CS 446: Machine Learning
Lecture 4: On-line Learning

1 Introduction

This section of the notes will discuss ways of quantifying the performance of various learning algorithms. It will be possible, then, to say something rigorous about performance and work towards determining which learning algorithm is best for a given task. This section will concentrate on discussing the number of examples that are needed in order to claim that our learned hypothesis is good.

2 Learning Conjunctions

Assume that there is a hidden conjunction which you, the learner, want to learn. How many examples are needed in order to learn it? How is it learned from these examples?

For example, assume that the hidden conjunction is:

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

This section will discuss learning the conjunction with three different learning protocols.

2.1 Protocol I

In the first learning protocol, the learner proposes instances as queries to the teacher. Assume we know we are after a *monotone conjunction*, so the possible

Table 1: Queries for Protocol I

Query	Example	Valuation	Conclusion
Is x_{100} in?	$< (1,1,1,\ldots,1,0), ? >$	$f(x) = 0$	Yes
Is x_{99} in?	$< (1,1,\ldots,1,0,1), ? >$	$f(x) = 1$	No
...			
Is x_1 in?	$< (0,1,\ldots,1,1,1), ? >$	$f(x) = 1$	No

terms are the variables x_1 through x_{100} and not their negations. A straightforward algorithm requires $n = 100$ queries, and it will produce the hidden conjunction exactly as a result:

The final result of the queries is that $h = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$. This algorithm gets the right function for the data, but it requires a query for each variable.

2.2 Protocol II

In the second protocol, the teacher knows what the hidden function is and provides training examples to the learner. The teacher gives the learner a superset of the positive examples from which implications can be made. Then each variable that is in the hidden function is shown to be important. For example, the learner has the hypothesis $< (0,1,1,1,1,0,\ldots,0,1), 1 >$, which is a superset of the good variables. In order for the teacher to show that each and only the variables in the hidden conjunction are needed, negative examples are provided. For example:

Table 2: Queries for Protocol II

Example	Needed Variable
$< (0,0,1,1,1,0,\ldots,0,1), 0 >$	x_2
$< (0,1,0,1,1,0,\ldots,0,1), 0 >$	x_3
...	
$< (0,1,1,1,1,0,\ldots,0,0), 0 >$	x_{100}

Model Teaching is tricky because it involves choosing the right examples needed to get the learner to make accurate inferences. A straightforward algorithm requires $k = 6$ examples to produce the hidden conjunction exactly. $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

2.3 Protocol III

In the third protocol, some random source, for example, Nature, provides training examples and a Teacher provides the labels $(f(x))$.
$< (1,1,1,1,1,1,\ldots,1,1),1 >$
$< (1,1,1,0,0,0,\ldots,0,0),0 >$
$< (1,1,1,1,1,0,\ldots,0,1,1),1 >$
$< (1,0,1,1,1,0,\ldots,0,1,1),0 >$
$< (1,1,1,1,1,0,\ldots,0,0,1),1 >$
$< (1,0,1,0,0,0,\ldots,0,1,1),0 >$
$< (1,1,1,1,1,1,\ldots,0,1),1 >$
$< (0,1,0,1,0,0,\ldots,0,1,1),0 >$

The basic algorithm for this protocol is one of elimination.

Elimination Algorithm

```
Start with the set of all literals as candidates
    Eliminate a literal that is not active in a positive example
```

The basic idea behind the algorithm is that, since the conjunctions are monotone, the literals that are not active (i.e., that have a zero) in positive examples can provide information. Specifically, we know that these literals are not part of the hidden conjunction.

In the example below, for each example, either the hypothesis is modified or nothing is learned:

Table 3: Queries for Protocol III

Example	Fact Learned
$< (1,1,1,1,1,1,\ldots,1,1),1 >$	$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$
$< (1,1,1,0,0,0,\ldots,0,0),0 >$	*learned nothing*
$< (1,1,1,1,1,0,\ldots,0,1,1),1 >$	$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
$< (1,0,1,1,0,0,\ldots,0,0,1),0 >$	*learned nothing*
$< (1,1,1,1,1,0,\ldots,0,0,1),1 >$	$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
$< (1,0,1,0,0,0,\ldots,0,1,1),0 >$	
$< (1,1,1,1,1,1,\ldots,0,1),1 >$	

The resulting hypothesis is not exactly right, but it is consistent with the right hypothesis:

h = x$_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

Most of the work that will be covered in these notes uses the third protocol: Some source gives the data, the teacher gives the label, and the learning problem is to find a hypothesis for the labels.

3 On-line Learning

On-Line learning constitutes a different approach to the learning problem. In order to understand the motivation for On-line Learning, consider a learning problem in a very high dimensional space.

$$\{x_1, x_2, x_3, \ldots, x_{1{,}000{,}000}\}$$

And assume that the function space is very sparse, so that every function of interest depends on a small number of attributes. For example:

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

An example of such a domain is context sensitive spelling. For example, if all of the words of a sentence are considered to be features, the space is very large. In this space, only one word per sentence will represent the target concept and only a few of the words in the sentence will be relevant for disambiguation.

