COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Tamara Broderick

Lecture #2
February 9, 2006

1 The Consistency Model

$c : X \to \{0, 1\}$ is a concept.
$\mathcal{C}$ is a concept class.
$\mathcal{C}$ is learnable (in the Consistency Model) if
There exists an algorithm A such that
For all given sets of labeled examples
$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
A finds a consistent $c \in \mathcal{C}$
or correctly returns that there is no consistent concept

A concept c is consistent if $\forall i, c(x_i) = y_i$ where i indexes the training examples, $x_i \in X$ are the instances (a.k.a. unlabeled training examples), and y_i are the boolean labels.

A few clarifications on the concept of learnability: It doesn't make sense to talk about a single function or pattern being learnable. We could program such a function directly into our algorithm. Rather, we consider the learnability of a class of patterns or concepts.

Also, note that if $\mathcal{C}$ is finite, we could always perform a brute force search. This search will provide the necessary algorithm A and trivially such a $\mathcal{C}$ is therefore learnable. In general, though, we are actually interested in efficient learning. What constitutes efficiency? For the remainder of these notes, we say that the class is learnable if it is learnable in "polynomial time" and later clarify what the variable of the polynomial is.

1.1 Example

$X = \{0, 1\}^n$ (that is, the examples are bit vectors of length n)
$\mathcal{C}$ is the set of monotone conjunctions

A monotone conjunction is a conjunction of the variables (a.k.a. features, attributes, dimensions) z_i such that no variable in the conjunction is negated.

Example boolean variables:
z_1 = "is a bird" ($z_1 = 1$ if this is true; otherwise it is 0)
z_2 = "is a mammal"
z_3 = "lays eggs"
An animal (an instance of X) is characterized by a bitstring:
$x = 10100$ describes a bird that lays eggs but is not a mammal and also does not satisfy the last two conditions.

Example monotone conjunction:
$z_2 \wedge z_3$ (characterizes a mammal that lays eggs)
The conjunction is true only if all of the boolean variables within it are true. The empty conjunction is defined to be always true.

$\mathcal{C}$ is learnable in this model.
The algorithm A is:

Take the bitwise "and" of all the positively labeled instances
(that is, find all the variables that are equal to 1
for all (+)-labeled instances)
Construct a monotone conjunction over the variables chosen this way.
Check the monotone conjunction on the negative examples.
If it returns (+) for any negative example, there is no consistent concept.
Otherwise, this monotone conjunction is the desired consistent concept.

Example set of instances and labels:

example	**label**
0 1 1 0 1	+
1 1 0 1 1	+
1 1 0 0 1	+
0 0 1 0 1	−
1 1 0 0 0	−

Here, the consistent concept is $z_2 \wedge z_5$, where the leftmost bit in any instance is z_1.

Justification:

By construction, the monotone conjunction generated from the positive examples is consistent with those labels. Moreover, it is the strictest concept consistent with those labels in the sense that deleting any variable from the conjunction could only introduce more satisfying examples of the conjunction. And adding any variable to the conjunction would cause it to be inconsistent with the positive examples. Therefore, if any negative example is classified as positive by the constructed concept, the concept cannot be modified to accommodate it. So there is no consistent concept in this case.

1.2 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of monotone disjunctions (e.g. $z_2 \vee z_5$).

Using DeMorgan's Law, we can reduce this case to the previous case.
DeMorgan's Law:

$$z_1 \vee z_2 = \overline{\overline{z_1} \wedge \overline{z_2}}$$

Therefore, we want to find a rule that returns 0 whenever a variable is 0 across all of the negative examples. Then we check for consistency with the positive examples.

1.3 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of all conjunctions, not necessarily monotone (e.g. $z_2 \wedge \overline{z_3} \wedge z_5$)

By adding a new variable that represents the negation of each existing variable, we can reduce this to the case of monotone conjunctions.

Essentially, the algorithm looks for any variable that is always assigned 1 throughout the positive examples (and adds it to the conjunction) and any variable that is always assigned 0 throughout the positive examples (and adds its negation to the conjunction). Then it checks for consistency with the negative examples.

Copying the table from above, the instances with new variables included would be:

example	**label**
0 1 1 0 1 — 1 0 0 1 0	+
1 1 0 1 1 — 0 0 1 0 0	+
1 1 0 0 1 — 0 0 1 1 0	+
0 0 1 0 1 — 1 1 0 1 0	−
1 1 0 0 0 — 0 0 1 1 1	−

1.4 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of k-CNF sentences

A sentence in Conjunctive Normal Form (CNF) is a conjunction of disjunctions. For example, $(z_1 \vee \overline{z_3}) \wedge (z_7) \wedge (z_2 \vee z_3)$. Each disjunction of variables delineated by two conjunctions is a clause. In this example, there are three clauses: $z_1 \vee \overline{z_3}$, z_7, and $z_2 \vee z_3$. A sentence is in k-CNF if every clause has at most k variables. We think of k as a small constant (e.g. 2 or 3).

