Project: Santander Customer Transaction Prediction

Sailesh Patra

19th February 2020

Table of Contents

[2.1 Problem Statement 3](#_Toc32970951)

[2.2 Data 3](#_Toc32970952)

[3.1 Pre-Processing 5](#_Toc32970953)

[3.2 Modelling 22](#_Toc32970954)

4.1 Summary and Conclusion……………………………………………………………………………………………………46

Chapter 1

1. Introduction

# Problem Statement

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals. Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

# Data

Our task is to build a classifier that will p­­­­redict whether a customer is going to do a transaction or not in future based the parameters given below:

Table 1.1 Santander Customer Transaction Train Data

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID\_code | target | var\_0 | var\_1 | var\_2 | var\_3 | ………. | ………. | var\_199 |
| train\_0 | 0 | 8.92 | -6.78 | 11.90 | 5.09 | ………. | ………. | -1.09 |
| train\_01 | 0 | 11.5 | -4.14 | 13.85 | 5.38 | ………. | ………. | 1.95 |
| train\_02 | 0 | 8.60 | -2.15 | 12.08 | 7.89 | ………. | ………. | 0.39 |

Table 1.2 Santander Customer Transaction Test Data

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID\_code | var\_0 | var\_1 | var\_2 | var\_3 | var\_4 | ………. | ………. | var\_199 |
| test\_0 | 11.06 | 7.77 | 12.95 | 9.42 | 11.43 | ………. | ………. | -8.71 |
| test\_1 | 8.53 | 1.25 | 11.30 | 5.18 | 9.19 | ………. | ………. | -20.97 |
| test\_2 | 5.48 | -10.35 | 10.14 | 7.04 | 10.26 | ………. | ………. | -23.17 |

Unfortunately, we don’t have any idea about what each of these variables denotes as all the variables are named anonymously. Hence, we will be lacking the domain knowledge while doing this project.

The only thing we know about the data is there are 200; all numeric variables and which will be used to predict whether the customer is going to do the transaction in future or not. Besides that we have an unique ID\_Code assigned to each row/observation.

Chapter 2

1. Methodology

# Pre-Processing

Pre-processing data is a crucial step and also the initial step in any data science project. Before developing a model and make it ready to understand and learn the hidden patterns from our data, it is necessary to make it noise-free. By cleaning the data, we restrict our model from learning the unclear patterns and cleaning data also helps us in reducing complexity of the data under analysis. Real-world data is often incomplete, inconsistent, and lacking certain behaviours or trends, and is likely to contain many errors which may obstruct in getting successful predictions. Pre-processing techniques transforms these raw data into an understandable format for our data.

Pre-processing is not limited to clean the data only, but also involves in deriving statistical information from the data, visualizing the data to know how they are distributed through graphs and plots. All these steps together well known as **Exploratory Data Analysis** in data science terminology.

* + 1. Data Exploration

We are provided with two datasets, one for training purposes and another for testing/evaluating purposes. Some details about both the datasets is given below:

Shape of our train dataset:

Shape of our train dataset:

(200000, 202)

Shape of our test dataset:

Shape of our test dataset:

(200000, 201)

Our test data does not have the target variable; hence we can’t use it for evaluation purpose. Hence, we will use the training data both for training and evaluation purpose of our model.

But we can predict the result using the predictor variables of test dataset which is expected in this project.

**Info about train data:**

Basic info about train dataset:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200000 entries, 0 to 199999

Columns: 202 entries, ID\_code to var\_199

dtypes: float64(200), int64(1), object(1)

memory usage: 308.2+ MB

**Info about test data:**

Basic info about train dataset:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200000 entries, 0 to 199999

Columns: 201 entries, ID\_code to var\_199

dtypes: float64(200), object(1)

memory usage: 306.7+ MB

Basic info about train data says that there are **200** columns of **float** type, **1** column of **int** type and another of **object** type and the test data is also almost same except there is no int type column in test data; which is the target variables that we have predict for test dataset.

From section [1.2](#_Data) also, it can be known that the “***target***” column is of integer type and “***ID\_code***” should be an object type data and hence the rest 200 columns will be of float type.

From basic sense, we can say that ID\_code columns is just an index of our dataset to uniquely identify an observation and will be of no use to classify our target.

* + - 1. Target class imbalance:

During analysis, we observe an issue with our data, that our data is having imbalance target class problem, where one target class has a major presence than other target class in our dataset.

In our dataset there is huge imbalance between the target classes. The ratio between the two classes is 9:1 as shown in the pie chart below:

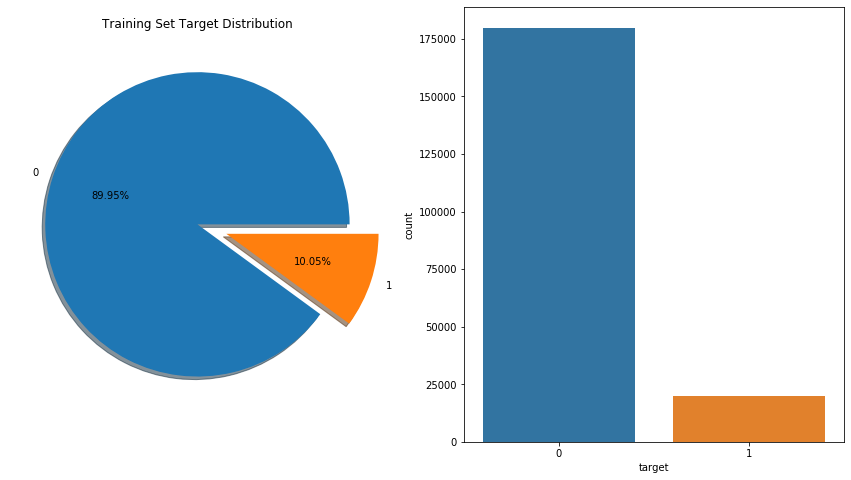


Figure 1 Pie chart and Histogram showing target class imbalance in our data

The ‘0’ class is our majority class here and ‘1’ class is the minority class. We have to handle this problem which we will do later in data partition section through **SMOTE(Synthetic Minority Over-sampling Technique)** technique. It aims to balance class distribution by randomly increasing minority class examples by replicating them.

If we won’t handle this problem, our model may bias more towards the majority class and may ignore the minority class to draw patterns from the minority class.

* + 1. Outlier analysis:

An outlier is a data point that differs significantly from other observations. An outlier can affect the mean of a data set by skewing the results so that the mean is no longer representative of the dataset. So, our dataset should be free from outliers for better performance of our model. There are mainly two ways to treat to outliers: Boxplot method or Replace with NAs. Outlier analysis can only be performed on numeric variables.

As we have class imbalance problem, the minority class will be treated as an outlier itself and hence it’s not recommended to remove/replace the outliers; which will be the minority class here by which we will even loose the small information about the minority class we have.

* + 1. Visualization:

After all these 3 major operations we are ready to draw insights from our data through different visualizations or statistical techniques as discussed in section [2.1](#_Pre-Processing). The next methods will include univariate and bivariate analysis of our data. As the shape of our data is so huge, visualization will be an effective way to observe the statistical distribution of our predictor variables than seeing each variable individually.

As all the predictor variables are numeric, we can observe each variable’s distribution through their distribution plot. Before that let’s see what information a normally distributed data share:

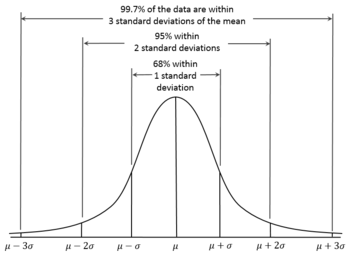


Figure 2 Normal Distribution

From the above figure, it can be derived that, If a data is normally distributed, 68% of the data is 1 standard deviation away from the mean, which is a huge part of the data and also 95% of the data is 2 standard deviation away from mean. Hence if a column is normally distributed, we can say that it is centred towards the mean and mean is the appropriate metric to judge the centrality of the data.

Now let’s check the distribution of the data provided to us through univariate analysis.

* + - 1. Univariate analysis:



Figure 3 Distribution of var\_0 to var\_99



Figure 4 Distribution plot of var\_100 to var\_199

Similarly let’s check the distribution of the test data available to us:



Figure 5 Distribution plot of first 100 test variables



Figure 6 Distribution plot of last 100 test variables

**Conclusion**: As all the variables we are dealing with are nearly normally distributed; we don’t have to do anything. Otherwise we need to normalize the data before proceeding to the next step.

* + - 1. Bivariate Analysis:

We’ll perform this under correlation analysis later in this process of EDA.

* + - 1. Check for duplicate rows:

It’s quite necessary to reduce the redundant information that we sent to our model, to make it easy for our model to extract the hidden patterns from the data. Each row should be unique and different from each other.

