Bank Customer Churn Prediction using Machine Learning Techniques

Sreejith Madhavankutty

July 24th, 2023

Hopkinton Credit Union – Customer Churn Prediction using Machine Learning Techniques

"No great marketing decisions have ever been made on qualitative data." — John Sculley (CEO of Apple Inc.)

Introduction

Customer Churn prediction means knowing which customers are likely to leave or unsubscribe from a service. For many organizations such as Hopkinton Credit Union (HCU), this is an important prediction. This is because acquiring new customers often costs more than retaining existing ones. Once you've identified customers at risk of churn, you need to know exactly what marketing efforts you should make with each customer to maximize their likelihood of staying.

Customers have different behaviors and preferences, and reasons for cancelling their subscriptions. Therefore, it is important to actively communicate with each of them to keep them on your customer list. You need to know which marketing activities are most effective for individual customers and when they are most effective. Impact of customer churn on businesses such as Hopkinton Credit Union (HCU) is huge because they recently they are lost many of their customer base to bigger banks such as Bank of America, JP Morgan etc. A company with a high churn rate loses many subscribers, resulting in lower growth rates and a greater impact on sales and profits. Companies with low churn rates can retain customers.

Problem Statement

In this project on **bank customer churn prediction** using machine learning, I am trying to explain how a local bank - Hopkinton Credit Union can use predictive analytics to help its marketing staff to identify which aspects of the service influence a customer's decision in this regard. Management can concentrate efforts on the improvement of service, keeping in mind these priorities. Using the data collected from existing customers, build a model that will help the marketing team identify potential customers who are relatively more likely to churn.

Given a Bank customer, we will build a **neural network-based classifier** that can determine whether they will leave or not in the next 6 months.

Data Collection

The Neural Network-based Classifier/ML Model work in this project will use the datasets taken from Open Source Dataset from Kaggle. This is a comprehensive dataset collected by the bank's marketing team. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, Credit Score, Geography, Gender, Age, Tenure, Balance, etc.

Data Dictionary

RowNumber: Row number.
CustomerId: Unique identification key for different customers.
Surname: Surname of the customer
Credit Score: Credit score is a measure of an individual's ability to pay back the borrowed amount. It is the numerical representation of their creditworthiness. A credit score is a 3-digit number that falls in the range of 300-900, 900 being the highest.
Geography: The country to which the customer belongs.
Gender: The gender of the customer.
Age: Age of the customer.
Tenure: The period of time a customer has been associated with the bank.
Balance: The account balance (the amount of money deposited in the bank account) of the customer.
NumOfProducts: How many accounts, bank account affiliated products the person has.
HasCrCard: Does the customer have a credit card through the bank?
IsActiveMember: Subjective, but for the concept
EstimatedSalary: Estimated salary of the customer.
Exited: Did they leave the bank after all?

Data Exploration

As part of the EDA, will perform Uni-variate analysis & bivariate analysis with Data Visualization. Generate insights from the dataset to find proportion of customers churned and retained. Some hypotheses to test are: -

(i) Customers with low credit score tend to churn, reasonable.
(ii) On average older customers are the most to churn, but it's questionable.
(iii) Customers with high balance are churning probably they are getting attracted by other banks offer to raise the wealth and this corresponds with their estimated salary.
(iv) Tenure, Credit card and being active mean are not explicitly helping in this case to highlight anything big for churn rate.

