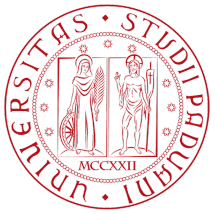

Università degli studi di Padova
Dipartimento di Fisica "Galileo Galilei"
Academic year 2019/2020

Master degree in Physics of Data

Notes of Machine Learning

Contents

What is Machine Learning?

The main subject o these notes is automated learning, or, as we will more often call it, Machine Learning (ML). Machine Learning is essentially "teaching to the computers so they can learn from inputs we give them". The input to a learning algorithm is training data, representing experience, and the output is some expertise, usually a new program that can independently perform a task.
A tipical example of a learning algorithm is a program that can distnguish between male and female people: it will be trained with a sample of different people (more heterogeneous as possible) to teach it the main differences of the two genres.
Another example is an algorithm that gives the probability of a person to have a certain height given, for example, the genre or the age.
A more common example is the algorithm that, between emails, can distinguish spam emails from the other ones.
Before starting the effective lectures, we should define a first small difference between the types of learning: since learning involves an interaction between the learner and the environment, one can divide learning tasks according to the nature of that interaction.

- *Supervised learning* describes a scenario in which the "experience", the training data, contains significant information that is missing in the unseen "test examples" to which the learned expertise is to be applied. In this setting, the acquired expertise is aimed to predict that missing information for the test data. In such cases, we can think of the environment as a teacher that "supervises" the learner by providing the extra information (labels). More abstractly, the algorithm can be seen as a process of "using experience to gain experties".

- In *Unsupervised learning*, instead, there is no distinction between training and test data: the learner processes input data with the goal of coming up with some summary, or compressed version of that data.

Chapter 1

Machine Learning Model

First of all, we give some definitions of the stuff the machine learnig algorithm has access to:

- **Domain set** (or **instance space** $\mathcal{X}$): is the set of all possible objects that we want to label, i.e. make prediction about. Usually this is a "vector of features".
- **Label set**: contains all possible prediction (labels). Usually correspond to the binary set $\{0, 1\}$ (that can mean $\{True, False\}$).
- **Training data**: $S = ((x_1, y_1), \ldots, (x_m, y_m))$ is a finite sequence of pairs in $\mathcal{X} \times \dagger$, and represent the input of the ML algorithm. The index m, instead, represent the dimension of the dataset. Despite the "set" notation, S is a sequence. In particular, the same example may appear twice in S and some algorithms can take into account the order of examples in S.
- **Prediction rule**: Is a function $h : \mathcal{X} \to \dagger$ (also called "predictor", "hypotheses" or "classifier") that can be used to predict the label of new domain points. We denote $A(S)$ the hypothesis that a learning algorithm A returns upon receiving the training sequence S.
- **Data-generation model**: We assume that the instances are generated by some probability distribution $\mathcal{D}$ (not known by the algorithm), and we assume that there is a "correct" labelling function $f : \mathcal{X} \to \dagger$ for which $y_i = f(x_i) \quad \forall i$. The labelling function isn't known by the algorithm: in fact, it is just what the program is trying to figure out. In summary, each pair in the training data is generated by first sampling a point x_i according to $\mathcal{D}$ and then labelling it by f.
- **Measure of success**: We define *error of the classifier* the probability that the algorithm does not predict the correct label on a random data point generated by distribution $\mathcal{D}$. In other words, the error of h i the probability to draw a random instance x, according to the distribution $\mathcal{D}$, such that $h(x) \neq f(x)$.

Usually, our sample will be a vector (so a set of numbers) $x \in \mathbb{R}^d$.
We must pay attention that what the algorithm learn depends on training data! So, if the training data are bad, the results will be bad aswell.

1.1 Measure of Success: Loss Function

Given a domain subset $A \subset \mathcal{X}$ and the probability distribution $\mathcal{D}$, $\mathcal{D}(A)$ represents the probability of observing a point $x \in A$.
In many cases, we refer to A as an *event* and express it using a function $\pi : \mathcal{X} \to \{0, 1\}$, that is $A = \{x \in \mathcal{X} : \pi(x) = 1\}$.
In this case denote $\mathcal{D}(A)$ as $\mathbb{P}_{x\sim\mathcal{D}}[\pi(x)]$, where $\sim$ indicates that x point is sampled according to $\mathcal{D}$.
Now, we define the **error of prediction rule**:

$$\ell_{\mathcal{D},f}(h) = \ell_{\mathcal{D}}(h) = \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)] = \mathcal{D}(\{x : h(x) \neq f(x)\}) \tag{1.1}$$

This is nothing but the probability of randomly choosing an example x for which $h(x) \neq f(x)$. It is also called *generalization error*, or **true error**.
It is important to understand that the algorithm doesn't know the probability distribution $\mathcal{D}$, so it is not able to compute the true error.

1.2 Empirical Risk Minimization

As mentioned earlier, a learning algorithm receives as input a training set S, sampled from an unknown distribution $\mathcal{D}$ and labeled by some target function f, and should output a predictor $h_S : \mathcal{X} \to \mathcal{Y}$ (the subscript S emphasizes the fact that the output predictor depends on the training set). The goal of the algorithm is to find h_S that minimizes the error with respect to the unknown $\mathcal{D}$ and f.
If, because of the true error can't be computed, we would like to try another estimate of it, we should define the error on the training data, called **training error**:

$$\ell_S(h) = \frac{|i : h(x_i) \neq y_i, i = 1, \ldots, m|}{m} \tag{1.2}$$

This division is simply the rate o correct prediction and the total samples (assuming classification problem and 0-1 loss), and it is also called *empirical error* or *empirical risk*.
This learning paradigm, which main goal is to find a predictor h that minimize ℓ_S, is called *Empirical Risk Minimization* or
, for short.

Although the ERM rule seems very natural, without being careful this approach may fail miserabily.
Let's consider, for example, the sample in figure 1.1 (on the left), assuming a probability distribution $\mathcal{D}$ for which instances x are taken uniformily at random in the square, and the labelling function f that assign the value 1 to the instances in the upper squares, and 0 to the others (essentially 0 to blue ones and 1 to red ones). If we collect the training set shown in figure 1.1 (on the right), to train the ML algorithm, a good predictor seems to be the function $h_S(x) = \begin{cases} 0 \text{ if x in left side} \\ 1 \text{ if x in right side} \end{cases}$, that effectively minimize the training loss ($L_S(h_S) = 0$, practically perfect). But is this a good predictor?
On the other hand, the true error of any classifier is, in this case $L_{\mathcal{D}}(h_S) = \frac{1}{2}$.

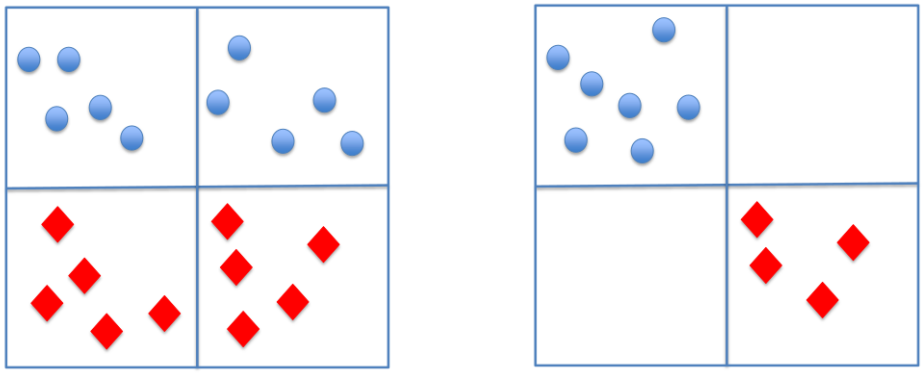

Figure 1.1: Example of overfitting

We have found a predictor whose performance on the training set is excellent, yet its performance on the true "world" is very poor. This phenomenon is called overfitting. Intuitively, overfitting occurs when our hypothesis fits the training data "too well", but doesn't have the same performances on a generic dataset. We obviously want to avoid this overfitting, because algorithms should work in general cases. Overfitting is less probable when the training set is large. The bigger is a training set, the smaller will be the probability to have regular distribution of data.

