Diabetes Prediction using Logistic Regression

(Machine Learning)

https://www.linkedin.com/in/nibedita-sarkar/

follow

logisticregression-1

June 8, 2023

1 Health Parameter Analysis and Diabetes Prediction using Logistic Regression

Logistic regression is a useful model for predicting binary outcomes, where there are only two possible classes. It is commonly used because it provides interpretable results, allows for estimating probabilities, and is computationally efficient. Logistic regression has fewer assumptions compared to other models, can handle non-linear relationships, and is robust to outliers. It also supports regularization techniques to prevent overfitting. Logistic regression is a well-studied and established model in statistics and machine learning. However, it may not be suitable for all classification problems, especially those with highly nonlinear relationships.

1.1 Define the Problem :

The problem is to create a logistic regression model based on the provided dataset that can predict the outcome of diabetes based on health parameters. The dataset contains various columns, including the outcome column, which represents whether a person has diabetes or not. The goal is to train the model using the training set and evaluate its performance on the testing set using the confusion matrix and accuracy score.

1.2 Importing necessary libraries :

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

1.3 Build the dataset:

To build the dataset, you need to perform the following steps:

a) Load the dataset using pandas: Use the pandas library to load the dataset from the 'diabetes.csv' file

[2]:
```python
data = pd.read_csv("diabetes.csv")
data.head(5)
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

b) Exploring the dataset Exploring the dataset is crucial for understanding its structure, identifying missing values, data distribution, correlations, and outliers, as well as for making informed decisions regarding data preprocessing and feature selection. It provides insights that guide data analysis and model building.

[3]:
```python
data.shape # Checking number of rows and coulmn in the dataset
```

[3]:
```
(768, 9)
```

[4]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

data.info() - Checking the information about the dataset, including the number of entries, the number of columns, column names, data types of each column, and any missing values. It will help in understanding the structure and properties of the dataset.

[5]:
```python
data.describe()
```

[5]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

data.describe() - Checks the descriptive statistics for the numerical columns in the dataset. The statistics include count, mean, standard deviation, minimum value, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum value. It provides a summary of the central tendency, spread, and distribution of the numerical data.

[6]:
```python
data.isnull().sum() # Checking if the dataset contains any null values.
```

[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

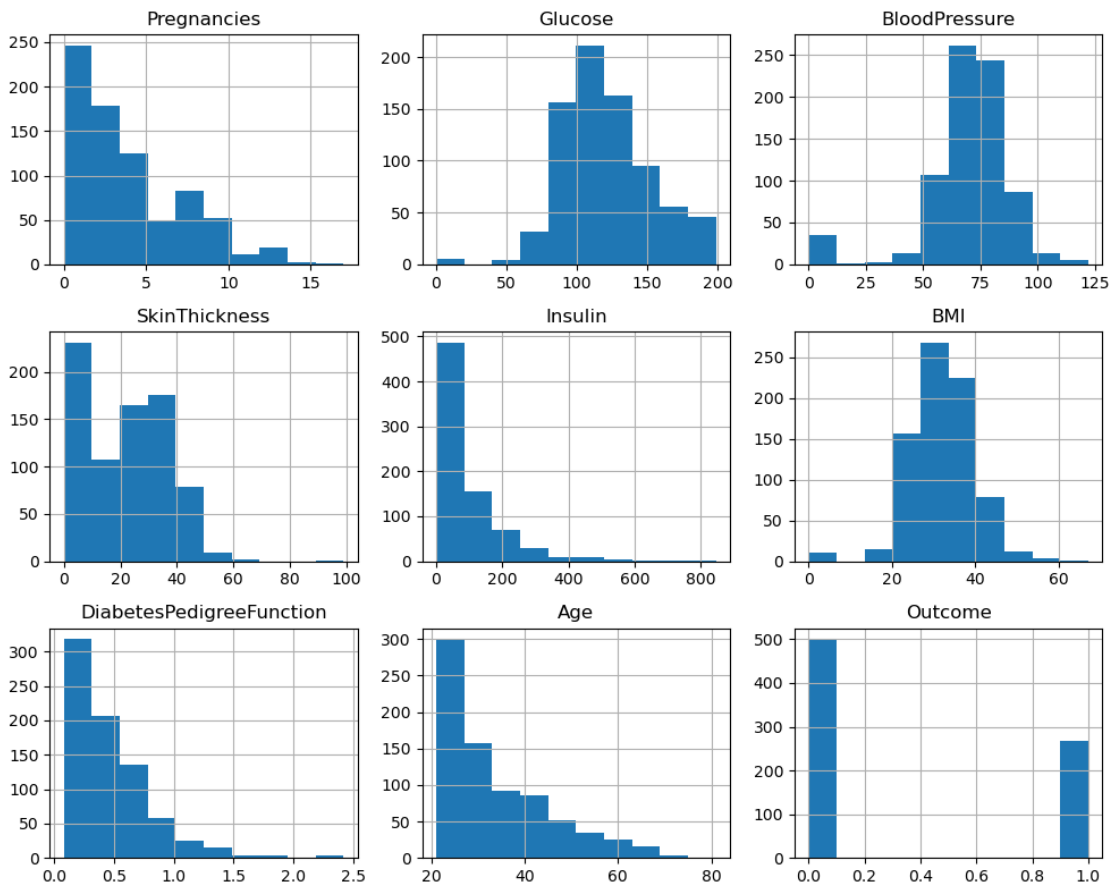
[7]:
```python
num_duplicates = data.duplicated().sum() # Checking if the dataset contains any␣
 ↪duplicate values

