CS 334 — Homework 5 Solutions

Jack Anstey

Problem 1: PCA

a) See pca.py

b) Using the normalized dataset, my code returned that 9 components were needed to capture at least 95% of the original data as seen in this print output:

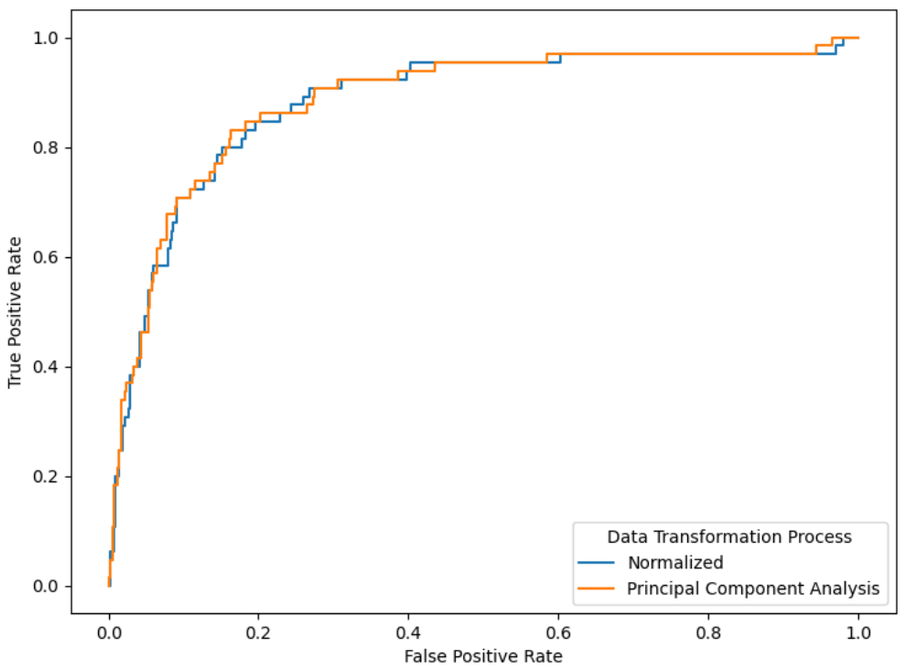
```
Number of components needed for 0.95 or greater variance: 9
```

Following this, I was able to get the values of the first 3 principal components and take their mean. The values of each are as follows:

```
Using the first 3 principal components, these are the average weights for each attribute:
total sulfur dioxide     0.307864
free sulfur dioxide      0.305687
citric acid              0.182892
sulphates                0.168425
residual sugar           0.144624
density                  0.101513
fixed acidity            0.091311
chlorides                0.083633
alcohol                 -0.019581
pH                      -0.124752
volatile acidity        -0.144197
```

Since these are all weights, things that are weighted higher are more important and those that have low or negative weights are less important. Using a threshold of 0.1, we can say that Total Sulfur Dioxide, Free Sulfur Dioxide, Citric Acid, Sulphates, Residual Sugar, and Density are all important features for this wine dataset.

c) The ROC curves when using the normalized dataset and the PCA dataset are as follows:



The differences between the Normalized and PCA ROC curves are minimal, which is great! This means that when we use PCA, not only do we gain the benefit of a good ROC curve with minimal loss, but we also gain the benefits of PCA which is namely dimension reduction. Our models can be both fast (especially when doing something like clustering) and accurate.

Problem 2: Almost Random Forest

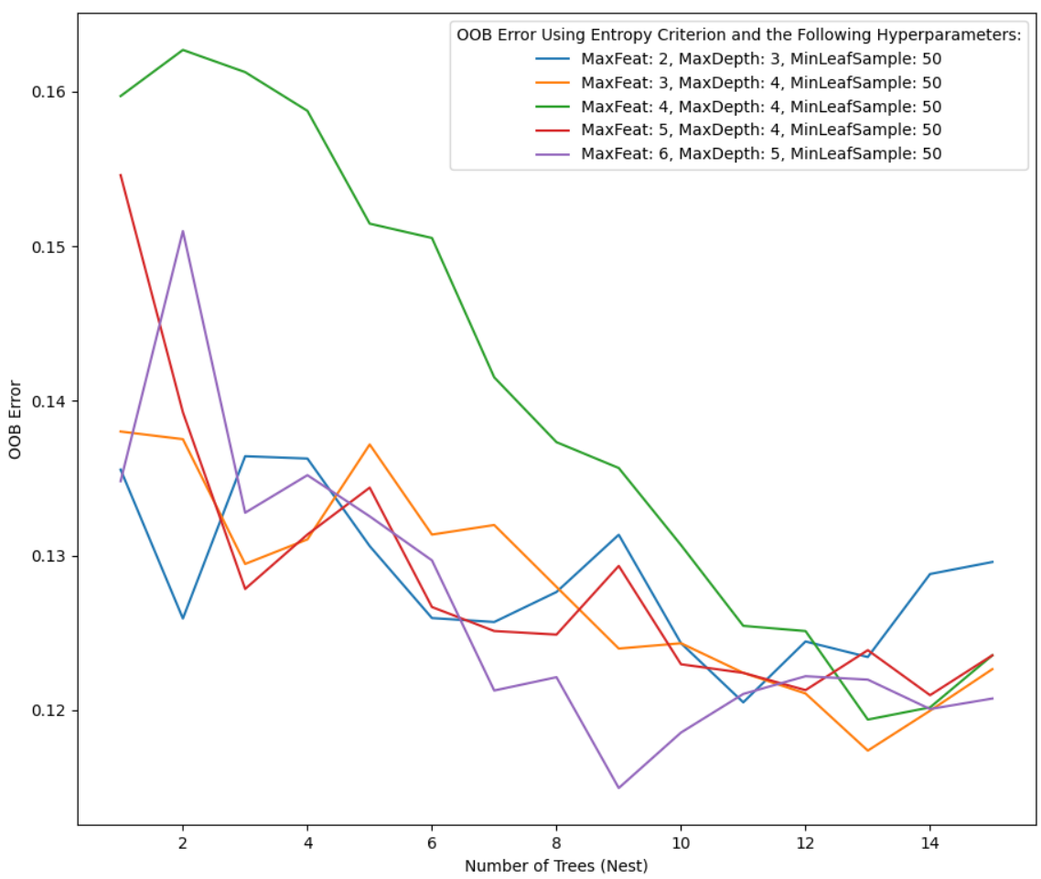
a) See rf.py

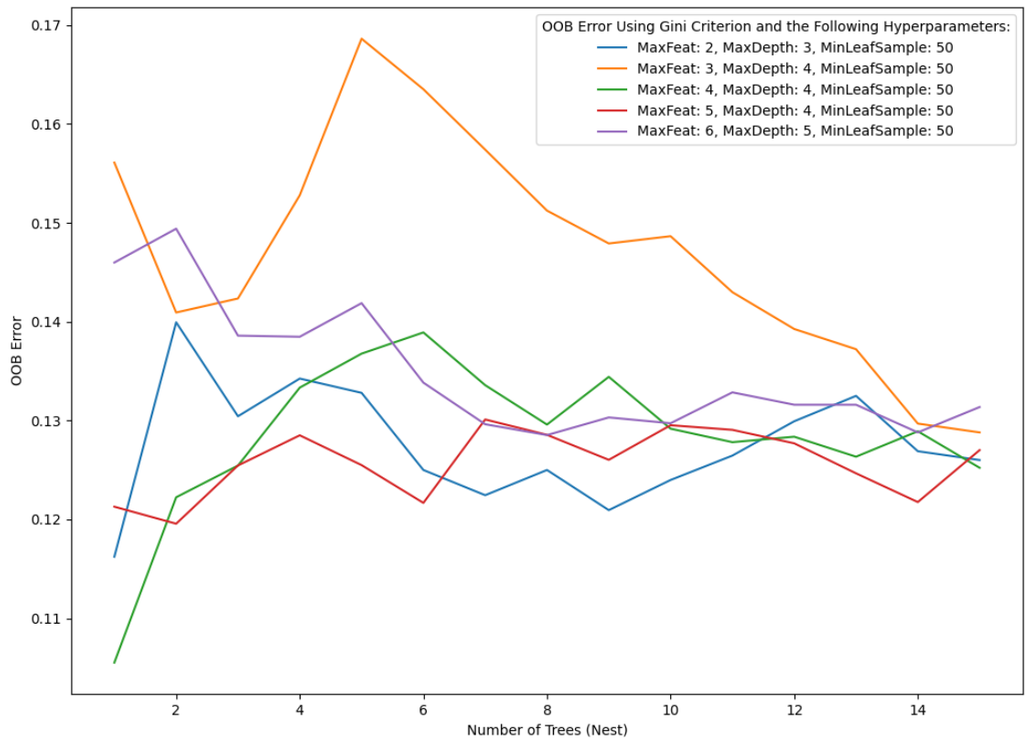
b) For parameter tuning, I chose to plot the OOB of 10 models, each with different hyperparameters. I chose to keep a total of two variables constant: nest and minLeafSample. We know from our second homework that minLeafSample tuning provides a minimal effect, so it would not be worth making the graphs more difficult to read. Therefore I set it to a constant value of 50. Nest was kept the same so that we can compare graphs - we can always change the value of nest given our findings later.

I chose some middle ground values for maxDepth, namely 2, 4, 4, 4, and 5. Trees can very complicated very quickly, so this range of depth I felt provided a good balance on both ends of the spectrum.

The hyperparameter maxFeat, however, got a bit more variety. It ranges from 2-6. Holding the number of features each tree has access to is new to us, so I felt that a wider range would give us a better understanding of their effect on the OOB and therefore the model's effectiveness on a test dataset.

I applied all of these different hyperparameters twice: once using entropy as the criterion and once while using gini. The plots that show the results are as follows:





Peering through the chaos of these graphs, we aim to look for a line that looks like a $log^{-1}(x)$ line. We aim to have our optimal nest use the same hyperparameters as the line that looks the most like $log^{-1}(x)$, but have a nest that has us sit at the end of the elbow of that curve.
Out of all of these lines, the one that looks most like $log^{-1}(x)$ belongs in the Entropy criterion graph with a maxFeat of 3, maxDepth of 4, and a minLeafSample of 50. The optimal nest value I believe here to be 10: the severity of the curve is pretty week so we need to be along a bit farther in order to get to the end of the elbow of the curve.

For future use, RandomForest will be called with the values of: nest=10, maxFeat=3, criterion=entropy, maxDepth=4, and minLeafSample=50.

c) Using the hyper parameters found in part B, I created, trained, and then tested the resulting Random Forest model. The results are as follows:

```
Accuracy: 0.8708333333333333

OOB during training: {1: 0.14009661835748793, 2: 0.11613876319758673, 3: 0.1198547215496368, 4: 0.12255965292841649, 5: 0.1256281407035
1758, 6: 0.12825458052073288, 7: 0.12930232558139534, 8: 0.12876712328767123, 9: 0.12431941923774954, 10: 0.1269126912691269}
```

It performs pretty well! With an accuracy of roughly 87%, we can be happy with our model's performance. We likely struck a good balance between bias and variance and therefore do not have a model that greatly suffers from overtraining. The estimated OOB tells us roughly what our accuracy does too. If we take $1 - 0.127$, we get 0.873. Convert that to a percentage and we get 87.3%, near exact what our accuracy score is when using the test data. This is good as it means that our results are consistent and again a sign that we struck a nice balance when learning and that we did not overtrain.