15-859(B) Machine Learning Theory

Avrim Blum Lecture 1: January 15, 2002

---

1 Administrivia

The course web page is `http://www.cs.cmu.edu/~avrim/ML02/`. There will be 6-7 homework assignments, a take-home final, and a presentation or project. You will also be asked to help grade one of the homeworks. The plan is for lecture notes to be posted the night before class, in case you want to print them out and bring them with you. The book for this course is Kearns and Vazirani *An Introduction to Computational Learning Theory*. It's not 100% crucial but good to have. It covers about 1/2 of the course material. The first few lectures (on learning in the "mistake-bound" model) will be on material *not* in the book, so you have a couple weeks to get it. See the course information guide on the web page for more details.

2 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 gobs of data.

Unfortunately, by its nature, machine learning is a little fuzzy. After all, the goal is to learn something we don't 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. The goals of computational 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?"

2.1 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 computational 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*. We are given data (say documents classified into topics) and we want to be able to learn from this data to classify future examples well. This seems 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. For example, in Pomerleau's ALVINN system for learning how to drive a car, 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 extremely optimistic, we could hope to make a guarantee of the form: "If there exists a polynomial-time program that (approximately) explains the data, then our algorithm will do so too." Unfortunately, cryptography tells us that this is too much to expect: the existence of hard cryptographic functions means that it's possible for simple functions to produce data that looks completely random. However, we *will* be able to make guarantees of this form for more reasonable goals.

3 Some definitions to get us started

- Examples are typically described by their values on some set of *features* or *variables* (we will use these words interchangeably). 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. If there are n boolean features, then we can think of examples as elements of $\{0,1\}^n$. 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* 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 example whose first two features are set to 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.

  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 number of bits in its representation (e.g., the number of nodes in the decision tree).

4 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 (time polynomial in the description length of S). 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.

AND functions (monotone conjunctions). This is the class of functions like $x_1x_4x_7$, 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) We can do the same as above except swapping positive/negative and 1/0. We throw out all variables set to 1 by some negative example and check to see if the OR of the remainder is consistent with the positive examples.

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., $x_4(x_1 \vee x_2)(x_2 \vee \bar{x}_3)$ 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. 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.

Decision lists. A *Decision List* is a list of if-then rules where each condition is a literal (a variable or its negation). For example, say I like to go for a walk if it's warm or if it's snowing and I have a jacket, so long as it's not raining. We could describe this as a decision list like this:

```
if rainy then no
else if warm then yes
else if not(have-jacket) then no
else if snowy then yes
else no.
```

If you like, you can think of this as a decision tree with just one long path. Let's find an algorithm to learn DLs in the consistency model. Extending this to the Mistake-bound model (which we'll talk about next time) is on the homework. Here's an algorithm:

1. Find some rule consistent with the current set of examples that applies to at least one of them. If no such rule exists, halt with failure.
2. Put the rule at the bottom of the hypothesis.
3. Throw out those examples classified by the hypothesis so far.
4. If there are any examples left, repeat from the beginning.

E.g.,

1	0	0	1	1	+
0	1	1	0	0	−
1	1	1	0	0	+
0	0	0	1	0	−
1	1	0	1	1	+
1	0	0	0	1	−

It is clear that this algorithm runs in polynomial time, since each iteration removes at least one example, and each iteration can be performed in polynomial time. What we now need to show is that if there exists a consistent list, this algorithm will find one.

Proof: If there is a decision list L consistent with remaining data, then there is at least one choice for step 1 (the highest rule in L satisfied by at least one example). Furthermore, if there was a consistent DL at the beginning, then there must exist a DL consistent with the remaining data — namely, the original one. ta da. (qed)

Linear separators. Here we can think of examples as being from $\{0,1\}^n$ or from R^n. The goal is to find a hyperplane $w \cdot x = w_0$ such that all positive examples are on one side and all negative examples are on other. I.e., $w \cdot x > w_0$ for positive x's and $w \cdot x < w_0$ for negative x's. We can solve this using linear programming. In practice, people tend to use incremental algorithms like perceptron or winnow or maximum entropy for this problem, because of tolerance to noise (and good speed in practice).

2-term DNF, 2-term CNF. A 2-term DNF is an OR of two conjunctions. A 2-term CNF is an AND of two disjunctions. An intersection of 2 halfspaces is a formula like "x is positive if $w \cdot x > w_0$ and $v \cdot x > v_0$". These learning problems are all NP-hard in the consistency model.

General DNF formulas. *Warning: this is misleading!!!* These are easy since you can just create one term for every positive example.

5 Problems with the consistency model

Let's think about some problems with the consistency model. One big problem just came up in the last example: the DNF learner doesn't really seem like it's learning. You wouldn't especially have any confidence in its ability to predict on new data. In a sense, the problem is that the size of its hypothesis grows linearly with the amount of training data. In fact, we'll see later a converse to this: being able to get a hypothesis of size sublinear in the amount of data *does* imply being able to predict on new examples, in the PAC model (this is a model in which examples come from a distribution, but we'll get back to that later).

A second problem is that it seems strange that k-term DNF aren't learnable but k-CNF are, even though the latter can express any function that the former can. In a sense, this is saying more about the hypothesis representation of the algorithm than the fundamental hardness of predicting. Sometimes this is what you want: for example, if you are designing a neural net algorithm (so you are committing to a specific hypothesis representation) it would be nice to know how hard the consistency problem is so you know what is reasonable to expect. (As it turns out, even for really simple neural nets, the consistency problem is NP-complete.)

Yet a third problem with the consistency model is often there isn't any consistent rule in the class, but you still would like to find a pretty good one if possible. The problem of simply finding the *best* rule in some class (i.e., that has the fewest disagreements with the data) is sometimes called *agnostic learning*, Unfortunately, this is NP-hard even for the class of conjunctions. So it may be hard to say much algorithmically. But we'll get back to this later when we get into models of noise in data.

6 Open problems

Let me end with two (very hard) open problems:

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

- **Learning an intersection of halfspaces via some other representation:** We saw that we can learn 2-term CNF via the class of 2-DNF. Is there an analogous way to learn the class of "ANDs of two halfspaces" via some other class (where the hypothesis size does not grow linearly with the number of examples)?