COS 424: Interacting with Data

Lecturer: Robert Schapire
Scribe: Soner Sevinc

Lecture #6
February 22, 2007

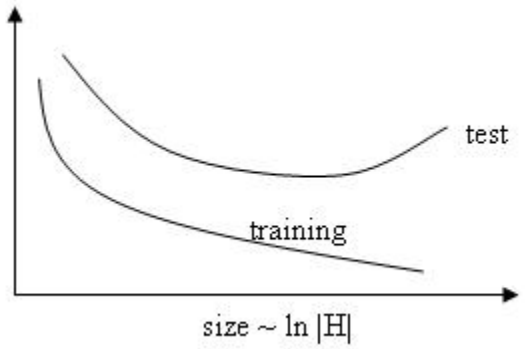
1 Theory (cont)

In the last lecture we have examined the prediction rules (hypotheses) from a theoretical point of view; specifically we have proved an upper bound on the error rate of the prediction rules which are consistent with their training data (fit the data without error). The following formula holds with probability $\geq 1 - \delta$ if $h_A \in H$ is consistent:

$$err(h_A) \leq \frac{\ln |H| + \ln \frac{1}{\delta}}{m}$$

In this lecture, we start by examining the prediction rules which can have training error (inconsistent with the training data). The training error will be added to the right hand side of the above inequality. Since this is an additional error, why use inconsistent rules? The reason is that, it can be possible to reduce the existing right hand side term by lowering H's complexity. It can be proved that:

$$\text{Generalization error} = err(h) \leq \text{training error} + O(\sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{m}})$$



For instance, as the size of a decision tree increases, the second term in the right hand side of the inequality will begin to rise, since tree size $\approx \ln |H|$. So, the preferred tree size does not always make the training error 0. As a result, the theoretical upper bound inequality becomes as above.

This inequality is meaningful only if $|H|$ is finite. For infinite hypothesis spaces, we can use the Vapnik-Chervonenkis dimension to measure complexity of algorithms. Shortly shown as VC-dim(H), it is meaningful even if we have an infinite number of hypotheses in H.

2 More Learning Algorithms

Here, we close the discussion of theoretical aspects of the decision rules, and move on to some specific learning algorithms. Previously we have seen two learning algorithms; the nearest neighbor algorithm and decision trees. But in practice, they are not preferred; now we describe some of the state of the art algorithms.

2.1 Boosting

This algorithm is used in real life scenarios. One example is building spam filters. It is very hard to come up with a complete rule to decide whether a mail is spam or not. It is easy to do a little better than random guessing; for example if a medicine name is in the text, then we say it can be spam. The aim of boosting is to come up with a highly accurate rule by combining the less accurate rules. Here is an outline:

- Devise a program for finding rough "rules of thumb".
- Run this program on a first subset of examples we have,
  Get the first rule of thumb.
- Run on the second subset
  Get the second rule of thumb.
- Repeat T times.
- Finally combine the rule of thumbs.

Some important questions:

- How to choose the subset of the examples at each step? If we use the same set of examples in each step, we will obtain the same rule of thumb, but we want the rules to be different from each other. Another idea in selecting the subsets can be random selection. The idea followed in boosting is to focus on the hard examples; to choose the examples which were misclassified in the previous rounds.

- How to combine the different rule of thumbs? One simple solution is to get the majority or the weighted majority vote.

We call the rules of thumb also as "weak (or base) hypotheses (WH)". WH's are obtained by the learning algorithm called the "weak (or base) learner (WL)". The boosting algorithm relies upon an assumption:

Weak Learning Assumption: Each of the weak learners is better than random guessing. A weak learner can consistently find weak hypotheses with $error < \frac{1}{2} - \gamma, (\gamma > 0)$. That means, the learning algorithm can come up with new hypotheses which are guaranteed to be little better than random guessing. Given this assumption, a boosting algorithm can drive the error down lower than ϵ.

