Identifying Credit Card Fraud using Machine Learning

Manoj Kumar Eega(M12708663)

Primary Reader : Yan Yu

Secondary Reader : Dungong Liu

Table of Contents

Abstract:

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. Even though the percent of a fraudulent transaction is very low in proportion to the total transactions, fraud transactions can hamper the consumer sentiment. This will be a damaging thing to the company, as the customer might not use the card later, and for the country, because a huge negative sentiment will put the consumption-based economies at risk. So, we decided to use our skills to build a model that can accurately classify fraud transactions. I found this problem particularly interesting because the problem at hand has a huge class imbalance. It is a challenge to address this issue and simultaneously build models that have good recall and accuracy. So, we want to use all the techniques available to address the class imbalance and build the best model that has the highest predictive capability. I am addressing a real-world problem with real-world data. All transaction monitoring companies such as credit card companies, banks, insurance companies, etc. can make use of this project.

Project Objectives:

1. Understand the European credit fraud data
2. Look at the summary statistics and identify the class imbalance
3. Clean, transform the data so that they are ready for modeling

4. Deal with class imbalance using techniques like undersampling, oversampling, SMOTE, etc.
5. Build several machine learning models like decision trees, logistic regression, support vector machines, and neural networks
6. Cross-validate the results and assess the performance of each model on test data
7. Select the champion model

Variables:

The data contains transactions made by credit cards in September 2013 by European cardholders. In the data, we have 492 fraud transactions out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

The dataset contains 30 numeric variables. Features V1, V2, … V28 are the 28 principal components obtained by doing principal component analysis, and 'Time' and 'Amount' are the other 2 numeric variables. The variable 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The variable 'Amount' is the transaction amount. The variable 'Class' is the binary response variable and it takes value ‘1’ in case of fraud and ‘0’ otherwise.

Data Access:

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. They have uploaded the anonymized data on Kaggle for public use and I downloaded it from Kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud). This is a single dataset and I didn't use any other data in the project.

Data cleaning:

1. There are no columns with missing values
2. The variable 'Time' is not needed. So, I have removed it from the data
3. The 28 principal component variables are scaled variables, but still, they have outliers. I have used outlier detection techniques like +/- 3 standard deviations away from mean. If there are many such outliers, I will remove only the extreme ones.

Data Transformation:

I have transformed the variable 'Amount' because it was extremely right-skewed. I used a log transformation to bring it closer to normal distribution. Even after log transformation, the variable is highly skewed. So, I scaled the variable using standardized scaling, which is making it normal by subtracting the mean and dividing by its standard deviation.

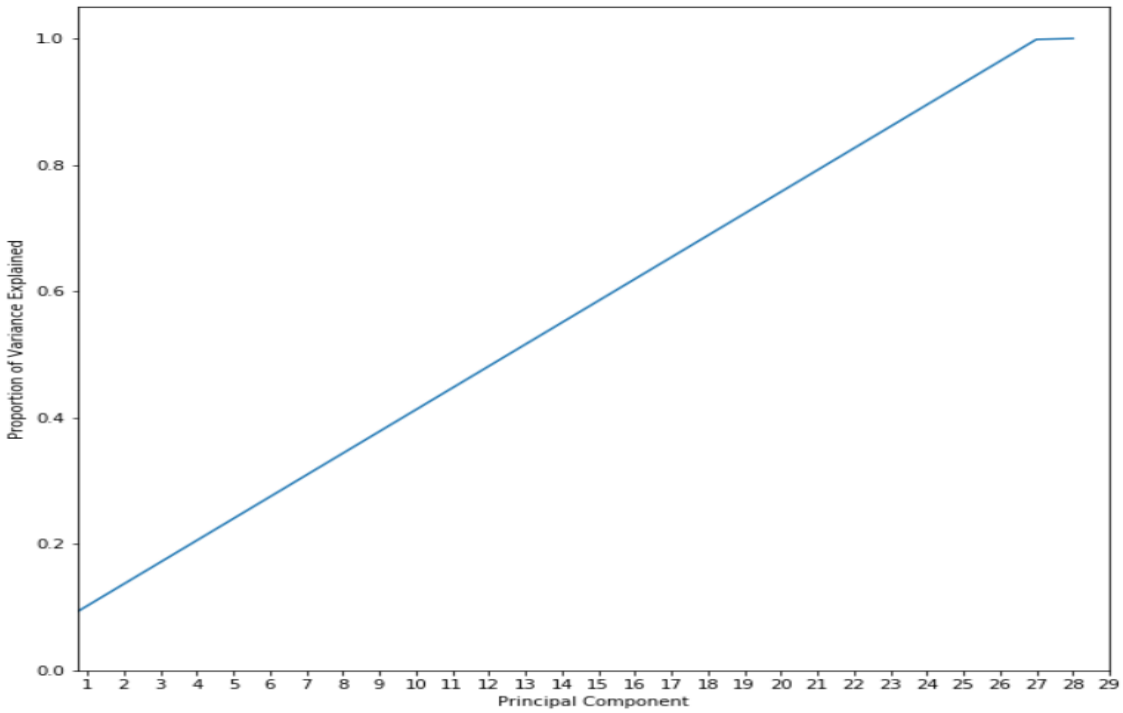
Data Reduction:

We have 28 variables that are the result of doing principal component analysis on the original data. There might be no need for any extra data reduction techniques as one such technique has already been applied to the data. However, there might be some variables that won't be needed in the analysis due to their poor predictive power. I will remove those variables during regularization or while using variable selection techniques such as decision trees, or stepwise logistic regression.

But to identify the data redundancy, if any exists in the data, I have performed principal components analysis on the 29 variables except 'time' and the target variable.

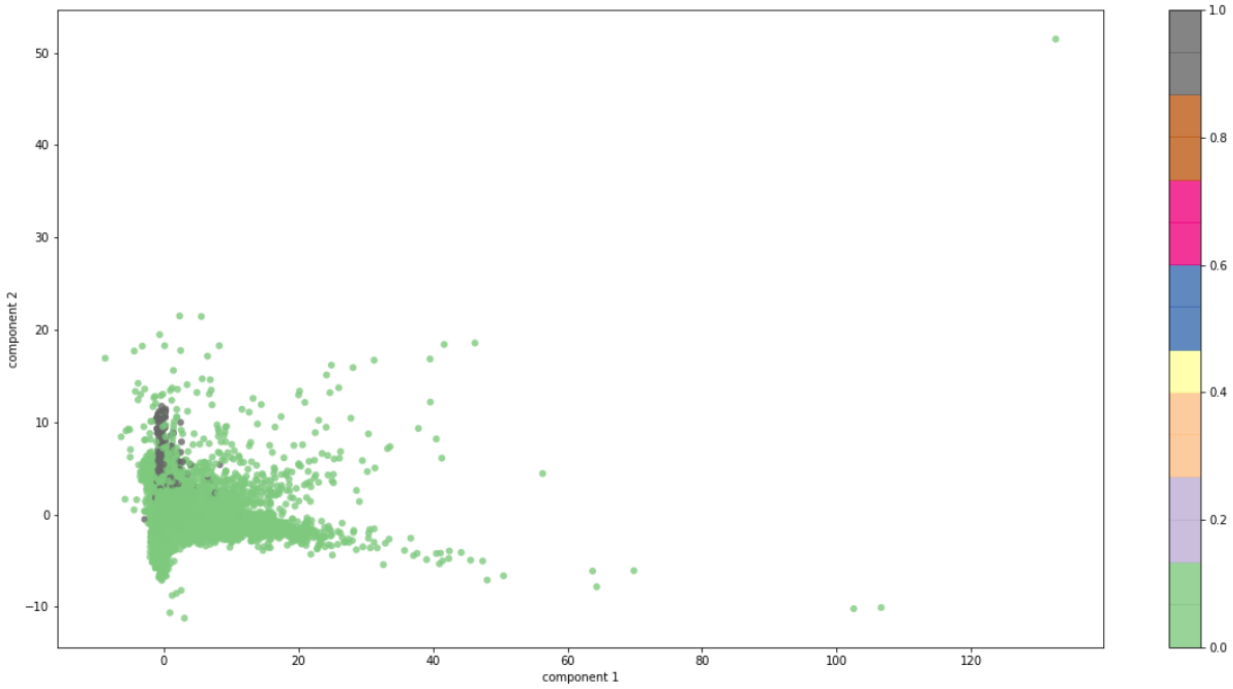
Principal component analysis (PCA) is a technique for reducing the dimensionality in datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance. Finding such new variables, the principal components, reduces to solving an eigenvalue/eigenvector problem, and the new variables are defined by the dataset at hand, not a priori.

