8803 Machine Learning Theory

Maria-Florina Balcan Lecture 1: August 23rd, 2011

Machine learning studies automatic methods for learning to make accurate predictions or useful decisions based on past observations and experience, and it has become a highly successful discipline with applications in many areas such as natural language processing, speech recognition, computer vision, or gene discovery.

This course on the design and theoretical analysis of machine learning methods will cover a broad range of important problems studied in theoretical machine learning. It will provide a basic arsenal of powerful mathematical tools for their analysis, focusing on both statistical and computational aspects. We will examine questions such as "What guarantees can we prove on the performance of learning algorithms?" and "What can we say about the inherent ease or difficulty of learning problems?". In addressing these and related questions we will make connections to statistics, algorithms, complexity theory, information theory, game theory, and empirical machine learning research.

Note: The course web page is `http://www.cc.gatech.edu/~ninamf/ML11/`.

Two useful books for this course are:

1. *An Introduction to Computational Learning Theory* by M. Kearns and U. Vazirani
2. *A Probabilistic Theory of Pattern Recognition* by L. Devroye, L. Gyorfi, G. Lugosi.

It's not 100% crucial to have them, but they are very useful for better understanding the material discussed in the class.

1 What is machine learning theory about?

The goal of creating programs or machines that learn with experience is one of the oldest in computer science. It's a key component of "strong AI". But also as a practical matter we want programs that can adapt to change, that can learn to do things that are hard to program explicitly, that can adapt to the needs of their users, and that can find useful information in large volumes of data.

The goals of machine learning theory include:

- To create mathematical models that capture key aspects of machine learning.

- To prove guarantees for algorithms (when will they succeed, how long will they take?), and to develop algorithms that provably meet desired criteria; to provide guidance about which algorithms to use when.
- To analyze the inherent ease or difficulty of learning problems.
- To mathematically analyze general issues, such as: "why is Occam's Razor a good idea?"

Models of Learning A mathematical model of machine learning needs to specify a number of things: the kind of task we are considering (learning a concept from data? learning to play a game?), the kind of data we have (are we passively shown examples? do we actively get to ask questions or play the game?), the kind of feedback we get (right away? only after the game is over? depends on the action we take?), and what is our criteria for success.

What is it that makes a model a good one? Sometimes a model is good because it accurately reflects common learning settings. The real test of a model, though, is whether one can gain important insights through it. We will see, for instance, that the main models used in computational learning theory are robust to variations in their definitions, and allow us to focus on fundamental issues.

Complexity Theory and Statistics/Information-Theory There are two main parts to machine learning theory. One is *complexity*-theoretic: how *computationally hard* is the learning problem? The other is statistical or *information*-theoretic: how *much data* do I need to see to be confident that good performance on that data really means something? These are both important questions, and we'll see a lot of interesting theory in both directions. It's important to keep in mind which of these is your focus at any given time (e.g., asking yourself "would this be trivial if I had unlimited computational power?" or "would this be intractable if I didn't?").

2 Passive Supervised Learning

For most of this course we will focus on the problem of *concept learning* in the passive supervised learning setting. We are given data (say documents classified into topics or email messages labeled as spam or not) and we want to be able to learn from this data to classify future examples well. This might seem like a very restricted form of learning, but it turns out that most methods for other kinds of learning end up solving this sort of problem at their core.

Formally, in the passive supervised learning setting, we assume assume that the input to a learning algorithm is a set S of labeled examples, let's say a set of emails labeled as spam or not.

S : $(x_1, y_1), \ldots, (x_m, y_m)$

Here $x_i \in X$ is the feature part of our example and and y_i is the label part; we assume for now that $y_i \in \{0, 1\}$, so we do binary classification (decide between spam and not spam).

- Examples are typically described by their values on some set of *features* or *variables* (we will use these words interchangeably) which we call the feature part. For instance, if we are trying to predict spam, the first feature might be whether or not the email contains the word "money", the second feature might be whether or not the email contains the word "pills", the third feature might be whether or not the email contains the word "Mr.", the fourth feature might be whether or not the email contains bad spelling, and the fitfh feature might be whether the email has a known sender or not.

  If there are n boolean features, then we can think of examples as elements of $\{0, 1\}^n$. In the spam example (in the handout) $X = \{0, 1\}^5$, and $x_1 = 10110$. If there are n real-valued features, then examples are points in R^n. The space that examples live in is called the *instance space* X.

