Jareth Conley Moyo

CS 552 Project 3 – Credit Card Fraud Detection

Introduction

The goal of the project was to explore a variety of classification algorithms in the task of performing credit card fraud detection. The data was divided in a way where we had 2 classes, (whether the given transaction was a fraud or otherwise). However, the data was highly imbalanced, therefore we could not just approach the problem as is. The pie chart in Figure 1 below shows the imbalance between the classes.

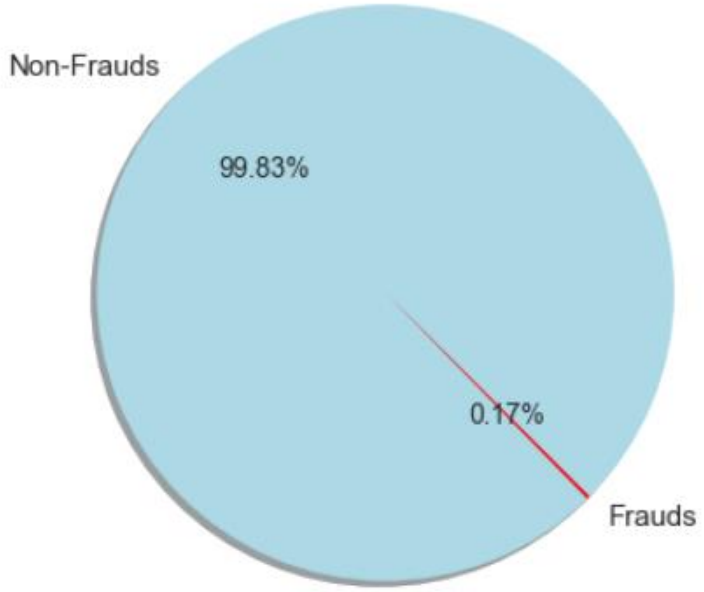

Figure 1 – Distribution of Fraudulent and Non-Fraudulent classes in the dataset

(There is class imbalance in the transactions dataset)

Imagine designing a classifier that would classify all the transactions as 0 (Non-Frauds). Such a classifier would achieve an accuracy of up to 99.83%. To the uninitiated this may seem like good performance but bear in mind that this classifier is not even learning from data, yet is achieving such "great" performance (the accuracy paradox). Immediately the alert reader can already detect something fishy with this approach, that is, we ought to take measures that deal with this class imbalance.

To begin with, our performance measure is something we ought to change in order to have a more accurate measure of what's really happening. By using ROC – curves for precision and recall, we can have a better understanding of what our accuracy truly is (it tends to be less misleading for reasons I'll get into later in the report), confusion matrices, etc.

So let us begin. First let us define the performance metric that we ought to use in the upcoming test. The ideal case would be to split the data into training and test set, perform k-fold cross validation on the training set and finally evaluate this performance on the test set. However, due to these class-imbalance issues, just in this instance (even though it may be considered bad machine learning practice), I will use the whole dataset for evaluation purposes (by performing 5-fold cross validation as suggested) and using the area under the ROC curves as the performance metric in each fold. To find the best techniques to use in handling class imbalance, I'll first use the default parameters of the classifiers I will be testing, then soon after (at the very end), I will optimize these parameters on the best technique found in an attempt to achieve the highest accuracy. For uniformity, I will use SVMs and decision trees/random forests. Generally, decision trees are known to handle class imbalances pretty well, so creating a pipeline using these and comparing them to the performance of SVMs might be an approach we ought to try out.

UnderSampling the Non-Fraudulent Transactions

The first technique I used was undersampling the non-fraudulent transactions in order to balance out the data. What this means is that, since we have way more fraudulent transactions, we will apply different sampling techniques and ratios in order to find the best way to handle this class imbalance.

Random Sampling

The first sampling technique I tried out was random sampling, for various sampling ratios. Figure 2 shows the confusion matrices I got as a result of exploring the sampling ratios 1:1 and 2:1.