$$\text{Can get error} < \epsilon \ \forall \epsilon > 0$$

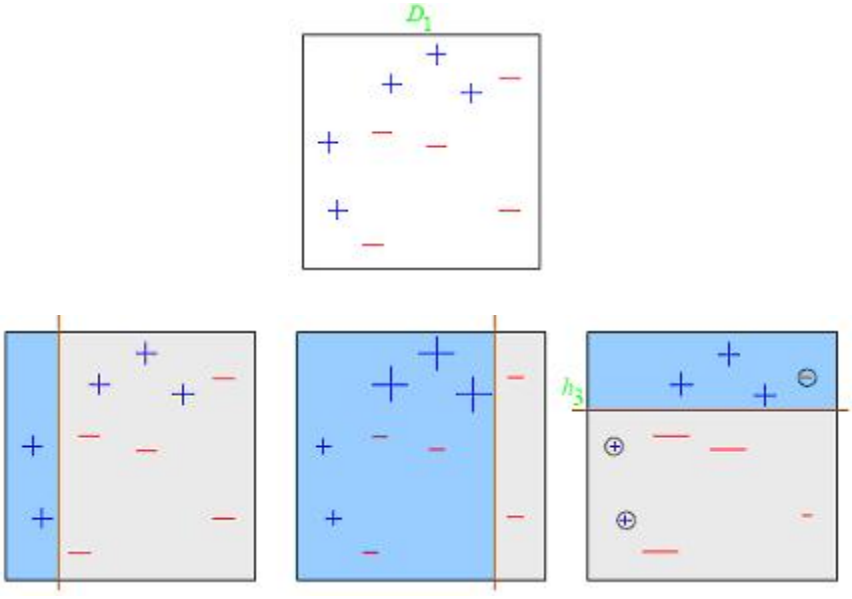
This gives: You either cannot do a little better than random guessing, or you can do close to perfect. There is nothing in the middle.

Boosting Algorithm: Now we give the boosting algorithm which is called the AdaBoost. Given $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m), x_i \in X$ examples, $y_i \in \{-1, 1\}$ labels, the algorithm works as follows:

$D_1(i) = \frac{1}{m}$ # initial weight vector.
for $t = 1, 2, \ldots, T$
  choose D_t # weight vector for examples.
    $D_t(i)$ = weight on example (x_i, y_i)
    ≥ 0 # always greater than 0.
    $\sum_{i=1}^{m} D_t(i) = 1$
  train WL on D_t => obtain a WH h_t # WL is itself a program.
  measure goodness of h_t by
    $\epsilon_t = Pr_{i \sim D_t}[y_i \neq h_t(x_i)]$
    $= \sum_{h_t(x_i) \neq y_i} D_t(i)$ # At most $\frac{1}{2} - \gamma$
  update D_t : increase the weight of misclassified examples:
    increase if $h_t(x_i) \neq y_i$
    decrease if $h_t(x_i) = y_i$

The algorithm finds a new weak hypothesis at each iteration. The training examples are given different weights in different iterations. At $t = 1$, the weights are the same for all examples; $\frac{1}{m}$. The sum of weights is always 1. After giving the data together with the weights to the learning algorithm, we obtain a WH h_t. Then, we calculate the error of the classifier by adding the misclassified example weights. The misclassified examples are given higher weights for the next iteration.

This algorithm can be illustrated with a toy example, in which the algorithm is trying to locate the regions of plus or minus in a rectangular area. The upper figure shows the layout of the region with plus and minus signs.



Each of the three figures below represents a hypothesis. The first hypothesis separates the region into two regions, but three pluses are misclassified, so their weights are increased.

In the next iteration, hypothesis two correctly separates pluses, but several minuses are misclassified, and iteration continues to reduce the total error again.

The updating of D_t which is at the end of the algorithm can be formalized as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{\alpha_t} \text{ if } h_t(x_i) \neq y_i$$
$$= \frac{D_t(i)}{Z_t} e^{-\alpha_t} \text{ if } h_t(x_i) = y_i$$

where $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} > 0$, Z_t = normalization constant.

Note that, $e^{\alpha_t} > 1$ and $e^{-\alpha_t} < 1$. So, if the sample i was misclassified, the weight is reduced; otherwise increased.

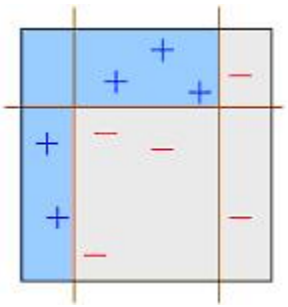
The meaning of this formula is that the weight is increased if previous hypothesis made wrong prediction, decreased otherwise. For our discussion, it is enough to know that α_t is a positive quantity. D_t is normalized by dividing by Z_t at each iteration.

Now, how to combine the hypotheses? We can simply get the weighted majority vote of the hypotheses. Final/combined hypothesis:

$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + ... + \alpha_T h_T(x)),$$

where α_t is the importance of each WH. This completes the algorithm definition.

In the toy example we gave, the combination of hypotheses results in such a separation of the region:



THEOREM: The next theorem gives a bound on the training error. By using the weak learning assumption, it is possible to compute an upper bound to the training error of the combined hypothesis H. T is the number of rounds.

