Project Name - Santander Customer Transaction Prediction

By: Deepanshu Vashisth

Contents

1. Introduction
   - 1.1 Problem Statement
   - 1.2 Data
2. Methodology
   - 2.1 Pre Processing
     - 2.1.1 Missing Value Analysis
     - 2.1.2 Feature Selection
     - 2.1.3 Outlier Analysis
   - 2.2 Modelling
     - 2.2.1 Model Selection
     - 2.2.2 Logistic Regression
     - 2.2.3 Random Forest
     - 2.2.4 Decision Tree
     - 2.2.5 K-Nearest Neighbour Classification
3. Conclusion
   - 3.1 Model Evaluation
   - 3.2 Model Selection

Appendix A – Python Code

Appendix B – R Code

Chapter 1

Introduction

1.1 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.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

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

1.2 Data

You are provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

In Data we have 202 variables: one is target variable(categorical), one ID_code variable(String) and 200 numerical variables. And we have around 200000 rows.

```
          var_0          var_1          ...    var_198          var_199
count  200000.000000  200000.000000  ...   200000.000000   200000.000000
mean      10.679914      -1.627622      ...     15.870720       -3.326537
std       3.040051       4.050044       ...     3.010945        10.438015
min       0.408400     -15.043400       ...     6.299300       -38.852800
25%       8.453850      -4.740025       ...     13.829700      -11.208475
50%       10.524750     -1.608050       ...     15.934050      -2.819550
75%       12.758200      1.358625       ...     18.064725       4.836800
max       20.315000      10.376800      ...     26.079100       28.500700
[8 rows x 202 columns]
```

Chapter 2

Methodology

2.1 Pre Processing

Any predictive modelling requires that we look at the data before we start modelling. 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.

2.1.1 Missing Value Analysis

One of the most common problems we face in Data Cleaning/Exploratory Analysis is handling the missing values. There are 2 ways to deal with missing data: one is my removing the rows and second is to impute the data with mean, median or knn imputation.

Here in our data there is no missing values.

2.1.2 Feature Selection

Feature selection is the process in which we select only informative feature and neglect less informative features. We do feature selection due to following reasons:

1. Curse of dimensionality : overfitting
2. Occam's Razor: We want our models to be simple and explainable.
3. Garbage in garbage out: poor quality input gives poor quality output

Here in our data we have 202 variables, so I have used Recursive Feature Elimination.

Recursive Feature Elimination: This is a wrapper based method, wrapper methods consider the selection of a set of features as a search problem.

From sklearn Documentation:

The goal of recursive feature elimination (RFE) is to select features by ***recursively considering smaller and smaller sets of features.*** *First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a* coef_ *attribute or through a* feature_importances_ *attribute. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.*

I have selected 50 features out of 202:

```
from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression

rfe_selector = RFE(estimator=LogisticRegression(), n_features_to_select=50, step=10, verbose=5)

rfe_selector.fit(data.values[:,1:202], data['target'])

rfe_support = rfe_selector.get_support()

rfe_feature = x.loc[:,rfe_support].columns.tolist()

print(str(len(rfe_feature)), 'selected features')
```

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.

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. Interquartile Range and Quartiles can also be used to detect outliers.

But Since we have too many feature still 50, so I have used z-score for the removal of outliers.

```
#Outlier Analysis
from scipy import stats
def drop_numerical_outliers(df, z_thresh=3):
  # Constrains will contain `True` or `False` depending on if it is a value below the threshold.
        constrains = df.select_dtypes(include=[np.number]) .apply(lambda x: \
               np.abs(stats.zscore(x)) < z_thresh).all(axis=1)
  # Drop (inplace) values set to be rejected
  df.drop(df.index[~constrains], inplace=True)
```

2.2 Modelling

2.2.1 Model Selection

Since our dependent variable is categorical thus only Classification Analysis can be done on the dataset. The models we'll be using are:

- Logistic Regression
- Random Forest
- Decision Tree
- K-Nearest Neighbour

2.2.2 Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Logistic Regression Assumptions

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

2.2.3 Random Forest

The random forest is a Machine learning algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

2.2.4 Decision Tree

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

2.2.5 K-Nearest Neighbour

K-Nearest Neighbors (KNN) is one of the simplest algorithms used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors. The data is assigned to the class which has the nearest neighbors. As you increase the number of nearest neighbors, the value of k, accuracy might increase.

Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance 2. Interpretability 3. Computational Efficiency

We will use Predictive performance as the criteria to compare and evaluate models. Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

3.1.1 Accuracy

Accuracy shows how accurately the test cases are predicted when compared to the original data.

```
accuracy_score(y_test, lg_predicts)

#Accuracy = 0.898

accuracy_score(y_test, rf_predict)

#Accuracy = 0.899

accuracy_score(y_test, dt_predicts)

#Accuracy = 0.899

accuracy_score(y_test, knn_predicts)
```

3.1.2 Precision

Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

#Logistic Regression

Precision = 0.999

#Decision Tree

Precision = 1.00

#Random Forest

Precision = 0.999

#KNN

Precision = 0.985

3.1.3 Recall

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

```
#Logistic Regression

Recall = 0.898

#Decision Tree

Recall = 0.898

#Random Forest

Recall = 0.898

#KNN

Recall = 0.899
```

3.2 Model Selection

Since the accuracy of all model is approx. same so on the basis of Precision and Recall I will finalize the Decision Tree model for this data.

And test cases have been predicted and saved in system.

