TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Prof. Dr. Björn Sprungk
Faculty of Mathematics and Computer Science
Institute of Stochastics

Mathematics of machine learning

Chapter 1: The formal learning model

Winter term 2024/25

Chapter 1: The formal learning model

What it's about?

- Learning the basic notions such as hypotheses, loss functions, empirical risk etc.
- Build up mathematical model for statistical learning
- Learn how to evaluate the outcome of machine learning

the goal is simple. we would like to learn y hypothesis h(xi) which approximates the yi for all i to m, making sure that it explains the given data well.

According to this theorm, we could consider a linear hypothesis that uses the sgn function of a transpose weighted features with the biases.

The sgn function, in terms of machine learning sgn function is used to predict points of classifiers into 3 classes. as 1 (this matter is could be a good success in prediction in the positive classifier (true positive)), 0 (indicates that the input is exactly on the decision boundary. it is worst cuz it is not in any classes yet), -1 (this could be success prediction in the negative classifier if the labels are correctness are true(true negative))

For transpose function for the weight vectors of features, it means here that this dot product would be the sum of the weight vector and features.

transpose(w)x = w1x1 + w2x2 + ... +wnxn

- the weight vector in the sgn function indicates the the importance of the feature. the more the weight the more the important the feature is.
- the bias serves as an additional parameter that helps adjust the output of the model. Without the bias, the hyperplane would always pass through the origin

in assumption of the randomness of data we set the Mu to be unknown distribution data on our data space in M independent realizations, and we set S to be realization of the random sample data.

A learning rule or algorithm is being defined as a measurable mapping in the data sapce with m dimensional to a map of set of all mappings. with this give sample random data we would have the learned hypothesis. h = A(s) as output.

Setting and goal of statistical learning

Setting (passive, batch, and supervised)

We are given a batch or sample of paired data

$$s = ((x_1, y_1), \ldots, (x_m, y_m))$$

consisting of features $x_i \in \mathcal{X}$ and associated labels $y_i \in \mathcal{Y}$, $i = 1, \ldots, m$, where $\mathcal{X}, \mathcal{Y}$ denote the underlying feature and label sets, respectively.

Goal

Learn or find a (sound) so-called hypothesis h, i.e., a mapping from features to labels

$$h \colon \mathcal{X} \to \mathcal{Y}$$

which explains the given data $s = ((x_1, y_1), \ldots, (x_m, y_m))$ well, e.g.

$$h(x_i) \approx y_i \qquad \forall i = 1, \ldots, m.$$

Examples for feature sets $\mathcal{X}$

- Grayscale images ($n \times m$ pixels):

$$\mathcal{X} = [0,1]^{n \times m}$$

- Text analysis (frequencies of n words/phrases):

$$\mathcal{X} = \mathbb{N}^n$$

- Medical data: here $\mathcal{X}$ is often a product set of continuous features (e.g. blood glucose) and discrete ones (age, gender,...)

Special cases of label set $\mathcal{Y}$

- Binary classification (e.g., healty/ill, spam/no spam...):

$$|\mathcal{Y}| = 2$$

- Multiclass classification (bird/horse/flower/...):

$$2 < |\mathcal{Y}| < \infty$$

- Scalar ($d = 1$) or multiple ($d > 1$) regression:

$$\mathcal{Y} = \mathbb{R}^d, \qquad d \in \mathbb{N}$$

without loss of generality (w.l.o.g)
Bedune az dast dadane koliat

bias also refers to the term that helps to adjust the output of a model, often represented as a constant added to the weighted sum of inputs.

Example of hypotheses

Given that $\mathcal{X} \subset \mathbb{R}^n$ and $|\mathcal{Y}| = 2$, e.g., $\mathcal{Y} = \{-1, +1\}$ w.l.o.g., we can consider (affine) linear hypotheses:

