CS 189 Introduction to Machine Learning
Fall 2019 Jennifer Listgarten & Stella Yu

HW09

Due: **Optional. No due date.**

1 Random Forest Motivation

Learning Goals: Motivate ensemble learning in random forests.
Ensemble learning is a general technique to combat overfitting, by combining the predictions of many varied models into a single prediction based on their average or majority vote.

(a) **The motivation of averaging.** Consider a set of uncorrelated random variables $\{Y_i\}_{i=1}^n$ with mean μ and variance σ^2. Calculate the expectation and variance of their average. (In the context of ensemble methods, these Y_i are analogous to the prediction made by classifier i.)

(b) **Ensemble Learning – Bagging.** In lecture, we covered bagging (Bootstrap AGGregatING). Bagging is a randomized method for creating many different learners from the same data set.

Given a training set of size n (where n is very large), generate B random subsamples of size n' by sampling with replacement. Some points may be chosen multiple times, while some may not be chosen at all. If $n' = n$, around 63% are chosen, and the remaining 37% are called out-of-bag (OOB) samples.

(a) Show why around 63% are chosen.

(b) If we use bagging to train our model, How should we choose the hyperparameter B? Recall, B is the number of subsamples, and typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set.

(c) In part (a), we see that averaging reduces variance for uncorrelated classifiers. Real world prediction will of course not be completely uncorrelated, but reducing correlation will generally reduce the final variance. Reconsider a set of correlate random variables $\{Z_i\}_{i=1}^n$. Suppose $\forall i \neq j$, $\text{Corr}(Z_i, Z_j) = \rho$. Calculate the variance of their average.

2 Boosted Decision Trees

Learning Goal: Develop the key concepts required to understand boosted decision trees using the AdaBoost algorithm.
We are given data $D = \{(X_i, y_i)\}_{i=1}^N$, where $X_i \in \mathbb{R}^d$ and $y_i \in C = \{-1, 1\}$. Recall that, for a node in a decision tree with data $S \subseteq D$, we compute the proportion of each label, then use those proportions to compute the entropy:

$$p_c = \frac{1}{|S|} \sum_{(X,y) \in S} I(y = c),$$

$$H(S) = \sum_{c \in C} -p_c \ln p_c.$$

(a) Let w_i be the **weight** of observation (X_i, y_i). The weight of an observation can be thought of as its importance. To incorporate weights into our decision tree, we redefine the way we compute proportions. Let $Z = \sum_{(X,y,w) \in S} w$, and

$$p_c = \frac{1}{Z} \sum_{(X,y,w) \in S} I(y = c)\, w.$$

Assume $w_i = a$ for all i. Show that $H(S)$ does not change for constant values of a.

(b) Like Random Forest, boosting is an ensemble method. We train several decision trees on weighted observations and combine their predictions to construct an overall improved classifier. The Adaboost algorithm starts by training the first decision tree G_1 using observation weights initialized to $w_i = \frac{1}{N}$. The weighted error rate of a trained tree G_t is given by

$$\text{err}_t = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_t(X_i))}{\sum_{i=1}^{N} w_i}.$$

Each tree G_t in a boosted ensemble is assigned a weight. The weight is computed using the negative logit function (you will show this is optimal in the lecture on Boosting)

$$\beta_t = \frac{1}{2} \ln\left(\frac{1 - \text{err}_t}{\text{err}_t}\right).$$

What are the minimum and maximum possible values, $\text{err}_{\min}$ and $\text{err}_{\max}$, of err_t so that $\text{err}_{\min} \leq \text{err}_t \leq \text{err}_{\max}$?

(c) After training T decision trees, the AdaBoost algorithm produces the following decision function

$$G(x) = \text{sign}\left[\sum_{t=1}^{T} \beta_t G_t(x)\right],$$

where $\text{sign}[x] = 1$ if $x \geq 0$, and -1 otherwise. Compute $\nabla_{\text{err}_t} \beta_t$. What do you notice about the rate of change of β_t when err_t is near its bounds?

(d) After each decision tree is trained, the weight for each observation $i = 1, ..., N$ is updated by the following update rule:

$$w_i \leftarrow w_i \exp\left(-\beta_t y_i G_t(x_i)\right).$$

Note that $-y_i G_t(x_i) = 2I(y_i \neq G_t x_i) - 1$. Since the -1 becomes a multiplicative constant, we can drop it to obtain

$$w_i \leftarrow w_i \exp\left(2\beta_t I(y_i \neq G_t x_i)\right).$$

The subsequent tree is trained on these updated weights.

