Mathematical Foundations of Machine Learning(CS 4783/5783)

Lecture 1: Setting Up The Machine Learning Problem

1 What is Machine Learning?

In traditional computer science, given a task or a set of tasks, the computer scientist or programmer writes a program that performs the task. Example, say the last is to sort a given sequence of numbers. The programmer writes a program to sort, say the quick sort program which given a sequence of numbers sorts it and outputs the sorted sequence. This approach is great for simpler tasks such as sorting, searching, scheduling etc. But what about more complicated tasks. Say we want a program that given an image needs to identify if there is a cat in the image or not, or a program that can detect if an email is spam or not or a program that can use stream from lidar and camera as input and can automatically drive a car well in traffic? It is nearly impossible for a human to write a program with all its details to perform such tasks. The idea in Machine Learning (ML) is to use examples or past experiences to learn to automatically generate such programs. Arthur Lee Samuel coined the term Machine Learning and informally described it as: **"the field of study that gives computers the ability to learn without being explicitly programmed"**. A more elaborate version of this definition is:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". - Tom Mitchell

2 Setting up the Problem

$\mathcal{X}$ is the so called instance space, input space or all possible questions. $\mathcal{Y}$ is the outcome space, all possible labels or answers. A hypothesis or model or program $f : \mathcal{X} \mapsto \mathcal{Y}$, is a mapping from input space to outcome space that is a candidate program to perform the given task. Given an instance $x \in \mathcal{X}$ and label or outcome $y \in \mathcal{Y}$ and a model f, the loss function $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ measures how well our candidate program or hypothesis f performed on input output pair (x, y). Specifically, we consider the value of $\ell(f(x), y)$ and the larger it is, the worse f performed on instance (x, y). As an example, consider the task of classifying a given image as containing a cat in it or not. Given all the poses, sizes, background, possible quality of images, different species of cats, it is impossible to write a program explicitly for this task. If we consider setting up this problem as a ML problem, $\mathcal{X}$ would be the set of all images. $\mathcal{Y} = \{\text{“}cat\text{”}, \text{“}notcat\text{”}\}$ and since its a binary classification task, given a $y' \in \mathcal{Y}$, $\ell(y', y) = \mathbb{1}_{\{y' \neq y\}}$. Specifically, given a program or model $f : \mathcal{X} \mapsto \mathcal{Y}$, the loss of the model f at classifying an image x with label y is given by $\ell(f(x), y) = \mathbb{1}_{\{f(x) \neq y\}}$.

2.1 A Simple Example:

For this lecture, we are going to consider a simple setting to give you a taste of what kind of ML set up we can have, what style of algorithms one can consider and how to do some simple analysis of them. While the example will be simple, it will demonstrate some key terminology, concepts and help us set up the right framework for the right kind of ML problems. Specifically, for this example, we will consider the case of binary classification. That is label set $\mathcal{Y} = \{+1, -1\}$ (Eg. cat, not cat or spam, not spam etc). Further, assume that we have N models $f_1, \ldots, f_N$ and the problem is such that one of these models (say f_{i^*}) is always correct. That is $y = f_{i^*}(x)$ for any (x, y) pair. However, the ML scientist or ML algorithm does not know which of these N models is the correct one but only has access to these N models. Now we consider two scenarios and will try to build algorithms for each of these and figure out an analysis for them. But before we do that, let U (stands for universe) be the set of all input $x \in \mathcal{X}$ that our system would ever encounter (or be asked to label). Note that while one can think of $U = \mathcal{X}$, no system will be asked to label arbitrary images, for instance, typical images encountered would be natural images. Think of U at the set of images the learning algorithm might typically encounter. Now with this basic setting, let us consider two scenarios.

Scenario 1: For scenario one, we randomly draw n examples $x_1, \ldots, x_n$ from U (with replacement but since $|U| >> n$ this does not matter really), and ask a human to label them as $y_1, \ldots, y_n$ (assume the human labels them exactly and that $y_t = f_{i^*}(x_t)$). We will refer to this set of labeled examples $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ as training set. Think about the case where size of U is way larger than n. We would like to design an learning algorithm that given access to this training sample, and knows the set of models $f_1, \ldots, f_N$ (but not the true model) will output a model whose error on future draws from set U is small. That is, we would like to ensure that the probability of a mistake in future draw of instances is as small as possible.

What is the algorithm? What kind of guarantee can we provide on its error? How does our guarantee (bound) on error depend on N the number of models, on n the number of samples we drew?

Before we answer these questions, let us consider the problems for which such a framework is ideal for. Consider the example of image classification. In such a scenario, one can think of U as the set of all natural images we might consider, for instance all images on the web. As a training sample, we might randomly take images from the web and ask humans to label them. This would give us the training set. More importantly, when future images from the web are randomly presented to us, we are exactly the set up here. Of course, in practice this may not be the exact scenario. The way we obtain training samples from the web might have biases and further, when we use the learnt model, the distribution of instances we might be asked to predict or classify on might be different from the one the training instances are drawn from. But baring too much disparity between how we sample training and testing samples from U, this scenario which is often referred to as statistical learning problem is a good model or framework for many tasks ML systems are now famous for. Of course, our assumption that one of the N models is perfect or even that we have a finite N number of models is a bit naive but we will work at this through the course. But now back to the questions for this simple scenario.