Deliverables

- EDA (Exploratory Data Analysis)
- Feature Engineering
- Build ANN & Train ANN (Use TensorFlow)
- Making Predictions and Evaluating Model using other ML Algorithms
    - Stochastic Gradient Descent (SGD) classifier
    - Logistic Regression classifier
    - Support Vector Machines (RBF kernel)
    - Support Vector Machines (Poly kernel)
    - Random Forest Classifier
    - Extreme Gradient Boost (XGBoost) classifier

- ML Classifiers - Visualize results (precision, recall, f1-score)
- Visualize ROC & AUC
- ML Classifier - Test Accuracy

Exploratory Data Analysis (EDA)

Pandas Profiling Report

Dataset statistics

Number of variables	11
Number of observations	10000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	859.5 KiB
Average record size in memory	88.0 B

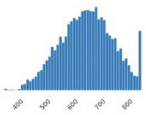
Variable types

Numeric	5
Categorical	6

Variables

CreditScore
Real number ($\mathbb{R}_{\geq 0}$)

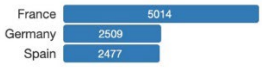
Distinct	460	Minimum	350
Distinct (%)	4.6%	Maximum	850
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	650.5288	Memory size	78.2 KiB



Toggle details

Geography
Categorical

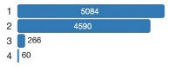
Distinct	3
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	78.2 KiB



Toggle details

NumOfProducts
Categorical

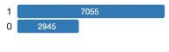
Distinct	4
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	78.2 KiB



Toggle details

HasCrCard
Categorical

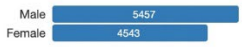
Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	78.2 KiB



Toggle details

Gender
Categorical

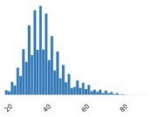
Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	78.2 KiB



Toggle details

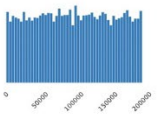
Age
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	70	**Minimum**	18
Distinct (%)	0.7%	**Maximum**	92
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	38.9218	**Memory size**	78.2 KiB



EstimatedSalary
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	9999	**Minimum**	11.58
Distinct (%)	> 99.9%	**Maximum**	199992.48
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	100090.2399	**Memory size**	78.2 KiB



Toggle details

Exited
Categorical

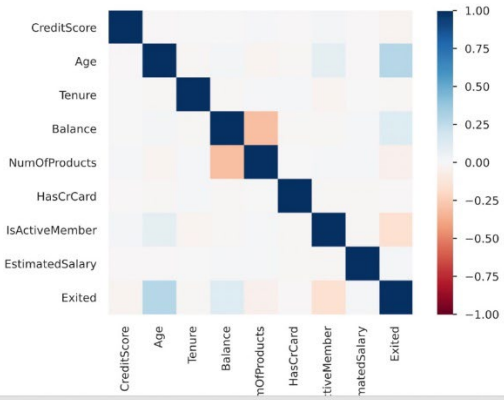
Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	78.2 KiB

Toggle details

Correlations

Pearson's r | Spearman's ρ | Kendall's τ | Phik (φk) | Cramér's V (φc)

Toggle correlation descriptions



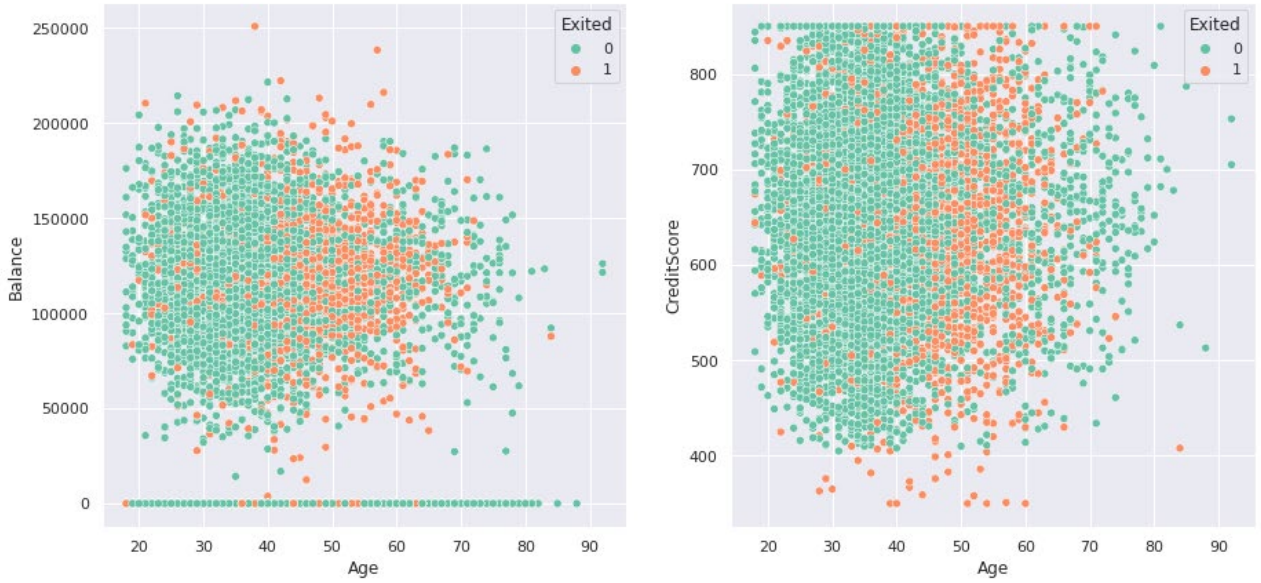
Perform bivariate analysis with Data Visualization

