Logistic Regression for machine Learning

March 10, 2025

[24]:
```
## Loading necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_curve, auc
import warnings
warnings.filterwarnings("ignore")
```

[25]:
```
## Import the dataset
df = pd.read_csv("C:/Users/PC/OneDrive/Desktop/Data Science/Datasets/Datasets/
 ↪Stroke.csv")
```

[26]:
```
## Data Exploration
## View the first view rows of the dataset
df.head()
```

[26]:
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  31112    Male  80.0             0              1          Yes
2  60182  Female  49.0             0              0          Yes
3   1665  Female  79.0             1              0          Yes
4  56669    Male  81.0             0              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1        Private          Rural             105.92  32.5     never smoked
2        Private          Urban             171.23  34.4           smokes
3  Self-employed          Rural             174.12  24.0     never smoked
4        Private          Urban             186.21  29.0  formerly smoked

   stroke
0       1
```

```
1       1
2       1
3       1
4       1
```

[27]:

```
## Check the structure of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4909 entries, 0 to 4908
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 4909 non-null   int64
 1   gender             4909 non-null   object
 2   age                4909 non-null   float64
 3   hypertension       4909 non-null   int64
 4   heart_disease      4909 non-null   int64
 5   ever_married       4909 non-null   object
 6   work_type          4909 non-null   object
 7   Residence_type     4909 non-null   object
 8   avg_glucose_level  4909 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     4909 non-null   object
 11  stroke             4909 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 460.3+ KB
```

[28]:

```
## Check data types
df.dtypes
```

[28]:

```
id                     int64
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

[29]:

```
## Check for duplicates
df.duplicated().sum()
```

```
[29]: 0
```

```
[30]: ## Check for missing values
      df.isnull().sum()
```

```
[30]: id                   0
      gender               0
      age                  0
      hypertension         0
      heart_disease        0
      ever_married         0
      work_type            0
      Residence_type       0
      avg_glucose_level    0
      bmi                  0
      smoking_status       0
      stroke               0
      dtype: int64
```

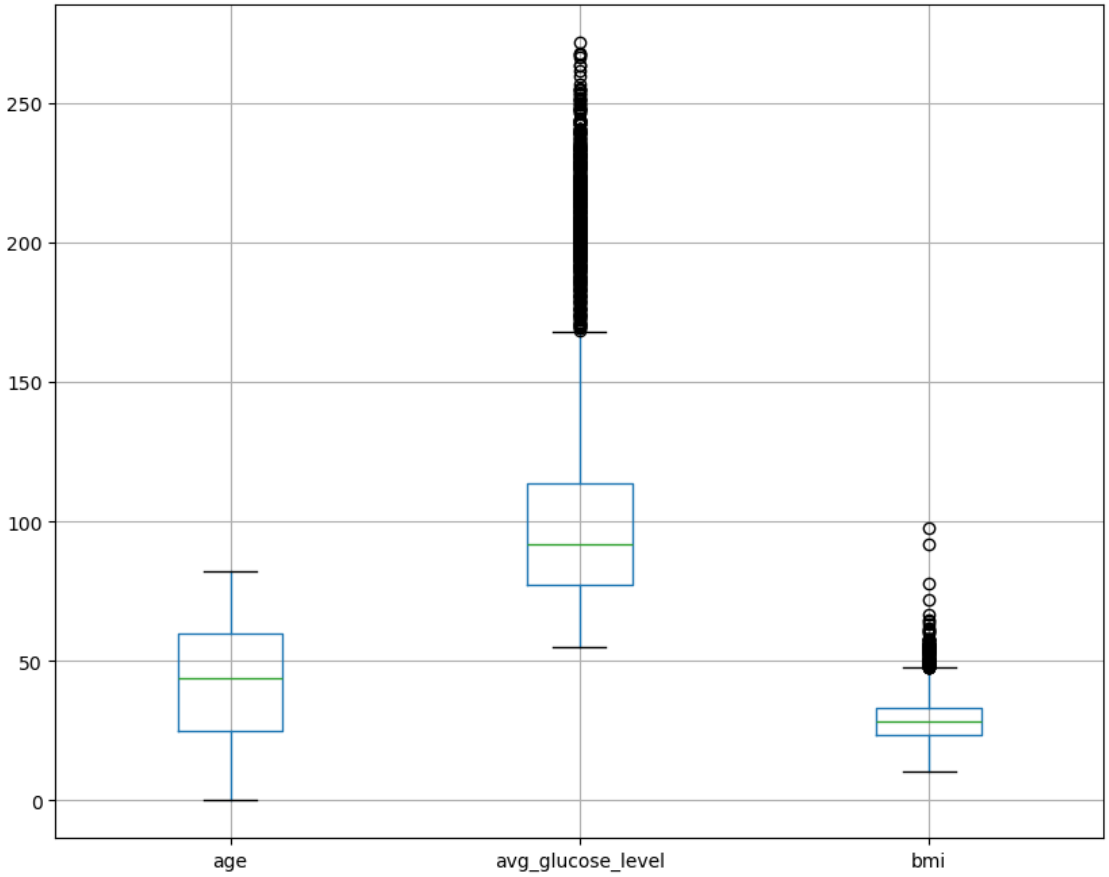
```
[31]: ## Summary statistics
      df.describe()
```

```
[31]:                  id          age  hypertension  heart_disease  \
      count   4909.000000  4909.000000   4909.000000    4909.000000
      mean   37064.313506    42.865374      0.091872       0.049501
      std    20995.098457    22.555115      0.288875       0.216934
      min       77.000000     0.080000      0.000000       0.000000
      25%    18605.000000    25.000000      0.000000       0.000000
      50%    37608.000000    44.000000      0.000000       0.000000
      75%    55220.000000    60.000000      0.000000       0.000000
      max    72940.000000    82.000000      1.000000       1.000000

             avg_glucose_level          bmi       stroke
      count        4909.000000  4909.000000  4909.000000
      mean          105.305150    28.893237     0.042575
      std            44.424341     7.854067     0.201917
      min            55.120000    10.300000     0.000000
      25%            77.070000    23.500000     0.000000
      50%            91.680000    28.100000     0.000000
      75%           113.570000    33.100000     0.000000
      max           271.740000    97.600000     1.000000
```

```
[32]: ## Data Preprocessing
      df["hypertension"]  = df["hypertension"].astype("object")
      df["heart_disease"]  = df["heart_disease"].astype("object")
      df["stroke"]  = df["stroke"].astype("object")
      ## Checking for outliers
```

```
numeric_cols = df.select_dtypes(include = "float64")
numeric_cols.boxplot(figsize = (10, 8))
plt.show()
```