1.3 Hypothesis class

Rather than giving up completely the ERM paradigm, we want to look at the conditions under which it is guarantee that ERM does not overfit.
A common solution is to apply the ERM learning rule over a restricted search space. Formally, the learner should choose in advance (before seeing the data) a set of predictors. This set is called a **hypothesis class** and is denoted by $\mathcal{H}$. Each $h \in \mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$. For a given class $\mathcal{H}$, and a training sample, S, the $ERM_{\mathcal{H}}$ learner uses the ERM rule to choose a predictor h, with the lowest possible error over S:

$$ERM_{\mathcal{H}} \in \underset{h \in \mathcal{H}`}{argmin}\, L_S(h) \tag{1.3}$$

where *argmin* stands for the set of hypotheses in $\mathcal{H}$ that achieve the minimum value of $L_S(h)$. Since the choice of such a restriction is determined before the learner sees the training data, it should ideally be based on some prior knowledge about the problem to be learned. Intuitively, choosing a more restricted hypothesis class better protects us against overfitting but at the same time might cause us a stronger inductive bias: if the class is too small you will not find a good estimator.
Now we would like to prove that, if $\mathcal{H}$ is a finite class $ERM_{\mathcal{H}}$ is guarantee not to overfit, provided it is based on a sufficiently large training sample (depending on the size of $\mathcal{H}$). Limiting the dimension of $\mathcal{H}$ ($|\mathcal{H}| < \infty$) seems not to be very realistic, because we usually have to deal with parameters defined in $\mathbb{R}^d$: the correct assumption uses the finite precision with which computers manage real numbers.
Let us now analyze the performance of the $ERM_{\mathcal{H}}$ learning rule assuming that $\mathcal{H}$ is a finite class. For a training sample, S, labeled according to some $f : \mathcal{X} \to \mathcal{Y}$, let h_S denote a result of applying

$ERM_{\mathcal{H}}$ to S, i.e. $h_s \in \underset{h \in \mathcal{H}}{argmin}\, L_S(h)$. Let's make some assumptions:

- (**Realizability**) There exist a $h^* \in \mathcal{H}$ such that $L_{\mathcal{D}}(h^*) = 0$: in practice we are asking for the existance of the perfect solution in our hypothesis class. Note that this assumption implies that with probability 1 over random samples, we have $L_S(h^*) = 0$.

- (**i.i.d.**) The examples in the training set are indipendently and identically distributed (i.i.d.) according to the distribution $\mathcal{D}$ and than labelled according the labeling function f. We denote this assumption by $S \sim \mathcal{D}^m$ (this notation represents the fact that we sample m times according to the distribution $\mathcal{D}$). This assumption seems to be unrealistic too: the sampling is usually biased on the situation; you will never be able to sample reproducing the real distribution and leave them indipendent between each other. Anyway, the larger the sample get, the more likely it is to reflect more accurately the distribution ad labelig used to generate it.

Since $L_{\mathcal{D}}(h_S)$ depends on the training set S, and that training set is picked up in a random process, there is a randomness in the choice of the predictor h_S and, consequently, in the risk $L_{\mathcal{D}}(h_S)$. It's never guarantee that the solution we find is the perfect one, because there is always some probability that the sampled training data happens to be very nonrepresentative of the underlying $\mathcal{D}$.
Usually we denote the probability o getting a nonrepresentative sample by δ, and call $(1 - \delta)$ the **confidence parameter** of our prediction.
On the top of that, since we cannot guarantee a perfect label prediction, we introduce another parameter, the **accuracy parameter**, commonly denoted by ε.
So, we interpret the event $L_{\mathcal{D}}(h_S) > \varepsilon$ as a failure of the learner, while if $L_{\mathcal{D}}(h_S) \leq \varepsilon$ we view the output of the algorithm as an approximately correct predictor.

Theorem 1. *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0, 1)$, $\varepsilon \in (0, 1)$, and $m \in \mathbb{N}$ (size of the training set, i.e. S contains m i.i.d. samples) such that*

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$$

Then, for any f and any $\mathcal{D}$ for which the realizability assumption holds, with probability $\geq 1 - \delta$ we have that for every ERM hypothesis h_S it holds that

$$L_{\mathcal{D},f}(h_S) \leq \varepsilon$$

Proof. RICORDARSI DI FARLA □

This theorem tell us that, for a sufficiently large m, the $ERM_{\mathcal{H}}$ rule over a finite hypothesis class will be *probably* (with confidence $1 - \delta$) *approximately* (up to an error of ε) correct.
Notes on the theorem:

- $|\mathcal{H}|$ is the cardinality (i.e. the dimension) of the hypothesis class

- The condition $m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ is just a sufficient (not necessary) condition. In fact it is a nearly weak request.

- m does not depend on f or on $\mathcal{D}$.