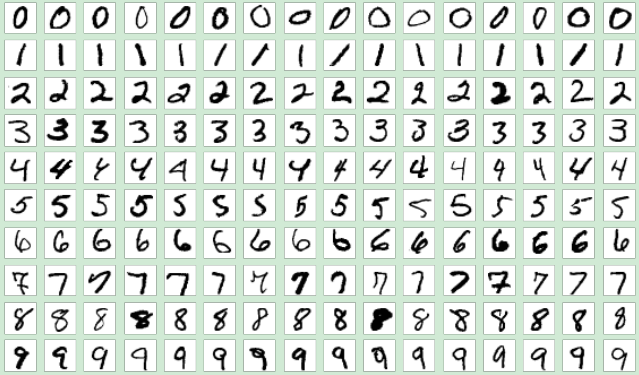
$$h(x) := \operatorname{sgn}\left(w^\top x + b\right)$$

dot product

where $w \in \mathbb{R}^n$ denotes the weights and $b \in \mathbb{R}$ the bias to be learned, i.e., fitted to the data (x_i, y_i).

Common example: MNIST

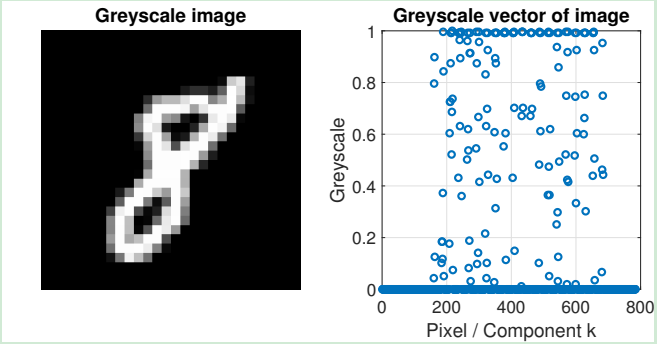
- The MNIST database is a publicly available data set from 1998 which consists of 70 000 greyscale images of handwritten figures:



Source: wikipedia

- It is a classical example / benchmark data set for machine learning
- Each image is of size 28x28 pixels and each pixel has a value between null (black) and one (white).

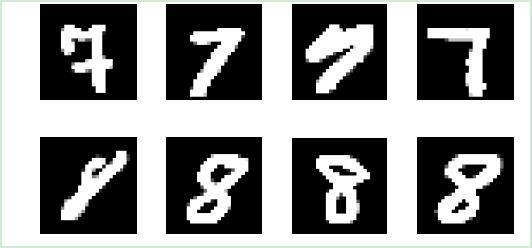
- Thus, we can describe each image by a vector $\mathbf{x} \in [0,1]^{784}$ containing the pixelwise greyscale values:

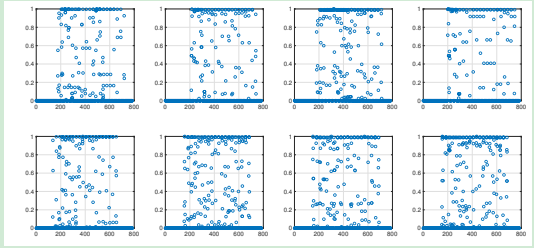


- Furthermore, each image is labeled by $y \in \{0, 1, \ldots, 9\}$ determing which figure is displayed
- Thus, we have as feature and label set, respectively,

$$\mathcal{X} = [0,1]^{784}, \qquad \mathcal{Y} = \{0, \ldots, 9\}$$

- We then like to learn to recognize/distuinguish the handwritten figures by the greyscale vectors x:

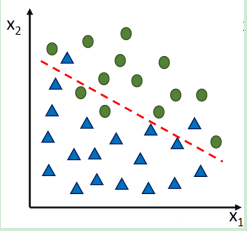




i.e., we would like to learn a hypothesis $h\colon [0,1]^{784} \to \{0,\dots,9\}$

- Hence, the MNIST data set yields a multilabel classification problem
- We later restrict ourselves to binary classification between figures "7" and "8" by an (affine) linear hypothesis

$$h\colon [0,1]^{784} \to \{-1,+1\}, \qquad h(x) = \operatorname{sgn}\left(w^\top x + b\right),$$



Source: data-science-blog

Neutral learning

- We define the data space $\mathcal{D} := \mathcal{X} \times \mathcal{Y}$
- We assume an underlying probability space $(\Omega, \mathcal{F}, \mathbb{P})$ (which will be of no importance later)
- and consider a random data pair $(X, Y) \colon \Omega \to \mathcal{D}$

