DISSERTATION PROJECT

NAME: TANUJ SUR

ROLL NO. : 469

SUPERVISOR: PROFESSOR DEBJIT SENGUPTA

TITLE: SANTANDER CUSTOMER TRANSACTION PREDICTION

I affirm that I have identified all my sources and that no part of my dissertation paper uses acknowledgement materials.

Signed,

Tanuj Sur

CONTENTS

INTRODUCTION

In this data driven world, data science and statistical models are used to predict a variety of outcomes like whether a patient has a particular disease or not, where to establish a business, what products to keep in a store so that revenue is maximum and so on. In this dissertation we aim to solve a real world problem similar to these. Santander is a Spanish multinational commercial bank which helps its customers understand their financial health and identify products and services which might help them achieve their monetary goals. The data on two lakh anonymous customers released by Santander is used in my study. However the data does not specify what these 200 data heads represent. This must have been done keeping in mind the privacy of the users and the data source.

The response variable, i.e. whether a customer makes a transaction or not, being binary (i.e. 0 and 1) makes this a classification problem. Unlike regression, where the output variable takes any value on the real line, classification problems involve categorical or nominal categories. We wish to fit a logistic classifier on this data followed by random forest algorithm. In a logistic classifier, a nonlinear function followed by a decision rule is used to predict the response variable. However in case of random forests, an ensemble of decision trees is used. Mode of the classes predicted by the individual decision trees is the final output. We compare the results in each of these algorithms and finally try to present the best model for this problem in hand.

LOGISTIC REGRESSION

Logistic regression is a classification model that is very easy to implement and is one of the most widely used algorithms for classification in industry. Logistic regression model is a linear model for binary classification that can be extended to multiclass classification via the One-vs.-Rest (OvR) method.

To explain the idea behind logistic regression as a probabilistic model, let's first introduce the odds ratio, which is the odds in favor of a particular event. The odds ratio can be written as $\frac{p}{1-p}$, where p stands for the probability of the positive event. The term positive event does not necessarily mean good, but refers to the event that we want to predict, for example, in this case, the probability that a customer will make a transaction or not. We can think of the positive event as class label $y = 1$. We can then further define the logit function, which is simply the logarithm of the odds ratio (log-odds):

$$logit(p) = \log(\frac{p}{1-p})$$

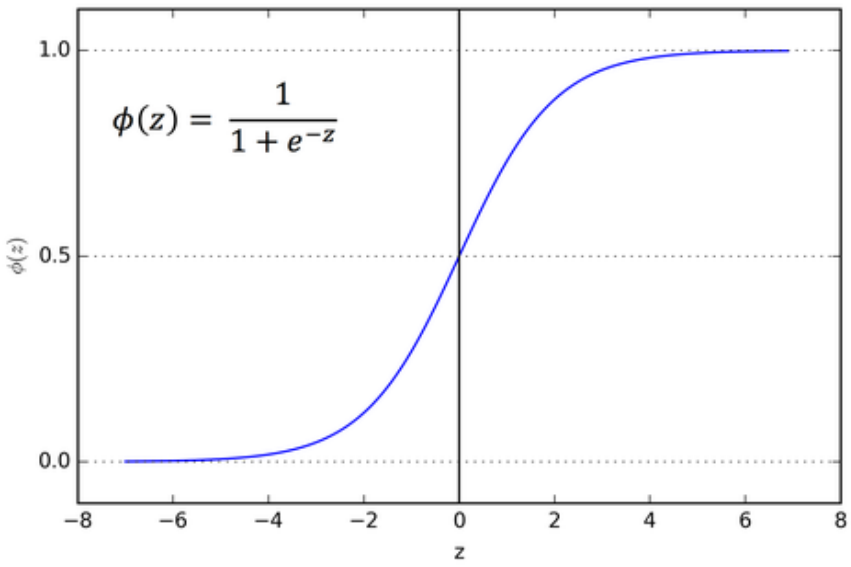
The logit function takes input values in the range 0 to 1 and transforms them to values over the entire real number range, which we can use to express a linear relationship between feature values and the log-odds:

$$logit\big(p(y = 1|x)\big) = w^T\, x$$

Here $p(y = 1|x)$ is the conditional probability that a particular sample belongs to class 1 given its features x. We are actually interested in predicting the probability that a certain sample belongs to a particular class, which is the inverse form of the logit function. It is also called the logistic function, simply abbreviated as sigmoid function due to its characteristic S-shape.