Fig1: Result of principal component transformation on the data



The results of applying principal component transformation on the standardized data are above. The image shows us that the proportion of cumulative variance explained increases linearly as we go up in principal component number, showing that none of the variables are redundant up to 28 variables. This is because of the fact that the variables are already principal component versions of some other features. The last component doesn't contribute much to the explained variance, and hence can be removed. But, selecting 28 among 29 variables doesn't make much of a sense as it is not contributing to reducing the data by a substantial degree. So, I have decided to not use the principal component results in the project.

Fig2: Scatter plot of observations against 1st and 2nd principal components



When the first 2 components are taken and plotted against the target variable levels, we can see that the majority class is distributed widely but the minority class is aggregated in a small lump in grey color. There is quite some distinction just by the 2 components. If we take more components, we will have a much clearer division.

Data Dictionary:

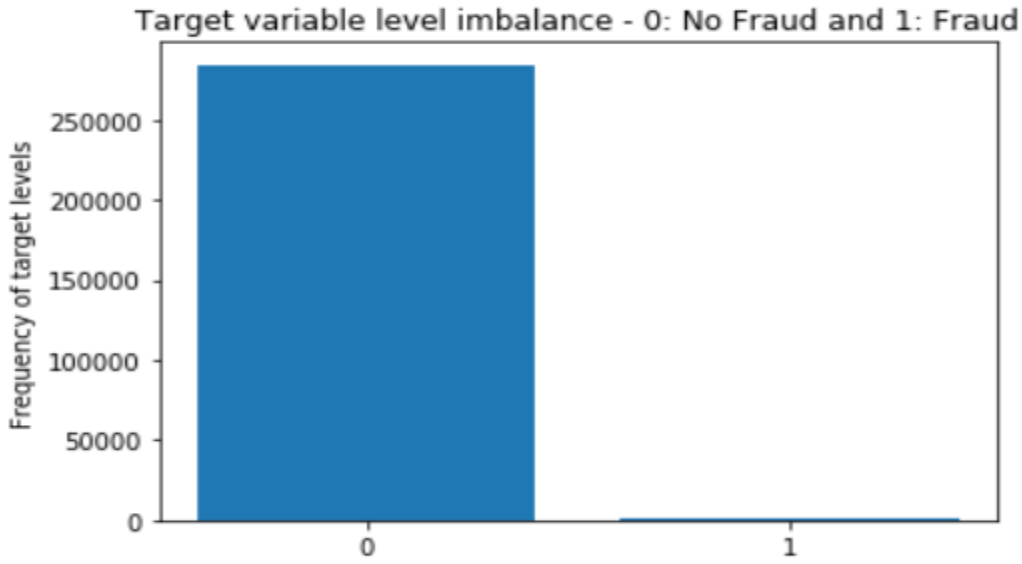
Attribute Name	Description	Data Type
Time	Number of seconds elapsed between this transaction and the first transaction in the dataset	Numeric
V1 - V28	Result of PCA Dimensionality reduction technique on several numeric variables to protect user identity and sensitive information	Numeric
Amount	Transaction amount	Numeric
Class	'1' for fraudulent transactions, '0' otherwise	Binary

Data Splitting and Sub-Sampling:

Class Imbalance:

The number of fraud transactions is 492 and the non-fraud transactions are 284,315. This has resulted in a heavy class imbalance. We can't directly apply a machine learning model to this data. If we do so, the model will predict all cases as non-fraud, and it will result in an accuracy of 284,315/ (284,315 + 492) = 99.82%. As good as it seems, the model won't develop any capability of real classification because it doesn't need to. So, to address this issue, we need to sample the data to bring both the levels of the target variable to equal proportion. Several established techniques exist in practice and we will explore 3 techniques in this project.

Fig3: Class Imbalance: Proportion of target variable levels



Random Under Sampling:

This sampling technique is applied to randomly under-sample the majority class to bring their count to match with that of the minority class. In this case, the technique

randomly selects 492 non-fraud cases and all fraud cases, resulting in an equal proportion of both the target levels. This, however, will lead to us vomiting most of the information because we are using only 984 rows out of 284,807 rows available in the data.

Fig4: Results of the dass proportion of target variable dasses after random undersampling

```
Undersampled training dataset shape Counter({0: 369, 1: 369})
Original training dataset shape Counter({0: 213236, 1: 369})
Original testing dataset shape Counter({0: 71079, 1: 123})
```

Random Over Sampling:

This sampling technique is applied to randomly oversample the minority class to bring their count to match with that of the minority class. In this case, the technique randomly duplicates all the fraud cases to the count of 284,315, resulting in an equal proportion of both the target levels. This, however, may lead to extreme overfitting of the model, as we are just adding more duplicate data points rather than adding more valuable information.

Fig5: Results of the dass proportion of target variable dasses after random oversampling

```
Oversampled training dataset shape Counter({0: 213236, 1: 213236})
Original training dataset shape Counter({0: 213236, 1: 369})
Original testing dataset shape Counter({0: 71079, 1: 123})
```

SMOTE-TOMEK Over Sampling:

This technique is a unique one in the sense that it will combine the nature of both the techniques mentioned above. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and

drawing a new sample at a point along that line. It synthesizes new data instead of duplicating the data. But, sometimes, the SMOTE method generates noisy data points by interpolation new points between marginal outliers and inliers. This issue can be solved by cleaning the space resulting from over-sampling. Tomek's link technique does the cleaning part. A Tomek's link exists if the two samples are the nearest neighbors of each other. Tomek links remove the unwanted overlap between classes where majority class links are removed until all minimally distanced nearest-neighbor pairs are of the same class.

