CS 6220 Big Data Homework 4 Report – Problem 1.2

By: Nipun Chhajed

Problem 1.2 – The Power of Random Forest

All screenshots in this report are taken from my Jupyter notebook, and all the outputs exist in there as well (it was submitted alongside this report in the zip file).

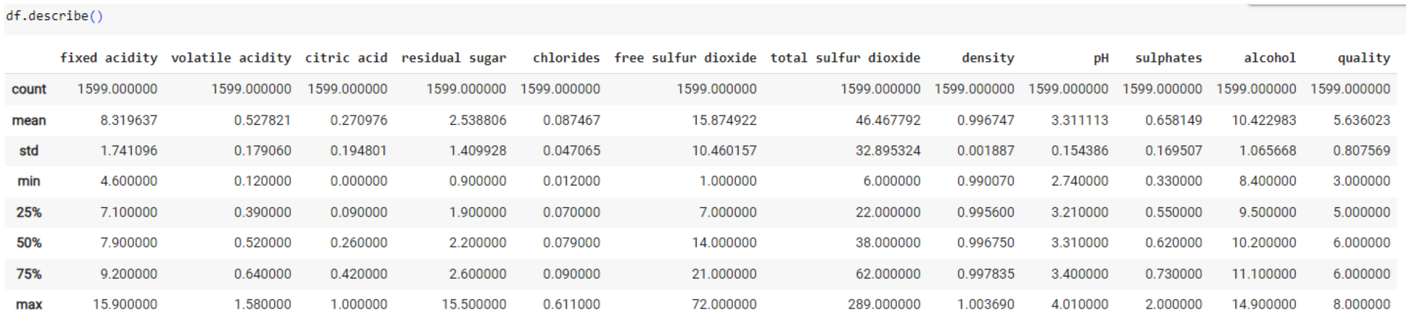
Or you can access the Google Colab notebook here (as that is where I wrote and ran the code): https://colab.research.google.com/drive/1Ofc_gQQZwRD-pFYEXiPDuDwQtOml3JAX?usp=sharing

Instructions to run the code are in the README.md file.

Dataset Introduction:

The dataset I used was the red wine quality dataset from Kaggle: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009/data

Here are the Dataset characteristics:

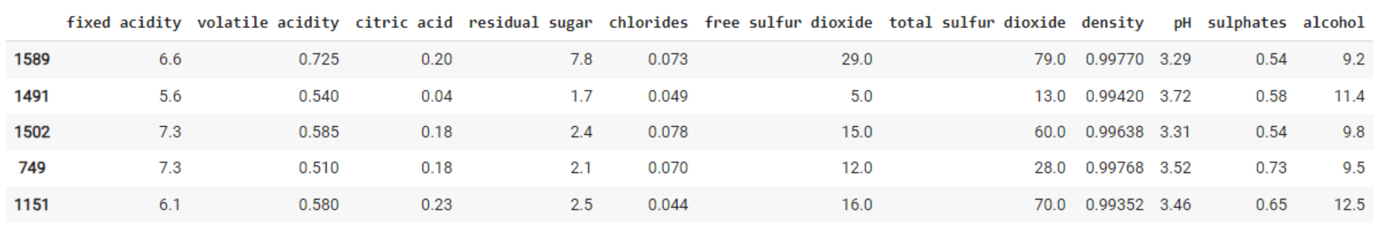


```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

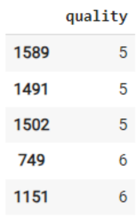
The "quality" column is the y-values we are trying to predict. The range for the quality values is [3, 8] inclusive.

After splitting the dataset into train and test, where 70% of the data was used for training and 30% of the data was used for testing, here is an example of 5 training datapoints:



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
1589	6.6	0.725	0.20	7.8	0.073	29.0	79.0	0.99770	3.29	0.54	9.2
1491	5.6	0.540	0.04	1.7	0.049	5.0	13.0	0.99420	3.72	0.58	11.4
1502	7.3	0.585	0.18	2.4	0.078	15.0	60.0	0.99638	3.31	0.54	9.8
749	7.3	0.510	0.18	2.1	0.070	12.0	28.0	0.99768	3.52	0.73	9.5
1151	6.1	0.580	0.23	2.5	0.044	16.0	70.0	0.99352	3.46	0.65	12.5

The corresponding quality values for these datapoints (y-values):



	quality
1589	5
1491	5
1502	5
749	6
1151	6

<u>**Code and Decision Tree and Random Forest Introduction:**</u>

The code I used for this project was from the following two Kaggle notebooks:

- https://www.kaggle.com/code/imprime/decision-tree-and-random-forest
- https://www.kaggle.com/code/udita3996/eda-logistic-regression-decision-tree

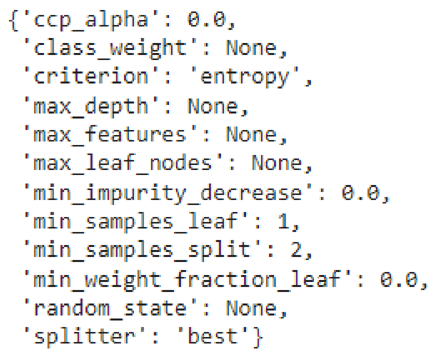
In my Jupyter notebook, I have cited everywhere (in each cell) that I used code from these two notebooks.

I also wrote my own code, for calculating the top 10 and bottom 10 trees in the Jupyter notebook section called ***Top 10/Bottom 10 Tree Analysis***, and for running the experiments for random forests in the Jupyter notebook section called ***Random Forest Experiment (changing number of trees)***.

For both the Decision Tree and Random Forest, I utilized the Scikit-Learn Python Library (sklearn) implementations:

- Random Forest Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Decision Tree Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

<u>After testing hyperparameters for the Decision Tree, I created a tree with the following hyperparameters:</u>

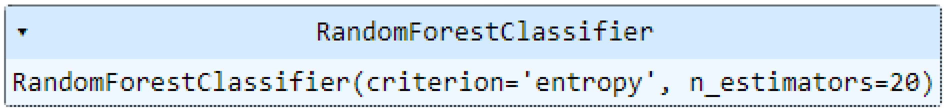


```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

The main thing is, I changed the ‘criterion’ to be ‘entropy’ and the ‘splitter’ to be ‘best’.

The criterion determines the importance of each feature of the dataset (the columns of the dataset) in correlation to the y-values of the dataset, which in this case, is quality. I chose to calculate the entropy value of each feature and use that as the criterion. The tree then splits on the feature with the highest criterion value.

For the Random Forest, I had the following hyperparameters:



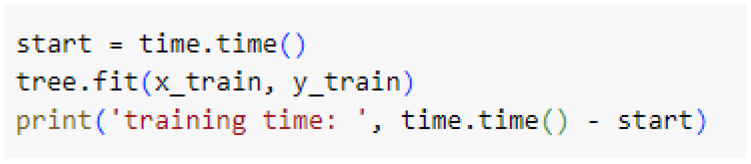
```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=20)
```

I essentially used the same hyperparameters as the Decision Tree, (‘criterion’ = ‘entropy’, ‘splitter’ = ‘best’), as a Random Forest is an ensemble of Decision Trees. However, I started with just 20 trees as the number of trees (‘n_estimators’) I wanted in my forest, as we needed to compute the Top 10 and Bottom 10 best performing trees in the forest, so a minimum of 20 trees was required.

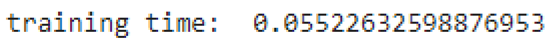
Test Accuracy and Training Time for Decision Tree and Random Forest:

Decision Tree:

- Training Time:

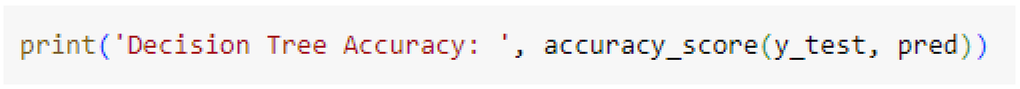


```
start = time.time()
tree.fit(x_train, y_train)
print('training time: ', time.time() - start)
```

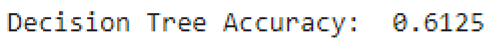
  - 

```
training time:  0.05522632598876953
```

- Test Accuracy:



```
print('Decision Tree Accuracy: ', accuracy_score(y_test, pred))
```

  - 

```
Decision Tree Accuracy:  0.6125
```

Random Forest:

- Training Time:

```
start = time.time()
rfc.fit(x_train, y_train)
print('training time: ', time.time() - start)
```

-
  ```
  training time:  0.10424160957336426
  ```

- Test Accuracy:

  ```
  print('Random Forest Accuracy: ', accuracy_score(y_test, rfc_pred))
  ```

  -
    ```
    Random Forest Accuracy:  0.6625
    ```

As expected, the training time for the random forest was higher than the training time for the single decision tree, by a factor of roughly 1.88.

And the accuracy for the random forest was also higher by 5%: 66.25% for the Random Forest vs 61.25% for the Decision Tree.

<u>**Top 10 and Bottom 10 Tree Accuracy (Within Random Forest):**</u>

Here are the top 10 and bottom 10 trees accuracies within my Random Forest:

	Top 10 Trees Accuracies	Bottom 10 Trees Accuracies
0	0.635417	0.604167
1	0.633333	0.604167
2	0.627083	0.604167
3	0.622917	0.600000
4	0.620833	0.597917
5	0.620833	0.597917
6	0.620833	0.595833
7	0.618750	0.593750
8	0.616667	0.593750
9	0.616667	0.589583

I generated a pandas data frame to print out the results, the code I wrote for this can be found in function ***getTopBottom10()*** within the notebook.

To obtain these accuracies, I had to individually train each tree on the training dataset, and then predict the test values for each of those trees. This is why the highest accuracy is lower than the accuracy of the random forest itself.

We can see that the best tree had an accuracy of 63.54% on the test dataset, which beat my initial decision tree, while the worst tree had an accuracy of 58.95%.

