Santander Customer Transaction Prediction

Sourabh Shivendra Pathak

11th Dec 2019

CONTENTS

Introduction

1.1 Background

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?

1.2 Problem Statement

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

1.3 DATA

Our goal is to develop a Classification model which will identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted based on following dataset

Rows: 200000 entries, 0 to 199999

Columns: 202 entries,
ID_code, target, var_0 to var_199

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267

Here ID_code is the index column target is categorical column with value {0,1} and var_0 to var_199 is numeric columns.

Methodology

2.1 Data Preprocessing and EDA

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and visual representation

Steps Involved in Data Preprocessing:

1. Data Cleaning:

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

2.DataTransformation:

This step is taken to transform the data in appropriate forms suitable for mining process. This involves following ways:

3.DataReduction:

Since data mining is a technique that is used to handle huge amount of data. While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we use data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

Let's discuss one by one the data preprocessing technique and EDA that has been used in this project

2.1.1 Missing value analysis

This situation arises when some data is missing in the dataset. It can be handled in various ways.
Some of them are:

1. **Ignore the entry:**
   This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.

2. **Fill the Missing values:**
   There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

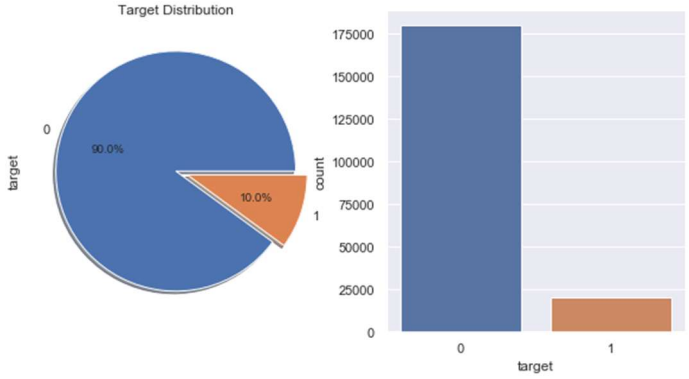
```
In [8]: missing_data_table
Out[8]:
```

	missing_val_status
target	0
var_0	0
var_1	0
var_2	0
var_3	0
...	...
var_195	0
var_196	0
var_197	0
var_198	0
var_199	0

There is no missing value in the dataset, so we don't have to go for any missing data imputation

2.1.2 Data distribution analysis

Since it is classification problem we need to check whether the dataset is balanced or not because few models is sensitive to data imbalance.



Clearly the data is imbalanced to deal with imbalanced dataset we can use following approach

Approach to handling Imbalanced Datasets

1 Data Level approach: Resampling Techniques

Dealing with imbalanced datasets entails strategies such as improving classification algorithms or balancing classes in the training data (data preprocessing) before providing the data as input to the machine learning algorithm. The later technique is preferred as it has wider application.

The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done in order to obtain approximately the same number of instances for both the classes. Let us look at a few resampling techniques:

Algorithmic Ensemble Techniques

The above section deals with handling imbalanced data by resampling original data to provide balanced classes. In this section, we are going to look at an alternate approach i.e. Modifying existing classification algorithms to make them appropriate for imbalanced data sets.

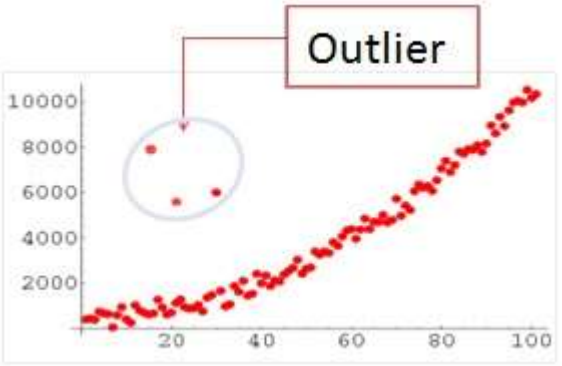
The main objective of ensemble methodology is to improve the performance of single classifiers. The approach involves constructing several two stage classifiers from the original data and then aggregate their predictions.

To learn more about these techniques you can follow these links

- https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/
- https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

2.1.3 Outlier analysis

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution as shown in figure below.



The easiest way to detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms can help to detect outliers. Alternatively, we can use mean and standard deviation to list out the outliers.It can be handled in various ways. Some of them are:

1. **Ignore the entry:**

   This approach is suitable only when the dataset we have is quite large and outliers are in some data

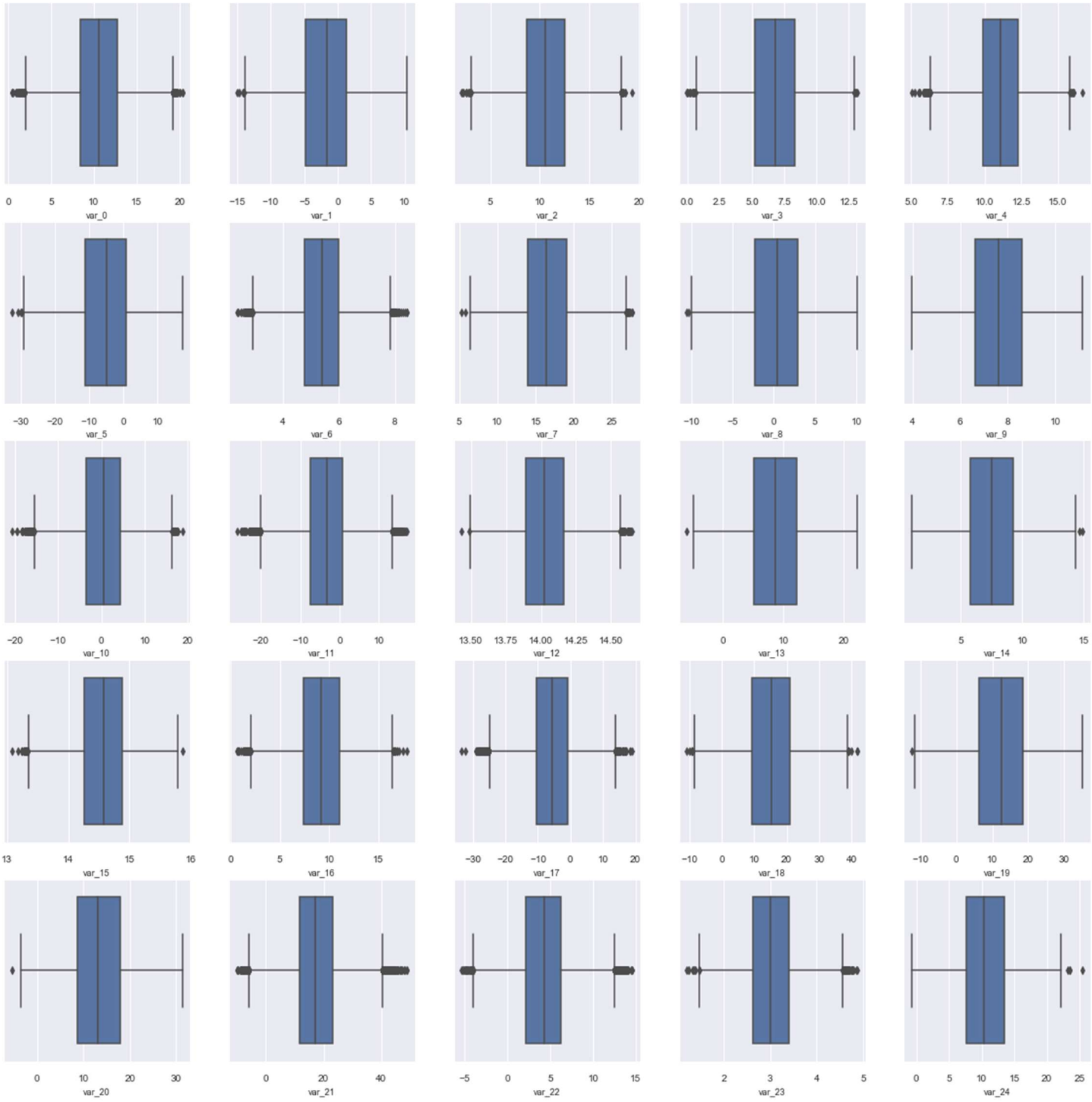
2. **Delete the outlier and Fill the Missing values:**

   There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

]:

	variables	Q3	Q1	IQR	minline	maxline	outliers
0	var_0	12.758200	8.453850	4.304350	1.997325	19.214725	104.0
1	var_1	1.358625	-4.740025	6.098650	-13.888000	10.506600	6.0
2	var_2	12.516700	8.722475	3.794225	3.031137	18.208038	49.0
3	var_3	8.324100	5.254075	3.070025	0.649038	12.929137	22.0
4	var_4	12.261125	9.883175	2.377950	6.316250	15.828050	76.0
...	...	...	...	...	...	...	...
195	var_195	0.829600	-1.170700	2.000300	-4.171150	3.830050	132.0
196	var_196	6.556725	-1.946925	8.503650	-14.702400	19.312200	0.0
197	var_197	9.593300	8.252800	1.340500	6.242050	11.604050	51.0
198	var_198	18.064725	13.829700	4.235025	7.477163	24.417262	94.0
199	var_199	4.836800	-11.208475	16.045275	-35.276387	28.904712	20.0

