**Identifying Fraud Credit Card Transactions**

1. **INTRODUCTION**

It is important that credit card companies can recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Some of the negative effects on the banking industry, customers and the economy include,

* Damage the image of trust and confidence that banks are expected to portray and exemplify,
* Customers lose the money they did not pay for,
* Insecure financial environment

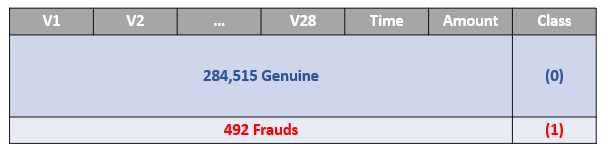
This project is an effort of applying machine learning to detect frauds. Financial firms offering credit cards to consumers may find this useful as it contributes to minimizing the risks of wrongly proceeding fraud transactions.

1. **DATA SET**

This dataset is available on Kaggle. The dataset contains transactions made by credit cards in September 2013 by European cardholders. It presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. All computational works of this project were done in python and its libraries for data science.

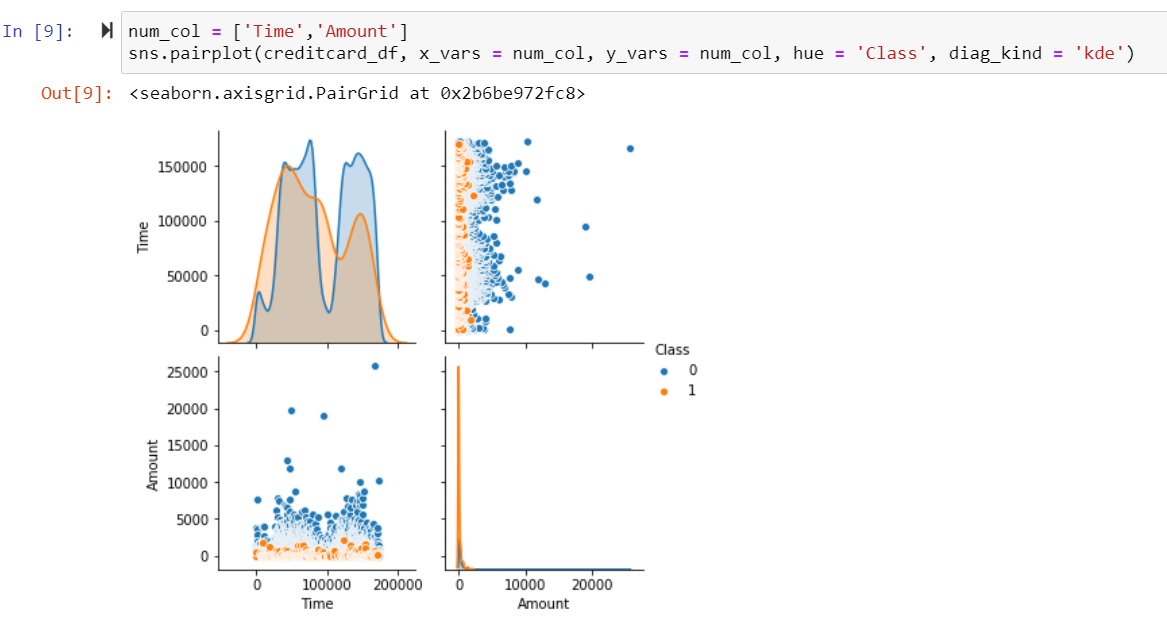
1. **EXPLORATORY DATA ANALYSIS and DATA WRANGLING**

The dataset contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise (Figure 1). The dataset from Kaggle is clean. I did not have to put effort for missing or invalid data points. I split the dataset into train-test sets with ratio of 80% and 20%.