Analysis of why Random Forest Outperforms Decision Tree:

I believe that a random forest ensemble outperforms a single decision tree mainly because it can generalize better, reduce overfitting, and it is less sensitive to noise within the data.

Random Forests can generalize better because it is aggregating multiple different decision trees into one model, where each decision tree learns different aspects and patterns within the data. In addition, if one tree overfits/underfits the data, it will be unlikely for other trees to do the same, and thus since the random forest aggregates all the trees outputs together (by taking the average for example), it will essentially reduce the impact of trees that are overfitting/underfitting, leading to better predictions overall. This is because the predictions are more stable and accurate, as it considers the bias and variance of each individual tree, and aggregates it into one prediction, leading to a stable overall prediction.

Random Forests also reduce overfitting for the same reason as described above, as the aggregation of multiple individual trees allows for trees that are overfitting to have less of an effect on the overall prediction of the forest, as it is counteracted by trees that aren't overfitting the data.

The same logic can also be applied to noise within a dataset. Aggregation of multiple trees, each of which react differently to the noise, leads to random forests being less affected by it, as certain trees are more affected by noise, while others aren't.

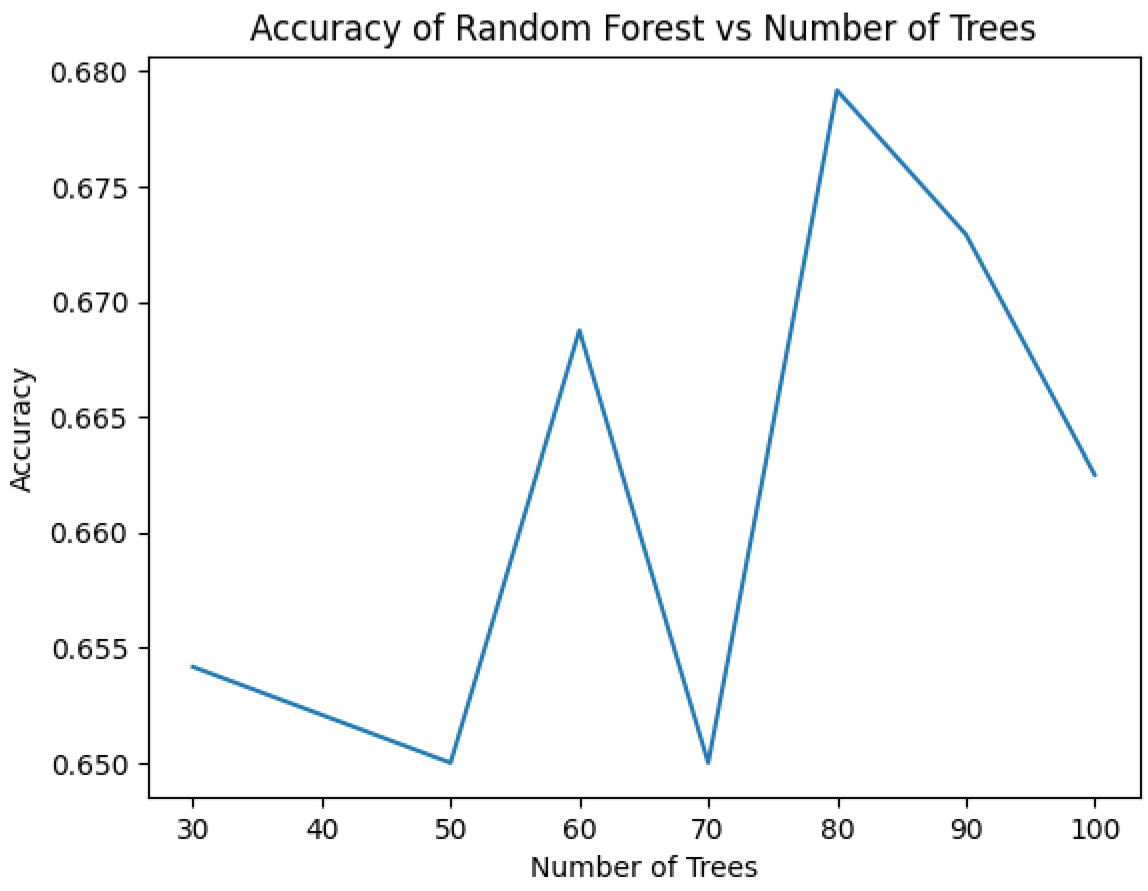
Essentially, since a random forest is a collection of individual decision trees, each decision tree can learn different patterns in the data, and when you combine all those decision trees outputs together, you create a model that has a much better understanding of all the different possible patterns that exist within the data, thus leading to a better overall performance.

Adding More Trees to Random Forest Experiment:

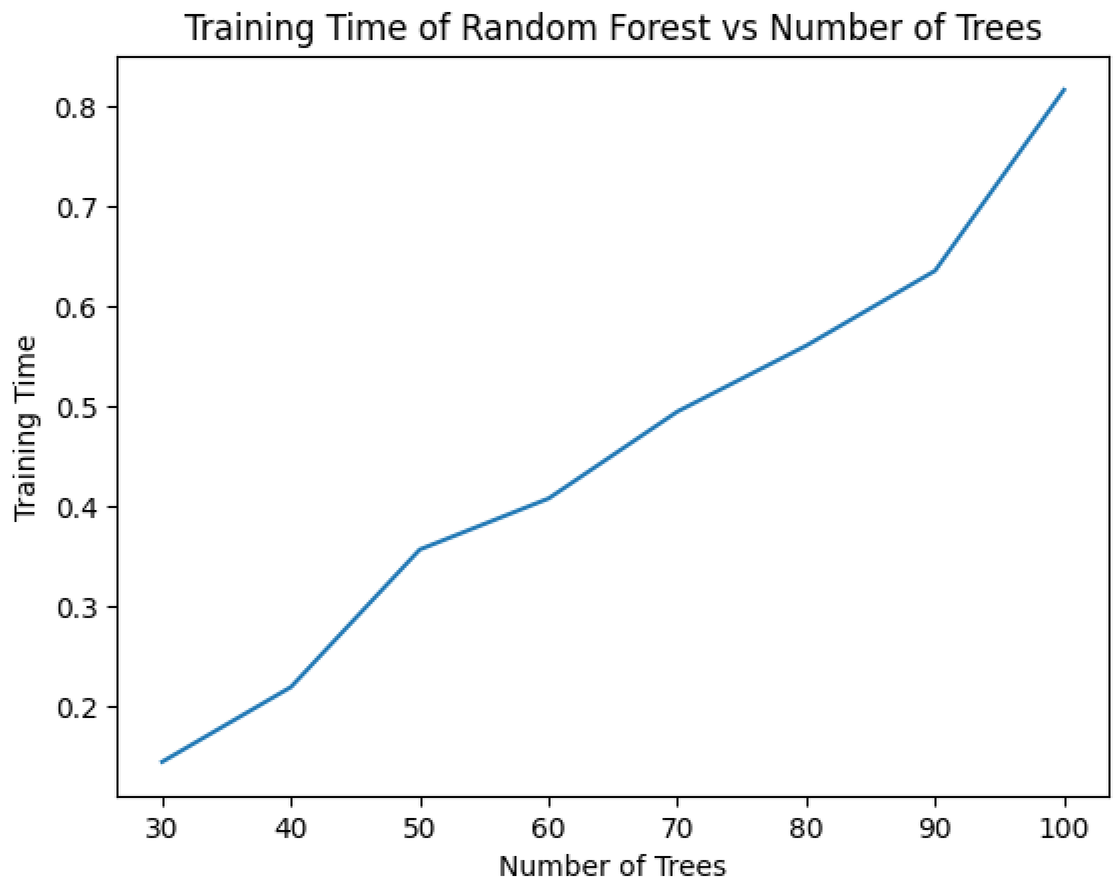
For this experiment, I essentially ran multiple random tree classifiers, each with varying number of trees within the classifier. I recorded and plotted the accuracy and training time of each classifier.

The number of trees for each classifier ranged from 30 to 100, and it was increments of 10, so the classifiers had 30 trees, 40 trees, 50 trees, etc. all the way up to 100 trees.

Here is the plot for Accuracy vs Number of Trees:



Here is the plot for Training Time (sec) vs Number of Trees:



So first off, as we can see from the data above, as the number of trees increases, the training time for the trees increases as well, with a roughly linear relationship.

<u>Now, does adding trees improve random forest accuracy?</u>

In the case of this wine quality dataset, not necessarily, as can be seen from the Accuracy vs Number of Trees graph, there is no overarching relationship that shows that increasing the number of trees improves the random forest classifier. As can be seen from the graph, the accuracy decreases in some places as the number of trees increases, for example from 30 trees to 50 trees, and from 80 trees to 100 trees. In this case, we can see that the maximum accuracy is reached with a random forest classifier made up of 80 trees. This 80-tree random forest had an accuracy of 68%.

After conducting some research, I found that in theory, adding more trees to a random forest should increase the accuracy of the random forest, however, that is rarely the case in actual experimentation. In addition, there is also a point of diminishing returns, which is a specific number of trees where adding more trees beyond that point won’t improve the random forest by much.

References:

[1] https://www.quora.com/Should-prediction-accuracy-go-up-if-more-trees-are-included-in-a-random-forest

[2] https://towardsdatascience.com/decision-tree-and-random-forest-explained-8d20ddabc9dd#:~:text=There%20is%20an%20additional%20parameter,do%20not%20improve%20the%20model.