Admission Prediction using LDA

Objective:

The goal of this analysis is to predict whether a student will get admission based on several factors, using a technique called Linear Discriminant Analysis (LDA).

STEP: 1

Data Description:

The dataset includes information about students who applied for admission, including:

- GRE Score: Standardized test score for **GRE (Graduate Record Examination).**

  # Higher scores positively impact the chances of admission.

- GPA Score: **GPA (Grade Point Average)** reflects the student's academic performance during their undergraduate studies.

  # higher GPA usually increases the admission probability.

- University Rating: Rank of the university.
- Admission Status: Whether the student was admitted (0: No, 1: Yes).

STEP: 2

Data Preprocessing:

1. Checked for missing values and found none, ensuring data completeness.
2. Define X as the independent variable (gpa score, gre score, rank) and Y as the dependent variable (admit).
3. Split the data into training (70%) and testing (30%) sets.
4. Scaled the numerical values using “StandardScaler” to bring all features to a common scale.

Key Code Snippet (Data Preprocessing):

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Splitting the data into training and testing sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=21)

# Standardizing the features for LDA
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step: 3

Model Building:

- Used LDA, which is a statistical method that finds a linear combination of features that best separates two classes.
- Trained the model using the training data and made predictions on the test data.

Key Code Snippet (Model Building):

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Building the LDA model
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train_scaled, Y_train)

# Predict on test data for Admitting or not Admitting the Student
LDA_pred = LDA.predict(X_test_scaled)
```

STEP: 4

Model Evaluation:

- Accuracy: 84.67%
- The model correctly predicted admission status in about 85 out of 100 cases.

Confusion Matrix Breakdown:

- **True Positive (TP):** Predicted admission correctly.
- **True Negative (TN):** Predicted no admission correctly.
- **False Positive (FP):** Incorrectly predicted admission.

- **False Negative (FN):** Incorrectly predicted no admission.

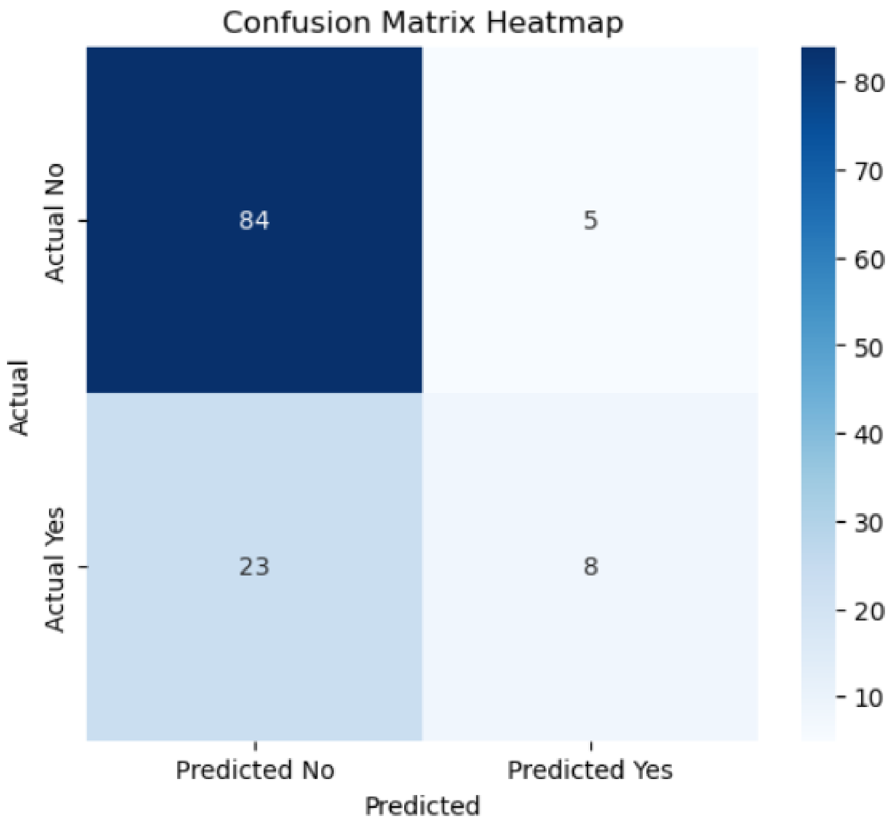
Key Code Snippet (Confusion Matrix):

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Creating the confusion matrix
cm = pd.crosstab(Y_test['admit'], Y_test['pred_admit'])

# Extracting TN, FP, FN, TP from the confusion matrix
tn, fp, fn, tp = cm.to_numpy().ravel()
print('True Positive:', tp)
print('True Negative:', tn)
print('False Positive:', fp)
print('False Negative:', fn)
```

STEP: 5

Visualization:

- Confusion Matrix Heatmap:
    - Visual representation of model performance, showing TP, TN, FP, and FN values.