Lets start with the question of algorithm. The only thing we know is that one of the N models is perfect, and that samples are drawn randomly from U and that in the future, samples are drawn from same distribution (that of randomly drawing from U). So perhaps the most straightforward

algorithm one can think of here is, take the training sample S, find a model $\hat{f}_S \in \{f_1, \ldots, f_N\}$ that makes 0 error on the training sample (one has top exist as the perfect one belongs to this set) and return that one. That is,

$$\hat{f}_S \in \left\{ f_i : i \in [N] \& \sum_{(x,y)\in S} \mathbb{1}_{\{f_i(x)\neq y\}} = 0 \right\} .$$

Now that we fixed our algorithm, how do we analyze it? Hint: our algorithm can pick any model consistent with training sample. So ask the question, what is the probability that a bad one will be consistent with our training sample.

Proposition 1. *For the algorithm above, for any $\delta > 0$, with probability at least $1 - \delta$ over draws of training samples (of size n),*

$$P(\hat{f}_S(x) \neq y) \leq \frac{\log(N/\delta)}{n}$$

Proof. Let $B_\epsilon = \{i \in [N] : P(f_i(x) \neq y) > \epsilon\}$. That is B_ϵ is the set of indices of all models (amongst the N models) that have probability of error larger than some ϵ. (the bad set). Now if we can bound the probability with which one of the bad models belonging to the above set, could be consistent with our training sample of size n, we would be done. To this end, consider an f_i such that $i \in B_\epsilon$. For any such f_i, if we draw (x, y) pair from U at random, the probability that we make an error is greater than ϵ by definition of B_ϵ. That is,

$$P(f_i(x) \neq y) = \mathbb{E}\left[\mathbb{1}_{\{f_i(x)\}} \neq y \right] \geq \epsilon$$

However, what is the probability that such an f_i is consistent with the training sample S? Well out of n random tries, where in each try, the probability we make an error is at least ϵ, the probability we make no errors is bounded by

$$P(\forall t \in [n], f_i(x_t) = y_t) = \prod_{t=1}^{n} P(f_i(x_t) = y_t) = (1 - P(f_i(x) \neq y))^n \leq (1-\epsilon)^n \leq \exp(-\epsilon n)$$

The above is because x_t, y_t are drawn independently so probability of being correct on every sample is the same. We also used the fact that $1 - x \leq \exp(-x)$. Next, note that by union bound,

$$P(\exists i \in B_\epsilon \text{ s.t. } \forall t \in [n], f_i(x_t) = y_t) \leq |B_\epsilon| \exp(-\epsilon n) \leq N \exp(-\epsilon n)$$

Thus we have thus shown that the probability that even one of the elements of B_ϵ (the bad set) could possibly be consistent with our training sample is upper bounded by $N \exp(-\epsilon n)$. Since this holds for any ϵ, for any $\delta > 0$, set $N \exp(-\epsilon n) = \delta$ (i.e. $\epsilon = \frac{\log(N/\delta)}{n}$). Then we notice that for any $\delta > 0$, the probability that an element of B_ϵ is consistent with the training set is bounded by δ. Thus, we conclude that with probability at least $1 - \delta$, no element of B_ϵ is consistent with training sample. Hence, with probability at least $1 - \delta$, any model consistent with training sample belongs to B_ϵ^c and so specifically for the model returned by our algorithm,

$$P(\hat{f}_S(x) \neq y) \leq \epsilon = \frac{\log(N/\delta)}{n}$$

□

It is important we parse the above proposition statement carefully. It says that probability of error/mistake is bounded with high probability over training sample. Such statements we will refer to as Probably Approximately Correct (PAC) framework. The approximately correct part refers to the fact that the learning algorithm in the end only has a guarantee that says error is bounded by something small and not that it is 0. The probably part refers to the fact that we have a probabilistic guarantee that the statement holds with probability at least $1 - \delta$. Of course we made the realizability assumption that our set of models has the perfect model. This is assumption is called realizability assumption. When one relaxes this assumption, the problem is called agnostic PAC. Agnostic standing for the fact that we will make no a priori assumptions on how y and x are related in general.

Scenario 2: Let us motivate the second scenario using a different example. Say we are interested in a binary classification problem again and still assume that once of the N models is perfect. Only this time, think about the application as being spam filtering. For the applications in the previous scenario, it made sense to assume that the sampling procedure to obtain training samples and the way we get future instances are the same. However, if we consider spam filtering example, then given the sample the system has collected so far consisting of examples of spam and not spam, we might try to build a model that classifies new emails as spam or not. However, in the mean time, the conniving spammer is also trying to beat you by picking the next instance not as a random element of U but rather by picking one that can be sneaked past the filter. In fact, for such applications it is obvious that one needs a ML system that is continually updating based on instances. So here scenario 2. Every round t, we are provided with a new instance $x_t \in X$ that could be picked in an arbitrary fashion (Eg. by a spammer wanting to thwart the system). Next the learning algorithm classifies this x_t into $+1$ or -1. Say $\hat{y}_t \in \mathcal{Y} = \{\pm 1\}$ is the classification for round t. Finally we assume that at the end of the round, after we make our classification, the true label $y_t = f_{i^*}(x_t)$ is revealed to us.