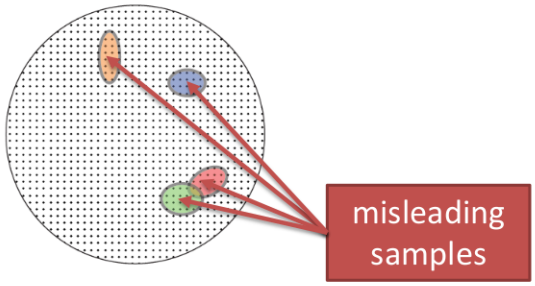

Figure 1.2: Each point in the large circle represents a possible m-tuple of instances. Each colored oval represents the set of "misleading" m-tuple of instances for some "bad" predictor $h \in \mathcal{H}_B$. The ERM can potentially overfit whenever it gets a misleading training set S. That is, for some $h \in \mathcal{H}_B$ we have $L_S(h) = 0$. The disequations found in the demostration of the theorem guarantee that for each individual bad hypothesis, $h \in \mathcal{H}_B$, atmost $(1 - \varepsilon)^m$-fraction of the training sets would be misleading. In particular, the larger m is, the smaller each of these colored ovals becomes. The union bound formalizes the fact that the area representing the training sets that are misleading with respect to some $h \in \mathcal{H}_B$ (that is, the training sets in M) is atmostthe sum of the areas of the colored ovals. Therefore, it is bounded by $\mathcal{H}_B$ times the maximum size of a colored oval. Any sample S outside the colored ovals cannot cause the ERM rule to overfit.

Chapter 2

PAC Learning

The title of this chapter stands for *Probably Approximately Correct* learning: since the data are sampled accordingly to $\mathcal{D}$, and since we can't find the perfect machine learning algorithm:

- we can only be approximately correct (accuracy parameter ε: we are satisfied with a good h_S for which $L_{\mathcal{D},f}(h_S) \leq \varepsilon$)
- we can only be probably correct (we want h_S to be a good hypothesis with probability $\geq 1 - \delta$)

Definition 1. ***(PAC learnability)***
A hypothesis class $\mathcal{H}$ is PAC learnable if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{0,1\}$, if the realizability assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of examples): $\ell_{\mathcal{D},f}(h) \leq \varepsilon$.

Note: $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ is the sample complexity of learning $\mathcal{H}$, in particular is the minimal integer that satisfies the requirements. This means that if we have an infinite number of possible hypotesis we cannot apply the theorem, but, at the same time, we can't be sure that the set is not learnable.

Corollary 1. *Every finite (sufficient condition, not necessary, as said before) hypothesis class is PAC learnable with sample complexity*

$$m_{\mathcal{H}}(\varepsilon, \delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon} \right\rceil$$

The definition of PAC learnability contains two approximation parameters. The accuracy parameter ε determines how far the output classifier can be from the optimal one and a confidence parameter δ indicating how likely the classifier is tomeet that accuracy requirement.

Now we would like to generalize this model, and we can do it dropping some assumptions we have made before:

- We remove the realizability assumption: it was interesting from a theorical point of view but too strong in many real world applications. Rejecting this means that there exist $h^* \in \mathcal{H}$ such that $L_{\mathcal{D},f}(h) = 0$.

- In many application it is not too realistic that the labeling is fully determined by the features we measure; for this reason it is convenient to replace the function f with something more flexible, for example assuming that $\mathcal{D}$ is a probability distribution over $\mathcal{X} \times \mathcal{Y}$ (i.e. the joint distribution over domain points and labels). One can view such a distribution as being composed of two parts: a distribution $\mathcal{D}_x$ over unlabeled domain points (*marginal distribution*) and a *conditional* probability over labels for each domain point $D((x, y)|x)$.

2.1 The empirical and the true error

For a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ we redefine the true error (or risk) of a prediction rule h to be

$$L_{\mathcal{D}}(h) = \mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] = \mathcal{D}([(x, y) : h(x) \neq y]) \tag{2.1}$$

Note: in the previous definition 1.1 the request was $[h(x) \neq f(x)]$ but, as we said before, the function $f(x)$ doesn't exist anymore because it was a too strong assumption asking that the labels were fully determined by a single function.
The definition of the empirical risk, instead, remains the same as before

$$L_S(h) = \frac{[i \in [m] : h(x_i) \neq y_i]}{m}$$

and still represent the probability that, for a pair (x_i, y_i) taken uniformily at random training data, the event "$h(x_i) \neq y_i$" holds.

2.2 The Bayes optimal predictor

Given any probability distribution $\mathcal{D}$ over $\mathcal{X} \times [0, 1]$, the best label predicting function (the one that minimize $L_{\mathcal{D}}(h)$) will be

$$f_{\mathcal{D}}(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y = 1|x] \geq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

It's easy to verify that the predictor $f_{\mathcal{D}}$ is optimal, in the sense that any other classifier $g : \mathcal{X} \to [0, 1]$, has a lower error, $L_{\mathcal{D}}(f_{\mathcal{D}}) \leq L_{\mathcal{D}}(g)$.
This optimal predictor is interensting from a theorical point of view, but unfortunately is not usable in practice because e don't know the distribution $\mathcal{D}$ and, consequently $\mathbb{P}[y = 1|x]$.

2.3 Agnostic PAC Learnability

Clearly, we cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error, that of the Bayes predictor. Furthermore, as we shall prove later, once we make no prior assumptions about the data-generating distribution, no algorithm can be guaranteed to find a predictor that is as good as the Bayes optimal one. Instead, we require that the learning algorithm will find a predictor whose error is not much larger than the best possible error of a predictor in some given benchmark hypothesis class. Of course, the strength of such a requirement depends on the choice of that hypothesis class.

Definition 2. *(**Agnostic PAC Learnability**)*
A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0,1)$ and for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running the algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of the m training examples):

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon$$

Notes on the definition:

- The agnostic PAC learning generalizes the definition of PAC learning, in the sense that, if the realizability assumption holds, this new definition provides the same guarantee as the previous one.
- The realizability assumption would implies that the quantity $\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h')$, that represents the performance of the best possible classifier (for example the Bayes' one), is always equal to zero.

Anyway a learner can still declare success if its error is not much larger than the best error achievable by a predictor from the class $\mathcal{H}$. This is in contrast to PAC learning, in which the learner is required to achieve a small error in absolute terms and not relative to the best error achievable by the hypothesis class.

We now want to extend our model so that it can be applied to a wide variety of learning tasks. Let's consider 3 different possible problems:

- *Binary and Multiclass classification*: our training sample will be a finite sequence of (feature vector,label) pairs, the learner's output will be a function from the domain set to the label set, and, finally, for our measure of success, we can use the probability, over pairs, of the event that our predictor suggests a wrong label.
- *Regression*: in this task, one wishes to find some simple pattern in the data, a function between $\mathcal{X}$ and $\mathcal{Y} = \mathbb{R}$. For the loss function, we can't use the same of the previous case, we need a new one.

2.3.1 Generalized loss function

Given any set $\mathcal{H}$ (that plays the role of our hypotheses, or models) and some domain Z let ℓ be any function from $\mathcal{H} \times Z$ to the set of nonnegative real numbers, $\ell : \mathcal{H} \times Z \to \mathbb{R}+$. We call such functions loss functions.
We now define the **risk function** to be the expected loss of a classifier $h \in \mathcal{H}$, with respect to a probability distribution $\mathcal{D}$ over Z, namely:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}} \left[\ell(h, z) \right] \tag{2.3}$$

Similarly, we define the **empirical risk** to be the expected loss over a given sample $S = (z_1, \ldots, z_m) \in Z^m$, namely:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i) \tag{2.4}$$

We must keep in mind that the loss always depends on the problem, there isn't a universal one. Sometimes it is useful to create a "personalized" loss function.
Common loss functions:

- *0-1 loss*: commonly use in binary or multiclass classification (obviously is not the only possible solution, is just the simpler and most common one).

$$\ell_{0-1}(h,(x,y)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

- *Squared loss L2*: commonly used in regression, penalize few large errors.

$$\ell_{sq}(h,(x,y)) = (h(x) - y)^2$$

- *Absolute value loss L1*: commonly used in regression, penalize many small errors:

$$\ell_{abs}(h,(x,y)) = |h(x) - y|$$

Example 1. *The loss function depends on our problem! Let's image a machine learning algorithm that verify fingerprints to guarantee the access to something. There are essentially two types of error that the computer can do:*

- *False accept: when accepts an unauthorized user.*
- *False reject: when doesn't accept an athorized user.*

If, for example, a supermarket implement the algorithm to give discounts:

- *False reject is costly; just the customer gets annoyed.*
- *False accept is minor, the supermarket lose just a discount and the intruder left his fingerprints.*

If, instead, a similar algorithm has been implemented by CIA for the security of a certain area:

- *False reject can be tolerade.*
- *False accept is a disaster!*

This example want to show that often the loss function need to be "calibrated" by our knowledge on the problem, to understand which errors are tolerable and which are not.