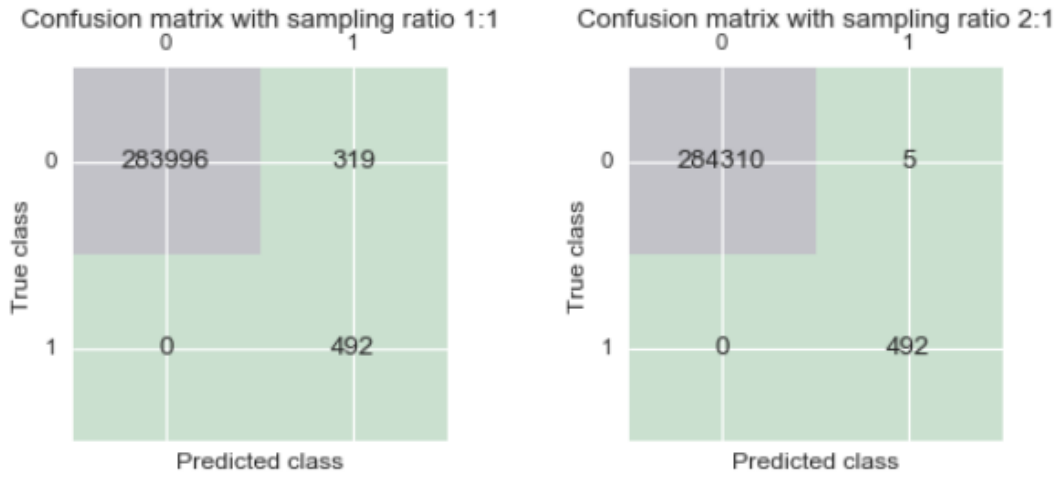

*Figure 2- Confusion matrix on whole dataset after classifying with **SVM** by using different sampling ratios*

First of all, I should explain what I mean by different sampling ratios. You should note that there are 492 total fraudulent transactions in the dataset. So a ratio of 1:1 means that I randomly sampled 492 instances from the non-fraudulent transactions and combined it with my 492 fraudulent transactions to create a new dataset. A ratio of 2:1 means that I sampled 2 times more (984) non-fraudulent

transactions. We define a positive as predicting that a transaction is fraudulent. With that definition, a sampling ratio of 1:1 gives us up to 319 False positives, while a sampling ratio of 2:1 gives us 5 false positives. False positive here means predicting that a transaction is fraudulent when it is in fact non-fraudulent. So it makes sense that the false positives reduce as we increase the number of samples in the non-fraudulent class (a ratio of 5:1 gives us 0 false positives, though this tends to create more imbalance, see notebook), so there ought to be a trade-off somewhere. Based on this alone, it seems as if the sampling ratio of 2:1 is preferable, but we cannot guarantee this on the confusion matrix alone. We need to compute the F-score, look at the ROC curves and such, to have a clearer view of the techniques we are using.

