CS260: Machine Learning Theory
Lecture 7: Online Classification and Mistake Bounds

October 17, 2011

Lecturer: Jennifer Wortman Vaughan

1 Online Classification

So far we have been considering the *batch* learning setting in which the learning algorithm is presented with a sample of training data and must produce a hypothesis that performs well on new data generated from the same distribution. For the next few weeks, we will shift our attention to the *online* learning setting, in which the learning algorithm is presented with a sequence of examples over time, and must repeatedly update its hypothesis based on these examples. The online learning setting can be used to model applications like spam filtering, in which the algorithm must adapt to feedback.

A Basic Online Classification Model

In the basic online setting, at each round $t \in \{1, 2, 3, \cdots\}$,

1. The learner is presented with a new example $\vec{x}_t$.
2. The learner must predict a label $\widehat{y}_t$ for this example.
3. After the prediction is made, the true label y_t of the example is revealed.
4. The learner updates its prediction rule based on $\vec{x}_t$, $\widehat{y}_t$, and y_t.

Unlike the PAC learning model, *no distributional assumptions* are made about the sequence of examples $\vec{x}_1, \vec{x}_2, \cdots$. The online learning setting is therefore "adversarial" in the sense that we can imagine the examples are generated by an adversary who would like to force our algorithm to make as many mistakes as possible. Because of this, there are a lot of connections between online learning and game theory, some of which we will discuss in upcoming classes.

The learning algorithm is said to make a mistake on any round t at which $y_t \neq \widehat{y}_t$. There are several reasonable goals that could be considered in this setting. We will discuss two:

1. Minimize the number of mistakes made by the algorithm. In this case, we would like to find a bound on the total number of mistakes made by the algorithm such that the ratio

$$\frac{\text{\# of mistakes}}{\text{\# of rounds}}$$

tends to zero as number of rounds gets large.

All CS260 lecture notes build on the scribes' notes written by UCLA students in the Fall 2010 offering of this course. Although they have been carefully reviewed, it is entirely possible that some of them contain errors. If you spot an error, please email Jenn.

In order to achieve such a goal, it is necessary to make some assumptions about the way in which the labels y_t are generated. For example, we might assume that there exists a target function c in a class $\mathcal{C}$ such that for all t, $y_t = c(\vec{x}_t)$. This is the analog of the realizable batch learning setting we discussed.

2. Minimize regret. This is the analog of the unrealizable batch learning setting, as we no longer need to assume the existence of a perfect target function. Instead, we minimize the difference between the number of mistakes the algorithm makes and the number of mistakes made by the best predictor or comparator in a class of functions. In short, we would like the ratio

$$\frac{\text{\# of mistakes} - \text{\# of mistakes by comparator}}{\text{\# of rounds}}$$

to tend to zero as number of rounds grows.

The regret minimization scenario may be more realistic and useful in real-world applications in which we do not have the luxury of assuming a perfect target exists or that our data is free of noise. However, to get started, we will focus on the first goal for the next few lectures. To get some intuition about the online classification setting and the classes that can or cannot be learned, we will establish some general upper and lower bounds for the Mistake Bound Model (which we will not define formally, but will discuss only informally; check out Avrim Blum's online learning survey[1] for a more formal treatment). In the next lecture, we will start looking at more interesting algorithms that can be applied to learn specific concept classes like linear threshold functions.

2 A Simple Upper Bound

We begin by deriving an upper bound for the Mistake Bound Model that can be applied to any finite concept class. The learning algorithm we consider, the Halving Algorithm, makes use of the notion of a *version space*. The version space is defined to be the set of all functions from the class $\mathcal{C}$ that are consistent with all of the data that the algorithm has seen so far. Over time, as the algorithm sees more examples and more functions become inconsistent, the version space decreases in size. Notice that this concept of version space is only sensible to consider when we make the assumption that there is a perfect target function in $\mathcal{C}$; otherwise the version space could become empty.

The halving algorithm works as follows. At each round $t = 1, 2, 3, \cdots$,

1. When a new example $\vec{x}_t$ arrives, set $\widehat{y_t}$ to be the label chosen for $\vec{x}_t$ by the majority of functions in the current version space, VS_t.
2. When the true label y_t is revealed, update the version space to VS_{t+1}.

Note that this algorithm can, in general, be terribly inefficient. As a result, the upper bound given in this theorem says nothing about what can be learned efficiently.

Theorem 1. *Let $\mathcal{C}$ be any finite concept class. If there exists a function $c \in \mathcal{C}$ such that for all rounds t, $y_t = c(\vec{x}_t)$, then the number of mistakes made by the Halving Algorithm is no more than* $\log_2 |C|$.

[1] `http://www.cs.cmu.edu/~avrim/Papers/survey.pdf`

Proof: We will analyze the size of the version space over time as the number of mistakes grows. We start with the simple observation that the version space cannot possibly be bigger than the size of the entire concept class $\mathcal{C}$. That is, $|\mathrm{VS}_t| \leq |C|$ for all t.

If a first mistake is made on some round t_1, it implies that the majority of the functions in VS_{t_1} were incorrect about the label of x_{t_1}, and are now inconsistent with the data. Therefore, at least half of the functions in the VS_{t_1} will be eliminated, and we know that $|\mathrm{VS}_t| \leq |\mathcal{C}|/2$ for all $t > t_1$.

If a second mistake is made on some round t_2, then half of the remaining functions in the version space will be eliminated. We have that $|\mathrm{VS}_t| \leq |\mathcal{C}|/4$ for all $t > t_2$.

More generally, if the kth mistake is made on some round t_k, then we know that $|\mathrm{VS}_t| \leq |\mathcal{C}|/2^k$ for all $t > t_k$. Since we are in a setting in which there exists a perfect target function in $\mathcal{C}$, we know that for all t, $|\mathrm{VS}_t| \geq 1$.

Combining these expressions, we get that on any round t after k mistakes have been made,

$$1 \leq |\mathrm{VS}_t| \leq \frac{|\mathcal{C}|}{2^k}$$

which implies that $k \leq \log |\mathcal{C}|$. If the halving algorithm made more than $\log |\mathcal{C}|$ mistakes, this condition would be violated, so it must make fewer. □

This result has the familiar logarithmic dependence on the size of the concept class, reflecting the Occam's razor principle that the bigger the class we start with, the harder it is to learn. As an example consider the class of monotone conjunctions where $|\mathcal{C}| = 2^n$ for n features. The number of mistakes is bounded by n for this class. However as mentioned above, this algorithm is generally computationally infeasible (except in special cases) as it requires tracking a version space of 2^n functions and computing a majority label by evaluating up to 2^n functions at every step of the algorithm.

3 Lower Bound

It is possible to derive simple lower bounds in this setting too. In particular, if $\mathcal{C}$ has VC dimension d, then we can force any deterministic algorithm $\mathcal{A}$ to make at least d mistakes. The key is that we are in an adversarial setting. By the definition of VC dimension, we know that there exists some set of d points that is shattered by $\mathcal{C}$. (If there is more than one, we can pick one arbitrarily.) The adversary can choose these d points as the first d examples, $\vec{x}_1, ..., \vec{x}_d$. Since these exist functions in $\mathcal{C}$ achieving every labeling of these points, the adversary can choose a target function in such a way that $\mathcal{A}$ makes a mistake on each of these rounds.

It is straight-forward to adapt this lower bound for randomized algorithms too and show that the adversary can cause at least $d/2$ mistakes *on expectation* by choosing the labels of the d shattered points uniformly at random.

4 The second half of class...

We spent the second half of class finishing our discussion of Problem Set 1.