CS57300: Assignment 4

Due date: Sunday March 31, 11:59 pm (submit via turnin)

Comparing Methods for Speed Dating Classification

In this programming assignment, you will be asked to implement Decision Trees, Bagging and Random Forests for the classification task that you explored in Assignments 2 and 3, and then compare the performance of different classifiers.

You should implement your solution using Python. You can use supporting libraries like numpy, scipy as before, but DO NOT use any publicly available code including but not limited to libraries such as **sklearn**. As before, you should submit your typed assignment report as a pdf along with your source code file.

In the following sections, we specify a number of steps you are asked to complete for this assignment. Note that all results in sample outputs are fictitious and for representation only.

1 Preprocessing

Consider the data file **dating-full.csv** that you used in Assignment 2. For this assignment, we will only consider the first 6500 speed dating events in this file. That is, you can discard the last 244 lines of the file. Write a Python script named **preprocess-assg4.py** that reads the first 6500 speed dating events in **dating-full.csv** as input and performs the following operations.

(i) For simplicity, drop the columns **race**, **race_o** and **field**.

(ii) For the categorical attribute **gender**, apply label encoding, as in Assignment 2. Then, repeat the preprocessing steps 1(iv) that you did in Assignment 2. (You can reuse the code there and you are not required to print any outputs.)

(iii) Discretize all the continuous-valued columns using 2 bins of equal-width, so that all the features you will use become binary. (You should use the cut function in **pandas** with number of bins as 2 and labels = [0, 1] to do so.)

(iv) Use the **sample** function from **pandas** with the parameters initialized as **random_state = 47, frac = 0.2** to take a random 20% sample from the entire dataset. This sample will serve as your test dataset, which you should output in **testSet.csv**; the rest will be your training dataset, which you should output in **trainingSet.csv**. (Note: The use of the random_state will ensure all students have the same training and test datasets; incorrect or no initialization of this parameter will lead to non-reproducible results).

2 Implement Decision Trees, Bagging and Random Forests (10 points)

Please put your code for this question in a file called **trees.py**. This script should take three arguments as input:

1. *trainingDataFilename*: the set of data that will be used to train your algorithms (e.g., **trainingSet.csv**).

2. *testDataFilename*: the set of data that will be used to test your algorithms (e.g., **testSet.csv**).

3. *modelIdx*: an integer to specify the model to use for classification (DT = 1, BT = 2, RF = 3, where DT refers to decision trees, BT refers to bagging, and RF refers to random forests).

(i) Write a function named ***decisionTree(trainingSet, testSet)*** that takes the training dataset and the testing dataset as input parameters. The purpose of this function is to train a decision tree classifier using the data in the training dataset, and then test the classifier's performance on the testing dataset.
Use Gini-gain as your feature selection criteria. Grow trees using a depth limit of 8 and an example limit of 50 (i.e., stop growing when either the depth of the tree reaches 8 or the number of examples in a node is smaller than 50).

(ii) Write a function named ***bagging(trainingSet, testSet)*** that takes the training dataset and the testing dataset as input parameters. The purpose of this function is to train a bagged decision tree classifier using the data in the training dataset, and then test the classifier's performance on the testing dataset.
Learn 30 trees, and the stopping criterion for growing each tree is the same as that in (i). Use sampling with replacement to construct pseudosamples (i.e., bootstrapped sample of the training data).

(iii) Write a function named ***randomForests(trainingSet, testSet)*** that takes the training dataset and the testing dataset as input parameters. The purpose of this function is to train a random forests classifier using the data in the training dataset, and then test the classifier's performance on the testing dataset.
Learn 30 trees, and the stopping criterion for growing each tree is the same as that in (i). Use sampling with replacement to construct pseudosamples, use $\sqrt{p}$ to downsample the features at each node of the tree (where p is the total number of features).

The sample inputs and outputs we expect to see are as follows (the numbers are fictitious):

```
$python trees.py trainingSet.csv testSet.csv 1
Training Accuracy DT: 0.71
Testing Accuracy DT: 0.68

$python trees.py trainingSet.csv testSet.csv 2
Training Accuracy BT: 0.75
Testing Accuracy BT: 0.74

$python trees.py trainingSet.csv testSet.csv 3
Training Accuracy RF: 0.73
Testing Accuracy RF: 0.77
```

3 The Influence of Tree Depth on Classifier Performance (10 points)

Please follow the procedure below to assess whether the depth of the tree affects classifier performance.

Use the **sample** function from **pandas** with the parameters initialized as **random_state = 18, frac = 1** to shuffle the training data (i.e., data in **trainingSet.csv**). Then, obtain a 50% sample of the above shuffled training data using **random_state = 32**.