$$\emptyset(z) = \frac{1}{1 + e^{-z}}$$

Here, z is the net input, that is, the linear combination of weights and sample features and can be calculated as $z = w^T\, x$.



We observe that $\emptyset(z)$ approaches 1 if z goes towards infinity ($z \rightarrow \infty$), since e^{-z} becomes very small for large values of z. Similarly, $\emptyset(z)$ goes towards 0 for $z \rightarrow -\infty$ as the result of an increasingly large denominator. Thus, we conclude that this sigmoid function takes real number values as input and transforms them to values in the range [0, 1] with an intercept at $\emptyset(z) = 0.5$. The output of the sigmoid function is then interpreted as the probability of particular sample belonging to class 1 $\emptyset(z) = p(y = 1|x; w)$ given its features x parameterized by the weights w.The predicted probability can then simply be converted into a binary outcome via a indicator function:

$$y = \begin{cases} 1 \; if \; \emptyset(z) \geq 0.5 \\ 0 \; otherwise \end{cases}$$

If we look at the preceding sigmoid plot, this is equivalent to the following:

$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; otherwise \end{cases}$$

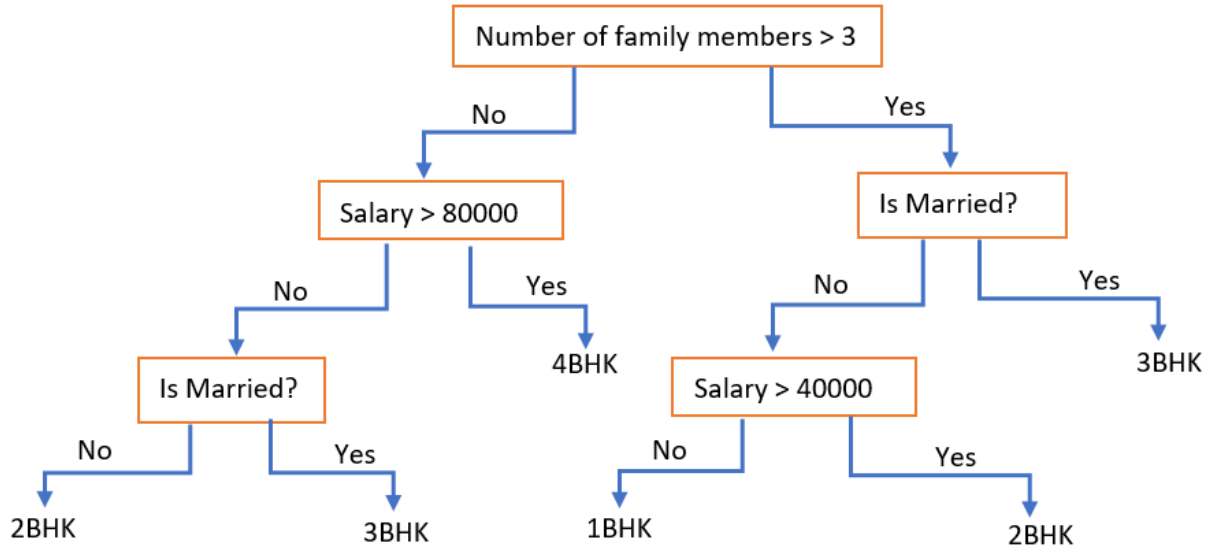
The Python code for fitting the given dataset with logistic regression is given by:

```
>> from sklearn.linear_model import LogisticRegression
>> lr = LogisticRegression()
>> lr.fit(X_train, Y_train)
>> y_pred = lr.predict(X_test)
```

The dataset has been split into training and testing data. X_train and Y_train are the predictor and response values respectively corresponding to training data. X_test is the predictor variable corresponding to the testing data.

