Galatasaray University

Engineering and Technology Faculty

Computer Engineering

Bank Customer Churn Prediction by Classification Methods

Data Mining Class Project

Doğa Yağmur Yılmaz

19401852

May 2023

Contents

Introduction

A classification problem involves assigning input data to a particular category or group, such as identifying whether a customer is likely to churn or not, or if an email is spam or not spam. The output variable in classification problems is typically a category or label, which can be used to draw conclusions about the input data. Classification models are used to predict the category of unknown data based on the information gathered from the input data. Overall, classification problems are prevalent in various fields and involve identifying the appropriate category for input data. Therefore, customer churn prediction is a typical classification problem.

Customer churn analyzes recent user behavior to identify the cessation of a partnership between a company and a customer, which can negatively impact the company's growth. Using machine learning algorithms, accurate predictions of customer churn can be made to adjust business strategies and facilitate decision-making for the progress of the corporation. Therefore, employing these methods enables companies to make informed decisions based on data analysis, which can ultimately improve business outcomes.

About the Project

Bank customer churn prediction by classification methods is the process of using machine learning algorithms to predict which banking customers are likely to leave or discontinue their relationship with a bank. Churn prediction is a common problem in the banking industry, as losing customers can result in significant financial losses and reduced growth.

Classification methods are used to predict bank customer churn by analyzing customer behavior and identifying patterns that are associated with churn. These methods typically involve building a model that can learn from historical data to identify the factors that contribute to customer churn. For example, some factors that may be associated with churn include the customer's age, account balance, credit score, and credit card usage.

Once the model is trained, it can be used to predict whether a new banking customer is likely to churn based on their characteristics and behavior. This information can then be used by the bank to take proactive measures to prevent churn, such as offering promotions, improving customer service, or identifying areas for product improvement.

Common classification methods used for bank customer churn prediction include decision trees, random forests, logistic regression, k-nearest neighbor, gaussian naïve bayes and support vector machines. These methods can provide accurate predictions and enable banks to take proactive measures to reduce customer churn, ultimately improving business outcomes. By retaining more customers, banks can increase their profitability and strengthen their customer relationships.

The Dataset

The dataset for bank customer churn prediction includes 10,000 records and 17 features. Some of them are more relevant than others in predicting customer churn. CustomerId and Surname are not considered relevant, while CreditScore, Geography (which is location of the customer), Gender, Age, Tenure (refers to the number of years that the customer has been a client of the bank), Balance, NumOfProducts, HasCrCard, IsActiveMember, and EstimatedSalary are all potentially informative features. These features can provide insights into factors that may contribute to customer churn, such as a customer's location, age, account balance, and level of activity. The dataset also includes information about whether they had a complaint and their satisfaction score. Additionally, the dataset includes information on the type of card held by the customer and the points earned for using the credit card.

The features, their non-null counts and data types are listed below:

```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   CustomerId          10000 non-null  int64
 1   Surname             10000 non-null  object
 2   CreditScore         10000 non-null  int64
 3   Geography           10000 non-null  object
 4   Gender              10000 non-null  object
 5   Age                 10000 non-null  int64
 6   Tenure              10000 non-null  int64
 7   Balance             10000 non-null  float64
 8   NumOfProducts       10000 non-null  int64
 9   HasCrCard           10000 non-null  int64
 10  IsActiveMember      10000 non-null  int64
 11  EstimatedSalary     10000 non-null  float64
 12  Complain            10000 non-null  int64
 13  Satisfaction_Score  10000 non-null  int64
 14  Card_Type           10000 non-null  object
 15  Point_Earned        10000 non-null  int64
 16  Exited              10000 non-null  int64
dtypes: float64(2), int64(11), object(4)
```