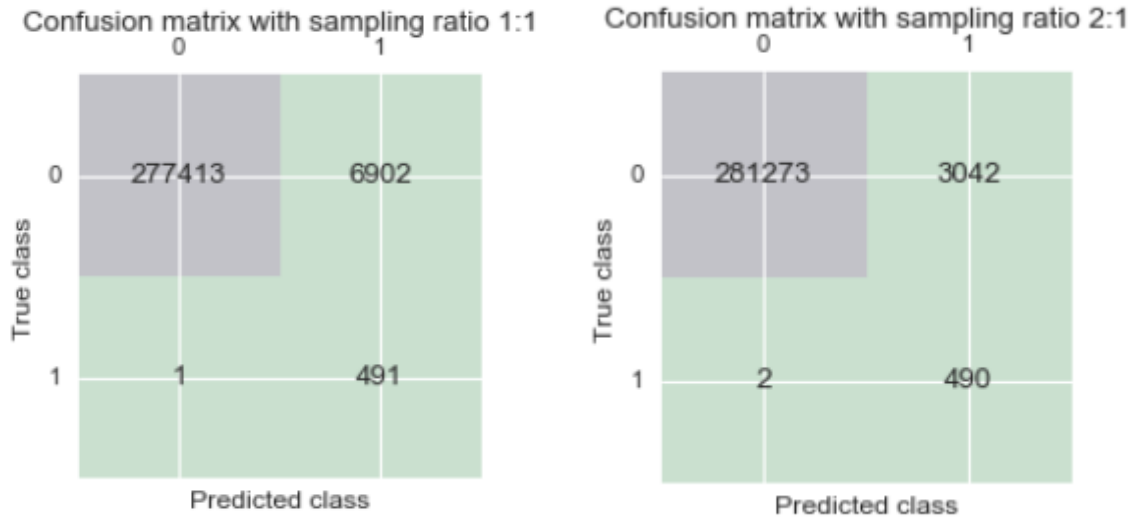

Figure 3 – Confusion matrix for different sampling ratios on ***unoptimized random forests***

Figure 3 shows that we need to optimize our random forests to get more optimal behavior (we'll get to that later), but what we get from this figure is the trade-off between false positives and false negatives, which can be translated to the trade-off between precision and recall. The ROC curve analysis will give us a better view of what's really happening here. We will perform this analysis in a 5 fold setting. Let us think a bit about why we may want to do that. Right now all the fraudulent cases we are predicting are already present in the training set. Therefore, there is a possibility that we may be overfitting the training data here.

ROC Analysis on various classification algorithms

So with the different sampling ratios in mind, I plotted the ROC curves for different algorithms. However, I will present the results here assuming a sampling ratio of 1:1 in my dataset, so that all approaches are evaluated on the same scale.

Firstly, evaluated the performance with SVMs, and the results are illustrated in Figure 4 below:

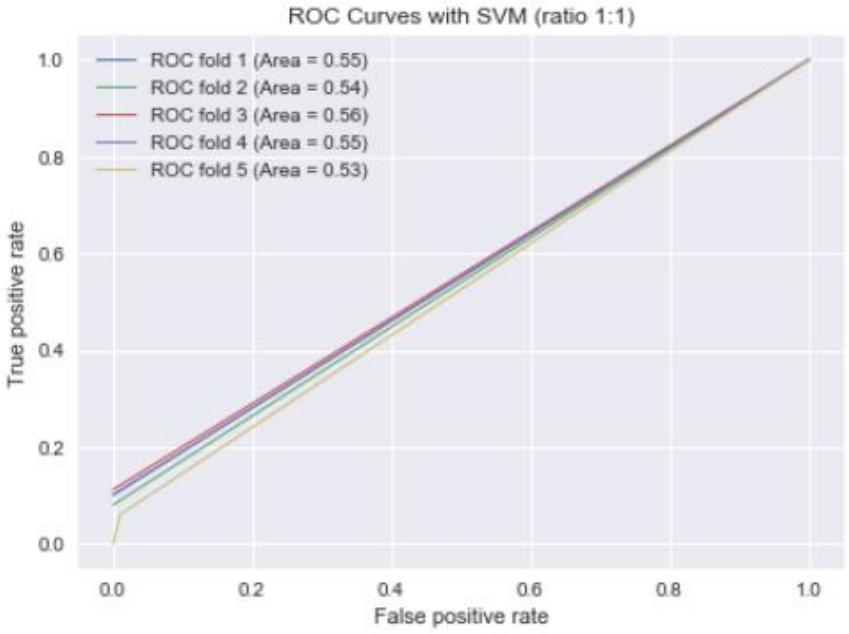

Figure 4 – AUROC using the Support Vector Machine (SVM)

These results are quite bad (just better than random guessing). They also show that the confusion matrix generated after testing on the whole dataset were very misleading, and we were clearly overfitting when we were using SVMs. Perhaps if we optimized the SVM, maybe applying feature scaling and such the results will improve, but I decided to focus my optimization methods on other approaches that were giving far better results. Figures 5 and 6 illustrate the results I got after applying the same evaluation metric on Random Forests and Logistic Regression.