(a) Let $w_j^{(i)}$ be the weight on the j^{th} observation at i iterations of the algorithm. What is the value of $w_j^{(1)}$ if observation j is the only observation that has not been classified correctly after 1 iteration?

(b) How does this influence the optimal split choice for nodes in decision tree G_2?

(c) What is $w_j^{(3)}$ if j is still the only observation which has not been classified correctly?

3 Orthogonal Matching Pursuit

Learning Goal: Review OMP from the 11/26 lecture since most of you probably didn't go:) Consider the problem setting, where you are given $\mathbf{X}$ and $\mathbf{y}$ where $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^n$. That is we have n observations that are given by linear combinations of d-features. How can we find the original $\mathbf{w}$ such that $\mathbf{y} = \mathbf{X}\mathbf{w}^*$? We've learned many techniques for this so far, but suppose you have an under-determined system $n \ll d$ and we want a k-sparse solution (a signal $\mathbf{w}$ is called k-sparse if $|\mathbf{w}|_0 = k$, where the zero-norm is defined as the number of non-zero elements of the vector w). In lecture, we have discussed Lasso (ℓ_1-regularized regression) as one of the powerful methods to solve such problems. And furthermore, we saw that coordinate descent can be employed to solve the lasso-penalized regression problem. With coordinate descent, we make progress along a randomly chosen coordinate or feature. A natural question arises: can we choose the coordinates in a smart way? And what if we optimize (rather than taking simply a step) along that coordinate?

Let's review one technique, that attempts a greedy coordinate selection procedure: orthogonal matching pursuit. (You might recall OMP was introduced in Q1 from Discussion 3. Here we present the algorithm in full.) Let us introduce some notation: we use $\mathbf{x}_j$ to denote the j-th column of the matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_d \\ | & | & & |. \end{bmatrix}$$

Note that the vector $\mathbf{x}_j \in \mathbb{R}^n$ denotes the j-th feature for all n data points.

OMP Algorithm:

1. Initialize the residue $\mathbf{r}^0 = \mathbf{y}$ and initialize the index set to be $I^0 = \emptyset$. Set your estimate $\hat{\mathbf{w}}^0 = 0$.
2. Repeat for $t = 1, \dots$ until $\|r^t\|_2 < T$(threshold) OR $t > k$(sparsity budget):
   - **Index update**: Find an index j^t which reduces the residue most. In other words, find the best one-feature linear fit to the existing residual vector, and then update the index set by including the index corresponding to the best feature. Mathematically this update can be written as:

$$\begin{aligned} \mathbf{r}^t &= \mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^{t-1} \\ j^t &= \arg\min_i (\min_v \|\mathbf{r}^t - v\mathbf{x}_i\|_2^2) \\ I^t &= I^{t-1} \cup \{j^t\} \end{aligned}$$

- **Estimate update**: Estimate the best linear fit of the target $\mathbf{y}$ using the features obtained so far. Given that we have found t good features, we now find the best linear fit for the target $\mathbf{y}$ using these t-features. Define $\mathbf{X}_t = \left[\mathbf{x}_{j^1}, \ldots, \mathbf{x}_{j^t}\right]$ made up of these t-features. Then we determine $\hat{\mathbf{w}}^t$ as the solution for the following least-squares problem:

$$\hat{\mathbf{w}}^t = \arg\min_{\mathbf{w}\in\mathbb{R}^t} \|\mathbf{y} - \mathbf{X}_t\mathbf{w}\|_2^2$$

We now discuss under what conditions can we expect such a greedy-coordinate-finding procedure to provide us a good solution.

(a) 1**-sparse noiseless case**: Suppose that the true signal is given by

$$\mathbf{w}^* = \begin{bmatrix} w_1^* \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad w_1^* \neq 0$$

(but we don't know the true signal) and we are given *noiseless* observations

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* = \sum_{i=1}^{d} \mathbf{x}_i w_i^* = \mathbf{x}_1 w_1^*.$$

When will OMP work for such a setting? First consider the case when columns of $\mathbf{X}$ are normalized to have unit norm, that is $\|\mathbf{x}_i\|_2 = 1$. **Is it necessary to have normalized columns for exact recovery in this setting?** Note that since we have one-sparse signal, OMP works correctly if it finds the right coordinate in the first ($t = 1$) step.

