Machine Learning and Computational Statistics
Homework 7: Ensemble Methods

Due: Tues December 12, 2017

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

1 [Optional] Gradient Boosting Machines

Recall the general gradient boosting algorithm[1], for a given loss function ℓ and a hypothesis space $\mathcal{F}$ of regression functions (i.e. functions mapping from the input space to $\mathbf{R}$):

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:

   (a) Compute:

$$\mathbf{g}_m = \left(\frac{\partial}{\partial f(x_j)} \sum_{i=1}^{n} \ell\left(y_i, f(x_i)\right) \Bigg|_{f(x_i)=f_{m-1}(x_i),\, i=1,\ldots,n} \right)_{j=1}^{n}$$

   (b) Fit regression model to $-\mathbf{g}_m$:

$$h_m = \underset{h\in\mathcal{F}}{\arg\min} \sum_{i=1}^{n} \left((-\mathbf{g}_m)_i - h(x_i) \right)^2 .$$

   (c) Choose fixed step size $\nu_m = \nu \in (0,1]$, or take

$$\nu_m = \underset{\nu>0}{\arg\min} \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) + \nu h_m(x_i)\right).$$

   (d) Take the step:

$$f_m(x) = f_{m-1}(x) + \nu_m h_m(x)$$

[1] Besides the lecture slides, you can find an accessible discussion of this approach in http://www.saedsayad.com/docs/gbm2.pdf, in one of the original references http://statweb.stanford.edu/~jhf/ftp/trebst.pdf, and in this review paper http://web.stanford.edu/~hastie/Papers/buehlmann.pdf.

3. Return f_M.

In this problem we'll derive two special cases of the general gradient boosting framework: L_2-Boosting and BinomialBoost.

1. Consider the regression framework, where $\mathcal{Y} = \mathbf{R}$. Suppose our loss function is given by

$$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2,$$

   and at the beginning of the m'th round of gradient boosting, we have the function $f_{m-1}(x)$. Show that the h_m chosen as the next basis function is given by

$$h_m = \underset{h \in \mathcal{F}}{\arg\min} \sum_{i=1}^{n} \left[(y_i - f_{m-1}(x_i)) - h(x_i)\right]^2.$$

   In other words, at each stage we find the base prediction function $h_m \in \mathcal{F}$ that is the best fit to the residuals from the previous stage. [Hint: Once you understand what's going on, this is a pretty easy problem.]

2. Now let's consider the classification framework, where $\mathcal{Y} = \{-1, 1\}$. In lecture, we noted that AdaBoost corresponds to forward stagewise additive modeling with the exponential loss, and that the exponential loss is not very robust to outliers (i.e. outliers can have a large effect on the final prediction function). Instead, let's consider the logistic loss

$$\ell(m) = \ln\left(1 + e^{-m}\right),$$

   where $m = yf(x)$ is the margin. Similar to what we did in the L_2-Boosting question, write an expression for h_m as an argmin over $\mathcal{F}$.

2 From Scores to Conditional Probabilities

[2]

Let's consider the classification setting, in which $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, 1\}$ are sampled i.i.d. from some unknown distribution. For a prediction function $f : \mathcal{X} \to \mathbf{R}$, we define the **margin** on an example (x, y) to be $m = yf(x)$. Since our class predictions are given by $\text{sign}(f(x))$, we see that a prediction is correct iff $m(x) > 0$. We have said we can interpret the magnitude of the score $|f(x)|$ as a measure of confidence. However, it is not clear what the "units" of the score are, so it is hard to interpret the magnitudes beyond saying one prediction score is more or less confident than another. In this problem, we investigate how we can translate the score into a conditional probability, which is much easier to interpret. In other words, we are looking for a mapping $f(x) \mapsto p(y = 1 \mid x)$.

In this problem we will consider margin-based losses. A loss function is a **margin-based loss** if it can be written in terms of the margin $m = yf(x)$. We are interested in how we can go from an empirical risk minimizer of a margin loss, $\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(y_i f(x_i))$, to a conditional probability estimator $\hat{\pi}(x) \approx p(y = 1 \mid x)$. Our approach will be to try to find a way to use

[2] This problem is based on Section 7.5.3 of Schapire and Freund's book *Boosting: Foundations and Algorithms*.

the Bayes prediction function[3] $f^* = \arg\min_f \mathbb{E}_{x,y}\left[\ell(yf(x)\right]$ to get the true conditional probability $p(y = 1 \mid x)$, and then apply the same mapping to the empirical risk minimizer. While there is plenty that can go wrong with this "plug-in" approach (primarily, the empirical risk minimizer from a [limited] hypothesis space $\mathcal{F}$ may be a poor estimate for the Bayes prediction function), it is at least well-motivated, and it can work well in practice. And **please note** that we can do better than just hoping for success: if you have enough validation data, you can directly assess how well "calibrated" the predicted probabilities are. This blog post has some discussion of calibration plots: https://jmetzen.github.io/2015-04-14/calibration.html.