if num_duplicates > 0:
    print(f"The dataset contains {num_duplicates} duplicate values")
    data = data.drop_duplicates
    print("Number of duplicate values after dropping:", num_duplicates)
```

```
else:
    print("The dataset doesn't contain any duplicate values.")
```

```
The dataset doesn't contain any duplicate values.
```

[8]:
```
data.hist(figsize=(10, 8)) # Checking Data Distribution
plt.tight_layout()
plt.show()
```



c) Extract data from the outcome column as a variable named Y: Extract the values from the ‘outcome’ column and assign them to a variable called Y.

[9]:
```
X = data.iloc[:,:-1]
X.head(5)
```

[9]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
```

```
3             1       89               66              23        94  28.1
4             0      137               40              35       168  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

d) Extract data from every column except the outcome column as a variable named X: Extract the data from all columns except the 'outcome' column and assign them to a variable called X.

[10]:
```
Y = data.iloc[:,-1]
Y.head(5)
```

[10]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

e) Divide the dataset into two parts for training and testing: Split the dataset into a training set and a testing set in a 70% - 30% proportion. This will be used to train the model on the training set and evaluate its performance on the testing set.

[11]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30,␣
  ↪random_state = 51)
```

[12]:
```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(537, 8)
(231, 8)
(537,)
(231,)
```

1.4 Train the model:

[13]:
```
logistic = LogisticRegression()
logistic.fit(X_train, Y_train)
```

[13]:
```
LogisticRegression()
```

1.5 Evaluate the model :

[14]:
```
Y_predict = logistic.predict(X_test)
print("Y_predict:\n",Y_predict)
```

```
Y_predict:
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1
 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0
 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0
 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1
 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 1 0
 0 0 0 0 1 0 1 0 0]
```

[15]:
```
print("Y_test:\n",Y_test)
```

```
Y_test:
 737    0
505    0
296    1
711    0
329    0
      ..
405    0
315    0
131    1
364    0
322    1
Name: Outcome, Length: 231, dtype: int64
```

[16]:
```
score =  accuracy_score(Y_test, Y_predict)
print("Accuracy Score: ",score * 100)
```

```
Accuracy Score:  79.22077922077922
```