200 rows × 7 columns

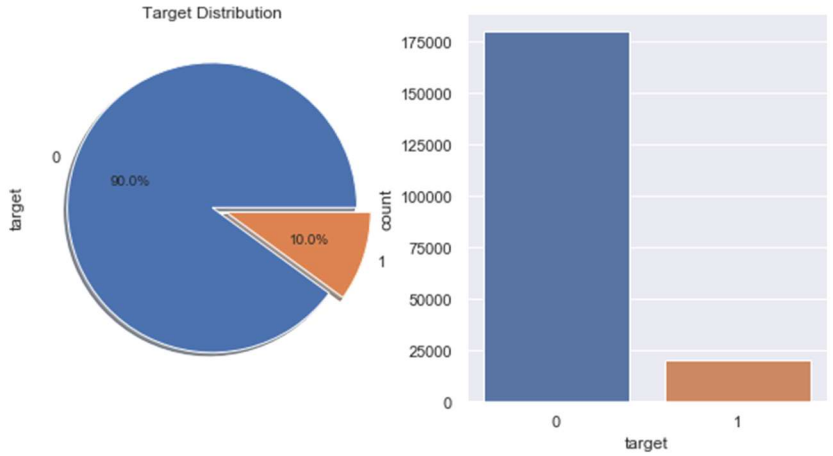


There are outliers in the dataset in the data set so we will check whether these outliers are contributing in target variable or they are just an exception or wrong observation

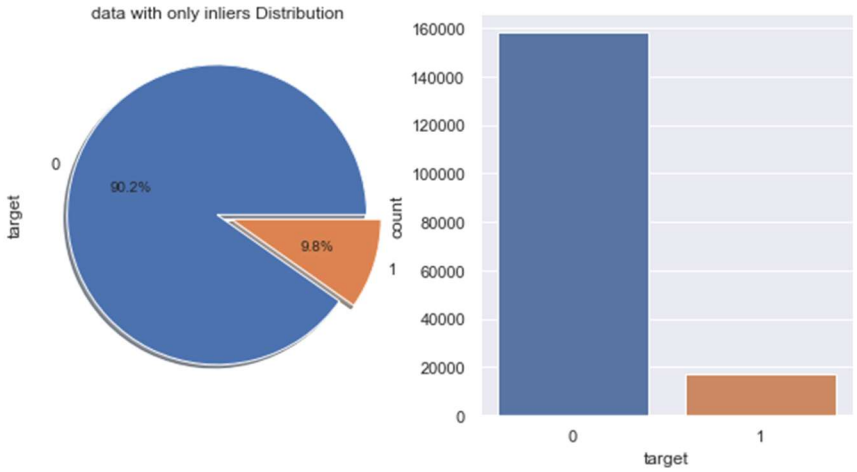
Since we don't have any domain idea we will check whether after removing the outliers is there any change in the target variable for that we will make two datasets

1. df_in: - this data set contains all the data which are not have any outliers
2. df_out: - this data set contains all the data that have outliers.

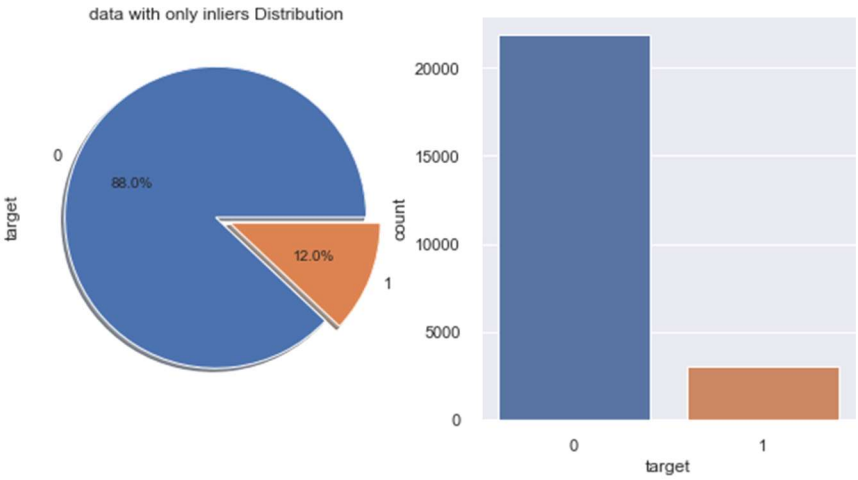
Now we will check what change happened in the target variable after removing outliers



Original data set



Dataset with inliers



Dataset with outliers

So, we can see that their some data loss due to removal of outliers moreover there are significant amount data in outliers with target value 1 so we will try make model with outliers as well as without outliers.

2.1.4 Feature Selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

Here are some benefit of feature selection

· **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.

· **Improves Accuracy**: Less misleading data means modeling accuracy improves.

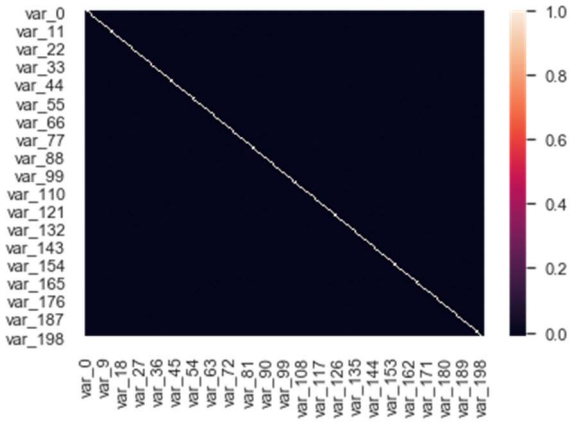
· **Reduces Training Time**: fewer data points reduce algorithm complexity and algorithms train faster.

The numerical variable should have following property

1-high correlation with dependent variables

2-low correlation with other independent variables

Let's check correlation between variable using correlation matrix and heatmap



As we can clearly see there is no correlation between any variable, so we can't reduce the feature.

2.1.5 Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, most classifiers calculate the distance between two points by the distance. If one of the features has a broad range of values, the distance will be governed by this feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_190	var_191	var_192	var_193	var_194	var
0	No	0.402177	0.292074	0.584832	0.360750	0.541944	0.428616	0.445950	0.596853	0.251762	...	0.548738	0.305883	0.647402	0.424233	0.521838	0.22
2	No	0.383803	0.458737	0.596195	0.589273	0.449393	0.432917	0.815398	0.399996	0.251797	...	0.489558	0.640687	0.469293	0.424001	0.685452	0.91
3	No	0.526233	0.483233	0.389996	0.532375	0.660389	0.588981	0.592615	0.415186	0.205050	...	0.549945	0.350653	0.353558	0.412239	0.761405	0.36
4	No	0.455137	0.510803	0.648538	0.486814	0.627993	0.681244	0.612404	0.627517	0.807079	...	0.319483	0.625223	0.248029	0.758024	0.243878	0.33
5	No	0.550401	0.476370	0.630965	0.649151	0.489398	0.705195	0.327156	0.429934	0.674031	...	0.133156	0.457491	0.909196	0.368593	0.331889	0.65

2.1.6 Sampling

In this project we have used

- Random Under Sampling
- Random Over Sampling
- Synthetic Minority Oversampling

Random Under-Sampling

Random Undersampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

- **Advantages**
  - It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

- **Disadvantages**
  - It can discard potentially useful information which could be important for building rule classifiers.
  - The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representative of the population. Thereby, resulting in inaccurate results with the actual test data set.

Random Under Sampling

```
[23]: from imblearn.under_sampling import RandomUnderSampler
      rus = RandomUnderSampler(return_indices=True)
      X_rus, y_rus, id_rus = rus.fit_sample(X_train,y_train)

[24]: y = pd.Series(y_rus)
      y.value_counts()

t[24]: 1    14049
       0    14049
       dtype: int64
```

Random Over-Sampling

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample

- **Advantages**
  - Unlike under sampling this method leads to no information loss.
  - Outperforms under sampling

- **Disadvantages**

- It increases the likelihood of overfitting since it replicates the minority class events.

Random over Sampling

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
X_ros, y_ros = ros.fit_sample(X_train,y_train)
```

```
y = pd.Series(y_ros)
y.value_counts()
```

```
1    125951
0    125951
dtype: int64
```

Cluster-Based Over Sampling

In this case, the K-means clustering algorithm is independently applied to minority and majority class instances. This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.

- **Advantages**
  - This clustering technique helps overcome the challenge between class imbalance. Where the number of examples representing positive class differs from the number of examples representing a negative class.
  - Also, overcome challenges within class imbalance, where a class is composed of different sub clusters. And each sub cluster does not contain the same number of examples.

- **Disadvantages**
  - The main drawback of this algorithm, like most oversampling techniques is the possibility of over-fitting the training data

Informed Over Sampling: Synthetic Minority Over-sampling Technique

This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic

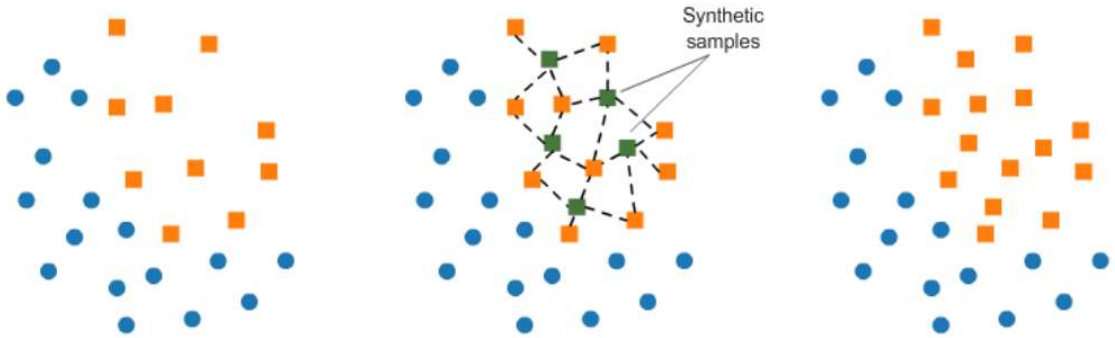
instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

- **Advantages**
  - Mitigates the problem of overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances
  - No loss of useful information

- **Disadvantages**
  - While generating synthetic examples SMOTE does not take into consideration neighboring examples from other classes. This can result in increase in overlapping of classes and can introduce additional noise
  - SMOTE is not very effective for high dimensional data

Over-sampling: SMOTE

SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class, based on those that already exist. It works randomly picingk a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.