Definition 3. *(**Agnostic PAC Learnability for General Loss Functions**)*
Recalling the definition 2, for which the hypothesis generate by the algorithm (with probability $\geq 1 - \delta$ over the training set) has a true error

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon$$

we can implement the definition we have written for the loss function deducing that:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}} \left[\ell(h, z) \right]$$

Chapter 3

Learning from Uniform Convergence

Recall that, given a hypothesis class $\mathcal{H}$, the ERM learning paradigm works as follow: upon receiving a training sample S, the learner evaluates the error of each $h \in \mathcal{H}$ on the given sample and outputs a member of the class that minimizes this empirical risk. The hope is that an h that minimizes the empirical risk with respect to S is a risk minimizer with respect to the true data probability distribution aswell. For this reason we want that the empirical error is a good approximation of the true error for all the solutions, and not only for the best one ($L_S(h)$ is similar to $L_{\mathcal{D}}(h)$, $\forall\, h$).

Definition 4. ***(ε-representative)***
A training set S is called ε-representative (with respect to domain Z, hypothesis class $\mathcal{H}$, loss function ℓ and distribution $\mathcal{D}$) if

$$\forall h \in \mathcal{H} \quad |L_S(h) - L_D(h)| \leq \varepsilon$$

This is a stronger request than previous ones because we do not more focus on the best hypothesis of the class.

Theorem 2. *Assume that the training set S is $\frac{\varepsilon}{2}$-representative. Then, any output of $ERM_{\mathcal{H}}(S)$ (i.e. any $h_S \in \underset{h\in\mathcal{H}}{argmin}\, L_S(h)$) satisfies:*

$$L_{\mathcal{D}}(h_S) \leq \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon$$

Proof. For every $h \in \mathcal{H}$,

$$L_{\mathcal{D}}(h_S) \leq L_S(h_S) + \frac{\varepsilon}{2} \leq L_{\mathcal{D}}(h) + \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = L_{\mathcal{D}}(h) + \varepsilon$$

where the first and the third inequalities are due to the assumption that S is $\frac{\varepsilon}{2}$- and the second inequality holds since h_S is an ERM predictor. □

The relation guarantee by the theorem is exactly the one required by the definition of agnostic PAC learnability: in fact, the consequence of this statement is that if, with probability at least $1-\delta$, a random training set S is ε-representative, then the ERM rule is an agnostic PAC learner.

Definition 5. ***(Uniform Convergence)***
A hypothesis class $\mathcal{H}$ has the uniform (same m for all h and all $\mathcal{D}$) convergence property (with respect to a domain Z and a loss function ℓ) if there exists a function $m_{\mathcal{H}}^{UC} : (0,1)^2 \to \mathbb{N}$ such that for every $\varepsilon, \delta \in (0,1)$ and for every probability distribution $\mathcal{D}$ over Z, if S is a sample of $m \geq m_{\mathcal{H}}^{UC}(\varepsilon, \delta)$ i.i.d. examples drawn from $\mathcal{D}$, then with probability $\geq 1-\delta$, S is ε-representative.

This means that the function $m_{\mathcal{H}}^{UC}$ measures the (minimal) sample complexity of obtaining the uniform convergence property, namely, how many examples we need to ensure that with probability at least $1-\delta$ the sample would be ε-representative.

Proposition 1. *If a class $\mathcal{H}$ has the uniform convergence property with a function $m_{\mathcal{H}}^{UC}$ then the class is agnostically PAC learnable with the sample complexity $m_{\mathcal{H}}(\varepsilon,\delta) \leq m_{\mathcal{H}}^{UC}(\varepsilon/2,\delta)$. Furthermore, in that case the $ERM_{\mathcal{H}}$ paradigm is a successful agnostic PAC learner for $\mathcal{H}$.*

Proposition 2. *Let $\mathcal{H}$ be a finite hypothesis class, let Z be a domain, and let $\ell : \mathcal{H} \times Z \to [0,1]$ be a loss function. Then:*