In [16]:
```
import seaborn as sns
sns.set(palette="Set2")
```

In [17]:
```
_, ax = plt.subplots(1, 3, figsize=(18, 6))
plt.subplots_adjust(wspace=0.3)
sns.countplot(x = "NumOfProducts", hue="Exited", data = df, ax= ax[0])
sns.countplot(x = "HasCrCard", hue="Exited", data = df, ax = ax[1])
sns.countplot(x = "IsActiveMember", hue="Exited", data = df, ax = ax[2])
```

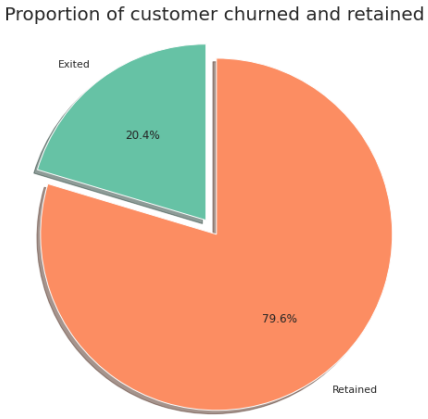
Out[17]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fb1d936a2d0>`

Insights : Customer with 3 or 4 products are higher chances to Churn



Insights :

- 40 to 70 years old customers are higher chances to churn
- Customer with CreditScore less then 400 are higher chances to churn

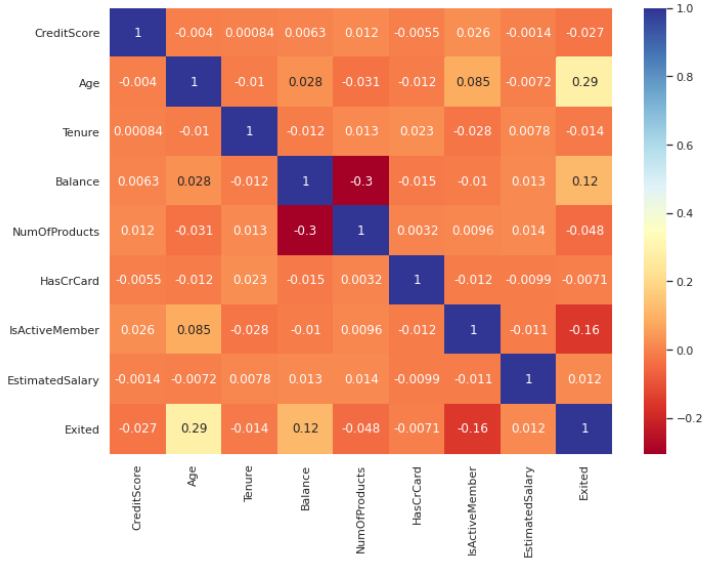


Insights :So about 20% of the customers have churned. So the baseline model could be to predict that 20% of the customers will churn. Given 20% is a small number, we need to ensure that the chosen model does predict with great accuracy this 20% as it is of interest to the bank to identify and keep this bunch as opposed to accurately predicting the customers that are retained.