No. of duplicate rows based on all columns are :

0

No. of duplicate rows based on all columns are :

0

* + 1. Feature Selection:

We get a lot of observations and variables in our raw data, but it is not necessary that all of those will be helpful in achieving our target. We have to selective while choosing variables that we are going to feed to our model, so that it won’t be complex for our model to draw out the patterns from our data. There are so many feature selection techniques.

During feature selection we have to keep in mind one thing that, there should be high dependency between our independent variables and target variable & there should be very less or no dependency in between the independent variables. To check the inter-dependency between the independent variables we’ll apply correlation analysis.

As the data is huge and has a huge dimension, a heatmap will be a pain to the eye to check each variable colour individually. So, to resolve this issue we can check the distribution of correlation values of our data.

To do this, we first calculate the correlation between each of the 200 numeric variables, removed the correlation between the same variable which will always be 1 and then plotted a distribution plot.

train\_corr = train\_corr.unstack()

train\_corr

var\_0 var\_0 1.000000

var\_1 0.000544

var\_2 0.006573

var\_3 0.003801

var\_4 0.001326

...

var\_199 var\_195 0.002042

var\_196 0.000607

var\_197 0.004991

var\_198 0.004731

var\_199 1.000000

Length: 40000, dtype: float64

train\_corr = train\_corr.sort\_values(kind="quicksort")

train\_corr

var\_75 var\_191 2.703975e-08

var\_191 var\_75 2.703975e-08

var\_173 var\_6 5.942735e-08

var\_6 var\_173 5.942735e-08

var\_126 var\_109 1.313947e-07

...

var\_128 var\_128 1.000000e+00

var\_127 var\_127 1.000000e+00

var\_126 var\_126 1.000000e+00

var\_124 var\_124 1.000000e+00

var\_199 var\_199 1.000000e+00

Length: 40000, dtype: float64

After doing the same thing for the test data; we can observe their distribution which is something like this:

train\_corr.iloc[:,2].describe()

count 3.980000e+04

mean 1.986439e-03

std 1.506084e-03

min 2.703975e-08

25% 7.903091e-04

50% 1.679507e-03

75% 2.874466e-03

max 9.844361e-03

Name: 0, dtype: float64

test\_corr.iloc[:,2].describe()

count 3.980000e+04

mean 1.853484e-03

std 1.399296e-03

min 1.477268e-07

25% 7.349334e-04

50% 1.560695e-03

75% 2.689444e-03

max 9.867773e-03

Name: 0, dtype: float64

From the above data we can say that there is very very less correlation between our predictor variables in both train and test data.

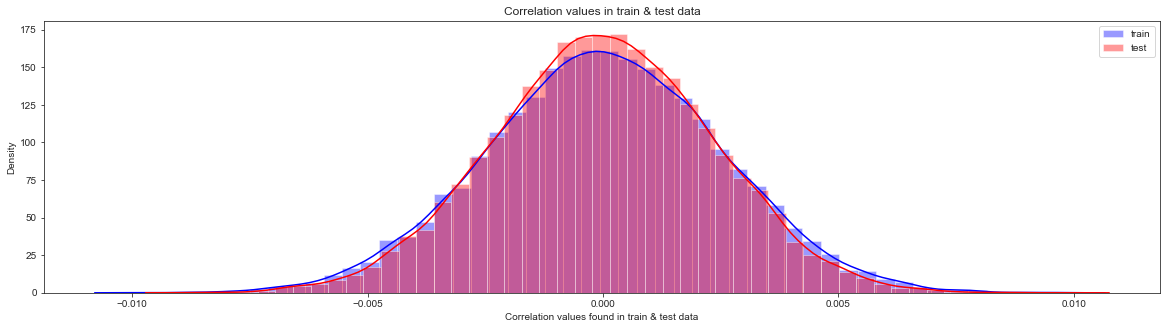


Figure 7 Correlation between train and test data

**Conclusion:**

With reference of section 2.1.3, we can say that 95% of the correlation values are only 2 standard deviation away from mean; i.e. 1.986439e-03 for train data and 1.853484e-03 for test data.

* + - 1. Feature Importance

We can also check the importance of every predictor variable that they contribute in predicting the target variable. For this dataset we got the result as below:

|  | **Variable** | **importance** |
| --- | --- | --- |
| 30 | var\_30 | 0.003018 |
| 27 | var\_27 | 0.003184 |
| 72 | var\_72 | 0.003242 |
| 38 | var\_38 | 0.003261 |
| 3 | var\_3 | 0.003287 |
| ... | ... | ... |
| 109 | var\_109 | 0.009395 |
| 80 | var\_80 | 0.009492 |
| 53 | var\_53 | 0.009571 |
| 139 | var\_139 | 0.010192 |
| 81 | var\_81 | 0.013020 |

200 rows × 2 columns

***Conclusion***:

Variable 81 is the most important variable in our dataset.

* + 1. Feature Scaling:

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. In this case the features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

Hence, to bring all the values in a range of 0 to 1, we’ll perform feature scaling operation.

* + 1. Dimensionality Reduction Technique:

In statistics, machine learning, and information theory, dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

Large number of variables arise the danger of overfitting your model to your data. By reducing the dimension of your feature space, you have fewer relationships between variables to consider and you are less likely to overfit your model. There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

* Feature Elimination
* Feature Extraction
  + - 1. Principal Component Analysis – PCA:

Principal component analysis is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the “least important” variables while still retaining the most valuable parts of all of the variables. As an added benefit, each of the “new” variables after PCA are all independent of one another. This is a benefit because the [assumptions of a linear model](http://people.duke.edu/~rnau/testing.htm) require our independent variables to be independent of one another. If we decide to fit a linear regression model with these “new” variables this assumption will necessarily be satisfied.

We have given the number of **components = 170** for our dataset which can be decided by hit-n-trial method in such a way that we can reduce the dimensions and get a good amount of variance of the whole data in those specified principal components.

from sklearn.decomposition import PCA

pca = PCA(n\_components=170)

principalComponents = pca.fit\_transform(x)

principalDf = pd.DataFrame(data = principalComponents)

print(sum(pca.explained\_variance\_ratio\_))

0.8631213066914379

Our data gives 86.31% of variance of the raw original data while selecting 170 principal components.

What PCA does is it creates principal components in such a manner that the 1st principal component is the highest variance sharing component among all the components, 2nd principal component is the second highest component in terms of variance and so on. Let’s understand this better through a Scree plot.