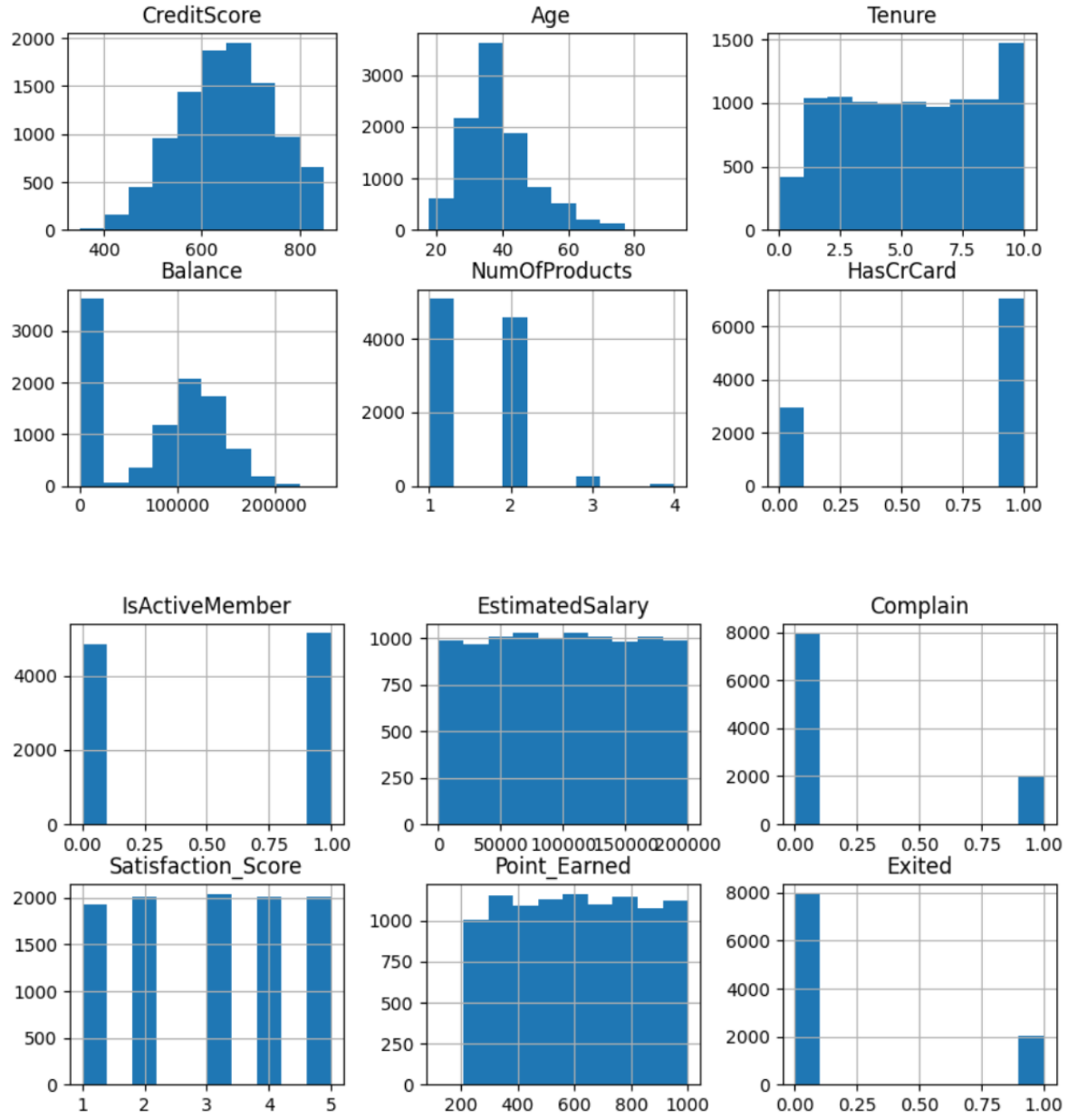
To see how many different values are in features, we can look at the data as follows:

```
Unique values :
 CustomerId            10000
Surname                2932
CreditScore             460
Geography                 3
Gender                    2
Age                      70
Tenure                   11
Balance                6382
NumOfProducts             4
HasCrCard                 2
IsActiveMember            2
EstimatedSalary        9999
Complain                  2
Satisfaction_Score        5
Card_Type                 4
Point_Earned            785
Exited                    2
```

Descriptive Data Analysis

As we saw above, the data type of surname, geography, gender and cardtype features are objet type. While running our classification models, we will map these feature values to integers.

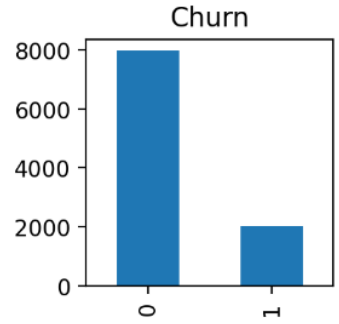
Let's print the histograms to see which of our other data type features are classified as binary.