How do we design an algorithm for this scenario? How many mistakes will the algorithm make, can we bound it?

As a first cut, let us use the same algorithm (expect we update it after every round) as we used for scenario 1. That is, on each round t, we pick amongst $f_1, \ldots, f_N$ some function that is consistent with all the instance label pairs we have witnessed so far. How many mistakes will such an algorithm make? Well note that at time 0, we can assume the entire set $\mathcal{F}_0 = \{f_1, \ldots, f_N\}$ is consistent. Subsequently, each round we pick an element of $\mathcal{F}_t$ where $\mathcal{F}_t$ is the set of models consistent with data so far. Notice that each time we make a mistake, we reduce the size of consistent models by at least one since a mistake indicates that at least one model amongst the current set $\mathcal{F}_t$ did not agree with the true label. Therefore, we can guarantee that we make at most N mistakes. That is, for this algorithm,

$$\sum_t \mathbf{1}_{\{\hat{y}_t \neq y_t\}} \leq N$$

Is this the best we can do?

Lets try a slight modification to the earlier mentioned strategy. Just as above, we maintain on every round t a set $\mathcal{F}_t$ of models that are consistent with all the instance label pairs we have so far. Except, this time, instead of picking one member from this set and using that model to predict

label for x_t, we instead take a majority vote amongst all the models in $\mathcal{F}_t$. That is, we set

$$\hat{y}_t = \text{sign}\left(\sum_{f \in \mathcal{F}_t} f(x_t)\right)$$

Why does this help?

This strategy helps because each time we make a mistake, it means that at least half the models in $\mathcal{F}_t$ made a mistake.

Proposition 2. *Let $\mathcal{F}_t = \{f_i : i \in [N], \forall s < t, f_i(x_s) = y_s\}$. Define $\hat{y}_t = \text{sign}\left(\sum_{f \in \mathcal{F}_t} f(x_t)\right)$. For this strategy, we have the following mistake bound.*

$$\sum_t \mathbb{1}_{\{\hat{y}_t \neq y_t\}} \leq \log_2 N$$

Proof. Simply note that $\mathcal{F}_1 = \{f_1, \ldots, f_N\}$ and so $|\mathcal{F}_1| = N$. Now further, note that for any round t fo which $\hat{y}_t \neq y_t$, we know that majority of $f \in \mathcal{F}_t$ are wrong and so we have that those wrong models will be dropped the next round. Hence,

$$|\mathcal{F}_{t+1}| \leq |\mathcal{F}_t|/2$$

Hence we conclude that $|\mathcal{F}_{t+1}| \leq |\mathcal{F}_1|/2^{\sum_{j=1}^t \mathbb{1}_{\{\hat{y}_j \neq y_j\}}} = N/2^{\sum_{j=1}^t \mathbb{1}_{\{\hat{y}_j \neq y_j\}}}$. Thus we can guarantee that we can never make more than $\log_2 N$ number of mistakes because we know that for any t, $|\mathcal{F}_t|$ is at least 1 (the one classifier that is perfect has to be in there). This proves the bound. □

This second scenario is a simple example in the so called online learning setup.

3 What will we cover in this course?

In the previous section, the scenario one we presented is in the so called Statistical Learning framework. Scenario two discussed was in the so called Online Learning framework. The example set up we considered was of course toyish in that we assumed we had a perfect predictor (realizable setting) and that it is amongst N models. In this class we will look at both statistical and online learning frameworks and for much more general set ups. Beyond just binary classification problems, or even just classification problems. We will look at not only realizable but more generally agnostic settings.

While this is a tentative list, the syllabus/list of topics we will cover include:

1. Statistical Learning theory
   (a) Generalization Error, Training Vs Test loss, Model Complexity
   (b) PAC model and VC theory
   (c) Rademacher Complexity and Uniform convergence
   (d) Role of Regularization in learning, model selection and validation
   (e) Algorithmic Stability
2. Online learning

   (a) Online Bit prediction and Cover's result
   (b) Perceptron, winnow
   (c) Online experts problem
   (d) Online Gradient descent and Mirror descent
3. Boosting
4. Stochastic Optimization and Learning: including understanding Stochastic Gradient Descent
5. Bandit problems: both stochastic and adversarial settings
6. A primer to theory of deep learning: new challenges
7. Computational Learning theory: Computational hardness or learning, proper vs improper learning
8. Societal aspects of ML: Differential Privacy, Right to be Forgotten, Fairness and ML