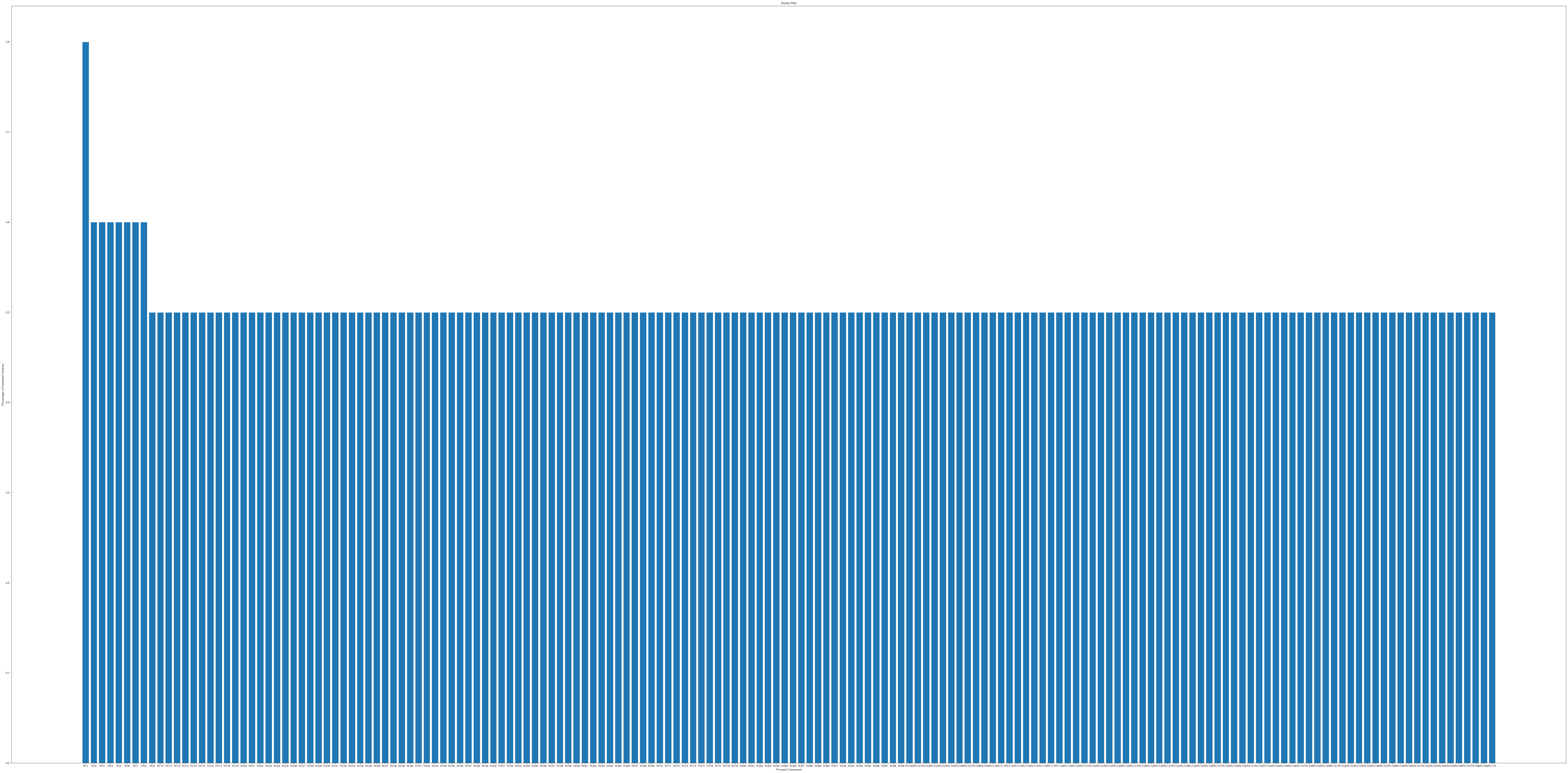


Figure 8 No. of PCS vs Percentage of Variance explained

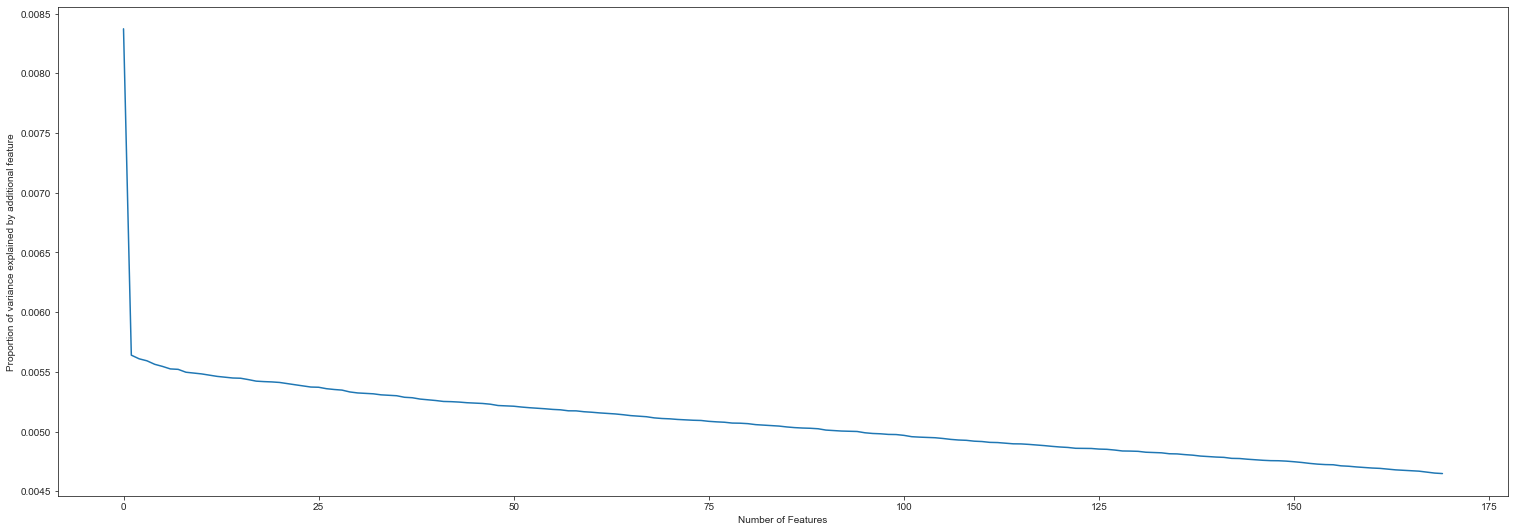


Figure 9 Elbow Curve showing how the variance explained reduces with increase in no. of components

As discussed above the variance explained by 1st principal component is highest among all and reduces thereafter with increasing no. of components is also shown from the figure above.

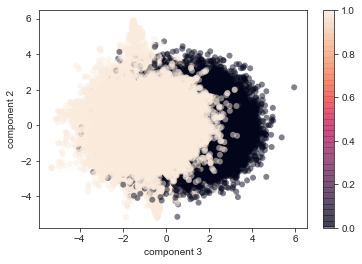


Figure 10 2D plot showing separability of Principal component 1 and 2

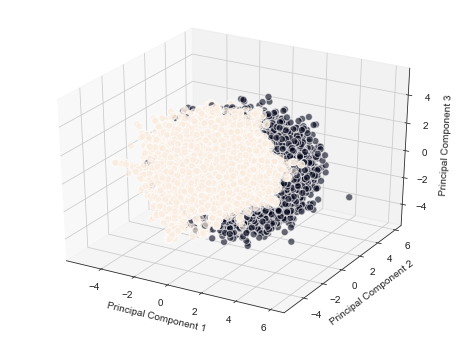


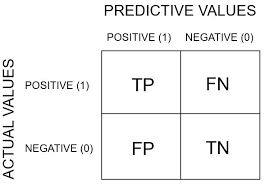
Figure 11 3D plot showing the separability of PC 1, PC2 and PC3

By so far, we got a general idea about our data and the important things to remember is our data has target class imbalance problem and also as it is very large dataset, dimensionality reduction will help in classifying our target by reducing the unwanted information and focusing on the principal components.

Now, before proceeding for the Modelling section, we should choose the evaluation metric for our model here. Now as our problem is a classification problem, roc-auc curve can be one metric, but as our data is imbalanced only roc-auc may not be the right metric or we can’t blindly believe in the results of roc-auc only. So, we’ll consider both ROC-AUC & Precision-Recall curve for evaluation purposes.

* + 1. Choosing the right metric:

**Confusion Matrix:**

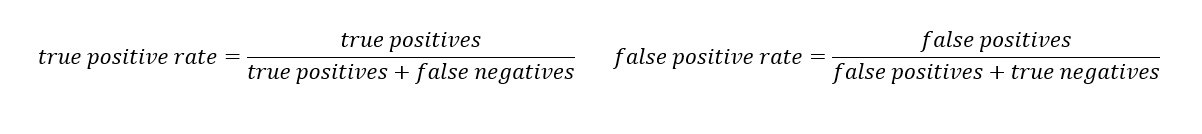


**True Positive:** You predicted positive and it’s true.

**True Negative:** You predicted negative and it’s true.

**False Positive:** You predicted positive and it’s false.

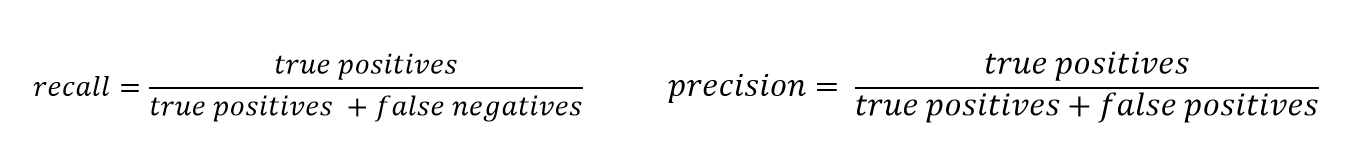
**False Negative:** You predicted negative and it’s false.



TPR: From all the actual positive classes (TP+FN), how many **actual** positive classes we have **correctly** predicted

FPR: From all the actual negative classes (FP+TN), how many **actual** negative classes we have **wrongly** predicted.

In both TPR & FPR we only talked about the positive target class and ignored the correctly/ wrongly predictions of the negative target class and this is the reason which Precision-Recall curve is the correct measure for class imbalance problem where we can evaluate both the target classes and we give importance to correct classification of both the classes in PR curve.

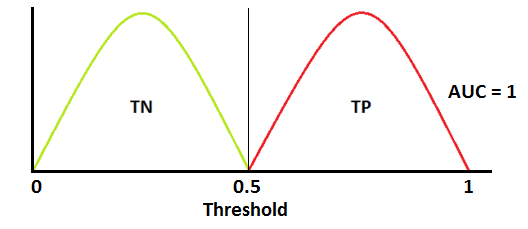
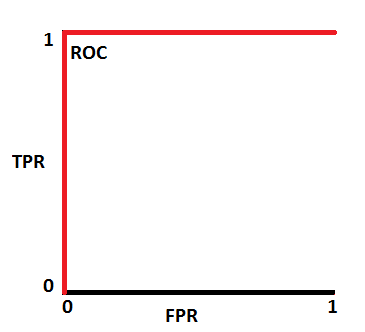


Recall: This is same as the TPR, i.e.; What proportion of the actual positive classes are correctly **predicted** as positive.