```
from imblearn.over_sampling import SMOTE
smote = SMOTE(ratio='minority')
X_sm, y_sm = smote.fit_sample(X_train,y_train)
```

```
y = pd.Series(y_sm)
y.value_counts()
```

```
1    125951
0    125951
dtype: int64
```

I didn't try other sampling technique as the dataset is large and my machine was not allowing me to do so.

Modeling

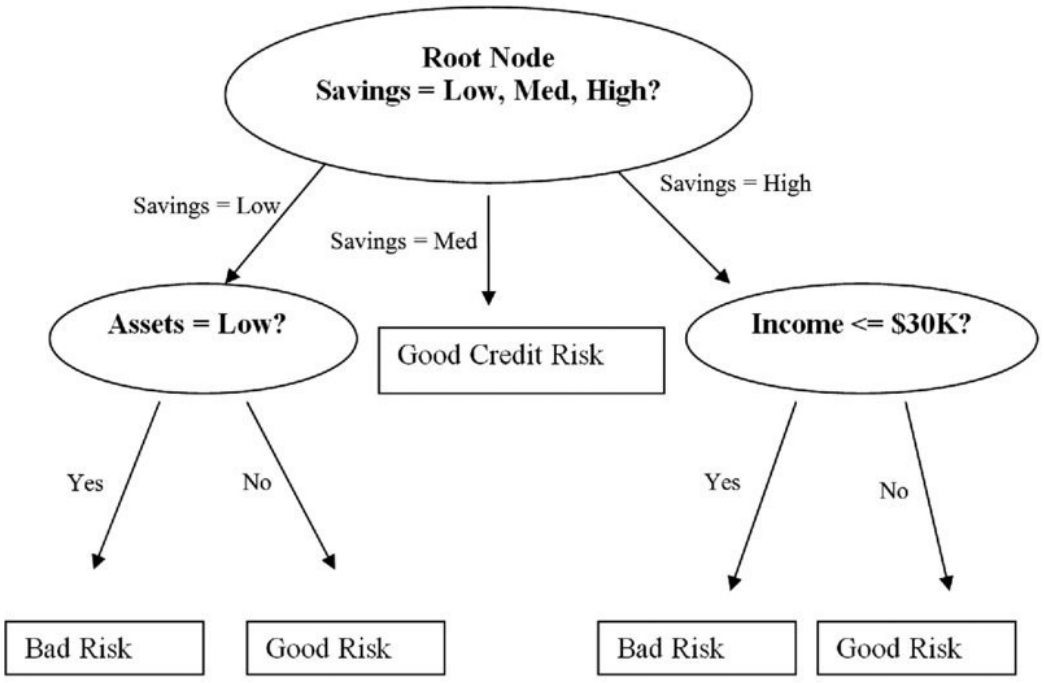
3.1 Decision Tree classification

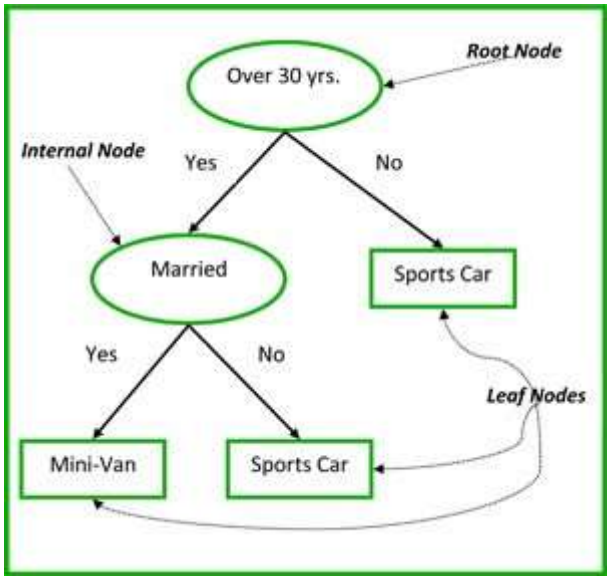
A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). Decision Tree algorithms are referred to as **CART (Classification and Regression Trees)**.

“The possible solutions to a given problem emerge as the leaves of a tree, each node representing a point of deliberation and decision.”





Common terms used with Decision trees:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report as CLR,average_precision_score,precision_recall_curve,roc_curve,auc,roc
```

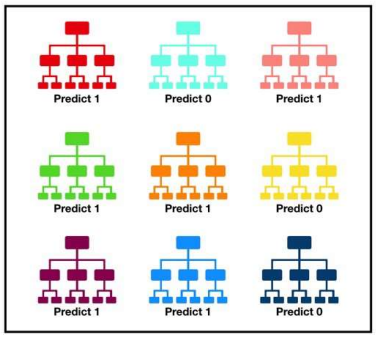
```
def c50_modelcreation(self,st,X_train, X_test, y_train, y_test):
    #Decision Tree
    if st=='unsampled':
        C50_model = tree.DecisionTreeClassifier(class_weight='balanced',criterion='entropy',max_features = 0.7, min_samples_
    else:
        C50_model = tree.DecisionTreeClassifier(criterion='entropy',max_features = 0.7, min_samples_leaf = 80).fit(X_train, y

    self.C50 = C50_model
    #predict new test cases
    C50_Predictions = C50_model.predict(X_test)
    c50_proba = C50_model.predict_proba(X_test)[:,1]
    cnm = confusion_matrix(y_test,C50_Predictions)
    TN,FP,FN,TP = cnm.ravel()
    accuracy = ((TP+TN)/len(X_test))*100
    Specificity = (TN/(TN+FP))*100
    Recall = (TP/(TP+FN))*100
    FPR = (FP/(FP+TN))*100
    FNR = (FN/(FN+TP))*100
    prec = (TP/(TP+FP))*100
    print('C50_Tree')
    print(st)
    self.modelperformance = self.modelperformance.append({'Model':'C50_Tree','ST':st,'TN':TN,'FN':FN,'TP':TP,'FP':FP,\
                                            'accuracy':accuracy,'Specificity':Specificity,'Recall':Recall,'FPR':FPR,\
                                           'FNR':FNR,'precession':prec}, ignore_index=True)
    fig= plt.figure(figsize=(10,7))
    sns.heatmap(cnm,cmap="Blues",annot=True,fmt="d",linewidths=1,linecolor='black',annot_kws={"color":'RED','size':20})
    plt.title("Confusion matrix")
```

We will discuss the result of all the model after discussing every model

3.2 Random forest Classification

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).



Tally: Six 1s and Three 0s
Prediction: 1

Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.

2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

```
def RF_modelcreation(self,st,X_train, X_test, y_train, y_test):
    if st=='unsampled':
        RF_model = RandomForestClassifier(n_estimators=100, random_state=2019, verbose=1,
                                          class_weight='balanced', max_features = 0.5,
                                           min_samples_leaf = 100).fit(X_train, y_train)
    else:
         RF_model = RandomForestClassifier(n_estimators=100, verbose=1,
                                           max_features = 0.5,min_samples_leaf = 100).fit(X_train, y_train)

    self.RF = RF_model
    RF_Predictions = RF_model.predict(X_test)
    RF_proba = RF_model.predict_proba(X_test)[:,1]
    cnm = confusion_matrix(y_test,RF_Predictions)
    TN,FP,FN,TP = cnm.ravel()
    accuracy = ((TP+TN)/len(X_test))*100
    Specificity = (TN/(TN+FP))*100
    Recall = (TP/(TP+FN))*100
    FPR = (FP/(FP+TN))*100
    FNR = (FN/(FN+TP))*100
    prec = (TP/(TP+FP))*100
    print('RF_model')
    print(st)
    self.modelperformance = self.modelperformance.append({'Model':'RF','ST':st,'TN':TN,'FN':FN,'TP':TP,'FP':FP,\
                                                          'accuracy':accuracy,'Specificity':Specificity,'Recall':Recall,'FPR':FPR,\
                                                         'FNR':FNR,'prec':prec}, ignore_index=True)
    clr = CLR(y_test,RF_Predictions)
    fig= plt.figure(figsize=(10,7))
    sns.heatmap(cnm,cmap="Blues",annot=True,fmt="d",linewidths=1,linecolor='black',annot_kws={"color":'RED','size':20})
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
```

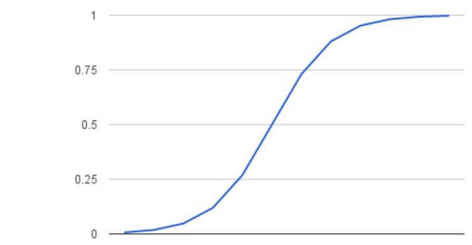
3.3 Logistic Regression Classification

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.



