Ada Boost

CS 360 Machine Learning
Week 4, Day 2

February 15, 2024

Contents

1 Ada Boost

To recap what we talked about with ensembles, recall ensembles can decrease our testing error and make our models less brittle. The overall goal is to lower both bias and variance, and by choosing a base classifier that has high variance and low bias, when we average over multiple models we can achieve both low(er) variance and low bias. Recall also that *bootstrap sampling* means sampling with replacement, which allows us to generate different training datasets that have different (potentially repeated) examples making up new datasets.

Recall that AdaBoost (adaptive boosting) starts with all examples equally weighted ($1/n$) and over T iterations you first learn the base classifier with these weights trained on the full training data set, and then change the weights for each example in the training dataset for future iterations (such that the sum of the weights is always 1). Recall that this is distinct from the ensemble methods we mostly discussed in the previous class since it uses the same training dataset but modifies the weights, and each iteration depends on the previous one. In order to perform testing / predictions, you get the predictions from all the base classifiers, and then vote based on how well each classifier did during training.

Now let's discuss the details of the AdaBoost algorithm.

1.1 AdaBoost Algorithm Overview

Input

A set of n training examples, each with p features (represented by matrix $\boldsymbol{X}$), and a vector of labels $\boldsymbol{y}$ where each $y \in \{-1, 1\}$ (binary classification). We also need to choose a family of classifiers (i.e. decision stumps) that will work with *weighted* training examples, and a number of iterations T.

Initialization

- Assign uniform weights to all training data points: $w_i^{(1)} = \frac{1}{n}$ for $i = 1, 2, \cdots, n$. Note that we require the weights to sum to 1.

Adaptive Procedure

For $t = 1, 2, \cdots T$, use the following procedure to find a new classifier and update the weights on the training examples:

(a) Fit a classifier to the weighted training set. We will call this classifier $h^{(t)}(\boldsymbol{x})$.

(b) Compute weighted classification *error* on the training set:

$$\varepsilon = \sum_{i=1}^{n} w_i^{(t)} \mathbb{1}\left(y_i \neq h^{(t)}(\boldsymbol{x}_i)\right)$$

Note that since the weights sum to 1, $0 \leq \varepsilon_t \leq 1$. However, since we are in a binary classification scenario, we should never have an error greater than 0.5.

(c) Compute the *score* of the classifier (we would like to have a high score):

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

The score is 0 when $\varepsilon_t = \frac{1}{2}$ (random guessing). As $\varepsilon_t \to 0$, $\alpha_t \to \infty$ (i.e. a very good classifier).

(d) Using the score, update the weights on all the training examples. These are the weights we will use for the next iteration:

$$w_i^{(t+1)} = c_t \; w_i^{(t)} \; \exp\left(-y_i \alpha_t h^{(t)}(\boldsymbol{x}_i)\right)$$

where c_t is a normalizer to ensure all the weights sum to 1:

$$c_t = \frac{1}{\sum_{i=1}^{n} w_i^{(t)} \; \exp\left(-y_i \alpha_t h^{(t)}(\boldsymbol{x}_i)\right)}$$

If the data point is classified correctly, y_i and $h^{(t)}(\boldsymbol{x}_i)$ have the same sign and thus the previous weight is multiplied by $e^{-\alpha_t}$. Since α_t is positive, this quantity is less than 1, so the weight is *decreased.* If the data point is classified incorrectly, y_i and $h^{(t)}(\boldsymbol{x}_i)$ have the opposite sign and thus the previous weight is multiplied by $e^{-\alpha_t}$. This is greater than 1, so the weight is *increased.*

This can also be expressed as:

$$w_i^{(t+1)} = \begin{cases} c_t \; w_i^{(t)} \; e^{-\alpha_t} & \text{if } y_i = h^{(t)}(\boldsymbol{x}_i) \text{ (down-weight correct examples)} \\ c_t \; w_i^{(t)} \; e^{\alpha_t} & \text{if } y_i \neq h^{(t)}(\boldsymbol{x}_i) \text{ (up-weight incorrect examples)} \end{cases}$$

Testing

For each test data point $\boldsymbol{x}$, classify it as 1 or -1 using each classifier. Then weight the predictions by the score of the classifier during training. So our final prediction/hypothesis for the example is:

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h^{(t)}(\boldsymbol{x})\right)$$

Decision Tress with Weighted Examples

AdaBoost is a general algorithm, but it requires that the classifiers be able to work with weighted training examples. For decision trees, this means we need to compute *weighted* conditional entropy, as well as weighted classification results at the leaves.

Recall from CS 260 that entropy is defined as:

$$H(Y) = -\sum_{i \in vals(y)} p(i) \log_2 p(i)$$

Conceptually, if half of the values in the dataset are "yes" and the other half have a "no" label, that's the highest possible entropy of the dataset, while a dataset with all "yes" labels has an entropy of 0 (and the same for a dataset where all the labels are "no.")

Conditional entropy considers $H(Y|X)$ where X is one of the features:

$$H(Y|X) = \sum_{v \in vals(x)} p(X = v)\ H(Y|X = v)$$

This essentially weights by the number of examples that fall into each value category, and helps us to understand how "ordered" or "predictable" the different subsets of the data are. For the j^{th} feature equalling value v, our definition of conditional entropy is:

$$H(Y|X_j = v) = -\sum_{c \in \text{vals}(y)} P(Y = c|X_j = v) \log P(Y = c|X_j = v)$$

This is the base computation that helps us to create decision trees, and other important models. These probabilities are computed empirically based on our data, i.e., we count the occurrence and co-occurrence of the values in the dataset:

$$p(Y = c|X = v) = \frac{count(Y = c, X = v)}{count(X = v)}$$

But then in order to do weighted training examples, we need to calculate some of these values with weights. Consider the conditional probability calculation, instead of counting the examples, we'll count their weights:

$$p(Y = c|X_j = v) = \frac{\sum_{i=1}^{n} w_i^{(t)} \mathbb{1}(y_i = c, x_{ij} = v)}{\sum_{i=1}^{n} w_i^{(t)} \mathbb{1}(x_{ij} = v)}$$

We need to use a similar procedure for *every* probability involved in our entropy computations. When we reach a leaf and need to decide which label to apply, we will again use weighted counts of the training examples that fall into the partition at this leaf:

$$P(\text{leaf label is 1}) = \frac{\sum_{i \text{ in leaf}} w_i^{(t)} \mathbb{1}(y_i = 1)}{\sum_{i \text{ in leaf}} w_i^{(t)}}$$

Note that in principle, the denominator is 1 since all the weights sum to 1. However, this is only if we have been updating the weights each time we partition the data as we build the tree. We can alternatively not modify the weights during tree building (since this interacts poorly with the AdaBoost algorithm) and then just count them up in the denominator.

2 Gradient Boosting

Gradient boosting is most standard in a regression context, while until now we've been talking about classification. Suppose that I have a training set which follows a noisy quadratic, and suppose that I have a model which is a piecewise constant. Now suppose that I have one model, this piecewise constant model, in my ensemble. Gradient boosting considers the residuals–the distance from the training set examples to the model. Then gradient boosting tries to fix a model to those residuals. The resulting model now adds the original model to the model of the residuals. This will still not exactly fit the data, but it'll be closer! You can repeat this trick (many times if you want), again adding a model of the residuals to create the full ensemble model. Since each classifier on its own is simple, this doesn't suffer from overfitting. See Chapter 7 of the textbook or the slides for a nice figure illustrating this process.