Fig6: Results of the class proportion of target variable classes after SMOTETOMEK oversampling

```
SmoteTomek dataset shape Counter({0: 213236, 1: 213236})
Original training dataset shape Counter({0: 213236, 1: 369})
Original testing dataset shape Counter({0: 71079, 1: 123})
```

Model's test performance evaluation:

A model's training performance is usually an optimistic metric that will overestimate its test performance. Because the model has already seen the training data and all the parameters are tuned to match the training data's underlying pattern. However, the model might be overfitting on the training data (follow the trend too closely) and may not correctly generalize on the test data. This inherent tendency of the model will leave us unsure of whether to deploy it in production. So, we need to ensure that its test performance is good. We are using 2 widely used approaches in this project.

Holdout sample:

We have divided the data into 2 parts, the first one contains 75% of the data and the second one is given 25%. The idea behind the splitting is that we build the model on the training data and use it to evaluate its performance on the test data. This test performance is a more accurate estimate of its true usability.

Since there is a heavy class imbalance, randomly splitting the data to a 3:1 ratio might not give correct distribution across the minority class. So, I sampled the majority and minority classes separately and then combine the training majority and minority classes and test majority and minority classes.

Cross-Validation:

Since we have only 492 fraud cases, we must use all the very less information we have to build our models. But the holdout method lets us use only 369 fraud cases and lets the remaining 123 for validation. With k-fold cv, we can eliminate that problem. We split the entire data to k folds, use k-1 folds to build the model, and the unused 1-fold to evaluate the test performance. We then iterate this procedure, so that each one of the k-folds will be used for evaluating the test performance only once. This procedure lets us use all the data to build the models, without leaving us unsure of the model's test performance.

Clustering:

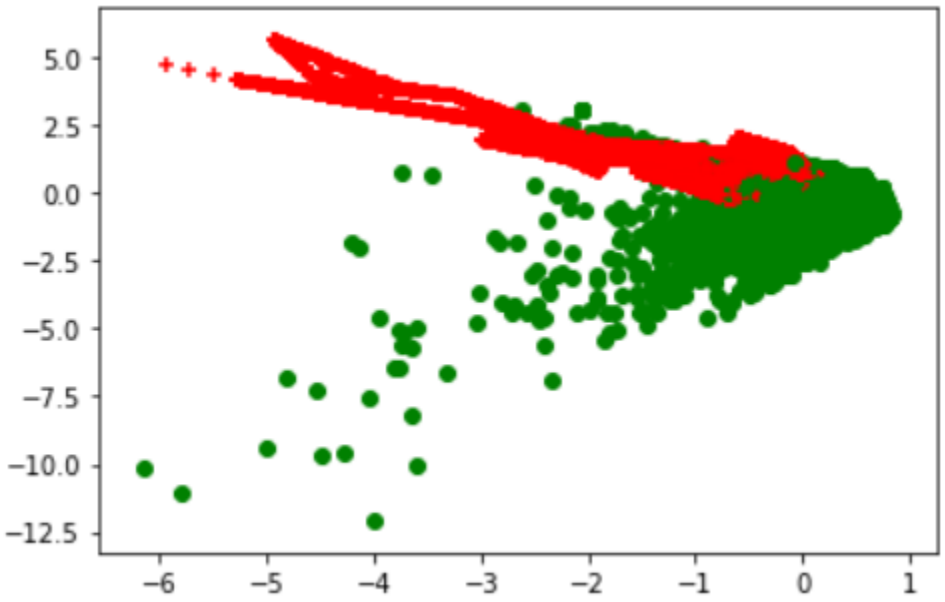
K means clustering is a great way to see the natural groupings in the data. It uses the information in all the numeric variables in the data and identifies clusters in the n-dimensional space that are clearly distinguishable. We have performed SMOTETOMEK oversampling on the data and built 2 means clusters on the oversampled data. The results are as follows.

Cluster 0 has 33.8% of its observations of the class label '1' and cluster 1 has 99.9% of its observations of the class label '1'. That means cluster 1 is almost entirely built on observations of the class label '1'. This indicates that there is a natural grouping in the data between the target level classes and the 2-means clustering is able to identify that.

Fig7: Results of the dass proportion of target variable dasses after K Means Clustering

	Class		
ClusterID	0	1	Proportion(%)
0	284,280	145,436	33.8
1	35	67,923	99.9

Fig8: Scatter plot of observations against the first 2 independent variables. One cluster is red and the other is green



The above image shows the screenshot of a 2-means cluster solution with the cluster center being marked in yellow color. There appears to be an overlap in the clusters because of the high number of observations in them.

Modeling:

There are several machine learning models available in python and R. I have used Logistic Regression, Decision Trees and its variants Random Forest and Gradient Boosting, SVM and Neural Networks.

Logistic Regression:

Logistic Regression is a machine learning algorithm that assumes a linear decision boundary between the target classes and uses a logit function to map the relationship between input and target variables. The most used transfer function is sigmoid. It outputs the predicted probability of the target class of interest (minority, or '1'). With a prespecified cut-off, we can classify the outcome as an event or non-event, depending on whether the predicted probability exceeded or didn't exceed the cut-off.

The model's overall ability to classify the labels correctly is shown by the area under the ROC curve. As discussed in the previous readings, accuracy is not the correct evaluation metric. Since we are more interested in fraud than on non-fraud, recall is the metric of interest, followed by AUC and precision.

To understand the consequences of following inaccurate procedures while we have a class imbalance, such as no sampling, and no data preprocessing (no standardization), we have built a logistic regression model (elastic net penalty and l1 ratio = 0.5) on training data (75% of randomly split whole data) and validated its performance on the validation data (25% of randomly split whole data).

Fig9: Results of Logistic Regression

Accuracy: 0.998
F - Score: 0.044
Recall: 0.0244
Precision: 0.214

The above statistics depict the model performance on the validation data. We can see that the model has extremely high accuracy, almost 0 sensitivity, and very low precision. This very high accuracy should not mislead us because it is just caused by the class imbalance in the validation data. The model has no real generalization ability and no real classification ability. The true positives are almost 0, and every case is classified as a true negative. It is the worst possible model that can be built. Using techniques like sampling to adjust for class imbalance while training the model and using standardized data should help us with increasing the performance of the models.

We have built 3 logistic regression models on the 3 differently sampled data (undersampled data, oversampled data and SMOTE data). All 3 sampling techniques are done on standardized data. Standardization makes the variables have 0 mean and a standard deviation of 1. This will remove the undue effect of large variables like 'Amount' exert high influence on model parameters, especially when we do regularization. I then evaluated each model on the test data that I have set aside for the model's true test performance evaluation.