Middle Eastern *deserts* are known for their sweetness

The question is, can we develop an algorithm that depends only weakly on the space dimensionality and mostly on the number of relevant attributes? If so, how should the hypothesis be represented? The goal it to find a simple and intuitive model that makes the smallest number of mistakes in the long run.

The basic model for On-line Learning is below:

Model:

Instance space: X $(dimensionality - n)$

Target: $f : X \rightarrow \{0, 1\}, f \in C$ (where C is a concept class parameterized by n)

Protocol: learner is given $x \in X$

learner predicts $h(x)$, and is then given $f(x)$ as feedback

Performance: learner makes a mistake when $h(x) \neq f(x)$

Some definitions used in the following algorithms are:

$M_A(f, S)$ is the number of mistakes algorithm A makes on sequence S of examples, for the target function f.

$M_A(C) = max_{f \in C, S} M_A(f, S)$

A is a *mistake bound algorithm* for the concept class C, if $M_A(C)$ is polynomial in n, the complexity parameter of the target concept.

We could ask how many mistakes do we need to get to $\epsilon - \delta$ PAC behaviour.[1] But, instead, we are looking for exact learning which is easier to analyze. In a worst-case model, there is no notion of distribution. In order to improve memory, we can get an example, update the hypothesis, then get rid of it. But there are drawbacks to this approach. First, it is too simple. Also, the behaviour is global, and it is not clear when the mistakes will be made. The advantage, however, is that it is simple. Many issues arise already in this setting. It is possible to do generic conversion to other learning models. In addition to this, it is equivalent to PAC for 'natural' problems.

3.1 Generic Mistake Bound Algorithms

This section will present a generic algorithm that sets a bound on the number of mistakes.

The CON algorithm

```
Let $C$ be a concept class. Learn $f \in C$
    CON:
    In the ith stage of the algorithm:
    For $C_i$, all concepts in $C$ consistent with all $i-1$ previously seen examples
    Choose randomly $f_i \in C_i$ and use it to predict the next example
```

[1]PAC will be described in 05-Lec.

The intuition behind the CON algorithm is that $C_{i+1} \subseteq C_i$ and, if a mistake is made on the ith example, then $|C_{i+1}| < |C_i|$. Progress is made in this way, and the CON algorithm makes at most $|C| - 1$ mistakes.

3.2 The Halving Algorithm

This algorithm attempts to solve the problem of setting a bound on the number of mistakes better than the CON algorithm.

The Halving Algorithm

```
Let $C$ be a concept class. Learn $f \in C$
    Halving:
    In the ith stage of the algorithm:
    For $C_i$, all concepts in $C$ consistent with all $i - 1$ previously seen examples
    Given an example $e_i$, consider the value $f_j(e_i)$ for all $f \in C_i$ and predict by
    majority.

      Predict 1 if
```

$$|\{f_j \in C_i; f_j(e_i) = 0\}| < |\{f_j \in C_i; f_j(e_i) = 1\}|$$

The intuition is that $C_{i+1} \subseteq C_i$ and if a mistake is made in the ith example, then $|C_{i+1}| < \frac{1}{2}|C_i|$. The Halving algorithm makes at most $log(|C|)$ mistakes.

The problem with the Halving algorithm is that it is hard to compute. When the class C is the class of all Boolean functions, then Halving is optimal. But, in general, to be optimal, instead of guessing in accordance with the majority of the valid concepts, we should guess according to the concept group that gives the least number of expected mistakes. This is, however, even harder to compute.

4 Learning Disjunctions

Assume that there is a hidden disjunction that the learner is to learn.

$$f = x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_{100}$$

There are 3^n possible disjunction. Each value can be positive, negative, or absent. For this case, $log(|C|) = n$. The elimination algorithm makes n mistakes.

The algorithm learns from negative examples and eliminates the literals which are active in negative examples. For $k - disjunctions$: Assume that only $k << n$ attributes occur in the disjunction. The number of k-disjunctions is: $2^k C(n, k) \approx 2^k n^k$. In this case, $log(|C|) = klog(n)$. Is it possible to learn efficiently with this number of mistakes?

$Theorem$:

> Given a sample on n attributes that is consistent with a disjunctive concept, it is NP-hard to find a pure disjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes.

The same theorem holds for Conjunctions. The intuition is that the problem can be reduced to a minimum set cover problem. Given a collection of sets that cover X, define a set of examples so that learning the best disjunction implies a minimal cover. Consequently, we can not learn the concept efficiently as a disjunction. In the later notes, we will see that it is possible, however, to learn the concept efficiently as a Linear Threshold Function.