- *$\mathcal{H}$ enjoys the uniform convergence property with sample complexity*

$$m_{\mathcal{H}}^{UC}(\varepsilon,\delta) \leq \left\lceil \frac{\log(2|\mathcal{H}|/\delta)}{2\varepsilon^2} \right\rceil$$

- *$\mathcal{H}$ is agnostically PAC learnable using the ERM algorithm with sample complexity*

$$m_{\mathcal{H}}(\varepsilon,\delta) \leq m_{\mathcal{H}}^{UC}(\varepsilon/2,\delta) \leq \left\lceil \frac{2\log(2|\mathcal{H}|/\delta)}{\varepsilon^2} \right\rceil$$

Proof. RICORDARSI DI FARE PURE QUESTA □

3.0.1 The Discretization Trick

While the previous theorem only applies on finite hypothesis classes, there is a simple trick that allow us to get a very good estimate of the practical sample complexity of infinite hypothesis classes. In many real world application, in fact, we consider classes determined by a set of parameters in $\mathbb{R}$.
The solution arises when we use computers. Using a pc, in fact, we will probably maintain real numbers using floating point representation, say, of 64 bits. It follows that, in practice, our hypothesis class is parameterized by the set of scalar that can be represented using a 64 bits floating point number. There are at most 2^{64} such numbers; hence the actual size of our hypothesis class is at most 2^{64}. More generally, if our hypothesis class is parameterized by d numbers, in practice we learn an hypothesis class of size at most 2^{64d}.
Applying the proposition, we obtain that the sample complexity of such classes is bounded by

$$m_{\mathcal{H}}(\varepsilon,\delta) \leq m_{\mathcal{H}}^{UC}(\varepsilon/2,\delta) \leq \frac{2\log\left(2\frac{2^{64d}}{\delta}\right)}{\varepsilon^2}$$

This upper bound on the sample complexity has the deficiency of being dependent on the specific representation of real numbers used by our machine.

Chapter 4

The Bias-Complexity Trade-Off

In one of the previous chapters we saw that, unless one is careful, the training data can mislead the learner, and result in overfitting. To overcome this problem, we restricted the search space to some hypothesis class $\mathcal{H}$. Such an hypothesis class can be viewed as reflecting some prior knowledge that the learner has about the task – a belief that one of the members of the class $\mathcal{H}$ is a low-error model for the task.
Our main objective was, given a training set S and a loss function, to find a function $\hat{h}$ for which the error $L_{\mathcal{D}}(\hat{h})$ is small (that brought to the definition of Agnostic PAC learnability (2)). For this reason we needed a large hypothesis class to contain the best solution, but at the same time a good algorithm to find it.
But, is there some kind of universal learner, that is, a learner who has no prior knowledge about a certain task and is ready to be challenged by any task, that predicts the best $\hat{h}$ for any distribution $\mathcal{D}$?
So, what about using the set of *all* the functions from $\mathcal{X}$ to $\mathcal{Y}$ as the hypothesis class? With this assumption we would be sure that the solution (if the problem is solvable) is inside the class.

Theorem 3. ***(No-Free Lunch)***
Let A be a learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain $\mathcal{X}$. Let m be any number smaller than $|\mathcal{X}|/2$, representing a training set size. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ such that:

- *there exists a function $f : \mathcal{X} \to \{0,1\}$ with $L_{\mathcal{D}}(f) = 0$ (so a perfect solution with 0 loss)*
- *with probability of at least $1/7$ over the choice $S \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

This means that there isn't a universal learner: the algorithm can fail even if there is a perfect solution for the training set. In fact, the theorem guarantees the existance of this perfect solution (with 0 loss), but doesn't guarantee a good performance instead.
The key message is that for every ML algorithm there exist a task on which it fails even if another ML algorithm is able to solve it.

Idea of the proof: our training set is smaller than half of the domain. This means that we have no information on what happens on the other half, but we can suppose that there is some target function f that works on the other half in a way that contradicts our estimated labels.

Let's consider an ERM predictor over the hypothesis class of all the functions $f : \mathcal{X} \to \{0,1\}$. This class represent a lack of knowledge: every possible function is considered a good candidate.

According to the previous theorem, any algorithm that chooses its output from an hypothesis belonging to that class, and in particular the ERM predictor, will fail on some learning task. Therefore, the class is not PAC learnable, as formalized in the following corollary.

Corollary 2. *Let $\mathcal{X}$ be an infinite domain set and let $\mathcal{H}$ be the set of all the functions from $\mathcal{X}$ to $\{0,1\}$. Then, $\mathcal{H}$ is not PAC learnable.*

Proof. Let's assume, by way of contradiction, that the class is learnable and let's choose some $\varepsilon < 1/8$ and $\delta < 1/7$. By the definition of PAC learnability (**??**), it exists an algorithm A and an integer $m = m(\varepsilon, \delta)$, such that for any data-generating distribution over $\mathcal{X} \times \{0,1\}$, if for some function f in that domain, $L_{\mathcal{D}}(f) = 0$, then with probability $\geq 1 - \delta$ when A is applied to samples S of size m, generated i.i.d. by $\mathcal{D}$, $L_{\mathcal{D}}(A(S)) \leq \varepsilon$.
However, applying the Free-Lunch theorem, since $|\mathcal{X}| > 2m$, for every learning algorithm (and in particular for A), there exists a distribution $\mathcal{D}$ such that with probability greater than $1/7 > \delta$, $L_{\mathcal{D}}(A(S)) > 1/8 > \varepsilon$, which leads to the desired contradiction. □

But how can we prevent such failures? We should, for example, use our knowledge on a specific learning task to avoid distributions that will cause us to fail.
But how should we choose a good hypothesis class? On the other hand we want to believe that this class includes the hypothesis that has no error at all (in the PAC setting), or at least that the smallest error achievable by a hypothesis from this class is indeed rather small (in the agnostic setting). On the other hand, we have just seen that we cannot simply choose the richest class i.e. the class of all functions over the given domain.

- If we choose a *small* $\mathcal{H}$ we will deal with *low approximation capabilities* (large L_S) but *good generalization properties* ($L_{\mathcal{D}} \sim L_S$). So the approximation found will work similarly on real data as on the training set, but it's not guarantee that the best hypothesis is in the class.
- If we choose a *large* $\mathcal{H}$, instead, we will have *good approximation capabilities* (small L_S) but there is a bigger *risk of overfitting* ($L_{\mathcal{D}} \gg L_S$). Moreover the no free lunch theorem suggests to not have too larger hypothesis classes.

4.1 Error Decomposition

To answer these questions we decompose the true error of an $ERM_{\mathcal{H}}$ predictor, h_S into two components, as follows:

$$L_{\mathcal{D}}(h_S) = \varepsilon_{app} + \varepsilon_{est}$$

- **Approximation error**

  $$\varepsilon_{app} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$$

  It is the minimum risk achievable by a predictor in the hypothesis class. It does not depend on the training set, or on the sample size, but only on the choice of the class. Enlarging the hypothesis class can decrease the approximation error. Under the realizability assumption, this error is zero.

- **Estimation error**

  $$\varepsilon_{est} = L_{\mathcal{D}}(h_S) - \min_{h \in \mathcal{H}}(h)$$

It is the difference between the approximation error and the error achieved by the ERM predictor. The estimation error results because the training error is only an estimate of the true risk, and so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk. This type of error depends on the training set size and on the size, or complexity, of the hypothesis class. For a finite class, the estimation error increases (logarithmically) with $|\mathcal{H}|$ and decreases with m (so the training error becomes a good estimate of the true error).

The following pictures schematized what we just said regarding the two types of error.
Since our goal is to minimize the total risk, we face a *tradeoff*, called the **bias complexity tradeoff**. On one hand, choosing $\mathcal{H}$ to be a very rich class decreases the approximation error but at the same time might increase the estimation error, as a rich $\mathcal{H}$ might lead to *overfitting*. On the other hand, choosing $\mathcal{H}$ to be a very small set reduces the estimation error but might increase the approximation error or, in other words, might lead to *underfitting*.
The differences between overfitting and underfitting can be seen in picture **??**.

We need to estimate the generalization error $L_{\mathcal{D}}(h)$ for a function h (the one selected with ERM). To do this, we use a *test set*, a new set of samples different (disjoint) from the training set used to pick h. Sometimes we use also a *validation set* for selecting the hyper-parameters of the algorithm or to evaluate errors while iterative training procedures are running.
To summarize the procedure of training a ML algorithm parametrized by some Hyper-Parameters (HP):

1. Select Hyper-Parameters values
2. Train the algorithm with the choosen parameters on the training set
3. Evaluate performances on the validation set
4. Go back to 1. and select new HP values
5. Select the final HP leading to the smallest validation error
6. Compute error estimation on the test set