Bank Customer Churn Modeling

Submitted By
Saurabh Manjrekar

Table of Contents

1. Feature Manipulation

The number of features in our dataset are not very large. But we will not be using all the features available to us and hence will be using only those features that are really important. This will help to improve the accuracy of the model as well as reduce the complexity that helps to train the algorithm faster.

We have 13 features in our dataset of which not all are of significance. Out of the given features RowNumber, CustomerId and Surname are eliminated as these features are not relevant in order to predict the future of customers.

Total number of independent variables remaining before applying the techniques: 10

Read data from CSV using pandas

In [73]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
dataset = pd.read_csv('Churn_Modelling.csv')
dataset = dataset.drop(["Surname","CustomerId","RowNumber"], axis=1)
y = dataset.iloc[:, 10].values # Exited
X = dataset.iloc[:,:10].values
print("Dataset shape:",dataset.shape)
print("X shape:      ",X.shape)
print("y shape:      ",y.shape)

dataset.iloc[:,:10].head()
```

```
Dataset shape: (10000, 11)
X shape:       (10000, 10)
y shape:       (10000,)
```

The three approaches used to extract the relevant features are:

1.1 Correlation Matrix

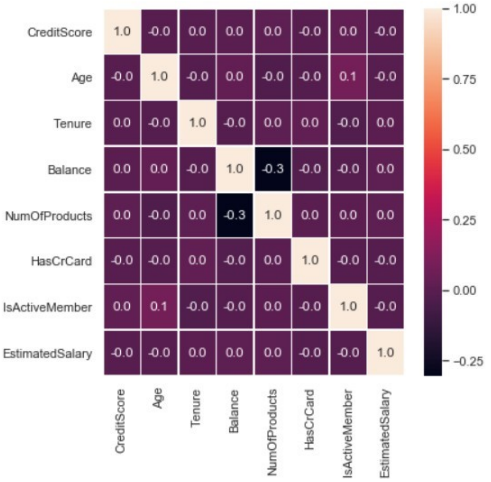
Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features. Correlation matrix or Heat-Map makes it easy to identify which features are most related to each other.

We plotted the correlation matrix to check multi-collinearity amongst the independent variables and no feature have correlation higher than 0.3. Hence, no feature is removed in this process

Total number of independent variables remaining after applying this technique: 10

```
In [63]: # Checking corelation between independent variables (X)
         import seaborn as sns # data visualization library
         #correlation map
         f,ax = plt.subplots(figsize=(6, 6))
         sns.heatmap(dataset.iloc[:,:10].corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x224dd376128>
```

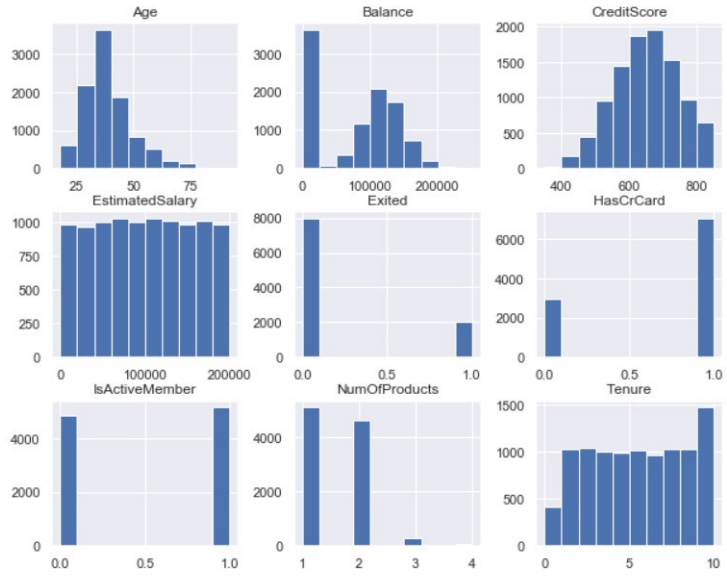


1.2 Improper Distribution

This technique is used to check the variables which are imbalanced. We plotted histograms for the variables and none of the variables exhibited an unequal distribution.

Total number of independent variables remaining after applying this technique: 10

