E0 370 Statistical Learning Theory **Lecture 1 (Aug 9, 2011)**

Probabilistic and Game-Theoretic Settings of Machine Learning

Lecturer: Shivani Agarwal *Scribe: Shivani Agarwal*

1 Introduction

Learning is a natural phenomenon in human beings and plays a critical role in human understanding and decision-making. Often, it is desirable to implement such a phenomenon in machines or artificial systems. Consider the following examples:

1. **Medical diagnosis.** Suppose we are given gene expression profiles for some number of patients, together with labels for these patients indicating whether or not they had a certain form of a disease. Can we learn from these labeled cases a model which, given the gene expression profile for a new patient, can predict with high accuracy whether the patient will have the disease?

2. **Computer vision.** Suppose we have installed a security camera at a traffic signal, and for some number of scenes recorded by the camera, we have human-judged labels indicating whether the scene contains an accident. Can we learn from these labeled scenes a model which, given a new scene, can predict accurately whether an accident has occurred?

3. **Stock markets.** We would like to maximize returns on our investments in the stock market. We do not know how individual stocks or funds will perform, but at any given time, may have access to the funds' past performance, together with predictions of their future performance made by various experts. Can we develop an adaptive strategy for dynamically allocating investments across the funds so as to get good returns over the next 40 years?

4. **Indian Premier League.** Your friends have challenged you to predict the outcomes (e.g. winning team or total score) of cricket matches in the upcoming IPL season. Each of your friends has their own strategy for making such predictions, which they are willing to share with you. You do not know which friends' predictions will be accurate. Can you develop an adaptive strategy for predicting the outcome of each match that gives good performance over the season compared to any of your friends?

In all of these examples, one needs to design an algorithm or system that can learn from experience in order to improve future performance. In this course, we will be concerned with the theoretical analysis of such learning settings and of corresponding algorithms.

In Examples 1 and 2 above, one gets a sample or 'batch' of labeled training cases, and needs to learn a predictive model from these cases that can then be deployed to make predictions for new test cases; such situations are naturally modeled via a probabilistic setting that relates the training and test cases in a manner that makes learning possible. In Examples 3 and 4, one needs to adapt one's predictions in a dynamic or 'online' manner, and perform well over any fixed sequence of cases; such situations are naturally modeled in a game-theoretic setting in which learning is viewed as an adversarial game between the learner and the environment. We discuss these settings briefly below. This being a statistical learning theory course, we will focus mostly on the probabilistic setting, but toward the end of the course will also have a few lectures on the game-theoretic setting, both because it is an interesting setting in its own right and because it has interesting relationships with the probabilistic setting.

2 Probabilistic/Batch Setting

We will assume an *instance space* $\mathcal{X}$ from which instances (cases) are drawn, and an *outcome space* $\mathcal{Y}$ from which outcomes (labels) are drawn. E.g. in the medical diagnosis example above, if the gene expression profile for each patient consists of a vector of real-valued expression levels of n genes, then the instance space would be some set $\mathcal{X} \subseteq \mathbb{R}^n$; the label space in this case can be represented as $\mathcal{Y} = \{-1, 1\}$, where a label of 1 corresponds to having the disease and -1 to not having the disease. Given a finite sample of labeled examples $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$, the goal of a learning algorithm is to learn a function $h : \mathcal{X} \to \mathcal{Y}$ which given a new instance $x \in \mathcal{X}$, predicts its label to be $\hat{y} = h(x)$.

In order to have any hope of making accurate predictions on new instances, we must assume some sort of relationship between the examples given to the algorithm during training and the new instances it will receive in the future; in particular, the training examples must be 'representative' in some way of the examples to be seen in the future.[1] This is usually done by assuming that there is some underlying probability distribution D over the example space $\mathcal{X} \times \mathcal{Y}$, and that all training examples as well as test examples that will be seen in the future are generated independently from this distribution. Note however that the distribution D is not known to the learner; the learner sees the distribution only through the training examples, and based on these examples, must learn to predict well on new instances from the same distribution. In fact, for most of this course, we will not make any assumptions on the form of D; thus the results we discuss will be *distribution-independent* or *distribution-free* (i.e. they will hold for all possible distributions D).

In order to measure the performance of a learning algorithm, we will assume a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, where $\ell(y, \hat{y})$ measures the loss incurred on predicting label $\hat{y}$ when the true label is y. The performance of any function $h : \mathcal{X} \to \mathcal{Y}$ is then measured in terms of its *generalization error*, which is its expected loss on a new example (x, y) drawn according to D:

$$\mathrm{er}_D^{\ell}[h] = \mathbf{E}_{(x,y)\sim D}\left[\ell(y, h(x))\right] . \tag{1}$$