Assumption of random data

We assume that the data set $s \in \mathcal{D}^m$ consists of m (independent) realizations (x_i, y_i) of

$$(X, Y) \sim \mu$$

where μ denotes the (unknown) probability distribution of the data on $\mathcal{D}$. Thus, the given sample $s \in \mathcal{D}^m$ is a realization of a random sample

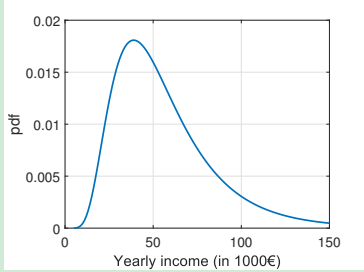
$$S = ((X_1, Y_1), \ldots, (X_m, Y_m)) \sim \mu^m,$$

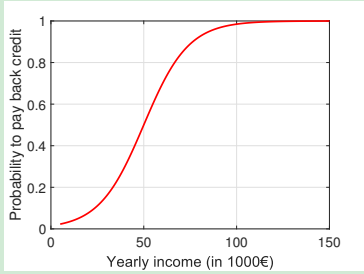
i.e., $(X_i, Y_i) \sim \mu$ independent and identically distributed (iid).

Example: Credit default

- We consider credit default data where the feature and label
  - x : yearly income of a household
  - y : household was able to payback a 100K credit within due time
- The yearly income among the population may be lognormally distributed $X \sim \mathrm{LogN}(\mu, \sigma^2)$, see probability density function (pdf)
- Whether or not a household with income x may be able to pay back the credit is random as well (due to many circumstances)...
- ... where the (conditional) probability for a payback given a certain income x of the household may be

$$\mathbb{P}(Y = 1 \mid X = x) = \frac{1}{1 + \exp(-\alpha(x - x_0))}$$





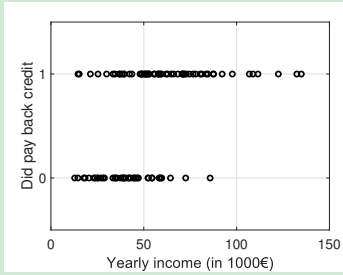
- The resulting joint probability distribution μ of (X, Y) on the data space

$$\mathcal{D} = [0, \infty) \times \{0, 1\}$$

then may yield the following sample

$$s = ((x_1, y_1), \dots (x_m, y_m))$$

shown on the righthand side



- Thus, the assumption above allows for the existence of a true hypothesis $h^\dagger \colon \mathcal{X} \to \mathcal{Y}$ such that

$$\mathbb{P}(h^\dagger(X) = Y) = 1$$

but does not require such a deterministic relation between features x and labels y.

- **Notation:** In the following $\mathbb{P}_\mu(\cdot)$ and $\mathbb{E}_\mu[\cdot]$ denotes probability and expectation w.r.t. $(X, Y) \sim \mu$.

How do we learn a hypothesis h from data?

Definition 1.1:

A learning rule or learning algorithm is a (measurable) mapping

$$A\colon \mathcal{D}^m \to \mathcal{Y}^{\mathcal{X}}$$

where $\mathcal{Y}^{\mathcal{X}}$ denotes the set of all (measurable) mappings $h\colon \mathcal{X} \to \mathcal{Y}$.

Given a data sample $s \in \mathcal{D}^m$, the learning algorithm outputs a learned hypothesis $h = A(s)$.

We can extend this definition to

$$A\colon \bigcup_{m\in\mathbb{N}} \mathcal{D}^m \to \mathcal{Y}^{\mathcal{X}},$$

to allow for finite data sets s of arbitrary size $|s| = m \in \mathbb{N}$.

We get to know several learning rules in this course. The most basic and important one is the rule of empirical risk minimization...

the loss functions are used to evaluate how well the predicted output has been. in this case we use the norm squared of the differences between the actual value and the predicted value. -> actual value - predicted value

here, we use norm of this difference because of multiple reasons:

1. penalizing larger errors more, which make the model more sensitive to outliers
2. make it smooth and reliable
3. justify the stats for using squared loss in regression models. by this, we minimize squared loss to maximizing likelihood when errors are normally distributed.

