligned} z_i &= \beta_i z_{i-1} + \beta_i \epsilon, \text{ where } \epsilon \sim \mathcal{N}\left(0,\, \sigma^2 \mathcal{I}\right) \text{ and thus} \\ q(z_i|z_{i-1}) &= \mathcal{N}\left(z_i|\beta_i z_i,\, \beta_i \mathcal{I}\right). \end{aligned}$$

Usually, β_i is fixed. This way, encoding (e.g. sampling a deep representation of an image) is much easier. By using only Gaussian, we find that the forward decoding posterior is also a Gaussian:

$$q(z_{i-1}|z_i, x) = \mathcal{N}\left(z_{i-1}|\tilde{\mu}_i(z_i, x),\, \tilde{\beta}_i \mathcal{I}\right),$$

where β_i and $\tilde{\mu}_i$ is defined on slide 21 of lecture 11A. We approximate this mean by a neural network. The variance stays a fixed constant. The ELBO is now even simpler (see slide 22). Indeed gradients can be computed simply over the square distance between the mean vectors. Yet, the results are very good!

5.2.3 Generative Adversarial Networks (GANs)

We now look at the implicit approach to density estimation and give up on trying to explicitly finding an distribution to describe the data. Instead, we want to design a sampler to generate samples from $P_{data}(\mathcal{X})$ a potentially high dimensional and very complex distribution.

Key Idea. We sample from a very simple distribution (Gaussian Noise) and learn a complex transformation of the data via a neural network to obtain the distribution of the training data. Training is done as a supervised two player game, in which an adversary learns to generalize from training data.

Training
Training is setup as a two player minimax (maxmax) game. Given the discriminator $D(x)$ and the generator $G(x)$, we aim to find

$$\min_G \max_D \left\{\mathbb{E}_{x\sim P_{data}}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_{noise}}\left[\log(1 - D(G(z)))\right]\right.\!.$$

The discriminator wants to maximize this objective such that $D(G(z))$ is close to 0 and $D(x)$ is close to 1. At the same time, the generator wants to minimize the objective such that $D(G(z))$ is close to 1. Note that it has been found to be better to run gradient ascent on the generator with the function $\mathbb{E}_{z\sim P_{noise}}\left[\log D(G(z))\right]$. This is because this cost, compared to the above, points the generator more strongly away from regions where it is bad (see slide 64, lecture 9). Therefore, we aim to do the following:

- At each training iteration, first train the discriminator for k separate steps to recognize the fakes by gradient ascent on
$$\mathbb{E}_{x\sim P_{data}}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_{noise}}\left[\log(1 - D(G(z)))\right].$$
Simply sample minibatches of noise and training data to feed the networks.

- At each training iteration, after having trained the discriminator, update the generator by using gradient ascent on
$$\mathbb{E}_{z\sim P_{noise}}\left[\log D(G(z))\right].$$
Here we only need to generate minibatches of noise samples.

Network Architectures. Usually, the generator is an up-sampling network with fractional convolutions and the discriminator is just a deep CNN. We use BatchNorm in both cases and ReLU or LeakyReLU as activations.

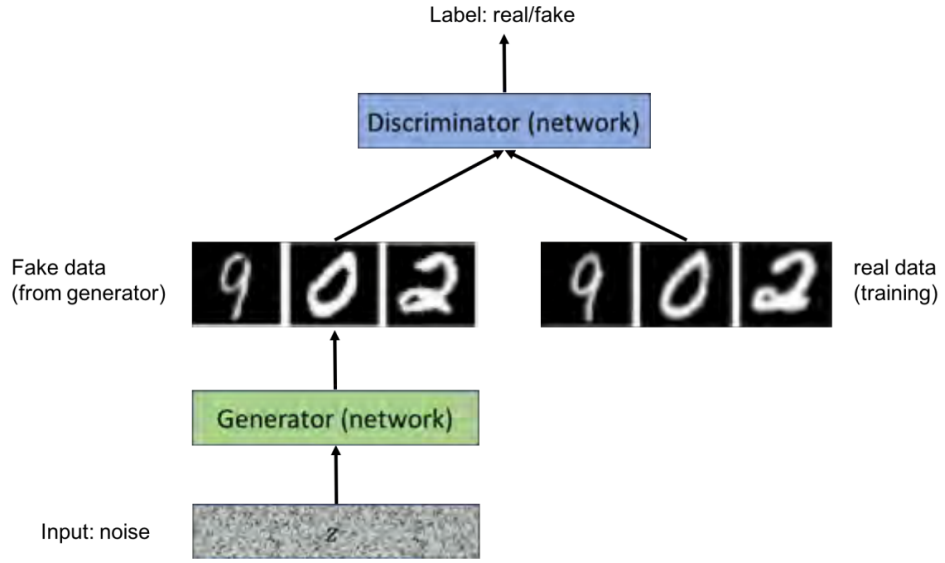


Figure 19: Generative Adverarial Networks (GANs) training setup. We aim to learn a transformation from Gaussian noise to the training distribution with the generator.

Issues. Note that while GANs have shown impressive results, there are still a few areas to improve, e.g. GANs tend to be unstable in training (better cost functions required) and very susceptible to adversarial injections.
The problem with the strategy is that even for the optimal discriminator, the generator will optimize a loss function based on two KL divergences. A proposed solution is to replace the KL divergence by a new metric. By using the **Wasserstein distance** $WG_r(p,q) < \infty$, gradients will become more stable and we do not get vanishing gradients. This will make gradient descent more stable. Such models are called Wasserstein-GANs (W-GANs).

