lab one

By :**Lijima Shrestha** Roll number : **ACEO79BCT035**

Logistic Regression for Heart Disease Prediction

Dataset Source: https://www.kaggle.com/datasets/neurocipher/heartdisease

This notebook implements the full Machine Learning pipeline using Logistic Regression for binary classification.

Background: AI, ML, DL, and Data Science

- **AI**:Creating systems that mimic human intelligence.
- **ML**:Algorithms that learn patterns from data.
- **DL**:Machine learning using deep neural networks.
- **Data Science**:Extracting insights from data with statistics and ML.
- **Logistic Regression**: Predicts outcomes with two possible classes.

Using the dataset Heart_Disease_Prediction.cvs

- Target variable: **Heart Disease**
- Outcome(In Binary):

```
#Dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report
```

Dependencies Explanation

- **pandas (`pd`)** - For handling and analyzing data tables.
- **numpy (`np`)** - For numerical operations and arrays.

- **matplotlib.pyplot (`plt`)** - For plotting graphs and visualizations.
- **train_test_split** - Split data into training and testing sets.
- **StandardScaler** - Scale numerical features.
- **OneHotEncoder** - Convert categorical features to numbers.
- **ColumnTransformer** - Apply different preprocessing to different columns.
- **Pipeline** - Chain preprocessing and model steps together.
- **LogisticRegression** - Classification model.
- **Metrics** - Evaluate model performance (accuracy, precision, recall, f1, confusion_matrix, classification_report).

Data Retrieval and Collection

```
df = pd.read_csv('Heart_Disease_Prediction.csv')
# Display dataset shape and column names
print("Dataset shape:", df.shape)
print("Columns:", df.columns)

# Show first 5 rows
df.head()
```

```
Dataset shape: (270, 14)
,Columns: Index(['Age', 'Sex', 'Chest pain type', 'BP', 'Cholesterol',
'FBS over 120',
,       'EKG results', 'Max HR', 'Exercise angina', 'ST depression',
,       'Slope of ST', 'Number of vessels fluro', 'Thallium', 'Heart
Disease'],
,      dtype='object')
```

```
   Age  Sex  Chest pain type   BP  Cholesterol  FBS over 120  EKG
results  \
0   70    1                4  130          322             0
2
1   67    0                3  115          564             0
2
2   57    1                2  124          261             0
0
3   64    1                4  128          263             0
0
4   74    0                2  120          269             0
2
```

```
   Max HR  Exercise angina  ST depression  Slope of ST  \
0     109                0            2.4            2
1     160                0            1.6            2
2     141                0            0.3            1
3     105                1            0.2            2
4     121                1            0.2            1

   Number of vessels fluro  Thallium Heart Disease
0                        3         3      Presence
1                        0         7       Absence
2                        0         7      Presence
3                        1         7       Absence
4                        1         3       Absence
```

Load and Inspect Dataset

- Load `Heart_Disease_Prediction.csv` into DataFrame `df`.
- Print dataset shape (`rows, columns`).
- Print column names to see features.
- Show first 5 rows with `df.head()` for a quick preview.

Data Cleaning

```
# Check for missing values
print(df.isnull().sum())

# Remove invalid cholesterol values
df = df[df['Cholesterol'] > 0]
# Fill missing cholesterol values with median (if any)
if df['Cholesterol'].isnull().sum() > 0:
    df['Cholesterol'] =
df['Cholesterol'].fillna(df['Cholesterol'].median())

df['Heart Disease'].unique()
print(df['Heart Disease'].value_counts())
#verify data type
print(df.dtypes)


Age                        0
,Sex                        0
,Chest pain type            0
,BP                         0
```

```
,Cholesterol                0
,FBS over 120               0
,EKG results                0
,Max HR                     0
,Exercise angina            0
,ST depression              0
,Slope of ST                0
,Number of vessels fluro    0
,Thallium                   0
,Heart Disease              0
,dtype: int64
,Heart Disease
,Absence     150
,Presence    120
,Name: count, dtype: int64
,Age                          int64
,Sex                          int64
,Chest pain type              int64
,BP                           int64
,Cholesterol                  int64
,FBS over 120                 int64
,EKG results                  int64
,Max HR                       int64
,Exercise angina              int64
,ST depression              float64
,Slope of ST                  int64
,Number of vessels fluro      int64
,Thallium                     int64
,Heart Disease               object
,dtype: object
```

Data Cleaning and Validation

- Check for missing values in each column using `isnull().sum()`.
- Remove rows where `Cholesterol` has invalid values (less than or equal to 0).
- If any cholesterol values are missing, fill them with the median value.
- Check unique values in the `Heart Disease` column to understand class labels.
- Count the number of samples in each heart disease category.
- Verify the data types of all columns to ensure correctness.

Task 1: Single Feature Logistic Regression (Cholesterol)

```
X1 = df[['Cholesterol']]
y = df['Heart Disease']

X1_train, X1_test, y_train, y_test = train_test_split(
    X1, y, test_size=0.2, random_state=42
)

model1 = LogisticRegression()
model1.fit(X1_train, y_train)

LogisticRegression()
```

Feature Selection and Model Training

- Select `Cholesterol` as the input feature (`X1`).
- Select `Heart Disease` as the target variable (`y`).
- Split the data into training (80%) and testing (20%) sets.
- Create a Logistic Regression model.
- Train the model using the training data.

```
y1_pred = model1.predict(X1_test)
print('Accuracy:', accuracy_score(y_test, y1_pred))
print('Precision:', precision_score(y_test, y1_pred))
print('Recall:', recall_score(y_test, y1_pred))
print('F1 Score:', f1_score(y_test, y1_pred))

print('\nConfusion Matrix:\n', confusion_matrix(y_test, y1_pred))
print('\nClassification Report:\n', classification_report(y_test,
y1_pred))
```

```
Accuracy: 0.6111111111111112
,Precision: 0.5
,Recall: 0.23809523809523808
,F1 Score: 0.3225806451612903
,
,Confusion Matrix:
, [[28  5]
, [16  5]]
,
,Classification Report:
,               precision    recall  f1-score   support
```

```
,
,           0       0.64      0.85      0.73        33
,           1       0.50      0.24      0.32        21
,
,    accuracy                           0.61        54
,   macro avg       0.57      0.54      0.52        54
,weighted avg       0.58      0.61      0.57        54
,
```

Model Evaluation

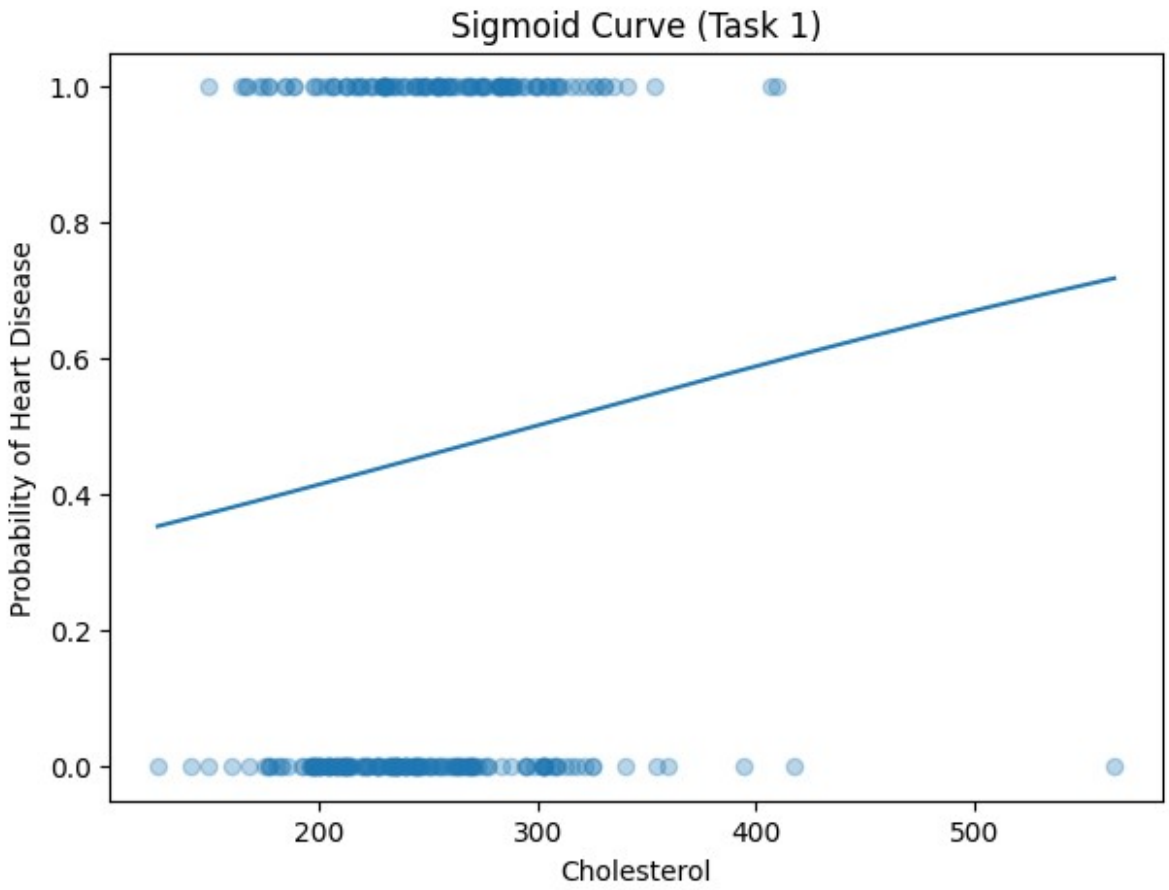
- Use the trained model to predict heart disease on the test data.
- Calculate **Accuracy** to measure overall correctness of predictions.
- Calculate **Precision** to see how many predicted positive cases are correct.
- Calculate **Recall** to measure how well the model identifies actual positive cases.
- Calculate **F1 Score** to balance precision and recall.
- Display the **Confusion Matrix** to show correct and incorrect predictions.