RANDOM FOREST ALGORITHM

Random Forests are simply a collection of Decision Trees that have been generated using a random subset of data. The name "Random Forest" comes from combining the randomness that is used to pick the subset of data with having a bunch of decision trees, hence a forest. A Decision Tree is simply a step by step process to go through to decide a category something belongs to. An e.g. of a decision tree is given below:



The above decision tree attempts to find a flat for a family based on the number of members, salary and marital status of its members. To use the decision tree, we start at the top and begin hitting each decision in series. At each point, we need to make a choice on which way to go between exactly two options. Eventually we reach the bottom and have a decision as to the outcome. Most severe limitations are the tendency for decision trees to over-fit their data. In many real world examples, it can be challenging to classify things based on the data that is given. We can have messy data and anomalies that don't generalize to the real world. A decision tree will not smooth out those anomalies. If we are trying to tell the difference between two object say oranges and grapefruit based on their size, a decision tree might break the data down into very small, specialized ranges that work for our data, but not for any random fruit that might come in. Random Forests attempt to fix this problem by using multiple decision trees and averaging the results. However, just generating multiple decision trees using the same set of data over and over is not beneficial since we will just get multiple copies of the same decision tree. (Or at least very similar decision trees, depending if there is any randomness in their creation or not). So Random Forests generate their decision trees using subsets of the full data set that are randomly selected. One of the limitations of using random forest is that they don't extrapolate.

The randomness in a Random Forest

All of the Decision Trees in a Random Forest use a slightly different set of data. They might be similar, but they are not the same. The final result is based on the votes from all the decision trees. The outcome of this is that anomalies tend to get smoothed over, since the data causing the anomalies will be in some of the decision trees, but not all of them, while the data that is more general will be in most if not all of the trees. When generating each tree, that tree has a unique set of data. That set is generated from a random subset of all of the available data, with replacement. This technique is known as bootstrapping. Each of the trees uses a set of data that is the same size of the original data set.

If we run this random sampling enough times, with large enough data sets, we will find that on average 63.2% of the original data set is in each tree. Since each tree is the same size as the original data set, that means the other 36.8% is duplicates of the original data set. Since most Random Forests contain anywhere from a few dozen to several hundred trees, then it is likely that each piece of data is included in at least some of the trees.

Number of trees In a Random Forest

More trees are usually better because they will do more to smooth out abnormalities in the data. But that is true only up to a point. This is an area of diminishing returns, where each additional tree will have less benefit than the one before it. Eventually the benefit will flatten, and more trees will not be useful much. The decision on how many trees to have in the Random Forest becomes a tradeoff dependent on our problem and our computing resources. Going from 10 trees to 100 trees might improve your results significantly, and not add very much time. However going from 1,000 trees to 50,000 trees might add substantial time without improving our results very much.

Criteria Selection

The other way that a random forest adds randomness to a decision tree is deciding which feature to split the tree on. Within any given feature, the split will be located at the location which maximizes the information gain on the tree, i.e. the best location. Additionally, if the decision tree evaluates multiple features, it will pick the best location in all the features that it looks at when deciding where to make the split. So if all of the trees looked at the same features, they would be very similar. The way that Random Forest deals with that is to not let the trees look at all of the features. At any given branch in the decision tree, only a subset of the features is available for it to classify on. Other branches, even higher or lower branches on the same tree, will have different features that they can classify on. Random Forest chooses best the *best location* to split the tree based on two measures namely ‘Gini impurity’ and ‘Information Gain’.

Gini Impurity

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

To compute Gini impurity for a set of items with J classes, suppose $\mathbf{j} \in \{1, 2, \ldots, J\}$

$$Gini = 1 - \sum_j p_j^2$$

Where $\mathbf{p_j}$ be the fraction of items labeled with class j in the set.