- A *labeled example* (x_i, y_i) is an example x_i together with a labeling y_i (e.g., positive or negative).

- A *concept* is a boolean function over an instance space. For instance, the concept $x_1 \wedge x_2$ over $\{0, 1\}^n$ is the boolean function that outputs 1 on any example whose first two features are set to 1. We will sometimes use the terms classifier, or hypothesis, or prediction rule to mean a concept.

- A *concept class* is a set of concepts, typically with an associated representation. The *size* of a concept is the number of bits needed to specify the concept in its representation.

  For instance, the class of "monotone conjunctions" over $\{0, 1\}^n$ consists of all concepts that can be expressed as a conjunction of variables. A common representation for a monotone conjunction is to store n bits, one for each variable, specifying whether the variable is in the conjunction or not; a much less efficient representation is one where we store 2^n bits specifying the value of the conjunction on each 2^n examples in the instance space. Yet another representation is one where we store the indices of all the variables appearing in the conjunction; so, if the conjunction has k relevant variables, this requires $k \log n$ bits.

  Depending on the representation, some concept classes contain both simpler and more complicated concepts. For example in the last representation, we have both simple and complicated monotone conjunctions. In the standard enconding[1], decision trees can be small or large.

[1] We can use $O(\log n)$ bits to give the index of the variable at the root or one of the constants "+" or "-". Then, if the root is not one of the constants, recursively describe the left subtree and the right subtree. In this way, the total number of bits stored is $O(k \log n)$ where k is the number of nodes in the tree.

2.1 The Consistency Model

The consistency model is not a particularly great model of learning, but it's simple and is a good place to start.

Definition 1 *We say that algorithm $\mathcal{A}$ learns class C in the consistency model if given any set of labeled examples S, the algorithm produces a concept $c \in C$ consistent with S if one exists, and outputs "there is no consistent concept" otherwise.*

We'd also like our algorithm to run in polynomial time (in the size of S and the size n of the examples). So, this should seem like very natural definition if you're an algorithms/complexity/optimization person.

Let's now consider the learnability of several simple classes in the consistency model. Then we'll critique the model at the end.

AND functions (monotone conjunctions). This is the class of functions like $x_1 \wedge x_4 \wedge x_7$, which is positive whenever the 1st, 4th, and 7th features are on. For example, the following set of data has a consistent monotone conjunction:

```
1 0 1 1 0 0 1 1   +
1 1 1 1 1 0 1 0   +
0 1 1 1 0 0 1 1   +
0 0 0 1 1 1 1 1   -
1 1 1 1 1 0 0 0   -
```

We can learn this class in the consistency model by the following method:

1. Throw out any feature that is set to 0 in any positive example. Notice that these cannot possibly be in the target function. Take the AND of all that are left.
2. If the resulting conjunction is also consistent with the negative examples, produce it as output. Otherwise halt with failure.

Since we only threw out features when absolutely necessary, if the conjunction after step 1 is not consistent with the negatives, then no conjunction will be.

OR functions (monotone disjunctions) This is the class of functions like $x_2 \vee x_5 \vee x_7$, which is positive whenever either the 2nd, or the 5th, or the 7th feature is on. Observe that any monotone disjunction can be expressed as a monotone conjunction if we negate each variable and apply De Morgan's law. That is, $x_2 \vee x_5 \vee x_7 = \overline{\overline{x_2} \wedge \overline{x_5} \wedge \overline{x_7}}$. If we replace each x_i with $x'_i = \bar{x}_i$ and flip all positive labels in the data set to negative and vice versa, we can use our learning algorithm for monotone conjunctions to learn a conjunction c on the modified instances. To obtain a concept from the original class, simply negate each variable in c and replace all the $\wedge$ with $\vee$.

Non-monotone conjunctions, disjunctions, k-CNF, k-DNF. What about functions like $x_1\bar{x}_4x_7$? Instead of thinking about this from scratch, we can just perform a reduction to the monotone case. If we define $y_i = \bar{x}_i$ then we can think of the target function as a monotone conjunction over this space of $2n$ variables and use our previous algorithm. k-CNF is the class of Conjunctive Normal Form formulas in which each clause has size at most k. E.g., $x_4 \wedge (x_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3)$ is a 2-CNF. So, the 3-CNF learning problem is like the inverse of the 3-SAT problem: instead of being given a formula and being asked to come up with a satisfying assignment, we are given assignments (some satisfying and some not) and are asked to come up with a formula. k-DNF is the class of Disjunctive Normal Form formulas in which each term has size at most k. We can learn these too by reduction: e.g., we can think of k-CNFs as conjunctions over a space of $O(n^k)$ variables, one for each possible clause.