Logistic Function

Now that we know what the logistic function is, let's see how it is used in logistic regression.

Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable**: This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge**: It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

```
def LR_modelcreation(self,st,X_train, X_test, y_train, y_test):

    if st=='unsampled':
        LR_model = LogisticRegression(class_weight='balanced').fit(X_train, y_train)
    else:
        LR_model = LogisticRegression().fit(X_train, y_train)


    self.LR = LR_model
    LR_Predictions=LR_model.predict(X_test)
    LR_proba = LR_model.predict_proba(X_test)[:,1]
    print('LR model prediction')
    print(st)
    cnm = confusion_matrix(y_test,LR_Predictions)
    TN,FP,FN,TP = cnm.ravel()
    accuracy = ((TP+TN)/len(X_test))*100
    Specificity = (TN/(TN+FP))*100
    Recall = (TP/(TP+FN))*100
    FPR = (FP/(FP+TN))*100
    FNR = (FN/(FN+TP))*100
    prec = (TP/(TP+FP))*100
    self.modelperformance = self.modelperformance.append({'Model':'LR','ST':st,'TN':TN,'FN':FN,'TP':TP,'FP':FP,\
                                                'accuracy':accuracy,'Specificity':Specificity,'Recall':Recall,'FPR':FPR,\
                                               'FNR':FNR,'prec':prec}, ignore_index=True)
    fig= plt.figure(figsize=(10,7))
    sns.heatmap(cnm,cmap="Blues", annot=True,fmt="d",linewidths=1,linecolor='black',annot_kws={"color":'RED','size':20})
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
    print(CLR(y_test,LR_Predictions))
```

3.4 KNN Classification

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

“Birds of a feather flock together.”

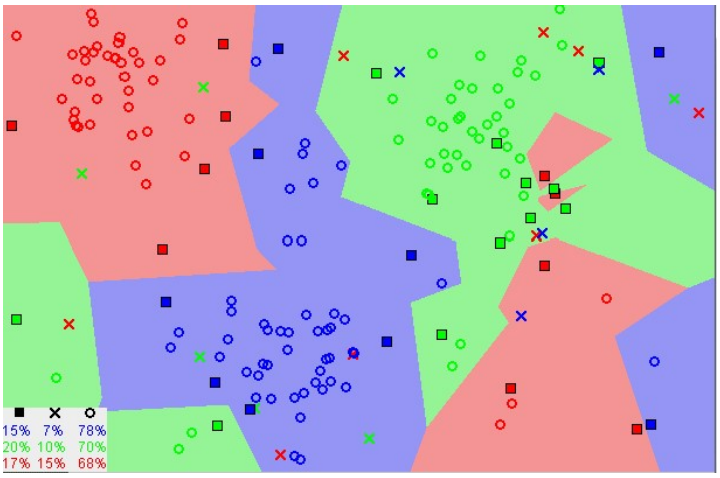


Image showing how similar data points typically exist close to each other

Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness)
KNN makes predictions using the training dataset directly.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.
Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

$$EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))$$

Other popular distance measures include:

- **Hamming Distance**: Calculate the distance between binary vectors (more).
- **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance (more).
- **Minkowski Distance**: Generalization of Euclidean and Manhattan distance (more).

There are many other distance measures that can be used, such as Tanimoto, Jaccard, Mahalanobis and cosine distance. You can choose the best distance metric based on the properties of your data. If you are unsure, you can experiment with different distance metrics and different values of K together and see which mix results in the most accurate models.
Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).

The value for K can be found by algorithm tuning. It is a good idea to try many different values for K (e.g. values from 1 to 21) and see what works best for your problem.

The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.

KNN has been around for a long time and has been very well studied. As such, different disciplines have different names for it, for example:

- **Instance-Based Learning**: The raw training instances are used to make predictions. As such KNN is often referred to as instance-based learning or a case-based learning (where each training instance is a case from the problem domain).
- **Lazy Learning**: No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.
- **Non-Parametric**: KNN makes no assumptions about the functional form of the problem being solved. As such KNN is referred to as a non-parametric machine learning algorithm.

KNN for Classification

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance. For example, in a binary classification problem (class is 0 or 1):

$$p(class=0) = count(class=0) / (count(class=0)+count(class=1))$$

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

Curse of Dimensionality

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large.

Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables x1 and x2, the input space would be 2-dimensional.

As the number of dimensions increases the volume of the input space increases at an exponential rate.

In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the “Curse of Dimensionality “.

```
def KNN_modelcreation(self,st,X_train, X_test, y_train, y_test):
    KNN_model = KNeighborsClassifier(n_neighbors = 9).fit(X_train, y_train)
    self.KNN = KNN_model
    KNR_Prediction = KNN_model.predict(X_test)
    KNR_proba = KNN_model.predict_proba(X_test)[:,1]
    print('KNN model prediction')
    print(st)
    cnm =  confusion_matrix(y_test,KNR_Prediction)
    TN,FP,FN,TP = cnm.ravel()
    accuracy = ((TP+TN)/len(X_test))*100
    Specificity = (TN/(TN+FP))*100
    Recall = (TP/(TP+FN))*100
    FPR = (FP/(FP+TN))*100
    FNR = (FN/(FN+TP))*100
    prec = (TP/(TP+FP))*100
    self.modelperformance = self.modelperformance.append({'Model':'LR','ST':st,'TN':TN,'FN':FN,'TP':TP,'FP':FP,\
                                            'accuracy':accuracy,'Specificity':Specificity,'Recall':Recall,'FPR':FPR,\
                                           'FNR':FNR,'prec':prec}, ignore_index=True)
    fig= plt.figure(figsize=(10,7))
    sns.heatmap(cnm,cmap="Blues",annot=True,fmt="d",linewidths=1,linecolor='black',annot_kws={"color":'RED','size':20})
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
    print(CLR(y_test,KNR_Prediction))
    Model_Creation.plot_precision_recall(y_test, KNR_proba)
```

3.5 Naive Bayes Classification

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

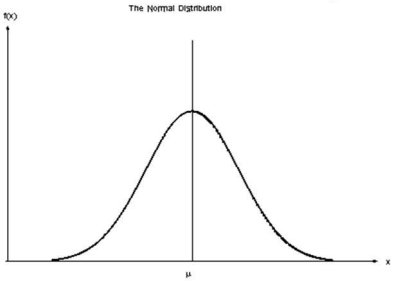
This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.



Gaussian Distribution(Normal Distribution)

Since the way the values are present in the dataset changes, the formula for conditional probability changes to,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

```
def NB_modelcreation(self,st,X_train, X_test, y_train, y_test):
    NB_model = GaussianNB().fit(X_train, y_train)
    self.NB = NB_model
    NB_Prediction = NB_model.predict(X_test)
    NB_proba = NB_model.predict_proba(X_test)[:,1]
    print('NB model prediction')
    print(st)
    cnm =  confusion_matrix(y_test,NB_Prediction)
    TN,FP,FN,TP = cnm.ravel()
    accuracy = ((TP+TN)/len(X_test))*100
    Specificity = (TN/(TN+FP))*100
    Recall = (TP/(TP+FN))*100
    FPR = (FP/(FP+TN))*100
    FNR = (FN/(FN+TP))*100
    prec = (TP/(TP+FP))*100
    self.modelperformance = self.modelperformance.append({'Model':'LR','ST':st,'TN':TN,'FN':FN,'TP':TP,\
                                                          'FP':FP,'accuracy':accuracy,'Specificity':Spec
                                                          'Recall':Recall,'FPR':FPR,'FNR':FNR,'prec':pre
    fig= plt.figure(figsize=(10,7))
    sns.heatmap(cnm,cmap="Blues",annot=True,fmt="d",linewidths=1,linecolor='black',annot_kws={"color":'RED','size':20})
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
    print(CLR(y_test,NB_Prediction))
    Model_Creation.plot_precision_recall(y_test, NB_proba)
    Model_Creation.plot_roc_curve(y_test, NB_proba)
```

Model Selection

4.1 Selection Metrics

In a typical binary diagnostic test, a positive or negative diagnosis is made for each individual (e.g., patient, subject, or unit). When the diagnosis is compared to the true (known) condition, there are four possible outcomes: true positive, true negative, false positive, false negative.

Classification Table

		Test Result: Positive	Test Result: Negative
True Condition	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

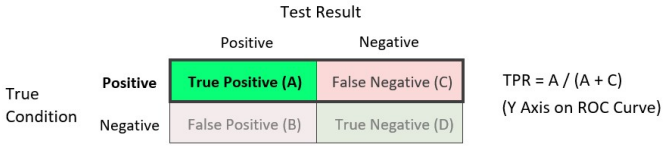
When all of the individuals are assigned to the four outcomes, a count for each outcome is produced. The four counts are labeled A, B, C, and D in the table below

		Test Result: Positive	Test Result: Negative	Total
True Condition	Positive	True Positive (A)	False Negative (C)	A + C
	Negative	False Positive (B)	True Negative (D)	B + D
	Total	A + B	C + D	A + B + C + D

Various rates (proportions) can be used to describe a classification table. Some rates are based on the true condition, some rates are based on the predicted condition, and some rates are based on the whole table. These rates will be described in the following sections.

