Introduction to learning theory

EECE 580G
Binghamton University

1 Reading assignment

This lecture covers a part of Chapter 5 from the textbook. Although the directions we will take here are slightly different from the textbook. Please read the following sections (9 pages) before this lecture:

- 5. Intro
- 5.1. Learning Algorithms. 5.1.1 to 5.1.3

2 Introduction

The main purpose of this handout is to lay out the basics of what we will call "learning theory," the building blocks of Machine Learning and Deep Learning. Learning theory aims to understand the fundamental principles of learning as a computational/mathematical process. The goal of this introduction is to define a learning algorithm and some of the various quantities involved in building such algorithms.

We will consider the following setting:

- m : samples or observations $x^{(i)}$ are collected from a subject of interest (e.g. for a pet classification task, samples are photos from multiple pets)
- We consider these observations to be labeled, i.e. each sample $x^{(i)}$ is assigned a label $y^{(i)}$ (e.g. for each pet photo we collected, we know the breed of the pet appearing in it)
- This setting is called "supervised learning"[1]
- $D = \left(x^{(i)}, y^{(i)}\right)_{i=i}^{m}$: is the set of the available tuples of observations and labels, this is the dataset at hand
- $X \to x^{(i)}$, and $Y \to y^{(i)}$: $x^{(i)}$ and $y^{(i)}$ are i.i.d. samples of the random variables X and Y
- $X \sim P_X$, and $Y \sim P_Y$,
- $(X, Y) \to D$ the dataset is an observation of the joint random variable (X, Y)
- $x^{(i)} \in \mathcal{X} = \mathbb{R}^n$: n is the dimensionality of the data (e.g. considering the pet photos being of size $H \times W$, $n = 3 \cdot H \cdot W$, the total number of pixels in one image represented in RGB space)
- $y^{(i)} \in \mathcal{Y} = \{0, 1\}$: we consider the simple case of a binary label (e.g. only two pet breeds) the task we are performing is a binary classification task[2]
- $D \in \mathcal{D} = \mathbb{R}^n \times \{0, 1\}$: the cartesian product of $\mathcal{X}$ and $\mathcal{Y}$

[1] Note that other branches of learning theory consider the case when no label $y^{(i)}$ is available "unsupervised learning," we will see more of these branches later.

[2] Note that in general $y^{(i)} \in \mathcal{Y}$, for a finite $\mathcal{Y}$ we will typically call the task a "multi-class" classification task, for a continuous $\mathcal{Y}$ we will typically call the task a regression task.

- We also consider that there exists a "ground truth" function f, that maps X to Y, i.e. $f(X) = Y$. This function is obviously unknown[3]
- To link with the definition from the textbook:
  - Task T = binary classification task
  - Experiences E = the dataset D
  - Performance measures P = loss function and notion of risk which will be introduced later

3 Functional definition of a learning algorithm

A learning algorithm can be defined functionally as an algorithm that outputs a desirable approximation $\widehat{f}$ of the unknown function f. In other terms:

$$\begin{aligned} \text{A learning algorithm}\colon \mathcal{D} &\to \mathcal{H} \subset \mathcal{Y}^{\mathcal{X}} \\ D &\mapsto \widehat{f}, \end{aligned}$$

A learning algorithm is a function that inputs a dataset D and outputs a function $\widehat{f} \in \mathcal{H}$. $\mathcal{H}$ is a subset of $\mathcal{Y}^{\mathcal{X}}$ (the set of the possible functions from $\mathcal{X}$ to $\mathcal{Y}$).

The approximated function $\widehat{f}$ should be the closest possible to f, i.e. $\forall (x, y) \leftarrow (X, Y)$, $\widehat{f}(x) = \widehat{y} \simeq y$. Where $\widehat{y}$ is called the prediction.

Note that we only have **a limited number of samples** from X and Y, but the goal is to approximate f for all possible samples. This is called "generalization." In the next session we will explore how to ensure that a learning algorithms outputs a suitable approximation $\widehat{y} \simeq y$.