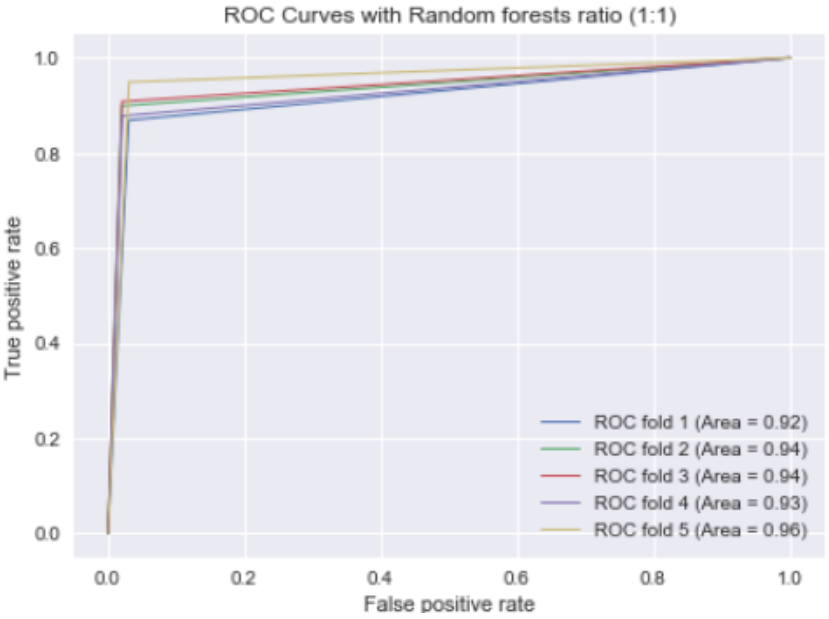

Figure 5 – AUROC using Random Forests

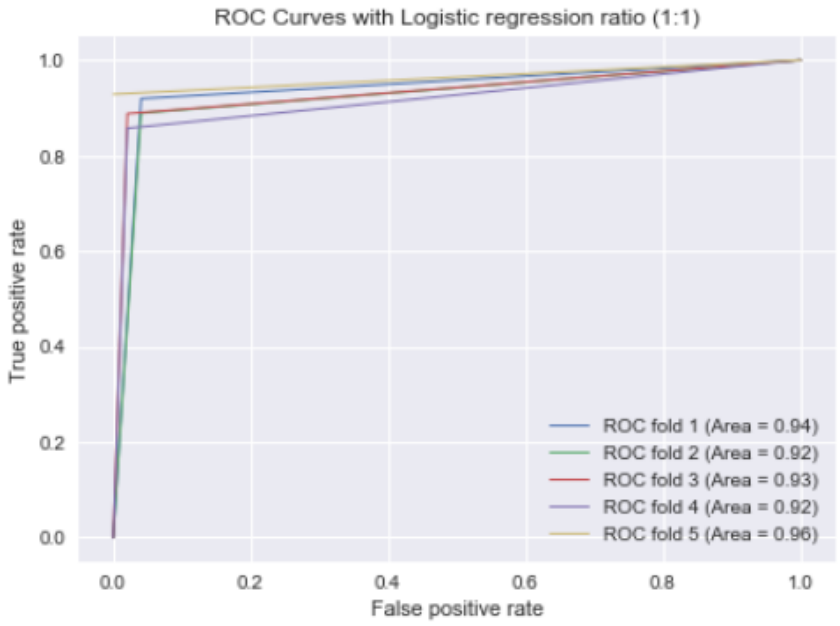

Figure 6 – AUROC using Logistic Regression

Figures 5 & 6 show that random forests and logistic regression achieve far better results than SVMs in this instance. I am not showing it here but the results using the sampling ratio 2:1 were pretty much the same (see notebook for reference). So it seems like undersampling already achieved some decent results on these classifiers, therefore the next logical step may be in optimizing them.

Let us take a step back and look at the data once again. Since we performed undersampling on the data, we lost most of the information, yet we still preserved all of the features. Since the majority of these features were as a result of performing PCA, we do not really know what they are, but what the question we ought to ask ourselves is, just how important are these features? Do we really need all of them for our classification purposes or some of these features are just adding noise to our data?

Assessing Feature Importance with Random Forests

So as it turns out, after assessing the feature importance the important features contributing most to the model are shown in Figure 7 below. The most important features turned out to be the time feature, and the least important the amount. Every other feature's importance seems to be determined by the variance present in it when doing PCA, e.g., after performing PCA, it is most probable that V1 had the highest variance, which would explain why it appeared as the second most important feature, after time. We were able to get this information from the under-sampled data, thereby further validating this approach to be sound.

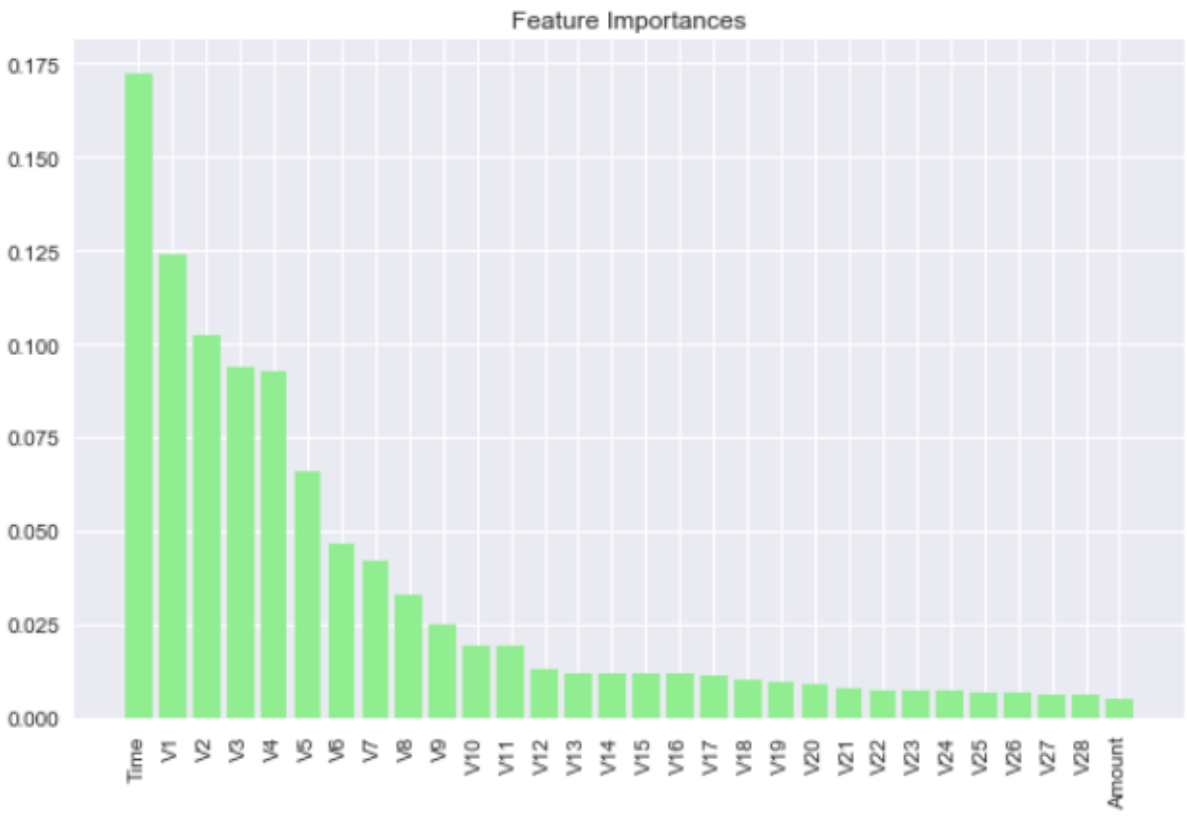

Figure 7 – Assessing feature importance with random forests

It seems that most of these features are somewhat important, and based on the graph above it looks like removing the low ranked ones won't do much in improving out our accuracy (something I tried and realized).

The next step to perform now was hyper-parameter optimization on the random forests, by doing grid search and using auroc score as the accuracy measure.

The results are shown in Figure 8 below.

There isn't that much improvement when using the parameters that were found by grid search. So as such, I decided to try optimizing the logistic regression approach. For this approach, my attempt was to optimize the regularization parameter C, and I plotted the regularization levels against the accuracy in both training and validation settings. The results of this are shown in Figure 9.

