bank-loan-1

December 24, 2025

1 BANK LOAN APPROVAL PREDICTION - LOGISTIC REGRESSION PROJECT

Predict whether a bank should approve a customer's loan

Goal:- Predict Approved=1 or Not Approved=0 using customer features

2 Project Covers:

Clean dataset loading

Feature selection

Train/Test split

Scaling

Training(Logistic Regression)

Evaluation (Accuracy, Precision, Recall, F1)

Confusion Matrix + Classification Report

Decision Thresholds

AUG-ROC Curve

Prediction function

3 Import libraries

[42]:
```
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.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import(
    accuracy_score, ## Accuracy: Out of all predictions, how many were correct?
    precision_score,## Precision: Out of alL YES predictions, how many were␣
↪actually YES?
    recall_score,# Recall: Out of all actual YES cases, how many did the model␣
↪find?
    f1_score,# F1 Score: A single score that balances Precision and Recall, # A␣
↪table that shows Correct and wrong predictions (YES/NO)
    confusion_matrix, # A table that shows Correct and wrong predictions (YES/
↪NO)
    classification_report, # Shows Accuracy, Precision, Recall, and F1 together␣
↪in one report
    roc_curve, # A graph that shows how well the model separates YES and NO
    roc_auc_score# A number that tells how good the model is at separating␣
↪classes
)
```

4 LOAD DATASET

[43]:
```
df=pd.read_csv('bank_loan.csv')
print(df.head())
print('\nShape',df.shape)
print('\nInfo',df.info())
print(df.isnull().sum())
```

```
   Age  Income  CreditScore EmploymentStatus  Approved
0   59   40358          812       Unemployed         0
1   49   23267          595       Unemployed         0
2   35  102745          619         Salaried         1
3   63  109588          871    Self-employed         0
4   28   58513          648         Salaried         1

Shape (500, 5)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Age               500 non-null    int64
 1   Income            500 non-null    int64
 2   CreditScore       500 non-null    int64
 3   EmploymentStatus  500 non-null    object
 4   Approved          500 non-null    int64
dtypes: int64(4), object(1)
memory usage: 19.7+ KB

Info None
Age                 0
```

```
Income               0
CreditScore          0
EmploymentStatus     0
Approved             0
dtype: int64
```

[44]:
```
#Count each employment status category
print('\n Employment Status Distribution')
print(df['EmploymentStatus'].value_counts())

#Previous last 10 rows
print('\n Last 10 rows of the dataset:')
print(df.tail(10))
```

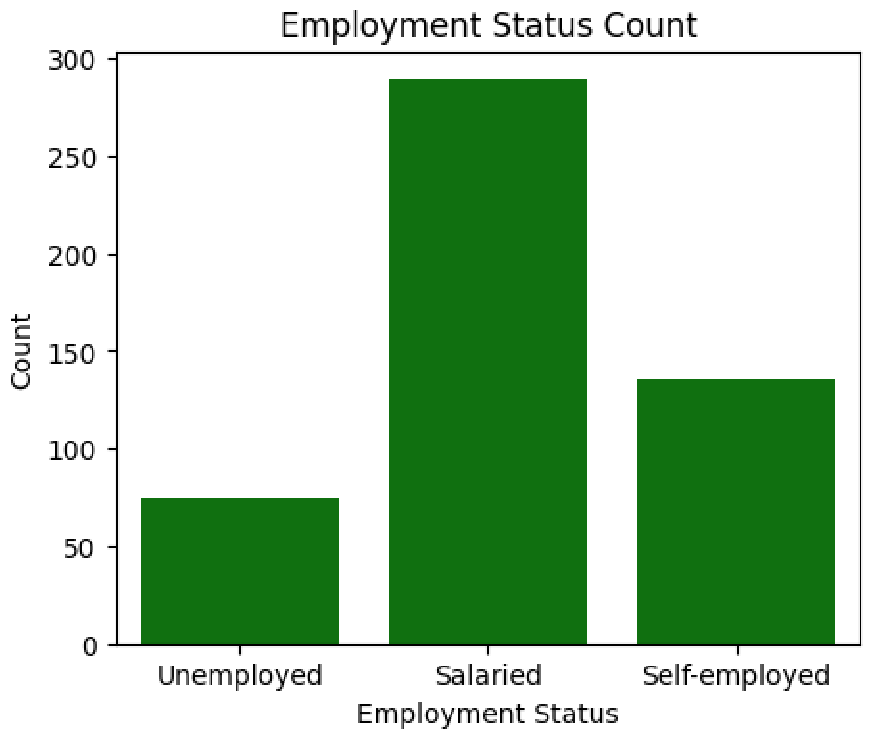
```
 Employment Status Distribution
EmploymentStatus
Salaried         289
Self-employed    136
Unemployed        75
Name: count, dtype: int64

 Last 10 rows of the dataset:
     Age  Income  CreditScore EmploymentStatus  Approved
490   43   32219          870    Self-employed         0
491   37   20235          662    Self-employed         0
492   46   62929          856    Self-employed         0
493   28  147309          644         Salaried         1
494   49   87444          469         Salaried         1
495   46  159639          655         Salaried         1
496   30  121834          490         Salaried         0
497   46  104555          721         Salaried         1
498   54   93698          306       Unemployed         0
499   61  144450          432         Salaried         0
```

