15-854 Machine Learning Theory

Lecturer: Avrim Blum — Lecture 1: January 12, 1998

1 What is machine learning theory about?

The goal of machines that learn with experience is one of the most intriguing in computer science. As the types of problems that we would like machines to solve get more complex, it is becoming increasingly necessary task as well.

Unfortunately, by its nature, machine learning is somewhat fuzzy. After all, the goal is to learn something we do not know. Nonetheless, we would like provide a mathematical framework and foundation for machine learning to aid in our understanding and improve our ability to make progress. Towards this end, a computational theory of machine learning has developed in recent years. The goals of this 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?"

1.1 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 machine learning theory are robust to variations in their definitions, and allow us to focus on fundamental issues.

For most of this course we will focus on the problem of *concept learning*. You are given data (say pictures of cars and pictures of trucks) and you want to be able to learn from this data to classify future examples (new unlabled pictures) well. This seems like a very restricted form of learning, but it turns out that most methods for other kinds of learning tasks are solving this sort of problem at their core. For example, in Pomerleau's ALVINN system for learning how to drive a car [1], the basic learning algorithm used is given picture images as input and steering directions as the label.

What is the best we could hope to guarantee for a learning algorithm? If we are optimistic, we could hope to make a guarantee of the form: "If there exists a 'reasonable' rule that (approximately) explains the data, then our algorithm will find one quickly," where 'reasonable' means a rule we can evaluate efficiently (rules that can't be evaluated aren't particularly useful for prediction). Unfortunately, cryptography tells us that this is unlikely: the existence of hard cryptographic functions means that there are simple functions that produce data that look completely random. However, we *will* be able to make guarantees of this form for more restricted notions of 'reasonable'.

2 Some definitions to get us started

- The *instance space* X is the set from which examples are drawn. The most common instance space we will consider is the set of n-bit vectors (the set $\{0,1\}^n$). The second most common case is the set R^n. Typically we will think of examples of this form as assigning values to *features* or *variables*. For instance, if we are trying to predict if it will rain, the first feature might be whether or not it was cloudy in the morning, the second feature might be whether the weather channel said it would rain, the third feature might be whether Channel 11 said it would rain, etc.

- A *labeled example* is an example together with a labeling (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 input in which the first two bits are set to 1. Sometimes we will think of a concept c as the set of its positive examples; i.e., $x \in c$ means that $c(x) = 1$.

- A *concept class* is a set of concepts, typically with an associated representation. For instance, the class of "monotone conjunctions" consists of all concepts that can be expressed as a conjunction of variables. Formally, one can talk of a concept class as a set of strings, each string representing some concept, along with an "evaluation function" that given a string and an example produces the label of that example under the associated concept. We will always assume that our representations are "reasonable" in the sense that this evaluation function runs in polynomial time. I.e., given a concept and an example, it will always be the case that one can quickly decide whether the example is positive or negative.

  Some concept classes contain both simpler and more complicated concepts. For instance, decision trees can be small or large. In these cases, we will need to talk about the *size* of a concept which will be some approximation of the length of its representation (e.g., the number of nodes in the decision tree).

- If one wants to be completely formal, one should talk about an instance space X as a union of spaces X_n, each defined for a different value of n, and the same for concept classes. We'll pretty much ignore this issue.

3 The Consistency Model

The consistency model is not a particularly good model of learning, but it's simple so let's begin with it.

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. The algorithm should run in polynomial time (time polynomial in the description length of S).*

Let's now consider the learnability of several simple classes in the consistency model.

Conjunctions. The class of conjunctions is the class of ANDs of literals (a literal is a variable or its negation). E.g., a typical concept might be $x_1\bar{x}_4x_7$. This class is learnable by the following method:

1. Produce the conjunction of all literals that are satisfied by every positive example. (I.e., throw out any literal falsified by some positive example.) By definition, this is consistent with all the positive examples. In fact, this is the most *specific* conjunction consistent with the positive examples because we only throw out literals when absolutely necessary; therefore, if any conjunction is consistent with *all* the examples, this one is.
2. If the above conjunction is also consistent with the negative examples, produce it as output. Otherwise halt with failure.

k-CNF, k-DNF. k-CNF is the class of Conjunctive Normal Form formulas in which each clause has size at most k. E.g., $x_4(x_1 \vee x_2)(x_2 \vee \bar{x}_3)$ 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. These classes are easy to learn too, using the same basic method as above. Alternatively, one can make a *reduction* to the problem of learning conjunctions by introducing a new variable for each possible clause in a k-CNF (or term in a k-DNF), and then negating the labels in the case of DNF.

k-term DNF, k-term CNF. These are CNF or DNF formulas restricted to have at most k terms. This learning problem is NP-hard in the consistency model, for any $k \geq 2$.

Truth tables. (By this, let's mean a partial truth table where any example not on the list is assumed to be negative). These are easy.

General DNF formulas. *Warning: this is misleading!!!* These are easy too since you can represent any truth table with a DNF formula.

To think about:

Truth tables are in some sense "more general than" conjunctions. Does this make the truth-table learner better than the conjunction learner? What about the DNF learner?

Is it strange that k-term DNF are harder to learn than k-CNF even though the latter can express any function that the former can?

These questions illustrate some problems with the consistency model. The main problem is that it is not clear why (or whether) finding a consistent hypothesis has anything to do with the goal of predicting well on new examples. We will remedy these problems in the improved "PAC" and "mistake bound" models.

4 "Agnostic" learning

Before we finish with the consistency model, one interesting modification to consider is: what if, in the case that there is no consistent concept in C, we require that the algorithm find the "best" hypothesis? We could call this the *max consistency model*: the algorithm's goal is to find the concept $c \in C$ that maximizes the number of examples in S consistent with c. Unfortunately, solving this problem is NP-hard even for the class of conjunctions. (We'll revisit this problem later when we get into models of noise in data.) Here is a (hard) open problem:

Approximate agnostic learning of conjunctions: Is there a polynomial time algorithm that given a set S of labeled examples over $\{0,1\}^n$ and a guarantee that there *exists* some conjunction consistent with at least 99% of them, will produce a conjunction consistent with at least 51%?

References

[1] D. Pomerleau. *Neural network perception for mobile robot guidance.* Kluwer Academic Publishers, 1993.