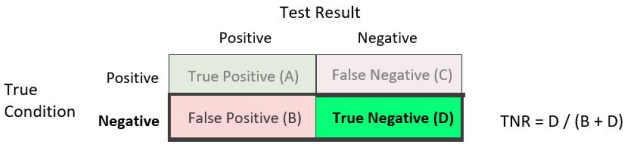
True Positive Rate (TPR) or Sensitivity = A / (A + C)

The true positive rate is the proportion of the individuals with a known positive condition for which the test result is positive.



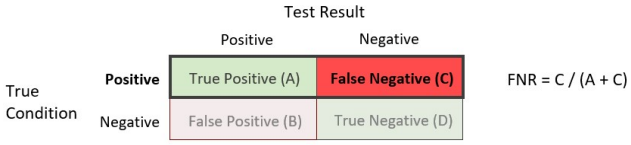
True Negative Rate (TNR) or Specificity = D / (B + D)

The true negative rate is the proportion of the individuals with a known negative condition for which the test result is negative. This rate is often called the specificity.



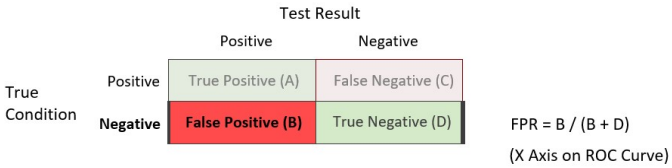
False Negative Rate (FNR) or Miss Rate = C / (A + C)

The false negative rate is the proportion of the individuals with a known positive condition for which the test result is negative. This rate is sometimes called the miss rate.



False Positive Rate (FPR) or Fall-out = B / (B + D)

The false positive rate is the proportion of the individuals with a known negative condition for which the test result is positive. This rate is sometimes called the fall-out.



Precision

Precision means the percentage of your results which are relevant.

Recall

Recall refers to the percentage of total relevant results correctly classified

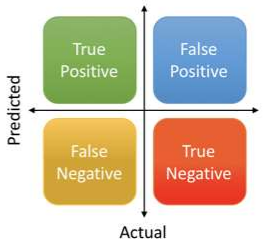
Accuracy

Accuracy is the fraction of predictions our model got right.

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

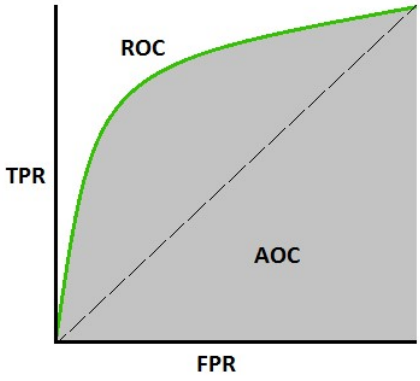
$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



AUC - ROC Curve

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



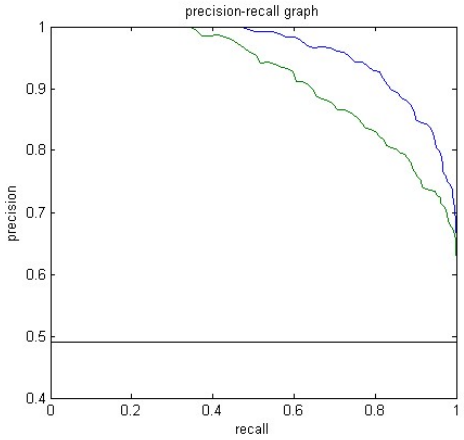
How to speculate the performance of the model?

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

Precision-Recall curve

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.

A no-skill classifier is one that cannot discriminate between the classes and would predict a random class or a constant class in all cases. The no-skill line changes based on the distribution of the positive to negative classes. It is a horizontal line with the value of the ratio of positive cases in the dataset.



It is desired that the algorithm should have both high precision, and high recall. However, most machine learning algorithms often involve a trade-off between the two. A good PR curve has greater AUC (area under curve). In the figure above, the classifier corresponding to the blue line has better performance than the classifier corresponding to the green line.

It is important to note that the classifier that has a higher AUC on the ROC curve will always have a higher AUC on the PR curve as well.

4.2 Model Evaluation

Models Used:

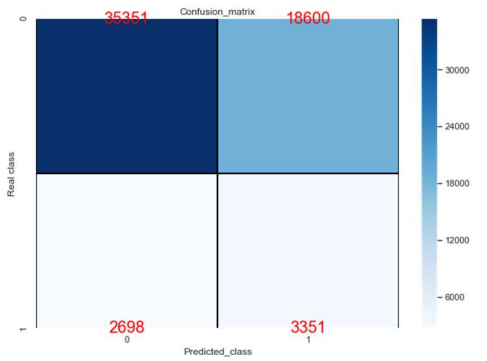
- Tree
- Random Forrest
- Naïve Bayes
- Logistic Regression

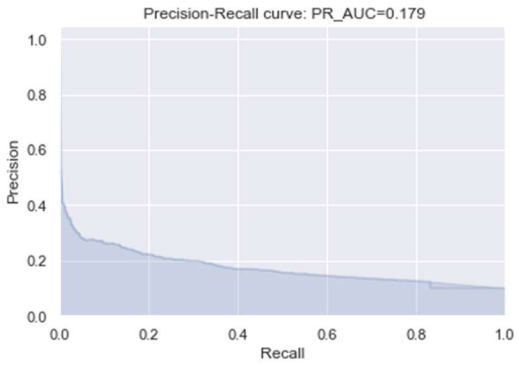
Below is the Table of performance of all the model with different sampling Technique.

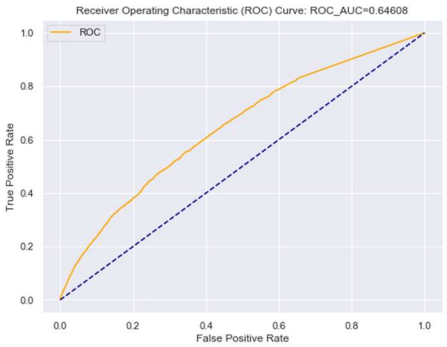
	Model	ST	Outliers	TN	FN	TP	FP	FNR	FPR	Recall	Specificity	accuracy	precession
0	C50_Tree	unsampled	True	35691	2768	3281	18260	45.759630	33.845526	54.240370	66.154474	64.953333	15.231419
1	RF	unsampled	True	46548	2978	3071	7403	49.231278	13.721710	50.768722	86.278290	82.698333	29.320222
2	LR	unsampled	True	42412	1385	4664	11539	22.896347	21.387926	77.103653	78.612074	78.460000	28.784793
3	NB	unsampled	True	53091	3868	2181	860	63.944454	1.594039	36.055546	98.405961	92.120000	71.719829
4	C50_Tree	RUS	True	32216	2169	3880	21735	35.857166	40.286556	64.142834	59.713444	60.160000	15.147375
5	RF	RUS	True	36856	1773	4276	17095	29.310630	31.686160	70.689370	68.313840	68.553333	20.008423
6	LR	RUS	True	42161	1380	4669	11790	22.813688	21.853163	77.186312	78.146837	78.050000	28.367459
7	NB	RUS	True	43773	1210	4839	10178	20.003306	18.865267	79.996694	81.134733	81.020000	32.223480
8	C50_Tree	ROS	True	40513	3437	2612	13438	56.819309	24.907787	43.180691	75.092213	71.875000	16.274143
9	RF	ROS	True	46508	3047	3002	7443	50.371962	13.795852	49.628038	86.204148	82.516667	28.741024
10	LR	ROS	True	42400	1377	4672	11551	22.764093	21.410168	77.235907	78.589832	78.453333	28.798619
11	NB	ROS	True	44011	1214	4835	9940	20.069433	18.424126	79.930567	81.575874	81.410000	32.724196
12	C50_Tree	SMOTE	True	39600	3930	2119	14351	64.969416	26.600063	35.030584	73.399937	69.531667	12.865817
13	RF	SMOTE	True	49233	4618	1431	4718	76.343197	8.744972	23.656803	91.255028	84.440000	23.272077
14	LR	SMOTE	True	42934	1458	4591	11017	24.103158	20.420381	75.896842	79.579619	79.208333	29.414403
15	NB	SMOTE	True	51520	5474	575	2431	90.494297	4.505941	9.505703	95.494059	86.825000	19.128410
16	C50_Tree	unsampled	False	31093	2232	2853	16354	43.893805	34.467933	56.106195	65.532067	64.619660	14.853959
17	RF	unsampled	False	41154	2610	2475	6293	51.327434	13.263220	48.672566	86.736780	83.052235	28.227646
18	LR	unsampled	False	37178	1188	3897	10269	23.362832	21.643097	76.637168	78.356903	78.190436	27.509530
19	NB	unsampled	False	46691	3311	1774	756	65.113078	1.593357	34.886922	98.406643	92.258052	70.118577
20	C50_Tree	RUS	False	27945	1785	3300	19502	35.103245	41.102704	64.896755	58.897296	59.478032	14.472415
21	RF	RUS	False	31126	1409	3676	16321	27.708948	34.398381	72.291052	65.601619	66.249143	18.382757
22	LR	RUS	False	37166	1200	3885	10281	23.598820	21.668388	76.401180	78.331612	78.144750	27.424820
23	NB	RUS	False	38589	1080	4005	8858	21.238938	18.669252	78.761062	81.330748	81.082007	31.135816
24	C50_Tree	ROS	False	35683	2967	2118	11764	58.348083	24.793981	41.651917	75.206019	71.958045	15.257168
25	RF	ROS	False	41038	2596	2489	6409	51.052114	13.507703	48.947886	86.492297	82.858067	27.972578
26	LR	ROS	False	37181	1188	3897	10266	23.362832	21.636774	76.637168	78.363226	78.196147	27.515357
27	NB	ROS	False	38658	1070	4015	8789	21.042281	18.523827	78.957719	81.476173	81.232392	31.357388
28	C50_Tree	SMOTE	False	35286	3332	1753	12161	65.526057	25.630704	34.473943	74.369296	70.507500	12.598821
29	RF	SMOTE	False	42769	3974	1111	4678	78.151426	9.859422	21.848574	90.140578	83.530039	19.191570
30	LR	SMOTE	False	37527	1232	3853	9920	24.228122	20.907539	75.771878	79.092461	78.771035	27.975024
31	NB	SMOTE	False	45286	4639	446	2161	91.229105	4.554556	8.770895	95.445444	87.055509	17.107787

