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**: $S = ((x_1, y_1), \dots, (x_m, y_m))$ 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 $\mathbb{P}_{x\sim\mathcal{D}}\left[\pi(x)\right]$, where $\sim$ indicates that x point is sampled according to $\mathcal{D}$.
Now, we define the **error of prediction rule**:

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

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 $h_S : \mathcal{X} \to \dagger$ (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**:

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

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 $\mathcal{L}_S$, is called *Empirical Risk Minimization* or ERM, 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 **??**, 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 **??**, 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 $L_{\mathcal{D}}(h_S) = \frac{1}{2}$.
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:

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

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.