CS 422/622 Project 1

Due 9/27 11:59pm

Logistics: You must implement everything stated in this project description that is marked with an **implement** tag. Whenever you see the **write-up** tag, that is something that must be addressed in the project report. Graduate students are required to implement everything including items tagged with **622**. Students in 422 do not need to complete these extra elements. **You cannot import any packages unless specifically noted by the instructor.** In this project, you may import the numpy and math (for log) packages. You are welcome to ask about particular packages as they come up. The idea is to implement everything from scratch.

Deliverables: Each student should submit a single ZIP file, containing their project code (∗.py files) and your report (README.txt or Latex file). You should create a folder called Project1 (exactly like this with the capital P) and put all your code and your write-up in this folder. Then zip this folder up. That way when I extract your folder I can run my tests from outside the directory without having to change anything. Your zip file should be named lastname_firstname_project1.zip. Your code should run without errors in a Linux environment. If your code does not run for a particular problem, you will lose 50% on that problem. You should submit two py files, named accordingly (decision_trees.py and data_storage.py). If your file does not match the filenames provided exactly, you will lose 50% on that problem.

Grading: I have provided you with two test scripts (test_script_dt.py and test_script_rf.py) you can use to test out your functions. test_script_dt.py is used for testing the decision tree functions, while test_script_rf.py is used for testing the random forest functions. Both function definitions for the decision tree and the random forest should be implemented in the decision_trees.py file you will be turning in. I will be using a similar test script, so if your code works with this script it should work with mine! The output of the test script for the decision tree should look somewhat like this if you have implemented everything correctly. Each student must work independently. You are to submit your own original work.

```
<class 'numpy.ndarray'>
[[1.  1.  1.  0.  1.]
[0.  0.  0.  0.  0.]
[0.  0.  1.  0.  1.]
[0.  0.  1.  1.  0.]
[1.  0.  1.  1.  0.]
[0.  0.  1.  0.  0.]]
[1 1 1 0 0 0]

<class 'list'>
[1.0, 1.0, 1.0, 0.0, 1.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 1.0, 0.0, 1.0]
[0.0, 0.0, 1.0, 1.0, 0.0]
[1.0, 0.0, 1.0, 1.0, 0.0]
[0.0, 0.0, 1.0, 0.0, 0.0]
[1, 1, 1, 0, 0, 0]

<class 'dict'>
{'pointy_ears':  1.0, 'long_snout':  1.0, 'long_tail':  1.0, 'retract_claws':  0.0, 'howl':  1.0}
{'pointy_ears':  0.0, 'long_snout':  0.0, 'long_tail':  0.0, 'retract_claws':  0.0, 'howl':  0.0}
{'pointy_ears':  0.0, 'long_snout':  0.0, 'long_tail':  1.0, 'retract_claws':  0.0, 'howl':  1.0}
{'pointy_ears':  0.0, 'long_snout':  0.0, 'long_tail':  1.0, 'retract_claws':  1.0, 'howl':  0.0}
{'pointy_ears':  1.0, 'long_snout':  0.0, 'long_tail':  1.0, 'retract_claws':  1.0, 'howl':  0.0}
{'pointy_ears':  0.0, 'long_snout':  0.0, 'long_tail':  1.0, 'retract_claws':  0.0, 'howl':  0.0}
{0:  1, 1:  1, 2:  1, 3:  0, 4:  0, 5:  0}

DT: 1.0
```

The output for the random forest test script should look similar (does not have to be exact) to this instead.

```
DT 0 :  0.73529
DT 1 :  0.65686
DT 2 :  0.65686
DT 3 :  0.73529
DT 4 :  0.70588
DT 5 :  0.56862
DT 6 :  0.58496
DT 7 :  0.70915
DT 8 :  0.58496
DT 9 :  0.73529
DT 10 :  0.56862
RF : 0.73856
```

1 Python Data Storage Methods (15 Points)

File name: **data_storage.py**

Implement a function in python:

`build_nparray(data)`

That takes a 2D array of string values. The goal is to take the values populating the feature vectors and convert them to a 2D numpy array of data type float. The label data points will be turned into their own 1D numpy array of data type int. The header values should be skipped. The function will return these two arrays as the training feature data and the training label data. For further reading on numpy arrays go to https://www.geeksforgeeks.org/basics-of-numpy-arrays/.

Implement a function in python:

`build_list(data)`

That takes a 2D array of string values. Convert the feature vector string variables into a 2D list of lists with data type float. Convert the label data values into a 1D list of data type int. Return both lists as the training feature data and the training label data. For further reading on lists go to https://www.geeksforgeeks.org/python-lists/.