[33]:
```
## Remove outliers
## Average glucose level
Q1 = df["avg_glucose_level"].quantile(0.25)
Q3 = df["avg_glucose_level"].quantile(0.75)
IQR = Q3 - Q1

## Define the lower and upper bound
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

## Remove outliers
df = df[(df["avg_glucose_level"] >= lower_bound) & (df["avg_glucose_level"] <=␣
 ↪upper_bound)]

## bmi
```

```python
Q1 = df["bmi"].quantile(0.25)
Q3 = df["bmi"].quantile(0.75)
IQR = Q3 - Q1

## Define the lower and upper bound
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

## Remove outliers
df = df[(df["bmi"] >= lower_bound) & (df["bmi"] <= upper_bound)]
```

[34]:

```python
## One hot encoding
## Select categorical columns
categorical_cols = df.select_dtypes(include = ["object"]).columns
## Initialize the label encoder
label_encoder = LabelEncoder()
## Apply label encooding to selected columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

[35]:

```python
## Drop study id
df = df.drop(columns = ["id"])
```

[36]:

```python
## Define Features and Target variable
X = df.drop(columns = ["stroke"])
y = df["stroke"]
```

[37]:

```python
## Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

[38]:

```python
## Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

[50]:

```python
## Train the Logistic Regression Model
model = LogisticRegression(class_weight = "balanced")
model.fit(X_train, y_train)
```

[50]:

```
LogisticRegression(class_weight='balanced')
```

[51]:

```python
## Make Predictions
y_pred = model.predict(X_test)
```

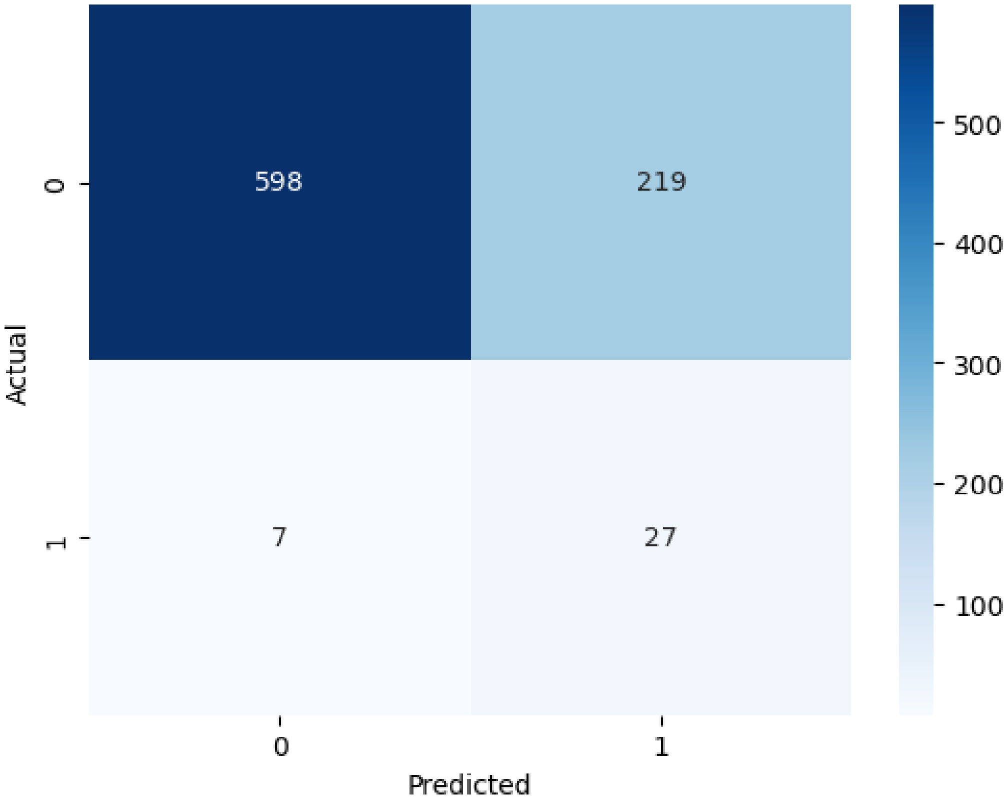
[52]:

```python
## Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
```

```
print(accuracy)
```

```
0.7344300822561692
```

[53]:
```
## Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = "d", cmap = "Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



[54]:
```
## Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.73      0.84       817
           1       0.11      0.79      0.19        34

    accuracy                           0.73       851
   macro avg       0.55      0.76      0.52       851
weighted avg       0.95      0.73      0.82       851
```