A possible algorithm is as follows. List all possible clauses of size at most k. There are $O(n^k)$ such clauses, a bound which is polynomial in n for fixed k. Create a new variable for each such clause. Then the problem is reduced to the case of monotone conjunction learnability.

1.5 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of k-DNF sentences

A sentence in Disjunctive Normal Form (DNF) is a disjunction of conjunctions. This example is similar to the previous one.

1.6 Example

$X = \mathbb{R}^2$ (that is, examples are points in the plane)
$\mathcal{C}$ is the set of rectangles in the plane with sides parallel to the horizontal and vertical axes

To determine if there is an enclosing rectangle and find such a rectangle if there is one, we use the following algorithm. Find the leftmost, rightmost, bottommost, and topmost points of the positively labeled examples. Use these to draw the tighest possible rectangle around the positive points. If there is a negative point in this rectangle, there is no consistent concept. Otherwise, the rectangle found is the desired concept.

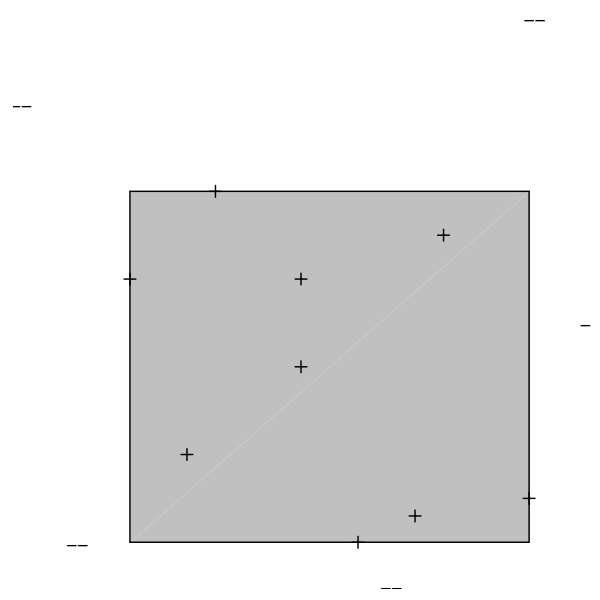

Figure 1: A training set and bounding rectangle

1.7 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of 2-term DNF sentences

A term is any conjunction delineated by two disjunctions in the DNF sentence. For instance, $z_1 z_3 \overline{z_5} \vee z_2 z_3 z_7 \overline{z_{11}} z_{15}$ is a two-term DNF sentence, where the $\wedge$ symbol is suppressed within each term for convenience. The first term in this sentence is $z_1 z_3 \overline{z_5}$.

It turns out that learning 2-term DNF sentences in the Consistency Model is NP-complete. In other words, there is no polynomial time algorithm that learns 2-term DNF sentences unless P = NP. This phenomenon seems bizarre because any 2-term DNF sentence can be reduced to a 2-CNF sentence, which is learnable in the model. Thus, we have an example of a larger class that is easier to learn in this model than a smaller class.

1.8 Example

$X = \{0,1\}^n$
$\mathcal{C}$ is the set of DNF sentences

This concept class is trivially learnable in the Consistency Model. Just form a conjunction corresponding to every positively labeled example and the disjunction of these conjunctions is a consistent DNF sentence. For example, copying the table above:

example	**conjunction**	**label**
0 1 1 0 1	$\overline{z_1} z_2 z_3 \overline{z_4} z_5$	+
1 1 0 1 1	$z_1 z_2 \overline{z_3} z_4 x5$	+
1 1 0 0 1	$z_1 z_2 \overline{z_3 z_4} z_5$	+
0 0 1 0 1		−
1 1 0 0 0		−

Then a DNF consistent with the training data is

$$\overline{z_1} z_2 z_3 \overline{z_4} z_5 \vee z_1 z_2 \overline{z_3} z_4 z_5 \vee z_1 z_2 \overline{z_3 z_4} z_5$$

However, this algorithm doesn't seem to have actually learned anything. The only examples that it will ever label positive are the positive examples in the training data.

1.9 Problems with the Consistency Model

The Consistency Model doesn't seem to have any connection to learning as we know it.

1. It makes no generalizations.
2. There is no justification for why the algorithms will perform well in classifying the test examples. In fact, there is no reason to think that they will.
3. There are cases where the learning is trivial (DNF).
4. It seems bizarre that a smaller class is unlearnable (according to our polynomial-time restriction) when a larger, encompassing class is learnable.
5. The model does not allow noise.