```
In [72]: import seaborn as sns
         import matplotlib.pyplot as plt
         from pandas.plotting import scatter_matrix
         dataset.hist()
         plt.gcf().set_size_inches(10, 8)
```



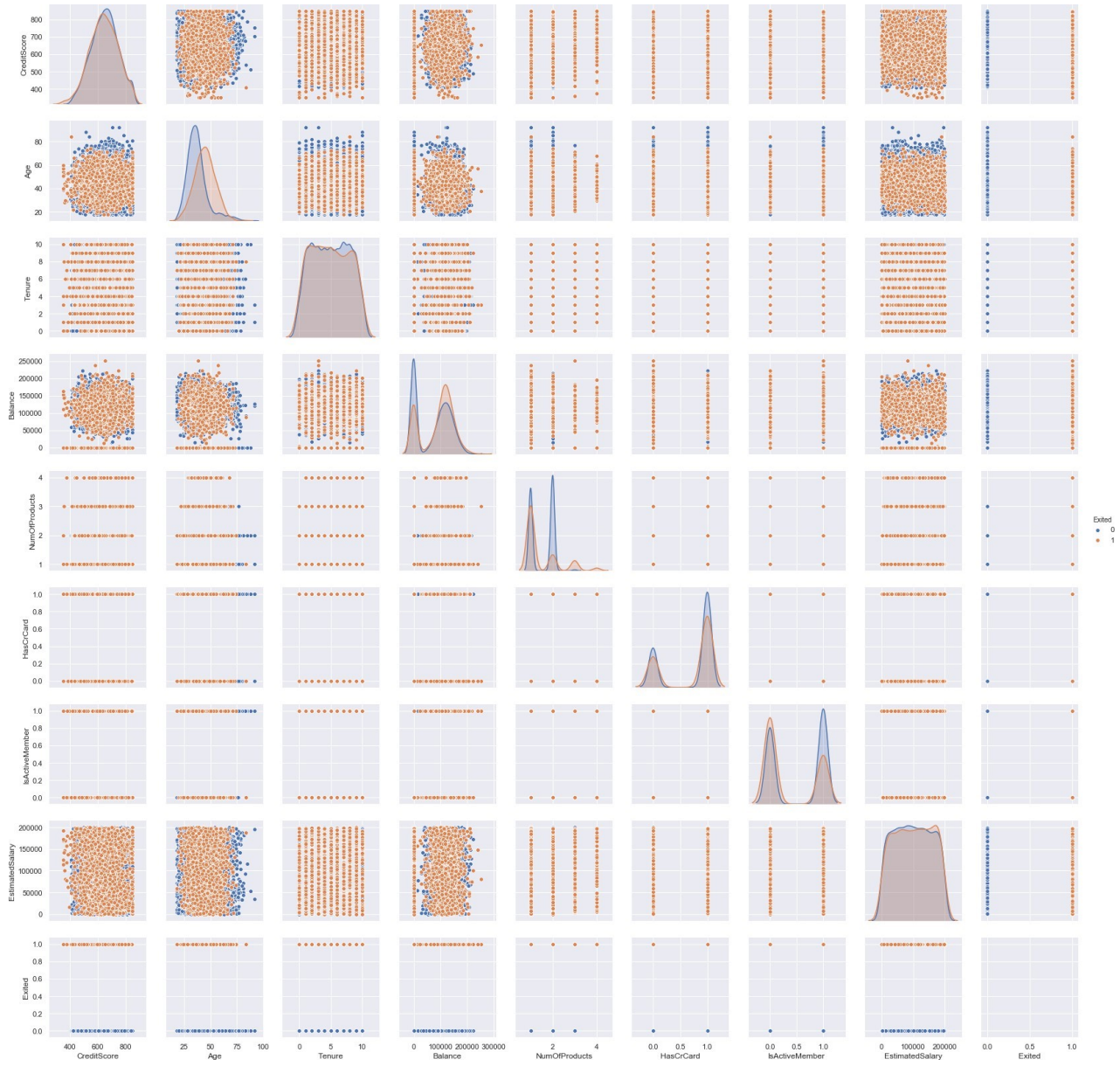
1.3 Scatter Plot

Scatter Plot tells us that every feature is able to seperate the exited and non-exited customers. We can conclude that all the remaining features are significant. The given Bank Customer dataset is biased towards customers who are still with the bank (Exited='0'). Without balancing the dataset, the model achieves High accuracy Low recall for Exited='1. Hence, sampling of the dataset is done by tuning parameter class_weight="balanced" for both the classifier models which helps the model to achieve better results.

Total number of independent variables remaining after applying this technique: 10

ScatterPlot