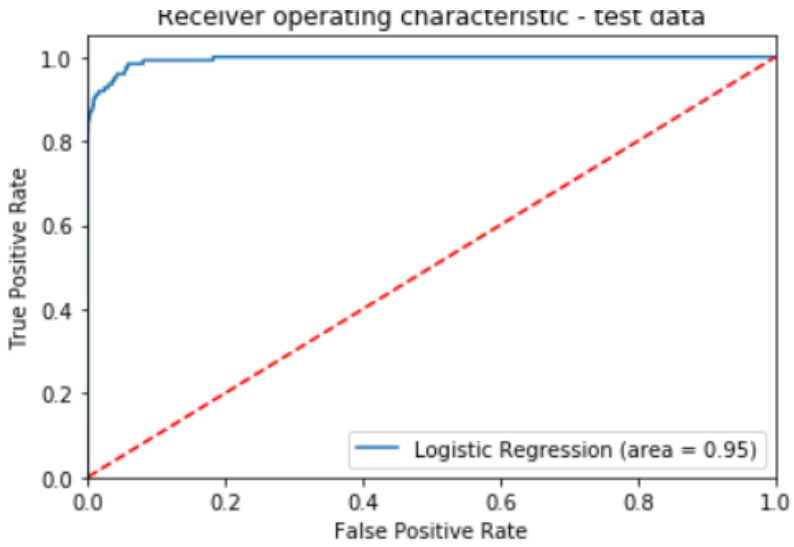
Based on the results, it can be seen SMOTETomek sampled data gave the best test results. Now, I used a 10-fold cv to evaluate the test performance of the model. The reason behind using a different method to evaluate the test performance is that it enables the model to use all the information in the data.

Fig10: Results of Logistic Regression with different Sampling

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Training data	0.955	0.92	0.99	0.95	0.96	Logistic Regression Model on Random Under sampled data
Validation data	0.953	0.93	0.03	0.06	0.94	
	Accuracy	Recall	Precision	F1 Score	AUC	
Training data	0.946	0.91	0.98	0.94	0.95	Logistic Regression Model on Random Over sampled data
Validation data	0.977	0.92	0.07	0.12	0.95	
	Accuracy	Recall	Precision	F1 Score	AUC	
Training data	0.942	0.97	0.91	0.94	0.94	Logistic Regression Model on SMOTETOMEK Oversampled data
Validation data	0.975	0.92	0.06	0.11	0.95	

The below image shows the ROC image of the Logistic Regression model (elastic net regularization and l1 ratio = 0.5) on the unsampled standardized validation data. We can get a recall of up to 85% with no False Positives, which is a great classification performance.

Fig11: ROC of Logistic Regression model (elastic net regularization and l1 ratio = 0.5) on the unsampled standardized validation data



I wanted to try a different technique to evaluate the true test performance of the model other than using the holdout method. So, I did 10-fold cross-validation. From the results, we have a cross-validated accuracy of 97.3%, recall of 87.6% and a precision of 6%. Note that the validation results of the model might depend on the initial seed of splitting the data. So, if we change the seed and do the splitting again, the results might differ a bit because we have only 492 positive events. Cross-validation suffers from this problem to a lesser extent, because it is averaging over 10 folds. Also, cross-validation lets the model using all the data to evaluate the test performance.

Fig12: Logistic Regression with Regulation Cross Validation Results

	Logistic Regression (solver = 'saga', penalty = 'elasticnet', l1_ratio = 0.5, random_state=0)									
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10

Accuracy	0.976	0.978	0.977	0.987	0.983	0.953	0.959	0.971	0.972	0.978
Recall	0.980	0.633	1.000	0.796	0.816	0.920	0.960	0.898	0.939	0.816
Precision	0.067	0.048	0.069	0.098	0.076	0.033	0.039	0.050	0.055	0.060
F-score	0.125	0.089	0.129	0.174	0.139	0.064	0.076	0.095	0.103	0.112

To understand how the logistic regression model would perform without any form of regularization, I did build a model and cross-validated it to evaluate the test performance. The results are shown below. We have a cross-validated accuracy of 98%, recall of 87.4% and a precision of 8%.

Comparing the cross-validation results of a regularized mode and a non-regularized model, we can see that recall was the same, while accuracy and precision improved for a non-regularized model. That means that the model is not overfitting on the training data, and it has a good generalization capacity.

Fig13: Logistic Regression without Regulation Cross Validation Results

	Logistic Regression (random_state = 0)									
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
Accuracy	0.980	0.986	0.983	0.991	0.988	0.959	0.971	0.976	0.980	0.982
Recall	0.980	0.694	1.000	0.755	0.816	0.900	0.960	0.878	0.939	0.816
Precision	0.077	0.080	0.091	0.129	0.104	0.038	0.055	0.061	0.075	0.075
F-score	0.143	0.144	0.166	0.220	0.185	0.072	0.105	0.113	0.139	0.137

Now that logistic regression has been performed, we shall explore what variables contribute to the explanation of the target.

Fig14: Logistic Regression Coefficients

Parameter	Coefficient	Parameter	Coefficient
V1	0.850743511	V16	-0.471335739
V2	0.12183352	V17	-0.711196904
V3	0.322697138	V18	-0.231815016
V4	1.195708865	V19	0.159531223
V5	0.566251504	V20	-0.235141214
V6	-0.449836422	V21	0.002338342
V7	-0.110689239	V22	0.249670167
V8	-0.599365047	V23	0.086124134
V9	-0.367919453	V24	0.05271161
V10	-1.014674845	V25	-0.024797707
V11	0.648154907	V26	-0.203533643
V12	-0.986493698	V27	0.096604072
V13	-0.252560252	V28	0.158704329
V14	-1.35655148	Amount	0.780569084
V15	0.085157871		

The above table shows the parameter estimates for each of the 29 features in our data. A few features have positive coefficients, and a few have negative coefficients, thereby affecting the target in both positive and negative ways.

Decision Trees:

Decision trees are a supervised non-parametric technique to solve regression and classification problems. They do employ a recursive partitioning method to identify variables that provide the best possible split of the target variable in the input space. This approach is greedy because it just looks at a partition that gives the best split, rather than looking at a partition that might giver better splits in the succession. So, this inherent structure of trees makes them highly flexible. They can be very accurate