[17]:
```
confusion_matrix = confusion_matrix(Y_test, Y_predict)
print("Confusion Matrix : \n",confusion_matrix)
```

```
Confusion Matrix :
 [[131  11]
 [ 37  52]]
```

1.6 Visually Understanding the performance of the model

[18]:
```
# Create a figure and axis
fig, ax = plt.subplots()

# Plot the actual outcomes
```

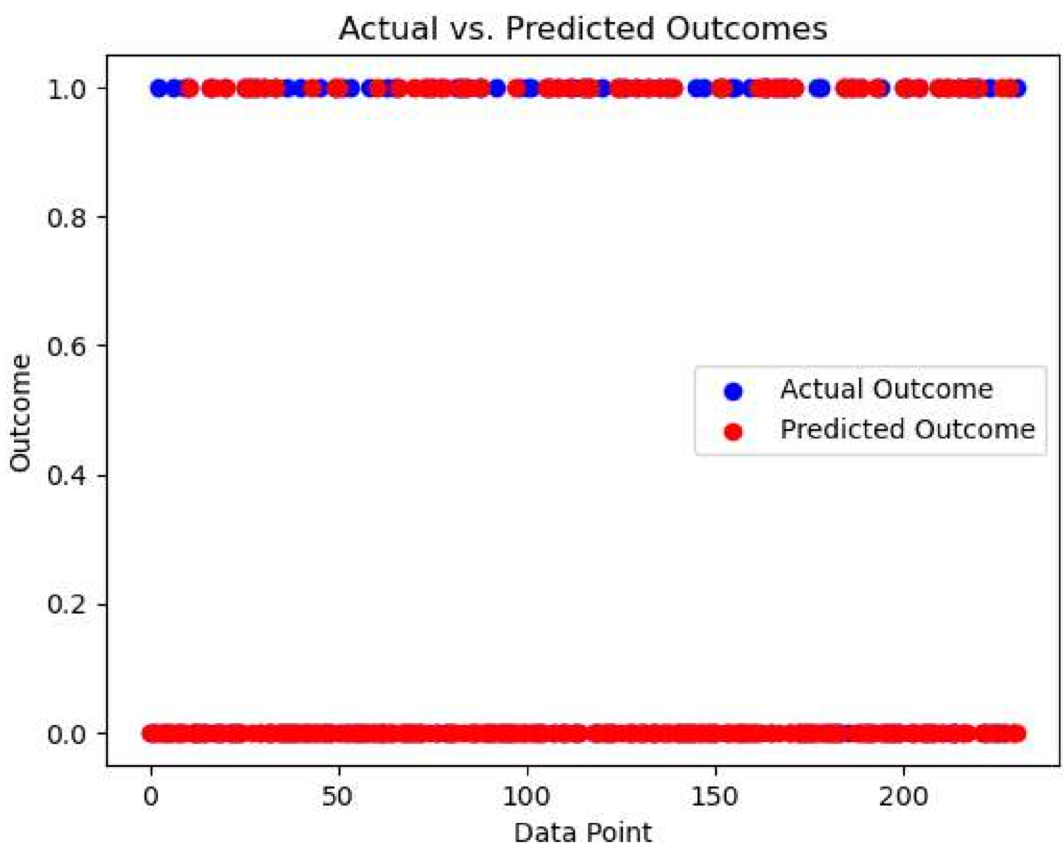
```
ax.scatter(range(len(Y_test)), Y_test, color='blue', label='Actual Outcome')

# Plot the predicted outcomes
ax.scatter(range(len(Y_predict)), Y_predict, color='red', label='Predicted␣
↪Outcome')

# Set axis labels and title
ax.set_xlabel('Data Point')
ax.set_ylabel('Outcome')
ax.set_title('Actual vs. Predicted Outcomes')

# Add a legend
ax.legend()

# Show the plot
plt.show()
```



1.7 Use the model:

Once the model is trained and evaluated, you can use it to make predictions on new, unseen data. This can be done by providing new input values to the model and using the predict function to obtain the predicted outcome.