Additional Insights from EDA/Bivariate Analysis:

- Customers with low credit score tend to churn, reasonable.
- On average older customers are the most to churn, but it's questionable.
- Customers with high balance are churning probably they are getting attracted by other banks offer to raise the wealth and this corresponds with their estimatedsalary.
- Tenure, Credit card and being active mean are not explicitly helping in this case to highlight anything big.
- Geographical location can determine the success of your business and can be a great tool to know how to play with your market.
- Apparently, it is possible that customers without credit card are churning, and it is obvious that the ones with it are not churning much.
- Female customers are churning than male this would be a factor of several things that can't be described without additional information's, also a great deal to consider gender so that the retention plan prepare promotions or offers based on affected gender,
- Not a surprise that inactive customers are churning than active ones.

Check for Correlation



Conclusion from EDA/Bivariate Analysis:

Overall conclusion is that all attributes have its impact to the performance for instance Tenure and Credit Score are functions of age the more you get older the more the relation of a customer and a bank become stronger as sign of loyalty, and of course balances depend on most of the case by your salary. These attributes are going to help us to engineer more case scenario by brin up new features that will help to punish negativity into predictions.

Feature Engineering

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning. To make machine learning work well on new tasks, it might be necessary to design and train better features. A “feature” is any measurable input that can be used in a predictive model — it could be the color of an object or the sound of someone’s voice. Feature engineering, in simple terms, is the act of converting raw observations into desired features using statistical or machine learning approaches.

The following steps were done as part of feature engineering in this project: -

- Prepare dataset.
- Encoding Categorical data
- Splitting data into training set and test set
- Apply feature scaling.

Prepare dataset.

```
In [32]: dataset = pd.read_csv("/content/gdrive/My Drive/data/bank.csv")
         # Exclude first 3 columns and the last column, because they are not useful.
         # Index starts at 0.
         x = dataset.iloc[:, 3:-1].values

         # Just get the last column of dataset for the dependent variable.
         y = dataset.iloc[:, -1].values
```

```
In [33]: print(x)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
In [34]: print(y)
```

```
[1 0 1 ... 1 1 0]
```

Encoding Categorical data

A one-hot encoding is a type of encoding in which an element of a finite set is represented by the index in that set, where only one element has its index set to “1” and all other elements are assigned indices within the range [0, n-1]. In contrast to binary encoding schemes, where each bit can represent 2 values (i.e., 0 and 1), this scheme assigns a unique value for each possible case.

```
In [35]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         # Get all the rwos for the 2 column and encode them.
         x[:, 2] = le.fit_transform(x[:, 2])
```

```
In [36]: print(x)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

```
In [38]: # We do one hot coding for geographical places because they have no relation among themselves like gender did.
         from sklearn.compose import ColumnTransformer
         from sklearn.preprocessing import OneHotEncoder
         ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
         x = np.array(ct.fit_transform(x))
```

```
In [39]: print(x)
```

```
[[1.0 0.0 1.0 ... 1 1 101348.88]
 [1.0 0.0 0.0 ... 0 1 112542.58]
 [1.0 0.0 1.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 1.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 1.0 ... 1 0 38190.78]]
```

Splitting data into training set and test set

```
In [40]: 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, random_state = 1)
```

Apply feature scaling.

Feature scaling is one of the most pervasive and difficult problems in machine learning, yet it’s one of the most important things to get right. In order to train a predictive model, we need data with a known set of features that needs to be scaled up or down as appropriate.

