Predicting Churn for Bank Customers

Karthik Murugadoss-R00183157

Abstract In this project we would be predicting customer churn from bank data. We would be adopting 8 different classification models and apply stratifiedKfold cross validation to check the performance of the models. Stratified K Fold is used because it is best for classification problems. The main challenge would be the imbalanced data which would be handled using SMOTE before training the model. After cross validation of the models, three top performing models are taken and finally hyperparameter tuning is done using grid search to identify ideal hyperparameters for best performance.

1 Introduction

The data used in this project is customer churn data of bank, taken from kaggle-datasets(link to the dataset is available in the hyperlink). Predicting customers churn would be of utmost importance to the business, because it's a lot easier to retain a customer than getting a new one for any business and especially banks. The data has 10000 samples and it is labelled telling if a customer exited or not. There are 13 features and 1 target column which has either 1 or 0 1 means the customer exited and 0 means the customer did not. Hence this would be a classification problem. The 13 features include, Surname, age, estimated salary, credit score etc. out of which few or numerical and categorical. By predicting what kind of customers exit the business, we can actually give more attention to them to retain them.

2 Research

The research topic here is about scaling we have used two different scaling one is minmax scaler and the other is standard scaler

2.1 MinMax Scaler

Reference taken from https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02

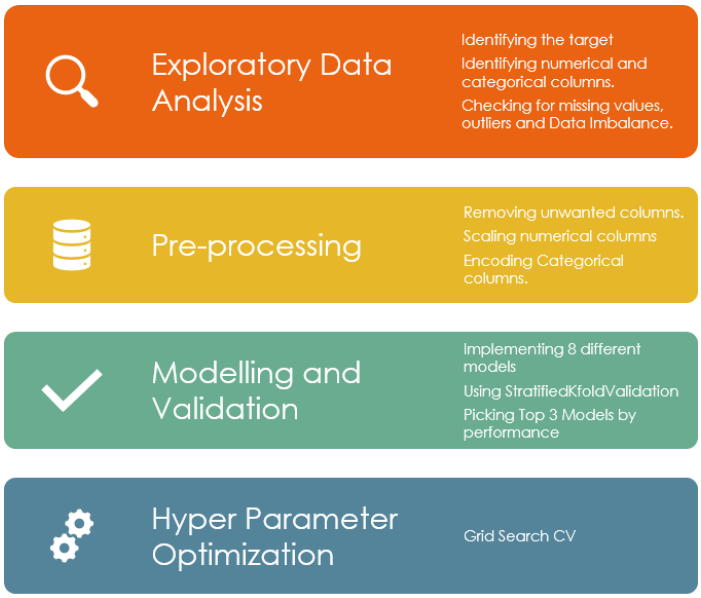
MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum.
MinMaxScaler preserves the shape of the original distribution. It doesn't meaningfully change the information embedded in the original data.

2.2 Standard Scaler

StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. StandardScaler results in a distribution with a standard deviation equal to 1.

3 Methodology

The Methodology shown in the below diagram would be implemented.



3.1 Exploratory Data Analysis

The churn data for bank customers has 14 columns and 10000 rows out of which 13 are the features and 1 is the target. The target column is *Exited* which says if the customer exited or not and has values 1 and 0 respectively and hence this would be a classification problem.

The below gives the description of the data along with its type if it is categorical or numerical

Label	Description
RowNumber	Row number of the Instance(Integer)
CustomerId	The unique ID given to each customer in a bank(Integer)
Surname	Last Name of the Customer (String)
CreditScore	Credit Score of the Customer(Integer)(Numerical)
Geography	Customer Location by Country (String)(Categorical)
Gender	Gender(String)(Categorical)
Age	Age of the customer (Integer)(Numerical)
Tenure	Number of years they customer was with the bank (Integer)(Numerical)
Balance	The balance money the customer had when this data was taken (Float)(Numerical)
NumOfProducts	The number of products the customer was interested in (Integer)(Numerical)
HasCrCard	Says, If a customer had credit card or not(Integer)(Categorical)
IsActiveMember	Says, If a customer was an active customer or not(Integer)(Categorical)
EstimatedSalary	The Estimated salary of the customer(Float)(Numerical)
Exited	The Target column which says If the customer is still with the bank or not.