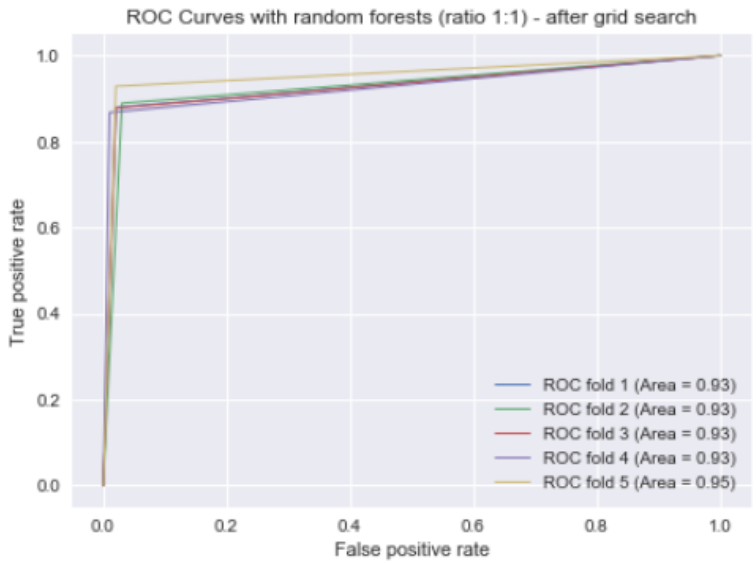

Figure 8 – AUROC after performing grid search on random forests

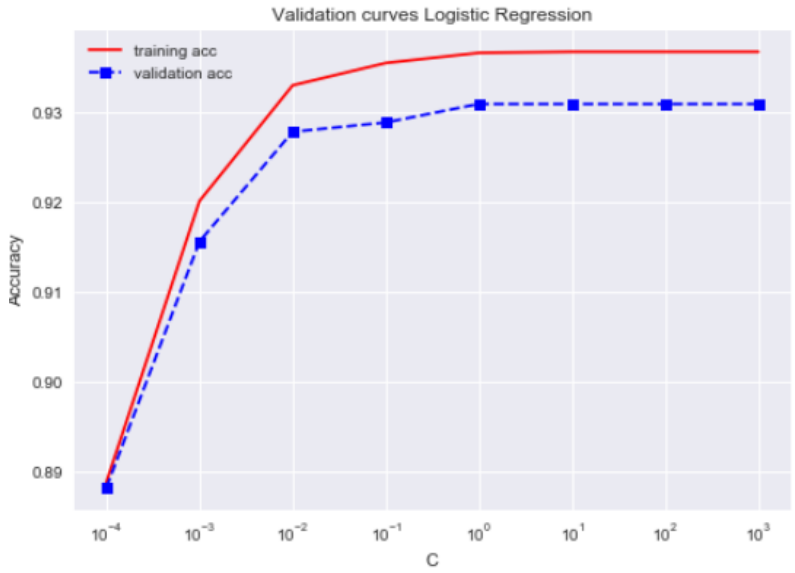

Figure 9 – Training and Validation Accuracies using Logistic Regression at different regularization levels

Gradient Boosting (Yet another tree based classifier?)

For an exhaustive search of applying different classifiers, I turned to Gradient Boosting. As such, I tried playing around with the max depth parameter, and the accuracies using this parameter are shown in Figure 10.

