customer-churn

November 13, 2023

1 Customer Churn Prediction

1.1 Importing Required Libraries

[24]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import tree
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

1.2 Reading the dataset

[2]:
```
df = pd.read_csv('Churn_Modelling.csv')
df.head()
```

[2]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
```

```
4       2  125510.82              1          1                1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

1.3 Removing the Columns not required for the prediction

[3]:
```
df = df.drop(['RowNumber','CustomerId'], axis=1)
df.head()
```

[3]:
```
    Surname  CreditScore Geography  Gender  Age  Tenure    Balance  \
0  Hargrave          619    France  Female   42       2       0.00
1      Hill          608     Spain  Female   41       1   83807.86
2      Onio          502    France  Female   42       8  159660.80
3      Boni          699    France  Female   39       1       0.00
4  Mitchell          850     Spain  Female   43       2  125510.82

   NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  Exited
0              1          1               1        101348.88       1
1              1          0               1        112542.58       0
2              3          1               0        113931.57       1
3              2          0               0         93826.63       0
4              1          1               1         79084.10       0
```

[4]:
```
df = df.drop(['Surname'], axis=1)
df.head()
```

[4]:
```
   CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0          619    France  Female   42       2       0.00              1
1          608     Spain  Female   41       1   83807.86              1
2          502    France  Female   42       8  159660.80              3
3          699    France  Female   39       1       0.00              2
4          850     Spain  Female   43       2  125510.82              1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
3          0               0         93826.63       0
4          1               1         79084.10       0
```

1.4 Checking for NULL Values

[5]:
```
df.isnull().sum()
```

[5]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

1.5 Handling Categorical Data

[6]:
```
df['Geography'].unique()
```

[6]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

1.5.1 Encoding categorical data

[7]:
```
df = pd.get_dummies(df, columns=['Geography','Gender'], drop_first=True)
df.head()
```

[7]:
```
   CreditScore  Age  Tenure    Balance  NumOfProducts  HasCrCard  \
0          619   42       2       0.00              1          1
1          608   41       1   83807.86              1          0
2          502   42       8  159660.80              3          1
3          699   39       1       0.00              2          0
4          850   43       2  125510.82              1          1

   IsActiveMember  EstimatedSalary  Exited  Geography_Germany  \
0               1        101348.88       1                  0
1               1        112542.58       0                  0
2               0        113931.57       1                  0
3               0         93826.63       0                  0
4               1         79084.10       0                  0

   Geography_Spain  Gender_Male
0                0            0
1                1            0
2                0            0
3                0            0
```

```
4                  1             0
```

1.6 Separating dependent and independent variables

[8]:
```python
X = df.drop(['Exited'], axis=1)
y = df['Exited']
```

1.7 Splitting the data into train and test set

[9]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

1.8 Scaling and Normalizing the data

[10]:
```python
scaler = MinMaxScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
```

2 Model Building and Prediction

3 Logistic Regression

[11]:
```python
logreg = LogisticRegression()

logreg.fit(scaled_X_train,y_train)
```

[11]:
```
LogisticRegression()
```

[12]:
```python
y_pred_log = logreg.predict(scaled_X_test)
y_pred_log
```

[12]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[13]:
```python
confusion_matrix(y_test,y_pred_log)
```

[13]:
```
array([[1550,   57],
       [ 318,   75]], dtype=int64)
```

[14]:
```python
print(classification_report(y_test,y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      1607
           1       0.57      0.19      0.29       393
```

```
    accuracy                           0.81      2000
   macro avg       0.70      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```

[15]:
```
accuracy_score(y_test,y_pred_log)
```

[15]:
```
0.8125
```

3.1 81% is not very good, let's try a different algorithm

4 Decision Tree

[16]:
```
dc = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)
dc.fit(scaled_X_train,y_train)
```

[16]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

[17]:
```
y_pred_dect = dc.predict(scaled_X_test)
```

[18]:
```
confusion_matrix(y_test,y_pred_dect)
```

[18]:
```
array([[1527,   80],
       [ 223,  170]], dtype=int64)
```

[19]:
```
print(classification_report(y_test,y_pred_dect))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      1607
           1       0.68      0.43      0.53       393

    accuracy                           0.85      2000
   macro avg       0.78      0.69      0.72      2000
weighted avg       0.83      0.85      0.83      2000
```

4.1 85% is better, but we can try a different algorithm

[20]:
```
randf = RandomForestClassifier(n_estimators=10,random_state=0)

randf.fit(scaled_X_train,y_train)
```

[20]:
```
RandomForestClassifier(n_estimators=10, random_state=0)
```

[21]:
```
y_pred_randf = randf.predict(scaled_X_test)
```

[22]:
```
confusion_matrix(y_test,y_pred_randf)
```

[22]:
```
array([[1549,   58],
       [ 226,  167]], dtype=int64)
```

[23]:
```
print(classification_report(y_test,y_pred_randf))
```

```
              precision    recall  f1-score   support

           0       0.87      0.96      0.92      1607
           1       0.74      0.42      0.54       393

    accuracy                           0.86      2000
   macro avg       0.81      0.69      0.73      2000
weighted avg       0.85      0.86      0.84      2000
```

4.2 86% is almost the same, Let's try some boosting method

5 Adaboost Classifier

[25]:
```
ada_boost = AdaBoostClassifier(n_estimators=50, random_state=0)

ada_boost.fit(scaled_X_train,y_train)
```

[25]:
```
AdaBoostClassifier(random_state=0)
```

[26]:
```
y_pred_adab = ada_boost.predict(scaled_X_test)
```

[27]:
```
confusion_matrix(y_test,y_pred_adab)
```

[27]:
```
array([[1523,   84],
       [ 201,  192]], dtype=int64)
```

[28]:
```
print(classification_report(y_test,y_pred_adab))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1607
           1       0.70      0.49      0.57       393

    accuracy                           0.86      2000
   macro avg       0.79      0.72      0.74      2000
weighted avg       0.85      0.86      0.85      2000
```

6 Gradient Boosting Classifier

[29]:
```
grad_boost = GradientBoostingClassifier(n_estimators=50,random_state=0)

grad_boost.fit(scaled_X_train,y_train)
```

[29]:
```
GradientBoostingClassifier(n_estimators=50, random_state=0)
```

[30]:
```
y_pred_gradb = grad_boost.predict(scaled_X_test)
```

[31]:
```
confusion_matrix(y_test,y_pred_gradb)
```

[31]:
```
array([[1553,   54],
       [ 215,  178]], dtype=int64)
```

[32]:
```
print(classification_report(y_test,y_pred_gradb))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1607
           1       0.77      0.45      0.57       393

    accuracy                           0.87      2000
   macro avg       0.82      0.71      0.74      2000
weighted avg       0.86      0.87      0.85      2000
```

6.1 So, 86% is the best accuracy we are having with this dataset using machine learning algorithms

[ ]:
```

```