Based on the feature with the lowest Gini impurity, the decision tree is split at this location. The best value that we could have is an impurity of 0. That would occur if we had a branch that is 100% single class.

Information Gain

Information gain is based on the concept of entropy and information content from information theory. Entropy is defined as

$$Entropy = -\sum_j p_j \log_2 p_j$$

Entropy is more computationally heavy due to the log in the equation. Like Gini, the basic idea is to gauge the disorder of a grouping by the target variable. Instead of utilizing simple probabilities, this method takes the log base 2 of the probabilities. The entropy equation uses logarithmic function which has several elegant resulting properties, and one of these properties is additivity. For entropy, just like the Gini criteria, the lower the number the better it is with the best being an Entropy of zero. For this equation, for each probability, we are multiplying that probability by the base 2 logarithm of that probability. Since each of these probabilities are a real number between 0 and 1, the base 2 logarithm will always be negative (or zero), which when multiplied by the negative sign in the equation will give a positive number for the total entropy summation.

The Python code to implement Random Forest is as follows:

```
>> from sklearn.ensemble import RandomForestRegressor
>> m = RandomForestRegressor(n_estimators=30)
>> m.fit(X_train, y_train)
>> predictions = m.predict(X_test)
```

Here we observe that the number of tress chosen is 30. By hit and trial we find that the classifier works best when number of trees is 30. Increasing it consumes more time and there is hardly any significant improvement in accuracy. The variable *predictions* store the prediction for the test data. However this prediction is a real number between 0 and 1. Since our target variable is a binary variable i.e. 0 or 1, we choose a threshold to convert the real numbers into binary ones. We convert the predicted value into 1 if its >0.25 else 0. We observe that this threshold provides the maximum accuracy.

CORRECTING CLASS IMBALANCE

Datasets having unequal number of target variables for different classes are said to be imbalanced. This means that there is an imbalance between the classes in the dataset due to a large difference between the number of instances belonging to each class. The class having comparatively fewer instances than the other is known as **minority class** with respect to the class having a comparatively larger number of the samples known as a **majority** class. For example in the Santander Customer Transaction Prediction dataset, we observe that 90% of the data is on customers who did not make any transactions and 10 % of the data is on customers who made transactions. If any predictive model is used to predict the target variable based on this dataset, it will be definitely biased. The model will be prone to make predictions of the class which it interacts with the most i.e. the majority class. For this purpose we use the following algorithms to correct class imbalance.

Synthetic Minority Over-Sampling Technique (SMOTE)

In this method, we correct imbalanced datasets by oversampling minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. SMOTE is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature space. There is a percentage of Over-Sampling which indicates the number of synthetic samples to be created and this percentage parameter of Over-sampling is always a multiple of 100. If the percentage of Over-sampling is 100, then for each instance, a new sample will be created. Hence, the number of minority class instances will get doubled. Similarly, if the percentage of Over-sampling is 200, then the total number of minority class samples will get tripled. In our case, the majority class is 9 times more than the minority class. Hence oversampling percentage is 900.

In SMOTE,

- For each minority instance, k number of nearest neighbors is found such that they also belong to the same class.
- The difference between the feature vector of the considered instance and the feature vectors of the k nearest neighbors are found. So, k number of difference vectors is obtained.
- The k difference vectors are each multiplied by a random number between 0 and 1 (excluding 0 and 1).
- Now, the difference vectors, after being multiplied by random numbers, are added to the feature vector of the considered instance (original minority instance) at each iteration.

And that is how we increase the size of the dataset belonging to the minority class to match that of the majority class.

The Python code to implement SMOTE is as follows:

```
>> from imblearn.over_sampling import SMOTE

>> smt = SMOTE()

>> X_train, Y_train = smt.fit_sample(X_train, Y_train)
```

The limitation of this method is that, the synthetic data created is not always what the real data would have been. There might have been some other properties present which could be important for the prediction of the target variable. In this project, where the minority class is oversampled 9 times of its size, the patterns and properties of the minority class is projected by the synthetic data. Such a huge data having attributes of the minority class can be misleading at times.

