Machine Learning Theory (CSC 482A/581A) - Lecture 2

Nishant Mehta

1 Realizable case and the Mistake Bound Model

We begin our study of machine learning theory with classification under the strong but plausible assumption of *realizability*: there exists some boolean function $c \in \mathcal{C}$ which determines the labels, so that labeled examples always take the form $(x, c(x))$. Here, $\mathcal{C}$ is the *concept class*, a set of boolean functions. Learner knows $\mathcal{C}$ but of course does not know c. The reason for using the notation $\mathcal{C}$ rather than $\mathcal{F}$ is to allow for the possibility that Learner predicts according to a different set of hypotheses than those in $\mathcal{C}$.

We first study this setting under the online learning protocol. Consider an arbitrary sequence of examples, where the length of the sequence also can be arbitrarily large. Since we are in the realizable case, there exists a perfect hypothesis $c \in \mathcal{C}$. Suppose that Learner takes $\mathcal{F} = \mathcal{C}$, eventually plays this hypothesis, and predicts according to it thereafter. Then the total number of mistakes made by Learner is simply the number of mistakes made by Learner prior to playing hypothesis c. Thus, it is natural to try to bound the total number of mistakes made by Learner.

Definition 1. An algorithm $\mathcal{A}$ learns a class $\mathcal{C}$ in the *mistake bound model* if there there is a polynomial function $m \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that, for any $c \in \mathcal{C}$ and any sequence of examples $(x_1, c(x_1)), \ldots, (x_T, c(x_T))$, the total number of mistakes made by $\mathcal{A}$ on the sequence is at most $m(d, |c|)$, where d is the size of an input.

Two remarks are in order. For many concept classes, there is a natural way to define the concept class for each choice of the dimension of the input. Hence, the dependence of m on the size of the input d. The more pressing matter is what is meant by $|c|$. Whenever we wish to refer to $|c|$, the concept class will be associated with some representation, and $|c|$ will then refer to the size (in bits) of the concept when expressed in this representation. For example, monotone conjunctions over $\{0,1\}^d$ (see the next subsection) can be represented using d bits.

In addition to desiring a polynomial bound on the number of mistakes, we typically also care about *efficient learnability*. That is, we would like the runtime of the algorithm to be at most $\text{poly}(d, |c|)$ for each round t.

1.1 Learning monotone conjunctions

Let the input space be $\{0,1\}^d$ and the label space $\mathcal{Y}$ be $\{0,1\}$. Consider the concept class $\mathcal{C}$ consisting of the set of all monotone conjunctions. That is, each element of $\mathcal{C}$ is of the form $x_{i_1} x_{i_2} \cdots x_{i_k}$ for some $k \in \{0, 1, \ldots, d\}$; if $k = 0$, then the predicted label is always positive.

There is a simple algorithm for learning monotone conjunctions in the mistake bound model:

1. Initialize hypothesis f to the conjunction $f(x) = x_1 x_2 \cdots x_n$.
2. While there are still examples in the sequence:

Predict the label of the next example using f. If the true label is 1 but the predicted label was 0, update f by removing from the conjunction all components x_j that are zero in the example.

The idea behind the above algorithm is to begin with the most restrictive hypothesis which labels everything but the all ones vector as negative. Thus, mistakes can occur only on positive examples. In the event that a mistake occurs on a positive example, we are guaranteed that any component set to zero in the example cannot be part of the correct conjunction c, and we may thus remove such components. The algorithm thus only removes terms from the conjunction in f which are inconsistent with the data observed thus far. Moreover, each mistake leads to the removal of at least one term from the conjunction, and so there can be at most d mistakes. Therefore, the above algorithm learns the class of monotone conjunctions in the mistake bound model and makes at most d mistakes.

2 Halving algorithm

When the concept class is finite, there is a surprisingly simple algorithm that obtains a mistake bound of $\log_2 |\mathcal{C}|$. This algorithm is called the Halving algorithm, and it uses two key ideas.

The first idea is that of a *version space*. The version space is the set of hypotheses that are consistent with the data observed thus far. Thus, at the start of round t, the version space $\mathcal{V}_t$ is the subset of hypotheses from $\mathcal{C}$ which are consistent with $(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})$.

The second idea is to predict according to a majority vote. For a set of hypothesis $\mathcal{F}$, define the majority vote based on $\mathcal{F}$ as

$$\textsc{mv}_{\mathcal{F}}(x) = \begin{cases} 1 & \text{if } |\{f \in \mathcal{F} : f(x) = 1\}| \geq |\mathcal{F}|/2; \\ 0 & \text{otherwise.} \end{cases}$$

The Halving algorithm simply predicts according to the majority vote with respect to the version space in every round.

Algorithm 1: Halving Algorithm

$\mathcal{V}_1 \leftarrow \mathcal{C}$
for $t = 1 \to T$ **do**
  Observe x_t
  $f_t \leftarrow \textsc{mv}_{\mathcal{V}_t}$ (and predict $\hat{y}_t = \textsc{mv}_{\mathcal{V}_t}(x_t)$)
  Observe true label $y_t = c(x_t)$
  Set $V_{t+1} \leftarrow \{f \in V_t : f(x_t) = y_t\}$
end

How many mistakes does this algorithm make? Because it predicts according to the majority vote, wherever the algorithm makes a mistake it is guaranteed that at least half the hypotheses in the version space were wrong; thus, the version space is halved on each mistake. Formally, if the algorithm makes a mistake in round t, it holds that $|\mathcal{V}_{t+1}| \leq |\mathcal{V}_t|/2$. We initially have $\mathcal{V}_1 = \mathcal{C}$, and so if we have made M_t mistakes at the beginning of round t, it follows that $|V_t| \leq |\mathcal{C}|/2^{M_t}$. Since there exists a perfect hypothesis $c \in \mathcal{C}$, the algorithm can make at most $\log_2 |\mathcal{C}|$ mistakes.

We have just shown that any finite concept class is learnable in the mistake bound model using the Halving algorithm.

Theorem 1. *The Halving algorithm learns any finite concept class $\mathcal{C}$ in the mistake bound model and makes at most $\log_2 |\mathcal{C}|$ mistakes.*

Unfortunately, the runtime of the Halving algorithm is exorbitant. In many situations, a concept c can be represented using roughly $\log |\mathcal{C}|$ bits, in which case the runtime of the Halving algorithm is *exponential* in $|c|$ (!). In other cases, such as the case of linear separators, the concept class can even be infinite.