Domain Invariant Representation Learning
We saw that sometimes unwanted biases are encoded in the representation. This can be address this with **conditional GANs**. We are given a dataset with triples of domain, samples and labels $D = \{(w_1, x_1, y_1)\ldots,(w_n, x_n, y_n)\} \in \mathcal{W}\times\mathcal{X}\times\mathcal{Y}$. The domain space can encode features we do not want the neural network to lean (e.g. biases like a face with glasses). We want to learn an encoder $E : \mathcal{X}\to\mathcal{Z}$, which learns a representation of the samples, and a classifier $F : \mathcal{Z}\to[0,1]^{\mathcal{Y}}$, which learns a distribution over labels. We use maximum likelihood for training, by minimizing the cross entropy:

$$\min_{E,F}\mathbb{E}_x\left[CE(p_{x|y},\hat{p}_{E(x)})\right] =: \mathcal{L}_c(E,F).$$

Lastly, we don't want our representations to contain information from the domain space. Thus, we also train adversarially a discriminator $D : \mathcal{Z}\times\mathcal{Y}\to[0,1]^{\mathcal{W}}$, which aims to learn the domain biases from the representations. We use again as a loss function an average over examples

$$\min_{D}\mathbb{E}_{x,y}\left[CE(p_{w|x,y},\hat{p}_{E(x),y})\right].$$

The encoder is trained on the combined loss function to minimize its own loss function and maximize the loss function of the discriminator.
Hopefully, the encoder will learn a representation of the data that does not use the information of the domain space (the biases). In training, the decoder will likely first learn a representation that uses the highly informative biases. Then, the punishment from the discriminator will kick in and force the encoder to shift the representations, such that they are not biases with domain information.

A different approach to extract domain invariant representations is **maximum-mean discrepancy (MMD)**. We again aim to learn an encoder $E : \mathcal{X} \to \mathcal{Z}$, which learns a representation of the samples, and a classifier $F : \mathcal{Z} \to [0,1]^{\mathcal{Y}}$, which learns a distribution over labels. Now however, we simply directly modify the loss function above with an extra regularization term

$$\mathcal{L}(E,F) = \mathcal{L}_c(E,F) + \mathcal{L}_{MMD}(E),$$

which quantifies to what degree the samples from one domain look like samples from another domain. It computes an average over all example pairs from different domains (empirical estimate). Formally, the MMD is difference of the expected value between two functions over the distribution, where we use all continuous functions:

$$MMD(p,q) = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_{x \sim p}\left[f(x)\right] - \mathbb{E}_{y \sim q}\left[f(y)\right] \right),$$

which is of course completely intractable. Instead, it can be shown that it sufficient to approximate the continuous functions by the unit sphere in a *reproducing kernel Hilbert space* (RKHS) $\mathcal{H}_0 \approx \mathcal{F}$. This is shown in the paper. In slides 25-33 of lecture 11B, we show that the expectations in the MMD can be approximated with inner products. If we choose our RHKS $\mathcal{H}$ as the space of all polynomials of the form $f(x) = \sum_i w_i x^i$ such that $\sum_i w_i^2 < \infty$ and define the inner product on it as $\langle f, g \rangle = w_f^T w_g$ with $w_f = (w_1, \ldots)$. Furthermore, we can define a norm on it $||f|| = \sqrt{\langle f, f \rangle}$, which gives us a valid RHKS (apparently??). Indeed we can now rewrite the expectation over x as an inner product $\mathbb{E}_{x \sim p}\left[f(x)\right] = \langle f, \mu_p \rangle$, where $\mu_p = \sum_i \mathbb{E}\left[x^i\right] x^i$ is a polynomial that characterizes the distribution $p(x)$. Thus means that the MMD becomes:

$$MMD(p,q) = \sup_{f \in \mathcal{H}_0} \langle f, \mu_p - \mu_q \rangle = ||\mu_p - \mu_q||^2 = \mathbb{E}_{x, x_1, x_2 \sim p, y, y_1, y_2 \sim q} \left[\sum_i x_1^i x_2^i - 2 \sum_i x^i y^i + \sum_i y_1^i y_2^i \right],$$

which only contains inner products of the data and can be kernelized. The empirical MMD is then very simple:

$$\hat{MMD}(\{z_1, \ldots, z_m\}, \{z'_1, \ldots, z'_n\}) = \frac{1}{m^2} \sum_{i,j \leq m} k(z_i, z_j) - \frac{1}{mn} \sum_{i \leq m, j \leq n} k(z_i, z'_j) + \frac{1}{n^2} \sum_{i,j \leq n} k(z'_i, z'_j),$$

where the sets z and z' come from different domains. This gives us a measure on how likely it is that two batches of examples from different domains came from the same distribution, which should be minimized because we want all examples to look like they come from one distribution. This is amazing because we do not require adversarial training and can simply measure how much the neural network encodes the domain information into the representations z.

6 Non-parametric Bayesian Methods

We return to the problem of density estimation. Given a data sample $X = \{x_1, \dots, x_n\}$, we want to build and estimator for the distribution the data came from.

6.1 Gaussian Mixture Models with infinitely many Clusters

We again use GMMs to motivate the theory of Non-parametric Bayesian methods. Recall the maximum likelihood approach, which is the frequentist approach. Howerver, MLE requires us to choose the number of clusters in advance, which gives us a parametric model (with finitely many parameters). Now, we are interested in fiting GMMs where the number of clusters is not known. This requires a **non-parametric Bayesian Inference (BI)** approach!

6.1.1 BI for a single cluster in $\mathbb{R}^d$

Recall the basic BI approach:

- 1) Impose a prior on the parameter θ by choosing a distribution (the prior) $q(\theta)$ over Θ.
- 2) Choose a parametric model $\mathcal{H}$ to define the likelihood $p(X|\theta) \in \mathcal{H}$.
- 3) Update the distribution $q(\theta)$ to find the posterior $p(\theta|X) = \frac{p(X|\theta)q(\theta)}{p(X)} = \frac{p(X|\theta)q(\theta)}{\int q(\theta)p(X|\theta)d\theta}$.
- 4) Estimate and event of interest ϕ - $p(\phi|X) = \int p(\theta|X)p(\phi|\theta)d\theta$.

For single clusters in $\mathbb{R}$, this is very tractable and easy when both the prior and the likelihood are Gaussian. An example is shown in lecture 12 (good exercise). If we cannot choose just Gaussian priors, it is advantageous to choose conjugate priors for the likelihood.

