customer-churn-prediction

August 30, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
```

[2]:
```
df = pd.read_csv('/content/customer churn.zip')
```

[3]:
```
df
```

[3]:
```
      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
…           …         …         …         …           …       …   …
9995       9996    15606229   Obijiaku          771    France    Male   39
9996       9997    15569892  Johnstone          516    France    Male   35
9997       9998    15584532        Liu          709    France  Female   36
9998       9999    15682355  Sabbatini          772   Germany    Male   42
9999      10000    15628319     Walker          792    France  Female   28

      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
…        …         …             …         …              …
9995       5       0.00              2          1               0
```

```
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
…                 …     …
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

[4]:
```
df.describe()
```

[4]:
```
           RowNumber    CustomerId   CreditScore           Age        Tenure  \
count    10000.00000  1.000000e+04  10000.000000  10000.000000  10000.000000
mean      5000.50000  1.569094e+07    650.528800     38.921800      5.012800
std       2886.89568  7.193619e+04     96.653299     10.487806      2.892174
min          1.00000  1.556570e+07    350.000000     18.000000      0.000000
25%       2500.75000  1.562853e+07    584.000000     32.000000      3.000000
50%       5000.50000  1.569074e+07    652.000000     37.000000      5.000000
75%       7500.25000  1.575323e+07    718.000000     44.000000      7.000000
max      10000.00000  1.581569e+07    850.000000     92.000000     10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

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

[5]:
```
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
```

[6]:
```
df.duplicated().sum()
```

[6]:
```
0
```

[7]:
```
df.columns
```

[7]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

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

[9]:
```
df['Exited'].value_counts().to_frame()
```

[9]:

	count
Exited	
0	7963
1	2037

[10]:
```
X=df.drop(columns=['Exited'])
```

[11]:
```
Y=df['Exited']
```

[12]:
```
X=X.drop(columns=['Geography'])
```

[13]:
```
X=X.drop(columns=['Surname'])
```

[14]:
```
X=X.drop(columns=['Gender'])
```

[15]:
```
sc=StandardScaler()
X=sc.fit_transform(X)
```

[16]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size= 0.
  ↪2,random_state=42)
```

[17]:
```
lr=LogisticRegression()
```

[18]:
```
gb=GradientBoostingClassifier()
```

[19]:
```
rfc=RandomForestClassifier()
```

[20]:
```
lr.fit(X_train,y_train)
```

[20]:
```
LogisticRegression()
```

[21]:
```python
lr.score(X_train,y_train)
```

[21]: 0.8065

[22]:
```python
y_pred=lr.predict(X_test)
```

[23]:
```python
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.10/dist-packages (from imblearn) (0.12.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn->imblearn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn->imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn)
(1.3.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn->imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn)
(3.5.0)
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

[24]:
```python
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled,y_resampled=smote.fit_resample(X_train,y_train)
```

[25]:
```python
lr.score(X_train,y_train)
```

[25]: 0.8065

[26]:
```python
y_pred=lr.predict(X_test)
```

[27]:
```python
from sklearn.metrics import ␣
↪confusion_matrix,recall_score,precision_score,accuracy_score,f1_score,ConfusionMatrixDispla
```

[28]:
```python
precision_score=(y_test,y_pred)
```

[29]:
```python
recall_score=(y_test,y_pred)
```

[30]:
```python
f1_score(y_test,y_pred)
```

[30]: 0.23412698412698418

[31]:
```
from sklearn.svm import SVC
```

[32]:
```
svc=SVC(kernel='rbf',gamma=2,C=1)
```

[33]:
```
svc.fit(X_train,y_train)
```

[33]: SVC(C=1, gamma=2)

[34]:
```
svc.score(X_train,y_train)
```

[34]: 0.991625

[35]:
```
svc.score(X_test,y_test)
```

[35]: 0.8055

[36]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[37]:
```
knn = KNeighborsClassifier(n_neighbors=5)
```

[38]:
```
knn.fit(X_train,y_train)
```

[38]: KNeighborsClassifier()

[39]:
```
y_pred=knn.predict(X_test)
```

[40]:
```
from sklearn.metrics import precision_score
```

[41]:
```
precision_score(y_test,y_pred)
```

[41]: 0.6351931330472103

[42]:
```
from sklearn.metrics import recall_score
```

[43]:
```
recall_score(y_test,y_pred)
```

[43]: 0.37659033078880405

[43]:
```

```

[44]:
```
from sklearn.metrics import f1_score
```

[45]:
```
f1_score(y_test,y_pred)
```

[45]: 0.47284345047923315

[46]:
```
accuracy_score(y_test,y_pred)
```

[46]: 0.835