Perform 10-fold cross validation on this sample of training data (consider the first 10% lines of the sampled data as your first fold, the second 10% lines of the sampled data as your second fold, and so on; notice you are asked to conduct cross validation on a *sample* of the training data to reduce the time cost of this assignment). Conduct the cross validation for each of the three models—decision tree, bagged trees, and random forests—where depth limit of the trees in each model is set to be $d \in [3, 5, 7, 9]$. The example limit of the trees is fixed at 50. Learn 30 trees for the ensemble models.

(a) Plot the average accuracy for 10-fold cross validation on y-axis, and depth limit of tree on x-axis. Include error bars that indicate ± 1 standard error. Please include the curves for the three models in one figure.

(b) Formulate a hypothesis about the performance difference you observe as the depth limit of trees change. Discuss whether the observed data support the hypothesis or not (i.e., are the observed differences significant?).

4 Compare Performance of Different Models (10 points)

Please follow the procedure below to assess whether ensemble methods improve performance.

Use the **sample** function from **pandas** with the parameters initialized as **random_state = 18, frac = 1** to shuffle the training data (i.e., data in **trainingSet.csv**). Conduct incremental cross validation, as described in Assignment 3, for the three models.

Specifically, please first divide the shuffled training data into 10 folds (i.e., the first 10% lines is your first fold, the second 10% lines is your second fold, and so on). Then, use fractions $t\_frac \in [0.05, 0.075, 0.1, 0.15, 0.2]$ with **random_state=32** to obtain *train_set* in each iteration of cross validation. The depth limit of tree is 8, and example limit is 50. Learn 30 trees for the ensemble models

(a) Plot the learning curves for the three models (in the same plot), with the average accuracy of the 10 trials on y-axis, and the training fraction on x-axis. Include error bars that indicate ± 1 standard error, from the evaluation based on the incremental cross validation results (as in Assignment 3, Q3(iii)).

(b) Formulate a hypothesis about the performance difference you observe between the decision tree and one of the ensemble methods. Discuss whether the observed data support the hypothesis or not (i.e., are the observed differences significant?).

5 The Influence of Number of Trees on Classifier Performance (10 points)

Please follow the procedure below to assess whether the number of trees affects performance.

Use the **sample** function from **pandas** with the parameters initialized as **random_state = 18, frac = 1** to shuffle the training data (i.e., data in **trainingSet.csv**). Then, obtain a 50% sample of the above shuffled training data using **random_state=32**.

Perform 10-fold cross validation on this sample of training data (consider the first 10% lines of the sampled data as your first fold, the second 10% lines of the sampled data as your second fold, and so on; notice you are asked to conduct cross validation on a *sample* of the training data to reduce the time cost of this assignment). Conduct the cross validation for each of the two ensemble methods, —bagged trees and random forests—where the number of trees in each model is set to be $t \in [10, 20, 40, 50]$. The depth limit of tree is 8, and example limit is 50.

(a) Plot the average accuracy for 10-fold cross validation on y-axis, and number of trees on x-axis. Include error bars that indicate ± 1 standard error. Please include the curves for the three models in one figure.

(b) Formulate a hypothesis about the performance difference you observe as the number of trees changes. Discuss whether the observed data support the hypothesis or not (i.e., are the observed differences significant?).

Bonus question (5 points)

Implement a suitable model of your choice that has not been included in Assignments 2, 3, or 4, (e.g., boosted decision trees, neural networks, etc.), along with the optimal set of hyper-parameters, that gives highest possible accuracy on the testing dataset (i.e., **testSet.csv**) Recall that you should not touch the testing dataset until you are satisfied with your model. Report your tuning procedure, the hyper-parameters you end up with, your model selection procedure, training and testing procedures, and the level of accuracy you get on the testing dataset. Note that you have to implement the complete model without using any available softwares such as Weka, or libraries like sklearn.

Submission Instructions:

After logging into data.cs.purdue.edu, please follow these steps to submit your assignment:

1. Make a directory named *yourFirstName_yourLastName* and copy all of your files to this directory.

2. While in the upper level directory (if the files are in /homes/yin/ming_yin, go to /homes/yin), execute the following command:

   turnin -c cs573 -p HW4 your_folder_name

   (e.g. your professor would use: turnin -c cs573 -p HW4 ming_yin to submit her work)
   Keep in mind that old submissions are overwritten with new ones whenever you execute this command.

   You can verify the contents of your submission by executing the following command:

   turnin -v -c cs573 -p HW4

   Do not forget the -v flag here, as otherwise your submission would be replaced with an empty one.

Your submission should include the following files:

1. The source code in python.

2. Your evaluation & analysis in .pdf format. Note that your analysis should include visualization plots as well as a discussion of results, as described in details in the questions above. The results obtained for all the questions must be mentioned in the report.

3. A README file containing your name, instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc).