

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 2

ML Experiment 2

Aim: To Implement & understand the basics of Logistic Regression.

Theory

Logistic Regression is a supervised ML Algorithm used for Classification task where the goal is to predict the probability that an instance belongs to a given class or not.

It is a Statistical Algo which analyzes the relation between two factors.

Types of Logistic Regression

- Binomial
- Multinomial
- Ordinal

$$Z = WX + b$$

X : Independent inputs

b : bias term (intercept)

W : Weights

Sigmoid function:

The output (Z) is mapped to get the probability

$$\sigma(Z) = \frac{1}{1 + e^{-Z}}$$

logistic regression Equation

$$P(x, b, w) = \frac{1}{1 + e^{-wx+b}}$$

Evaluating Model Methods

- Accuracy
- Precision
- Recall
- F1 Score

~~Conclusion~~ Thus, we implemented logistic Regression

Implementation:

```

from sklearn import linear_model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math

[ ] data = pd.read_csv('dataset/bowling.csv')
x = data[['row', '1']]
y = data[['row', '2']]

[ ] from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)

logr = linear_model.LogisticRegression()
logr.fit(X_train, y_train)
y_pred = logr.predict(X_test)

[ ] from sklearn import metrics

conf_matrix = metrics.confusion_matrix(y_test, y_pred)
conf_matrix

metrics.accuracy_score(y_test, y_pred)

0.78

[ ] import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression

# Sample data (assuming x_test and y_test are numpy arrays)
# x_test = np.array([[200], [100], [150], [100]])
# y_test = np.array([0, 1, 1, 0])

# Check for Datatypes and correct if necessary
if isinstance(X_test, pd.DataFrame):

```

```

X_test = X_test.values.tolist()

# Check for Datatypes and correct if necessary
if isinstance(X_test, pd.DataFrame):
    X_test = X_test.values.tolist() # Assuming you want to use only the first column

# Convert labels to numerical representation if needed
le = LabelEncoder()
y_test = le.fit_transform(y_test)

# Normalize data (optional)
scaler = MinMaxScaler()
X_test_scaled = scaler.fit_transform(X_test.reshape(-1, 1))

# Fit logistic regression model
logr = LogisticRegression()
logr.fit(X_test_scaled, y_test)

# Create Scatterplot
# If = pd.DataFrame({'Study Hours': X_test.flatten(),
#                  'Pass/Fail': y_test,
#                  'Predicted': logr.predict(X_test_scaled)})

# Create scatterplot
# plt.scatter(x='Study Hours', y='Pass/Fail', s=50, label='Actual')

# Generate x-values for prediction
x_values = np.linspace(X_test.min(), X_test.max(), 100).reshape(-1, 1)

# Normalize x-values for prediction (if normalized earlier)
if 'scaler' in locals():
    x_values_scaled = scaler.transform(x_values)
    y_values = logr.predict_proba(x_values_scaled)[0, 1]
else:
    y_values = logr.predict_proba(x_values)[0, 1]

# Plot logistic regression curve
plt.plot(x_values.flatten(), y_values, color='red', label='Logistic Regression Curve')

# Set labels and title
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail')

```

```

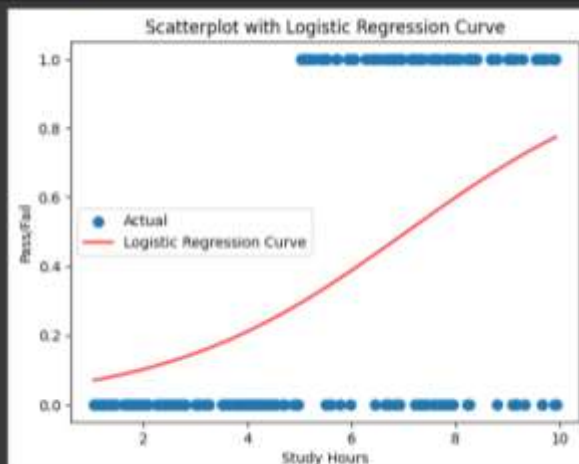
# Plot logistic regression curve
plt.plot(x_values.flatten(), y_values, color='red', label='Logistic Regression Curve')

# Set labels and title
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail')
plt.title('Scatterplot with Logistic Regression Curve')

# Display legend
plt.legend()

# Show the plot
plt.show()

```



```

[ ] w0 = 1
    w1 = 3
    alpha = 0.001

[ ] def find_w(x, w0, w1):
    w = []
    for i in range(len(x)):
        w.append(w0 + w1*x[i])
    return w

[ ] def calculate_jw(x, w, y):
    temp = []
    for i in range(len(x)):
        temp.append((y[i]-1/(1+math.exp(-w[i]))))
    return sum(temp)

[ ] x=data['Study Hours']
    y=data['Pass/Fail']

    for i in range(125):
        w = find_w(x, w0, w1)
        jw = calculate_jw(x, w, y)
        prevw=w
        prevj=jw
        w0 = w0 + alpha*(jw)
        w1 = w1 + alpha*(jw)
        if abs(w0-prevw) or abs(w1-prevj):
            break

[ ] pred=[]
    for i in w:
        print(1/(1+math.exp(-i)),end=" ")
        if(1/(1+math.exp(-i))>0.5):
            pred.append(1)
        else:
            pred.append(0)
    print()

```
