Machine Learning Theory (CSC 482A/581B) - Lecture 1

Nishant Mehta

1 Administrative stuff

All the information about this course can be found on the course webpage:
`http://web.uvic.ca/~nmehta/ml_theory_spring2019`

Who is this course for?

This course is designed for two types of students:

- Students who previously have taken a course in machine learning and would like to gain a deeper understanding of the theory behind machine learning algorithms.
- Students from theoretical computer science who are curious about machine learning and would like an introduction to the theory of machine learning

Please note that this course is primarily for graduate students; however, advanced 4th year undergraduate students may also take the course if they have the requisite background.

Why is this course?

The goal of this course is to equip you with the ability to understand theoretical machine learning papers and begin doing research in machine learning theory. The emphasis will be on providing provable guarantees and understanding proofs. We will predominantly look at the *statistical theory* behind machine learning, but in the last third of the course we will study online learning.

This is a theory course and will not involve any implementation/coding.

Coursework

- Take-home quiz on first day of class - this does not count towards your grade but is required if you wish to register for (or remain registered in) the course
- 4 Problem sets: You will have at least 2 weeks to work on each problem set. You must work on problem sets individually.
- Take-home final exam: There will be a take-home final exam taking place near the end of the course and possibly overlapping with the examination period.
- Project (graduate students only)

Marks breakdown

Graduate students	
4 Problem sets	15% each
Take-home final exam	20%
Project	20%

Undergraduate students	
4 Problem sets	20% each
Take-home final exam	20%

Project

All graduate students will work on a project, starting around the midpoint of the course (at which time I also will release more information about the project and example topics). You may work either individually or in groups of 2. You will submit a brief project report and give a pedagogical presentation to the class near the end of the course or in the exam period. Because there is much to learn, all undergraduate and graduate students must attend the course project presentations.

2 Machine learning

The goal of machine learning is to devise algorithms that learn from experience. The "experience" typically is encapsulated by data consisting of *labeled examples* $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where

- The *input* x belongs to an *input space* $\mathcal{X}$, such as the set of all 100×100 pixel grayscale images;
- The *label* y belongs to a *label space* $\mathcal{Y}$, such as $\{0, 1\}$ in binary classification.

Given a collection of examples, a learning algorithm seeks to output a *hypothesis* $\hat{f} \colon \mathcal{X} \to \mathcal{Y}$ that maps each input $x \in \mathcal{X}$ to a predicted label $\hat{y} \in \mathcal{Y}$. The hypothesis $\hat{f}$ output by the learning algorithm is often restricted to belong to some set of hypotheses $\mathcal{F}$. Given the true label y and the predicted label $\hat{y}$, the way we charge for errors is determined by some notion of discrepancy between y and $\hat{y} = \hat{f}(x)$:

Definition 1. A *loss function* is defined as a mapping $\ell \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, where the first argument is the actual label and the second argument is the predicted label.

The two most widely used examples of loss functions are shown in the table below.

	$\ell(y, \hat{y})$	$\mathcal{Y}$	**Problem**
0-1 loss	$\mathbf{1}[\hat{y} \neq y]$	$\{0, 1\}$	binary classification
Squared loss	$(\hat{y} - y)^2$	$\mathbb{R}$	regression

3 Statistical learning vs Online learning

With the basic setup in place, we can now introduce the two fundamental types of learning problems that are the focus of this course: statistical learning and online learning. We will often describe examples as coming from some (potentially benevolent) opponent Nature and call the learning algorithm Learner.

3.1 Statistical learning

In a statistical learning problem, each labeled example is an independent and identically distributed (i.i.d.) draw from some fixed but unknown probability distribution P over $\mathcal{X} \times \mathcal{Y}$. Learner first observes a batch of n labeled examples drawn from P and is charged for errors according to how well it predicts the label of new examples drawn from the same distribution P. The statistical learning protocol is shown below.

Protocol:

1. Nature draws n labeled examples $(x_1, y_1), \ldots, (x_n, y_n)$ i.i.d. from P and reveals them to Learner.
2. Learner plays hypothesis $\hat{f} \in \mathcal{F}$.
3. Learner suffers error $\mathsf{E}_{(X,Y)\sim P}[\ell(Y, \hat{f}(X))]$.

In the above protocol, the n labeled examples often are called "training examples", and the collection thereof is referred to as the "training set"[1] or the "training data". In the last step of the protocol, the learner suffers some error based on its expected performance in making predictions on new examples drawn from the same distribution P. This expected future loss is called the risk:

Definition 2. Given a probability distribution P and a loss function ℓ, the *risk* of a hypothesis f is defined as

$$R(f) = \mathsf{E}_{(X,Y)\sim P}\left[\ell(Y, f(X))\right].$$

There is a clear connection between the training examples and the new, random example appearing in the definition of the risk, since they all are drawn from the same distribution P. Consider instead the following hopeless learning problem: let $\mathcal{Y} = \{-1, +1\}$, and let the training examples be drawn from P but the expectation in the risk instead taken with respect to a different distribution Q. Moreover, suppose that there is a function f^* such that under distribution P we always have $Y = f^*(X)$ while under distribution Q we instead have $Y = -f^*(X)$. Obtaining low risk in the new problem is impossible due to the complete mismatch between a good hypothesis for the risk under P versus a good hypothesis for the risk under Q.

Example 1. Spam classification is a by-now-classical example of binary classification. The training examples consist of a corpus of emails that are labeled spam or not spam. An email is represented as a binary vector that indicates whether not each word from a vocabulary of size d occurs in the email. The label $+1$ corresponds to "spam" and the label -1 corresponds to "not spam". After learning some hypothesis $\hat{f}$ based on the training set, Learner is assessed according to their misclassification probability on new examples, i.e., their expected 0-1 loss: $\mathsf{E}\left[\mathbf{1}[\hat{f}(X) \neq Y]\right] = \Pr(\hat{f}(X) \neq Y)$.