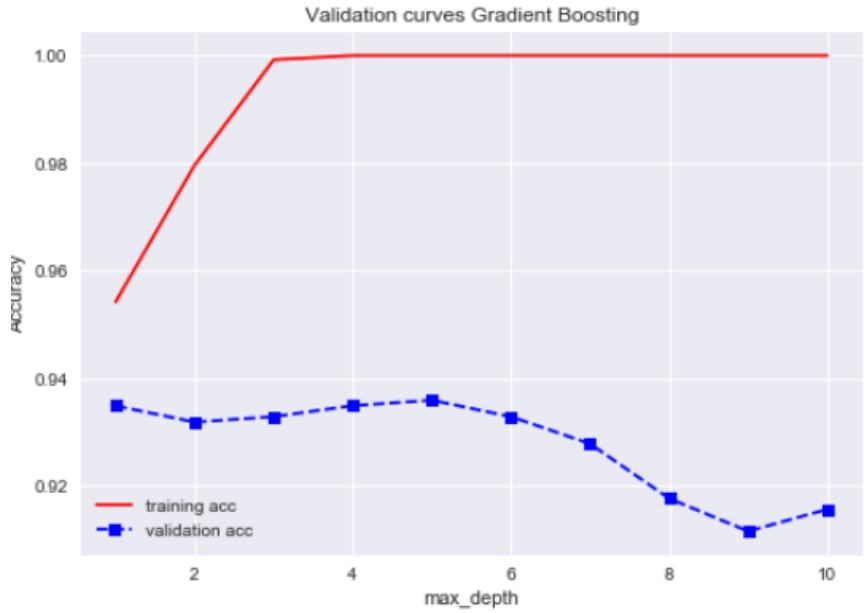

Figure 10 – Validation Curves for gradient boosting

What Figure 10 shows is that with a max depth of 5, we can get the best performance in the validation setting. Now after tuning intuition based parameters (based on the data itself), with gradient boosting the accuracies are shown in Figure 11 below:

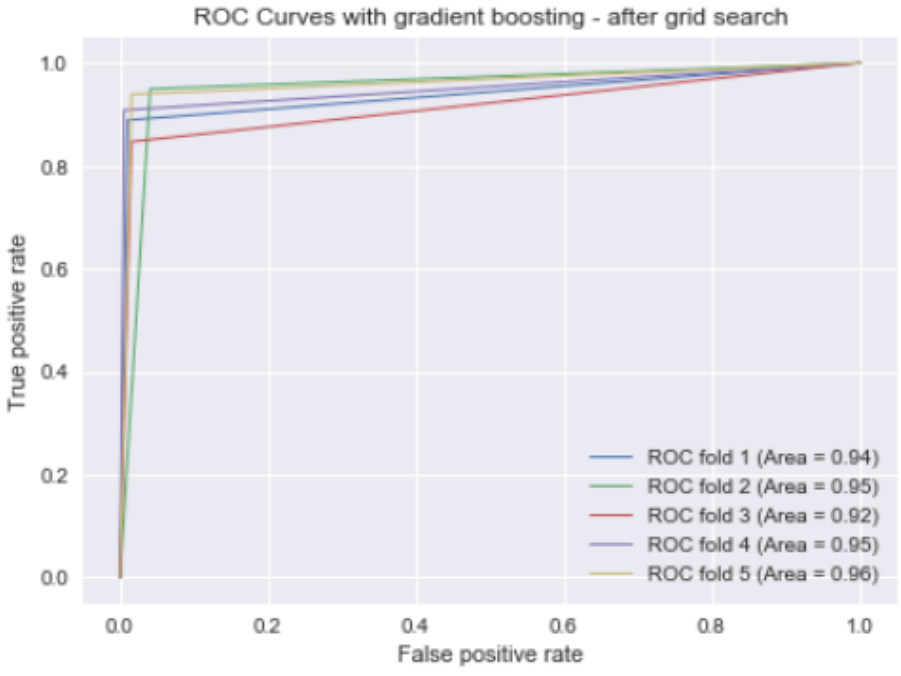

Figure 11 – AUROC after applying gradient boosting (An Increase in performance)

Figure 11 shows that there was an increase in performance after applying gradient boosting as compared to prior methods, and this was with minimal optimization. I attempted performing grid search with gradient boosting but due to limited computational power, I could not run the parameters I intended and the minimal experiments I did run did not lead to an improvement in the results shown above.

For completion I also evaluated the classifiers using just 5-fold accuracy as it was specified in the assignment description. The mean of these accuracies is shown in the table below for different classifiers.

Classifier	Mean Accuracy
SVM	68.9%
Logistic Regression	93.4%
Random Forests	94.6%
Gradient Boosting	95.4%

Table 1 – Comparison between different classifiers by taking the mean of the 5-fold validation accuracy

It is quite apparent from the table above that Tree based classifiers perform best in this dataset, hence, any further optimizations ought to be in this direction.