```
In [41]: # Apply feature scaling to all the features of both the training and the test set.
         from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         # Scaling fitted only to training set to avoid information leakage.
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

Build ANN & Train ANN using Tensorflow

Installed libraries from https://www.tensorflow.org/install

Initiliaze ANN

In [43]:
```
import tensorflow as tf
```

In [44]:
```
# Create a var to represtn the ANN as an instance of the sequential class that initializes our ANN.
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

In [45]:
```
# add() method belongs to the sequential class.
# add a fully connected layer (hidden), which will be a new object for the dense class.
# Argument for dense() -
# 'units' - There are called hyperparameters and are the hidden neurons.
# 'activation' - activation function in the hidden layer must be a rectifier activation function.
ann.add(tf.keras.layers.Dense(units=6, activation='relu')) # 'relu' - codename for rectifier activation function.
```

Adding the second hidden layer

In [46]:
```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

In [47]:
```
# As the output layer is binary we only need one neuron to match with the output layer, so 'units' = 1
# We need a sigmoid activation function, because it will give the predictions, as well as the probabilities.
# If we were predicting a categorical variable, activation = 'sigmax'
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Train ANN

Compile ANN

In [48]:
```
# Arguments -
# 'optimizer' - We choose 'adam' optimizer as it can perform stochastic gradient descent.
#  The stochastic gradient descent will optimize the weights to minimize the loss between the actual and the predicted values.
# 'loss' - When we are doing a binary classification when we are predicting a binary variable use 'binary_crossentropy' and
#  if we had categorical variable to predict we would have used 'categorical_crossentropy'
# 'metrics' -
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Train ANN on Training Dataset

In [49]:
```
# Batch learning, default batch size to be compared to the actual results = 32
ann.fit(X_train, y_train, batch_size=32, epochs=100)
# As we can see the accuracy converges around 0.86 around epoch = 20, so we didn't need 100 epochs.
```

```
Epoch 1/100
250/250 [==============================] - 1s 1ms/step - loss: 0.5903 - accuracy: 0.7968
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4895 - accuracy: 0.7985
```

Making Predictions and Evaluating Models using different ML Algorithms

Predict Test Results

In [51]:
```
# These are predicted probabilities
y_predicted = ann.predict(X_test)
# Convert Predicted probabilities into binary outcome
y_predicted = (y_predicted > 0.5)
print(np.concatenate((y_predicted.reshape(len(y_predicted),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [1 0]]
```

Making Confusion Matrix

In [53]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_predicted)
print(cm)
accuracy_score(y_test, y_predicted)
```

```
[[1518   67]
 [ 209  206]]
```

Out[53]: 0.862

Therefore, Accuracy = 86%

Total correct predictions that customers stay in bank = 1518

Total correct predictions that customers leave the bank = 206

Total incorrect predictions that customers stay in the bank = 209

Total incorrect predictions that customers leave in the bank = 67

Create a training data set with the categorical variables and continuous variables. Include the 'Exited' target variable in the data.

In [8]:
```
data = pd.read_csv("/content/gdrive/My Drive/data/bank.csv")
```

In [9]:
```
data.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, inplace=True)
```

In [10]:
```
data.head()
```

Out[10]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [12]:
```
# list of continuous and categorical variables/features

continuous_vars = ['CreditScore',  'Age', 'Tenure', 'Balance','NumOfProducts', 'EstimatedSalary']
categorical_vars = ['HasCrCard', 'IsActiveMember','Geography', 'Gender']

# separating the train and test data using a 80%-20% split

data_train = data.sample(frac=0.8, random_state=100)
data_test = data.drop(data_train.index)

# check the number of rows in each data set for verification

print('Number of rows in train data: ', len(data_train))
print('Number of rows in test data: ', len(data_test))

print()

data_train = data_train[['Exited'] + continuous_vars + categorical_vars]
data_train.head()
```