Random Under-Sampling Technique

In random under-sampling technique the majority class size is reduced to match with the minority class size by randomly selecting elements from the majority class and removing them from the dataset. At the end we have two classes of equal number of samples. However, a major drawback of this method is that when the entries belonging to the majority class are randomly chosen and eliminated, it increases the risk of losing important information. Especially in datasets like the Santander Customer Transaction Prediction where the majority class size is 9 times more than that of the minority class, a lot of important features and information may be lost. This is detrimental to the accuracy of the model predictions and the validity of the model as well.

The Python code to implement this is as follows:

```
>> from imblearn.under_sampling import RandomUnderSampler

>> rus = RandomUnderSampler(random_state=0)
>> X_train, Y_train = rus.fit_resample(X_train, Y_train)
```

PRINCIPAL COMPONENT ANALYSIS

Principal component analysis (**PCA**) is an unsupervised linear transformation technique that is widely used across different fields, most prominently for dimensionality reduction. Other popular applications of PCA include exploratory data analyses and de-noising of signals in stock market trading, and the analysis genome data and gene expression levels in the field of bioinformatics. PCA helps us to identify patterns in data based on the correlation between features. In a nutshell, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions that the original one. The orthogonal axes (principal components) of the new subspace can be interpreted as the directions of maximum variance given the constraint that the new feature axes are orthogonal to each other. If we use PCA for dimensionality reduction, we construct a $d \times k$ dimensional transformation matrix $\boldsymbol{W}$ that allows us to map a sample vector $\boldsymbol{x}$ onto a new k -dimensional feature subspace that has fewer dimensions than the original d -dimensional feature space.

As a result of transforming the original d-dimensional data onto this new k-dimensional subspace (typically$k \ll d$), the first principal component will have the largest possible variance, and all consequent principal components will have the largest possible variance given that they are uncorrelated (orthogonal) to the other principal components. We need to note that the PCA directions are highly sensitive to data scaling, and we need to standardize the features *prior* to PCA if the features were measured on different scales and we want to assign equal importance to all features. We summarize the approach of applying PCA as follows:

1. Standardize the d-dimensional dataset.
2. Construct the covariance matrix.
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace ($k \leq d$).
5. Construct a projection matrix $\boldsymbol{W}$ from the "top" k eigenvectors.
6. Transform the d -dimensional input dataset $\boldsymbol{X}$ using the projection matrix W to obtain the new k-dimensional feature subspace.

Since we want to reduce the dimensionality of our dataset by compressing it onto a new feature subspace, we only select the subset of the eigenvectors (principal components) that contains most of the information (variance). Since the eigenvalues define the magnitude of the eigenvectors, we have to sort the eigenvalues by decreasing magnitude. We also need to keep in mind, how much of the total variance is explained by this k-dimensional feature subspace (Eigen values). The variance explained ratio of an eigenvalue λ_j is simply the fraction of an eigenvalue λ_j and the total sum of the eigenvalues.

The Python code to implement PCA is as follows :