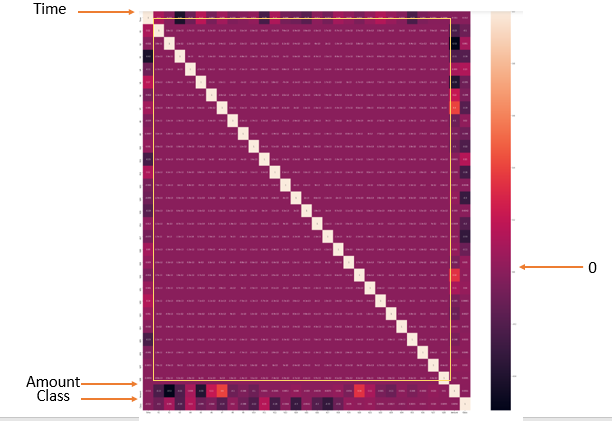
**Figure 1. An Overall Picture of the Dataset**

As V1-V28 are components of a transformed dimensions, it is guaranteed that they have zero correlation between each other. However, ‘Time’ and ‘Amount’ are not parts of this dimension. We will start exploring distributions and correlation between them first (Figure 2).



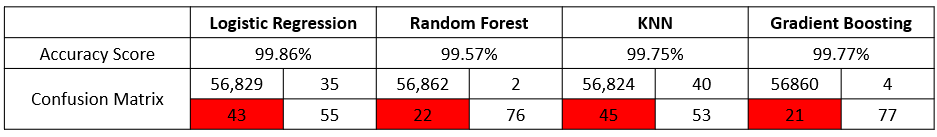
**Figure 2. 'Time' and 'Amount' Variables**

The plots indicate that frauds can occur any time along with genuine transactions. They tend to be at lower amount relative to the non-frauds. A correlation matrix is also generated to understand the correlations between variables (Figure 3). As mentioned above, correlation between components of V1-V28 should be 0. Therefore, it is not worth paying attention to area in yellow box in Figure 3. The matrix also indicates there are some correlation between Time and Amount and V1-V28 (bright and dark spots in the matrix). These linear correlations will play key factors when it comes to considering machine learning algorithms to model this dataset. We do not want to utilize models being unstable with correlated variables.



**Figure 3. Correlation Matrix**

As found above, the data is very imbalanced. It is risky to model this original data without any resampling technique. The accuracy score can be extremely high, say 99%, but it might detect only the non-fraud transactions. This is because the non-fraud transactions occupy 99% of this dataset. To model this, some potential algorithms (logistic regression, random forest, k-nearest neighbors, gradient boosting) were quickly fitted to the training set. The fitted models were then tested against the testing set.

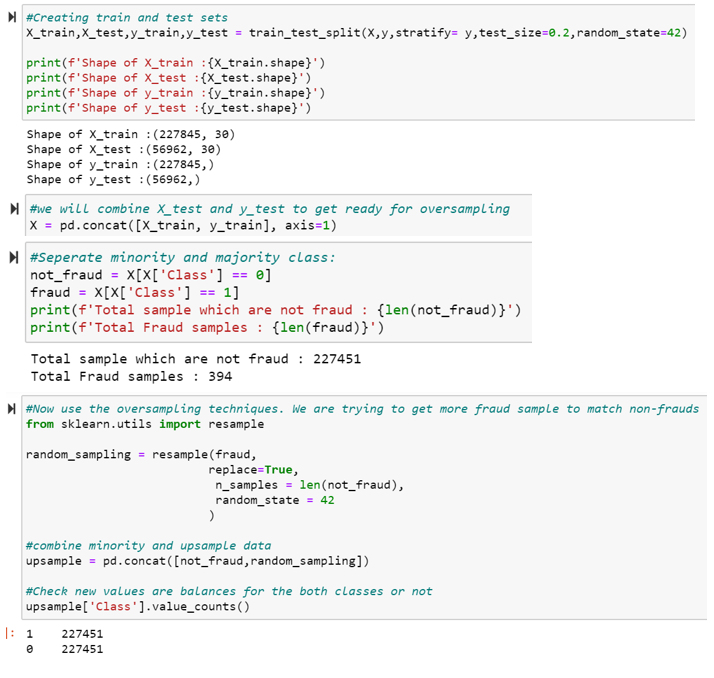


**Figure 4. Confusion Matrix for Each Model without Resampling**

Figure 4 shows the confusion matrixes for the models. The second row of each matrix sum up to be the total number of frauds. The red highlighted cells show the frauds mistakenly identified as non-frauds by the models. They have shown that the accuracy can be extremely high for these models. However, they do not detect the fraud transactions very well.

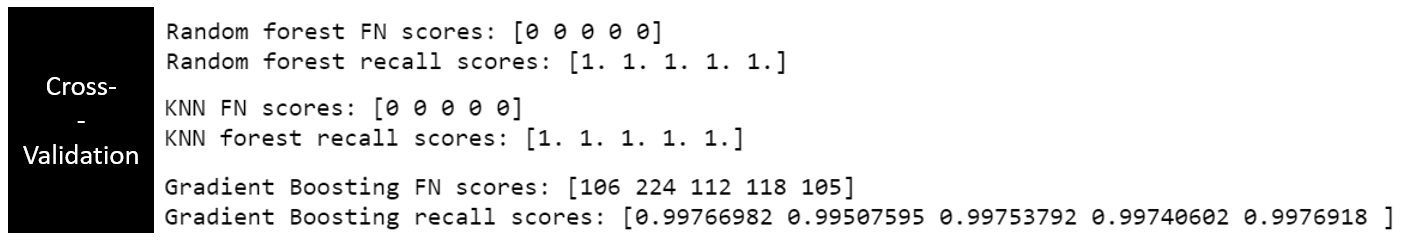
As mentioned earlier, the variables have multicollinearities. For this reason, we will set aside modeling with logistic regression for now to avoid the instability of the prediction. Further analyses will include only random forest (RF), k-nearest neighbors (KNN) and gradient boosting (GB).

The first resampling attempt is oversampling. The minority class (frauds) will be oversampled to match the majority class (non-frauds). The screenshot below shows the python script and result from oversampling.



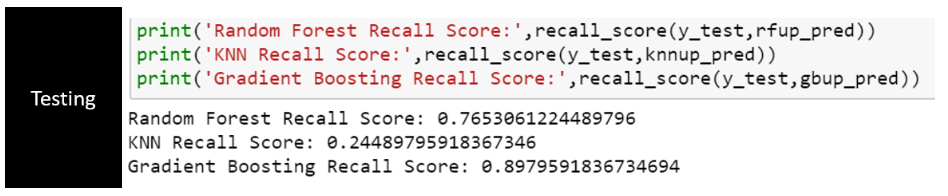
**Figure 5. Oversampling**

When oversampling the dataset, many unactual data might have been generated. This could cause overfitting. To verify this, a cross-validation process has been done to train the models. Then they were tested with the testing set. The cross-validation results already indicate the sign of over fitting. We are seeing a lot of perfect or almost perfect score (figure 6)



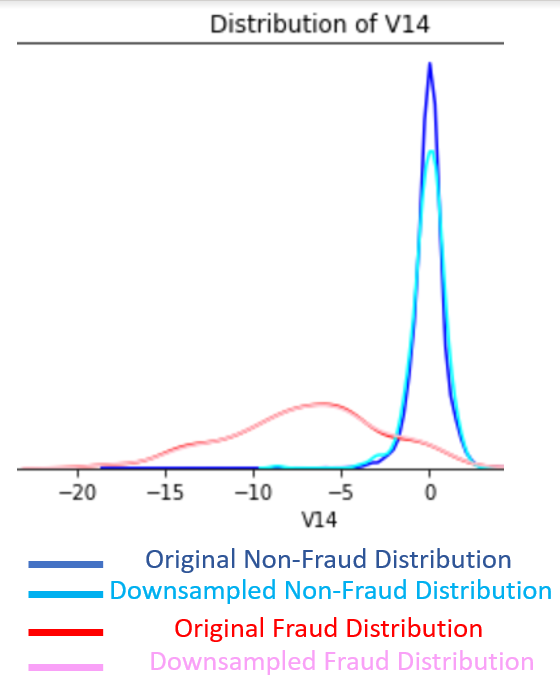
**Figure 6. Cross-validation Result for Oversampled Data**

If we test these models for a recall score (which reflect how good the model is at detecting frauds), we see that the result is not very desirable (Figure 7). In addition, generating a lot of data also consumes a lot of computing power. In this economic and competitive world, the faster the prediction, the better the company performs.



**Figure 7. Test Result for the Oversampled Data Fitted with Models**

My next approach to handle that, I tried downsampling. One of the drawbacks of this method is that information can be lost as we cut off many data points in the majority class. However, if data points are close to each other, downsampling will not affect much and can yield good result. Therefore, after I obtained the down sampled, I made a validation check. I plotted distribution of the original and downsampled variables. If there is not much change in the distribution shape, I assume the downsampled data is effective and can be used for training the algorithms. Figure 8 is an example of this sanity check to make sure the undersampled and original set share remarkably similar distribution.

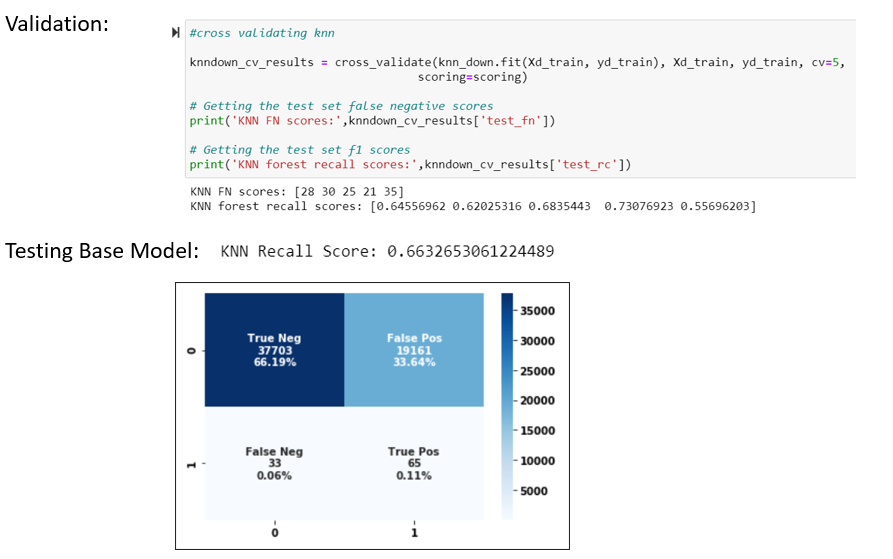


**Figure 8. Distribution of V14 before and after undersampling**

1. **MODELING**

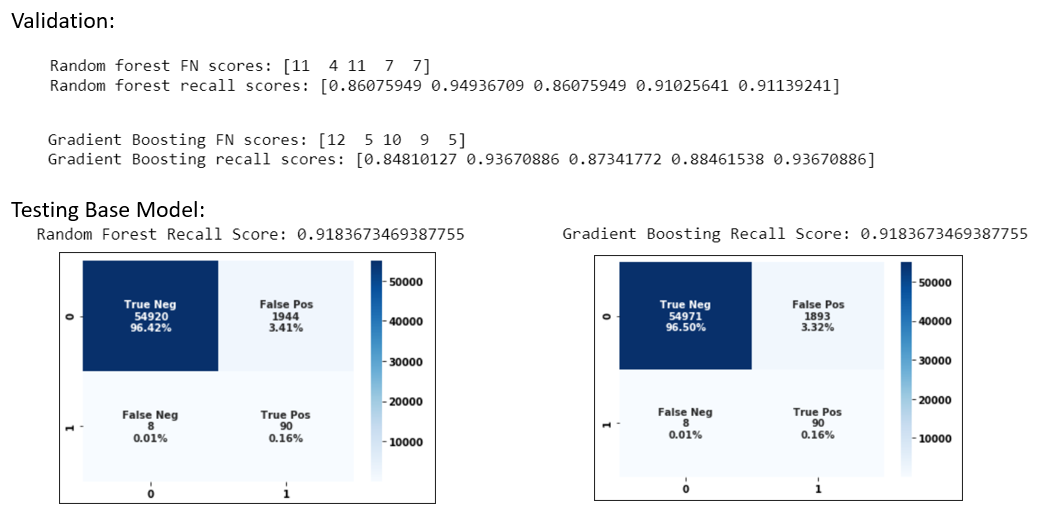
With the significant finding above, we will proceed to modeling with the undersampled dataset. As a standard process, we will cross-validate the models and select the candidate we want to bring to tuning for optimized hyperparameters. Once tuned, we will test the model with the unseen data to confirm the detective power.

Figure 9 shows the validation and testing result for KNN models. The fitting was done with setting the number of nearest neighbors to 10. The result still does not look great.



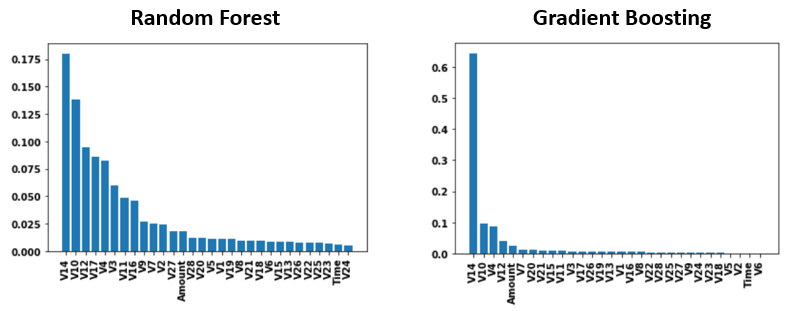
**Figure 9. Cross-validation and Testing Results for KNN Model**

Figure 10 shows training and testing results for RF and GB models. These results look promising. The cross-validated score and testing score a not far off. Therefore, overfitting is not the issue in this scenario. The performance of RF and GB are remarkably close to each other. We would want to tune them both and compare the tuned models for a finalized prediction.



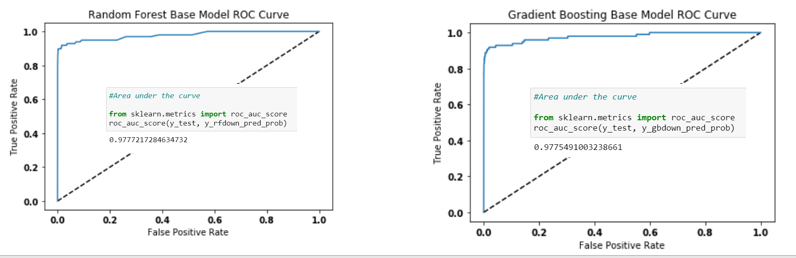
**Figure 10. Cross-Validation and Testing Result for RF and GB Model**

We also want to check for feature importance as we implemented RF and GB. The two models show similar importance of the features. Especially, V14 is the most important variable in both models.



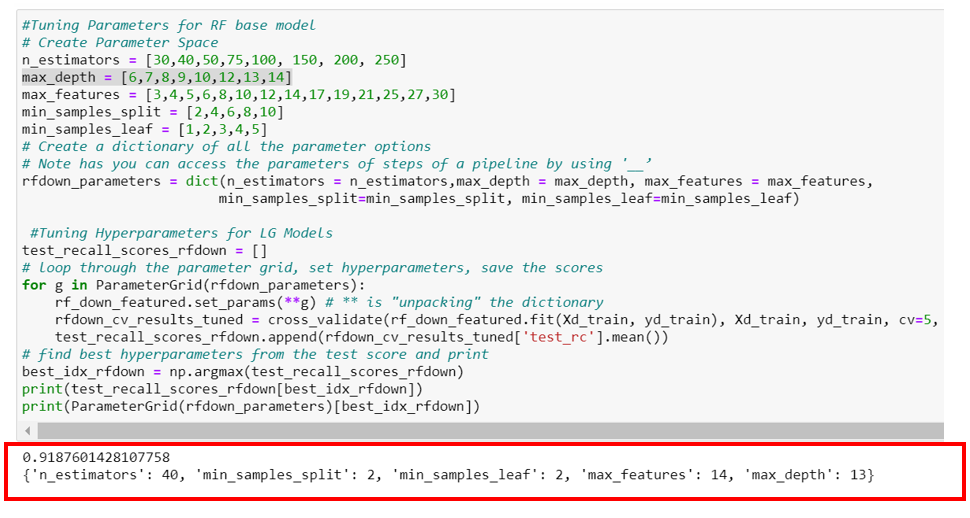
**Figure 11. Feature Importance of RF and GB**

In addition, we want to make sure the model is predicting instead of guessing or mislabeling. Therefore, we want to check the ROC curve and the aera under it. The calculated area under the ROC curve is over 97%. This is great because it tells us our model is predicting.



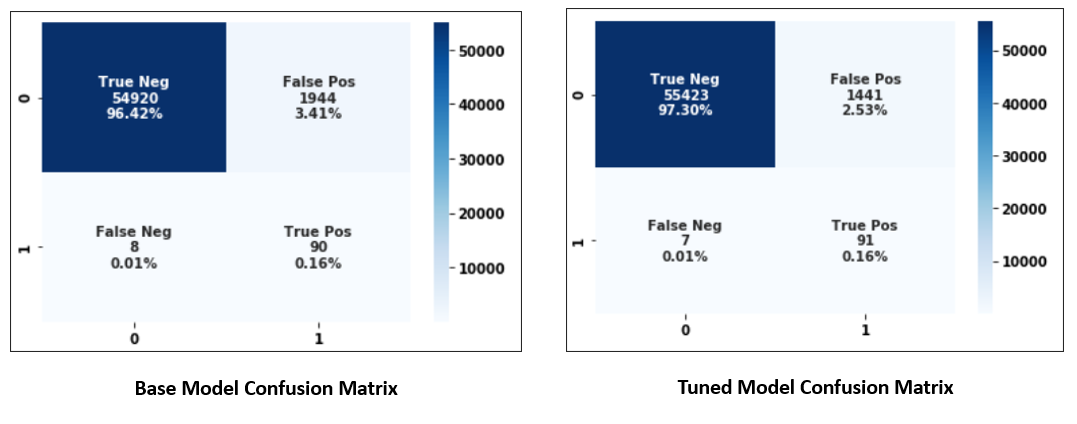
**Figure 12. ROC Curve and AUC**

Figure 13 shows the Python script for tuning the RF model and the optimized hyperparameters.



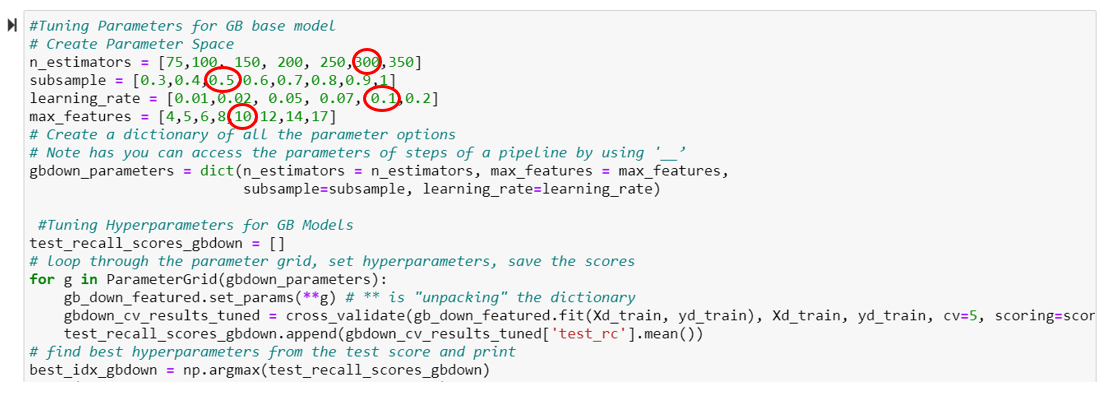
**Figure 13. Hyper Parameter Tuning for RF Model**

We are expecting the model to perform better with the testing set. Figure 14 shows the comparison between the base and tuned models. The tuned model performs better in terms of identifying the correct transactions (both frauds and non-frauds). In the base model, it misclassifies 8/98 frauds. The tuned model misclassified 7/98 frauds and identify more correct non-frauds.



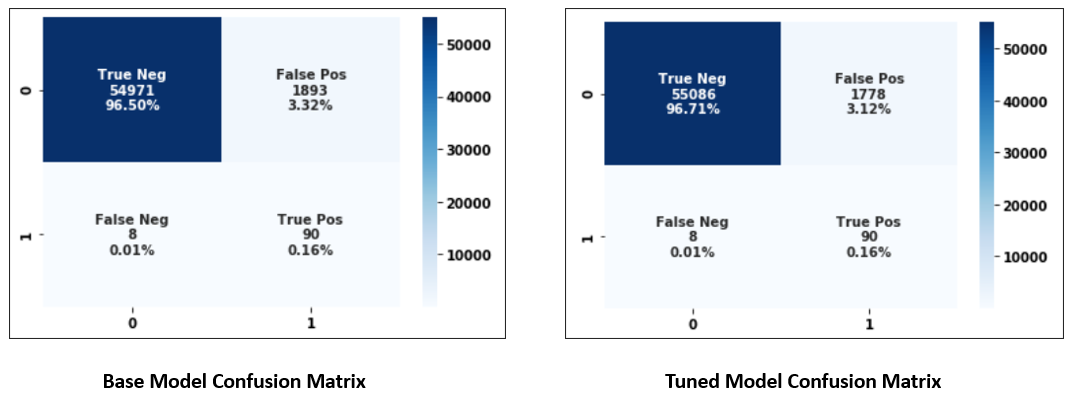
**Figure 14. Base and Tuned Model Comparisons**

Figure 15 shows the Python script to tune hyperparameters for GB model. The values in red circle are the optimized hyperparameters.



**Figure 15. Hyperparameter Tuning for GB Model**

We want to check the confusion matrix for base and tuned model to see any improvement. Figure 16 indicates that the tuned GB model only improve the identification of non-fraud transactions. The misclassification remains 8/98 for both base and tuned model.



**Figure 16. Base and Tuned GB Model Confusion Matrix**

With the above result, we would want to finalize with the random forest model to tackle this fraud detection problem.

1. **CONCLUSIONS**

Some key points from this analysis includes,

* For this credit card transaction dataset, undersampling has been necessary to avoid overfitting
* The success of undersampling is ensured by validating each variable’s distribution before and after such process.
* RF and GB models are validated to detect 90/98 frauds transactions from the testing sample. Testing the tuned models shows that it the tuned RF model improve detecting more fraud transactions.
* RF and GB yield close prediction power. RF perform slightly better than GB in terms of recognizing more genuine and fraud transactions.

For future improve, the following works can be considered,

* Research the sensitivity of the model to the size of the training set
* Stabilize logistic regression model by orthogonalize the features
* Try different resampling techniques (SMOTE, GAN) to oversample the minor class without introducing overfitting. Hopefully, this can improve the fraud detective power
* Adjust the predicted probability to get more true positives