C50 Tree

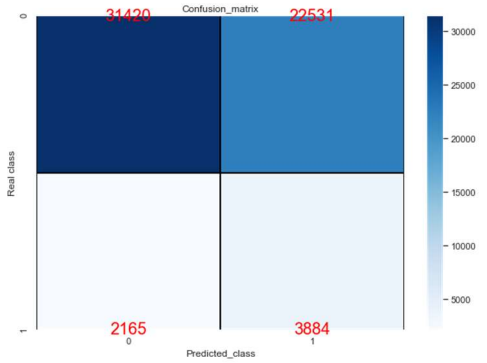
Unsampled Data

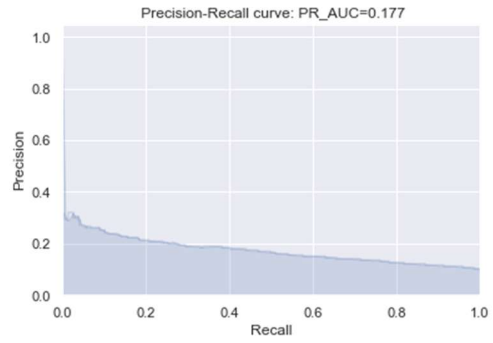


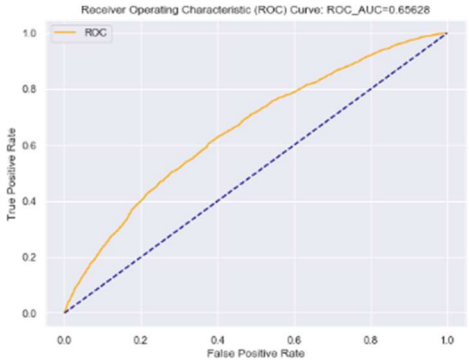




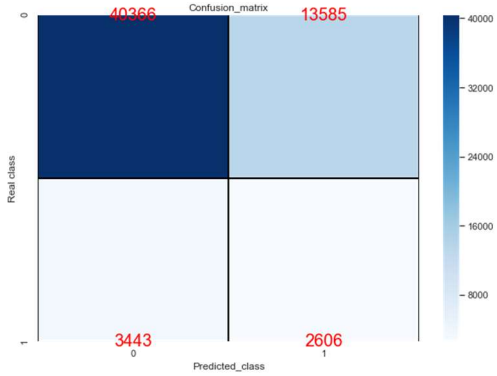
RUS

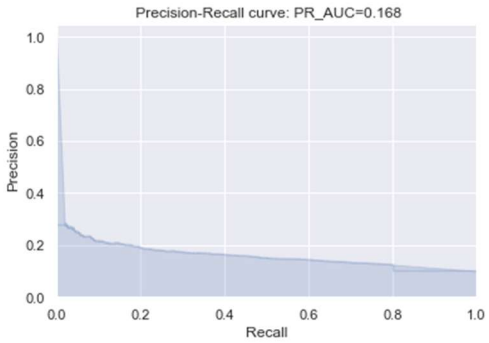


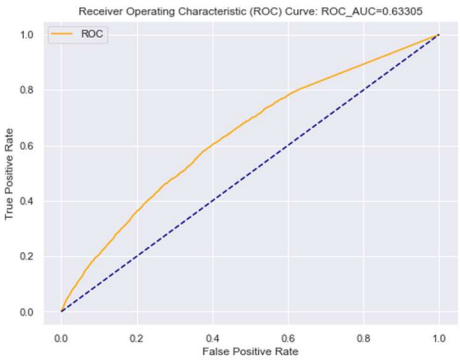




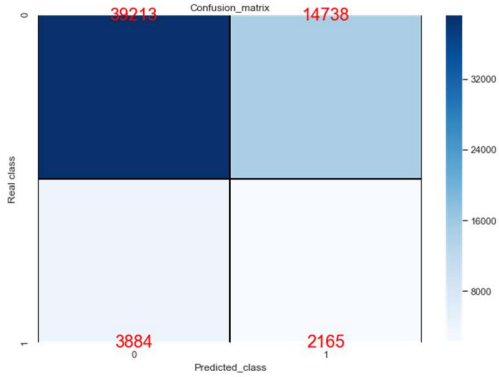
ROS

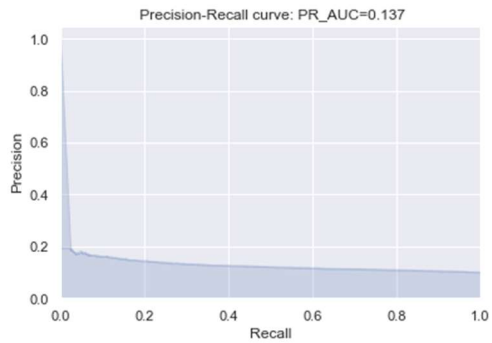


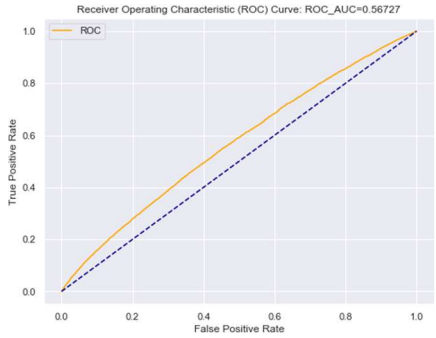




SMOTE

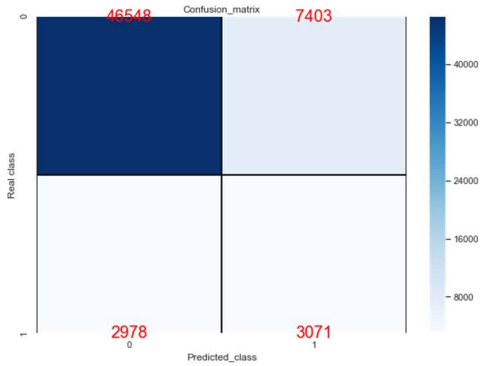


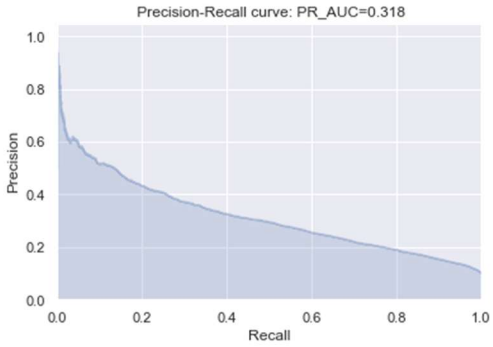


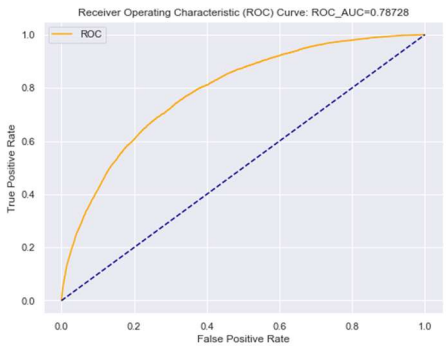


RF_MODEL

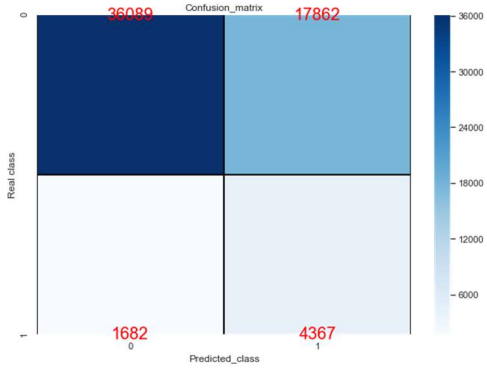
Unsampled Data

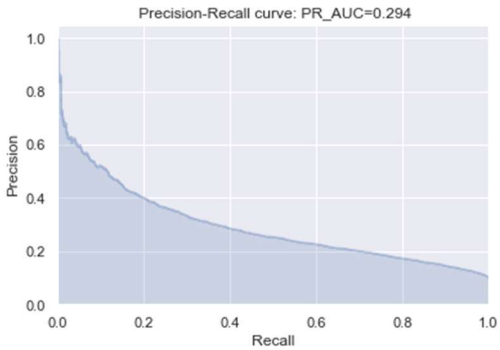


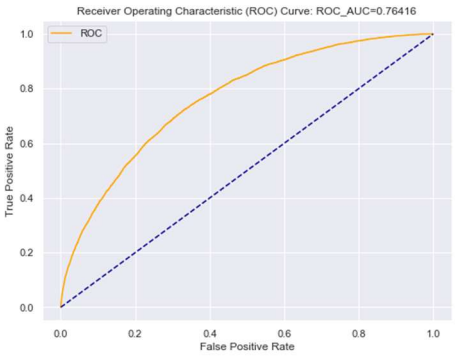




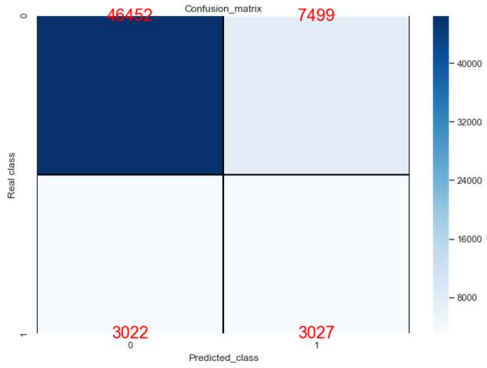
RUS

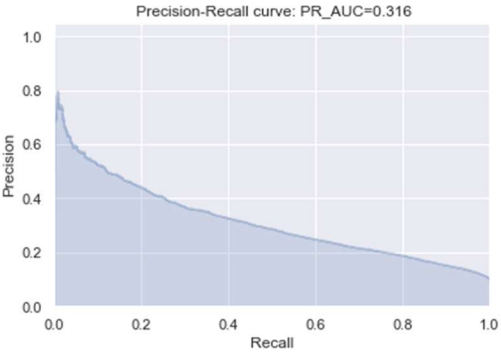


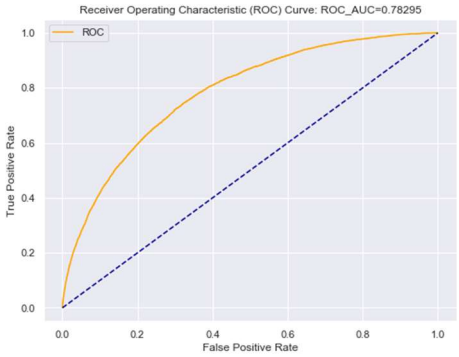




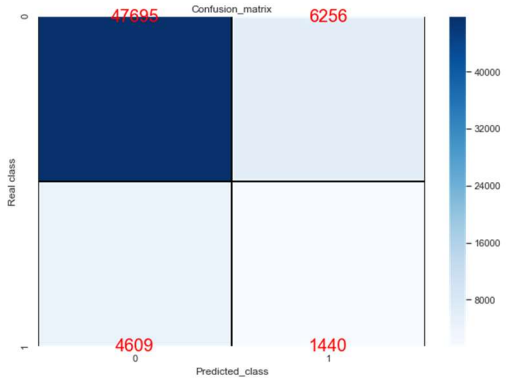
ROS

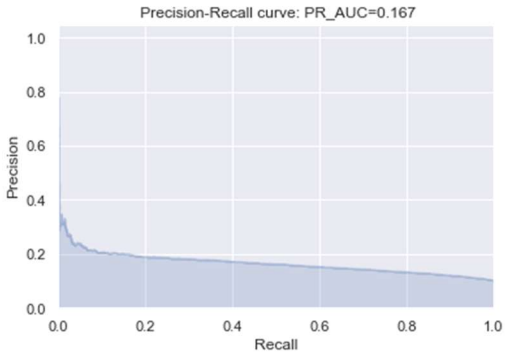