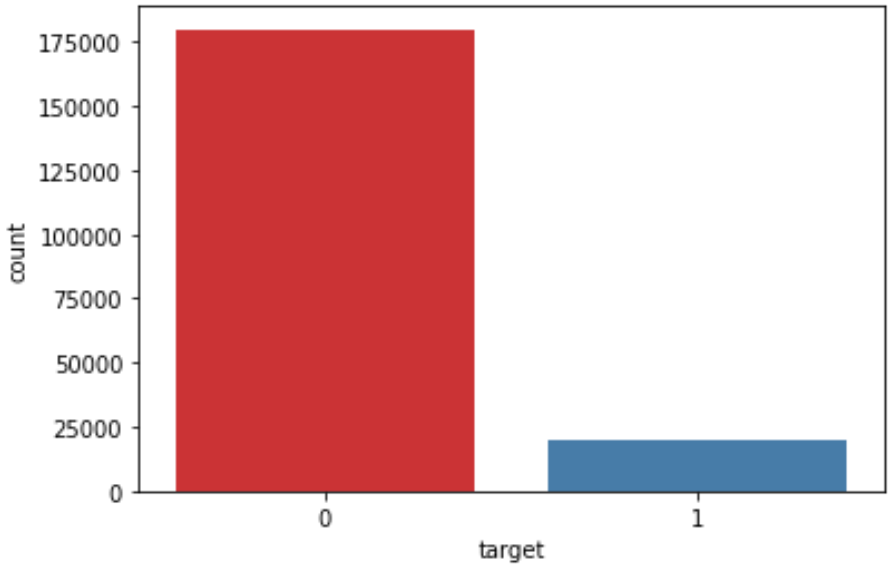
```
>> from sklearn.preprocessing import StandardScaler
>> from sklearn.decomposition import PCA

>> sc = StandardScaler()
>> X_train = sc.fit_transform(X_train)
>> X_test = sc.transform(X_test)

>> pca = PCA(.80)
>> X_train = pca.fit_transform(X_train)
>> X_test = pca.transform(X_test)
>> explained_variance = pca.explained_variance_ratio_
>> print(explained_variance)
```

In the above code, firstly we standardize the data. Then we reduce the dimension of the data through PCA ensuring that atleast 80% of the total variance is explained by the remaining data. As a result of this, dimension of the data is reduced from 198 to 158. We observed that for higher percentages of variance explained, there was hardly any reduction in the dimension of the dataset. Hence we chose 80% as our threshold.

ANALYSING THE DATA AND FITTING OF MODELS

Key features of Santander Customer Transaction dataset:

1. Total no. of entries = 200000
2. Total no. of predictor variables = 200
3. The predicted variable **Target** consists of binary data 0 and 1. The remaining 200 predictor variables are all numerical continuous data.
4. The minimum correlation observed between the predictor variables is 1.556144e-07 and the maximum correlation is 0.009844.
5. No null values present.
6. This dataset is imbalanced with respect to the **Target** column as shown below.



Key observations:

1. The predicted variable **Target** being categorical in nature makes this a classification problem. Our aim is to predict whether a customer makes a transaction or not.
2. The correlation between the predictor variables being so low indicates that they are independent of each other.
3. As no null values are present, the entire dataset is used for the purpose of modeling.
4. This dataset is skewed towards **Target** 0 as 90% of the data present belongs to that class compared to the data that belongs to the class 1.

Accuracy measures:

Confusion matrix: It is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one like linear regression, logistic regression and random forest. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class.

Confusion Matrix	**Predicted 0**	**Predicted 1**
Actual 0	True Positive	False Negative
Actual 1	False Positive	True Negative

Area under ROC curve: Area under the curve plotted between True positive (T_p) rate and false positive (F_p) rate. Higher the value for a model, better is the accuracy of the model.

Precision-Recall (PR) score will be more informative than ROC when dealing with highly skewed datasets. Because Precision is directly influenced by class imbalance so the Precision-recall scores are better to highlight differences between models for highly imbalanced data sets. When we compare different models with imbalanced settings, the area under the Precision-Recall curve will be more sensitive than the area under the ROC curve. Higher is its value for a model, better is the accuracy of the model.

Precision (P) = $T_p / (T_p+F_p)$

Recall (R) = $T_p / (T_p+F_n)$

Average precision-recall score = $\Sigma(R_n - R_{n-1})P_n$

Fitting of predictive models

- **Logistic Regression**:

As the prediction variable is categorical as well as binary in nature i.e. 0 if the customer makes a purchase and 1 if not, the first model that comes into our mind is Logistic regression. Since the dataset is imbalanced based on the target variable, first we will fit a logistic regression without correcting class imbalance and then by correcting it.