4 Example: 1-Nearest-Neighbor algorithm

$$\widehat{f}(x) \mapsto y^{(j)}, j = \text{argmin}_i \left\| x - x^{(i)} \right\|$$

The 1-Nearest-Neighbor (1-NN) function approximation is simply the label of the closest sample in D. This kind of functions are called Voronoi partitions, $\mathcal{H}$ is the set of all Voronoi partitions of $\mathcal{X} = \mathbb{R}^n$. Figure 1 is drawn for $n = 2$ and shows a Voronoi partition function. Each new observation x will be assigned the color (label) of the closest sample in D, D has 10 samples (blue points).

Note that $\mathcal{H}$ is infinite, but given any m data samples, $\mathcal{H}$ can be (effectively) described using m parameters (the data points themselves). This is called the "representational capacity" c, i.e. the number of parameters that you can vary to chose a function from $\mathcal{H}$, the example of the 1-NN is a special example, usually the capacity of a learning algorithm is fixed and does not depend on the dataset.

This is one of the simplest learning algorithms. We will see soon more interesting ones (Linear/Logistic regression, deep neural networks, etc.)

5 Loss functions

In order to quantify the goodness of an approximation $\widehat{f}$, we define a loss function which informs of the "cost" of predicting $\widehat{y}$ instead of y: $l\left(\widehat{f}(x), y\right)$.

5.1 0-1 Loss

$$l\left(\widehat{f}(x), y\right) = [\widehat{f}(x) = y]$$

[3]Note that the results described in this handout can be generalized to cases when a "perfect" ground truth function doesn't exist, in which case f will correspond to the best possible mapping function (e.g. Bayes optimal detection)

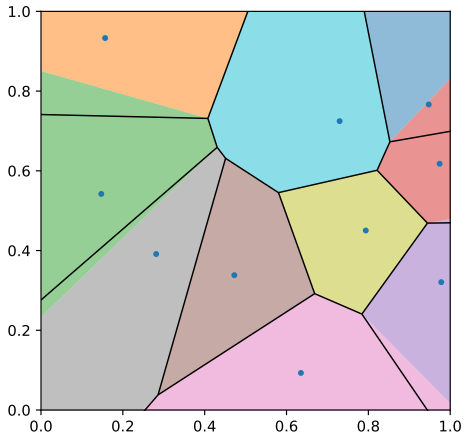

Figure 1: Voronoi diagram

Where [.] is the Iverson bracket which converts any logical proposition into a number that is 1 if the proposition is satisfied, and 0 otherwise.

The 0-1 Loss assigns a loss of 1 in case of error, and 0 in case of a correct prediction

5.2 Loss matrix

For classification tasks, $l\left(\widehat{f}(x), y\right)$ can be seen as a matrix $C \in \mathbb{R}^{|\mathcal{Y}|\times|\mathcal{Y}|}$, for binary classification it is a 2×2 matrix:

$$C = \begin{pmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{pmatrix}$$

- $c_{0,0}$ is the cost of a correct prediction of a negative sample (true negative)
- $c_{1,1}$ is the cost of a correct prediction of a positive sample (true positive)
- $c_{0,1}$ is the cost of predicting $\widehat{y} = 0$ while $y = 1$ (false negative)
- $c_{1,0}$ is the cost of predicting $\widehat{y} = 1$ while $y = 0$ (false positive)

The 0-1 loss is the special case of $C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$.

6 Empirical vs Generalization risk

We call the empirical risk $\widetilde{L}(h)$ of a function h.

$$\widetilde{L}(h) = \frac{1}{m}\sum_{i=1}^{m} l(h(x^{(i)}), y^{(i)})$$

Which aggregates the loss values across the dataset. In order to find the "best" approximation of f, the learning algorithm chooses the function with the least empirical risk. We call these kind of algorithms "empirical risk minimizers:"

$$\begin{aligned} \text{Empirical risk minimizer}\colon \mathcal{D} &\to \mathcal{H} \subset \mathcal{Y}^{\mathcal{X}} \\ D &\mapsto \widehat{f} = \text{argmin}_{h\in\mathcal{H}}\widetilde{L}(h), \end{aligned}$$