Sigmoid Curve

```
X_range = np.linspace(df['Cholesterol'].min(),
df['Cholesterol'].max(), 300).reshape(-1,1)
probs = model1.predict_proba(X_range)[:,1]

plt.figure(figsize=(7,5))
plt.scatter(df['Cholesterol'], df['Heart Disease'], alpha=0.3)
plt.plot(X_range, probs)
plt.xlabel('Cholesterol')
plt.ylabel('Probability of Heart Disease')
plt.title('Sigmoid Curve (Task 1)')
plt.show()
```

```
c:\Users\pranab\Desktop\ailab1\myenv\Lib\site-packages\sklearn\utils\
validation.py:2691: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
,  warnings.warn(
```



Task 2: Multi-Feature Logistic Regression

```
X2 = df.drop('Heart Disease', axis=1)
y = df['Heart Disease']

cat_cols = ['Sex', 'Chest pain type', 'EKG results', 'Exercise
angina', 'Slope of ST']
num_cols = [c for c in X2.columns if c not in cat_cols]

preprocess = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(drop='first'), cat_cols)
    ]
)

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y, test_size=0.2, random_state=42
)

model2 = Pipeline(steps=[
    ('preprocess', preprocess),
```

```
    ('classifier', LogisticRegression(max_iter=1000))
])

model2.fit(X2_train, y2_train)
```

```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                  ['Age', 'BP',
'Cholesterol',
                                                   'FBS over 120',
'Max HR',
                                                   'ST depression',
                                                   'Number of vessels
fluro',
                                                   'Thallium']),
                                                 ('cat',

OneHotEncoder(drop='first'),
                                                  ['Sex', 'Chest pain
type',
                                                   'EKG results',
                                                   'Exercise angina',
                                                   'Slope of ST'])])),
                ('classifier', LogisticRegression(max_iter=1000))])
```

Multiple Feature Model with Preprocessing

- Select all features except `Heart Disease` as input data (`X2`).
- Set `Heart Disease` as the target variable (`y`).
- Identify categorical and numerical columns.
- Scale numerical features and one-hot encode categorical features.
- Split the dataset into training (80%) and testing (20%) sets.
- Build a pipeline with preprocessing and Logistic Regression model.
- Train the model using the training data.

```
y2_pred = model2.predict(X2_test)

print('Accuracy:', accuracy_score(y2_test, y2_pred))
print('Precision:', precision_score(y2_test, y2_pred))
```

```
print('Recall:', recall_score(y2_test, y2_pred))
print('F1 Score:', f1_score(y2_test, y2_pred))

print('\nConfusion Matrix:\n', confusion_matrix(y2_test, y2_pred))
print('\nClassification Report:\n', classification_report(y2_test,
y2_pred))
```

```
Accuracy: 0.8518518518518519
,Precision: 0.9333333333333333
,Recall: 0.6666666666666666
,F1 Score: 0.7777777777777778
,
,Confusion Matrix:
, [[32  1]
, [ 7 14]]
,
,Classification Report:
,               precision    recall  f1-score   support
,
,           0       0.82      0.97      0.89        33
,           1       0.93      0.67      0.78        21
,
,    accuracy                           0.85        54
,   macro avg       0.88      0.82      0.83        54
,weighted avg       0.86      0.85      0.85        54
,
```

Evaluation of Multi-Feature Logistic Regression Model

- Predict heart disease outcomes using the test dataset.
- Calculate **Accuracy** to measure overall model performance.
- Calculate **Precision** to evaluate correctness of positive predictions.
- Calculate **Recall** to measure how well the model detects heart disease cases.
- Calculate **F1 Score** to balance precision and recall.
- Display the **Confusion Matrix** to show correct and incorrect predictions.
- Show the **Classification Report** with detailed performance metrics.

Questions

- **Which model performs better and why?**
  Answer: Between the two models, the multi-feature logistic regression performs better. This is because it incorporates several relevant predictors simultaneously, allowing the model to capture the combined effect of multiple risk factors associated with heart disease. In contrast, the single-feature model relies only on cholesterol, which limits its predictive capability and overlooks other important variables. As a result, the multi-feature model achieves higher accuracy and overall better predictive performance.
- **How does adding more feature affect accuracy and recll?**
  Answer: Adding more features allows the model to capture more information and underlying patterns in the data. This generally improves both accuracy and recall, as the model can make better predictions and identify positive cases more effectively. However, it's important to ensure the features are relevant; irrelevant or redundant features can sometimes reduce performance.
- **Trade-offs between interpretability and performance**
  Answer:

  -**Single-feature model**: Very easy to interpret and explain, but its predictive performance is limited.

  -**Multi-feature model**: More complex and harder to interpret, but it captures more patterns in the data, leading to better overall performance.

Conclusion

This assignment applied logistic regression to predict heart disease using both a single-feature and a multi-feature approach. The analysis showed that a cholesterol-only model is insufficient for accurate prediction, whereas combining multiple clinical features leads to significantly improved results. The study confirms that logistic regression is an effective baseline model for binary medical classification problems and emphasizes the importance of using comprehensive feature sets in healthcare analytics.