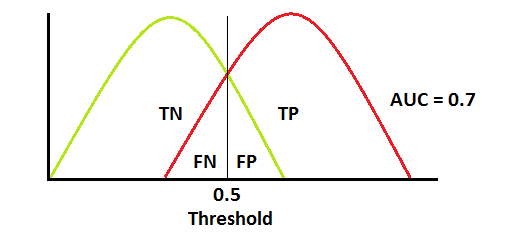
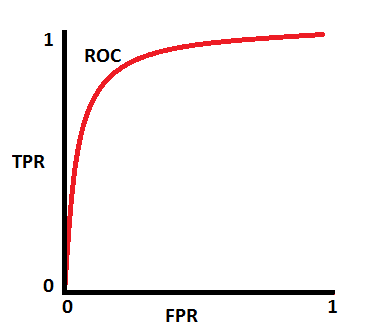
Precision: What proportion of the predicted as positive classes are **actually** positive.

The following pictures denotes how ROC-AUC variates with no. of False classifications:

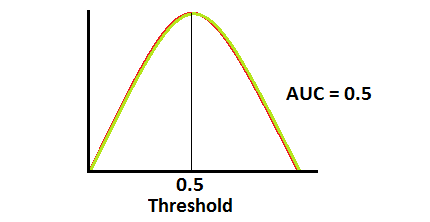
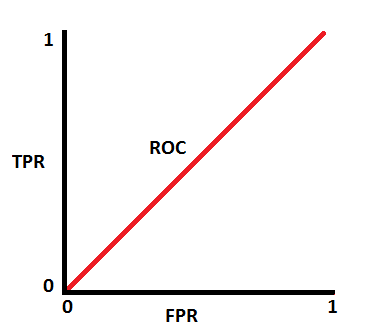
Red distribution curve is of the positive class and green distribution curve is of negative class.

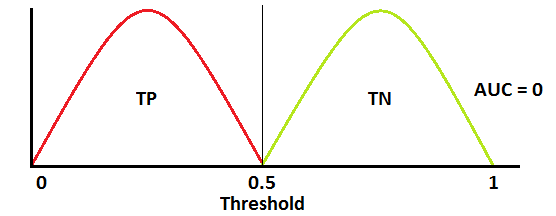
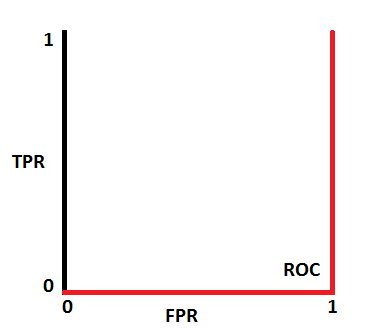
Ideal Case-Very rare

Standard Scenario that we’ll face normally

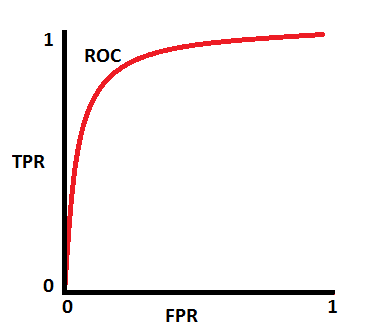
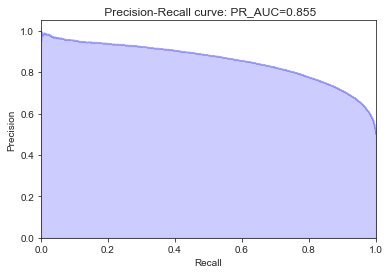
 

Ambiguous Situation

Worst case scenario

An acceptable ROC-AUC curve vs Precision-Recall Curve:

We expect our model’s ROC-AUC to have a high y-value and low x-value, i.e.; High TPR and low FPR and to have high x and y value in case of Precision-Recall curve.

# **Modelling**

* + 1. Model Selection

The target we are chasing to predict is a categorical variable. Hence, we have a classification problem. Classification problems can be solved by many algorithms, e.g. Decision tree, Logistic regression, Random forest, Naïve Bayes, Boosting algorithms and many more.

Here we will apply all algorithms mentioned above one by one and compare them to know the best model for this particular problem.

To know the how our model performs we will apply each of the machine learning algorithm on 3 datasets; one on the original dataset, one after applying oversampling and another on the principal components.

Before applying any model, I would like to show the code structure used for development and evaluation of our model:

1. **def** draw\_confusion\_mx(y\_test,y\_pred):
2. **print**('\n######### Confusion Matrix #########\n')
3. cm=pd.crosstab(y\_test,y\_pred)
4. **print**(cm)
6. **def** draw\_classification\_report(y\_test,y\_pred):
7. **print**('\n######### Classification Report #########\n')
8. **print**(classification\_report(y\_test,y\_pred))
10. **def** draw\_roc\_auc(y\_test,y\_pred):  ##y\_pred in form of probabilities
11. ns\_probs = [0 **for** \_ **in** range(len(y\_test))]
12. ns\_fpr, ns\_tpr, \_ = roc\_curve(y\_test, ns\_probs)
13. lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, y\_pred)
14. plt2.plot(ns\_fpr, ns\_tpr, linestyle='--', label='No Skill')
15. plt2.plot(lr\_fpr, lr\_tpr, marker='.', label='Logistic')
16. auc\_score=auc(lr\_fpr,lr\_tpr)
17. plt2.title('ROC(area=%0.3f)' %auc\_score)
19. plt2.xlabel('False Positive Rate')
20. plt2.ylabel('True Positive Rate')
22. plt2.legend()
24. plt2.show()
26. **def** draw\_precision\_recall(y\_test,y\_pred):
27. precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred)
29. # In matplotlib < 1.5, plt.fill\_between does not have a 'step' argument
30. step\_kwargs = ({'step': 'post'}
31. **if** 'step' **in** signature(plt2.fill\_between).parameters
32. **else** {})
33. plt2.step(recall, precision, color='b', alpha=0.2,
34. where='post')
35. plt2.fill\_between(recall, precision, alpha=0.2, color='b', \*\*step\_kwargs)
37. plt2.xlabel('Recall')
38. plt2.ylabel('Precision')
39. plt2.ylim([0.0, 1.05])
40. plt2.xlim([0.0, 1.0])
41. plt2.title(' Precision-Recall curve: PR\_AUC={0:0.3f}'.format( auc(recall, precision)))
42. plt2.show()
44. **def** fit\_N\_predict(model,X\_train,X\_test,y\_train,y\_test,model\_code,testData,PCA=0):
46. model.fit(X\_train,y\_train)
47. y\_pred = model.predict(X\_test)
48. y\_pred2 = model.predict\_proba(X\_test)
49. y\_pred2 = y\_pred2[:,1]
51. draw\_confusion\_mx(y\_test,y\_pred)
53. draw\_classification\_report(y\_test,y\_pred)
55. draw\_roc\_auc(y\_test,y\_pred2)
57. draw\_precision\_recall(y\_test,y\_pred2)
58. **if**(PCA == 0):
59. **if**(model\_code!="XGB"):
60. **print**('\n\nModel performance on test data:\n',)
61. **print**(model.predict(testData.drop(['ID\_code'],axis=1)))
62. **else**:
63. **print**('\n\nModel performance on test data:\n',)
64. **print**(model.predict(testData.drop(['ID\_code'],axis=1).values))

**fit\_N\_predict:** A single function is made for easy modification and to maintain code reusability, which takes a model object, train data, test data and a unique model code for each model and returns back the evaluation metric we set.

**Logistic Regression:**

######### Confusion Matrix #########

col\_0 0 1

target

0 42282 11669

1 1375 4674

######### Classification Report #########

precision recall f1-score support

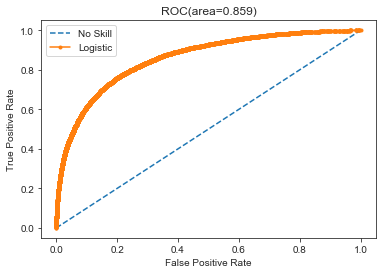
0 0.97 0.78 0.87 53951

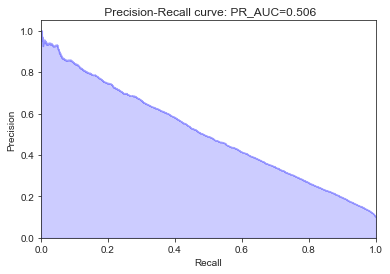
1 0.29 0.77 0.42 6049

accuracy 0.78 60000

macro avg 0.63 0.78 0.64 60000

weighted avg 0.90 0.78 0.82 60000

****

****

Model performance on test data:

[0 0 0 ... 0 0 0]