We now define the generalization risk $L(h)$ of a function h as:

$$L(h) = E\left[l(h(X), Y)\right].$$

And the generalization risk minimizer f^*:

$$f^* = \text{argmin}_{h \in \mathcal{H}} L(h).$$

Let's note that the empirical risk is only used as a proxy to the generalization risk, which we can't compute. But the ultimate goal of a learning algorithm is the generalization risk, because minimizing it ensures that the learned function will generalize beyond the training set. Note that for a given function $h \in \mathcal{H}$ independent of $\left(x^{(i)}, y^{(i)}\right)_{i=1}^{m}$:

$$E\left[\widetilde{L}(h)\right] = E\left[\frac{1}{m}\sum_{i=1}^{m} l(h(x^{(i)}), y^{(i)})\right] = E\left[l(h(X), Y)\right] = L(h)$$

However, this is not true for $\widehat{f}$ the output of the empirical risk minimizer, because $\widehat{f}$ depends on $D = \left(x^{(i)}, y^{(i)}\right)_{i=1}^{m}$ (it takes it as an input), which means that $l(\widehat{f}(x^{(i)}), y^{(i)})$ are not i.i.d anymore. Thus

$$E\left[\widetilde{L}(\widehat{f})\right] \neq L(\widehat{f})$$

We say that $\widetilde{L}(\widehat{f})$ is a biased estimator of the generalization risk. Which means that in expectation the empirical risk minimizer $\widehat{f}$ is different from the generalization risk minimizer f^*, which is unknown and in practice impossible to compute (because we don't know the joint probability (X, Y) to compute the expectation).

Recall that:

$$\begin{aligned} \widehat{f} &= \text{argmin}_{h \in \mathcal{H}} \widetilde{L}(h) \\ f^* &= \text{argmin}_{h \in \mathcal{H}} L(h) \end{aligned}$$

Proposition 1. *Bound 1*

$$L(f^*) \leq L(\widehat{f}) \leq L(f^*) + 2 \max_{h \in \mathcal{H}} \left| L(h) - \widetilde{L}(h) \right|$$

Proof. The first inequality is trivial

For the second part:

$$\begin{aligned} L(\widehat{f}) &= L(\widehat{f}) - \widetilde{L}(\widehat{f}) + \underbrace{\widetilde{L}(\widehat{f}) - \widetilde{L}(f^*)}_{\leq 0} + \widetilde{L}(f^*) - L(f^*) + L(f^*) \\ &\leq L(\widehat{f}) - \widetilde{L}(\widehat{f}) + \widetilde{L}(f^*) - L(f^*) + L(f^*) \\ &\leq L(f^*) + \left| L(\widehat{f}) - \widetilde{L}(\widehat{f}) \right| + \left| \widetilde{L}(f^*) - L(f^*) \right| \\ L(\widehat{f}) &\leq \underbrace{L(f^*)}_{\text{Bias}} + \underbrace{2 \max_{h \in \mathcal{H}} \left| L(h) - \widetilde{L}(h) \right|}_{\text{Fluctuation upper bound}} \end{aligned}$$

□

- $L(\widehat{f})$: The expected risk of the empirical risk minimizer
- $L(f^*)$: The expected risk of the generalization risk minimizer
- $\max_{h \in \mathcal{H}} \left| L(h) - \widetilde{L}(h) \right|$: The maximum fluctuation between the generalization risk and the empirical risk for any $h \in \mathcal{H}$

Note that the larger $\mathcal{H}$ is the higher $\max_{h \in \mathcal{H}} \left| L(h) - \widetilde{L}(h) \right|$ will be, however the larger $\mathcal{H}$ is the smaller $L(f^*) = \min_{h \in \mathcal{H}} L(h)$. This is called the **bias-variance trade-off**. Increasing the size of the possible functions $\mathcal{H}$, therefore allowing more "complex" functions will make the generalization error smaller, but will also make the fluctuation (variance) term larger... We need to find the trade-off (usually done empirically). When the Bias term is dominant, we say the the learning algorithm is "underfitting," when the fluctuation term is dominant we say that the learning algorithm is "overfitting."