3.2 Online learning

Whereas in statistical learning, Learner observes a single batch of training examples and plays a single action (the hypothesis $\hat{f}$), in online learning the examples are revealed sequentially, and prior to observing the label of a given example Learner must first predict the label and suffer the corresponding loss. The online learning protocol is formalized below.

[1] Note, however, that they need not constitute a set and should instead be thought of as a sequence.

Protocol:

For round $t = 1 \to T$:

1. Nature selects example (x_t, y_t) and reveals x_t to Learner.
2. Learner plays[2] hypothesis f_t, yielding prediction $\hat{y}_t = f_t(x_t)$.
3. Nature reveals true label y_t and Learner suffers loss $\ell(y_t, \hat{y}_t)$.

In the online learning setting, we in general make no assumptions about Nature's choice for the sequence $(x_1, y_1), \ldots, (x_T, y_T)$. That is, not only is it possible that the examples are not drawn i.i.d. from some probability distribution; it might even be the case that Nature is *adversarial* and devises a sequence which reacts to the previous predictions of Learner in order to maximize the amount of loss suffered by Learner.

Consider for a moment the implications of a deterministic Learner playing against a fully adversarial Nature in the case of 0-1 loss. Since Learner's algorithm is deterministic, for any input x_t, Nature can always ensure that Learner makes a mistake by playing $y_t = -\hat{y}_t$. Thus, Learner always suffers the maximum possible cumulative loss of T against such an adversary. There are at least two potential resolutions to this trivial outcome:

1. Place restrictions on Nature, so that there is some underlying pattern that ties together the labels to the inputs. This is the restriction that we will look at for the time being.

2. Instead of seeking to minimize the cumulative loss suffered by Learner, instead seek to minimize the difference between Learner's cumulative loss and the cumulative loss of some benchmark online prediction strategy. We will explore this in more detail when we get to the online learning portion of this course.

4 Realizable case and the Mistake Bound Model

We begin our study of machine learning theory with classification under the strong but plausible assumption of *realizability*: there exists some boolean function $c \in \mathcal{C}$ which determines the labels, so that labeled examples always take the form $(x, c(x))$. Here, $\mathcal{C}$ is the *concept class*, a set of boolean functions. Learner knows $\mathcal{C}$ but of course does not know c. The reason for using the notation $\mathcal{C}$ rather than $\mathcal{F}$ is to allow for the possibility that Learner predicts according to a different set of hypotheses than those in $\mathcal{C}$.

We first study this setting under the online learning protocol. Consider an arbitrary sequence of examples, where the length of the sequence also can be arbitrarily large. Since we are in the realizable case, there exists a perfect hypothesis $c \in \mathcal{C}$. Suppose that Learner takes $\mathcal{F} = \mathcal{C}$, eventually plays this hypothesis, and predicts according to it thereafter. Then the total number of mistakes made by Learner is simply the number of mistakes made by Learner prior to playing hypothesis c. Thus, it is natural to try to bound the total number of mistakes made by Learner.

Definition 1. An algorithm $\mathcal{A}$ learns a class $\mathcal{C}$ in the *mistake bound model* if there there is a constant $M < \infty$ such that, for any $c \in \mathcal{C}$ and any sequence of examples $(x_1, c(x_1)), \ldots, (x_T, c(x_T))$ of any length T, the total number of mistakes made by $\mathcal{A}$ on the sequence is at most M.

[2]Technically, Learner need only specify a single prediction $\hat{y}_t$ for the particular x_t it received, rather than playing a hypothesis f_t belonging to some class of hypotheses $\mathcal{F}$. The protocol is presented using f_t so as to *(i)* better show the connection to the statistical learning protocol and *(ii)* at a later time facilitate study of learning algorithms that limit their predictions $\hat{y}_t$ by restricting $\mathcal{F}$ in different ways.

Two remarks are in order. First, for many concept classes there is a natural way to define the concept class for each choice of the dimension d of the input. We then would typically want a mistake bound M that grows only polynomially in d. Second, we typically also care about *efficient learnability*. That is, we would like the runtime of the algorithm to be at most $\mathrm{poly}(d)$ for each round t.

4.1 Learning monotone conjunctions

Let the input space be $\{0,1\}^d$ and the label space $\mathcal{Y}$ be $\{0,1\}$. Consider the concept class $\mathcal{C}$ consisting of the set of all monotone conjunctions. That is, each element of $\mathcal{C}$ is of the form $x_{i_1}x_{i_2}\cdots x_{i_k}$ for some $k \in \{0,1,\ldots,d\}$; if $k=0$, then the predicted label is always positive.

There is a simple algorithm for learning monotone conjunctions in the mistake bound model:

1. Initialize hypothesis f to the conjunction $f(x) = x_1x_2\cdots x_d$.

2. While there are still examples in the sequence:

   Predict the label of the next example using f. If the true label is 1 but the predicted label was 0, update f by removing from the conjunction all components x_j that are zero in the example.

The idea behind the above algorithm is to begin with the most restrictive hypothesis which labels everything but the all ones vector as negative. Thus, mistakes can occur only on positive examples. In the event that a mistake occurs on a positive example, we are guaranteed that any component set to zero in the example cannot be part of the correct conjunction c, and we may thus remove such components. The algorithm thus only removes terms from the conjunction in f which are inconsistent with the data observed thus far. Moreover, each mistake leads to the removal of at least one term from the conjunction, and so there can be at most d mistakes. Therefore, the above algorithm learns the class of monotone conjunctions in the mistake bound model and makes at most d mistakes.