**Santander Customer Transaction Prediction**

**Chandrakanth Bachu**

**Jan 2020**

**Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **Introduction** | | | **3** |
|  | 1.1 | Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 3 |
|  | 1.2 | Data . | . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| **2** | **Methodology** | | | **5** |
|  | 2.1 | Data Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 5 |
|  |  | 2.1.1 | Univariate Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
|  |  | 2.1.2 | Bivariate Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
|  |  | 2.1.3 |  |  |
|  | 2.2 | Preprocessing Techinues. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 10 |
|  |  | 2.2.1 | Missing value analysis | 10 |
|  |  | 2.2.2 | Outlier analysis | 11 |
|  |  | 2.2.3 | Feature Selection | 13 |
|  |  | 2.2.4 | Feature Scaling | 15 |
| **3** | **Modelling** | |  | **16** |
|  | 3.1 | Model Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
|  |  | [3.1.1 Evaluating Regression Model](#page16)  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
|  |  |  | |  |
|  | 3.2 | Decision Tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 17 |
|  | 3.3 | Random Forest | | 19 |
|  |  |  | |  |
|  |  |  | |  |
| **Appendix A – Python code** | | | | **21** |
| Reference | | | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  | | |  |  |

1. **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

1. **Data**

Our task is to build the classification model to predict which customer will make specific Transaction in the future , irrespective of amount of money transacted.

Table 1.1: Santander Customer Transaction data Description

|  |  |
| --- | --- |
| Description | counts/type |
| #No of variables In Train data | 202 |
| #No of variable in test Data | 201 |
| #No of records in Train data | 200000 |
| #No of records in Test data | 200000 |
| Target column Name | target |
| data types of 200 Independent variable | float64 |
| data type of target variable | object type |

There are 200 Independent variables along with customer id to predict the transaction of the customer.

All the 200 independent variables types are float64 type so, there are no categorical variable in independent variables and target variable is having two labels [0 , 1] this is binary classification problem.

**Chapter 2**

**Methodology**

1. **Pre Processing**

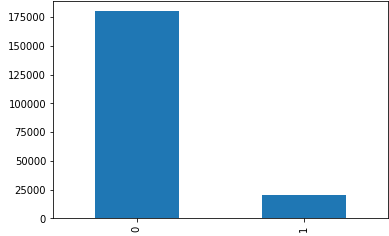
Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process we will first try and look at class imbalance of Target variable in most of the classification class imbalance will create severe problems during the modelling/

**2.1.1 Univariate Analysis**

Target Value ‘Target’ contains 89.5% of data contains customers with no transaction and 11.5 % of data contains transactions , it may be chance that **class imbalance** problem may occurs because of less proportion of data contains Customer transactions , we should be very careful on during evaluation of Model instead of concentration on only **Accuracy** we should also concentrate on **Precision and Recall** also and we should make sure that **Precision and Recall** should also be **high**.

Table

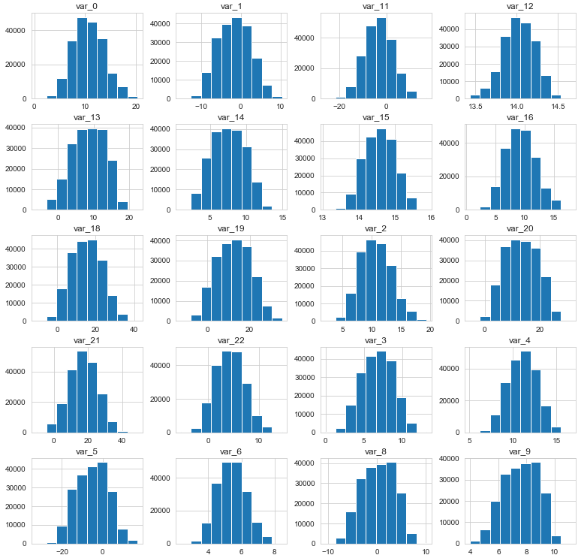
|  |  |  |
| --- | --- | --- |
| values | Count | Proportions |
| No | 17992 | 89.5 |
| Yes | 20098 | 11.5 |



**Distribution of Dependent Numeric Variables :**

In Figure 2.2 it is clearly showing almost all the dependent variables are normally distributed and it is also showing that all the variables closely distributed mean,median and standard deviation are very close is there any chance of outlier in outlier sections.

Figure 2.2 showing distribution of top 20 dependent variable based on ttest (python code in Appendix A)



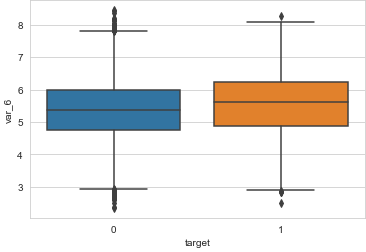
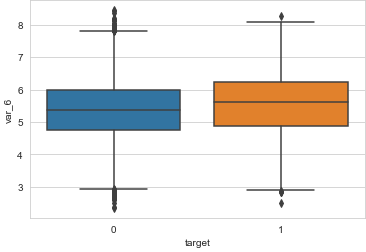
**2.1.2 Bivariate Analysis**

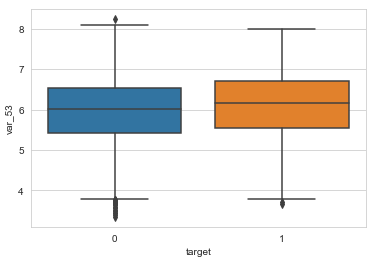
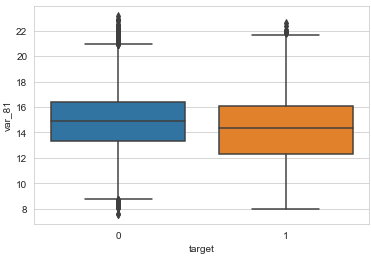
**Relationship between Target Variable “Churn” and top 20 Numeric Variables based on statistical ttest** :

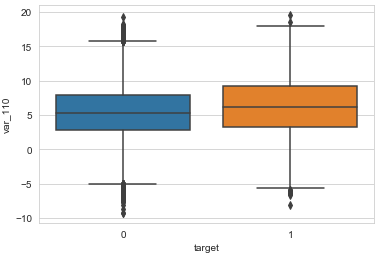
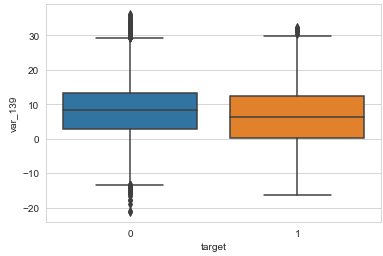
Below Figure 2.3 showing that “Total\_day\_charge” , “Total\_intl\_charge” and “Number\_customer\_service\_charge” FOr medians ,IQR and Ranges of Boxplot is different for “Unchur” and “Churn” so these features are clearly showing are important to prediction.

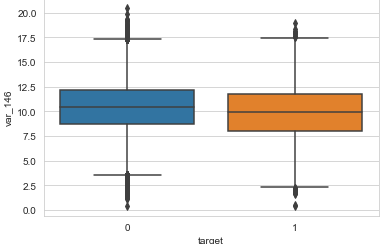
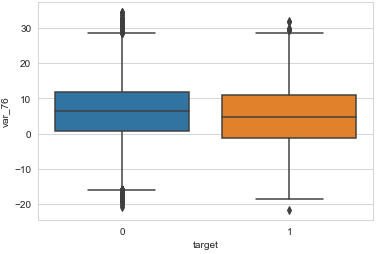
For other features Boxplot Median , IQR, Ranges are looking almost same. Here it is stating Feature Engineering is important to find the relationship between the variables.

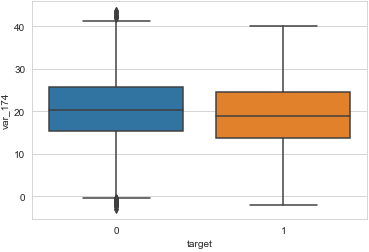
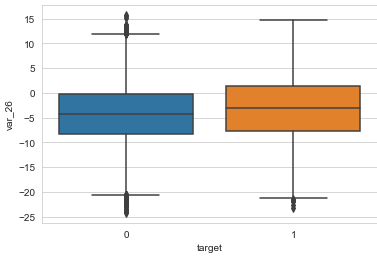
Figure 2.3 relationship between Numeric variables (python code in Appendix A)

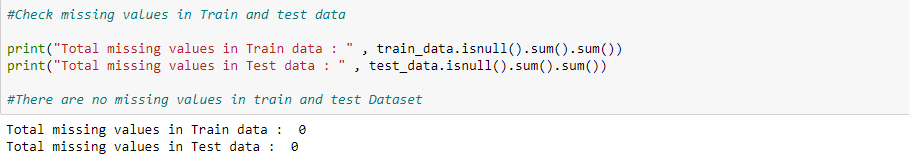
 

**2.2.1 Missing Value Analysis**

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

Below table illustrate no missing value present in the data.



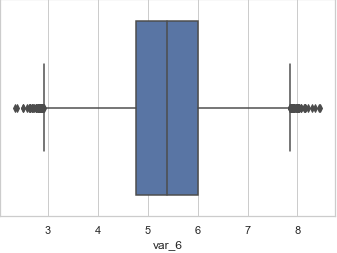
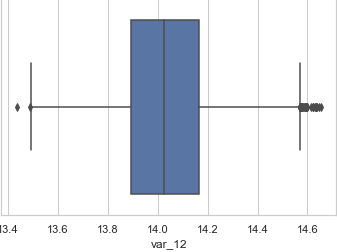
**2.2.2 Outlier Analysis**

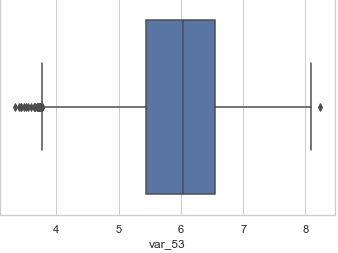
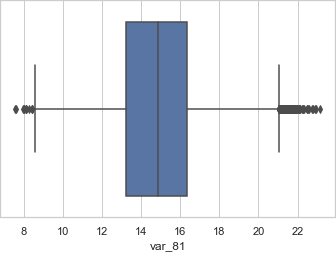
The Other steps of Preprocessing Technique is Outliers analysis , an outlier is an observation point that is distant from other observations. Outliers in data can distort predictions and affect the accuracy metrics , if you don’t detect and handle them appropriately especially in regression models..

As we are observed in fig 2.2 the data is almost all the variables are normal distribution and having less skewed so here chances are less to have outliers but still it is better to analyze.

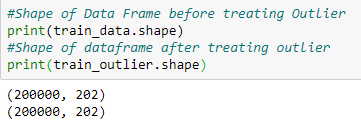
one of the best method to detect outliers is Boxplot

Fig 2.4 shows presence of Outliers in variable var\_6,var\_12,var\_53,var\_81(checking top 4 variables as per ttest)

Here Dimensions of Data frame is same after treating outliers using IQR logic , though boxplot is showing Outliers which is near to lower extreme but those data points are not crossing IQR range so ther.e is no outliers in Dataset



Boxplot :-  boxplot is a method for graphically depicting groups of numerical data through their [quartiles](https://en.wikipedia.org/wiki/Quartile). Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles

**2.2.3 Features Selections**

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.

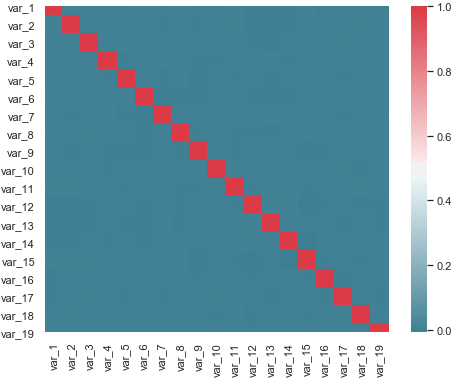
This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – “Sometimes, less is better!”.

We should consider the selection of feature for model based on below criteria

1. The relationship between two independent variable should be less and
2. The relationship between Independent and Target variables should be high.

Below fig 2.6 illustrates that relationship between all numeric variables using Corrgram plot .

Figure 2.6 correlation plot of 20 numeric variables (Python code in Appendix A)



Color dark blue indicates there is no strong relationship and if darkness is decreasing indicates relation between variables are increases.

Color dark Red indicates there is strong relationship and if darkness is decreasing indicates relationship between variables are decreasing.

Corrgram : it help us visualize the data in correlation matrices. correlograms are implimented through the **corrgram(x, order = , panel=, lower.panel=, upper.panel=, text.panel=, diag.panel=)**

**2.4.1 Dimensionality Reduction for numeric variables**

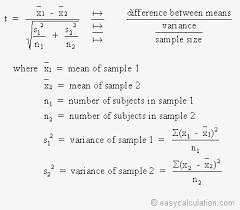
Above Fig 2.6 is showing

The above figure is showing there is no relationship between independent variables so it is good for regression that relationship between the independent variables should be less.

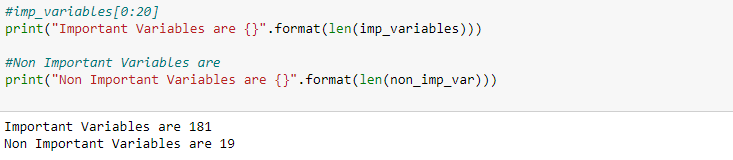
**2.4.2 Dimensional Reduction using Anova t-test.**

To fin the relationship between Target categorical variable and numeric variables we are using statistical ttest.

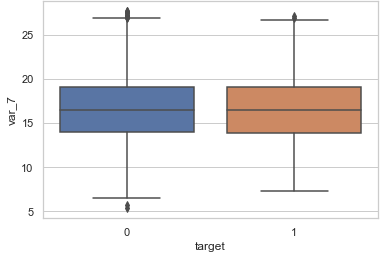
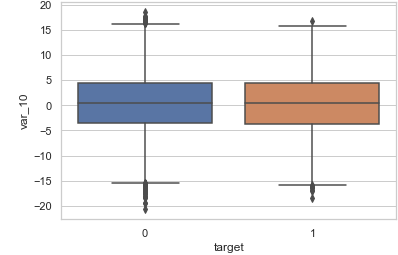
Figure 2.7 Chi- Square P Value with categorical Variable and Churn



Based on T-test important variables are 181 and there are 19 variables which are not describing any variance with target so we are removing those variables.



Below Boxplot is illustrating the non important variables distribution with target variable.



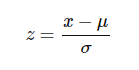
Here it is clearly showing median and IQR of the boxplot is almost same for Target variable categories 0 and 1.

**2.2.4 Features Scaling Using Standardization**

Most of the Machine Learning algorithms performance depends on data we are passing through it ,

If two variable are in different ranges than there is chance that Model will bias towards that higher range variable so it is important to Scale Numeric variables in same range.

As we observed in Univariate analysis that there are almost all the variable are normal form so, we are using Standardization(Z - Score) technique to scale the Numeric Variable.



**Chapter 3**

**Modelling**

**3.1 Model Selection**

In out earlier stage of analysis we have come to understand that few variables like ‘number\_day\_charges’ ,number\_customer\_service\_calls etc‘ are going to play key role in model development , for model development dependent variable may fall under below categories

1. Nominal
2. Ordinal
3. Interval
4. Ratio

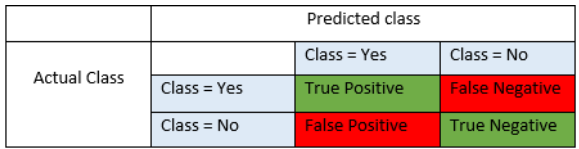
In our case dependent variable is ordinal(Categorical) so, the predictive analysis that we can perform is **Classifiction** Analysis

We will start our model building from Decision Tree .

**3.1.1 Evaluating Regression Model**

When building a model first we have to check is if the Model even works on the data it was Trained from. In this Model as it is Classification problem statement we are using Confusing Matrix to find the Accuracy of the Model. By using Confusion Matrix we are defining below measures to evaluate the Model.

**Confusion Matrix**



**Precision** : Precision is fraction of items the classifier flags as being in the class actually are in the class.

**Precision = TP/TP+FP**

**Recall** : - What fraction of things that are in the class are detected by the classifier.

**Recall : TP/TP + FN**

**Accuracy** : Below is the actual over all Accuracy of the Model

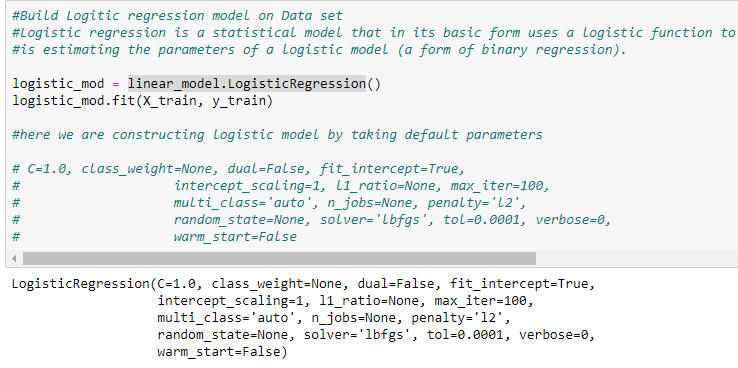
**Accuracy = (TP+TN)/(TP+FP+TN+FN)**

**F1 Score :**  It is the combination of the Precision and recall

**F1 Score : 2\*(Precision\*Recall)/(Precision+Recall)**

**3.2.Logistic Regression**

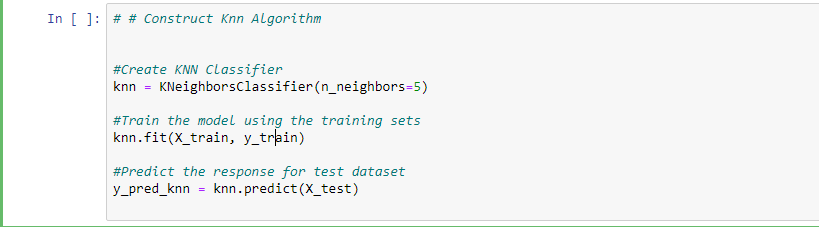
Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.



**3.2 KNN**

The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. [KNN](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm) is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. It is a lazy learning algorithm since it doesn't have a specialized training phase. Rather, it uses all of the data for training while classifying a new data point or instance. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. This is an extremely useful feature since most of the real world data doesn't really follow any theoretical assumption e.g. linear-separability, uniform distribution, etc.

Figure 3.3.1 KNN Implementation



**3.3 Random Forest**

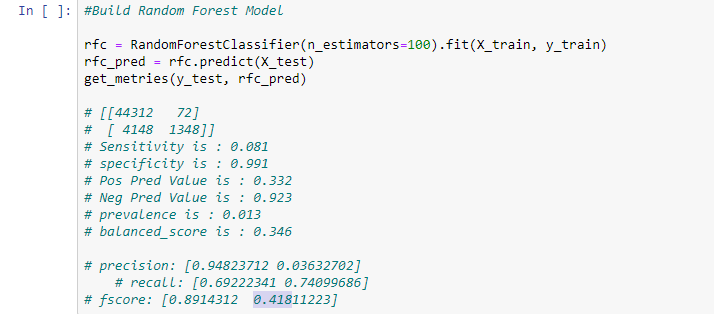
Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set).

Random forest functions in below way

1. Draws a bootstrap sample from training data.
2. For each sample grow a decision tree and at each node of the tree
3. Ramdomly draws a subset of mtry variable and p total of features that are available
4. Picks the best variable and best split from the subset of mtry variable
5. Continues until the tree is fully grown.

As we saw in section 3.2 Decision tree is quite good order to improve the Precision and recall of the model we are developing model using Random Forest.

Figure 3.3.1 Random Forest Implementation



Mtry : Number of variables to split at each node i.e. 7.

Nodesize : size of each node is 10

Our Random Forest model is looking quite good where it utilized maximum variables to predict the count values

**Below Table illustrate the accuracy of three models Logistic Regression,KNN and Random Forest**

|  |  |  |  |
| --- | --- | --- | --- |
| **Programming** : Python | | | |
| **Data** : Santander Customer Transaction | | | |
| **ML Model :** | Logistic Regression | KNN | Random Forest |
| Accuracy | 0.9182 | 0.9079 | 0.901 |
| Sensitivity | 0.265 | 0.231 | 0.081 |
| Specificity | 0.987 | 0.991 | 0.991 |
| Pos Pred Value | 0.695 | 0.579 | 0.332 |
| Neg Pred Value | 0.922 | 0.911 | 0.923 |
| Prevalence | 0.027 | 0.019 | 0.013 |
| Balanced Accuracy | 0.626 | 0.568 | 0.346 |
| Precision | 0.29 | 0.236 | 0.036 |
| Recall | 0.77 | 0.76 | 0.74 |
| F1 Score | o.42 | 0.38 | 0.418 |

**3.4 : Handling Data Imbalance**

**3.4.1. Over samplimng minority class:**

Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don’t have a ton of data to work with.

We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.

**Under sampling majority class :**

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

We will again use the resampling module from Scikit-Learn to randomly remove samples from the majority class.

**SMOTE :**

A technique similar to upsampling is to create synthetic samples. Here we will use [imblearn’s](https://imbalanced-learn.readthedocs.io/en/stable/index.html) SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

Below table illustrate accuracy of model applying different techniques:

|  |  |  |  |
| --- | --- | --- | --- |
| **Programming** : Python | | | |
| **Data** : Santander Customer Transaction | | | |
| **Data :Imbalance Technique** | Oversampling lower class | Under sampling higer class | SMOTE |
| Accuracy | 0.9182 | 0.9079 | 0.901 |
| Sensitivity | 0.265 | 0.783 | 0.771 |
| Specificity | 0.987 | 0.785 | 0.792 |
| Pos Pred Value | 0.695 | 0.293 | 0.296 |
| Neg Pred Value | 0.922 | 0.97 | 0.968 |
| Prevalence | 0.027 | 0.08 | 0.079 |
| Balanced Accuracy | 0.626 | 0.784 | 0.782 |
| Precision | 0.29 | 0.783 | 0.296 |
| Recall | 0.77 | 0.42 | 0.77 |
| F1 Score | o.42 | 0.38 | 0.42 |

Model Selection :

As per Above models accuracy Logistic regression with SMOTE is the best fit for the dataset.

**References**

[WWW.Edwisor.com](http://WWW.Edwisor.com)

WWW. Stackoverflow.com

**Appandix :A**

**Import Libraries**

In [ ]:

In [ ]:

*#Import all Libraries*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd** *#Python pakage which is fast,flexible to work on structural data*

**import** **numpy** **as** **np** *#Numeric python core library for scientific computing*

**import** **sys** *#used to manipulate different parts of the Python runtime environment.*

**import** **os** *#To interact with File system*

**import** **seaborn** **as** **sns** *#to plot interactive plots*

**import** **matplotlib.pyplot** **as** **plt** *# for plottingcs*

**import** **scipy.stats** *#statistics for data analysisfrom scipy import stats*

**from** **scipy** **import** stats *#statistics for data analysisfrom scipy import stats*

**from** **scipy.stats** **import** ttest\_ind *#WHich is used to test variance in groups of two means*

**import** **operator** *#having standard operators as function like add, sub,sort etc.*

**import** **sklearn.model\_selection** **as** **ms** *#Goint use for Train test split*

**import** **numpy.random** **as** **nr** *#generate seeds for random numbers*

**from** **sklearn** **import** linear\_model *#for logistic regression model*

**from** **sklearn.metrics** **import** confusion\_matrix *# to get confusion matrix*

**from** **sklearn.metrics** **import** balanced\_accuracy\_score *#to get balance accuracy score of the Model*

**from** **sklearn.neighbors** **import** KNeighborsClassifier *#For KNearest Neighbour Model*

*#Handling Imbalance data using synthetic dataset*

**from** **imblearn.over\_sampling** **import** SMOTE *#for creating synthetic data*

*#Try Random Forest Algorithm*

**from** **sklearn.ensemble** **import** RandomForestClassifier *#to build random forest model*

**from** **sklearn.utils** **import** resample *#for resampling data*

**from** **sklearn.linear\_model** **import** Lasso, LogisticRegression *#for feature selection*

**from** **sklearn.feature\_selection** **import** SelectFromModel *#for feature selection*

**from** **sklearn.metrics** **import** precision\_recall\_fscore\_support **as** score

**from** **sklearn.svm** **import** SVC *# "Support Vector Classifier"*

In [ ]:

*# Create all the methods which are going to use in the workbook*

**def** create\_frequncy\_tables\_plot(data,col\_categorical):

*""" This function will take the data frame and categorical columns as input and*

*will give the counts and proportions of each label in univariate categorical variables*

*"""*

*# couunts using count\_values and proportions using cross table*

cross\_tab=pd.crosstab(col\_categorical ,columns="count")

cross\_tab=pd.crosstab(col\_categorical ,columns="count\_percentage").apply(**lambda** r: r/len(data), axis=1)

**return** churn\_customers,cross\_tab

**def** change\_data\_type(data,col\_names,convert\_type):

*"""" This function will take the data frame and columns and conversion type as input*

*will give the converted columns from one datatype to another datatype*

*"""*

**for** col **in** col\_names:

print(col , "before convert" , data[col].dtype)

data[col] = data[col].astype(convert\_type)

print(col,"after convert",data[col].dtype)

**return** data

**def** box\_plot(data,variables):

*"""*

*This Function will take the Input as Data and List of variables and plot the Boxplot*

*"""*

**for** i **in** variables:

sns.set(style="whitegrid")

tips = sns.load\_dataset("tips")

ax = sns.boxplot(x=data[i])

plt.show()

**def** treat\_outlier(data,numeric\_columns):

*""" This function will take the input as data frame and numeric values and return output dataframe*

*after treating the outliers"""*

**for** i **in** numeric\_columns:

print(i)

q75, q25 = np.percentile(data.loc[:,i], [75 ,25])

iqr = q75 - q25

mini = q25 - (iqr\*1.5)

maxi = q75 + (iqr\*1.5)

print(mini)

print(maxi)

data\_outlier = data.drop(data[data.loc[:,i] < mini].index)

data\_outlier = data.drop(data[data.loc[:,i] > maxi].index)

**return** data\_outlier

**def** fun\_numeric\_relation(data):

*"""" This function will give output of plot of relationship between numeric variables in data frame """*

f, ax = plt.subplots(figsize=(8, 6))

corr = data.corr()

print(corr)

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=**True**), square=**True**, ax=ax)

**def** perform\_ttest(data,num\_cols,target\_col):

*"""*

*This function will take input as data, numeric independent variables and categorical target variable*

*and give output of ttest p-values in Dictionary*

*"""*

ttest\_dict = []

**for** var **in** num\_cols:

f,p = ttest\_ind(data[var][data[target\_col] == 0],

data[var][data[target\_col] == 1],equal\_var = **False**

)

ttest\_dict.append((var,p))

*#sorted\_new = [(name, "%.150f" % float(x)) for name,x in ttest\_dict]*

**return** ttest\_dict

**def** plot\_box(data, cols, col\_x):

*"""" This function will display the box plot, to show relationship between numeric*

*variables(Cols) and and target categorical variable (Col\_x)*

*"""*

**for** col **in** cols:

sns.set\_style("whitegrid")

sns.boxplot(col\_x, col, data=data)

plt.xlabel(col\_x) *# Set text for the x axis*

plt.ylabel(col)*# Set text for y axis*

plt.show()

**def** standardform\_convert(data ,numeric\_columns):

*""" This functin will take input as data frame and numerical columns and convert those numerical data into standardization*

*form and gives output and converted data frame"""*

**for** i **in** numeric\_columns:

*#print(i)*

data[i] = (data[i] - data[i].mean())/data[i].std()

**return** data

**def** get\_metries(y\_test1, y\_pred1):

*"""This Function will take input as y\_test and y\_pred and get the metrices like*

*i. sensitivity*

*ii. specificity*

*iii. pos\_pred\_val*

*iv. neg\_pred\_val*

*v. prevalence*

*vi. balanced\_score*

*"""*

CM\_logistic = confusion\_matrix(y\_test1, y\_pred1)

print(CM\_logistic)

TN = CM\_logistic[0][0]

FN = CM\_logistic[1][0]

TP = CM\_logistic[1][1]

FP = CM\_logistic[0][1]

*# tn, fp, fn, tp = confusion\_matrix(Y\_test, Y\_pred).ravel()*

*# sensitivity,specificity = tp/(tp+fn),tn/(tn+fp)*

sensitivity = TP / (TP+FN)

specificity = TN / (TN+FP)

pos\_pred\_val = TP/ (TP+FP)

neg\_pred\_val = TN/ (TN+FN)

prevalence = TP /(TP+FP+TN+FN )

balanced\_score = balanced\_accuracy\_score(y\_test1, y\_pred1)

print("Sensitivity is : **{}**".format(round(sensitivity,3)))

print("specificity is : **{}**".format(round(specificity,3)))

print("Pos Pred Value is : **{}** ".format(round(pos\_pred\_val,3)))

print("Neg Pred Value is : **{}** ".format(round(neg\_pred\_val,3)))

print("prevalence is : **{}** ".format(round(prevalence,3)))

print("balanced\_score is : **{}** ".format(round(balanced\_score,3)))

precision, recall, fscore, support = score(y\_test, y\_pred)

print('precision: **{}**'.format(precision))

print('recall: **{}**'.format(recall))

print('fscore: **{}**'.format(fscore))

print('support: **{}**'.format(support))

**def** plot\_auc(labels, probs):

*## Compute the false positive rate, true positive rate*

*## and threshold along with the AUC*

fpr, tpr, threshold = sklm.roc\_curve(labels, probs[:,1])

auc = sklm.auc(fpr, tpr)

*## Plot the result*

plt.title('Receiver Operating Characteristic')

plt.plot(fpr, tpr, color = 'orange', label = 'AUC = **%0.2f**' % auc)

plt.legend(loc = 'lower right')

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

**Loading Train and Test Data**

In [ ]:

*# Set the Working Directory*

os.getcwd()

*#change working directory*

os.chdir("D:/data science prsctise")

os.getcwd()

In [ ]:

*# Load Train and test data*

train\_data = pd.read\_csv("train.csv")

test\_data = pd.read\_csv("test.csv")

In [ ]:

*#Anallye the train and test data*

train\_data.head()

print(train\_data.shape)

*#Train datset contains*

*#202 columns and 200000 records*

test\_data.head()

print(test\_data.shape)

*#Test data contains*

*#201 columns and 200000 records*

In [ ]:

*#check the datatype of Trains and test variables*

*#print(train\_data.info())*

print(train\_data.info())

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

*#So 200 variables are float type and 1 variable is int type and 1 variable in object type*

*#check the dtypes of test data*

print(test\_data.info())

*#dtypes: float64(200), object(1)*

*#200 variables are float type and 1 variable is object type*

*#check the column which is having dtype= object*

print(list(train\_data.select\_dtypes(include=['object'])))

print(list(test\_data.select\_dtypes(include=['object'])))

*#ID\_code is object type which is making sense*

*# all the numeric variables are in float type and target variable is having int type(actually it should be categorical)*

**Univariate Analysis**

In [ ]:

*# Analyse the distribution of target variable*

*#Analys the distribution of target value in train data*

train\_data['target'].value\_counts().plot(kind='bar')

*#check the percentage of each value 0 & 1 in target variable*

print(train\_data['target'].value\_counts())

print(train\_data['target'].value\_counts(normalize = **True**) \*100)

*#Transactions data is 10.049 and non transaction data is 89.951*

*#0 89.951*

*#1 10.049*

*#Might be chance of facing data imbalance have to be careful during modelling stage whike measuring the accuracy*

In [ ]:

*#Below is the Distrivution of top 20 Correlated variables with Target variable based on ttest*

*# Now check the distribution of Numeric variables*

pd.set\_option("display.max\_columns", **None**)

train\_data.describe()

*#Below are the quick observation*

*# data is closely distributed min max and standard distribution is looking quick close*

*#might be less chance of getting outliers*

**Missing Values Analysis**

In [ ]:

*#Check missing values in Train and test data*

print("Total missing values in Train data : " , train\_data.isnull().sum().sum())

print("Total missing values in Test data : " , test\_data.isnull().sum().sum())

*#There are no missing values in train and test Dataset*

**Outlier Anlaysis**

In [ ]:

*#Call the Boxplot function and pass first 10 varables to the function*

box\_plot(train\_data,train\_data.columns[2:10])

*#This below plots clearly showing that there are few outliers in the variables*

*# but which are very near to the lower/Upper extreme boundries*

*#This is also one the sign data is closely distributed*

In [ ]:

*#Lets check dataset after removing the outliers*

*#As we have all independlent variables are Numeric type so it is better to check outliers of the variables*

*#Analyse if outliers are present in the variables*

train\_outlier = treat\_outlier(train\_data,train\_data.columns[2:])

In [ ]:

*#Shape of Data Frame before treating Outlier*

print(train\_data.shape)

*#Shape of dataframe after treating outlier*

print(train\_outlier.shape)

*# Asp per the IQR Range there is no data points has been removed from the data*

*#No need to remove any data eventhough boxplots is showingfew outliers*

**Feature Engineer**

In [ ]:

*# Analyze the relationship between Independent variables using correlation plot*

fun\_numeric\_relation(train\_data[train\_data.columns[2:][1:20]])

*#This plot is hsowing almost blue that is there is no relation between the independent variables*

*#So its is clearly satisfying the conditions of regression that there should not be any relation between*

*#Independent Variables*

In [ ]:

*#Analyse the Relation between numeric & Independent variables*

*#The independent t-test is used to compare the means of a condition between 2 groups*

ttest\_dict = perform\_ttest(train\_data,train\_data.columns[2:],'target')

*#order the values in dictionary based on p-values*

sorted\_new = [(name, "**%.150f**" % float(x)) **for** name,x **in** ttest\_dict]

sorted\_new.sort(key=operator.itemgetter(1))

In [ ]:

*# As per hypothesis test if p-value <= 0.05 than rejecting null hypoythesis and accepting alternative hypothesis*

*# so there is relation between the variables are very high*

*#if p-value > 0.05 than accepting Null hypothesis which says the relationship between target variable and*

*#Independent variables is less*

imp\_variables = [x[0] **for** x **in** ttest\_dict **if** float(x[1]) < 0.05]

*#list(filter( lambda p: p[] > 0.05, sorted\_new))*

non\_imp\_var = [x[0] **for** x **in** ttest\_dict **if** float(x[1]) > 0.05]

In [ ]:

*#imp\_variables[0:20]*

print("Important Variables are **{}**".format(len(imp\_variables)))

*#Non Important Variables are*

print("Non Important Variables are **{}**".format(len(non\_imp\_var)))

*# Important Variables are 181*

*# Non Important Variables are 19*

In [ ]:

*#Just for visualizing the mean of groups plot the boxplot*

plot\_box(train\_data,non\_imp\_var[0:5],'target')

*# per ther below boxplot there is less diffrence betwen the mean of two groups*

*#so, t-test result we can remove this variables from the dataset*

In [ ]:

*#As per t-test analyse the relation between top 5 low p-value variables with target variable*

*#using boxplot*

[top\_imp\_variables] = [[t[0] **for** t **in** sorted\_new]]

plot\_box(train\_data,top\_imp\_variables[0:5],'target')

*#Here in boxplot there is difference between the median is showing clearly but its less*

*#because data is closeely distributed*

**Feature Scaling**

In [ ]:

*#standardizing the independent variables is a simple method to reduce multicollinearity that is produced by higher-order terms*

train\_data = standardform\_convert(train\_data,imp\_variables)

test\_data = standardform\_convert(test\_data,imp\_variables)

print(train\_data.shape)

print(test\_data.shape)

In [ ]:

*#Split the dataset into train and test*

*#Since we have have procedure to evaluate the model*

*#so splitting train data to train and test data contain train data 75% and test data 25%*

X\_train, X\_test, y\_train, y\_test = ms.train\_test\_split(train\_data[imp\_variables], train\_data['target'], test\_size=0.25,random\_state = 123)

print(X\_train.shape, y\_train.shape)

print(X\_test.shape, y\_test.shape)

**Modelling**

In [ ]:

*#Build Logitic regression model on Data set*

*#Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression)*

*#is estimating the parameters of a logistic model (a form of binary regression).*

logistic\_mod = linear\_model.LogisticRegression()

logistic\_mod.fit(X\_train, y\_train)

*#here we are constructing logistic model by taking default parameters*

*# C=1.0, class\_weight=None, dual=False, fit\_intercept=True,*

*# intercept\_scaling=1, l1\_ratio=None, max\_iter=100,*

*# multi\_class='auto', n\_jobs=None, penalty='l2',*

*# random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,*

*# warm\_start=False*

In [ ]:

*#Predict the model on test data*

y\_pred = logistic\_mod.predict(X\_test)

print(y\_pred)

In [ ]:

In [ ]:

get\_metries(y\_test, y\_pred)

*# [[44312 592]*

*# [ 3748 1348]]*

*# Sensitivity is : 0.265*

*# specificity is : 0.987*

*# Pos Pred Value is : 0.695*

*# Neg Pred Value is : 0.922*

*# prevalence is : 0.027*

*# # balanced\_score is : 0.626*

*# precision: [0.96823712 0.29632702]*

*# recall: [0.79222341 0.77099686]*

*# fscore: [0.8714312 0.42811223]*

*#If we see the above metrices Precision and recall is very less*

*# Might be because of Unbalance data*

*#we have to chec the metrices for other algorithm*

In [ ]:

*# # Construct Knn Algorithm*

*#Create KNN Classifier*

knn = KNeighborsClassifier(n\_neighbors=5)

*#Train the model using the training sets*

knn.fit(X\_train, y\_train)

*#Predict the response for test dataset*

y\_pred\_knn = knn.predict(X\_test)

In [ ]:

get\_metries(y\_test, y\_pred\_knn)

*# [[44312 472]*

*# [ 3748 1348]]*

*# Sensitivity is : 0.231*

*# specificity is : 0.991*

*# Pos Pred Value is : 0.579*

*# Neg Pred Value is : 0.911*

*# prevalence is : 0.019*

*# balanced\_score is : 0.568*

*# precision: [0.97823712 0.23632702]*

*# recall: [0.75222341 0.76099686]*

*# fscore: [0.9114312 0.38811223]*

In [ ]:

*#Build Random Forest Model*

rfc = RandomForestClassifier(n\_estimators=100).fit(X\_train, y\_train)

rfc\_pred = rfc.predict(X\_test)

get\_metries(y\_test, rfc\_pred)

*# [[44312 72]*

*# [ 4148 1348]]*

*# Sensitivity is : 0.081*

*# specificity is : 0.991*

*# Pos Pred Value is : 0.332*

*# Neg Pred Value is : 0.923*

*# prevalence is : 0.013*

*# balanced\_score is : 0.346*

*# precision: [0.94823712 0.03632702]*

*# recall: [0.69222341 0.74099686]*

*# fscore: [0.8914312 0.41811223]*

**Data Imbalance**

In [ ]:

*# #Resampling Techniques — Oversample minority class*

*# Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don’t have a ton of data to work with.*

*# We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.*

**from** **sklearn.utils** **import** resample

*# concatenate our training data back together*

X = pd.concat([X\_train, y\_train], axis=1)

*# separate minority and majority classes*

not\_transact = train\_data[train\_data.target==0]

transact = train\_data[train\_data.target==1]

transact\_upsampled = resample(transact,

replace=**True**, *# sample with replacement*

n\_samples=len(not\_transact), *# match number in majority class*

random\_state=27) *# reproducible results*

upsampled = pd.concat([not\_transact, transact\_upsampled])

*# check new class counts*

upsampled.target.value\_counts()

In [ ]:

*#Now Build Logistic regression on data*

*# trying logistic regression again with the balanced dataset*

y\_train\_us = upsampled.target

X\_train\_us = upsampled.drop('target', axis=1)

upsampled = linear\_model.LogisticRegression(solver='liblinear').fit(X\_train\_us[imp\_variables], y\_train\_us)

In [ ]:

*# Undersample majority class*

*#Undersampling can be defined as removing some observations of the majority class.*

*#Undersampling can be a good choice when you have a ton of data -think millions of rows*

not\_transact\_down\_sample = resample(not\_transact,

replace = **False**, *# sample without replacement*

n\_samples = len(transact), *# match minority n*

random\_state = 27) *# reproducible results*

*# combine minority and downsampled majority*

downsampled = pd.concat([not\_transact\_down\_sample, transact])

*# checking counts*

downsampled.target.value\_counts()

In [ ]:

y\_train\_ds = downsampled.target

X\_train\_ds = downsampled.drop('target', axis=1)

undersampled = linear\_model.LogisticRegression(solver='liblinear').fit(X\_train\_ds[imp\_variables], y\_train\_ds)

undersampled\_pred = undersampled.predict(X\_test)

In [ ]:

get\_metries(y\_test,undersampled\_pred)

*# [[35264 9640]*

*# [ 1104 3992]]*

*# Sensitivity is : 0.783*

*# specificity is : 0.785*

*# Pos Pred Value is : 0.293*

*# Neg Pred Value is : 0.97*

*# prevalence is : 0.08*

*# balanced\_score is : 0.784*

*# precision: [0.96964364 0.29284038]*

*# recall: [0.78531979 0.7833595 ]*

*# fscore: [0.86780195 0.42631354]*

*# support: [44904 5096]*

In [ ]:

*# # SMOTE*

*# A technique similar to upsampling is to create synthetic samples.*

*#Here we will use imblearn’s SMOTE or Synthetic Minority Oversampling Technique.*

*#SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.*

sm = SMOTE(random\_state=27)

X\_train, y\_train = sm.fit\_sample(X\_train, y\_train)

In [ ]:

smote\_logistic = linear\_model.LogisticRegression(solver='liblinear').fit(X\_train[imp\_variables], y\_train)

smote\_pred = smote\_logistic.predict(X\_test)

In [ ]:

get\_metries(y\_test,smote\_pred)

*# [[35574 9330]*

*# [ 1167 3929]]*

*# Sensitivity is : 0.771*

*# specificity is : 0.792*

*# Pos Pred Value is : 0.296*

*# Neg Pred Value is : 0.968*

*# prevalence is : 0.079*

*# balanced\_score is : 0.782*

*# precision: [0.96823712 0.29632702]*

*# recall: [0.79222341 0.77099686]*

*# fscore: [0.8714312 0.42811223]*

*# support: [44904 5096]*

In [ ]:

In [ ]: