Predict_Credit_Card_Approvals

August 15, 2025

[34]:
```
# Import libraries
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
```

[24]:
```
# Load dataset
dataset_url = "https://raw.githubusercontent.com/xiaosx-GlintAI/
 ↪predict-credit-card-approvals/refs/heads/main/cc_approvals.data"
cc_apps = pd.read_csv(dataset_url, header=None)
```

[25]:
```
# Data exploration
# Print first, last any sample data points
print(cc_apps.head())
print(cc_apps.tail())
print(cc_apps.sample(10))

print("\n-----Dataframe Info-----")
cc_apps.info()

print("\n-----Columns-----")
print(cc_apps.columns)
```

```
   0      1      2  3  4  5  6     7  8  9  10 11   12 13
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  g    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  g  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  g  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  g    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  s    0  +
     0      1       2  3  4   5   6     7  8  9  10 11   12 13
685  b  21.08  10.085  y  p   e   h  1.25  f  f   0  g    0  -
```

```
686  a  22.67   0.750  u  g   c   v  2.00  f  t   2  g  394  -
687  a  25.25  13.500  y  p  ff  ff  2.00  f  t   1  g    1  -
688  b  17.92   0.205  u  g  aa   v  0.04  f  f   0  g  750  -
689  b  35.00   3.375  u  g   c   h  8.29  f  f   0  g    0  -
   0      1      2  3  4    5    6      7  8  9   10 11   12 13
606  b  16.17  0.040  u  g   c   v  0.040  f  f   0  g    0  +
651  a  15.83  7.625  u  g   q   v  0.125  f  t   1  g  160  -
19   a  19.17  8.585  u  g  cc   h  0.750  t  t   7  g    0  +
582  b  48.50  4.250  u  g   m   v  0.125  t  f   0  g    0  +
632  a  38.75  1.500  u  g  ff  ff  0.000  f  f   0  g    0  -
196  b  33.17  3.165  y  p   x   v  3.165  t  t   3  g    0  +
123  a  44.17  6.665  u  g   q   v  7.375  t  t   3  g    0  +
572  b  21.92  0.540  y  p   x   v  0.040  t  t   1  g   59  +
63   a  20.42  0.835  u  g   q   v  1.585  t  t   1  g    0  +
379  b  33.58  0.250  u  g   i  bb  4.000  f  f   0  s    0  -

-----Dataframe Info-----
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 14 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       690 non-null    object 
 1   1       690 non-null    object 
 2   2       690 non-null    float64
 3   3       690 non-null    object 
 4   4       690 non-null    object 
 5   5       690 non-null    object 
 6   6       690 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    int64  
 13  13      690 non-null    object 
dtypes: float64(2), int64(2), object(10)
memory usage: 75.6+ KB

-----Columns-----
Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], dtype='int64')
```

[26]:
```
print("\n-----Numerical Columns Summary-----")
print(cc_apps.describe())

print("\n-----Categorical Columns Summary-----")
print(cc_apps.describe(include=['object']))
```

```
-----Numerical Columns Summary-----
                2           7         10              12
count  690.000000  690.000000  690.00000      690.000000
mean     4.758725    2.223406    2.40000     1017.385507
std      4.978163    3.346513    4.86294     5210.102598
min      0.000000    0.000000    0.00000        0.000000
25%      1.000000    0.165000    0.00000        0.000000
50%      2.750000    1.000000    0.00000        5.000000
75%      7.207500    2.625000    3.00000      395.500000
max     28.000000   28.500000   67.00000   100000.000000

-----Categorical Columns Summary-----
         0    1    3    4    5    6    8    9   11   13
count  690  690  690  690  690  690  690  690  690  690
unique   3  350    4    4   15   10    2    2    3    2
top      b    ?    u    g    c    v    t    f    g    -
freq   468   12  519  519  137  399  361  395  625  383
```

[27]:
```
# Data Preprocessing
# Replace '?' with NaN
cc_apps_nanreplaced = cc_apps.replace('?', np.nan)

# Change column type
cc_apps_nanreplaced[1] = cc_apps_nanreplaced[1].astype(float)

print("\n-----Change 1 column type from object to float-----")
print(cc_apps_nanreplaced.describe())
```

```
-----Change 1 column type from object to float-----
                1           2           7         10              12
count  678.000000  690.000000  690.000000  690.00000      690.000000
mean    31.568171    4.758725    2.223406    2.40000     1017.385507
std     11.957862    4.978163    3.346513    4.86294     5210.102598
min     13.750000    0.000000    0.000000    0.00000        0.000000
25%     22.602500    1.000000    0.165000    0.00000        0.000000
50%     28.460000    2.750000    1.000000    0.00000        5.000000
75%     38.230000    7.207500    2.625000    3.00000      395.500000
max     80.250000   28.000000   28.500000   67.00000   100000.000000
```

[33]:

```
# Copy data
cc_apps_imputed = cc_apps_nanreplaced.copy()

# Iterate every column to impute missing value, for categorical column use␣
 ↪most
# frequency value and for numerical column use mean value
for col in cc_apps_imputed.columns:
  if cc_apps_imputed[col].dtypes == 'object':
    cc_apps_imputed[col] = cc_apps_imputed[col].fillna(cc_apps_imputed[col].
 ↪value_counts().index[0])
  else:
    cc_apps_imputed[col] = cc_apps_imputed[col].fillna(cc_apps_imputed[col].
 ↪mean())

print("\n-----Check missing values-----")
print(cc_apps_imputed.isnull().sum())
```

```
-----Check missing values-----
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
dtype: int64
```

[38]:

```
# Dummify the categorical features
cc_apps_encoded = pd.get_dummies(cc_apps_imputed, drop_first=True)

# Extract the last column as target variable
X = cc_apps_encoded.iloc[:, :-1].values
y = cc_apps_encoded.iloc[:, -1].values

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)

# Instantiate StandardScaler and use it to rescale X_train and X_test
scaler = StandardScaler()
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
```

[39]:

```
# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train, y_train)

# Use logreg to predict instances from the training set
y_train_pred = logreg.predict(rescaledX_train)

# Print the confusion matrix of the logreg model
print(confusion_matrix(y_train, y_train_pred))
```

```
[[185  19]
 [ 32 226]]
```

[40]:

```
# Define the grid of values for tol and max_iter
tol = [0.01, 0.001, 0.0001]
max_iter = [100, 150, 200]

# Create a dictionary where tol and max_iter are keys and the lists of their␣
 ↪values are the corresponding values
param_grid = dict(tol=tol, max_iter=max_iter)

# Instantiate GridSearchCV with the required parameters
grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5)

# Fit grid_model to the data
grid_model_result = grid_model.fit(rescaledX_train, y_train)

# Summarize results
best_train_score, best_train_params = grid_model_result.best_score_,␣
 ↪grid_model_result.best_params_
print("Best: %f using %s" % (best_train_score, best_train_params))
```

```
Best: 0.850701 using {'max_iter': 100, 'tol': 0.0001}
```

[41]:

```
# Extract the best model and evaluate it on the test set
best_model = grid_model_result.best_estimator_
best_score = best_model.score(rescaledX_test, y_test)

print("Accuracy of logistic regression classifier: ", best_score)
```

```
Accuracy of logistic regression classifier:  0.8289473684210527
```