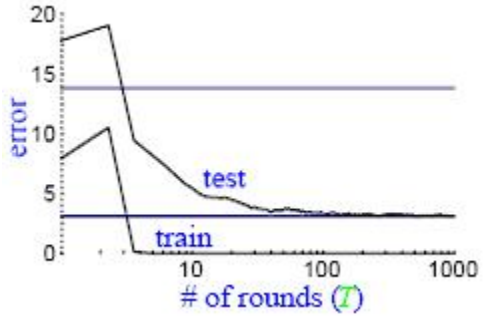
If $\epsilon_t = \frac{1}{2} - \gamma_t$, then
train err(H) $\leq exp(-2 \sum_{t=1}^{T} \gamma_t^2)$

For instance, if $\epsilon_t \leq 0.45$, then it means that $\gamma_t \geq 0.05$
So, train error $\leq exp(-2 * (0.05)^2 * T)$,
which decreases exponentially fast in the number of rounds T.

In the example we see that the training error of the algorithm is bounded by a value which is a function of number of rounds T, and this bound is dropping exponentially as T increases. This behavior is consistent with the idea that the training error will decrease

as the decision tree size is increased. If we regard each weak hypothesis as a decision tree, more rounds will cause the combined tree to be larger.

In the below figure, the results of an actual typical run are depicted. As the number of rounds T increases, the training error decreases as we expected.

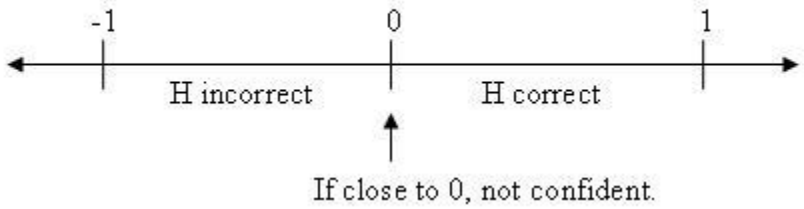


Another question is how the test error will change according to the number of rounds T changes. In the previous learning algorithm that we have studied, namely decision trees, we have seen that as the tree size increases, the test error tends to fall first, and then rise (Occam's Razor principle). In the boosting algorithm we also expect a similar behavior; as the number of rounds T increases, the total tree size for the learning algorithm increases, so we should observe the Occam's Razor taking effect. But, in the above figure of the actual typical run, we see that this does not happen in practice. The test error continues to fall, and approaches a constant value as T increases.

The fact that the test error does not rise as T increases is contradictory to the Occam's Razor principle. We explain this phenomenon with another interpretation of the test error with respect to the number of rounds. Below, a new perspective to predict the test error with respect to T is explained.

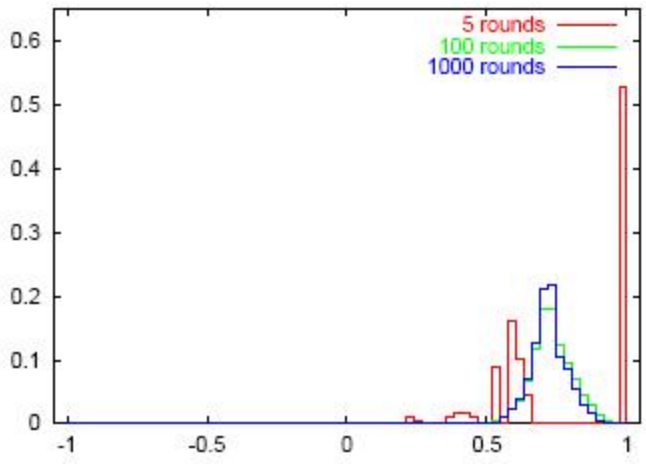
The margin distribution: It is the distribution of the margins of the training examples. The margin of an example gives us clue about the confidence of that result.

Margin = (weighted fraction of the WHs correct for a specific example)- (weighted fraction of the WHs incorrect for that example)



The above figure shows the values that the margin can take. If the margin is close to 0, that means the label predicted for the example has less confidence.

When we examine the behavior of the margin distribution with respect to T, we see that the margin increases as T increases. This situation is depicted in the below figures. The first figure tells that as T is increased, the mean of the margin distribution is more likely to lie closer to 1. In the second figure we can see the minimum margins of the training examples as T is increased. As T increases, the minimum observed margin increases, and the error on the test set decreases.



	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

At the last part of the lecture, we saw an application of the boosting algorithm. Face detection is performed by simple WHs, and by boosting the algorithm can learn and detect the human faces to a great accuracy. The example weak hypotheses are shown in the first two figures below. By checking the arrangement of the dark and light areas in a picture, these weak hypotheses aim at estimating whether a picture is a face or not. During the lecture, we are shown a video in which the faces appearing in the video are detected automatically by the algorithm which is obtained by boosting.