```
sns.pairplot(dataset,hue='Exited')
```



1.4 Categorical Feature Encoding

Categorical features like 'Gender' and 'Geography' needs to be encoded to numerical values using label encoders. 'Geography' feature is again encoded using one hot encoding for binarization of the feature which was assigned ordinal values (0, 1, 2) earlier during label encoding.

Encoding for categorical values

In [74]:
```
# We can see the blue dots are at edges and evry feature is able to seperate the exited and non exited customers
# Hence each feature is important for us

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
laben_Country = LabelEncoder()
print(X[:8,1], '... will now become:')
X[:,1] = laben_Country.fit_transform(X[:,1])
print(X[:8,1],"\n")
laben_Gender = LabelEncoder()
print(X[:8,2], '... will now become: ')
X[:,2] = laben_Gender.fit_transform(X[:,2])
print(X[:8,2],"\n")

countryhotencoder = OneHotEncoder(categorical_features = [1]) # 1 is the country column
X = countryhotencoder.fit_transform(X).toarray()
X=X[:,1:]

print("Encoded X shape:",X.shape)
print(X)
```

```
['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'Germany'] ... will now become:
[0 2 0 0 2 2 0 1]

['Female' 'Female' 'Female' 'Female' 'Female' 'Male' 'Male' 'Female'] ... will now become:
[0 0 0 0 0 1 1 0]

Encoded X shape: (10000, 11)
```

So, after initial data pre-processing and feature selection, we have got 11 features which are relevant for our model building.

2. Model Development

Here we need to classify whether a bank customer will leave the bank or not. Hence our problem statement is a particular case of supervised learning and binary classification having classes 0 (i.e. the customer will not leave the bank) and 1 (i.e. the customer will leave the bank).

2.1 Data Partitioning

Data split for training and testing

In [45]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
X_train_original, X_test, y_train_original, y_test = train_test_split(X, y, test_size=0.3,random_state=42)

print('Training Set Shape:   ', X_train_original.shape, y_train_original.shape)
print('Test Set Shape:       ', X_test.shape, y_test.shape)
# sns.set(color_codes=True)
# Every feature has atleast 2 different values across the dataset
```

```
Training Set Shape:    (7000, 11) (7000,)
Test Set Shape:        (3000, 11) (3000,)
```

2.2 Feature Scaling using StandardScaler

In our dataset some columns like ‘Gender’ and ‘Geography’ are having only binary values but some columns were having range of values (like 1- 900, 1- 100000) as well. So as part of data pre-processing, we have applied feature scaling on the dataset to change all features to the same scale. This allows for faster convergence on learning, and more uniformity for all weights.

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train_original)
X_test_scaled = sc.transform(X_test)
```

2.3 Model Building

Here we need to classify whether a customer will leave the bank or not and hence we can use various classification algorithms.

Also, Confusion matrix are printed to describe the performance of all the classification model on a set of test data for which the true values are known. It displays the performance of an algorithm and summarizes the prediction results.

2.3.1 Random Forest Classifier with hyper parameter tuning