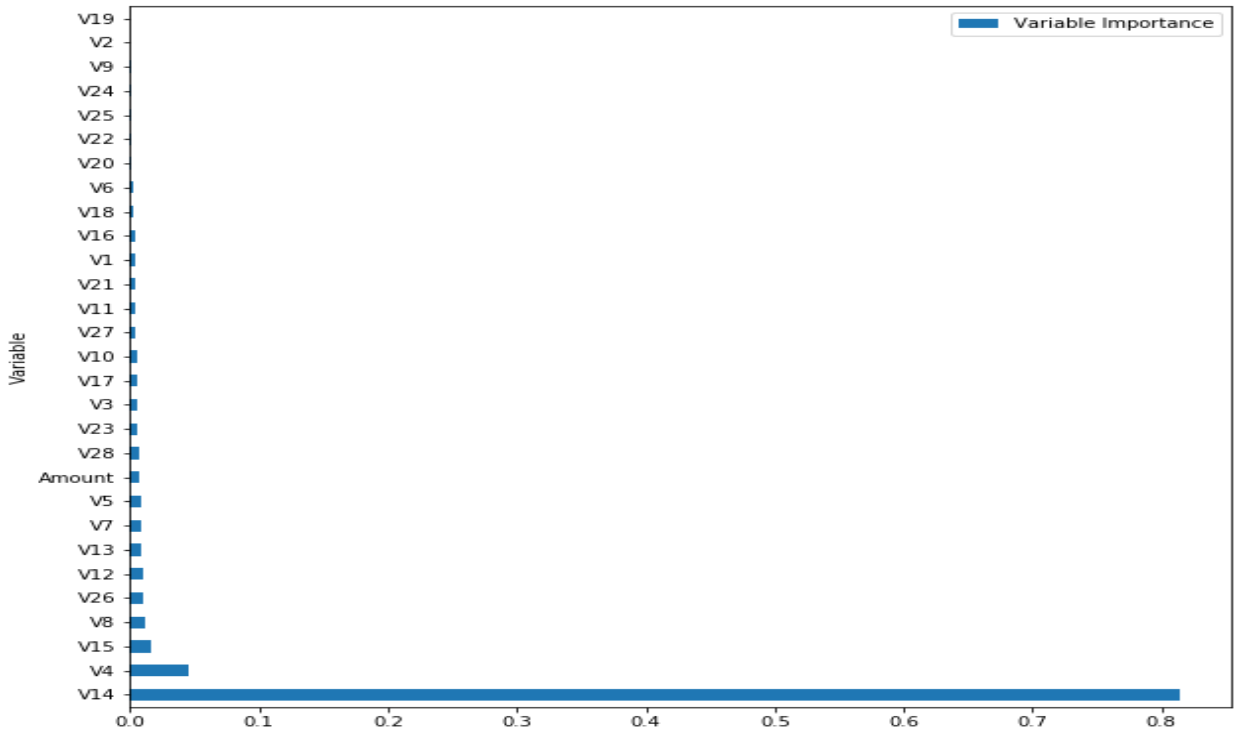
though. To solve this overfitting problem, we can prune the trees or control their growth using certain parameters. Specifying the maximum depth of trees or specifying a certain number of observations needed per split or constraining the leaf nodes to have a certain number of observations can prevent the wild growth of trees.

In this project, I have used a tree with a maximum depth of 8. I am using SMOTETomek oversampled and standardized data for training and standardized unsampled data for validation. Since we are dealing with 412569 observations, the tree we build will be highly overfitting. So, I am using a maximum depth of 8 to prevent the problem.

Fig15: Variable Importance table through Decision Trees

Variable	Variable Importance	Variable	Variable Importance
V1	0.003559407	V16	0.003389735
V2	0.000121276	V17	0.005952725
V3	0.006040017	V18	0.002657552
V4	0.045895146	V19	0.000103775
V5	0.007933762	V20	0.001444311
V6	0.002129155	V21	0.003602076
V7	0.008612809	V22	0.001320566
V8	0.011186383	V23	0.006300505
V9	0.000488152	V24	0.000574495
V10	0.005192327	V25	0.00087985
V11	0.003828224	V26	0.010725611
V12	0.009484185	V27	0.004758319
V13	0.009163463	V28	0.006721908
V14	0.813799877	Amount	0.007810566
V15	0.016323824		

Fig16: Variable importance chart of the decision tree model with a depth of 8



The above graph shows the variable importance of each variable in the tree building process. V14 is the variable with the most explanatory power followed by V4 and V15. V19 offers the least explanatory power.

Fig17: Results of Decision tree model

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Training data	0.980	0.97	0.98	0.98	0.98	Decision Tree Model trained on SMOTETomek over sampled data (maximum depth = 8)
Validation data	0.980	0.89	0.06	0.12	0.94	

From the above results, we can see that the Decision Tree is doing great but is having a poor precision.

Fig18: ROC on the validation data of a decision tree model with depth 8

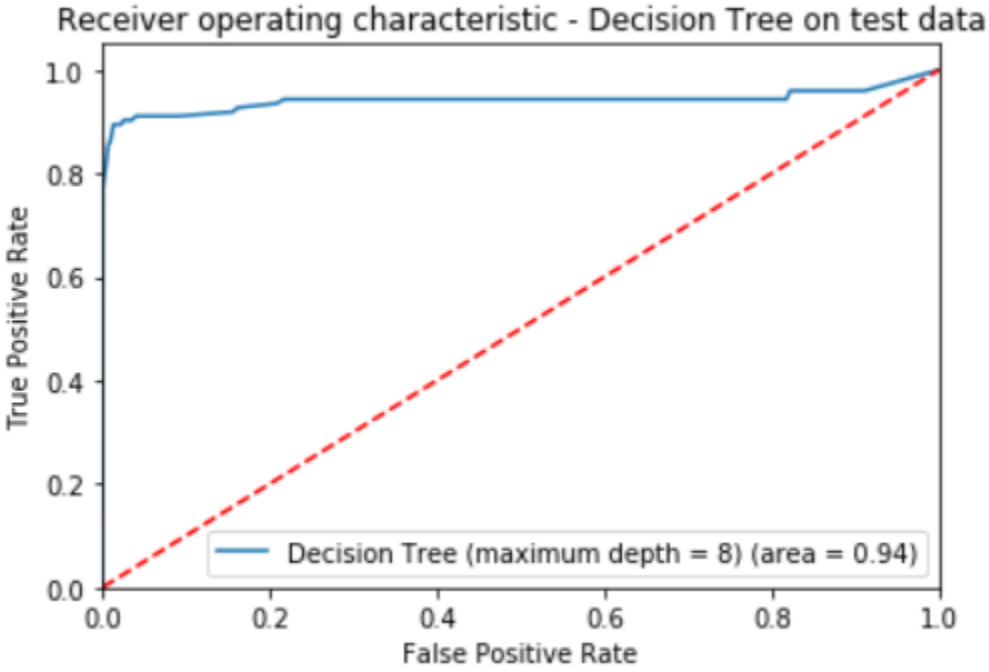


Fig19: Decision Tree Model Cross Validated Results

	Decision Tree Model trained on SMOTETomek oversampled data, cross-validated results									
Folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
Recall	0.837	0.469	0.918	0.694	0.755	0.700	0.800	0.653	0.694	0.633

The average cross-validated recall is 71.5%. This is much lesser than the training recall. It means that the decision tree model is overfitting on training data.

Decision Trees, by the very nature they are built (recursive binary splitting) are binary in nature and are prone to overfitting. To prevent this problem, bagging and boosting techniques are developed. Bagging is used to reduce the overfitting of trees and boosting is developed to improve the accuracy (or reduce the bias)

Random Forest:

We have discussed that individual decision trees are highly variable, even though they may have a low bias. Random Forest is a type of bagging procedure applied to full-grown decision trees to reduce the variance of trees. It builds several trees in parallel on bootstrapped training data and for each tree, it allows only a limited number of features to split the nodes. This tweaking is used to decorrelate the trees and induce randomness into the procedure. This will improve the bias and reduce the variance of the estimates.

