code

September 3, 2023

0.1 Assignment 1

0.1.1 Name: Bannuru Rohit Kumar Reddy

0.1.2 Roll Number: 21CS30011

[48]:
```
# import all the necessary libraries here
import pandas as pd

import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
```

[49]:
```
df = pd.read_csv('../../dataset/cross-validation.csv')
print(df.shape)
```

```
(614, 13)
```

Analysing the data

[50]:
```
# Printing the basic information about the data

print(df.head())
print(df.dtypes)
print(df.describe())
```

```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0
```

```
   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

Check for Missing Values

[51]:
```
df.isnull().sum()
```

[51]:
```
Loan_ID               0
Gender               13
Married               3
```

```
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Since the number of missing values is not too large, We will drop all the rows which have missing values

[52]:
```
df = df.dropna()
print(df.shape)
```

```
(480, 13)
```

Separating the data into the input feautres and the output feature Note : We are removing the column 'Loan_ID' from the set of input features as the id will not affect in the prediction of loan status

[53]:
```
X_df = df.drop(['Loan_Status','Loan_ID'], axis=1)
X_df_withloanID = df.drop(['Loan_Status'], axis=1)
y_df = df['Loan_Status']

print(X_df.shape)
print(y_df.shape)
```

```
(480, 11)
(480,)
```

Dealing with categorical data We will first have to convert the categorical data into numerical columns before we can train our model, as the model can only take numbers as input

[54]:
```
# Changing the output variable into categories :

y_df = y_df.replace('N', 0)  # Replacing N with 0
y_df = y_df.replace('Y', 1)  # Replacing Y with 0

y_df.head()

# Changing the input feature variables into categories :

# Columns to be encoded into integers :  Gender, Married, Education,␣
 ↪Self_Employed, Property_Area
```

```
# Changing gender column to 0 and 1 here
X_df['Gender'] = X_df['Gender'].apply(lambda x: 0 if x == 'Male' else 1)

# Changing married column to 0 and 1 here
X_df['Married'] = X_df['Married'].apply(lambda x: 1 if x == 'Yes' else 0)

# Changing education column to 0 and 1 here
X_df['Education'] = X_df['Education'].apply(lambda x: 0 if x == 'Graduate' else␣
↪1)

# Changing self employed column to 0 and 1 here
X_df['Self_Employed'] = X_df['Self_Employed'].apply(lambda x: 0 if x == 'Yes'␣
↪else 1)

# encoding the property area column here
X_final = pd.get_dummies(X_df, columns=['Property_Area'])

# Preprocess the 'Dependents' column to convert '3+' to a numeric value
X_final['Dependents'] = X_final['Dependents'].replace('3+', 3).astype(float)
```

Applying the Standard Scalar on the numerical data

[55]:
```
scaler = StandardScaler()
X_final[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
↪'Loan_Amount_Term', 'Credit_History', 'Dependents']] = scaler.
↪fit_transform(X_final[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
↪'Loan_Amount_Term', 'Credit_History', 'Dependents']])
X_final.head()
```

[55]:
```
   Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
1       0        1    0.218599          0              1        -0.137970
2       0        1   -0.762033          0              0        -0.417536
3       0        1   -0.762033          1              1        -0.491180
4       0        0   -0.762033          0              1         0.112280
5       0        1    1.199231          0              0         0.009319

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1          -0.027952   -0.208089          0.275542        0.413197
2          -0.604633   -0.979001          0.275542        0.413197
3           0.297100   -0.307562          0.275542        0.413197
4          -0.604633   -0.046446          0.275542        0.413197
5           0.999978    1.520245          0.275542        0.413197

   Property_Area_Rural  Property_Area_Semiurban  Property_Area_Urban
1                 True                    False                False
2                False                    False                 True
```

```
3                 False                        False                    True
4                 False                        False                    True
5                 False                        False                    True
```

[56]:

```
import numpy as np

# Create an array of shuffled indexes
data_indexes = np.arange(len(X_final))
np.random.shuffle(data_indexes)

# Define the number of folds for cross-validation
num_folds = 5

# Calculate the size of each fold
fold_size = len(data_indexes) // num_folds

# Initialize lists to store evaluation metrics
fold_accuracies = []
fold_precisions = []
fold_recalls = []

# Perform k-fold cross-validation
for fold_num in range(num_folds):
    # Determine the current fold's start and end indexes
    start_idx = fold_num * fold_size
    end_idx = (fold_num + 1) * fold_size if fold_num < (num_folds - 1) else␣
 ↪len(data_indexes)

    # Extract the current fold's indexes
    current_fold_indexes = data_indexes[start_idx:end_idx]

    # Create training and testing sets based on the fold indexes
    training_indexes = [idx for idx in data_indexes if idx not in␣
 ↪current_fold_indexes]
    X_train, y_train = X_final.iloc[training_indexes], y_df.
 ↪iloc[training_indexes]
    X_test, y_test = X_final.iloc[current_fold_indexes], y_df.
 ↪iloc[current_fold_indexes]

    # Train your model (replace this with your model training code)
    # For example, you can use a Logistic Regression model
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics (replace this with your evaluation code)
```

```
    # For example, you can calculate accuracy, precision, and recall
    fold_accuracy = np.mean(y_pred == y_test)
    fold_precision = np.sum((y_pred == 1) & (y_test == 1)) / np.sum(y_pred == 1)
    fold_recall = np.sum((y_pred == 1) & (y_test == 1)) / np.sum(y_test == 1)

    # Append evaluation metrics to the respective lists
    fold_accuracies.append(fold_accuracy)
    fold_precisions.append(fold_precision)
    fold_recalls.append(fold_recall)

# Calculate and print the mean evaluation metrics across all folds
mean_fold_accuracy = np.mean(fold_accuracies)
mean_fold_precision = np.mean(fold_precisions)
mean_fold_recall = np.mean(fold_recalls)

# Print the mean evaluation metrics
print(f"Mean Accuracy: {mean_fold_accuracy:.4f}")
print(f"Mean Precision: {mean_fold_precision:.4f}")
print(f"Mean Recall: {mean_fold_recall:.4f}")
```

```
Mean Accuracy: 0.8021
Mean Precision: 0.7926
Mean Recall: 0.9675
```

[ ]:

```

```