Supervised Learning

Matthieu R. Bloch

1 A framework for supervised learning

One of the main objectives of the course is to understand *why* and *how* we can learn. Although we all have an intuitive understanding of what learning means, making clear mathematical statements requires us to explicitly specify the components of a learning model. Without such clear statements, it would be hard to reason about learning and we would not be able to design an engineering methodology.

Definition 1.1. *Assume that there exists an* unknown *function* $f : \mathbb{R}^d \to \mathbb{R}$ *that takes a feature vector* $\mathbf{x}$ *as input and outputs a label* $y = f(\mathbf{x})$. *The supervised learning problem consists of the following components.*

1. *A* dataset $\mathcal{D} \triangleq \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ *comprised of* N *pairs of* feature vectors $\mathbf{x}_i$ *and their associated labels* y_i. *Our goal is to use* $\mathcal{D}$ *to infer something about* f.
   - $\{\mathbf{x}_i\}_{i=1}^N$ *are assumed to be drawn independent and identically distributed (i.i.d.) from an* unknown *probability distribution* $P_{\mathbf{x}}$ *on* $\mathbb{R}^d$
   - $\{y_i\}_{i=1}^N$ *are the corresponding labels, which are assumed to be drawn according to an* unknown *conditional distribution* $P_{y|\mathbf{x}}$ *on* $\mathbb{R}$.
2. *A set of hypotheses* $\mathcal{H}$ *containing candidate functions that could explain what* f *is.*
3. *A* loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ : (\hat{y}, y) \mapsto \ell(\hat{y}, y)$ *capturing the cost of making a prediction* $\hat{y}$ *instead of* y.
4. *An* algorithm ***ALG*** *to find the* $h \in \mathcal{H}$ *that best explains* f *in terms of minimizing the cost incurred by* h.

There are many subtle aspects behind this definition that we now discuss in details.

The assumption that f exists is not innocent. If you do not believe that there exists a magic formula to distinguish pictures of cats from pictures of dogs then there is nothing to learn! Another implicit assumption is also that we cannot derive f from first principles in mathematics and physics, which we shall call a *top-down* approach. If we could infer f using a top-down approach, there would be no need to learn f from data. Most traditional engineering disciplines follow a top-down approach and this often works extremely well. Machine learning is only useful if you face a situation in which the function f is too complicated to be derived from first principles. Assuming this is the case, machine learning takes a *bottom-up* approach and exploits data to infer what f could be.

The dataset provides examples of what the function f computes, and we hope to identify f through these examples. The fact that the data consists of feature vectors together with a label is what makes the learning problem *supervised*. Acquiring data is sometimes costly and difficult, therefore a related question that we will try to answer is *how much* data is required to learn. Having data is not enough to talk about learning in a mathematical way. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, one can come up with infinitely many different ways of explaining how labels are associated to the feature vectors. Said differently, there could be infinitely many *possible* ways of explaining how the labels are obtained. The key insight to circumvent this problem is to assume the existence of an *unknown* distribution $P_{\mathbf{x}}$ from which the feature vectors are sampled i.i.d.. Note that we only assume that $P_{\mathbf{x}}$ exists and not that it is known; however, the existence of a probability distribution will allow us to make statements about what function f is *probable*. Saying that the dataset consists of i.i.d. samples is a means of saying that samples have to be representative examples of what f predicts. For instance, it would be hard to distinguish cats from dogs if all our examples consisted of pictures of the *same* cat.

We also assume that the labels are generated from the feature vectors according to a conditional distribution (Probability Mass Function (PMF) or Probability Density Function (PDF)) $P_{y|\mathbf{x}}$. There is no loss of generality since this includes situations in which the labels are deterministic functions of the feature vectors. However, by allowing the labels to be a random map of the feature vectors, we allow the possibility that i) we could observe noisy labels of the form $f(\mathbf{x})+n$ where n is some noise; or ii) that there might not be absolutely true labels because some samples are inherently confusing. Note that the roles of $P_{y|\mathbf{x}}$ and $P_{\mathbf{x}}$ are different in our model.

Assuming that we try to explain f by picking a candidate h in a set $\mathcal{H}$ does not in principle constitute a loss of generality. In principle, we could pick $\mathcal{H}$ to consists of all possible functions in the universe. However, we shall see that there is a compromise to be made when choosing the set $\mathcal{H}$. For now suffice to say that $\mathcal{H}$ should be rich enough to explain in part what f computes but not so large that we could memorize the dataset. In practice, $\mathcal{H}$ could be a the set of neural nets with a specific architecture (number of layers, neurons, activation functions, etc.).

Our model also includes a loss function $\ell(\cdot,\cdot)$, which is crucial to measure the *performance* of a candidate function $h \in \mathcal{H}$ through $\ell(h(\mathbf{x}), y)$. Without a cost function, we cannot quantify how great or how poor this specific choice of h is. Our model, however, does not dictate which loss function to choose. We will see various choices of loss functions throughout this class, and which one to use is ultimately application dependent.

Finally, given a dataset, a set of hypotheses, and a loss function, one needs an algorithm to select a good (ideally the best) function $h \in \mathcal{H}$ to explain f. We clarify what we mean by "good" in the next section. For now, suffice to say that the algorithm is the machinery that learns f from the dataset.

2 Generalization and empirical risk

An important aspect of learning is that it should be different from memorizing the dataset. Said differently, our goal is not to find $h \in \mathcal{H}$ that accurately assigns values to elements of $\mathcal{D}$ but to find $h \in \mathcal{H}$ that accurately predicts values of *unseen* samples.

Consider a hypothesis $h \in \mathcal{H}$ that we somehow learned from the dataset. To quantify the quality of the choice h, we can compute the *empirical risk* (a.k.a. in-sample error) of the dataset as

$$\widehat{R}_N(h) \triangleq \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, h(\mathbf{x}_i)). \tag{1}$$

However, what we really care about is the *true risk* (a.k.a. out-sample error)

$$R(h) \triangleq \mathbb{E}_{\mathbf{x}y}(\ell(y, h(\mathbf{x}))), \tag{2}$$

which represents the average performance of h on an unseen sample drawn according to $P_{y|\mathbf{x}}P_{\mathbf{x}}$.

A central question of learning is whether one can *generalize* h, in the sense of quantifying whether the realization of $\widehat{R}_N(h)$ is likely to be close to $R(h)$. Another central question is whether we can learn well, in the sense of trying to identify the best hypothesis is $h^\sharp \triangleq \operatorname{argmin}_{h\in\mathcal{H}} R(h)$ that minimizes the true risk. We could design an algorithm called *empirical risk minimization* that could find $h^* \triangleq \operatorname{argmin}_{h\in\mathcal{H}} \widehat{R}_N(h)$ but it is not a priori obvious if $\widehat{R}_N(h^*)$ close to $R(h^\sharp)$. Furthermore, it is not even clear if $R(h^\sharp)$ is small.

Remark 2.1. *Minimizing the empirical risk is not the only way to select a good candidate $h \in \mathcal{H}$. We shall see later examples of algorithms (support vector machines) that learn by minimizing other metrics. For now, we shall concentrate on empirical risk minimization to answer the questions raised above.*