The data is explored in depth to see if there are any unwanted features in the dataset, to identify that we check the unique number of values in each feature and below are the results

```
#Checking number of unique values in each columns
inputDataFrame.nunique()
```

```
RowNumber          10000
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
IsActiveMember         2
EstimatedSalary     9999
Exited                 2
dtype: int64
```

From the data we can infer that RowNumber and CustomerId has unique values in all instance and it can be removed since it doesn't have any correlation between the target column. We can also remove the Surname column which is just last names of people and does not have much importance.

The data is also checked for missing values, and looks likes there are no missing values in the data.

```
#Checking for missing values
inputDataFrame.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Now the data is checked for Imbalanced data. And we could that there is imbalance in the data.

```
dataY.columns=['target']
dataY.value_counts(normalize=True)
```

```
0    0.7963
1    0.2037
Name: Exited, dtype: float64
```

Almost 80% of the data is of class exited and the remaining 20% of the data is of not exited class.

3.2 Preprocessing

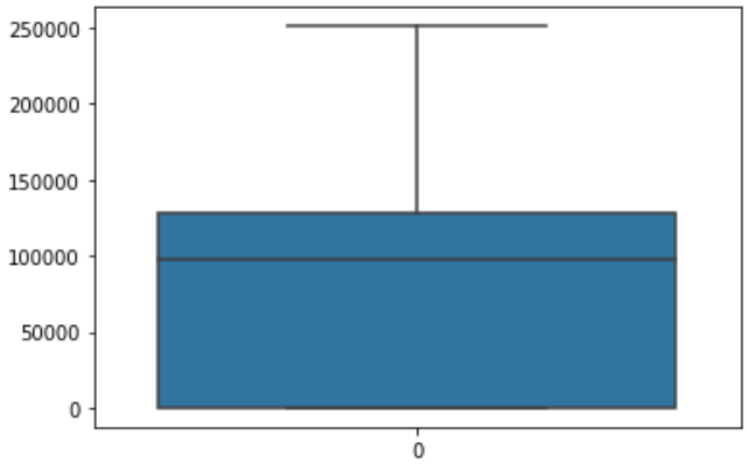
From the EDA(Exploratory Data Analysis) we can remove unwanted columns like RowNumber, Surname, CustomerId.

Outlier Detection:

The data is checked for Outliers and there is no outliers and no major outliers has been found. The same can be observed from the below boxplots of each of the features.

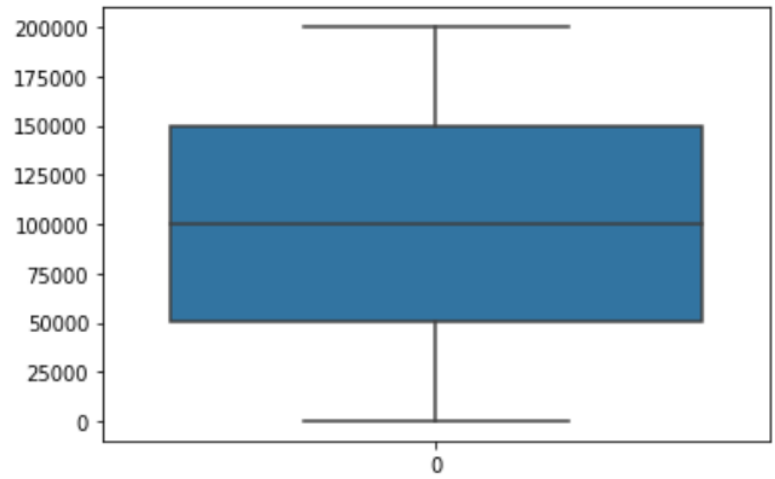
```
sns.boxplot(data=requiredColsDF['Balance'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18d32dd5518>
```



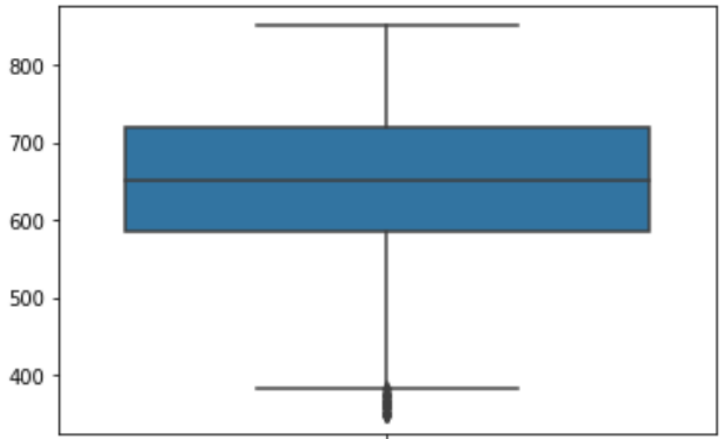