```
Number of rows in train data:  8000
Number of rows in test data:  2000
```

Machine Learning classifiers (selecting optimal parameters)

Try to train different machine learning classification models to our data. Once we get the model details for each of the models, we can select the best model from them for our training and testing purposes.

In [23]:
```
# important libraries

from sklearn.model_selection import GridSearchCV

# models

from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# metrics

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

import time
```

In [17]:
```
# this method will show us the details of each model
# which will help us in deciding the best model

def best_model(model):
  print(model.best_score_)
  print(model.best_params_)
  print(model.best_estimator_)
```

SGD Classifier

In [24]:
```
# SGD classifier
start_time = time.time()

parameters = {'loss': ['hinge', 'log'],
              'max_iter': [50, 100, 200, 300],
              'fit_intercept':[True],
              'penalty':['l2'],
              'tol':[0.00001, 0.0001, 0.000001]}

SGD_grid_model = GridSearchCV(SGDClassifier(),
                              param_grid=parameters,
                              cv=10,
                              refit=True,
                              verbose=0)

SGD_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(SGD_grid_model)
```

```
[INFO] Time taken: 24.3 seconds.

0.81375
{'fit_intercept': True, 'loss': 'log', 'max_iter': 50, 'penalty': 'l2', 'tol': 0.0001}
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=50,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.0001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Logistic Regression Classifier

In [25]:
```
# Logistic Regression classifier

start_time = time.time()

parameters = {'C': [0.1, 0.5, 1, 5, 10, 50, 100],
              'max_iter': [50, 100, 200, 300],
              'fit_intercept':[True],
              'intercept_scaling':[1],
              'penalty':['l2'],
              'tol':[0.00001, 0.0001, 0.000001]}

LR_grid_model = GridSearchCV(LogisticRegression(),
                             param_grid=parameters,
                             cv=10,
                             refit=True,
                             verbose=0)

LR_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(LR_grid_model)
```

```
[INFO] Time taken: 60.9 seconds.

0.813875
{'C': 0.1, 'fit_intercept': True, 'intercept_scaling': 1, 'max_iter': 50, 'penalty': 'l2', 'tol': 1e-05}
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=50,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=1e-05, verbose=0,
                   warm_start=False)
```

Support Vector Machines (RBF Kernel)

In [26]:
```
# Support Vector Machines (RBF kernel)

start_time = time.time()

parameters = {'C': [1, 10, 50, 100],
              'gamma': [0.1, 0.01, 0.001],
              'probability': [True],
              'kernel': ['rbf']}

SVM_rbf_grid_model = GridSearchCV(SVC(),
                                  parameters,
                                  cv=5,
                                  refit=True,
                                  verbose=0)

SVM_rbf_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(SVM_rbf_grid_model)
```

```
[INFO] Time taken: 464.3 seconds.

0.8466249999999998
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf', 'probability': True}
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

Support Vector Machines (Poly Kernel)

In [28]:
```
# Support Vector Machines (Poly kernel)

start_time = time.time()

parameters = {'C': [1, 10, 50, 100],
              'gamma': [0.1, 0.01, 0.001],
              'probability': [True],
              'kernel': ['poly'],
              'degree': [2, 3]}

SVM_poly_grid_model = GridSearchCV(SVC(),
                                   parameters,
                                   cv=5,
                                   refit=True,
                                   verbose=0)

SVM_poly_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(SVM_poly_grid_model)
```

```
[INFO] Time taken: 561.0 seconds.

0.852125
{'C': 100, 'degree': 2, 'gamma': 0.1, 'kernel': 'poly', 'probability': True}
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma=0.1, kernel='poly',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

Random Forest Classifier

In [29]:
```
# Random Forest Classifier

start_time = time.time()

parameters = {'max_depth': [6, 7, 8, 9, 10],
              'max_features': [5, 6, 7, 8, 9],
              'n_estimators':[10, 50, 100],
              'min_samples_split': [3, 5, 6, 7]}

RF_grid_model = GridSearchCV(RandomForestClassifier(),
                             parameters,
                             cv=10,
                             refit=True,
                             verbose=0)

RF_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(RF_grid_model)
```

```
[INFO] Time taken: 2152.8 seconds.

0.8654999999999999
{'max_depth': 9, 'max_features': 9, 'min_samples_split': 6, 'n_estimators': 50}
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=9, max_features=9,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=6,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Extreme Gradient Boost (XGBoost) Classifier

In [30]:
```
# Extreme Gradient Boost (XGBoost) classifier

start_time = time.time()

parameters = {'max_depth': [5, 6, 7, 8],
              'gamma': [0.01, 0.001, 0.001],
              'min_child_weight': [1, 5, 10],
              'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
              'n_estimators': [5, 10, 20, 100]}

XGB_grid_model = GridSearchCV(XGBClassifier(),
                              parameters,
                              cv=10,
                              refit=True,
                              verbose=0)

XGB_grid_model.fit(data_train.loc[:, data_train.columns != 'Exited'], data_train.Exited)

print('[INFO] Time taken: %.1f seconds.\n' % (time.time() - start_time))

best_model(XGB_grid_model)
```

```
[INFO] Time taken: 2450.2 seconds.

0.8630000000000001
{'gamma': 0.001, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 100}
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.001,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

ML Classifiers - Visualize results (precision, recall, f1-score)

[INFO] SGD classifier:

	precision	recall	f1-score	support
0	0.83	0.95	0.89	6382
1	0.55	0.24	0.33	1618
accuracy			0.81	8000
macro avg	0.69	0.59	0.61	8000
weighted avg	0.77	0.81	0.77	8000

[INFO] Logistic Regression classifier:

	precision	recall	f1-score	support
0	0.82	0.98	0.89	6382
1	0.69	0.15	0.25	1618
accuracy			0.81	8000
macro avg	0.75	0.57	0.57	8000
weighted avg	0.79	0.81	0.76	8000

[INFO] SVM (RBF) classifier:

	precision	recall	f1-score	support
0	0.86	0.98	0.92	6382
1	0.84	0.38	0.53	1618
accuracy			0.86	8000
macro avg	0.85	0.68	0.72	8000
weighted avg	0.86	0.86	0.84	8000

[INFO] SVM (Poly) classifier:

	precision	recall	f1-score	support
0	0.86	0.98	0.91	6382
1	0.81	0.36	0.50	1618
accuracy			0.85	8000
macro avg	0.83	0.67	0.71	8000
weighted avg	0.85	0.85	0.83	8000

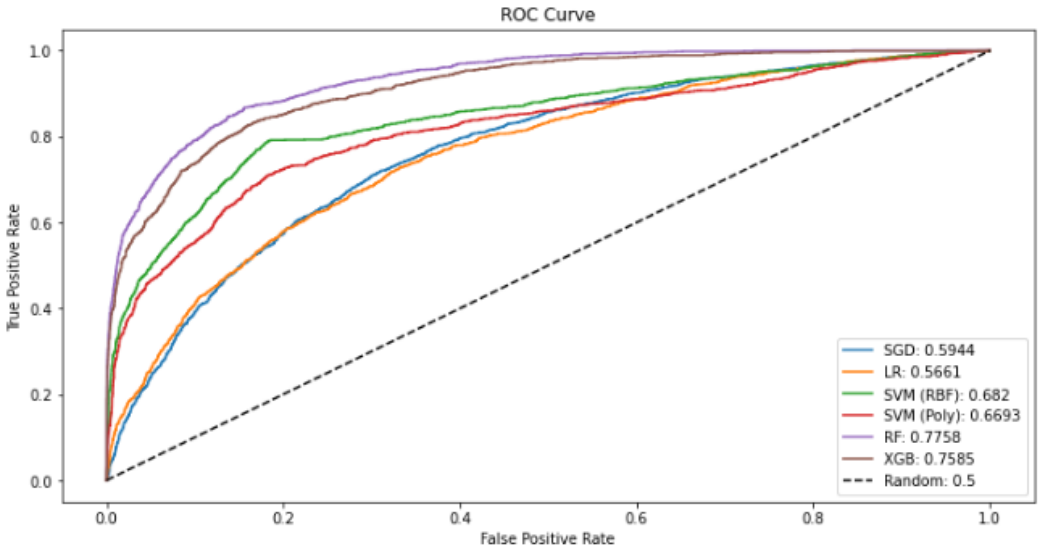
[INFO] Random Forest classifier:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	6382
1	0.89	0.57	0.69	1618
accuracy			0.90	8000
macro avg	0.89	0.78	0.82	8000
weighted avg	0.90	0.90	0.89	8000

[INFO] Extreme Gradient Boost (XGB) classifier:

	precision	recall	f1-score	support
0	0.89	0.98	0.93	6382
1	0.85	0.54	0.66	1618
accuracy			0.89	8000
macro avg	0.87	0.76	0.80	8000
weighted avg	0.88	0.89	0.88	8000

Visualize ROC & AUC



ML Classifier - Test Accuracy

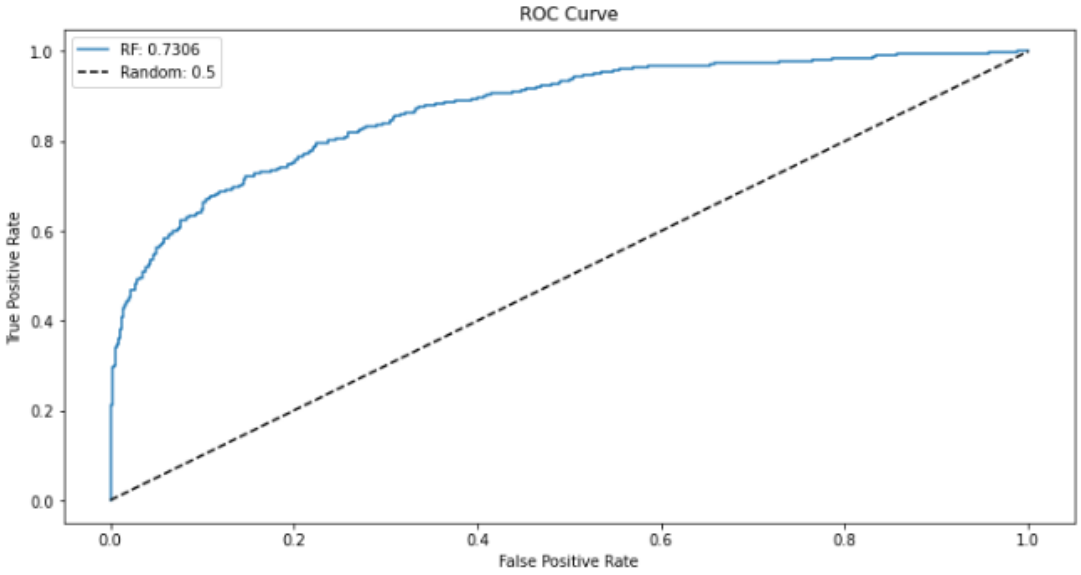
In [52]:
```
# classification report for the test data

print(classification_report(data_test.Exited,  rf_model.predict(data_test.loc[:, data_test.columns != 'Exited'])))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1581
           1       0.81      0.49      0.61       419

    accuracy                           0.87      2000
   macro avg       0.84      0.73      0.77      2000
weighted avg       0.86      0.87      0.86      2000
```



Conclusion

The precision of the model on previously unseen test data is slightly higher with regard to predicting 1's i.e., those customers that churn. However, in as much as the model has a high accuracy, it still misses about half of those who end up churning. This could be improved by providing retraining the model with more data over time.

Recommendations for Hopkinton Credit Union Bank

The most important signs to look out for customer churn are the following: -

- client's age
- credit speed
- expected profit.
- account balance
- number of products

So, it's better to plan the marketing strategy and product offering/service by carefully evaluating these.

To predict customer churn, use a model based on the Random Forest algorithm.

References

- *S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22–31, June 2014*
- *S. Moro, R. Laureano and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology.*
- *scikit-learn.org*
- *Tensorflow*
- *Google Colab*

Appendix

Github repository
Project Website