**Logistic Regression on oversampled dataset:**

######### Confusion Matrix #########

col\_0 0 1

target

0 42258 11693

1 11565 42386

######### Classification Report #########

precision recall f1-score support

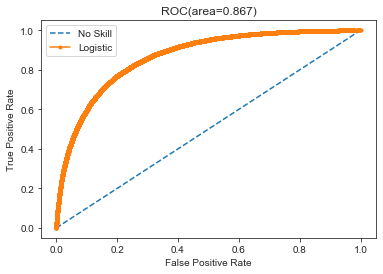
0 0.79 0.78 0.78 53951

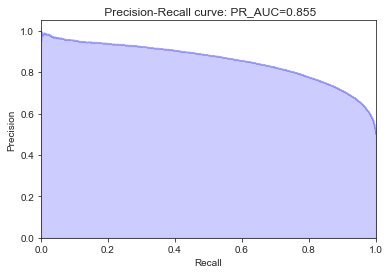
1 0.78 0.79 0.78 53951

accuracy 0.78 107902

macro avg 0.78 0.78 0.78 107902

weighted avg 0.78 0.78 0.78 107902

****

****

Model performance on test data:

[1 1 0 ... 0 0 1]

**Logistic Regression on PCA of oversampled data:**

######### Confusion Matrix #########

col\_0 0 1

target

0 35480 9342

1 8802 36327

######### Classification Report #########

precision recall f1-score support

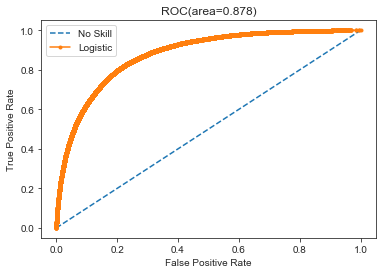
0 0.80 0.79 0.80 44822

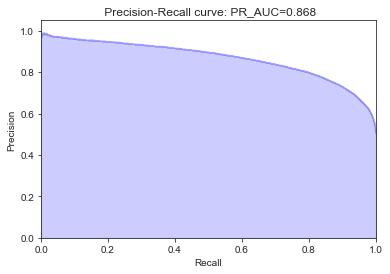
1 0.80 0.80 0.80 45129

accuracy 0.80 89951

macro avg 0.80 0.80 0.80 89951

weighted avg 0.80 0.80 0.80 89951





**Decision Tree:**

######### Confusion Matrix #########

col\_0 0 1

target

0 35288 18663

1 2631 3418

######### Classification Report #########

precision recall f1-score support

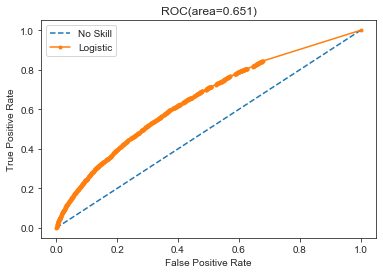
0 0.93 0.65 0.77 53951

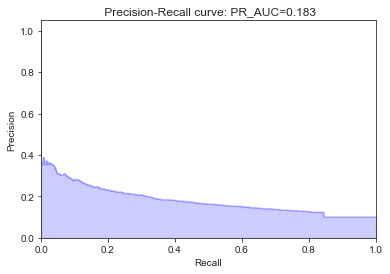
1 0.15 0.57 0.24 6049

accuracy 0.65 60000

macro avg 0.54 0.61 0.51 60000

weighted avg 0.85 0.65 0.72 60000

****

****

Model performance on test data:

[0 0 0 ... 0 1 0]

**Decision Tree on oversampled dataset:**

######### Confusion Matrix #########

col\_0 0 1

target

0 39740 14211

1 23352 30599

######### Classification Report #########

precision recall f1-score support

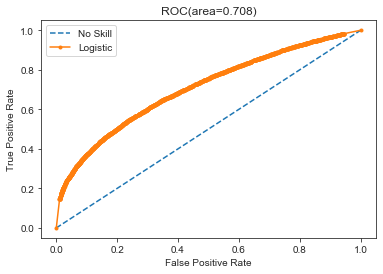
0 0.63 0.74 0.68 53951

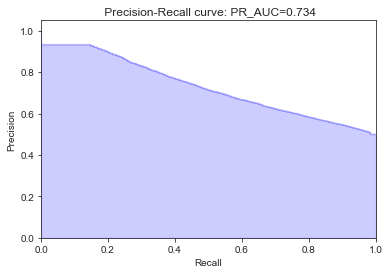
1 0.68 0.57 0.62 53951

accuracy 0.65 107902

macro avg 0.66 0.65 0.65 107902

weighted avg 0.66 0.65 0.65 107902

****

****

Model performance on test data:

[0 0 1 ... 1 0 1]

**Decision tree on PCA of oversampled data::**

######### Confusion Matrix #########

col\_0 0 1

target

0 36194 8628

1 8386 36743

######### Classification Report #########

precision recall f1-score support

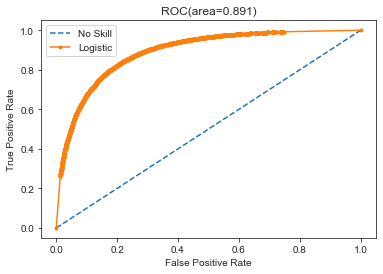
0 0.81 0.81 0.81 44822

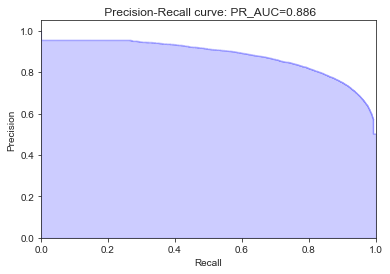
1 0.81 0.81 0.81 45129

accuracy 0.81 89951

macro avg 0.81 0.81 0.81 89951

weighted avg 0.81 0.81 0.81 89951

****

****

**Random Forest:**

######### Confusion Matrix #########

col\_0 0 1

target

0 46483 7468

1 2970 3079

######### Classification Report #########

precision recall f1-score support

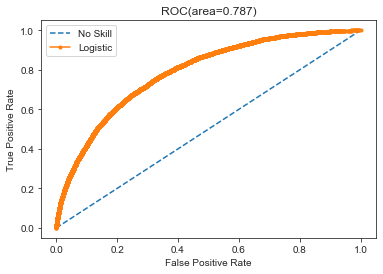
0 0.94 0.86 0.90 53951

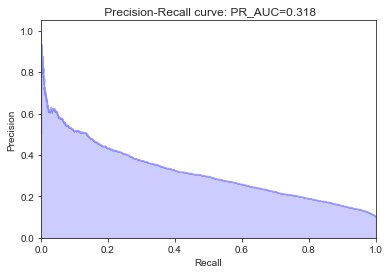
1 0.29 0.51 0.37 6049

accuracy 0.83 60000

macro avg 0.62 0.69 0.64 60000

weighted avg 0.87 0.83 0.85 60000

****

****

Model performance on test data:

[1 1 1 ... 0 1 1]

**Random forest on oversampled dataset:**

######### Confusion Matrix #########

col\_0 0 1

target

0 47641 6310

1 20146 33805

######### Classification Report #########

precision recall f1-score support

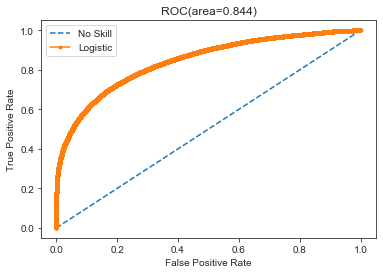
0 0.70 0.88 0.78 53951

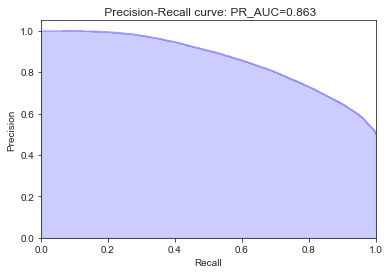
1 0.84 0.63 0.72 53951

accuracy 0.75 107902

macro avg 0.77 0.75 0.75 107902

weighted avg 0.77 0.75 0.75 107902

****

****

Model performance on test data:

[0 1 0 ... 0 0 0]

**Random forest on PCA of oversampled data:**

######### Confusion Matrix #########

col\_0 0 1

target

0 37035 7787

1 7010 38119

######### Classification Report #########

precision recall f1-score support

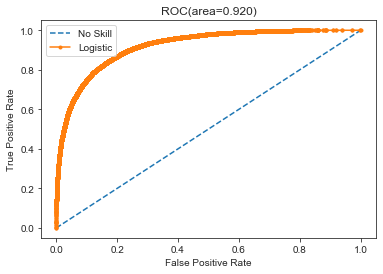
0 0.84 0.83 0.83 44822

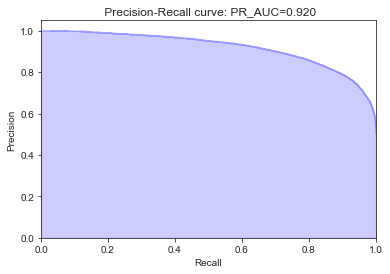
1 0.83 0.84 0.84 45129

accuracy 0.84 89951

macro avg 0.84 0.84 0.84 89951

weighted avg 0.84 0.84 0.84 89951

****

****

**Naïve Bayes:**

######### Confusion Matrix #########

col\_0 0 1

target

0 53077 874

1 3857 2192

######### Classification Report #########

precision recall f1-score support

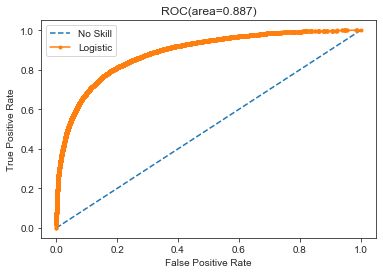
0 0.93 0.98 0.96 53951

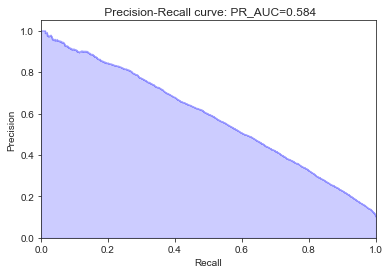
1 0.71 0.36 0.48 6049

accuracy 0.92 60000

macro avg 0.82 0.67 0.72 60000

weighted avg 0.91 0.92 0.91 60000

****

****

Model performance on test data:

[1 1 1 ... 1 1 1]

**Naïve Bayes on oversampled dataset:**

######### Confusion Matrix #########

col\_0 0 1

target

0 51757 2194

1 12190 41761

######### Classification Report #########

precision recall f1-score support

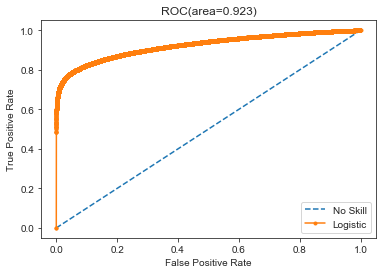
0 0.81 0.96 0.88 53951

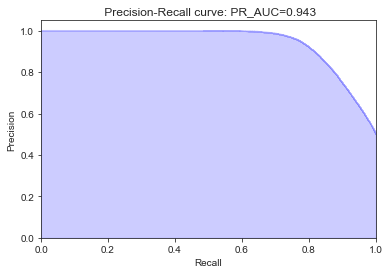
1 0.95 0.77 0.85 53951

accuracy 0.87 107902

macro avg 0.88 0.87 0.87 107902

weighted avg 0.88 0.87 0.87 107902

****

****

Model performance on test data:

[0 0 0 ... 0 0 1]

**Naïve Bayes on PCA of oversampled data:**

######### Confusion Matrix #########

col\_0 0 1

target

0 42001 2821

1 9757 35372

######### Classification Report #########

precision recall f1-score support

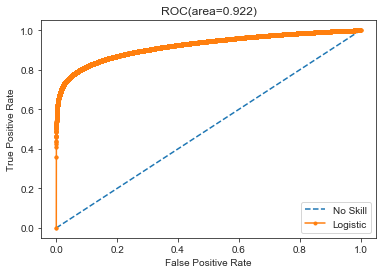
0 0.81 0.94 0.87 44822

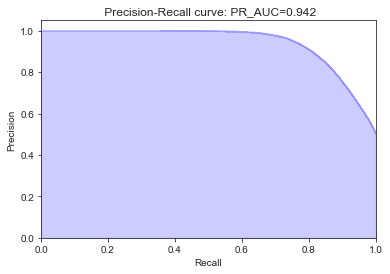
1 0.93 0.78 0.85 45129

accuracy 0.86 89951

macro avg 0.87 0.86 0.86 89951

weighted avg 0.87 0.86 0.86 89951

****

****

**XGBoost Regressor:**

######### Confusion Matrix #########

col\_0 0 1

target

0 52794 1157

1 3672 2377

######### Classification Report #########

precision recall f1-score support

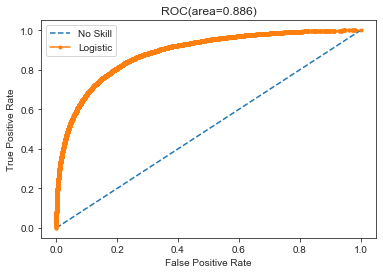
0 0.93 0.98 0.96 53951

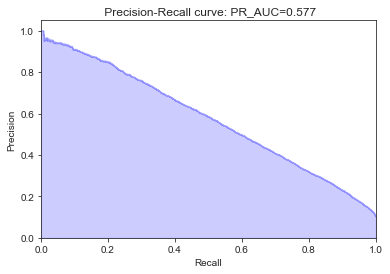
1 0.67 0.39 0.50 6049

accuracy 0.92 60000

macro avg 0.80 0.69 0.73 60000

weighted avg 0.91 0.92 0.91 60000

****

****

Model performance on test data:

[1 1 1 ... 1 1 1]

**XGBoost Regressor on oversampled dataset**

######### Confusion Matrix #########

col\_0 0 1

target

0 47976 5975

1 7324 46627

######### Classification Report #########

precision recall f1-score support

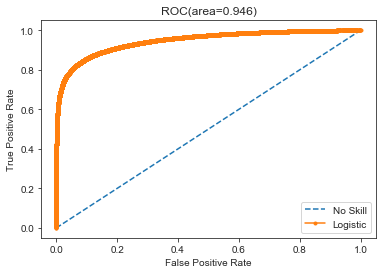
0 0.87 0.89 0.88 53951

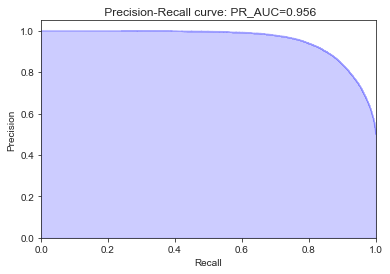
1 0.89 0.86 0.88 53951

accuracy 0.88 107902

macro avg 0.88 0.88 0.88 107902

weighted avg 0.88 0.88 0.88 107902

****

****

Model performance on test data:

[0 0 0 ... 0 0 1]

**XGBoost Regressor on PCA of oversampled data:**

######### Confusion Matrix #########

col\_0 0 1

target

0 38121 6701

1 2800 42329

######### Classification Report #########

precision recall f1-score support

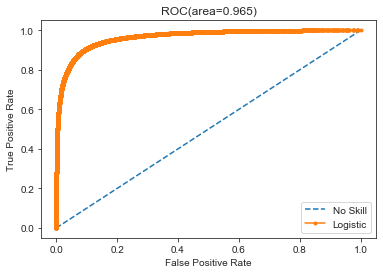
0 0.93 0.85 0.89 44822

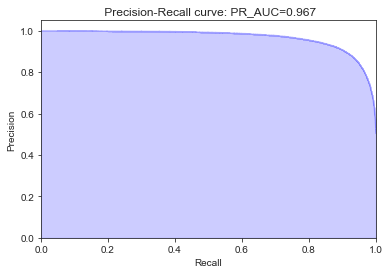
1 0.86 0.94 0.90 45129

accuracy 0.89 89951

macro avg 0.90 0.89 0.89 89951

weighted avg 0.90 0.89 0.89 89951

****

****

As the XGBoost algorithms gave the best result (AUPRC: 96.7%) out of all other algorithms, we should use XGBoost Classifier as our final model.

**­**

Chapter 3

1. Conclusion

Summary & Conclusion:

* Hence, after seeing the scores from all the model, we came to a conclusion that, Naïve Bayes and XGBoost algorithms works the best for our data.
* We can improve the model’s performance by hyper-parameter tuning also.
* As our data has so many dimensions, running PCA to reduce the dimensionality of our data has a great role in improving model’s performance.
* Our dataset also has class imbalance problem for which we have applied SMOTE technique to maintain the target class ratio.
* Choosing the right evaluation metric is very important in such sensitive matters like class imbalance problem.
* As the dataset is very huge, a good computer with high configuration will be helpful specifically for running programming languages like R.
* As the variables are anonymous we lacked the domain knowledge throughout the project.