   ERM only cares about minimizing the error on the training data, which is why it is often considered naïve, which is why sometimes it encounter to over fitting. (if the hypothesis class is too complex like neural networks, it may perfectly memorize the training data but fails unseen data. which leads to over fitting.)

   also ERM Ignores the Complexity of the Hypothesis. and more.
   the best approach in order to solve all these ERM problems, are:

   1. Regularization (L1, L2 penalities): which penalizes overly complex models to prevent overfitting. specially ridge regression (L2 regulraization, could do it.)
   2. PAC learn ability or Probably Approximately Correct and bounds like VCD help to ERM.
   3. another way is to avoid over fitting is to use a NON EMPTY SUBSET CLASS OF HYPOTHESIS.

Loss

To evaluate how well a hypothesis $h\colon \mathcal{X} \to \mathcal{Y}$ fits the given data (x_i, y_i) (or, in general, the conditional distribution $\mathbb{P}(Y \mid X)$ of $(X, Y) \sim \mu$) we use a loss function

$$\ell\colon \mathcal{Y}^{\mathcal{X}} \times \mathcal{D} \to [0, \infty),$$

which quantifies the loss due to the prediction $h(x)$ for the true label y via $\ell(h, (x, y)) \geq 0$

0-1-loss and quadratic loss

A classical loss for (binary) classification is the 0-1-loss

$$\ell(h, (x, y)) := \begin{cases} 0, & \text{if } h(x) = y, \\ 1, & \text{else.} \end{cases}$$

For regression, i.e., $\mathcal{Y} = \mathbb{R}^d$, the quadratic loss is often used

$$\ell(h, (x, y)) := \|y - h(x)\|^2$$

Risk

We then can compare different hypotheses h by their average loss over the data:

Definition 1.2:

Given a loss function ℓ and a(n unknown) data distribution μ on $\mathcal{D}$ we define the (expected) risk of a hypothesis $h\colon \mathcal{X} \to \mathcal{Y}$ by

$$\mathcal{R}_\mu(h) := \mathbb{E}_\mu[\ell(h,(X,Y))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(h,(x,y))\,\mu(\mathrm{d}x\ \mathrm{d}y)$$

Example: 0-1-loss

For the 0-1-loss the risk

$$\mathcal{R}_\mu(h) = \mathbb{E}_\mu\left[\mathbf{1}_{h(X)\neq Y}\right] = \mathbb{P}_\mu\left(h(X)\neq Y\right)$$

is the probability for a wrong prediction or misclassification

Our goal

Find the best hypothesis $h^\star$ in terms of lowest risk

$$h^\star \in \operatorname*{argmin}_{h\colon \mathcal{X}\to\mathcal{Y}} \mathcal{R}_\mu(h).$$

Problem: We do not know μ in practice!

Empirical risk

Definition 1.3:

Given a loss function ℓ and a data sample $s = ((x_1, y_1), \dots, (x_m, y_m)) \in \mathcal{D}^m$ the empirical risk of a hypothesis $h\colon \mathcal{X} \to \mathcal{Y}$ is given by

$$\mathcal{R}_s(h) := \frac{1}{m} \sum_{i=1}^{m} \ell(h, (x_i, y_i))$$

Usage: Since $\mathcal{R}_s(h)$ is an unbiased empirical approximation of $\mathcal{R}_\mu(h)$ and, i.e., for random sample $S \sim \mu^m$

$$\mathbb{E}[\mathcal{R}_S(h)] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^{m} \ell(h, (X_i, Y_i))\right] = \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_\mu[\ell(h, (X_i, Y_i))] = \mathcal{R}_\mu(h)$$

we can compute $\operatorname{argmin}_h \mathcal{R}_s(h)$ as a proxy for $\operatorname{argmin}_h \mathcal{R}_\mu(h)$ ...

Empirical Risk Minimization (ERM)

Thus, we obtain as a first **learning algorithm:**

The naive ERM rule

Input:

- Training data $s = \big((x_1, y_1), \dots, (x_m, y_m)\big) \in \mathcal{D}^m$,
- loss function $\ell\colon \mathcal{Y}^{\mathcal{X}} \times \mathcal{D} \to [0, \infty)$.

Output:

$$\mathrm{ERM}(s) := \operatorname*{argmin}_{h \in \mathcal{Y}^{\mathcal{X}}} \mathcal{R}_s(h) = \operatorname*{argmin}_{h \in \mathcal{Y}^{\mathcal{X}}} \frac{1}{m} \sum_{i=1}^{m} \ell\left(h, (x_i, y_i)\right),$$

or in case of multiple minimizers $\mathrm{ERM}(S) \in \operatorname*{argmin}_{h \in \mathcal{Y}^{\mathcal{X}}} \mathcal{R}_s(h)$.

argmin: This notation indicates that we are looking for the hypothesis hh that minimizes the empirical risk Rs(h)Rs(h).

Question: Why is this learning rule called naive?

Example: The Problem of Overfitting

Let $\mathcal{Y} = \{0, 1\}$ and choose the 0-1-loss ℓ. Then for a given training sample $s \in \mathcal{D}^m$ we have

$$\mathcal{R}_s(h) = \frac{1}{m} \left| \{ i = 1, \dots, m \colon \ h(x_i) \neq y_i \} \right|.$$

Assuming that $x_i \neq x_j$ for $i \neq j$ we obtain

$$h_s(x) := \begin{cases} y_i, & \text{if } x = x_i, \\ 0, & \text{else,} \end{cases} \qquad x \in \mathcal{X},$$

as a minimizer of the empirical risk, since $\mathcal{R}_s(h_s) = 0$.

But: The risk $\mathcal{R}_\mu(h_s)$ can be arbitrarily close to 1!

Example: Let $\mathcal{X} = [0, 1]$ and μ be the law of $(X, \mathbf{1}_{\{x \geq \frac{1}{2}\}}(X))$ for $X \sim \mathrm{U}[0, 1]$. Then almost surely

$$\mathcal{R}_S(h_S) = 0, \quad \text{but} \quad \mathcal{R}_\mu(h_S) = \frac{1}{2}.$$

Hypotheses classes

- In order to avoid overfitting, one can restrict the ERM rule to a set of "sound" hypotheses (cf. conductive bias of rats)
- To this end, we consider a (nonempty) subset $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ of hypotheses which we call a hypothesis class

The (biased) ERM rule

Input:

- Training data $s = \big((x_1, y_1), \dots, (x_m, y_m)\big) \in \mathcal{D}^m$,
- Hypothesis class $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$,
- loss function $\ell\colon \mathcal{H} \times \mathcal{D} \to [0, \infty)$.

Output:

$$\mathrm{ERM}_{\mathcal{H}}(s) := \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{R}_s(h) = \operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \ell\left(h, (x_i, y_i)\right),$$

or in case of multiple minimizers $\mathrm{ERM}_{\mathcal{H}}(s) \in \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{R}_s(h)$.

Example: Linear hypotheses

- Linear hypotheses are a very important class of hypotheses for binary classification – we study them in detail in Chapter 3.

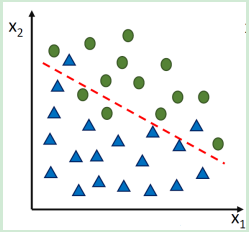
- Given two classes $\mathcal{Y} = \{-1, +1\}$ we try to find a hyperplane through the feature set $\mathcal{X} \subseteq \mathbb{R}^d$ which separates the data best:

$$\mathcal{H} = \left\{ h_{w,b} \mid h_{w,b}(x) = \operatorname{sgn}(w^\top x + b),\ w \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

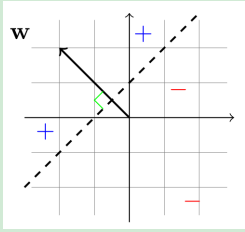
  where sgn denotes the sign function

- Given training data s we learn a suitable weight vector w and bias value b via

$$\mathrm{ERM}_{\mathcal{H}}(s) = \operatorname*{argmin}_{(w,b) \in \mathbb{R}^{d+1}} \frac{1}{m} \sum_{i=1}^{m} \ell\left(h_{w,b}, (x_i, y_i)\right)$$