Implement a function in python:

`build_dict(data)`

That takes a 2D array of string values. The feature variables should be converted to data type float, and each data point in the sample (each row of the 2D array) should be created with a key value pair where the key is the corresponding feature from the csv file header. For example, a single entry in the dictionary would read like {feature_1: 1}. The dictionary for the labels should have the keys for each label be their index position in the array. For further reading on lists go to https://www.geeksforgeeks.org/python-dictionary/.

2 Decision Trees (50 Points)

File name: **decision_trees.py**

Implement a function in python:

`DT_train_binary(X,Y,max_depth)`

that takes training data as input. The labels and the features are binary, but the feature vectors can be of any finite dimension. The training feature data (**`X`**) can be structured as a 2D numpy array, with each row corresponding to a single sample. The training labels (**`Y`**) can be structured as a 1D numpy array, with each element corresponding to a single label. **`Y`** should have the same number of elements as **`X`** has rows. **`max_depth`** is an integer that indicates the maximum depth for the resulting decision tree. **`DT_train_binary(X,Y,max_depth)`** should return the decision tree generated using information gain, limited by some given maximum depth. If **`max_depth`** is set to -1 then learning only stops when we run out of features or our information gain is 0. You may store a decision tree however you would like, i.e. a list of lists, a class, a dictionary, etc. Bulleted lists are okay! Binary data for testing can be found in in data_1.csv, and cat_dog_data.csv

Implement a function in python:

`DT_test_binary(X,Y,DT)`

that takes test data **`X`** and test labels **`Y`** and a learned decision tree model **`DT`**, and returns the accuracy (from 0 to 1) on the test data using the decision tree for predictions.

Implement a function in python:

`DT_make_prediction(x,DT)`

This function should take a single sample and a trained decision tree and return a single classification. The output should be a scalar value.

622 Implement a function in python:

`DT_train_real(X,Y,max_depth)`
`DT_test_real(X,Y,DT)`

These functions are defined similarly to those above except that the features are now real values. The labels are still binary. Your decision tree will need to use questions with inequalities: $>$, $\geq$, $<$, $\leq$. Real-valued data for testing is provided in data_2.csv

Write-up: Did you implement your decision tree functions iteratively or recursively? Which data structure did you choose and were you happy with that choice? If you were unhappy with that choice which other data structure would have built the model with?

3 Random Forests (25 Points)

Explanation: Random forests are the more practical application of decision trees in the real world. Random forests are an ensemble learning method that uses a voting system to determine the most appropriate prediction. They generate a number of decision trees where each is individually trained, before passing the prediction sample to all the trees. Whichever prediction value generates the most 'votes' is the selected final prediction. For reading on the explanations of what random forests are in more detail, https://www.ibm.com/cloud/learn/random-forest. For more reading on example applications of random forests,
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Note, you are still expected to implement the functions from scratch.

File name: add the functions to the **decision_trees.py**

Implement a function in python:

`RF_build_random_forest(X,Y,max_depth,num_of_trees)`

that takes training data (**`X`**) and training labels (**`Y`**). **`max_depth`** is used to define how many layers deep each individual tree will be allowed to build towards, while **`num_of_trees`** determines the total number of trees that are built within the forest. You are allowed to test with a different number of trees, but the project will be graded with the assumption the model is being built with 11 trees. Each tree's training data will be built from a random sampling of 10% of the total data found in the provided file (approximately 30 samples). Each tree should output its individual accuracy. If done correctly, there should be decent variance in the accuracy of the trees. The function should return a data structure (array, list, dict) of all the individually trained trees. The data for the random forest function will come from haberman.csv.

Write-up: Why might the individual trees have such variance in their accuracy? How would you reduce this variance and potentially improve accuracy?

Implement a function in python:

`RF_test_random_forest(X,Y,RF)`

that takes training data (**`X`**) and training labels (**`Y`**). **`RF`** is the fully generated and trained random forest. To test the forest, all eleven trees should run the prediction function on an individual sample. Each tree will output a prediction, and the prediction with the majority is the final prediction for the forest. This should be done for all samples to determine the accuracy.

Write-up: Why is it beneficial for the random forest to use an odd number of individual trees?

4 Report (10 Points)

Write-up: Overall, if you are still feeling uncomfortable working with python, what aspect of the coding language do you feel you are struggling with the most? If you do feel comfortable, what part of python do you feel you should continue practicing?

622 is required to use Latex for their report write-up. 422 can create a general README.txt but will be awarded extra-credit for using Latex. https://www.overleaf.com/ is an excellent tool for learning and creating Latex documents. If using Latex for the project report name the file Project1.pdf