For a single cluster in $\mathbb{R}^d$, the Bayesian approach is already not so simple. I has been established, that the normal inverse Wishart $\mu, \Sigma \sim NIW(m_0, K_0, v_0, S_0)$ distribution is a conjugate prior for the multivariate Gaussian likelihood $p(x|\theta) \sim \mathcal{N}(\mu, \Sigma)$. This means that the posterior is also a normal inverse Wishart. The predictive posterior is then analytically tractable.
If we are not lucky enough to find conjugate priors, we can do **approximate BI with semi-conjugate priors and Gibbs sampling**. For semi-conjugate priors, the posterior is is still intractable, but the conditional posteriors $\mu|\Sigma, X$ and $\Sigma|\mu, X$ are tractable. We note that the predictive distribution is approximated by an expectation:

$$p(\phi|X) = \int p(\theta|X)p(\phi|\theta)d\theta \approx \frac{1}{M}\sum_{t \leq M} p(\phi|\theta_t).$$

We can now draw samples of parameters with **Gibbs sampling**. Since the conditional posteriors $\mu|\Sigma, X$ and $\Sigma|\mu, X$ are tractable, we do chain like sampling

$$\mu_0 \to \Sigma_0 \to \mu_1 \to \Sigma_2 \to \dots \to \mu_n \to \Sigma_n.$$

More technically, in Gibbs sampling, we choose the next parameter to sample at random (random order of parameter sampling). To get a good approximation, we leave the first examples aside. It can be shown that these samples look like samples from the true posterior in the asymptotic limit.

6.1.2 BI for finitely many clusters in $\mathbb{R}^d$

For finitely many clusters in $\mathbb{R}^d$, we cannot find conjugate priors for the GMM likelihood. However, if we impose a normal inverse Wishart on each mean μ_k and covariance Σ_k and also impose a Dirichlet distribution for the mixing coefficients $\pi \sim Dir(\alpha_0, \dots \alpha_n)$, then all the priors are semi-conjugate (have conditional tractable solutions).

The Dirichlet Distribution is a multivariate generalization of the Beta β distribution. It is a distribution over categorical distributions for k objects. I.e. for k separate objects, the Dirichlet distribution is a

distribution over all possible distributions π over the k objects. The probability to draw a distribution π is $Dit(\pi|\alpha) = \frac{1}{B(\alpha)\Pi_{i\leq k}}$, where $B(\alpha)$ is a function involving Gamma functions $\Gamma(\alpha_i)$.

We could now do Gibbs sampling to approximate the posterior induced by the above priors. However, this is very unstable, which is why we will look at a more stable sampling method - **collapsed Gibbs sampling**. First, we look at our GMM model as a Bayesian network. This is shown in the figure below.

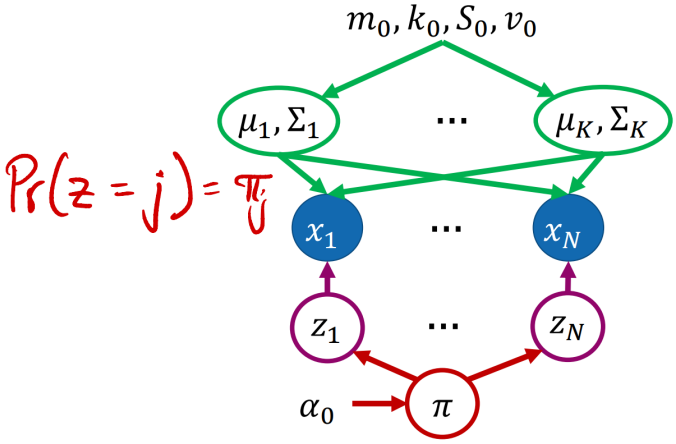


Figure 20: Bayesian Network describing a GMM. Note that fully colored nodes are observed, while white nodes are not observed. The parameters on top are inputs to the normal inverse Wishart. Recall $z_i = k$ iff the point i comes from the cluster k.

d-separation. A path of nodes is d-separated if there are 3 consecutive nodes in it with a special form (e.g. not observed or observed followed by an observed node followed again by a not observed or observed node). The forms are shown on slide 23 of lecture 12. This is important because two random variables A and B are conditionally independent given the observed nodes X if every path of nodes between A and B is d-separated $p(A, B|X) = p(A|Z)p(B|X)$.

If we assume that the z values in our graph are also observed, then we can drop the π dependence of the mean and covariance and get $p(\mu_k, \Sigma_k|z, \pi, X) = p(\mu_k, \Sigma_k|z, X)$. Furthermore, the posterior of the distribution π is independent of all the means and covariances and depends only on the observed z: $p(\pi|z, \mu_k, \Sigma_k, \ldots) = p(\pi|z)$. Lastly, if we are given all the data points except one data point x_i, denoted by X_{-i}, then we find that $p(X_{-i}|z) = p(X_{-i}|z_{-i})$.
Now some of the green arrows become redundant and we can design a new network where connections from (μ_k, Σ_k) only go to all the points x_i with $z_i = k$.

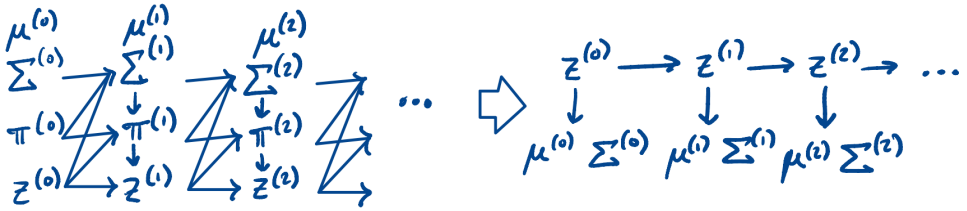


Figure 21: Collapsed Gibbs sampling. Instead of sampling all the variables in a complex procedure, we sample the collections z first and then the rest conditioned on them

Collapsed Gibbs sampling.
We can now instead of doing the complex Gibbs sampling process with all the variables π, z, μ, Σ, we first just sample a set of $\{z_i\}$ and then use these to sample the μ_k, Σ_k. By the **Rao-Blackwell theorem**, this converges faster and produces better estimates.

First, how do we sample the z? We need to compute the probability of the example i coming from cluster k, given that we have seen all the other z:

$$p(z_i = k|z_{-i}, X) \propto p(z_i = k|z_{-i})p(X|z_i = k, z_{-i}) \propto p(z_i = k|z_{-i})p(x_i|\{x_j : j \leq N, i \neq j, z_j = k\}).$$

The last expression is analytically tractable. This is an important exercise.

We now look at the corollary of the **Rao-Blackwell theorem** that justifies the convergence of this algorithm. Let θ, Z be random variables with an intractable joint distribution $p_{\theta,Z}$. Let $f(\theta, Z)$ be a statistic of interest and we want to compute its expected value. In our case this expected value is $p(\phi|X) = \int p(\theta|X)p(\phi|\theta)d\theta$ where θ are the parameters of the GMMs without Z. We saw already that we can approximate this with samples from the distribution $\theta_t, z_t \sim p_{\theta,Z}$, which we find with collapsed Gibbs sampling. In the collapsed case, all the z are sampled first and only one $\theta^{(i)}$ is drawn form the $z^{(i)}$ samples (note the superscript, we are talking about collections of z). We can show that the variance of the z when using the collapsed approach is actually lower than the variance of the samples obtained with standard Gibbs sampling:

$$\mathbb{V}_Z \left[\mathbb{E}_\theta \left[f(\theta, Z)|Z\right]\right] \leq \mathbb{V}_{\theta,Z} \left[f(\theta, Z)\right].$$

Look at the proof as an exercise (using Jensen's inequality).

6.1.3 BI for non-parametric GMMs

Although there exists a conjugate prior for this problem, it is analytically intractable. We instead impose semi-conjugate priors on the parameters. The $\mu_k, \Sigma_k \sim NIW$ are again sampled from a normal inverse Wishart distribution. The $\pi \sim GEM(\alpha)$, where this GEM process produces distributions over all natural numbers (since we have infinite clusters). As an intuition, the higher the α, the more clusters the GEM process produces.
Alternatively, it can be shown that this prior is equivalent to a **Chinese Restaurant Process** (CRP) over the $z \sim CRP$. We will look at how z_i can be drawn from the CRP. First, we must open a restaurant with infinitely many tables (clusters). First, we place z_1 at table $k = 1$. Then, for the next customers z_i for $i = 2, \ldots, n$, the probability that this customer will joining a table k that is not empty is given by

$$\frac{\#\text{customers at table } k}{\alpha + i - 1},$$

and the probability that he will join the leftmost empty table (new cluster) is given by

$$\frac{\alpha}{\alpha + i - 1}.$$

Note that α determines how social the customers are, i.e. how likely it is that points are assigned to completely new clusters. The number of tables after drawing n samples is $O(\alpha \log n)$.
Note that the $z1, \ldots, z_n$ are not independent. However, they are exchangeable, which means that any permutation of the values will have the same distribution. This is a useful exercise.

Finally, we saw that actually the joint distribution of all the parameters $\mu_k, \Sigma_k, z \sim DP$ follows a **Dirichlet Process** (DP), which is a distribution over distributions (probability measures). We now sample the collection of z.

Finally, we saw that by the DeFinetti-Hewitt-Savage theorem any exchangeable sequence of random variables admits a mixtures model (that can be fitted to them).

7 Probably Approximately Correct (PAC) Learning

Mathematics and computational models have achieved amazing results which have allowed us to understand many things about the world. The immediate questions that arise are

- Can computers compute everything?
- Can formal logic prove any statement?

The answer to both questions is NO (Turing's halting problem, Gödel's incompleteness theorem). Similarly, we can ask for learnability: can we learn any function to arbitrary precision with high probability? And if something is learnable, how well can we learn it by empirically minimizing some cost function?
The field of PAC learning is a sub-field of statistical learning theory that concerns itself with these questions.

7.1 Problem Setup and Definitions

We consider the learning problem in the following. We are given some examples drawn from a concept

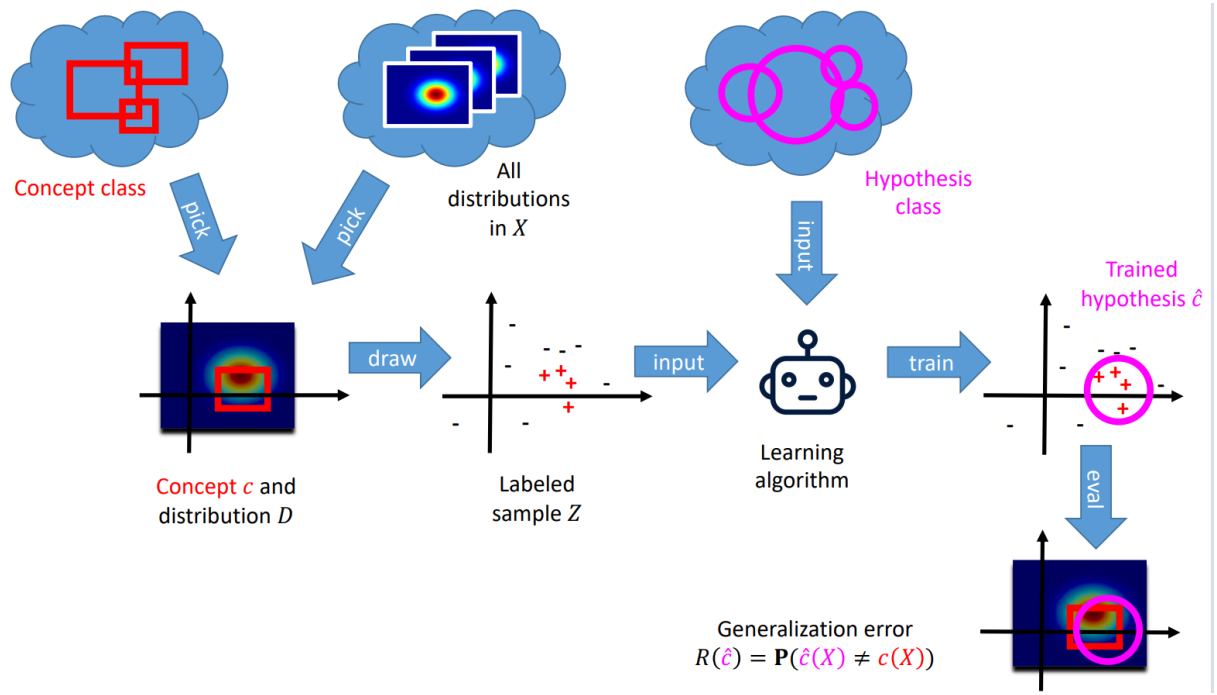


Figure 22: The learning problem.

(ground truth) according to some distribution picked by mother nature. We would then like to learn some hypothesis $\hat{c}$ that gets close to the true concept c with minimal generalization error $\mathcal{R}(\hat{c}) = P(\hat{c}(X) \neq c(X)$. The generalization error is the probability that the learned hypothesis is not the true concept. We are given the dataset X with true labels $y = c(x)$ (provided by some oracle) represented in an instance space $\mathcal{X}$ accessible to the learner. Both the concept class $\mathcal{C}$ and hypothesis class $\mathcal{H}$ are most often very general classes of functions. Model selection is a hard problem, since the concept class is not know. Furthermore, the learner can generally not compute the generalization error. Instead, we compute the empirical generalization error over some test dataset

$$\hat{\mathcal{R}}(\hat{c}) := \frac{1}{n} \sum_{i \leq n} \mathbf{1}_{\hat{c}(x_i) \neq c(x_i)},$$

which is an unbiased estimator of the generalization error.

Learning Algorithms and Learnability

A learning algorithm $\mathcal{A}$ is an algorithm that receives as input a labeled sample $\mathcal{Z} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ with $y_i = c(x_i)$ and outputs some hypothesis $\hat{c} \in \mathcal{H}$.

We say that a learning algorithm $\mathcal{A}$ can learn a concept $c \in \mathcal{C}$ if, given as input a sufficiently large sample, it outputs a hypothesis $\hat{c} \in \mathcal{H}$ that generalizes well with high probability with high probability.
More formally, a learning algorithm $\mathcal{A}$ can learn a concept $c \in \mathcal{C}$ if there is a polynomial function *poly* such that

- 1) for any $c \in \mathcal{C}$ and
- 2) for any distribution $\mathcal{D}$ on $\mathcal{X}$ and
- 3) for any $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$,

if $\mathcal{A}$ receives a sample $\mathcal{Z}$ of size $n \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(c))$, then $\mathcal{A}$ outputs $\hat{c} \in \mathcal{C}$ such that:

$$P_{\mathcal{Z} \sim \mathcal{D}^n} \left(\mathcal{R}(\hat{c}) \leq \epsilon \right) \geq 1 - \delta.$$

$size(c)$ denotes the size of the representation of the concept c, e.g. the nr. of parameters of a neural net or the depth of a decision tree.

We say that the concept class $\mathcal{C}$ is **PAC learnable** from a hypothesis class $\mathcal{H}$ is there is an algorithm $\mathcal{A}$ that can learn any concept in $\mathcal{C}$.
Furthermore, a concept class $\mathcal{C}$ is **efficiently PAC learnable** if $\mathcal{A}$ runs in time polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Here ϵ is called the *error parameter* and δ is called the *confidence parameter*. Note that no assumptions on the distribution of instances in the instance space $\mathcal{X}$ are made.

Example: The axis-aligned rectangles
Let $\mathcal{C}$ be the concept of all axis aligned rectangles in $\mathbb{R}^2$. We show that $\mathcal{C}$ can be learned from $\mathcal{H} = \mathcal{C}$.
Consider the algorithm $\mathcal{A}$ that outputs the smallest rectangle $\hat{R}$ containing all the positively labeled points of the data $\mathcal{X}$. We show that $\mathcal{A}$ can learn any $R \in \mathcal{C}$. The proof for this should be done on paper to fully grasp the example. I will only give an outline and not go into the details here.

Proof Sketch. We aim to show that there exists a polynomial s.t. if $\mathcal{A}$ is given a dataset with size greater than polynomial, it will generalize well. We do so in three steps:

- Define the $\hat{R}IG$ event, which is the event where our prediction $\hat{R}$ is separated from the true R by at region with probability mass smaller than ϵ under our distribution.
- Crucially, show that the probability of $\hat{R}IG$ happening is bounded from below $Pr(\hat{R}IG) \leq 1 - 4e^{n\epsilon/4}$. Hint: $(1-x)^n \leq e^{xn}$.
- Lastly, define the polynomial cleverly as $poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(R) = 4) = \frac{4 \cdot 4}{\epsilon\delta}$. and connect the chain of inequalities by showing that $1 - 4e^{n\epsilon/4} \geq 1 - \delta$.

Theorem
If we know the true concept class s.t $\mathcal{H} = \mathcal{C}$, an algorithm $\mathcal{A}$ that overfits the training set $\mathcal{Z}$ perfectly for $|\mathcal{Z}| = n \geq poly = \frac{1}{\epsilon}(\log|\mathcal{H}| + \log\frac{1}{\delta})$, i.e. $\hat{\mathcal{R}}(\hat{h}) = 0$, then the algorithm will generalize well, i.e. $Pr(\mathcal{R}(\hat{h}) \leq \epsilon) \leq 1 - \delta$.

Proof Sketch. The proof is very illustrative and should be done as an exercise. We first show that

$$n \geq poly = \frac{1}{\epsilon}(\log|\mathcal{H}| + \log\frac{1}{\delta}) \iff |\mathcal{H}|(1-\epsilon)^n\delta,$$

using rearranging and a Taylor expansion. Then we demonstrate that

$$\hat{\mathcal{R}}(\hat{h}) = h \implies Pr(\mathcal{R}(\hat{h} \geq \epsilon) \leq |\mathcal{H}|(1-\epsilon),$$

using the product rule and the definition of the generalization error.

7.1.1 The general Stochastic Setting

In general, an instance's label is not determined by the underlying concept (we have noise). This is modeled with a distribution $\mathcal{D}$ on an extended instance space $\mathcal{X} \times \{0,1\}$, which reflects the fact that two instances with the same features may have different labels. The training set is again a sample $\mathcal{Z} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ with $y_i = c(x_i)$ from $\mathcal{D}$. The goal is then to find a hypothesis $\hat{c} \in \mathcal{H}$ with small generalization error:

$$\mathcal{R}(\hat{c}) = P_{\mathcal{X},\mathcal{Y}\sim\mathcal{D}}(\hat{c} \neq y) = \mathbb{E}_{\mathcal{X},\mathcal{Y}\sim\mathcal{D}}\left[\mathbf{1}_{\hat{c}\neq y}\right].$$

If the Bayes optimal classifier is not in the hypothesis class $\mathcal{H}$, then it is impossible to obtain $\mathcal{R}(\hat{c}) \leq \epsilon$ for all $0 < \epsilon < \frac{1}{2}$. Instead, we must aim to find the best solution given the hypothesis class:

$$\mathcal{R}(\hat{c}) - \inf_{c\in\mathcal{H}} \mathcal{R}(\hat{c}) \geq \epsilon.$$

We therefore now have anew definition of learnability:
An algorithm $\mathcal{A}$ can learn a concept $c \in \mathcal{C}$ if there is a polynomial function *poly* such that

- 1) for any $c \in \mathcal{C}$ and
- 2) for any distribution $\mathcal{D}$ on $\mathcal{X} \times \{0,1\}$ and
- 3) for any $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$,

if $\mathcal{A}$ receives a sample $\mathcal{Z}$ of size $n \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(c))$, then $\mathcal{A}$ outputs $\hat{c} \in \mathcal{C}$ such that:

$$P_{\mathcal{Z}\sim\mathcal{D}^n}\left(\mathcal{R}(\hat{c}) - \inf_{c\in\mathcal{H}} \mathcal{R}(\hat{c}) \leq \epsilon\right) \geq 1 - \delta.$$

This implies that we must leave the nice assumption that we can actually figure out anything and instead aim to derive upper bounds on certain methods that show, that when we are given enough data, we can get close to the optimal $\inf_{c\in\mathcal{C}} \mathcal{R}(c)$. In the last week, Prof. Buhmann showed us a proof on the lower bound of this quantity for the **empirical risk minimizer** if we have a finite concept class. I will show some of the proofs I consider important below. Buhmann notes that the discrepancy between the empirical risk and the expected risk is the reason why he no longer believes in optimization... more on this later.

VC dimension and shattering

Shattering. A set A of instances can be *shattered* by a concept class $\mathcal{C}$ if for every subset $S \subseteq A$, there is a concept $c \in \mathcal{C}$ such that $S = c \cap A$. This means that for every subset $S \subseteq A$, there is a concept that can capture exactly only that subset and no other elements of A. Note that S can be the empty set $S = \varnothing \subseteq A$.

Examples. The set of two points in $\mathbb{R}^2$ can be shattered by the class of axis aligned rectangles.
No set of three points in $\mathbb{R}$ can be shattered by the set of intervals.
Sets of points in $\mathbb{R}^2$ that are not colinear can be shattered by the set of axis aligned rectangles.

The VC dimension is the size of the largest set $S \subseteq \mathcal{X}$ that we can shatter in our instance space. Note that by leaving away points, for any size $n \leq |S|$ we can also find a set $S' \subseteq \mathcal{X}$ with $|S'| = n$ that can be shattered. We can express this as an algorithm:

- Starting from $VC(\mathcal{C}) = 0 =: n$
- while $\exists S \subseteq \mathcal{X} :\ |S| = n + 1$ and S can be shattered by $\mathcal{C}$, assign n$n \leftarrow n + 1$.
- Return the VC dimension $VC(\mathcal{C}) = n$.

The VC dimension can be used to measure the complexity of the concept class $\mathcal{C}$ on the instance space $\mathcal{X}$ and proof useful upper bounds on the generalization error of learning algorithms. An example is, if $\mathcal{C}$ has finite VC-dimension:

$$P\left(\mathcal{R}(\hat{c}) - \inf_{c\in\mathcal{H}} \mathcal{R}(\hat{c}) > \epsilon\right) \leq 9n^{VC(\mathcal{C})} \exp -\frac{n\epsilon^2}{32} \to 0, \text{ as } n \to \infty.$$

Some Proofs and Notes from the last Lecture
First, the Vapnik Chervonenkis Inequality. The uniform convergence (the sup) across classifiers is sufficient and necessary for classification and regression. Indeed, giving only one "optimal" solution for an optimization

Theorem 1 *(Vapnik Chervonenkis 1974)* $(c^\star = \text{argmin}_{c\in\mathcal{C}}\,\mathcal{R}(c))$

$$\begin{aligned}\mathcal{R}(\hat{c}_n^\star) - \inf_{c\in\mathcal{C}}\mathcal{R}(c) &= \mathcal{R}(\hat{c}_n^\star) - \hat{\mathcal{R}}_n(\hat{c}_n^\star) + \hat{\mathcal{R}}_n(\hat{c}_n^\star) - \inf_{c\in\mathcal{C}}\mathcal{R}(c)\\ &\le \underbrace{\mathcal{R}(\hat{c}_n^\star) - \hat{\mathcal{R}}_n(\hat{c}_n^\star)}+ \underbrace{\hat{\mathcal{R}}_n(c^\star) - \mathcal{R}(c^\star)}\\ &\le \sup_{c\in\mathcal{C}}|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)| + \sup_{c\in\mathcal{C}}|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)| \;\le\; 2\sup_{c\in\mathcal{C}}|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)|\end{aligned}$$

Bound on suboptimality of $\hat{c}_n^\star$
The bound requires uniform convergence since we minimize $\hat{c}_n^\star$ w.r.t. empirical risk!

$$\mathbf{P}\{\mathcal{R}(\hat{c}_n^\star) - \inf_{c\in\mathcal{C}}\mathcal{R}(c) > \epsilon\} \;\le\; \mathbf{P}\{\sup_{c\in\mathcal{C}}|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)| > \epsilon/2\}$$

Figure 23: The Vapnik-Chervonenkis Inequality (1974.

problem might lead to problems. We must understand that optimization of the empirical risk may be learning something irrelevant, because the shape of the empirical risk might be "funny" for some distributions (see other figure). The richness of our solution space might trick us if we do not assume anything about the probability distribution on the dataset X. If our concept class is also very complex, then this problem can also arise. I think this is why Buhmann is so big on validation. We must always make sure our solutions are robust (in whatever way). Sometimes, returning only one solution "best" might not even be adequate.

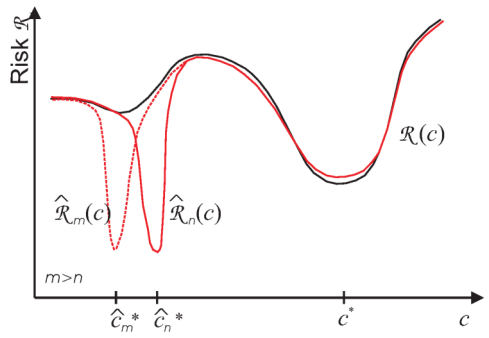


Figure 24: Insight about the problem of optimizing the empirical risk and only returning one solution. In some cases, we might actual get a lower optimal empirical risk compared to the optimal general risk.

One way to get around the issue of uniform convergence is to choose a classifier before we see the data and not use empirical risk minimization to select a better classifier type.
We can show a lower bound on the discrepancy between the empirical risk and the expected risk given the **empirical risk minimizer** $c_n^* = \arg\max_{c\in\mathcal{C}} \frac{1}{n}|\{(x_i, y_i) : c(x_i) \neq y_i, \forall i \le n$ if we have a finite concept class $|\mathcal{C}| < \infty$:

$$P\left(\mathcal{R}(c_n^*) - \inf_{c\in\mathcal{H}}\mathcal{R}(\hat{c}) > \epsilon\right) \le 2|\mathcal{C}|e^{-2n\epsilon^2}.$$

This and the similar equality for finite VC dim illustrates the point above. If we have large and complex concept class, the upper bound grows quickly and we can guarantee less and less.

Buhmann showed some more things here related to the proof, all very technical bounds and sometimes sloppy (union bounds). This may or may not be a good exercise. The thing that counts in the end is the following inequality, which is an upper bound on the expected error:

The inequality $\mathcal{R}(c) - \hat{\mathcal{R}}_n(c) \leq \epsilon$ holds with high probability for all functions c. This becomes:

$$\mathcal{R}(c) \leq \hat{\mathcal{R}}_n(c) + \sqrt{\frac{\log|\mathcal{C}| - \log(\delta/2)}{2n}},$$

where the second term the algebraic relation for the error parameter and can be interpreted as variance.

Empirical Risk Minimization for Hyperplanes
Our hypothesis class $\mathcal{H}$ is $\sum_{i=1}^{d} a_i x_i + a_0 = 0$ the set of all hyperplanes in d dimensions. The idea is to select $\binom{n}{d}$ hyperplanes from the infinite set of all hyperplanes and estimate their error rates. This way, we are quantizing the hypothesis space. Given n data points, we can ask in how many ways can they be separated by a hyperplanes. Note that we cannot realize all possible classifications using only hyperplanes. Thus, the idea is to only use separators that go through the data points, which does not sacrifice generality since classifications of close separators is the same except for the data defining the hyperplanes (see image). In d dimensions, we need d points to specify the hyperplane. Thus we select $\binom{n}{d}$ classifiers which gives us an equivalence class w.r.t to the hypothesis class. Each hyperplane defined by the data points defines two

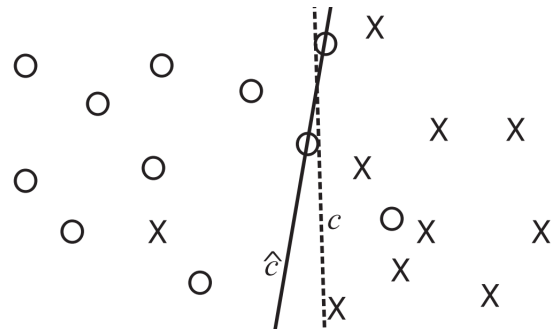


Figure 25: Separating hyperplanes are equivalent to the ones defined on the data points, except for the points defining the hyperplane.

classifiers (assigning the data points to one side or the other). Thus, we have $|\mathcal{H}| = 2\binom{n}{d}$. The best empirical classifier is defined by d samples:

$$\hat{c} = \arg \min_{i \leq 2\binom{n}{d}} \hat{\mathcal{R}}(c_i).$$

Let c be any linear classifier. Then there exists a hyperplane containing d data points such that the empirical error deviates at most by $\frac{d}{n}$, i.e.:

$$\hat{\mathcal{R}}(c) \geq \hat{\mathcal{R}}(\hat{c}) - \frac{d}{n}$$

Another theorem (theorem 6) shows more precise upper bounds on the error discrepancy for hyperplanes.

8 Useful Equations for Calculations

$$\begin{aligned}
\text{Tr}(\mathbf{A}) &= \textstyle\sum_i A_{ii} \\
\text{Tr}(\mathbf{A}) &= \textstyle\sum_i \lambda_i, \qquad \lambda_i = \text{eig}(\mathbf{A}) \\
\text{Tr}(\mathbf{A}) &= \text{Tr}(\mathbf{A}^T) \\
\text{Tr}(\mathbf{AB}) &= \text{Tr}(\mathbf{BA}) \\
\text{Tr}(\mathbf{A}+\mathbf{B}) &= \text{Tr}(\mathbf{A}) + \text{Tr}(\mathbf{B}) \\
\text{Tr}(\mathbf{ABC}) &= \text{Tr}(\mathbf{BCA}) = \text{Tr}(\mathbf{CAB}) \\
\mathbf{a}^T\mathbf{a} &= \text{Tr}(\mathbf{a}\mathbf{a}^T)
\end{aligned}$$

(a) Trace.

$$\begin{aligned}
\det(\mathbf{A}) &= \textstyle\prod_i \lambda_i \qquad \lambda_i = \text{eig}(\mathbf{A}) \\
\det(c\mathbf{A}) &= c^n \det(\mathbf{A}), \qquad \text{if } \mathbf{A} \in \mathbb{R}^{n\times n} \\
\det(\mathbf{A}^T) &= \det(\mathbf{A}) \\
\det(\mathbf{AB}) &= \det(\mathbf{A})\det(\mathbf{B}) \\
\det(\mathbf{A}^{-1}) &= 1/\det(\mathbf{A}) \\
\det(\mathbf{A}^n) &= \det(\mathbf{A})^n \\
\det(\mathbf{I}+\mathbf{u}\mathbf{v}^T) &= 1+\mathbf{u}^T\mathbf{v}
\end{aligned}$$

(b) Determinant.

Derivatives of matrices and vectors. In general:

$$\left[\frac{\partial \mathbf{x}}{\partial y_i}\right]_i = \frac{\partial x_i}{\partial y_i} \;;\; \left[\frac{\partial x}{\partial \mathbf{y}}\right]_i = \frac{\partial x}{\partial y_i} \;;\; \left[\frac{\partial \mathbf{x}}{\partial \mathbf{y}}\right]_{ij} = \frac{\partial x_i}{\partial y_i}.$$

- $\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$ for vectors x and a.
- $\frac{\partial AB}{\partial x} = \frac{\partial A}{\partial x}B + A\frac{\partial B}{\partial x}$ for matrices $A(x)$ and $B(x)$.
- $\frac{\partial A^{-1}}{\partial x}A + A^{-1}\frac{\partial A}{\partial x} = 0$ for a matrix $A(x)$.
- $\frac{\partial}{\partial A}(x^T A y) = xy^T$ and $\frac{\partial}{\partial A}(x^T A^T y) = yx^T$ and $\frac{\partial}{\partial A}(x^T A x) = \frac{\partial}{\partial A}(x^T A^T x) = xx^T$
- $\partial Tr(A) = Tr(\partial A)$ and $\partial(det(A)) = det(A)Tr(A^{-1}\partial A)$
- $\partial A^T = (\partial A)^T$

Assume $\mathbf{W}$ is symmetric, then

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{s}}(\mathbf{x}-\mathbf{A}\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) &= -2\mathbf{A}^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) \\
\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{s}) &= 2\mathbf{W}(\mathbf{x}-\mathbf{s}) \\
\frac{\partial}{\partial \mathbf{s}}(\mathbf{x}-\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{s}) &= -2\mathbf{W}(\mathbf{x}-\mathbf{s}) \\
\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{A}\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) &= 2\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) \\
\frac{\partial}{\partial \mathbf{A}}(\mathbf{x}-\mathbf{A}\mathbf{s})^T\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s}) &= -2\mathbf{W}(\mathbf{x}-\mathbf{A}\mathbf{s})\mathbf{s}^T
\end{aligned}$$

Multivariate Gaussian Distributions
The density of the multivariate Gaussian $x \sim \mathcal{N}(\mu, \Sigma)$ is:

$$p(x) = \frac{1}{\sqrt{(2\pi)^d det(\Sigma)}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right].$$

Σ is square $d \times d$, symmetric $\Sigma = \Sigma^T$ and positive semi-definite. The derivatives of the densities are:

$$\begin{aligned}
\frac{\partial p(x)}{\partial x} &= -p(x)\Sigma^{-1}(x-\mu) \\
\frac{\partial^2 p(x)}{\partial x^2} &= p(x)\left(\Sigma^{-1}(x-\mu)(x-\mu)^T\Sigma^{-1} - \Sigma^{-1}\right)
\end{aligned}$$

Assume we are given $x \sim \mathcal{N}(\mu, \Sigma)$ s.t.:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ and } \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \text{ and } \Sigma = \begin{bmatrix} \Sigma_1 & \Sigma_3 \\ \Sigma_3^T & \Sigma_2 \end{bmatrix}$$

The marginals are:

$$p(x_1) = \mathcal{N}(\mu_1, \Sigma_1)$$
$$p(x_2) = \mathcal{N}(\mu_2, \Sigma_2)$$

And the conditional $p(x_1|x_2) = \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right)$ has moments:

$$\hat{\mu} = \mu_1 + \Sigma_3\Sigma_2^{-1}(x_2 - \mu_2)$$
$$\hat{\Sigma} = \Sigma_1 + \Sigma_3\Sigma_2^{-1}\Sigma_3^T$$

for $p(x_2|x_1) = \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right)$, exchange Σ_3^T for Σ_3 and vice versa.

The Jensen Inequality. Let X be a random variable and f be a convex function. Then:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$

The Markov Inequality. Let X be a non-negative random variable. Then:

$$\mathbf{P}(X \geq \epsilon) \leq \frac{\mathbb{E}[X]}{\epsilon}.$$

The Union Bound For countable sets of events A_i, we have:

$$\mathbf{P}(A_1 \cup A_2 \cup \cdots) \leq \mathbf{P}(A_1) + \mathbf{P}(A_2) + \cdots$$