2 Probability Overview

Terminology:

- event
- random variable X; $\Pr[X = x] = p$
- joint distribution $\Pr[X, Y]$
- expected value $E[X]$

  $E[c] = c$

  $E[cX] = cE[X]$

  $E[X + Y] = E[X] + E[Y]$

  $E[XY] =$? in general
- conditional probability $\Pr[a|b]$
- independence

  If X, Y are independent, then $E[XY] = E[X]E[Y]$
- union bound: $\Pr[a \vee b] \leq \Pr[a] + \Pr[b]$

  Equality holds if a and b are disjoint
- disjoint: a and b are disjoint if whenever a holds, b cannot be true

Example use of union bound:

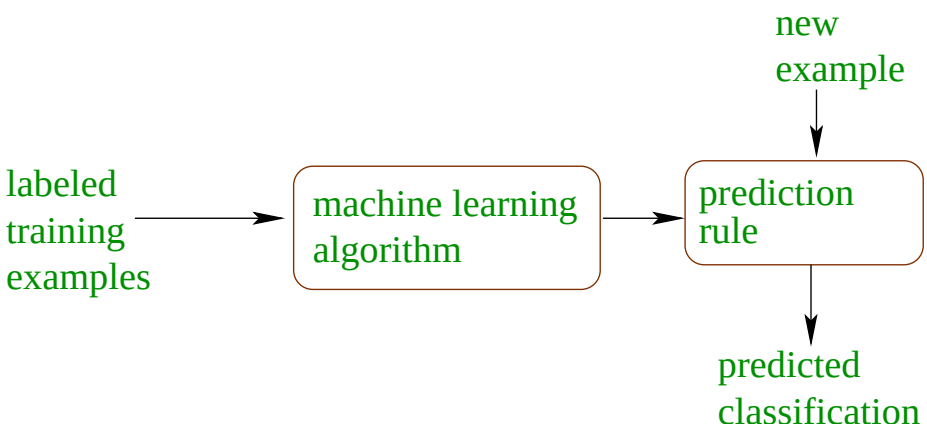

Figure 2: Diagram of a typical learning problem.

- l_i is the event that the i^{th} person will be hit by lightning, and $\Pr[l_i] = 1/1000$.
- There are 1 million people

Then $\Pr[l_1 \vee l_2 \vee \cdots \vee l_{1000}] \leq \sum_{i=1}^{1000} \Pr[l_i] = 10^3/10^6 = 1/1000$. Note that the union bound rule is generally meaningful only for small probabilities and relatively few events.

2.1 Marginalization

General Form:

$$\begin{aligned} \Pr[a] &= \sum_x \Pr[a \wedge X = x] \\ &= \sum_x \Pr[X = x] \Pr[a | X = x] \\ &= E_X[\Pr[a | X = x]] \end{aligned}$$

The last line is the expected value taken over the random setting of the random variable X.

3 A Better Model

In the Consistency Model, we didn't look at the accuracy of the predictions of our hypothesis or where the data was coming from. We need to know the latter to measure prediction accuracy.

We want to include the following assumptions in any new model:

- Examples are random according to some probability distribution D, which is called the target distribution.
- The model should be distribution-free. That is, the model works for any distribution; the learning algorithm does not have any information about the distribution.
- The distribution is the same for both training and testing.

- There is some target concept c that belongs to a concept class $\mathcal{C}$ and that labels the examples. This c is the unknown function we are trying to learn.

If the training data is random (one of our assumptions), then there is always a small possibility δ that it will be highly unrepresentative. We want to control this probability of failure.

Also, let h be the output hypothesis. Then the true error or generalization error is defined to be

$$err_D(h) = \Pr_D[h(x) \neq c(x)].$$

We want to find h with a low error; in other words, we want to set an upper bound of $\epsilon > 0$ on the error given above. Our hypothesis should be "approximately correct."

But there's always a chance that it is impossible to arbitrarily bound the error of h because of a highly abnormal training set. Thus, we want the algorithm generating h to be "probably approximately correct" (PAC).

3.1 PAC-Learnability

$\mathcal{C}$, $\mathcal{H}$ are concept classes; they are not necessarily the same

$\mathcal{C}$ is PAC-learnable by $\mathcal{H}$ if

There exists an algorithm A such that
$\forall c \in \mathcal{C}$
$\forall D$
$\forall \epsilon > 0$
$\forall \delta > 0$

A takes $m = \text{poly}\left(\frac{1}{\epsilon}, \frac{1}{\delta}, \ldots\right)$ examples from D

$S =< x_1, c(x_1), \ldots, x_m, c(x_m) >$ (the training set)

A produces $h \in \mathcal{H}$ such that

$\Pr[err_D(h) \leq \epsilon] \geq 1 - \delta$

The probability in the last line is over the choice of S. If A is randomized, it is also over A's randomization.