Fig20: Results of Random Forest Model

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Training data	1.00	1.00	1.00	1.00	1.00	Random Forest Model trained on SMOTETomek oversampled data, with 100 estimators and sampling with replacement
Validation data	0.9995	0.79	0.82	0.80	0.99	

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Training data	1.00	1.00	1.00	1.00	1.00	Random Forest Model trained on SMOTETomek oversampled data, with 200 estimators and sampling with replacement
Validation data	0.9995	0.77	0.95	0.85	0.9991	

Fig21: ROC curve of a Random Forest model on validation data

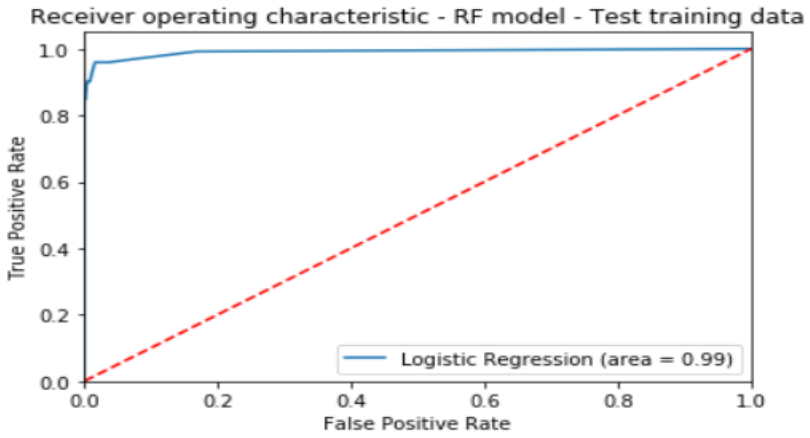
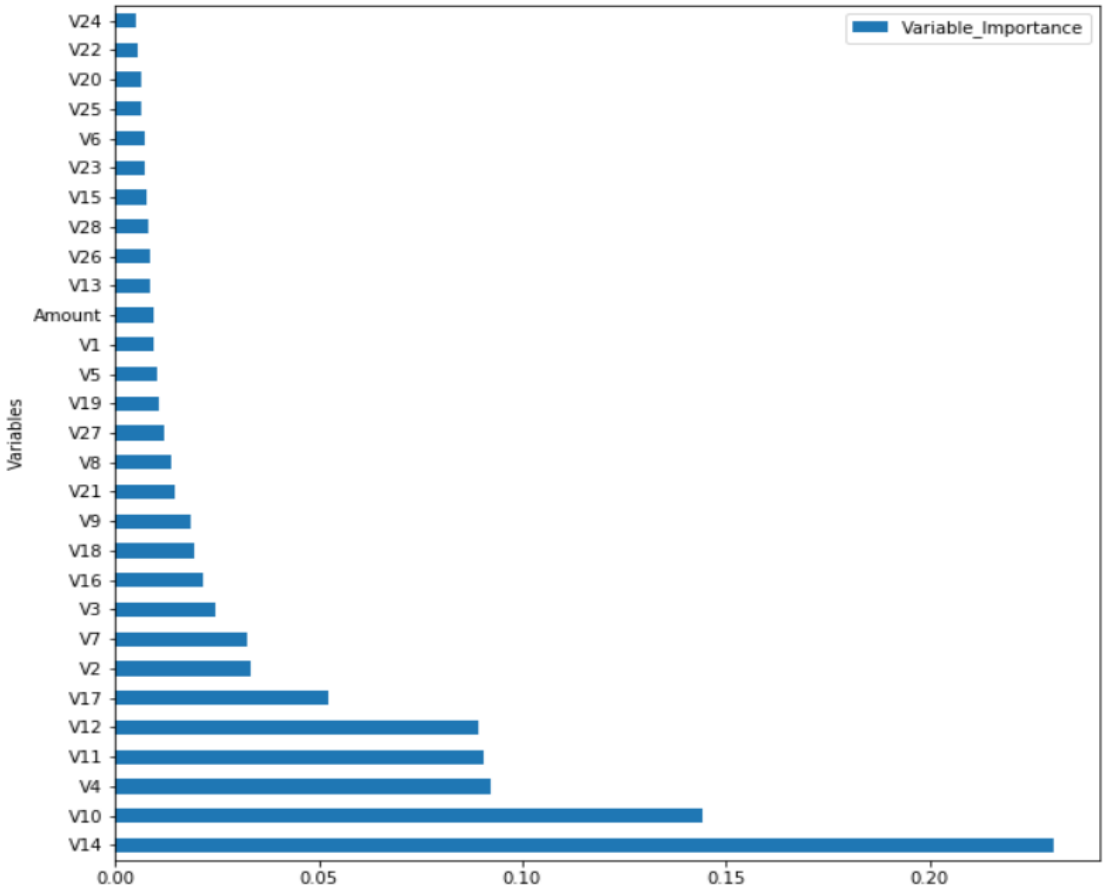


Fig22: Variable Importance chart of a random forest model



From the feature importance of the Random Forest model, we can see that the importance of variables is more spread, and no single variable entirely dominates the

picture. This is caused by the decorrelation of trees and not allowing all the features to take part in the bagging process.

Increasing the estimators has made the model a little more precise and a little less sensitive to true positives. With 200 iterations, the feature importance shifted. This means that a random forest tree needs optimization on the number of estimators. Or, the feature importance doesn't really matter, as we have all features that have really good explanatory power.

Random forests don't need any cross-validation since we are bagging the trees. This will eliminate overfitting. Also, there is not much difference in the training and validation metrics.

Gradient Boosting:

Bagged trees like Random Forests provide a good start towards reducing the variability of the estimates of individual decision trees. Boosting offers an even more theoretical advantage of reducing the bias at the cost of a little overfitting. Trees are built sequentially, instead of parallelly as in bagging, and each tree tries to correct the errors of its previous version. This procedure, if allowed to run for sufficiently long iterations, will wipe out the training bias. It may overfit if the number of trees goes too high. Gradient Boosting is a boosting procedure that doesn't suffer from the overfitting problem while maintaining very low bias.

The learning rate in gradient boosting is a very crucial hyperparameter that dictates how much deep a tree can move to correct the errors of its predecessor. I built 7 gradient boosting models for 7 different learning rates ([0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]) and found that learning rate of 1 is ideal in our case. My judgment is based on validation accuracy and the difference between training and validation accuracy (no overfitting/underfitting).