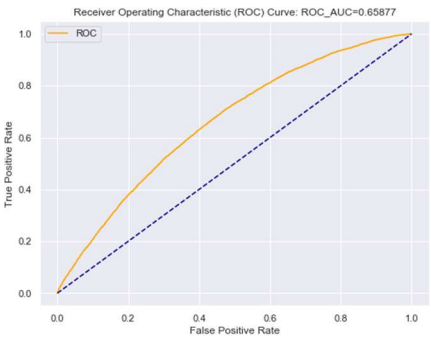




SMOTE

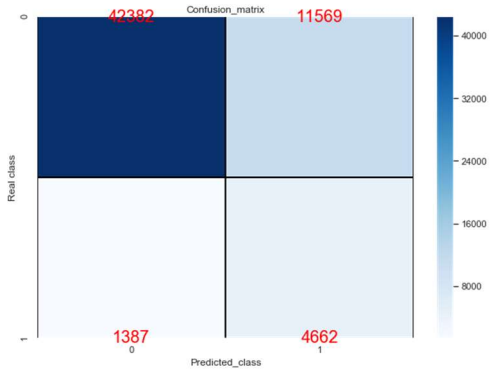


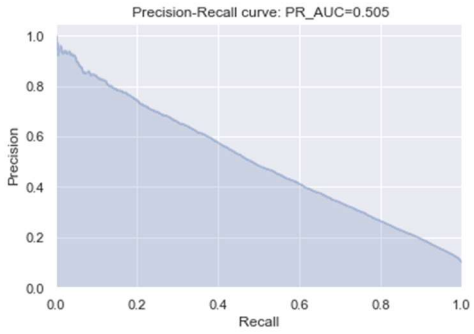


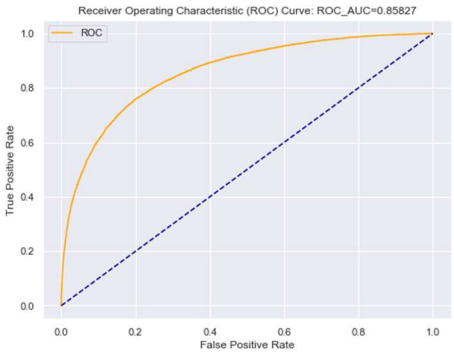


LR_model

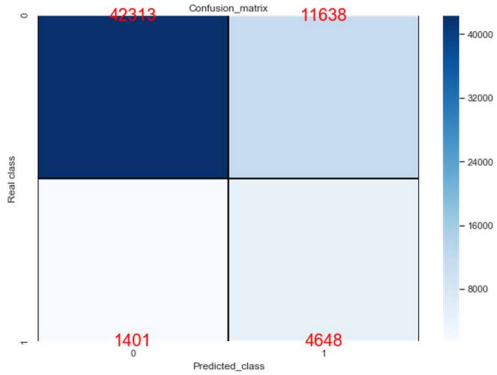
Unsampled Data

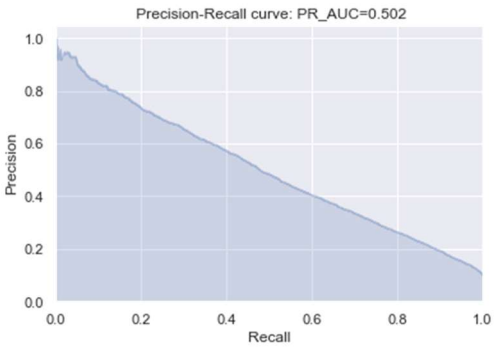


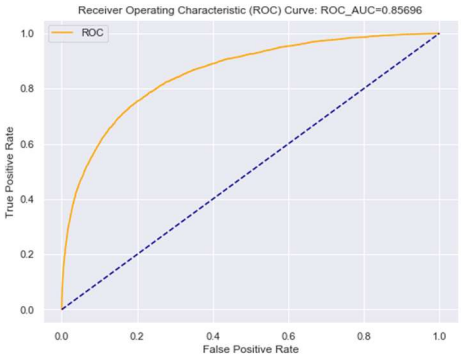




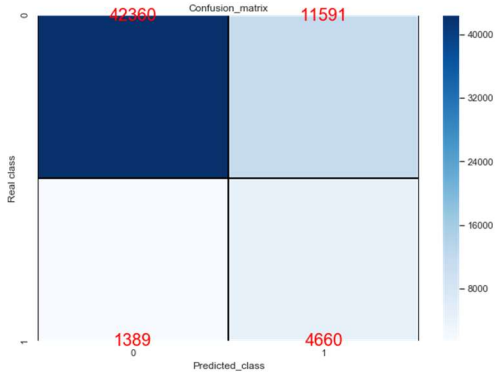
RUS

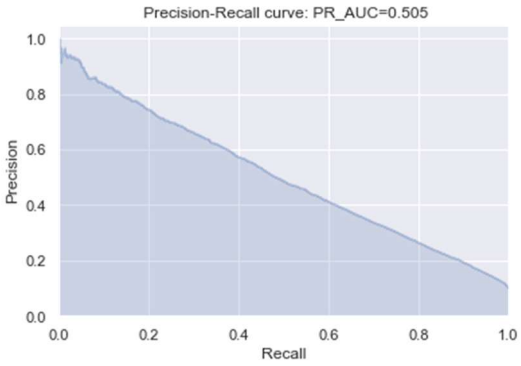


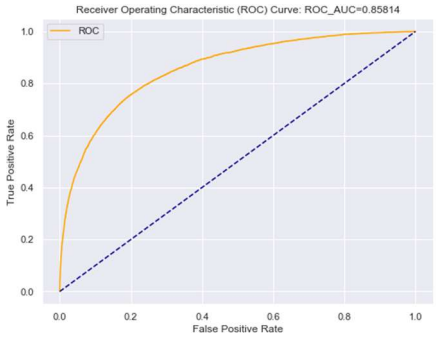




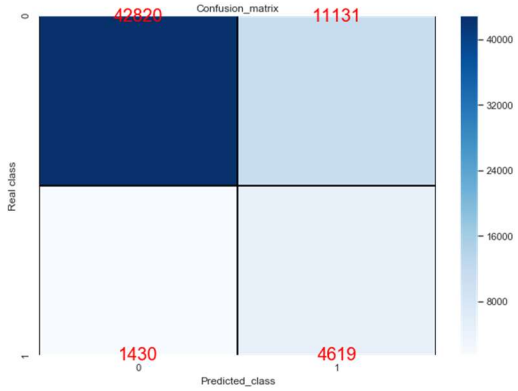
ROS

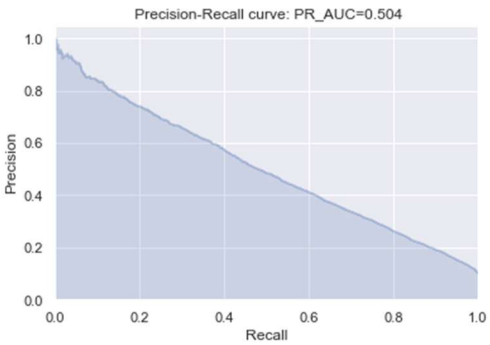


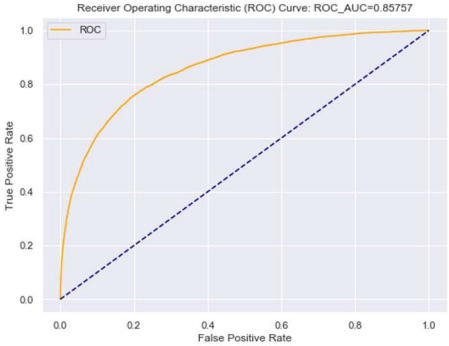




SMOTE

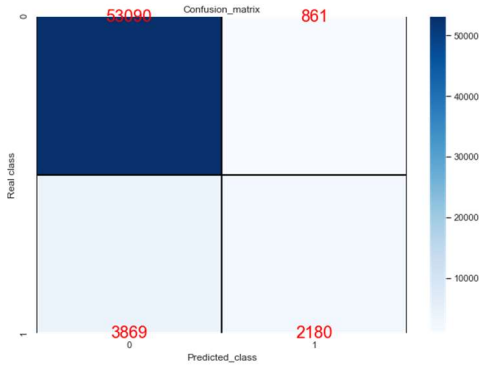


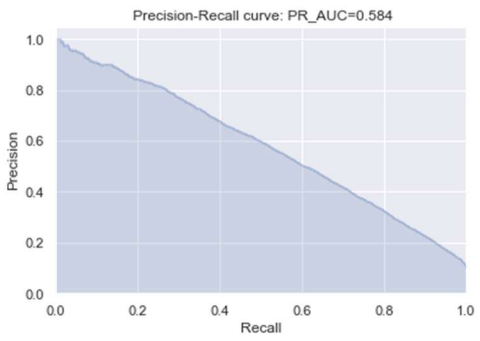


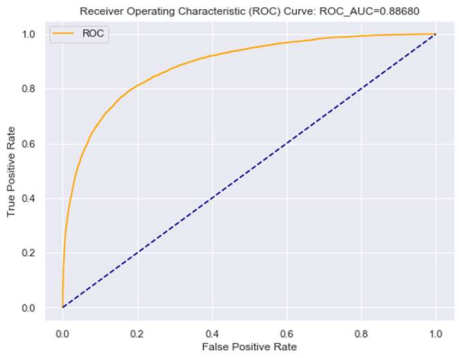


NB_model

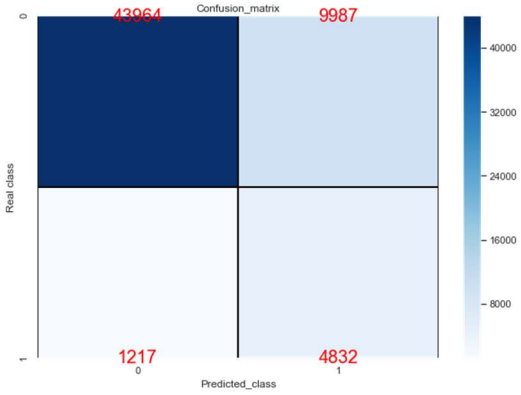
Unsampled Data

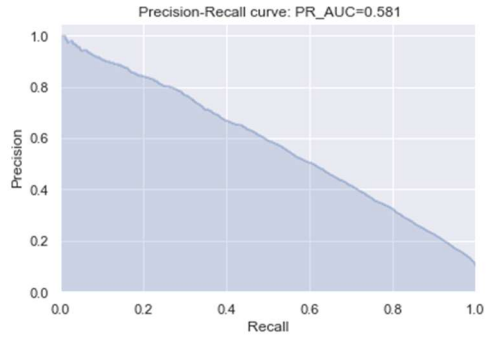


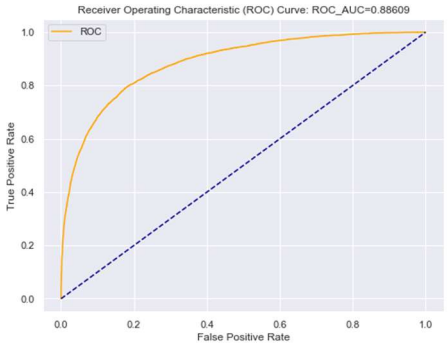




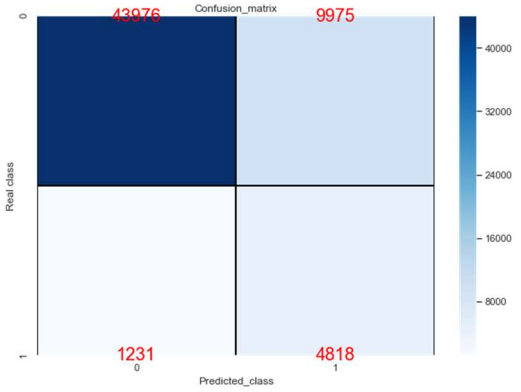
RUS

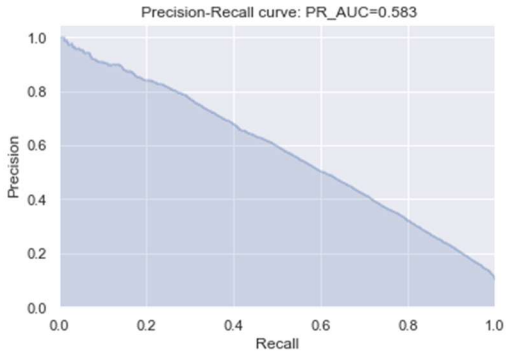