Let's see how many 0's and 1's there are in the "Exited" feature, which we will consider as Outcome, and see our distribution of "churn" and "non-churn" customers.

```
0    7962
1    2038
Name: Exited, dtype: int64
```

```
Text(0.5, 1.0, 'Churn')
```



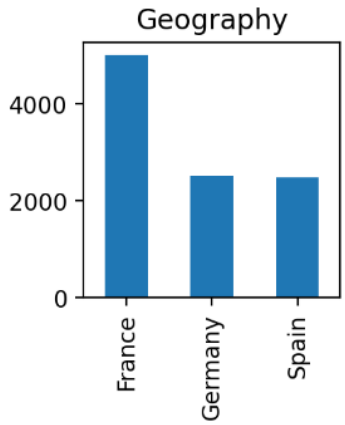
Next, we will analyze the data distributions and impacts of the features that belong to the object data type.

Let's look at the distribution of customers and the number of customers leaving the bank by country.

As we can see, the number of customers of the bank in France is twice the number of customers in Germany and Spain. Looking at the number of churned customers by country, we see that Spain and France have the same proportional churn value, while despite Germany having half the number of customers compared to France, they have the same number of customers churning, and we see that 1/3 of their customers churn internally.

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
Text(0.5, 1.0, 'Geography')
```



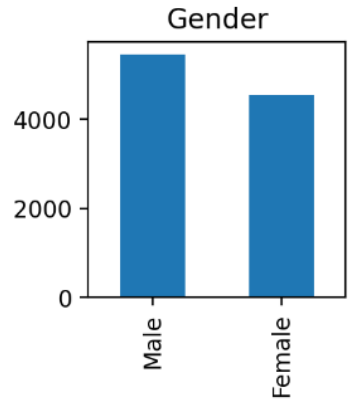

```
pd.crosstab(df.Geography,df.Exited)
```

Exited	0	1
Geography		
France	4203	811
Germany	1695	814
Spain	2064	413

Upon examining the gender distribution and churn rates of the data, it becomes evident that 45% of the customers are female, and 55% are male. Despite the lower proportion of female customers in the data, we observe that the number of churners among females is higher.

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

```
Text(0.5, 1.0, 'Gender')
```

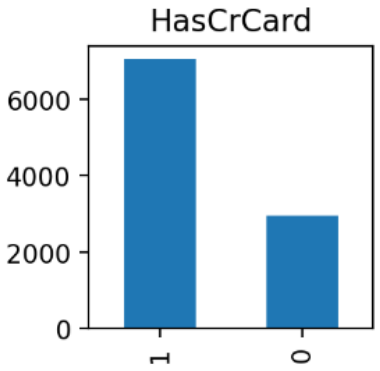


Exited	0	1
Gender		
Female	3404	1139
Male	4558	899

When we examine the relationship between the number of credit card holders and churn, we observe that approximately 21% of non-credit card holders churned while approximately 20% of credit card holders churned. Therefore, even though numerically, more credit card holders seem to churn, proportionally, there is no significant effect.

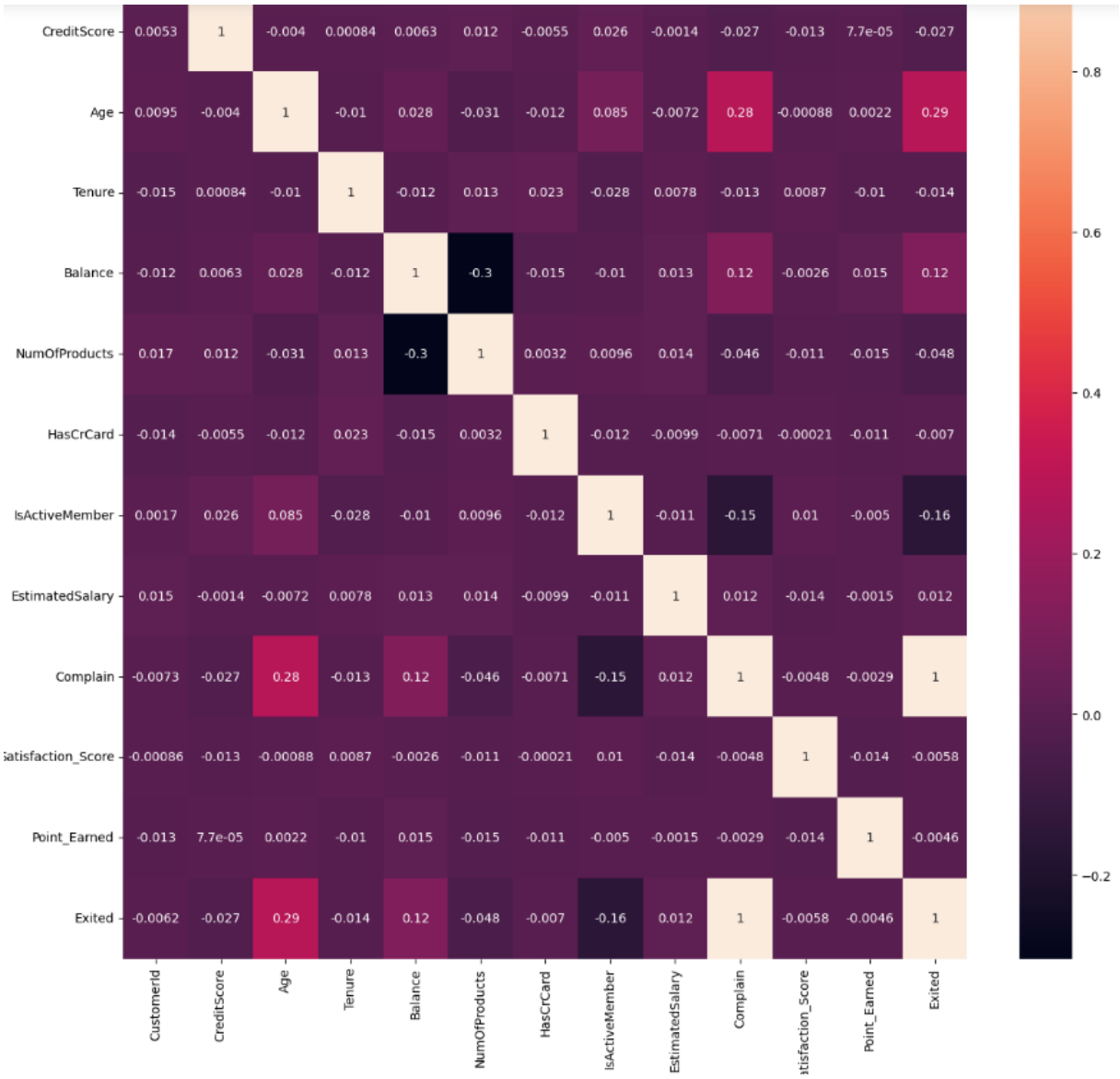
```
1    7055
0    2945
Name: HasCrCard, dtype: int64
```

```
Text(0.5, 1.0, 'HasCrCard')
```



Exited	0	1
HasCrCard		
0	2332	613
1	5630	1425

Let's examine the relationship between the features more clearly by using a heatmap.



According to what we see in the heatmap, the correlation value between "complain" and "exited" is “1”, which means that every customer who has complained has also left the bank. Therefore, if we run any algorithm to predict churn, it will most likely make accurate predictions near-perfect accuracy of 1, as it will predict that every customer with a complaint value of 1 has churned.

```
data[(data['Complain']==0)&(data['Exited']==0)].shape[0]
```

```
7952
```

```
data[(data['Complain']==1)&(data['Exited']==1)].shape[0]
```

```
2034
```

```
data[(data['Complain']==0)&(data['Exited']==1)].shape[0]
```

```
4
```

```
data[(data['Complain']==1)&(data['Exited']==0)].shape[0]
```

```
10
```

```
print(classification_report(data['Complain'], data['Exited']))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7956
           1       1.00      1.00      1.00      2044

    accuracy                           1.00     10000
   macro avg       1.00      1.00      1.00     10000
weighted avg       1.00      1.00      1.00     10000
```

As seen, only 14 out of 10,000 records have complained but did not churn. Therefore, in the data preprocessing stage, we should drop the 'complain' feature to avoid encountering overfitting.

Furthermore, since there is no high correlation between other features that I think will have an impact on churn and have analyzed, I will include them in the model.

Data Preprocessing

After making sure that there are no missing values in the data, I create a copy of the data. I drop the 'CustomerId' and 'Surname' features, which are not useful in the classification. For the data that belongs to the object data type, I find the classes and map them to integer values. Additionally, I need to drop the 'complain' feature to avoid encountering overfitting.

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

```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CreditScore        10000 non-null  int64
 1   Geography          10000 non-null  object
 2   Gender             10000 non-null  object
 3   Age                10000 non-null  int64
 4   Tenure             10000 non-null  int64
 5   Balance            10000 non-null  float64
 6   NumOfProducts      10000 non-null  int64
 7   HasCrCard          10000 non-null  int64
 8   IsActiveMember     10000 non-null  int64
 9   EstimatedSalary    10000 non-null  float64
 10  Complain           10000 non-null  int64
 11  Satisfaction_Score 10000 non-null  int64
 12  Card_Type          10000 non-null  object
 13  Point_Earned       10000 non-null  int64
 14  Exited             10000 non-null  int64
dtypes: float64(2), int64(10), object(3)
```

```
data['Geography']=data['Geography'].map({'France':1,'Spain':3,'Germany':2})
data['Gender']=data['Gender'].map({'Male':1,'Female':0})
data['Card_Type']=data['Card_Type'].map({'DIAMOND':3, 'GOLD':1, 'SILVER':0, 'PLATINUM':2})
```

```
data.drop(columns=['Complain'],inplace=True)
```

I separate the 'exited' feature as “Y” and the other features as “X” for prediction.

```
X = data.iloc[:, :-1].values
Y = data.iloc[:, -1].values
```

```
Y
```

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

I split the dataset into the training set and test set. I am setting aside 3000 out of my 10000 data for the test.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
```

Then I normalize the data sets.

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification Algorithms to Use

I use various classification algorithms to make accurate predictions. These algorithms include Decision Tree, XGBoost, Random Forest, Support Vector Machine (SVM), Logistic Regression, K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), and Neural Network. Decision Tree is a tree-based algorithm that uses entropy or Gini impurity to split the dataset into smaller subsets. XGBoost is an ensemble learning method that combines multiple decision trees and applies gradient boosting to improve model accuracy. Random Forest is another ensemble learning method that uses bagging to improve decision tree models. SVM is a supervised learning algorithm that uses hyperplanes to separate different classes. Logistic Regression is a statistical method that estimates the probability of a binary outcome using a logistic function. KNN is a simple algorithm that calculates the distance between the data points to classify them into different classes. GNB is a probabilistic algorithm that uses Bayes' theorem to calculate the conditional probabilities of different outcomes. Neural Network is a complex algorithm that simulates the structure and function of the human brain using artificial neurons and layers to learn complex patterns in the data. By combining these different algorithms and comparing their performance, I can select the most accurate model for my bank customer churn prediction project.

Results

Accuracy

Accuracy is a common metric used in machine learning to measure the performance of a classification model. It refers to the percentage of correct predictions made by the model out of all the predictions made. In other words, it measures how often the model is correct in its predictions. The accuracy score is calculated by dividing the number of correct predictions (which are number of TP+TN) by the total number of predictions made (number of data in the dataset), and it can range from 0 to 1, or 0% to 100%. A higher accuracy score indicates that the model is making more correct predictions, while a lower accuracy score suggests that the model is making more incorrect predictions.

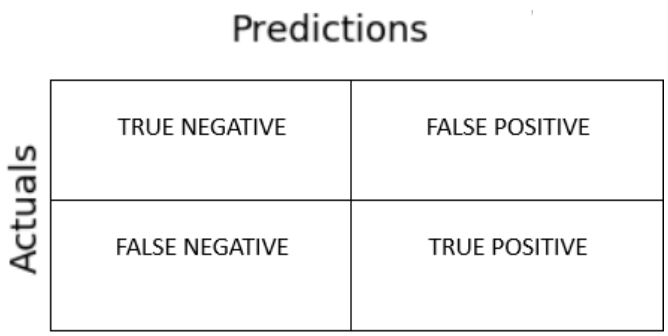
```
Accuracy of the Decision Tree Classifier:  0.797
Accuracy of the XGBoost Classifier:  0.86
Accuracy of the Random Forest Classifier:  0.867
Accuracy of the SVM classifier:  0.8616666666666667
Accuracy of the Logistic Regression Classifier:  0.8163333333333334
Accuracy of the KNN Classifier:  0.8323333333333334
Accuracy of the GNB Classifier:  0.8356666666666667
Accuracy of the Neural Network Classifier:  0.8586666666666667
```

Based on the results obtained, the Random Forest Classifier has the highest accuracy with 86.7%, followed closely by the SVM, XGBoost and Neural Network classifiers. The Logistic Regression and Decision Tree classifiers have the lowest accuracy with around 80%. It's worth noting that these results may not be entirely representative of the model's performance, and further evaluation such as cross-validation should be performed to ensure the model's robustness. Overall, the accuracy results suggest that the selected algorithms have the potential to accurately predict bank customer churn.

The variation in accuracy between the different classification algorithms may be due to the differences in their underlying techniques and assumptions. For instance, the Decision Tree and Random Forest classifiers are both tree-based algorithms. XGBoost is a gradient boosting algorithm which can efficiently handle large datasets and can help reduce overfitting. SVM, Logistic Regression, KNN, and GNB classifiers use different techniques for finding the decision boundary between classes.

Accuracy should not be the only metric used to evaluate a model's performance.

Confusion Matrix



(The confusion matrix fields in the “sklearn.metrics” library display the metrics as shown above, that is different from the normal.)

Confusion matrix is a metric used to evaluate the performance of a classification algorithm. It calculates the number of correct and incorrect predictions by comparing the numerical values of the actual and predicted classes. Confusion matrix is typically represented as a 2x2 matrix and has four different cells: true positive (TP) which means correctly predicted positive instances, false positive (FP), true negative (TN), and false negative (FN). Confusion matrix can be used to calculate many different metrics to evaluate the performance of a classification model, such as accuracy, precision, specificity, and F1 score.

Precision measures the percentage of true positives out of all instances predicted as positive. It is a measure of how accurate the positive predictions are. Precision is calculated as:

➔ precision = TP / (TP + FP)

Recall (sensitivity) measures the percentage of true positives out of all actual positive instances. It is a measure of how well the classifier can identify positive instances. Recall is calculated as:

➔ recall = TP / (TP + FN)

F1 score is a weighted harmonic mean of precision and recall, which balances both metrics. It is often used as a single measure of a classifier's performance. F1 score is calculated as:

➔ F1 = 2 * (precision * recall) / (precision + recall)

The "precision", "recall", and "f1" values calculated in the classification_report function that I used are shown in the line of customers who have churned (which are represented as 1).

Decision Tree Algorithm

```
CMDecisionTree = confusion_matrix(Y_test,DTpred)
CMDecisionTree
```

```
array([[2095,  321],
       [ 288,  296]], dtype=int64)
```

```
print(classification_report(Y_test, DTpred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      2416
           1       0.48      0.51      0.49       584

    accuracy                           0.80      3000
   macro avg       0.68      0.69      0.68      3000
weighted avg       0.80      0.80      0.80      3000
```

We can obtain the accuracy values calculated above by using the numbers which are seen in the confusion matrix, by calculating "(TP+TN)/full dataset" as "(2095+296)/3000".

Looking at the performance metrics of the Decision Tree algorithm, we see that it does not have a sufficient accuracy value. Also, “precision”, “recall”, and “f1” values calculated for churned customers (which are represented as 1) are also low for a good prediction algorithm.

As a result, the decision tree algorithm does not seem suitable and improvable for use in this dataset and this problem.

XGBoost Algorithm

```
cmXGB = confusion_matrix(Y_test,XGBpred)
cmXGB
```

```
array([[2302,  114],
       [ 306,  278]], dtype=int64)
```

```
print(classification_report(Y_test, XGBpred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      2416
           1       0.71      0.48      0.57       584

    accuracy                           0.86      3000
   macro avg       0.80      0.71      0.74      3000
weighted avg       0.85      0.86      0.85      3000
```

Looking at the performance metrics of the XGBoost algorithm, despite having the third-best accuracy value among the algorithms tested, we see that the recall value calculated for customers who have churned from the bank is quite low.

Random Forest Algorithm

```
cmRF = confusion_matrix(Y_test,RFpred)
cmRF
```

```
array([[2346,   70],
       [ 329,  255]], dtype=int64)
```

```
print(classification_report(Y_test, RFpred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      2416
           1       0.78      0.44      0.56       584

    accuracy                           0.87      3000
   macro avg       0.83      0.70      0.74      3000
weighted avg       0.86      0.87      0.85      3000
```

Looking at the performance metrics of the Random Forest Algorithm despite having the best accuracy value among the algorithms tested, we see that the recall value calculated for customers who have churned from the bank is quite low.

Support Vector Machine (SVM) Algorithm

```
print(classification_report(Y_test, SVMpred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      2416
           1       0.83      0.37      0.51       584

    accuracy                           0.86      3000
   macro avg       0.85      0.67      0.71      3000
weighted avg       0.86      0.86      0.84      3000
```

Looking at the performance metrics of the SVM despite having the second-best accuracy value among the algorithms tested, we see that the recall value calculated for customers who have churned from the bank is very low but the precision value is better than the first two algorithms.

Logistic Regression Algorithm

```
CMLOGISTIC = confusion_matrix(Y_test,LOGISTICpred)
CMLOGISTIC
```

```
array([[2346,   70],
       [ 481,  103]], dtype=int64)
```

```
print(classification_report(Y_test, LOGISTICpred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.97      0.89      2416
           1       0.60      0.18      0.27       584

    accuracy                           0.82      3000
   macro avg       0.71      0.57      0.58      3000
weighted avg       0.78      0.82      0.77      3000
```

Looking at the performance metrics of the Logistic Regression Algorithm, we can see the worst recall and f1-score values among the algorithms tested. So, this algorithm does not seem suitable and improvable for use in this problem with this dataset.

KNN Algorithm

```
CMknn = confusion_matrix(Y_test,KNNpred)
CMknn
```

```
array([[2295,  121],
       [ 382,  202]], dtype=int64)
```

```
print(classification_report(Y_test, KNNpred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      2416
           1       0.63      0.35      0.45       584

    accuracy                           0.83      3000
   macro avg       0.74      0.65      0.67      3000
weighted avg       0.81      0.83      0.81      3000
```

Looking at the performance metrics of the KNN algorithm, we can see the recall and f1-score values are quite low for a good prediction model.

GNB Algorithm

```
CMnb = confusion_matrix(Y_test,GNBpred)
CMnb
```

```
array([[2361,   55],
       [ 438,  146]], dtype=int64)
```

```
print(classification_report(Y_test, GNBpred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.91      2416
           1       0.73      0.25      0.37       584

    accuracy                           0.84      3000
   macro avg       0.78      0.61      0.64      3000
weighted avg       0.82      0.84      0.80      3000
```

Looking at the performance metrics of the GNB algorithm, we can see the recall and f1-score values are quite low for a good prediction model.

Neural Network Algorithm

```
cmNN = confusion_matrix(Y_test,NNpred)
cmNN
```

```
array([[2297,  119],
       [ 305,  279]], dtype=int64)
```

```
print(classification_report(Y_test, NNpred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      2416
           1       0.70      0.48      0.57       584

    accuracy                           0.86      3000
   macro avg       0.79      0.71      0.74      3000
weighted avg       0.85      0.86      0.85      3000
```

Looking at the performance metrics of the Neural Network Algorithm, its recall value is quite low but its f1-score is approximately equal to random forest algorithm's. Also its accuracy value is the fourth one with small differences.

Conclusion

To summarize with a table:

Algorithm	accuracy	precision	recall	f-1
Random Forest	0,87	0,78	0,44	0,56
SVM	0,861	0,83	0,37	0,51
XGBoost	0,860	0,71	0,48	0,57
Neural Network	0,858	0,70	0,48	0,57

No algorithm had any time-related issues during the execution. The algorithms ran in less than a minute.

The aim of the project is to accurately identify churners and prevent customers from leaving the bank by taking various actions. In this direction, as mentioned in the above analysis, the recall value is important for us. That is, the rate of correctly predicting churners among the churned customers is the most important metric. Another important value is precision, which is a metric that calculates the ratio of actually churned in predicted as churned.

When we look at the accuracy values for all four algorithms, there is not much difference. When we look at the F-1 score values, the SVM algorithm will be eliminated as its recall value is the lowest. The F-1 score values of the other three algorithms are very close. The recall value is low in all algorithms, and since having such a low recall value is not desirable, it is more logical to choose the XGBoost algorithm, which has the highest recall value and also has a higher precision value, over the Neural Network algorithm. In a scenario where the bank stated that it could achieve a precision value above 70%, this algorithm is in an applicable position.

Retaining existing customers is more cost-effective than acquiring new ones, and for this reason, understanding why customers decide to leave a company is crucial for businesses, particularly for banks. By preventing churn, banks can develop strategies such as loyalty programs and retention campaigns to retain as many customers as possible and maintain profitability.