- Fitting on imbalanced dataset

On fitting a logistic regression on 80% of the data as training data and 20% as test data, the following things were observed:

1. Area under ROC curve = 0.625

2. Average precision-recall score = 0.259

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**35414**	**489**
Actual 1	**3004**	**1093**

Comment: As we can see from the ROC score that the fitting is not so good. From the Confusion matrix it is observed that in the test data (created by randomly selecting 20 % data), 35414 entries with target variable 0 are predicted correctly (true positive) while 489 entries are misclassified (false positive) as 1. 3004 observations with target variable 1 are predicted incorrectly as 0 (false negative) while 1093 observations are predicted correctly (true negative).

- Fitting of data after applying SMOTE

After applying SMOTE to the data, both the classes (0's and 1's) have a size of 1,43,999. Now a logistic regression is fitted on the training data and accuracy is measured on the test data. Following observations were made:

1. Area under ROC curve = 0.7749

2. Average precision-recall score = 0.2456

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**28258**	**7645**
Actual 1	**972**	**3125**

Comment: Though the Area Under ROC curve has increased considerably compared to the previous case, Average precision has decreased. From the Confusion matrix it is evident that no. of true positives has decreased and false positives have increased considerably. This shows that over sampling has led to over fitting of the data. It is interesting to observe that no. of true negatives have increased by leaps and bounds following the oversampling technique.

- Fitting of data after applying Random Under Sampling technique

Following observations were made after fitting a logistic regression to it:

1. Area under ROC curve = 0.7757

2. Average precision-recall score =0.2434

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**27973**	**7930**
Actual 1	**934**	**3163**

Comment: Number of true positives further decreases and no of true negatives increases. Clearly this method also results in over fitting of the model.

- **RANDOM FOREST**

In this case the prediction of random forest is on continuous interval. Since the target variable is binary, we convert the predictions to 1 if the predicted value is >0.25 else 0. This cut off value is chosen as it gives the best accuracy till now.

- Fitting on imbalanced dataset

On fitting a random forest on 80% of the data as training data and 20% as test data and **30 estimators (trees)** the following things were observed:

1. Area under ROC curve = 0.80

2. Average precision-recall score = 0.21

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**34228**	**1675**
Actual 1	**2806**	**1291**

Comment: Area under ROC curve is better for Random Forest compared to that of Logistic regression. However average precision-recall score is other way around.

- Fitting on Oversampled dataset

After applying SMOTE to the data, both the classes (0's and 1's) have a size of 1,43,999.On fitting a random forest on 80% of the data as training data and 20% as test data and **30 estimators (trees)** the following things were observed:

1. Area under ROC curve = 0.64

2. Average precision-recall score = 0.13

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**14399**	**21504**
Actual 1	**702**	**3395**

Comment: No. of false positives further increases and no of true negatives increases. Clearly this method results in under fitting of the model.

- Fitting of data after applying Random Under Sampling technique

Random Under Sampling technique randomly selects a subset of size of the minority class from the majority class. Each class contains 16001 entries. However, there can be loss of information using this algorithm for under sampling. Following observations were made after fitting a Random Forest classifier, of **30 estimators (tress)**, to it:

1. Area under ROC curve = 0.56

2. Average precision-recall score =0.12

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**4963**	**30940**
Actual 1	**57**	**4040**

Comment: We observe that this model is highly biased in predicting 1 against 0.

Fitting of model after applying Principal Component Analysis

The dimension of the data is reduced from 198 to 158 by applying Principal Component Analysis. After reducing the data, 80% of the variance of the original data is retained.

- **RANDOM FOREST**

In this case the prediction of random forest is on continuous interval. Since the target variable is binary, we convert the predictions to 1 if the predicted value is >0.25 else 0. This cut off value is chosen as it gives the best accuracy till now.

- Fitting on imbalanced dataset

On fitting a random forest on 80% of the data as training data and 20% as test data and **30 estimators (trees)** the following things were observed:

1. Area under ROC curve = 0.73

2. Average precision-recall score = 0.29

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**33270**	**2633**
Actual 1	**1888**	**2209**

Comment: We observe higher values of both the metrics compared to that of Random Forest fitted without PCA.