Appendix A – Python Code

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns


data = pd.read_csv('train.csv')

test = pd.read_csv('test.csv')


data.isnull().sum()

data.describe()

del data['ID_code']


data['target'] = data['target'].astype('category')


#Feature Selection

x = data.values[:,1:202]

x = pd.DataFrame(x)

from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression

rfe_selector = RFE(estimator=LogisticRegression(), n_features_to_select=50,
step=10, verbose=5)
```

```
rfe_selector.fit(data.values[:,1:202], data['target'])

rfe_support = rfe_selector.get_support()

rfe_feature = x.loc[:,rfe_support].columns.tolist()

print(str(len(rfe_feature)), 'selected features')


df = data.iloc[: , rfe_feature]

test = test.iloc[: , rfe_feature]


#Outlier Analysis

from scipy import stats

def drop_numerical_outliers(df, z_thresh=3):

   # Constrains will contain `True` or `False` depending on if it is a value below
the threshold.

   constrains = df.select_dtypes(include=[np.number]) \

      .apply(lambda x: np.abs(stats.zscore(x)) < z_thresh) \

      .all(axis=1)

   # Drop (inplace) values set to be rejected

   df.drop(df.index[~constrains], inplace=True)


drop_numerical_outliers(df)



#Splitting in train and test

x = df.values[:,1:50]
```

```
y = df.values[:,0]

x = pd.DataFrame(x)

y = pd.DataFrame(y)

y = y.astype(int)


from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)




#Decision Tree

from sklearn import tree

dt_model = tree.DecisionTreeClassifier(max_depth=1).fit(x_train, y_train)

dt_predicts = dt_model.predict(x_test)

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score


confusion_matrix(y_test, dt_predicts)

accuracy_score(y_test, dt_predicts)

#Accuracy = 0.899

#Precision = 1.000

#Recall = 0.898




#Random Forest
```

```
from sklearn.ensemble import RandomForestClassifier

RF_model = RandomForestClassifier(n_estimators = 100).fit(x_train, y_train)

rf_predict = RF_model.predict(x_test)

confusion_matrix(y_test, rf_predict)

accuracy_score(y_test, rf_predict)

#Accuracy = 0.899

#Precision = 0.999

#Recall = 0.898



#KNN Algorithm

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors = 3).fit(x_train, y_train)

knn_predicts = knn_model.predict(x_test)

confusion_matrix(y_test, knn_predicts)

accuracy_score(y_test, knn_predicts)

#Accuracy = 0.887

#Precision = 0.985

#Recall = 0.899



#Logistic Regression

xx = x_train

import statsmodels.api as sm
```

```
logit_model=sm.Logit(y_train,xx).fit()

logit_model.summary2()

xx = xx.drop(axis = 1, columns = [4, 8, 10, 22, 33, 40])


from sklearn.linear_model import LogisticRegression

lg_model = LogisticRegression().fit(xx, y_train)

lg_predicts = lg_model.predict(x_test.drop(axis = 1, columns = [4, 8, 10, 22, 33,
40]))


confusion_matrix(y_test, lg_predicts)

accuracy_score(y_test, lg_predicts)

#Accuracy = 0.898

#Precision = 0.999

#Recall = 0.898



#Predicting the Test data

target = dt_model.predict(test.iloc[:,1:50])


test['target'] = target


test.to_csv('target.csv')
```

Appendix B - R Code

```
data = read.csv('train.csv')

target = read.csv('test.csv')

summary(data$ID_code)

str(data$ID_code)

colnames(data)


data$target = as.factor(data$target)

data = subset(data, select = -c(ID_code))


#Missing Value Analysis

sum(is.na(data))


#Feature Selection

library(caret)

set.seed(100)

options(warn=-1)


ctrl <- rfeControl(functions = lmFuncs,

                method = "repeatedcv",

                repeats = 5,

                verbose = FALSE)
```

```
lmProfile <- rfe(x=data[,-1], y=data$target, rfeControl = ctrl)

lmProfile

df = data[,c("target","var_68", "var_12", "var_148", "var_108", "var_91")]


#Outlier Analysis

for(i in colnames(df[,-1])){

  print(i)

  val = df[,i][df[,i]%in%boxplot.stats(df[,i])$out]

  df = df[which(!df[,i]%in%val),]

}


#Splitting into test and train

library('caret')

set.seed(1234)

train.index = createDataPartition(df$target,p=0.80,list=FALSE)

train = df[train.index,]

test = df[-train.index,]



#Decision Tree

library(rpart)

library('e1071')

dt_model = rpart(target~., data = train, method = 'class')
```

```
dt_predict = predict(dt_model, test[,-1], type = 'class')

confusionMatrix(dt_predict, test[,1])

#Accuracy = 0.8995

#precision = 0.899

#Recall = 1.00



#Random Forest

library(randomForest)

rf_model = randomForest(target~., train, importance = TRUE, ntree = 100)

rf_predict = predict(rf_model, test[,-1])

confusionMatrix(rf_predict, test[,1])

#Accuracy = 0.8989

#precision = 0.899

#Recall = 0.998




#Knn Model

library(class)

knn_model = knn(train[,-1], test[,-1], cl= train$target, k=7)

confusionMatrix(knn_model, test[,1])

#Accuracy = 0.895

#precision = 0.900
```

```
#Recall = 0.994



#Logistic Regression
lr_model = glm(target~., data = train, family = 'binomial')
lr_predict = predict(lr_model, test[,-1])
lr_predict = ifelse(lr_predict>0.5,1,0)
lr_predict = as.factor(lr_predict)
confusionMatrix(lr_predict, test[,1])
#Accuracy = 0.8995
#precision = 0.899
#Recall = 1



target = target[,c("var_68", "var_12", "var_148", "var_108", "var_91")]
target$target = predict(rf_model,target)

write.csv(target, 'target.csv')
```