Name – Alamin Seikh

Roll No – 19

Date 11/10/25

Practical – Logistic Regression

Objective - The main objective of what we've been doing is to build a model that can predict whether someone is likely to complete a 50-mile ultra race based on the number of miles they run per week. We've been using logistic regression for this, which is a suitable model for binary classification tasks like this one (completing the race or not completing it).

1. Data Loading and Encoding:

In this step, we load the data into a pandas DataFrame and convert the categorical target variable 'completed_50m_ultra' into numerical form.

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
```

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder

d = {'miles_per_week': [37,39,46,51,88,17,18,20,21,22,23,24,25,27,28,29,30,31,32,33,34,38,40,42,57,68,35,36,41,43,45,47,49,50,52,53,54,55,56,58,59,60,61,63,64,65,66,69,70,72,73,75,76,77,78,80,81,82,83,84,85,86,87,89,91,92,93,95,96,97,98,99,100,101,102,103,104,105,106,107,109,110,111,113,114,115,116,116,118,119,120,121,123,124,126,62,67,74,79,90,112],
'completed_50m_ultra': ['no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','yes','yes','yes','yes','no','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes',]}
df = pd.DataFrame(data=d)

finisfed_race = ['yes','no'] # Define the order of categories
enc = OrdinalEncoder(categories=[finisfed_race]) # Initialize the encoder with the specified order
df['completed_50m_ultra'] = enc.fit_transform(df[['completed_50m_ultra']]) # Apply the encoding

display(df.head()) # Display the first few rows of the modified DataFrame
```

- We import pandas for data manipulation and OrdinalEncoder from sklearn.preprocessing to convert categorical data to numerical.
- We create a dictionary d containing our data.

- We create a pandas DataFrame df from the dictionary.
- We define the desired order for encoding the 'completed_50m_ultra' column in the finished_race list. 'yes' is placed before 'no', so 'yes' will be encoded as 0.0 and 'no' as 1.0.
- We initialize OrdinalEncoder with the specified categories.
- We apply the encoding to the 'completed_50m_ultra' column and update the DataFrame.
- Finally, we display the head of the DataFrame to show the result of the encoding.

The output completed_50m_ultra column now contains 1.0 for 'no' and 0.0 for 'yes'.

2. Data Splitting:

Here, we separate the features (input) from the target variable (output) and split the data into training and testing sets.

```
from sklearn.model_selection import train_test_split



x = df.iloc[:,:1] # Select the 'miles_per_week' column as features

y = df.iloc[:,1] # Select the 'completed_50m_ultra' column as the target

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.8,random_state=11) # Split the data

print("Shape of x_train:", x_train.shape) # Print the shape of the training features

print("Shape of x_test:", x_test.shape)   # Print the shape of the testing features
```

- We import train_test_split from sklearn.model_selection.
- We define x as the feature (miles_per_week) and y as the target (completed_50m_ultra). We use .iloc to select the columns by their integer location.
- We use train_test_split to divide the data into training and testing sets. test_size=0.8 means 80% of the data will be used for testing, and the remaining 20% for training. random_state ensures the split is the same every time we run the code.
- We print the shapes of the training and testing feature sets to verify the split.

The output shows the number of rows and columns in the training and testing feature sets.

3. Model Training:

In this step, we initialize and train the Logistic Regression model using the training data.

```
from sklearn.linear_model import LogisticRegression


model = LogisticRegression() # Initialize the Logistic Regression model
model.fit(x_train,y_train) # Train the model using the training data
```

- We import LogisticRegression from sklearn.linear_model.

- We create an instance of the LogisticRegression model and assign it to the variable model.
- We use the fit() method to train the model. We pass the training features (x_train) and the training target variable (y_train) to the method. The model learns the relationship between the features and the target during this step.

The output confirms that a LogisticRegression() model object has been created and trained.

4. Model Evaluation:

Finally, we evaluate the performance of the trained model on the unseen testing data.

```
from sklearn.metrics import confusion_matrix, classification_report

y_pred = model.predict(x_test) # Make predictions on the testing data

print("Model Score:", model.score(x_test,y_test)) # Print the accuracy score

print("\nConfusion Matrix:\n", confusion_matrix(y_test,y_pred)) # Print the confusion matrix

print("\nClassification Report:\n", classification_report(y_test,y_pred)) # Print the classification
report
```

- We import confusion_matrix and classification_report from sklearn.metrics to evaluate the model's performance.
- We use the predict() method of the trained model to make predictions on the testing features (x_test). The predictions are stored in y_pred.
- We use the score() method to calculate the accuracy of the model on the testing data.
- We generate and print the confusion matrix, which shows the number of true positive, true negative, false positive, and false negative predictions.
- We generate and print the classification report, which provides detailed metrics like precision, recall, and f1-score for each class (completing the race or not).

The output provides the model's accuracy score, the confusion matrix, and the classification report, which helps us understand how well the model performed in predicting race completion.