CS 686-02 - Assignment 5 Spring, 2018

Assignment 5 - Implementation of Random Forest

The goal of this assignment is a) to implement a simple version of the random forest algorithm using an object-oriented framework and b) to test this implementation against an income dataset and c) to understand the implications of research and findings in machine learning.

Background

We have created a class, `decision_tree`, as part of Lab 3, which — unlike most Classification and Regression Trees (CART) — is an n-ary tree. You will extend this class to create an unconventional but working implementation of the Random Forest algorithm. As with our other implementations, the Random Forest implementation will be based on the base class, `classifier`, for our classification algorithms. Not counting the constructor, this base class has two functions: `fit` and `predict`:

- The `fit` function accepts a list of (training) features and their labels as parameters, changing the internal state of the classifier as needed.
- The `predict` function accepts a list of (test) features and outputs the classifier's hypothesis, in order, about the class membership of the instance represented by the features.

The Random Forest algorithm is one such classification approach which is generally useful. The `fit` function creates a number of decision trees on randomly-generated subsets of the data and randomly-selected features from those available during training. The `predict` function hypothesizes the label of each item in the input by determining the most populous label from among the trained trees.

Requirements

You will complete this assignment by performing the following three (3) requirements:

1. Implement "`randomforest.py`", a class in object-oriented python. Your randomforest.py must be a subclass of classifier, https://github.com/dbrizan/cs686-2018-01/blob/master/classifier.py. Specifically, it must have three functions at minimum: `__init__(self, trees=10, max_depth=-1)`, `fit(self, X, Y)` and `predict(self, X)`... where "`self`" refers to the class instance, "`X`" refers to the list of feature vectors in train or test and "`Y`" refers to a list of labels associated with each feature vector. It must use one or more instances of `decision_tree` from Lab 3[1]. In the `__init__` function, there are two optional parameters:
    a. "`trees`" refers to the number of decision trees used by the implementation.
    b. "`max_depth`" refers to the depth of each tree, with "-1" being "unlimited.

[1] Speak with the instructor if you do not have a working decision tree.

You may use the “cut” function in pandas to “bin” continuous variables, such as age, into discrete ranges. Continuing with “age” as an example:

```
age = df.iloc[:,0]
pd.cut(age, 8)
```

… will generate 8 bins for a continuous “age” range. You may use any number of bins (greater than 2) for your implementation. The output of this function is a list of tuples, such as the following items, the first 5 from the training file:

```
0     (44.375, 53.5]
1    (35.25, 44.375]
2     (44.375, 53.5]
3    (26.125, 35.25]
4    (35.25, 44.375]
```

2. Test your “`randomforest.py`” implementation by running it against the Adult Data Set dat, located at https://archive.ics.uci.edu/ml/datasets/adult. This data contains a label for each adult (<=50K or >50K) indicating the income of that person. There are 14 features for each instance of the data. Your implementation must use the adult.data file for training (i.e. `fit()`) and adult.test for testing (i.e. `predict()`). Other than `randomforest.py`, any modules other need not be object oriented, but they are expected to conform to reasonable guidelines of structured programming and style, including variable naming and code documenting. See “PEP 8: Style Guide for Python Code” for one guide on how to do so.
3. Determine and output the accuracy of your classifier implementation, `randomforest.py`, when trained on the training set, defined above and tested on the testing set[2].

Submission

You are required to submit two items:

1. Your source code on Canvas. (While you are free and encouraged to store your implementation on github, you may not submit your github repo.)
2. The performance of your model.

Grading

The requirements are serially dependent — i.e. completing the third item depends on completion of the second, which depends on completion of the first. You are required and expected to complete all items above perfectly. As such, your grade starts as 100%, and the penalties for errors are listed below. Your grade may be affected by one or more penalty.

- -10% = Implementation lacks reasonable style for code or documentation.
- -15% = Implementation works but contains minor errors (eg. does not contain performance).
- -40% = Implementation is missing one required item.
- -100% = Implementation not attempted or not submitted on time.

[2] Your grade for this assignment does not depend on the performance of your model because of the many possible differences in implementation. However, you are required to determine and report an accuracy score.