```
from sklearn.model_selection import RandomizedSearchCV

# number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 50, num = 5)]
# number of features at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 50, num = 6)]
max_depth.append(None)
criterion=['entropy','gini']
# Method of selecting samples for training each tree
bootstrap = [True,False]
# create random grid
random_grid = {
 'n_estimators': n_estimators,
 'max_features': max_features,
 'max_depth': max_depth,
 'criterion': criterion,
 'bootstrap': bootstrap
 }
# Random search of parameters
rfc_random = RandomizedSearchCV(RandomForestClassifier(), param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, ra
# Fit the model
rfc_random.fit(X_train_scaled, y_train_original)
# print results
print("Best Parameters for Random Forest: ",rfc_random.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    4.7s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   11.9s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   21.6s finished
Best Parameters for Random Forest:  {'n_estimators': 40, 'max_features': 'auto', 'max_depth': 10, 'criterion': 'gini', 'bootstr
ap': False}
```

Prediction Using Random Forest Classifier With Tuned parameters

In [56]:
```
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, log_loss
from sklearn.metrics import roc_curve, auc, classification_report, confusion_matrix, accuracy_score
Forestclassifier = RandomForestClassifier(n_estimators= 40, criterion="gini", bootstrap = False, max_depth=10, max_features = 'a
Forestclassifier.fit(X_train_scaled,y_train_original)
y_predict = Forestclassifier.predict(X_test_scaled)
y_predict
```

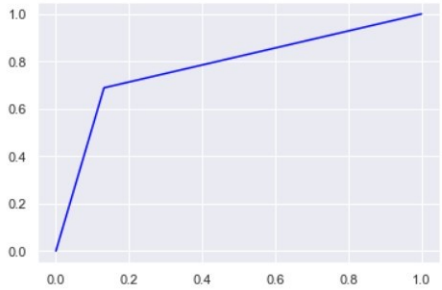
Out[56]:
```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

In [57]:
```
print("Accuracy using Random Forest Classifier:",accuracy_score(y_test, y_predict))
print("Confusion matrix \n",confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
fpr, tpr, thresholds= metrics.roc_curve(y_test,y_predict)
auc = metrics.roc_auc_score(y_test,y_predict, average='macro', sample_weight=None)
sns.set_style('darkgrid')
sns.lineplot(fpr,tpr,color ='blue')
plt.show()
```

```
Accuracy using Random Forest Classifier: 0.8333333333333334
Confusion matrix
 [[2098  318]
 [ 182  402]]
             precision    recall  f1-score   support

          0       0.92      0.87      0.89      2416
          1       0.56      0.69      0.62       584

avg / total       0.85      0.83      0.84      3000
```



2.3.2 Logistic Regression Classifier

Prediction Using Logistic Regression

In [58]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0,class_weight="balanced")
classifier.fit(X_train_scaled,y_train_original)
```

Out[58]:
```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=0,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [59]:
```
y_predict = classifier.predict(X_test_scaled)
y_predict
```

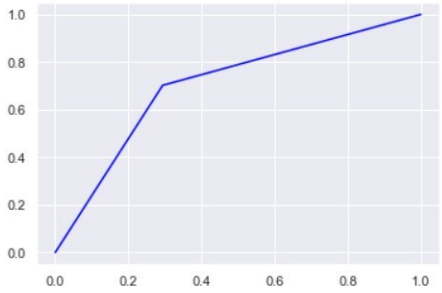
Out[59]:
```
array([1, 0, 1, ..., 0, 0, 0], dtype=int64)
```

```
In [60]: print("Accuracy using Logistic regression:",accuracy_score(y_test, y_predict))
         print("Confusion matrix \n",confusion_matrix(y_test,y_predict))
         print(classification_report(y_test,y_predict))
         fpr, tpr, thresholds= metrics.roc_curve(y_test,y_predict)
         auc = metrics.roc_auc_score(y_test,y_predict, average='macro', sample_weight=None)
         sns.set_style('darkgrid')
         sns.lineplot(fpr,tpr,color ='blue')
         plt.show()
```

```
Accuracy using Logistic regression: 0.7053333333333334
Confusion matrix
 [[1706  710]
 [ 174  410]]
             precision    recall  f1-score   support

          0       0.91      0.71      0.79      2416
          1       0.37      0.70      0.48       584

avg / total       0.80      0.71      0.73      3000
```



3. Model Comparison

After applying the classification algorithms, we have compared the accuracy of the models on the test set. Here we can see that **Random Forest Classifier** provides the best accuracy (83.33%) and best recall rate for Exited= '1' (69%) among all the classifiers.

Classifier	Accuracy	Recall
Random Forest	83.33 %	For 0's: 87.00 %
		For 1's: 69.00 %
Logistic Regression	70.53	For 0's: 71.00 %
		For 1's: 70.00 %

4. Future Improvement

For the dataset with fewer features, low computational models like Random Forest and Logistic Regression gives good accuracy and recall. But for better results, we can implement the ANN model.