Hint: Let's define $\mu = \max_{i\neq j} \frac{|\mathbf{x}_i^\top \mathbf{x}_j|}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}$. What condition on μ do we need for exact recovery?

(b) 1**-sparse noisy case**: For simplicity, let us assume that we have normalized columns and that the observations are corrupted by Gaussian noise. We continue to assume that the true signal is 1-sparse:

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{Z} = \sum_{i=1}^{d} \mathbf{x}_i w_i^* + \mathbf{Z} = \mathbf{x}_1 w_1^* + \mathbf{Z}$$

where $w_1^* \neq 0$ and $\mathbf{Z} \sim \mathcal{N}(0, \sigma^2 I)$ is an n-dimensional Gaussian random vector. **When will OMP recover the true support of the signal?** Note that true signal magnitude can not be recovered exactly due to noise, but here we investigate if we can find the right index using OMP.

Hint: We have to again consider the quantity $\mu = \max_{i\neq j} \frac{|\mathbf{x}_i^\top \mathbf{x}_j|}{\|\|\mathbf{x}_i\|\|\mathbf{x}_j\|}$. For $Z_i \sim \mathcal{N}(0, \sigma^2)$ (not necessarily independent), we have

$$\mathbf{P}\left\{\max_{i\in\{1,2,\ldots,d\}} |Z_i| \geq 2\sigma\sqrt{\log d}\right\} \leq \frac{1}{d}.$$

In other words, we have that the minimum and maximum of d Gaussian random variables with zero mean and σ^2 variance are bounded between $[-2\sigma\sqrt{\log d}, 2\sigma\sqrt{\log d}]$ with high probability (when d is large).

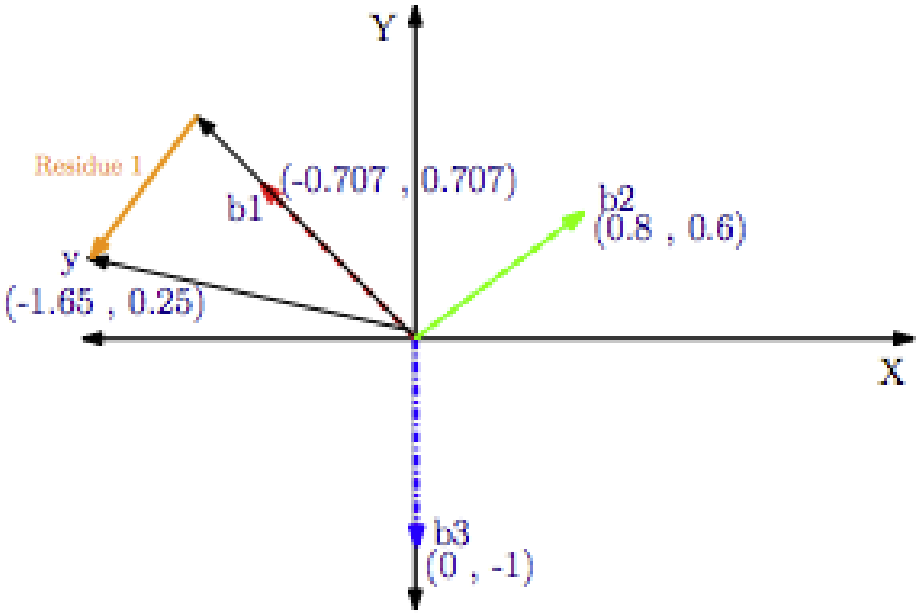

Figure 1: Figure depicting y and residue 1 where b1, b2, and b3 are basis vectors of matrix X.

(c) OMP is also related to the classical idea of Boosting. As you have already seen (or may see in the next few lectures) boosting is a powerful recipe of training a set of weak learners (that do not fit the data very well by themselves) and combining them to find a strong learner (that fits the data well). The general idea is as follows: In the first step, we use one weak learner to fit a given dataset. In the next step, we use another weak learner to improve upon the first learner, by putting more weights on the data points that the first learner was unable to fit well (wrong classification or large squared error in regression). We repeat this process until a desirable accuracy is achieved.

Can you see OMP as an illustration of boosting for regression?