Machine Learning Coms-4771

Machine Learning Theory
The Mistake-Bound model

Lecture 6

Based entirely on Avrim Blum's notes (see the link at the web page)

Goals of ML theory:

- develop and analyze models of learning that capture the key aspects of machine learning
- help understand what type of guarantees we can hope to achieve, and what type of learning problems we can hope to solve

Two main learning models:

1. Distributional (PAC) setting (a "batch" model)
   - Assumption: training and test examples come (independently) from some fixed probability distribution D over the instance space, labeled by an unknown target function (from a known hypothesis class).
   - Basic question: How much data do I need to see so that if I do well over it, I can expect to do well on new points drawn from D?
2. On-line setting: No distributional assumptions. An adversary selects the order in which examples are presented. No separate training set. The learner attempts to predict on every example seen. Count the number of mistakes.

We are going to start with (2).

Mistake Bound Model

Learning is in stages:

- The learner gets an unlabeled example $x \in X$
- The learner predicts its classification
- The learner is told the correct label $f(x)$

Goal: minimize the number of mistakes

Definition

Algorithm A learns class of functions C with mistake bound M if A makes at most M mistakes on any sequence of examples consistent with some $f \in C$.

Can't talk about past performance predicting future performance, or the number of samples needed to learn (e.g., we may end up seeing the same example over and over again, so we don't learn anything about f, but it's Ok, because we won't be making many mistakes either).

Example: Disjunctions

$X = \{0, 1\}^n$. Assumption: examples are consistent with a disjunction (an OR on a subset of features)

Example: Disjunctions

$X = \{0,1\}^n$. Assumption: examples are consistent with a disjunction (an OR on a subset of features)

- Start with $h(x) = x_1 \vee x_2 \vee \cdots \vee x_n$
- Get x, answer according to h.
- Can't make a mistake on positive. Mistake on negative: throw out all variables in h set to 1 in x.
- Each mistake removes at least one variable, so the total number of mistakes is at most n.

Example: Disjunctions

No deterministic algorithm can do better:

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Any labeling of these examples is consistent with some disjunction. So regardless of what the algorithm predicts, there is always a consistent disjunction that disagrees with the algorithm on every single example.

Halving Algorithm

- Learn *arbitrary* concept class C with at most $\log(|C|)$ mistakes. (The number of monotone disjunctions on n variables is 2^n, so this makes at most n mistakes, one mistake per bit)
- Take majority vote over all $h \in C$ consistent with all examples seen so far.
- Proof: Each mistake cuts down the number of available concepts at least in half.

What if our concept class has functions of different sizes (e.g., decision trees)?

- C_b = the set of functions in C that take at most b bits to represent, $|C_b| < 2^b$.
- "Guess and double" on b. If the target function takes s bits and $2^b \leq s \leq 2^{b+1}$, the number of mistakes is at most $1 + 2 + 4 + ... + 2s < 4s$.
- Can get rid of the factor of 4:
  - Give each h a weight of $(1/2)^{\text{size}(h)}$. If our description language is a prefix-code (no concept is a prefix of another), the total sum of weights is at most 1.
  - Take weighted vote. Each mistake removes at least 1/2 of total weight left. So if the target has size s, then the number of mistakes is at most $\log(2^s) = s$. So we are paying one mistake per bit of the description of the target function (if you don't care about computation time).

Standard Optimal Algorithm

Online Learning as a 2-player game between Algorithm and Adversary

- Adversary selects x which splits C into two sets: $C_0(x)$—functions that label x negative, and $C_1(x)$—functions that labels x positive.
- The algorithm gets to pick one of $C_0(x)$ and $C_1(x)$ to throw away (by predicting postive and making a mistake it throws out $C_1(x)$, or by predicting negative and making a mistake it throws out $C_0(x)$)
- Repeat until only one function is left.

The value of this game (to the opponent) is the number of rounds the game is played.

- Well-defined number $\text{opt}(C)$ that is the optimal mistake bound for concept class C (minimum over all algorithms).
- Well defined optimal strategy for each player: Given an example x, we "just" calculate $\text{opt}(C_0(x))$ and $\text{opt}(C_1(x))$ (by applying this idea recursively), and throw out whichever set has larger mistake bound.

Is Halving Algorithm optimal?

- Halving algorithm: throw out larger set
- Optimal algorithm: throw out set with larger mistake bound
- Not always the same (so halving is not always optimal)! It's possible for the mistake bound for a set of functions C to be much smaller than $\log |C|$.
- You will come up with an example in your next homework.

What if there is no perfect function?

- Think of functions in C as experts giving advice. You want to do as well as the best expert in hindsight (regret bounds).
- Next lecture: Flexible version of the halving algoritm (instead of throwing out inconsistent functions, reduce their weight) (Weighted Majority)
- Next lecture: efficient algorithms that can handle large number of irrelevant features (Winnow)