[45]:
```
print(df.describe())

print('\n Unique Employment Status')
print(df['EmploymentStatus'].unique())
```

```
              Age         Income  CreditScore    Approved
count  500.000000     500.000000   500.000000  500.000000
mean    43.116000  109045.696000   583.490000    0.502000
std     12.733217   52974.205023   171.614805    0.500497
min     21.000000   20235.000000   300.000000    0.000000
25%     32.000000   62479.250000   440.000000    0.000000
50%     44.000000  109228.500000   585.500000    1.000000
75%     53.000000  156195.000000   723.500000    1.000000
```

```
max     64.000000  199208.000000   898.000000    1.000000

 Unique Employment Status
['Unemployed' 'Salaried' 'Self-employed']
```

[46]:
```
#4.Employment status count Plat
plt.figure(figsize=(5,4))
sns.countplot(x='EmploymentStatus',data=df,color='Green')
plt.title('Employment Status Count')
plt.xlabel('Employment Status')
plt.ylabel('Count')
plt.show()
```



5 Feature Selection And Cleaning

[47]:
```
df=df[['Age','Income','CreditScore','EmploymentStatus','Approved']]
```

6 Encode Employment Status(numeric encoding)

[48]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['EmploymentEncoded']=le.fit_transform(df['EmploymentStatus'])
```

[49]:
```
#Final Feature/Target
X=df[['Age','Income','CreditScore','EmploymentEncoded']]
y=df['Approved']
print('\n final features:\n',X.head())
```

```
 final features:
    Age  Income  CreditScore  EmploymentEncoded
0   59   40358          812                  2
1   49   23267          595                  2
2   35  102745          619                  0
3   63  109588          871                  1
4   28   58513          648                  0
```

7 TRAIN-TEST SPLIT

[50]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪25,random_state=42,stratify=y)
print('train size:',X_train.shape)
print('test size:',X_test.shape)
```

```
train size: (375, 4)
test size: (125, 4)
```

8 Feature Scaling

[51]:
```
Scaler=StandardScaler()
X_train=Scaler.fit_transform(X_train)
X_test=Scaler.transform(X_test)
```

9 Train Logistic Regression Model

[52]:
```
model=LogisticRegression(max_iter=1000)#Improves Accuracy.#Create a logistic␣
 ↪regression model
model.fit(X_train,y_train)#Train the model using scaled training data and their␣
 ↪correct labels
print('Model Training Complete')
```

```
Model Training Complete
```

10 Model Predictions

[53]:
```
y_pred=model.predict(X_test)   #Predicts class labels(0 or 1)
y_prob=model.predict_proba(X_test)[:,1]  #Get probability of class 1(Loan␣
↪Approved) for each test sample
```

11 Evaluation Metrics

[54]:
```
print("\n-------------MODEL EVALUATION-----------------")
print("\nAccuracy Score:",accuracy_score(y_test,y_pred)) #How many total␣
↪predictions are correct
print("\nPrecision Score:",precision_score(y_test,y_pred)) #Out of all␣
↪predicted YES, how many were actually Yes(Controls False Positive)
print("\nRecall Score:",recall_score(y_test,y_pred)) #Out of all actual YES,how␣
↪many the model correctly found (Controls False Positive)
print("\nF1 Score:",f1_score(y_test,y_pred)) #Balance of precision and recall␣
↪in one score
```

```
-------------MODEL EVALUATION-----------------

Accuracy Score: 0.624

Precision Score: 0.6212121212121212

Recall Score: 0.6507936507936508

F1 Score: 0.6356589147286822
```

12 Model Evalution

Accuracy = 0.62 (62%) Out of 100 loan predictions, about 62 are correct.

Precision = 0.62 (62%) When the model says "YES - loan approved", it is correct 62 times out of 100.

Recall = 0.65 (65%) Out of all people who should get the loan, the model correctly finds 65% of them.

F1 Score = 0.63(63%) Overall balanced performance between Precision and Recall.

13 Confusion Matrix + Classification Report

[55]:
```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test,y_pred))
#Shows TP,Fp,TN,FN to understand correct and incorrect predictions
```

```
Confusion Matrix:
[[37 25]
 [22 41]]
```

14 Confusion Matrix

37 → Correctly predicted NO

25 → Wrongly predicted YES (but actually NO)

22 → Wrongly predicted NO (but actually YES)

41 → Correctly predicted YES

```
[56]: print("\nClassification Report:")
      print(classification_report(y_test,y_pred))
      #Shows Precision,Recall,F1 score and support for each class
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.60      0.61        62
           1       0.62      0.65      0.64        63

    accuracy                           0.62       125
   macro avg       0.62      0.62      0.62       125
weighted avg       0.62      0.62      0.62       125
```

15 Class 0 (NO / Loan Rejected)

Precision 63% → When model says NO, it is correct 63 times out of 100

Recall 60% → Found 60% of actual NO cases

F1-score 61% → Balanced score for NO class

Support 62 → Total 62 NO records

16 Class 1 (YES / Loan Approved)

Precision 62% → When model says YES, it is correct 62 times out of 100

Recall 65% → Found 65% of actual YES cases F1-score 64% → Balanced score for YES class

Support 63 → Total 63 YES records

Accuracy = 62% → Model predicted 62 correct out of 100

Macro Avg → Simple average of both classes

Weighted Avg → Average based on number of records

17 DECISION THRESHOLD ANALYSIS

Decision Threshold Analysis means deciding at what probability value the model should say YES or NO.

=>0.5 -> Yes

< 0.5 -> No

[57]:
```
threshold=0.5;#Equal importance to YES and NO / Balanced datasets

print("\n Default Threshold=",threshold)

print("Prediction Probabilities(First 10):")

print(y_prob[:10]); #Shows the first 10 probability values predict by the model
```

```
 Default Threshold= 0.5
Prediction Probabilities(First 10):
[0.32716288 0.68291553 0.52806798 0.675654   0.38418969 0.71675992
 0.69488846 0.63420475 0.59271349 0.5549581 ]
```

18 AUC-ROC Curve

AUC_ROC tells us how well a model can separate positive and negative cases.

It measures how good the model is at distinguishing YES vs NO.

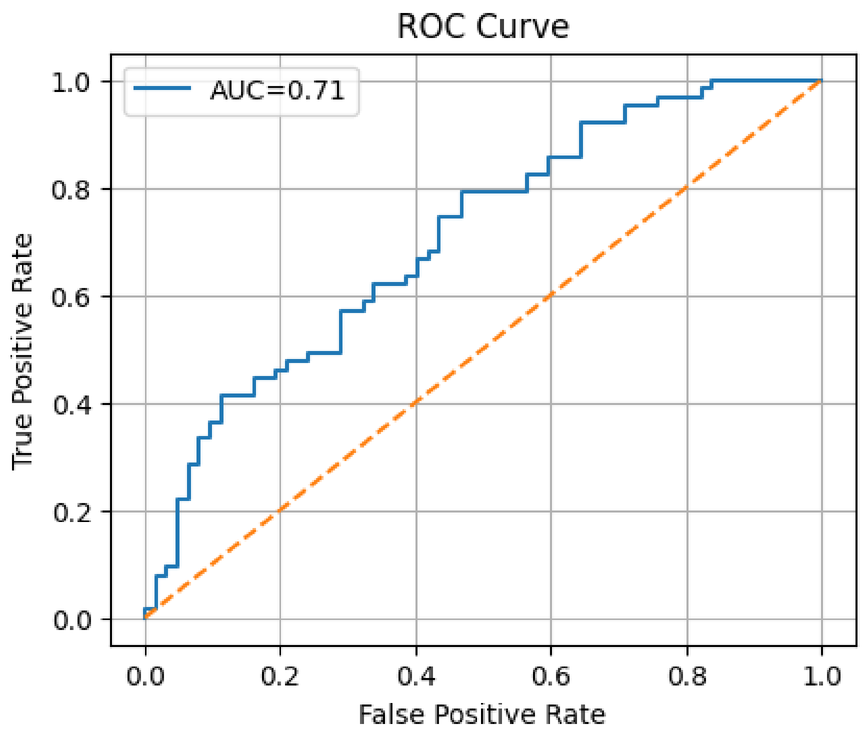
[58]:
```
# AUC tells how well the model separates Yes vs No.

# Step 1: Calculate False Positive Rate, True Positive Rate and Thresholds
fpr, tpr, thresholds = roc_curve(y_test,y_prob)
# fpr = how many wrong positives at each threshold
# tpr = how many correct positives at each threshold
# thresholds = different probability cut-off values

# Step 2: Calculate AUC score
auc = roc_auc_score(y_test,y_prob)
# AUC tells how well the model separates class 0 and class 1
# Higher AUC = better model
print(" \nAUC Score:", auc)
```

```
AUC Score: 0.7073732718894009
```

19 AUC Score = 0.71

Means 71% chance the model will correctly separates Yes vs No

```
[59]: plt.figure(figsize=(5,4))
plt.plot(fpr,tpr,label=f"AUC={auc:.2f}")
#PLot the ROC CURVE

plt.plot([0,1],[0,1],'--') #Draws a random guessing line / Used for comparison
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```



20 PREDICTION FUNCTION

```
[60]: mapping = {'Salaried': 0, 'Self-employed': 1, 'Unemployed': 2}

def predict_loan(age,income,credit,employment):
  #1. Put the customer data into the DataFrame
  row=pd.
 ↪DataFrame([[age,income,credit,mapping[employment]]],columns=['Age','Income','CreditScore','

  #2. Scale the data(same scaling used during training)
  row_scaled=Scaler.transform(row);

  #3. Get probability of loan approved
  prob=model.predict_proba(row_scaled)[0][1]

  #4.Final prediction (0 or 1) using threshold 0.5
  pred=1 if prob >= 0.5 else 0;

  #5. Print Simple Output
  print(f"\n Loan Approval Probability={prob:.2f}")

  if pred == 1:
    print('Loan Approved')
  else:
    print('Loan Not Approved')

predict_loan(35,190000,800,'Salaried')
predict_loan(55,30000,600,'Unemployed')
```

```
 Loan Approval Probability=0.88
Loan Approved

 Loan Approval Probability=0.17
Loan Not Approved
```