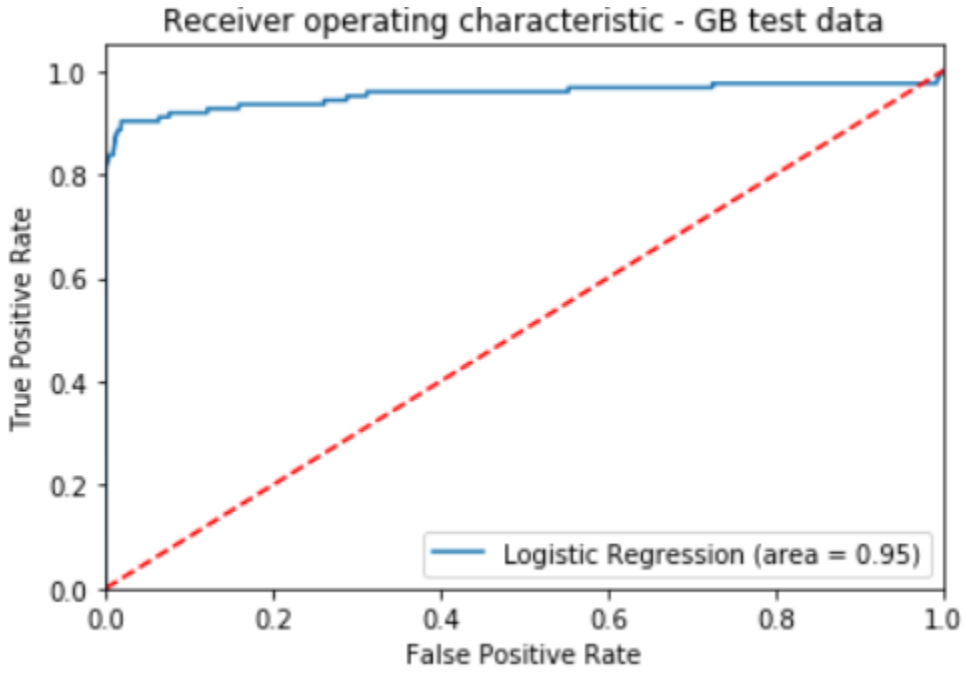
Fig23: Results of Gradient Boosting

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Validation data	0.980	0.85	0.32	0.46	0.96	Gradient Boosting Model trained on SMOTETomek over sampled data (n_estimators=100, subsample = 0.8, max_features="sqrt", max_depth=3)
	Accuracy	Recall	Precision	F1 Score	AUC	
Validation data	0.970	0.93	0.06	0.11	0.99	Gradient Boosting Model trained on SMOTETomek over sampled data (n_estimators=100, subsample = 0.85, max_features="sqrt", max_depth=1)
	Accuracy	Recall	Precision	F1 Score	AUC	
Validation data	0.990	0.82	0.51	0.63	0.95	Gradient Boosting Model trained on SMOTETomek over sampled data (n_estimators=100, subsample = 0.85, max_features="sqrt", max_depth=4)
	Accuracy	Recall	Precision	F1 Score	AUC	
Validation data	0.990	0.86	0.17	0.28	0.97	Gradient Boosting Model trained on SMOTETomek over sampled data (n_estimators=50, subsample

						= 0.5, max_features=2, max_depth=4)
	Accuracy	Recall	Precision	F1 Score	AUC	
Validation data	0.990	0.82	0.69	0.75	0.96	Gradient Boosting Model trained on SMOTETomek over sampled data (n_estimators=100, subsample = 1, max_features=2, max_depth=8)

From the above table, we can see that playing with the hyperparameters of the gradient boosting model will allow us the flexibility to optimize any metric, say recall or precision. Increasing the depth of the trees (growing bigger trees) is making the model more precise and less sensitive to true positives. It makes sense as we make our trees grow like the way grow trees in random forests (full trees).

Fig24: ROC curve of a Gradient Boosting model on validation data



Support Vector Machines:

SVM is a technique that utilizes the natural separation of the data dictated by the target variable classes. It can be used in regression and classification problems. The idea is to separate the data containing n features by an n-1 dimensional hyperplane so that data belonging to one target class falls in one direction and to the other target class in the other direction. When there is no clear separation among the data to build a hyperplane, we can use kernels which are functions that transform the data to higher dimensions, where there might be clear separation among the data.

I have built SVM using several possible kernels and C parameters on the randomly undersampled data. Here are the results. The results do not come close to the results from other models.

Fig25: Results of SVM model

	Accuracy	Recall	Precision	F1 Score	Model Type (Linear Models)
Training data	0.950	0.91	0.99	0.95	SVM Model on Random Under sampled data, using Linear kernel and C = 0.1
Validation data	0.967	0.93	0.05	0.09	
	Accuracy	Recall	Precision	F1 Score	
Training data	0.955	0.92	0.99	0.95	SVM Model on Random Under sampled data, using Linear kernel and C = 0.5
Validation data	0.963	0.93	0.04	0.08	
	Accuracy	Recall	Precision	F1 Score	

Training data	0.957	0.92	0.99	0.95	SVM Model on Random Under sampled data, using Linear kernel and C = 1
Validation data	0.962	0.92	0.04	0.08	
	Accuracy	Recall	Precision	F1 Score	Model Type (Sigmoid Models)
Training data	0.923	0.88	0.97	0.92	SVM Model on Random Under sampled data, using Sigmoid kernel and C = 1
Validation data	0.973	0.89	0.05	0.1	
	Accuracy	Recall	Precision	F1 Score	
Training data	0.901	0.8	1	0.89	SVM Model on Random Under sampled data, using Sigmoid kernel and C = 0.1
Validation data	0.999	0.8	0.8	0.8	
	Accuracy	Recall	Precision	F1 Score	
Training data	0.901	0.8	1	0.89	SVM Model on Random Under sampled data, using Sigmoid kernel and C = 0.05
Validation data	0.999	0.83	0.79	0.81	

K Nearest Neighbors:

KNN is a non-parametric approach that is used for classification and regression tasks. It is a simple algorithm, where it takes the K nearest neighbors of a data point and assigns the mode of the class of the K nearest neighbors' classes. The closeness is calculated by several distance metrics depending on the data type. Euclidean, Mahalanobis, Jaccard are a few metrics. This algorithm assumes no functional dependency between the variables and hence is efficient in low dimensional settings when there is no true functional relationship between the predictors and targets. The ideal value of K is to be chosen by cross-validation.

I have applied KNN on the randomly under sampled standardized data and validated on the standardized unsampled validation data. Since the value of K is to be chosen and not known prior, I have plotted the values of testing accuracy, recall, and precision for 100 values of K, from 1 to 100.