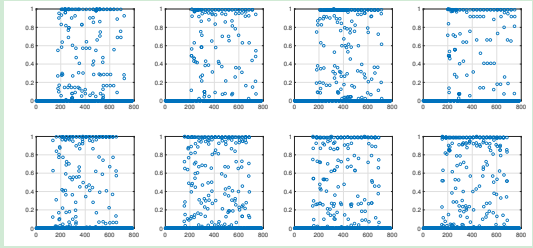
Source: data-science-blog



Source: "Understanding Machine Learning" (2014)

Example: Linear hypotheses for MNIST data set

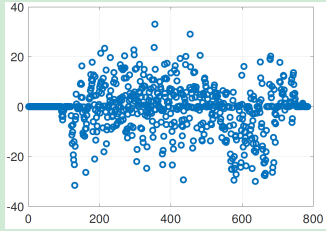
- We learn to distinguish between handwritten "7" and "8" by a learning a separting hyperplane $h(x) = \operatorname{sgn}(w^\top x + b)$ through $\mathcal{X} = [0,1]^{784}$



- By minimizing the empirical risk $\mathcal{R}_s$ for the 0-1-loss given $m = 12,116$ training images we obtain a linear hypothesis h_s with bias $b_s = 74$ and weight vector $w_s \in \mathbb{R}^{784}$ shown on the left

- The obtained empirical risk is $\mathcal{R}_s(h_s) = 0.005$, i.e., we could correctly classify 99.5% of the training data

Learned weight w_s



Errors:

1. Error Generalization: Quantifies the quality of the hypothesis performance

$$\epsilon_{gen}(h_s) = R_\mu(h_s) \rightarrow = \epsilon_{app} + \epsilon_{est} + \epsilon_{opt}$$

2. Error Approximation: calculates the best Expected Risk withing Chosen H

$$\epsilon_{app}(H) = \inf_{h \in H} R_\mu(h)$$

3. Error Estimate: quantifies the effects of the empritical approximation of Rs of Rmu

$$\epsilon_{est}(H, S) = R_\mu(h_s) - \inf_{h \in H} R_\mu(h)$$

4. Error Optimzation: quantifies how well the hypothesis is being working in the defined Class of H

$$\epsilon_{opt}(K, H, S) = R_\mu(h_s^{(K)}) - R_\mu(h_s)$$

$h_s = A(s)$
h = predicted
$|H|$ = defined class
K = count of iteration

How well did we learn?

Definition 1.4:

Given a learned hypothesis $h_{s,\mathcal{H}} = \mathrm{ERM}_{\mathcal{H}}(s)$ using a loss ℓ and training data $s \in \mathcal{D}^m$ drawn w. r. t. μ^m the resulting generalization error is given by

$$\varepsilon_{\text{gen}} = \varepsilon_{\text{gen}}(\mathcal{H}, s) = \mathcal{R}_\mu(h_{s,\mathcal{H}}).$$

This definition also holds for hypotheses $h_{s,\mathcal{H}} = A_{\mathcal{H}}(s)$ learned by any other learning algorithm $A = A_{\mathcal{H}}$.

- ε_{gen} quantifies how well the trained hypothesis performs for new data
- The choice of $\mathcal{H}$ effects the achievable generalization error:

$$\varepsilon_{\text{gen}}(\mathcal{H}, s) \;\geq\; \varepsilon_{\text{gen}}(\mathcal{Y}^{\mathcal{X}}, s) \;\geq\; \min_{h \in \mathcal{Y}^{\mathcal{X}}} \mathcal{R}_\mu(h).$$

  We call the first discrepancy the inductive bias, the latter one is of statistical nature.

- Thus, generalization error results from a superposition of different effects

Definition 1.5:

Given a loss function ℓ and data distribution μ the approximation error of a hypothesis class $\mathcal{H}$ is given by

$$\varepsilon_{\text{app}} = \varepsilon_{\text{app}}(\mathcal{H}) := \min_{h \in \mathcal{H}} \mathcal{R}_\mu(h).$$

Furthermore, given training data $s \in \mathcal{D}^m$ and a learning algorithm A we define the estimation error by

$$\varepsilon_{\text{est}} = \varepsilon_{\text{est}}(s) := \mathcal{R}_\mu(A(s)) - \min_{h \in \mathcal{H}} \mathcal{R}_\mu(h).$$

- $\varepsilon_{\text{app}}(\mathcal{H})$ quantifies the best achievable risk within the chosen $\mathcal{H}$
- $\varepsilon_{\text{est}}(s)$ quantifies the effect of the empirical approximation $\mathcal{R}_s$ of $\mathcal{R}_\mu$

$$\frac{1}{m}\sum_{i=1}^{m} \ell(h,(x_i,y_i)) = \mathcal{R}_\mu(h)$$

Idealized Error Decomposition

$$\varepsilon_{\text{gen}}(s,\mathcal{H}) = \varepsilon_{\text{app}}(\mathcal{H}) + \varepsilon_{\text{est}}(s).$$

Another Source of Error

- In practice we usually never find the exact minimizer for the ERM rule

$$h_s = \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{R}_s(h) \tag{ERM}$$

- We typically use iterative optimization methods for solving (ERM). These recursively compute iterates $h_s^{(k)} \in \mathcal{H}$, $k \in \mathbb{N}$, such that (hopefully) $h_s^{(k)} \to h_s$ as $k \to \infty$.

- However, we have to stop the iteration at some point, say after $k \in \mathbb{N}$ steps. This yields the optimization error

$$\varepsilon_{\mathsf{opt}} = \varepsilon_{\mathsf{opt}}(k) := \mathcal{R}_\mu\left(h_s^{(k)}\right) - \mathcal{R}_\mu(h_s).$$

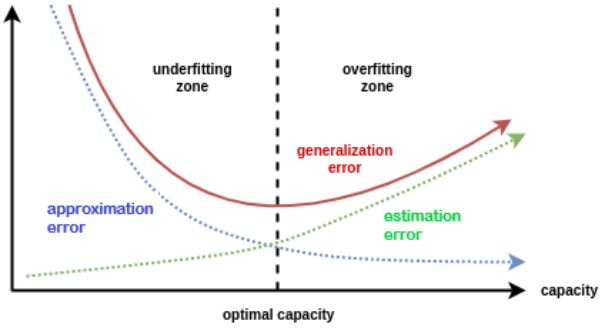
Realistic Error Decomposition

$$\varepsilon_{\mathsf{gen}}(s, \mathcal{H}, k) := \mathcal{R}_\mu\left(h_{s,\mathcal{H}}^{(k)}\right) = \varepsilon_{\mathsf{app}}(\mathcal{H}) + \varepsilon_{\mathsf{est}}(s) + \varepsilon_{\mathsf{opt}}(k)$$

First Conclusions

1. How to choose hypothesis classes $\mathcal{H}$?
   - $\mathcal{H}$ should be sufficiently large to allow for a small $\varepsilon_{\text{app}}(\mathcal{H})$,
   - $\mathcal{H}$ should be "easy to learn" i.e., $\operatorname{argmin}_{h\in\mathcal{H}} \mathcal{R}_\mu(h) \approx \min_{h\in\mathcal{H}} \mathcal{R}_s(h)$ for moderate $|s|$
   - $\mathcal{H}$ and ℓ should be "nice" to allow for efficient optimization methods and a small $\varepsilon_{\text{opt}}(k)$

As we enlarge our family of predictors our approximation error monotonically decreases, as we are able to capture more complex relationships. However, as our family of predictors increases, our estimation error increases as we over-fit more.



should the Hypothesis class be large so error app with be small, and the Hypothesis class H should be also small enough so the error estimation not lead to overfitting.

2. Which fields of mathematics are important in machine learning?
   - Analysis and approximation theory to study ε_{app},
   - Stochastics and statistical learning theory to study ε_{est},
   - (Numerical) Optimization to study ε_{opt}.

3. How does this course continue?
   - We will deal with statistical learning theory to understand ε_{est},
   - We will consider a bit of approximation theory to study ε_{app},
   - We will learn about special classes $\mathcal{H}$, such linear hypotheses and neuronal networks, and their approximation and estimation error,
   - We will discuss numerical and stochastic optimization for the ERM rule to study ε_{opt}.

Take home messages

- What is our formal learning model for statistical learning?
- Where do you find supervised, neutral, passive, and batch-learning in there?
- What is risk and empirical risk?
- Which learning rule do we consider?
- What is overfitting and how can we prevent that?
- Which errors occur in machine learning?
- Artificial intelligence and machine learning that is purely computer science, right?