Next time we will discuss other more interesting concept classes such as Decision lists, linear separators, etc.

Unfortunately, the consistency mode does not address the generalization issue at all. We discuss next the PAC model that can deal with this aspect.

2.2 The PAC Model

The basic idea of the PAC model is to assume that examples are being provided from a fixed (but perhaps unknown) distribution over the instance space. The assumption of a fixed distribution gives us hope that what we learn based on some training data will carry over to new test data we haven't seen yet. A nice feature of this assumption is that it provides us a well-defined notion of the error of a hypothesis with respect to target concept.

Definition 2 *Given a example distribution $\mathcal{D}$, the* error *of a hypothesis h with respect to a target concept c is* $\mathbf{Prob}_{x\in\mathcal{D}}[h(x) \neq c(x)]$. *(*$\mathbf{Prob}_{x\in\mathcal{D}}(A)$ *means the probability of event A given that x is selected according to distribution $\mathcal{D}$.)*

In the PAC model we assume that the input to the learning algorithm is a set of labeled examples

$S:$ $\quad (x_1, y_1), \ldots, (x_m, y_m)$

where x_i are drawn i.i.d. from some fixed but unknown distrution D over the the instance space X and that they are labeled by some target concept c^* in some concept class C. So $y_i = c^*(x_i)$. Here the goal is to do optimization over the given sample S in order to find a hypothesis $h : X \to \{0, 1\}$, that has small error over whole distribution D.

What kind of guarantee could we hope to make?

- We converge quickly to the target concept (or equivalent). But, what if our distribution places low weight on some part of X?

- We converge quickly to an approximation of the target concept. But, what if the examples we see don't correctly reflect the distribution?
- With high probability we converge to an approximation of the target concept. This is the idea of **P**robably **A**pproximately **C**orrect learning.

2.2.1 Conjunctions

Here a nice guarantee for the case of learning conjunction in this model.

Theorem 1 *Let C be the class of conjunctions over $\{0,1\}^n$. Let D be an arbitrary, fixed unknown probability distribution over X and let c^* be an arbitrary unknown target function. For any ϵ, $\delta > 0$, if we draw a sample from D of size*

$$m = \frac{1}{\epsilon}\left[n\ln(3) + \ln\left(\frac{1}{\delta}\right)\right],$$

then with probability at least $1-\delta$, all concepts in C with error $\geq \epsilon$ are inconsistent with the data (or alternatively, with probability at least $1-\delta$ any conjunction consistent with the data will have error at most ϵ.)

Note: Since we have an algorithm that finds a consistent conjunction whenever one exists, this means that if the target function is a conjunction, then we can use this algorithm to produce a hypothesis with error at most ϵ with probability at least $1-\delta$, in time and sample size polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and n (we simply run it on a large enough sample).

Proof: The proof involves the following steps:

1. Consider some specific "bad" conjunction whose error is at least ϵ. The probability that this bad conjunction is consistent with m examples drawn from $\mathcal{D}$ is at most $(1-\epsilon)^m$.
2. Notice that there are (only) 3^n conjunctions over n variables.
3. (1) and (2) imply that given m examples drawn from $\mathcal{D}$, the probability *there exists* a bad conjunction consistent with all of them is at most $3^n(1-\epsilon)^m$. Suppose that m is sufficiently large so that this quantity is at most δ. That means that with probability $(1-\delta)$ there are *no* consistent conjunctions whose error is more than ϵ.
4. The final step is to calculate the value m needed to satisfy

$$3^n(1-\epsilon)^m \leq \delta. \tag{1}$$

   Using the inequality $1-x \leq e^{-x}$, it is simple to verify that (1) is true as long as:

$$m = \frac{1}{\epsilon}\left[n\ln(3) + \ln\left(\frac{1}{\delta}\right)\right].$$

■

Note: Another way to write the bound in Theorem 1 is as follows:

For any ϵ, $\delta > 0$, if we draw a sample from D of size m then with probability at least $1 - \delta$, any conjunction consistent with the data will have error at most

$$\frac{1}{m}\left[n\ln(3) + \ln\left(\frac{1}{\delta}\right)\right].$$

This is the more "statistical learning theory style" way of writing the same bound.