It turns out it is straightforward to find the Bayes prediction function f^* for margin losses, at least in terms of the data-generating distribution: For any given $x \in \mathcal{X}$, we'll find the best possible prediction $\hat{y}$. This will be the $\hat{y}$ that minimizes

$$\mathbb{E}_y\left[\ell\left(y\hat{y}\right) \mid x\right].$$

If we can calculate this $\hat{y}$ for all $x \in \mathcal{X}$, then we will have determined $f^*(x)$. We will simply take

$$f^*(x) = \underset{\hat{y}}{\arg\min}\, \mathbb{E}_y\left[\ell\left(y\hat{y}\right) \mid x\right].$$

Below we'll calculate f^* for several loss functions. It will be convenient to let $\pi(x) = \mathbb{P}\left(y = 1 \mid x\right)$ in the work below.

1. Write $\mathbb{E}_y\left[\ell\left(yf(x)\right) \mid x\right]$ in terms of $\pi(x)$ and $\ell\left(f(x)\right)$. [Hint: Use the fact that $y \in \{-1, 1\}$.]

2. Show that the Bayes prediction function $f^*(x)$ for the exponential loss function $\ell\left(y, f(x)\right) = e^{-yf(x)}$ is given by

$$f^*(x) = \frac{1}{2}\ln\left(\frac{\pi(x)}{1-\pi(x)}\right)$$

and, given the Bayes prediction function f^*, we can recover the conditional probabilities by

$$\pi(x) = \frac{1}{1+e^{-2f^*(x)}}.$$

[Hint: Differentiate the expression in the previous problem with respect to $f(x)$. To make things a little less confusing, and also to write less, you may find it useful to change variables a bit: Fix an $x \in \mathcal{X}$. Then write $p = \pi(x)$ and $\hat{y} = f(x)$. After substituting these into the expression you had for the previous problem, you'll want to find $\hat{y}$ that minimizes the expression. Use differential calculus. Once you've done it for a single x, it's easy to write the solution as a function of x.]

3. Show that the Bayes prediction function $f^*(x)$ for the logistic loss function $\ell\left(y, f(x)\right) = \ln\left(1 + e^{-yf(x)}\right)$ is given by

$$f^*(x) = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right)$$

[3]In this context, the Bayes prediction function is often referred to as the "population minimizer." In our case, "population" referes to the fact that we are minimizing with respect to the true distribution, rather than a sample. The term "population" arises from the context where we are using a sample to approximate some statistic of an entire population (e.g. a population of people or trees).

and the conditional probabilities are given by

$$\pi(x) = \frac{1}{1 + e^{-f^*(x)}}.$$

Again, we may assume that $\pi(x) \in (0, 1)$.

4. [Optional] Show that the Bayes prediction function $f^*(x)$ for the hinge loss function $\ell(y, f(x)) = \max(0, 1 - yf(x))$ is given by

$$f^*(x) = \operatorname{sign}\left(\pi(x) - \frac{1}{2}\right).$$

Note that it is impossible to recover $\pi(x)$ from $f^*(x)$ in this scenario. However, in practice we work with an empirical risk minimizer, from which we may still be able to recover a reasonable estimate for $\pi(x)$. An early approach to this problem is known as "Platt scaling": https://en.wikipedia.org/wiki/Platt_scaling.

3 [Optional] Decision Tree Implementation

In this problem we'll implement decision trees for both classification and regression. The strategy will be to implement a generic class, called Decision_Tree, which we'll supply with the loss function we want to use to make node splitting decisions, as well as the estimator we'll use to come up with the prediction associated with each leaf node. For classification, this prediction could be a vector of probabilities, but for simplicity we'll just consider hard classifications here. We'll work with the classification and regression data sets from Homework #4.

1. [Optional] Complete the class Decision_Tree, given in the skeleton code. The intended implementation is as follows: Each object of type Decision_Tree represents a single node of the tree. The depth of that node is represented by the variable self.depth, with the root node having depth 0. The main job of the fit function is to decide, given the data provided, how to split the node or whether it should remain a leaf node. If the node will split, then the splitting feature and splitting value are recorded, and the left and right subtrees are fit on the relevant portions of the data. Thus tree-building is a recursive procedure. We should have as many Decision_Tree objects as there are nodes in the tree. We will not implement pruning here. Some additional details are given in the skeleton code.

2. [Optional] Complete either the compute_entropy or compute_gini functions. Run the code provided that builds trees for the two-dimensional classification data. Include the results. For debugging, you may want to compare results with sklearn's decision tree. For visualization, you'll need to install graphviz.

3. [Optional] Complete the function mean_absolute_deviation_around_median (MAE). Use the code provided to fit the Regression_Tree to the krr dataset using both the MAE loss and median predictions. Include the plots for the 6 fits.

4 Gradient Boosting Implementation

This method goes by many names, including gradient boosting machines (GBM), generalized boosting models (GBM), AnyBoost, and gradient boosted regression trees (GBRT), among others. Although one of the nice aspects of gradient boosting is that it can be applied to any problem with a subdifferentiable loss function, here we'll keep things simple and consider the standard regression setting with square loss.

1. Complete the gradient_boosting class. As the base regression algorithm, you may use sklearn's regression tree. You should use the square loss for the tree splitting rule and the mean function for the leaf prediction rule. Run the code provided to build gradient boosting models on the classification and regression data sets, and include the plots generated. Note that we are using square loss to fit the classification data, as well as the regression data.

2. [Optional] Repeat the previous runs on the classification data set, but use a different classification loss, such as logistic loss or hinge loss. Include the new code and plots of your results. Note that you should still use the same regression tree settings for the base regression algorithm.