```
sns.boxplot(data=requiredColsDF['EstimatedSalary'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18d32fa92b0>
```



```
sns.boxplot(data=requiredColsDF['CreditScore'])
```

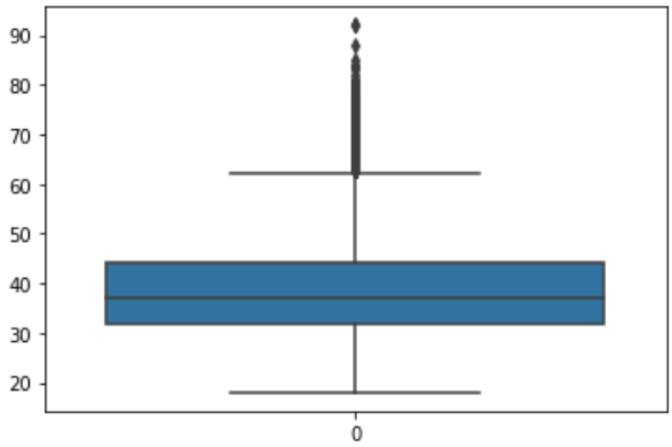
```
<matplotlib.axes._subplots.AxesSubplot at 0x18d32e97668>
```



We could see some datapoints near the whisker, since it is a cluster we will not be considering it as an outlier. It might be denoting a pattern in the data.

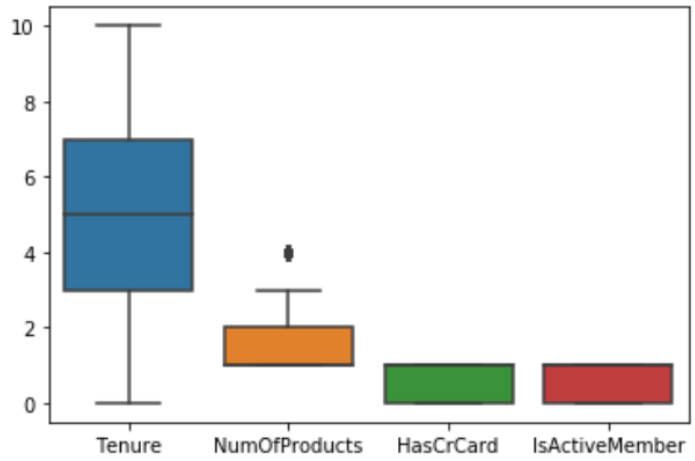
```
sns.boxplot(data=requiredColsDF['Age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18d32efc5f8>
```