For example, in *binary classification* problems, where there are two labels $\mathcal{Y} = \{-1, 1\}$ (as in the medical diagnosis example above), a common loss function is simply the *zero-one loss* which counts an error of one if the wrong label is predicted, and zero otherwise:

$$\ell_{0\text{-}1}(y, \hat{y}) = \mathbf{1}(\hat{y} \neq y) = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Here $\mathbf{1}(\cdot)$ is an indicator function that takes the value 1 if its argument is true and 0 otherwise. In this case the generalization error of a classifier $h : \mathcal{X} \to \mathcal{Y}$ is simply the probability that it predicts the wrong label on a randomly drawn example:

$$\mathrm{er}_D^{0\text{-}1}[h] = \mathbf{P}_{(x,y)\sim D}\left(h(x) \neq y\right) . \tag{3}$$

In general, there may be different costs associated with different types of mistakes, and the loss function can be used to capture this. As another example, in *regression* problems, where the goal is to predict real-valued labels $\mathcal{Y} = \mathbb{R}$ (for example predicting the amount of rainfall in a region from a satellite image), a common loss function is the *squared loss*:

$$\ell_{\mathrm{sq}}(y, \hat{y}) = (\hat{y} - y)^2 . \tag{4}$$

Different choices for the outcome space $\mathcal{Y}$ and loss function ℓ lead to different learning problems: if there are $k > 2$ discrete labels $\mathcal{Y} = \{1, 2, \ldots, k\}$ and the zero-one loss is used, one gets a *multiclass classification* problem; if there are discrete labels with an ordering among them such as $\mathcal{Y} = \{1, 2, \ldots, k\}$ and the loss function takes this ordering into account, one gets an *ordinal regression* problem; for more complex outcome spaces $\mathcal{Y}$ and corresponding loss functions, one gets a *structured prediction* problem.[2] In this course, we will focus mostly on binary classification and regression problems.

Some comments on notation. 1. When intended to be implicit or clear from context, we will drop the loss function ℓ from notation such as $\mathrm{er}_D^{\ell}[h]$. 2. In the probability and statistics literature, it is common

[1] It is unrealistic to show me examples of only small flying birds such as sparrows and parakeets and expect me to generalize to an ostrich!

[2] It is also worth noting that there are natural learning problems (such as ranking) that differ from the setting described here. However such problems can often be analyzed via extensions of the methods that will be discussed in this course for the problems described here.

practice to denote random variables by capital letters. Here we use the more liberal practice of allowing random quantities to be denoted by lower case letters, as is common in computer science; it should be clear from context when a quantity is considered random. 3. The generalization error is often called the *risk*, a term that has its roots in statistical decision theory; consequently, in the literature, the notation $R(h)$ or $R_D[h]$ is often used in place of the notation $\mathrm{er}_D[h]$ we use here.

In the above probabilistic setting, given a training sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ drawn according to D^m, the goal of a learning algorithm is to learn a function $h_S : \mathcal{X} \to \mathcal{Y}$ that has low generalization error $\mathrm{er}_D[h_S]$ (with respect to an appropriate loss function). Typically, one designs a learning algorithm that selects a function h_S from some class of functions $\mathcal{H}$. Some natural questions arise:

First, in the finite sample setting:

- **Generalization error bounds.** Since D is unknown, the actual generalization error of a function cannot be computed directly. Can we estimate or bound the generalization error of the learned function h_S, $\mathrm{er}_D[h_S]$? Since h_S depends on the random sample S, $\mathrm{er}_D[h_S]$ is a random variable; we would like to bound $\mathrm{er}_D[h_S]$ in expectation or with high probability over the random draw of S.
- **Model selection.** Can we select a good function class $\mathcal{H}$, such that the generalization error $\mathrm{er}_D[h_S]$ of a function h_S learned by the algorithm from $\mathcal{H}$ is likely to be small?

In the first part of the course, we will focus mostly on generalization error bounds and say a little about model selection. Next, as the sample size approaches infinity:

- **Estimation error.** Does the generalization error of the learned function h_S, $\mathrm{er}_D[h_S]$, approach the optimal error in $\mathcal{H}$, $\mathrm{er}_D[\mathcal{H}] = \inf_{h \in \mathcal{H}} \mathrm{er}_D[h]$? In other words, does the estimation error $(\mathrm{er}_D[h_S] - \mathrm{er}_D[\mathcal{H}])$ approach zero (again, in expectation or with high probability over the random draw of S)? If so, at what rate?
- **Approximation error.** Does the optimal error in $\mathcal{H}$, $\mathrm{er}_D[\mathcal{H}] = \inf_{h \in \mathcal{H}} \mathrm{er}_D[h]$, approximate well the optimal error over all possible functions, $\mathrm{er}^*_D = \inf_{h : \mathcal{X} \to \mathcal{Y}} \mathrm{er}_D[h]$? In other words, is the approximation error $(\mathrm{er}_D[\mathcal{H}] - \mathrm{er}^*_D)$ small? (Note that this is a property of only $\mathcal{H}$ and D (and the loss function ℓ), and does not involve the learning algorithm.)

In the second part of the course, we will discuss estimation error in certain settings (specifically, estimation error is related to the notions of PAC learning and statistical consistency, certain aspects of which will be discussed in the course). We will have less to say about the approximation error. Finally:

- **Computational complexity.** Can we *efficiently* find a function in $\mathcal{H}$ with low generalization error?

Computational complexity is clearly an important aspect in the design of any practical learning system. We will make some observations about the computational complexity of some of the algorithms we will consider in this course (and it will play an important role in some of the definitions related to PAC learning, as well as in providing motivation for some of the classification methods based on optimizing convex losses); however, it will not be a major focus of the course.

Before closing this section, we note that in binary classification, the optimal error over all possible classifiers $h : \mathcal{X} \to \{-1, 1\}$ for a given distribution D is called the *Bayes error* (or *Bayes risk*) associated with D:

$$\mathrm{er}^*_D = \inf_{h : \mathcal{X} \to \{-1,1\}} \mathrm{er}^{0\text{-}1}_D[h] \,. \tag{5}$$

It is easily verified that, if $\eta(x)$ is defined as the conditional probability (under D) of a positive label given x, $\eta(x) = \mathbf{P}(y = 1|x)$, then the classifier $h^* : \mathcal{X} \to \{-1, 1\}$ given by

$$h^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq 1/2 \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

achieves the Bayes error. (In fact, any classifier $h : \mathcal{X} \to \{-1, 1\}$ that agrees with h^* on all x such that $\eta(x) \neq 1/2$ also achieves the Bayes error.) Such a classifier is termed a *Bayes classifier*. In general, η is unknown so the above classifier cannot be constructed directly. However later in the course we will see that it is possible to design classification methods that achieve the Bayes error in the limit.

3 Game-Theoretic/Online Setting

Assume again an instance space $\mathcal{X}$, an outcome space $\mathcal{Y}$, and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$. Consider now a situation where learning proceeds in trials: on each trial t, the learning algorithm receives an instance $x^t \in \mathcal{X}$, and must make a prediction $\hat{y}^t \in \mathcal{Y}$; the true label $y^t \in \mathcal{Y}$ is then revealed, and the algorithm incurs a loss $\ell(y^t, \hat{y}^t)$. The goal of the algorithm is to minimize its losses over any such sequence of trials.

In this case, we can view learning as a game between the learner and the environment: on each round of the game, the environment generates an instance, the learner makes its prediction, and the environment then responds with a true label. The learner's goal is to minimize its losses in the game. However, the environment, which can be viewed as an adversary, can clearly generate arbitrarily difficult sequences of instances and labels that force the learner to incur a large loss (in particular, note that the adversary can select the true label after observing the learner's prediction). In order to account for this, one usually assumes some 'comparator' class of predictors against which the learner will be compared, and requires only that the learner perform well with respect to the best predictor in this class. Formally, let $\mathcal{H}$ be some class of predictors (for example, each $h \in \mathcal{H}$ could be a function $h : \mathcal{X} \to \mathcal{Y}$ that predicts the label of x^t according to $\hat{y}_h^t = h(x^t)$). The *cumulative loss* of a learning algorithm A over a sequence of trials $S = ((x^1, y^1), \ldots, (x^T, y^T))$ is given by

$$L_S^\ell[A] = \sum_{t=1}^{T} \ell(y^t, \hat{y}^t)\,, \tag{7}$$

where $\hat{y}^t$ is the prediction made by A on trial t; similarly, the cumulative loss of a predictor $h \in \mathcal{H}$ over S is given by

$$L_S^\ell[h] = \sum_{t=1}^{T} \ell(y^t, \hat{y}_h^t)\,, \tag{8}$$

where $\hat{y}_h^t$ is the prediction made by h on trial t. The performance of A on S can then be measured in terms of its *relative loss* or *regret* with respect to $\mathcal{H}$, defined simply as

$$R_{\mathcal{H},S}^\ell[A] = L_S^\ell[A] - \inf_{h \in \mathcal{H}} L_S^\ell[h]\,. \tag{9}$$

The goal of a learning algorithm then is to achieve low regret over all possible sequences S. A natural question that arises is that of **regret bounds**, i.e. bounding the regret of an algorithm over the worst-case sequence S (for a given length T). It is also of interest to study the minimax value of the above game between the learner and environment, or the **minimax regret**, which is the best possible regret achievable by *any* algorithm over all possible sequences (for a given length T). In the third part of the course, we will analyze these quantities for a variety of online learning problems, including online classification and regression and learning from expert advice.[3] We will also consider online-to-batch conversions that allow online learning algorithms to be translated to algorithms in the probabilistic setting, and the corresponding regret bounds to be translated to bounds on the generalization error.

4 Next Lecture

In the next lecture we will review support vector machines, one of the most widely used learning algorithms in the probabilistic/batch setting, and will motivate some of the questions that will then be addressed in the next several lectures.

Acknowledgments. Some ideas for presentation of the material in this lecture were taken from the first lecture of a course offered by Peter Bartlett at UC Berkeley in Spring 2008.

[3] We note that the problem of learning from expert advice, where each expert (such as an investment manager) makes a prediction and incurs a loss, and the goal is to dynamically allocate weights among these experts so as to minimize the overall loss, differs somewhat from the formulation above; however the basic ideas can be used to analyze such problems as well.