1. Appendix A :: Python Code
2. **import** os
3. **import** numpy as np
4. **import** pandas as pd
5. **import** matplotlib as plt
6. **import** matplotlib.pyplot as plt2
7. **import** seaborn as sns
8. **import** sys
10. **from** sklearn.model\_selection **import** train\_test\_split,cross\_val\_predict,cross\_val\_score,StratifiedKFold
11. **from** sklearn.ensemble **import** RandomForestClassifier
12. **from** sklearn.linear\_model **import** LogisticRegression
13. **from** sklearn.metrics **import** confusion\_matrix,roc\_auc\_score,roc\_curve,classification\_report,roc\_curve,auc
15. **from** inspect **import** signature
16. **from** sklearn.metrics **import** average\_precision\_score,precision\_recall\_curve
18. **from** imblearn.over\_sampling **import** SMOTE
20. **from** sklearn.tree **import** DecisionTreeClassifier
21. **from** sklearn.neighbors **import** KNeighborsClassifier
22. **from** mpl\_toolkits.mplot3d **import** Axes3D

25. **if** **not** sys.warnoptions:
26. **import** warnings
27. warnings.simplefilter("ignore")

30. os.chdir("C:/Users/Acesocloud/Downloads/Kaggle/Santander Customer Transaction Prediction/Sailesh Santander")
32. df\_santander = pd.read\_csv("train.csv")
34. df\_santander\_test = pd.read\_csv("test.csv")
36. **print**('Shape of our dataset:')
37. **print**(df\_santander.shape,'\n')
39. pd.options.display.max\_columns = None

42. # # Exploratory Data Analysis

45. **print**('\*'\*25,'Exploratory Data Analysis: ','\*'\*25,'\n')
47. **print**('Showing 1st few rows of our dataset: \n')
48. **print**(df\_santander.head(5))
50. **print**("Basic info about dataset:\n")
51. **print**(df\_santander.info())
53. **print**("Data Description:\n")
55. # ### Target Class Count
57. target\_count = df\_santander['target'].value\_counts()
59. **print**("Count of categories of the target variable:\n", target\_count)
61. **print**("Percentage of each category of the target variable:\n", ((target\_count/df\_santander.shape[0]))\*100)
63. # ### Data Visualization
65. f, ax = plt2.subplots(1,2,figsize=(15,8))
66. pie\_data = df\_santander['target'].value\_counts()
67. pie\_data.plot.pie(explode=[0,0.2], autopct='%1.2f%%', ax = ax[0], shadow = True)
68. ax[0].set\_title('Training Set Target Distribution')
69. ax[0].set\_ylabel('')
71. sns.countplot('target', data = df\_santander, ax = ax[1])
72. plt2.show()
74. # ### Missing Value Analysis
76. train\_missing = df\_santander.isnull().sum()
78. **print**("No. of rows having missing values in train data:")
79. **print**(train\_missing.loc[train\_missing > 0].shape[0])
81. test\_missing = df\_santander\_test.isnull().sum()
82. **print**("No. of rows having missing values in test data:")
83. **print**(test\_missing.loc[test\_missing > 0].shape[0])

86. # ### Outlier Analysis
88. #      Can not perform as we have imbalance dataset
90. # ### Distribution of training data
92. **def** plot\_train\_data\_dist(cat\_0,cat\_1, label1, label2, columns):
93. i = 0
94. sns.set\_style('darkgrid')
96. fig = plt2.figure()
97. ax = plt2.subplots(10,10,figsize=(22,18))
99. **for** col **in** columns:
100. i += 1
101. plt2.subplot(10,10,i)
102. sns.distplot(cat\_0[col], hist=False, label=label1)
103. sns.distplot(cat\_1[col], hist=False, label=label2)
104. plt2.legend()
105. plt2.xlabel('Attribute',)
106. plt2.show()
108. cat\_0 = df\_santander.loc[df\_santander['target'] == 0]
109. cat\_1 = df\_santander.loc[df\_santander['target'] == 1]
111. label1 = '0'
112. label2 = '1'
114. columns = df\_santander.columns.values[2:102]
115. plot\_train\_data\_dist(cat\_0, cat\_1, label1, label2, columns)
117. columns = df\_santander.columns.values[102:202]
118. plot\_train\_data\_dist(cat\_0, cat\_1, label1, label2, columns)
120. # ### Distribution of test data
122. **def** plot\_test\_data\_dist(test\_attributes):
123. i=0
124. sns.set\_style('whitegrid')
126. fig=plt2.figure()
127. ax=plt2.subplots(10,10,figsize=(22,18))
129. **for** attribute **in** test\_attributes:
130. i+=1
131. plt2.subplot(10,10,i)
132. sns.distplot(df\_santander\_test[attribute],hist=False)
133. plt2.xlabel('Attribute',)
134. sns.set\_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
135. plt2.show()
137. test\_attributes=df\_santander\_test.columns.values[1:101]
138. plot\_test\_data\_dist(test\_attributes)
140. test\_attributes=df\_santander\_test.columns.values[102:202]
141. plot\_test\_data\_dist(test\_attributes)
143. # ### Check for duplicate rows
145. duplicateRowsDF = df\_santander[df\_santander.duplicated()]
147. **print**("No. of duplicate rows based on all columns are :")
148. **print**(duplicateRowsDF.shape[0])
150. duplicateRowsDF = df\_santander\_test[df\_santander\_test.duplicated()]
152. **print**("No. of duplicate rows based on all columns are :")
153. **print**(duplicateRowsDF.shape[0])
155. # ### Correlation Analysis
157. num\_train = df\_santander.columns.values[2:202]
158. num\_test = df\_santander\_test.columns.values[1:201]

161. # #### Correlation between train data
163. train\_corr = df\_santander[num\_train].corr().abs()
165. train\_corr = train\_corr.unstack()
166. train\_corr
168. train\_corr = train\_corr.sort\_values(kind="quicksort")
169. train\_corr
171. train\_corr = train\_corr.reset\_index()
172. train\_corr
174. train\_corr
176. # #### Correlation between test data
178. test\_corr = df\_santander\_test[num\_test].corr().abs()
179. test\_corr = test\_corr.unstack()
180. test\_corr = test\_corr.sort\_values(kind="quicksort")
181. test\_corr = test\_corr.reset\_index()
183. # #### Excluding correlation between same variables as that will be 1 always
184. train\_corr = train\_corr[train\_corr['level\_0']!=train\_corr['level\_1']]
185. train\_corr
186. test\_corr = test\_corr[test\_corr['level\_0']!=test\_corr['level\_1']]
187. test\_corr.iloc[:,2].describe()
188. train\_corr.iloc[:,2].describe()
189. train\_corr=df\_santander[num\_train].corr()
190. train\_corr
191. train\_corr=train\_corr.values.flatten()
192. train\_corr
193. train\_corr=train\_corr[train\_corr!=1]
194. test\_corr=df\_santander\_test[num\_test].corr()
195. test\_corr = test\_corr.values.flatten()
196. test\_corr=test\_corr[test\_corr!=1]
198. plt2.figure(figsize=(20,5))
199. sns.distplot(train\_corr,color="blue",label="train")
200. sns.distplot(test\_corr,color="red",label="test")
201. plt2.xlabel("Correlation values found in train & test data")
202. plt2.ylabel("Density")
203. plt2.title ("Correlation values in train & test data")
204. plt2.legend()

207. # ### Feature Importance
208. X = df\_santander.drop(columns=['ID\_code', 'target'], axis=1)
209. y = df\_santander['target']
211. X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=42)
213. rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42)
214. rf\_model.fit(X\_test,y\_test)
216. importance = pd.DataFrame(rf\_model.feature\_importances\_, columns = ['Feature Importance'])
218. columns = pd.DataFrame(data=X.columns.values);
220. columns['imporatance'] = importance
222. columns = columns.rename(columns={0: "Variable"})
224. columns = columns.rename(columns={'imporatance':'importance'})
226. columns.sort\_values(by=['importance'], inplace=True)
228. columns
230. # Var\_81 most important
232. X=df\_santander.drop(['ID\_code','target'],axis=1)
233. y=df\_santander['target']
235. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 2019)
237. sm = SMOTE(random\_state=42)
238. X\_smote,y\_smote=sm.fit\_sample(X\_train,y\_train)
239. X\_smote\_v,y\_smote\_v=sm.fit\_sample(X\_test,y\_test)
241. x = pd.concat([X\_smote,y\_smote],axis=1)
243. y = pd.concat([X\_smote\_v,y\_smote\_v], axis=1)
245. xy = pd.concat([x,y],axis=0)
247. X\_train.head()

250. # ### Feature Scaling
252. **from** sklearn.preprocessing **import** StandardScaler
254. X\_train = StandardScaler().fit\_transform(X\_train)
256. X\_test = StandardScaler().fit\_transform(X\_test)

259. # ## PCA
260. **from** sklearn.preprocessing **import** StandardScaler
262. x = StandardScaler().fit\_transform(xy.drop(['target'],axis=1))
264. **from** sklearn.decomposition **import** PCA
265. pca = PCA(n\_components=170)
266. principalComponents = pca.fit\_transform(x)
267. principalDf = pd.DataFrame(data = principalComponents)
268. **print**(sum(pca.explained\_variance\_))
269. **print**(sum(pca.explained\_variance\_ratio\_))
271. X1 = principalDf
272. y1 = xy['target']
273. X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC=train\_test\_split(X1,y1,random\_state=42)
275. plt2.scatter(principalDf.iloc[:, 0], principalDf.iloc[:, 1],
276. c=xy['target'], edgecolor='none', alpha=0.5)
277. plt2.xlabel('component 3')
278. plt2.ylabel('component 2')
279. plt2.colorbar();
281. fig = plt2.figure(figsize=(8, 6))
282. ax = fig.add\_subplot(111, projection='3d')
284. xs = principalDf.iloc[:,0]
285. ys = principalDf.iloc[:,1]
286. zs = principalDf.iloc[:,2]
287. # size = list(df\_santander['target'])
288. ax.scatter(xs, ys, zs, alpha=0.6, edgecolors='w',c=xy['target'],s=50)
290. ax.set\_xlabel('Principal Component 1')
291. ax.set\_ylabel('Principal Component 2')
292. ax.set\_zlabel('Principal Component 3')
294. per\_var = np.round(pca.explained\_variance\_ratio\_\*100, decimals=1)
296. labels = ['PC'+str(x) **for** x **in** range(1,len(per\_var)+1)]
298. fig= plt2.figure(figsize=(100,50))
299. plt2.bar(x=range(1,len(per\_var)+1), height=per\_var, tick\_label=labels)
300. plt2.ylabel('Percentage of Explained Variance')
301. plt2.xlabel('Principal Component')
302. plt2.title('Scree Plot')
303. plt2.show()
305. plt2.figure(figsize=(26,9))
306. plt2.plot(pca.explained\_variance\_ratio\_)
307. # plt2.xticks(range(80))
308. plt2.xlabel("Number of Features")
309. plt2.ylabel("Proportion of variance explained by additional feature")