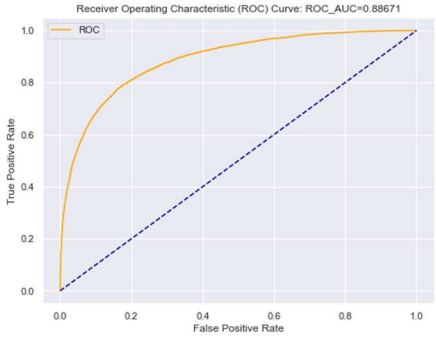




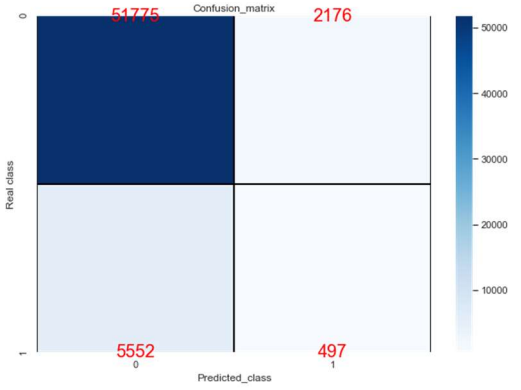
ROS

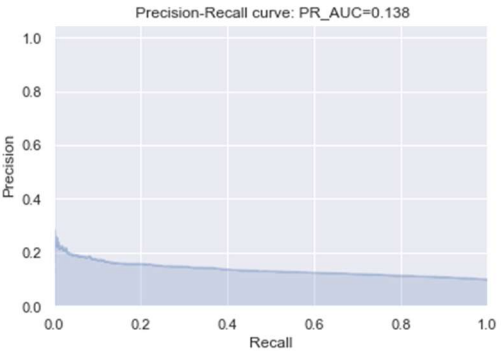


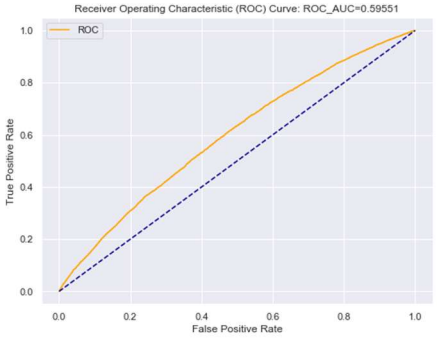




SMOTE







Here We can see That Naïve Bayes performing best among all the Model With Or Without Outliers but model with outlier is performing better that models without outliers and then the Naïve Bayes model with Random Oversampling is able to maximum true positive rate but False Positive Rate is high while in Model with unsampled Data in NB model the FPR rate is very less but TPR is very low

So Naïve Bayes with random oversampling is the choice for this dataset

PS- I the duration of project I tried to implement different technique like xgboost,adaboost,gbm,extra tree etc but the performance couldn't be increased

Although few model that showed some promise and voting technique couldn't be applied as my machine was not allowing such performance