```
sns.boxplot(data=requiredColsDF[numericalColumns])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18d32f63940>
```



From the above box plots we can see that there are no major outliers and we are not doing any outlier handling with this dataset

Missing Values:

From the EDA we saw that there are no missing values and since there are no outliers was made as missing values, no missing value treatment is done

Handling Categorical Values:

The next step is to handle all the categorical features which are text and encode them in such a way that we can train the model. We can use Label Encoder, Ordinal Encoder or One hot encoding. If we use Label Encoder the algorithm might assume that there is some inherent ordering based on the number assigned to each of the class in the feature and hence cannot use it. Ordinal Encoder cant be used because the two features Geography and Gender does not have any ordering in them so we will be using One-Hot Encoding to encode the categorical features.

Scaling:

By Looking at the data from the min and max of all the columns we could tell that the data is in different ranges. The features with higher magnitudes would dominate the classification process especially when using algorithms like kNearest Neighbors which uses Euclidean distance and it is more sensitive to different range of values[1]

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000

Handling Imbalance

During the EDA we noticed that there was imbalance in the data, it has to be handled. If it is not handled we might get a good accuracy but the model would predict well on the majority class and perform poorly on the smaller class, which we can see from the below results. The accuracy is 84.95% but when we have a look at the confusion matrix we can see that it gave an accuracy of 95.5% for the majority class and 42.75% for the minority class and hence we need to balance the data.

```
print("Test Data shape:", test_y.shape)
print(test_y.value_counts(normalize=True))
clf = RandomForestClassifier()
clf.fit(train_X,train_y)
print("Accuracy:",clf.score(test_X,test_y))
results=clf.predict(test_X)
print("Confusion Matrix:")
confusionMatrix= metrics.confusion_matrix(y_true= test_y, y_pred=results )
print(confusionMatrix)
```

```
Test Data shape: (2000,)
0    0.7925
1    0.2075
Name: Exited, dtype: float64
Accuracy: 0.8495
Confusion Matrix:
[[1528   57]
 [ 244  171]]
```

The data is then balanced using SMOTE.

Feature Selection

Feature selection is done using RFECV and the results from the feature selection is to use 11 features out of the 13 features.

After evaluating the models with the default parameters the following algorithms performed well and we could see that from the results.

KNearest Neighbors

```
Accuracy: 0.8486986516149263
[[4771. 1607.]
 [ 323. 6055.]]
```

Decision Tree Classifier

```
 Accuracy: 0.8405456255879585
[[5285. 1093.]
 [ 941. 5437.]]
```

Random Forest Classifier Results

```
Accuracy: 0.8900909375979932
[[5829.  549.]
[ 853. 5525.]]
```

The above models are not just selected based on the accuracy but also based on the confusion matrix results

3.3 Modelling and Validation

Once the preprocessing is done, we would be training 8 different models using stratifiedKfold validation to check which model performs well out of which top performing three models would be selected and would go through hyper parameter optimization.

The following are the models used in this project:

- LinearSVC
- KNeighborsClassifier
- DecisionTreeClassifier
- LogisticRegression
- GaussianNB
- RandomForestClassifier
- AdaBoostClassifier
- NearestCentroid

All the models are used with default parameters. Before training the model the data is balanced using SMOTE and the results can be seen below from the

screenshot, that the prediction for both class is approx. 70% which is the impact of balancing the data.

Sample screenshot of RandomForestClassifier

```
Accuracy: 0.7057522439080536
[[4540. 1838.]
 [ 516. 1106.]]
```

3.4 Hyperparameter Optimization

The selected models undergo hyperparameter optimization using gridsearchCV method and the ideal hyperparameters are identified.

K Neighbors classifier

Best parameters set found on development set:
{'n_neighbors': 1, 'p': 1} with a score of 0.911884603323926

Random Forest Classifier

Best parameters set found on development set:
{'criterion': 'gini', 'n_estimators': 104} with a score of 0.9159611163374098

Decision Tree Classifier

Best parameters set found on development set:
{'criterion': 'entropy', 'splitter': 'random'} with a score of 0.8564597052367513

4 Evaluation

We have used Scaling, One-Hot Encoding, Feature Selection and SMOTE preprocessing techniques.

In this section we would be evaluating the impact of preprocessing techniques and hyper paramter optimization.

Results before preprocessing techniques :

```
Accuracy: 0.852
Confusion Matrix:
[[1525   60]
[ 236  179]]
```

Results after preprocessing techniques:

```
Accuracy: 0.8949513954217623
Confusion Matrix:
[[5840.  538.]
[ 802. 5576.]]
```

From the above results we can see that the same Random Forest Classifier which was only predicting with higher accuracy for majority class is now also predicting well for the minority class as well because of balanced data. The overall accuracy has also increased after other pre processing techniques.

Results of crossvalidation on 8 different models :

```
classifiers=[LinearSVC(),\
        KNeighborsClassifier(),\
        DecisionTreeClassifier(),\
        LogisticRegression(),\
        GaussianNB(),\
        RandomForestClassifier(),\
        AdaBoostClassifier(),\
        NearestCentroid()]
```

LinearSVC()

```
Accuracy: 0.7042960175603636
[[4542. 1836.]
 [1936. 4442.]]
```

KNeighborsClassifier()

```
Accuracy: 0.8494825964252116
[[4747. 1631.]
 [ 289. 6089.]]
```

DecisionTreeClassifier()

```
Accuracy: 0.8400752587017873
[[5277. 1101.]
 [ 939. 5439.]]
```

LogisticRegression()

```
Accuracy: 0.705236751332706
[[4532. 1846.]
 [1914. 4464.]]
```

GaussianNB()

```
Accuracy: 0.7191125744747571
[[4739. 1639.]
 [1944. 4434.]]
```

RandomForestClassifier()

```
Accuracy: 0.8920507996237065
[[5816.  562.]
 [ 815. 5563.]]
```

AdaBoostClassifier()

```
Accuracy: 0.8297271872060206
[[5352. 1026.]
 [1146. 5232.]]
```

NearestCentroid()

```
Accuracy: 0.6999843211037944
[[4530. 1848.]
 [1979. 4399.]]
```

Based on the above data we can see that the **Knearest classifier , Random forest classifier and Decision tree classifer** performed really well on the data. The performance is not just based on the accuracy but also based on the confusion matrix.

Hence the three models will undergo hyperparameter optimization. The results for after applying the hyperparameters suggested by the gridsearchCV is shown below. *Shown in Hyperparameter Tuning section*

```
KNeighbors classifier

 Best parameters set found on development set:
{'n_neighbors': 1, 'p': 1} with a score of  0.9132173095014111
Random Forest Classifier

 Best parameters set found on development set:
{'criterion': 'entropy', 'n_estimators': 102} with a score of  0.9157259328943242
Decision Tree Classifier

 Best parameters set found on development set:
{'criterion': 'entropy', 'splitter': 'random'} with a score of  0.8573220445280653
```

Finally the top 3 models are applied to the actual test data which was not involved in the crossvalidation is checked with by default hyper parameters and after applying hyperparameters is shown below

Results before hyperparameter optimization

Knearest classifier

```
0.775
[[1259  326]
 [ 124  291]]
```

Random forest classifier

0.845

[[1475 110]

 [200 215]]

Decision tree classifer

0.7815

[[1344 241]

 [196 219]]

Results after hyperparameter optimization

Knearest classifier

0.804

[[1398 187]

 [205 210]]

Random forest classifier

0.852

[[1475 110]

 [186 229]]

Decision tree classifer

0.7805

[[1339 246]

 [193 222]]

Comparison of prediction accuracy based on different scaling technique is shown below

Standard scaled data

Accuracy: 0.8485

Confusion Matrix:

[[1530   55]

 [248 167]]

```
Minmax scaled data
Accuracy: 0.835
Confusion Matrix:
[[1522   63]
 [ 267  148]]
```

From the results we can see that the Standard scaler gives better results.

5 Conclusion

From the evaluation we could see that the over all results degraded a little with the actual test data. This could be because of the overfitting from hyperparameter optimization by gridsearch. This could be a potential area to work on.

.

References [1]

1. https://pdfs.semanticscholar.org/0890/20f8ee2a269f523d5fec62819f6e264aac6e.pdf
2. https://www-sciencedirect-om.cit.idm.oclc.org/science/article/pii/S0957417408004326
3. https://www-sciencedirect-com.cit.idm.oclc.org/science/article/pii/S1568494614000507
4. https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

[1] Note that the references cited are fictitious for this example.