Confusion Matrix Breakdown:

- **True Negative (TN) = 84:**
    - The model correctly predicted **"No"** (not admitted) for 84 instances.
- **False Positive (FP) = 5:**
    - The model incorrectly predicted **"Yes"** (admitted) when it was actually **"No"** (not admitted).
- **False Negative (FN) = 23:**
    - The model incorrectly predicted **"No"** (not admitted) when it was actually **"Yes"** (admitted).
- **True Positive (TP) = 8:**
    - The model correctly predicted **"Yes"** (admitted) for 8 instances.

Key Code Snippet (Visualization):

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Confusion matrix values
tp, tn, fp, fn = 8, 84, 5, 23

# Create a confusion matrix array
conf_matrix = np.array([[tn, fp], [fn, tp]])

# Plotting the heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True,
            xticklabels=['Predicted No', 'Predicted Yes'],
            yticklabels=['Actual No', 'Actual Yes'])

# Adding titles and labels
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# Display the heatmap
plt.show()
```

STEP: 6

Accuracy:

- **Accuracy Calculation Interpretation:**

  To calculate the accuracy of the model, we use the following formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Explanation:

- **True Positive (TP):** Correctly predicted positive cases (admitted).
- **True Negative (TN):** Correctly predicted negative cases (not admitted).
- **False Positive (FP):** Incorrectly predicted positive cases (predicted admitted, but actually not admitted).
- **False Negative (FN):** Incorrectly predicted negative cases (predicted not admitted, but actually admitted).

Key Code Snippet (Accuracy):

```
# Calculating accuracy manually using the confusion matrix values
accuracy = (tp + tn) / (tp + tn + fp + fn)
accuracy
```

Conclusion:

The accuracy of the model is **76.7%**, meaning the model correctly predicts the admission status in approximately **76.7% of cases**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay,precision_score,recall_score,confusion_matrix

# import "Binary.csv" Dataset
df = pd.read_csv("D:/binary.csv")
print("no. of rows = ", df.shape[0],"\nno. of columns = ", df.shape[1])
df.head()

# Checking for missing values
df.isnull().sum()

# Assigning independent features as X and dependent as Y
X = df.drop('admit', axis = 1)  #Using "gre", "gpa", "rank"
Y = df['admit']

# Splitting the data into training and testing sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=21)

print(X_train.shape, Y_train.shape)   # Should match in row count

# Standardizing the features for LDA
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building the LDA model
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train_scaled, Y_train)

# Predict on test data for Admitting or not Admitting the Student
LDA_pred = LDA.predict(X_test_scaled)
```

```python
# Predict on test data for Admitting or not Admitting the Student
LDA_pred = LDA.predict(X_test_scaled)

# Adding the predictions to the test set as a new column 'pred_admit'
Y_test = Y_test.reset_index(drop=True)
Y_test = pd.DataFrame(Y_test)
Y_test['pred_admit'] = LDA_pred
Y_test.head()

# Creating the confusion matrix
cm = pd.crosstab(Y_test['admit'], Y_test['pred_admit'])
cm

# Extracting TN, FP, FN, TP from the confusion matrix
tn, fp, fn, tp = cm.to_numpy().ravel()

# The Confusion Matrix(cm) is in Dataframe form then we can'nt directly use "tn, fp, fn, tp = cm.ravel()"
# this code we can use "numpy" before directly calling ravel to convert the dataframe into array first.


print('True Positive:', tp)
print('True Negative:', tn)
print('False Positive:', fp)
print('False Negative:', fn)

# Calculating accuracy manually using the confusion matrix values
accuracy = (tp + tn) / (tp + tn + fp + fn) * 100
accuracy

# Accuracy: 76.67%

# Confusion matrix values
tp, tn, fp, fn = 8, 84, 5, 23

# Create a confusion matrix array
conf_matrix = np.array([[tn, fp], [fn, tp]])
```


```python
# Accuracy: 76.67%

# Confusion matrix values
tp, tn, fp, fn = 8, 84, 5, 23

# Create a confusion matrix array
conf_matrix = np.array([[tn, fp], [fn, tp]])
```


```python
# Plotting the heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True,
            xticklabels=['Predicted No', 'Predicted Yes'],
            yticklabels=['Actual No', 'Actual Yes'])

# Adding titles and labels
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# Display the heatmap
plt.show()
```