- Fitting on Oversampled dataset

After applying SMOTE to the data, both the classes (0's and 1's) have a size of 1,43,999.On fitting a random forest on 80% of the data as training data and 20% as test data and **30 estimators (trees)** the following things were observed:

1. Area under ROC curve = 0.74

2. Average precision-recall score = 0.19

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**23367**	**12536**
Actual 1	**638**	**3459**

Comment: Compared to the previous case where Random Forest is fitted on a PCA augmented imbalanced data, this model performs moderately.

- Fitting of data after applying Random Under Sampling technique

Random Under Sampling technique randomly selects a subset of size of the minority class from the majority class. Each class contains 16001 entries. However, there can be loss of information using this algorithm for under sampling. Following observations were made after fitting a Random Forest classifier, of **30 estimators (tress)**, to it:

1. Area under ROC curve = 0.72

2. Average precision-recall score =0.17

3. Confusion matrix:

	Predicted 0	**Predicted 1**
Actual 0	**18192**	**17711**
Actual 1	**311**	**3786**

Comment: Compared to the case where Random Forest is fitted on a PCA augmented imbalanced data, this model performs moderately.

OBSERVATION AND CONCLUSION

After fitting the models to the data in the previous section, we make comprehensive tables stating the results.

Table showing Misclassification errors for Logistic regression model

Model	Misclassification Error for 0	Misclassification Error for 1	Total misclassification error
1. Logistic Classification			
a. Imbalanced data set	0.013	0.73	0.08
b. Oversampled data	0.21	0.23	0.22
c. Under sampled data	0.22	0.22	0.22

Observation: We observe that for Logistic regression fitted on over-sampled and under sampled data, total misclassification error may not be the least but individual misclassification errors are comparatively less or close to that of imbalanced data. For the model fitted on imbalanced data, misclassification error for class 1 is very high compared to the other cases.

Table showing Misclassification errors for Random Forest model

Model	Misclassification Error for 0	Misclassification Error for 1	Total misclassification error
1. Random Forest			
a. Imbalanced data set	0.046	0.68	0.11
b. Oversampled data	0.58	0.17	0.5
c. Under sampled data	0.86	0.01	0.77

Observation: We observe that the random forest fitted on Imbalanced data is highly skewed in misclassifying class 1 compared to class 0. Though its overall misclassification error is least compared to other cases but this kind of biased model is not much suitable.

Table showing Misclassification errors for Random Forest model Under Dimensional Reduction

Model	Misclassification Error for 0	Misclassification Error for 1	Total misclassification error
1. Random Forest			
a. Imbalanced data set	0.07	0.46	0.11
b. Oversampled data	0.35	0.16	0.33
c. Under sampled data	0.49	0.07	0.45

Observation: We observe that after applying PCA, misclassification errors have drastically gone down.

Conclusion:

We observe that the Random Forest model works better when the dimension of the data is reduced. In case of the imbalanced dataset, the total misclassification error remains the same in the dimension-reduced and complete dataset for a Random Forest model. However, for the Random Forest model fitted on PCA augmented over-sampled and under-sampled data, total misclassification errors have reduced drastically. In case of Logistic regression, no such significant observation was made for model fitted on PCA augmented data. That is why no such comprehensive table was shown.

Comparing the Random Forest model fitted on PCA augmented data, to the Logistic regression model fitted on complete data, we can conclude that a model which is biased in classifying a particular class is not a very good predictive model. Hence we choose the Logistic regression model fitted on over-sampled data to be the best predictive model. Its overall misclassification error is low and the model is equally biased in misclassifying both the classes.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude towards my supervisor and mentor Professor Debjit Sengupta, for his endless and tireless support. Without his encouragement and guidance, it would have been impossible for me to complete this project. I would also like to thank all the professors of the Department of Statistics, St. Xavier's College, Kolkata for their support and guidance.

BIBLIOGRAPHY

I would like to cite the following sources that helped in the completion of my dissertation project:

- Python Machine learning by Sebastian Raschka
- Machine Learning With Random Forests And Decision Trees By Scott Hartshorn