CS 334: Homework #5

Submission Instructions: The homework is due on Nov 18th at 11:59 PM ET. Make sure your program uses Python 3.7. Your submission should consist of two steps (detailed in Homeworks #1, #2). If *either of the two steps are late, then your assignment is late.*

Dataset Description: For this homework, you will be explore the effects of reducing the dimensionality of the data and running random forest and `xgboost` on the wine-quality dataset (from homework 1).

1. **(5+10+10+10=35 pts) PCA & NMF**

   For this problem, we will try to quantify the impact of dimensionality reduction on logistic regression.

   (a) Normalize the features of the wine quality dataset (where applicable). Train an *unregularized* logistic regression model on the normalized dataset and predict the probabilities on the normalized test data.

   (b) Run PCA on the normalized training dataset. How many components were needed to capture at least 95% of the variance in the original data. Discuss what characterizes the first 3 principal components (i.e., which original features are important).

   (c) Run NMF on the original dataset. How many components do you think are appropriate for NMF? Justify your selection of R for NMF.

   (d) Train two *unregularized* logistic regression models using the PCA and NMF dataset and predict the probabilities on the appropriately transformed test data (i.e., for PCA, the test data should be transformed to reflect the loadings on the k principal components). Plot the ROC curves for all 3 models (normalized dataset, PCA dataset, NMF dataset) on the same graph. Discuss your findings from the ROC plot.

2. **(30+10+5=45 pts) Almost Random Forest**

   For this problem, you will be implementing a variant of the random forest using the decision trees from `scikit-learn`. However, instead of subsetting the features for each node of each tree in your forest, you will choose a random subspace that the tree will be created on. In other words, each tree will be built using a *bootstrap sample and random subset of the features.* The template code for the random forest is available in `rf.py`

   (a) Build the adaptation of the random forest. Note that the forest will support the following parameters.

   - nest: the number of trees in the forest
   - maxFeat: the maximum number of features to consider in each tree
   - criterion: the split criterion – either gini or entropy
   - maxDepth: the maximum depth of each tree
   - minSamplesLeaf: the minimum number of samples per leaf node

   Note that you'll need to implement the `train` and `predict` function in the template code.

   - `train`: Given a feature matrix and the labels, learn the random forest using the data. The return value should be the OOB error associated with the trees up to that point. For example, at 5 trees, calculate the random forest predictor by averaging only those trees where the bootstrap sample does not contain the observation.

- `predict`: Given a feature matrix, predict the responses for each sample.

(b) Find the best parameters on the wine quality training dataset based on classification error. Justify your selection with a few plots or tables.

(c) Using your optimal parameters, how well does your version of the random forest perform on the test data? How does this compare to the estimated OOB error?

3. **(15+5=20 points) Gradient Boosting**

For this problem, you will explore the popular `xgboost` package for predicting wine-quality. Follow the instructions on the xgboost website (`https://xgboost.readthedocs.io/en/latest/build.html`) for installing the package. Note that installation can be tricky depending on your Python setup, so do not assume this will be fast!

(a) Find the best parameters on the wine quality training dataset for `xgboot`. The parameters you'll want to consider tuning are `num_rounds,learning_rate,max_depth`. As a hint, you'll want to start learning rate from 0.1 and max depth of 3 and vary it according to your dataset. Justify your selection with a few plots or tables.

(b) Using the optimal parameters, how well does the `xgboost` model perform on the test data? How does this compare to your results from 2(c)?