312. # ## Model
314. **def** draw\_confusion\_mx(y\_test,y\_pred):
315. **print**('\n######### Confusion Matrix #########\n')
316. cm=pd.crosstab(y\_test,y\_pred)
317. **print**(cm)
319. **def** draw\_classification\_report(y\_test,y\_pred):
320. **print**('\n######### Classification Report #########\n')
321. **print**(classification\_report(y\_test,y\_pred))
323. **def** draw\_roc\_auc(y\_test,y\_pred):  ##y\_pred in form of probabilities
324. ns\_probs = [0 **for** \_ **in** range(len(y\_test))]
325. ns\_fpr, ns\_tpr, \_ = roc\_curve(y\_test, ns\_probs)
326. lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, y\_pred)
327. plt2.plot(ns\_fpr, ns\_tpr, linestyle='--', label='No Skill')
328. plt2.plot(lr\_fpr, lr\_tpr, marker='.', label='Logistic')
329. auc\_score=auc(lr\_fpr,lr\_tpr)
330. plt2.title('ROC(area=%0.3f)' %auc\_score)
332. plt2.xlabel('False Positive Rate')
333. plt2.ylabel('True Positive Rate')
335. plt2.legend()
337. plt2.show()
339. **def** draw\_precision\_recall(y\_test,y\_pred):
340. precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred)
342. # In matplotlib < 1.5, plt.fill\_between does not have a 'step' argument
343. step\_kwargs = ({'step': 'post'}
344. **if** 'step' **in** signature(plt2.fill\_between).parameters
345. **else** {})
346. plt2.step(recall, precision, color='b', alpha=0.2,
347. where='post')
348. plt2.fill\_between(recall, precision, alpha=0.2, color='b', \*\*step\_kwargs)
350. plt2.xlabel('Recall')
351. plt2.ylabel('Precision')
352. plt2.ylim([0.0, 1.05])
353. plt2.xlim([0.0, 1.0])
354. plt2.title(' Precision-Recall curve: PR\_AUC={0:0.3f}'.format( auc(recall, precision)))
355. plt2.show()
357. **def** fit\_N\_predict(model,X\_train,X\_test,y\_train,y\_test,model\_code,testData,PCA=0):
359. model.fit(X\_train,y\_train)
360. y\_pred = model.predict(X\_test)
361. y\_pred2 = model.predict\_proba(X\_test)
362. y\_pred2 = y\_pred2[:,1]
364. draw\_confusion\_mx(y\_test,y\_pred)
366. draw\_classification\_report(y\_test,y\_pred)
368. draw\_roc\_auc(y\_test,y\_pred2)
370. draw\_precision\_recall(y\_test,y\_pred2)
371. **if**(PCA == 0):
372. **if**(model\_code!="XGB"):
373. **print**('\n\nModel performance on test data:\n',)
374. **print**(model.predict(testData.drop(['ID\_code'],axis=1)))
375. **else**:
376. **print**('\n\nModel performance on test data:\n',)
377. **print**(model.predict(testData.drop(['ID\_code'],axis=1).values))



382. # ### Logistic Regression Model
384. lr\_model=LogisticRegression(random\_state=42,class\_weight = 'balanced')
386. **print**("LOGISTIC REGRESSION ON ORIGINAL DATASET\n\n")
387. fit\_N\_predict(lr\_model,X\_train,X\_test,y\_train,y\_test,model\_code='LR',testData=df\_santander\_test)
389. # ## Logistic Regression after applying SMOTE
391. **print**("LOGISTIC REGRESSION SMOTE DATASET\n\n")
392. fit\_N\_predict(lr\_model,X\_smote,X\_smote\_v,y\_smote,y\_smote\_v,model\_code='LR',testData=df\_santander\_test)
394. # ## LR on SMOTE dataset and PCA
396. **print**("LOGISTIC REGRESSION ON PCA+SMOTE DATASET\n\n")
397. fit\_N\_predict(lr\_model,X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC,model\_code='LR',testData = df\_santander\_test,PCA=1)

400. # # Decision Tree
402. tree\_clf = DecisionTreeClassifier(class\_weight='balanced', random\_state = 2019,
403. max\_features = 0.7, min\_samples\_leaf = 80)
405. **print**("DECISION TREE ON ORIGINAL DATASET\n\n")
406. fit\_N\_predict(tree\_clf,X\_train,X\_test,y\_train,y\_test,model\_code='DT',testData=df\_santander\_test)

409. # ### Decision Tree after applying SMOTE
411. **print**("DECISION TREE ON SMOTE DATASET\n\n")
412. fit\_N\_predict(tree\_clf,X\_smote,X\_smote\_v,y\_smote,y\_smote\_v,model\_code='DT',testData=df\_santander\_test)

415. # ## DT + SMOTE + PCA
417. **print**("DECISION TREE ON PCA+SMOTE DATASET\n\n")
418. fit\_N\_predict(tree\_clf,X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC,model\_code='DT',testData=df\_santander\_test,PCA=1)

421. # ## Random Forest
423. random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=2019, verbose=1,
424. class\_weight='balanced', max\_features = 0.5,
425. min\_samples\_leaf = 100,n\_jobs=-1)
427. **print**("RANDOM FOREST ON ORIGINAL DATASET\n\n")
428. fit\_N\_predict(random\_forest,X\_train,X\_test,y\_train,y\_test,model\_code='RF',testData=df\_santander\_test)
430. **print**("RANDOM FOREST ON SMOTE DATASET\n\n")
431. fit\_N\_predict(random\_forest,X\_smote,X\_smote\_v,y\_smote,y\_smote\_v,model\_code='RF',testData=df\_santander\_test)
433. # ### RF + SMOTE + PCA
435. **print**("RANDOM FOREST ON PCA+SMOTE DATASET\n\n")
436. fit\_N\_predict(random\_forest,X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC,model\_code='RF',testData=df\_santander\_test,PCA=1)

439. # ## NaiveBayes
441. **from** sklearn.naive\_bayes **import** GaussianNB
442. NB\_model = GaussianNB()
444. **print**("NAIVE BAYES ON ORIGINAL DATASET\n\n")
445. fit\_N\_predict(NB\_model,X\_train,X\_test,y\_train,y\_test,model\_code='NB',testData=df\_santander\_test)
447. **print**("NAIVE BAYES ON SMOTE DATASET\n\n")
448. fit\_N\_predict(NB\_model,X\_smote,X\_smote\_v,y\_smote,y\_smote\_v,model\_code='NB',testData=df\_santander\_test)
450. **print**("NAIVE BAYES ON PCA+SMOTE DATASET\n\n")
451. fit\_N\_predict(NB\_model,X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC,model\_code='NB',testData=df\_santander\_test,PCA=1)

454. # ## XGBoost
456. **from** xgboost **import** XGBClassifier
458. XGB = XGBClassifier(learning\_rate =0.1,
459. n\_estimators=800,
460. max\_depth=5,
461. min\_child\_weight=1,
462. gamma=0,
463. subsample=0.8,
464. colsample\_bytree=0.8,
465. objective= 'binary:logistic',
466. nthread=4,
467. seed=27,scale\_pos\_weight=2)
469. **print**("XGBOOST CLASSIFIER ON ORIGINAL DATASET\n\n")
470. fit\_N\_predict(XGB,X\_train,X\_test,y\_train,y\_test,model\_code='XGB',testData=df\_santander\_test)
472. **print**("XGBOOST CLASSIFIER ON SMOTE DATASET\n\n")
473. fit\_N\_predict(XGB,X\_smote,X\_smote\_v,y\_smote,y\_smote\_v,model\_code='XGB\_SM',testData=df\_santander\_test)
475. **print**("XGBOOST CLASSIFIER ON SMOTE ON PCA DATASET\n\n")
476. fit\_N\_predict(XGB,X\_train\_PC,X\_test\_PC,y\_train\_PC,y\_test\_PC,model\_code='XGB',testData=df\_santander\_test,PCA=1)

R Code:

Link:

**https://drive.google.com/drive/folders/1TjY1rnyMhfuuVPNprOJXbuFeHo3OB3ct?usp=sharing**