Fig26: Validation accuracy of the KNN model against K

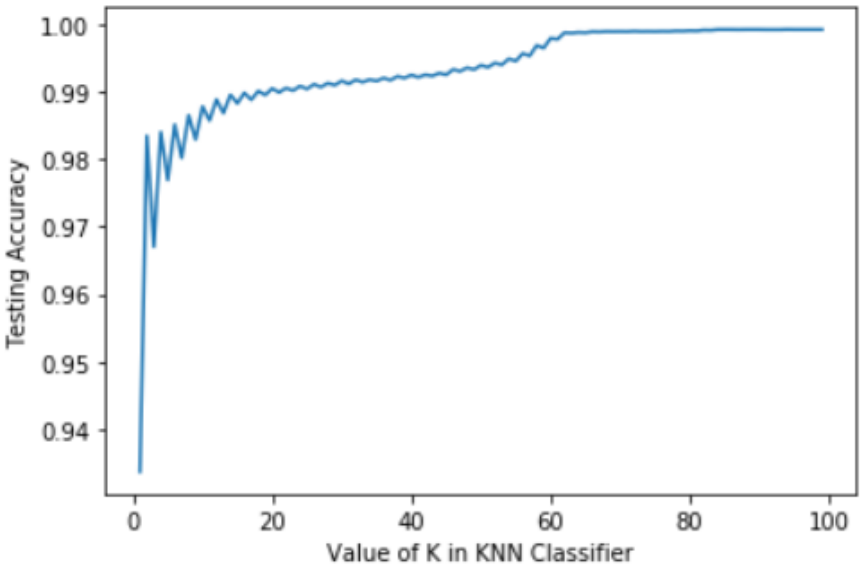


Fig27: Validation precision of the KNN model against K

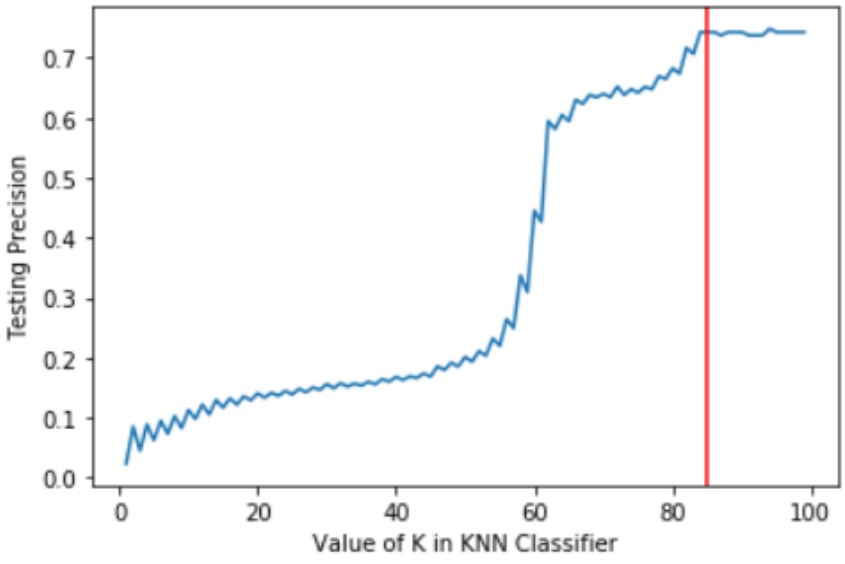
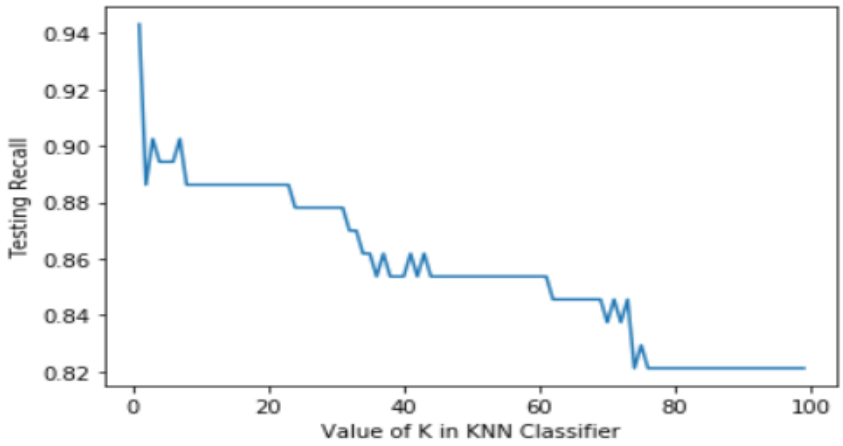


Fig28: Validation recall of the KNN model against K



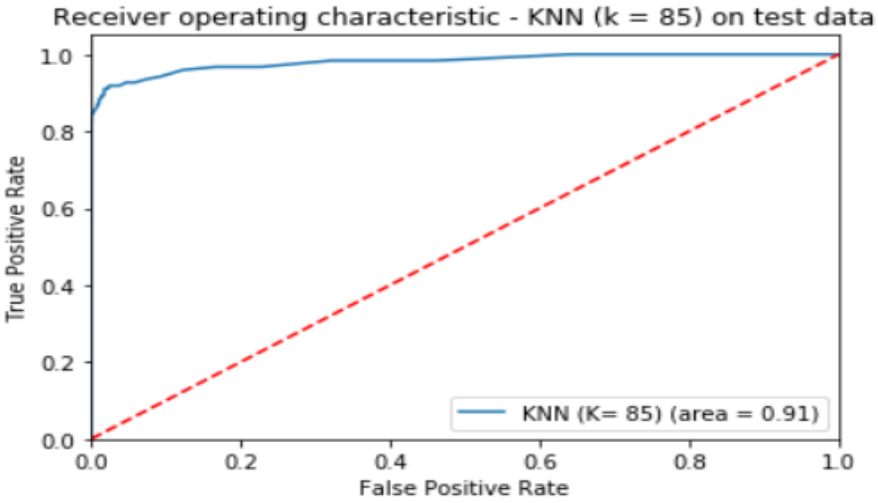
We can see that the accuracy of our classifier on testing data is increasing and asymptotically reaching 1 as we go up in K, and the exact reverse pattern is observed in recall i.e., it is decreasing albeit not as smooth, as we go up in K. But there is a sigmoid shaped increase in precision as we go up in K. Selecting the value of K from the above 3 graphs requires subjective judgment. I will pick a value of k =85, because it is where the recall is flat, and precision is improving. If I choose a higher k, there is no reduction in the recall, but the computational load on the machine will increase. If I choose lower K, the precision will drop below 70%.

Fig29: Results of KNN model with 30 neighbors

	Accuracy	Recall	Precision	F1 Score	Model Type
Training data	0.913	0.83	1.00	0.91	KNN trained on random undersampled data, with 30 neighbors
Validation data	0.999	0.82	0.74	0.78	

Training the KNN with 85 neighbors gave us a good precision and recall, comparable to the random forest model.

Fig30: Validation ROC of the KNN (k=85) model



Assessing the Models:

Based on all the models built, the Random Forest model is identified as the best one based on validation recall, AUC and precision. No other models other than KNN come close to random forests when it comes to balancing the validation recall and validation precision. All other models are able to capture the true positives perfectly, but also are taking an overly cautious approach and classifying several false positives as true positives. This reduces the confidence in our model's predictions. This can be seen from the value of validation precision at 7% for the majority of the models. Random forest with 200 trees in parallel has a validation recall of 0.77 and a validation precision of 0.95. Even though the recall is not as high as other models the precision

is much higher, thus giving the business much higher confidence in flagging fraud cases.

Fig31: Results of the best model

	Accuracy	Recall	Precision	F1 Score	AUC	Model Type
Training data	1.00	1.00	1.00	1.00	1.00	Random Forest Model trained on SMOTETomek oversampled data, with 200 estimators and sampling with replacement
Validation data	0.9995	0.77	0.95	0.85	0.9991	

The champion model has all perfect training metrics. That means it is explaining the training data perfectly. The validation accuracy is 0.9995 and validation AUC is 0.9991. Both the values are very close to 1, which means that the model is not overfitting.

We can safely deploy this model in production. We can confidently say that our model is 99.91% accurate in identifying class labels, has the ability to find 77% of the fraud cases with a 95% precision in identifying the true positives.

Conclusion:

The problem at hand is a unique one. We have a huge classification task but directly going to modeling is not the correct way. First, we dealt with class imbalance, then data standardization. Now, another huge task is choosing the right metric to evaluate models.

For data with a class imbalance in the binary target, accuracy is not the perfect metric to judge the models. Especially when the cost of false positive and false negative is not the same. In our case, it is more important for us identifying the true positives than true negatives and it is more important to us to reduce the number of false-negatives than false positives. So, recall should be a better metric for us to judge the model than accuracy or AUC. However, AUC and precision are also better metrics than accuracy when it comes to deciding the discriminatory power of a model.

References :

1. Kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud)
2. Wikipedia(Wikipedia.org)
3. Towards Data Science (http://towardsdatascience.com/)