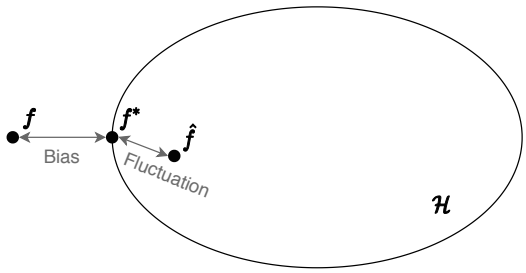

Figure 2: Schematic view of the function space $\mathcal{H}$, f, $\widehat{f}$, and f^*.

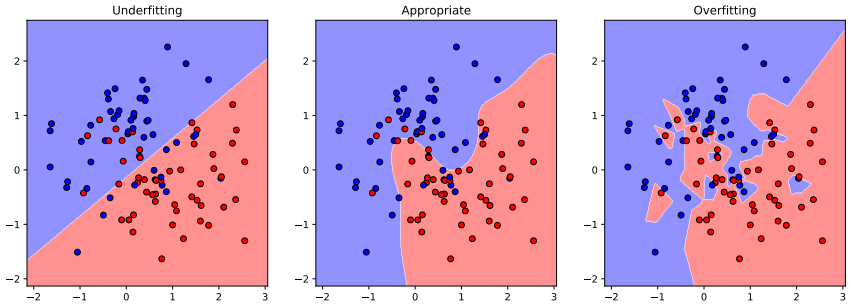

Figure 3: Example of 3 different learning algorithms, from left to right: an underfitting algorithm which produces a biased function approximation, a "good" learning algorithm, an overfitting learning algorithm which produces a high variance (fluctuation) function approximation.

7 The hypothesis space $\mathcal{H}$:

Recall that a learning algorithm is defined by a given function (hypothesis) space $\mathcal{H}$. We can come up with a tighter bound $L(\widehat{f})$ but only in probability.[4]

Proposition 2. *Bound 2*

With probability $\geq 1 - \delta$

$$L(\widehat{f}) \leq L(f^*) + 2\sqrt{\frac{\log(|\mathcal{H}|) + \log(2/\delta)}{2m}}$$

The proof is not included in this handout, but the bound comes from Hoeffding's concentration inequality. The second bound shows the same phenomenon as the first bound. A larger $\mathcal{H}$ increases the fluctuation term, but this can be fixed by collecting more data!

Note that this bound can also be generalized for infinite $\mathcal{H}$ (then writing $|\mathcal{H}|$ does not make sense), using the Vapnik-Chervonenkis dimension, which is not in the scope of this class. However the comments stay the same for infinite $\mathcal{H}$, enlarging $\mathcal{H}$ means for example enlarging the representational capacity c (number of parameters to define a function in $\mathcal{H}$), increasing c will again decrease the generalization error and increase the fluctuation term.

8 The training/validation set

Recall that in general, $E\left[\widetilde{L}(\widehat{f})\right] \neq L(\widehat{f})$ because $\widehat{f}$ depends on the data. This means that if we use all the training samples $D = \left(x^{(i)}, y^{(i)}\right)_{i=1}^{m}$ in the learning algorithm, we are enable to estimate the generalization error of the approximation $L(\widehat{f})$. To go around that, we usually split the data into

[4] This is called a PAC result: Probably Approximately Correct.

D^{Train} and $D^{\text{Validation}}$, where only D^{Train} is used in the learning algorithm, and $D^{\text{Validation}}$ is used to monitor $\widetilde{L}^{\text{Validation}}(\widehat{f}) = \frac{1}{m} \sum_{\left(x^{(i)}, y^{(i)}\right) \in D^{\text{Validation}}} l(\widehat{f}(x^{(i)}), y^{(i)})$ which is now an unbiased estimator of $L(\widehat{f})$.