Experiment No. 3

Expt No. 3	Implementation of Logistic Regression
Date :	

Aim:

To implement and understand the working of Logistic Regression, a statistical method used for binary classification problems in Machine Learning.

System Requirements:

- Operating System: Windows 8 or above / Linux / macOS
- Memory (RAM): Minimum 4 GB
- Processor: Minimum 2.33 GHz (Dual Core or higher)

Software/Tools Required:

Jupyter Notebook / Anaconda Navigator / Google Colaboratory / Spyder
Python 3.x with the following libraries:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn

Purpose of the Experiment:

- To understand how different passenger attributes can be used to predict the likelihood of survival.

- To gain hands-on experience in implementing and evaluating Logistic Regression for binary classification.
- To learn the significance of data preprocessing, feature selection, and model evaluation in predictive modeling.

Expected Outcomes:

- Understand the principles and fundamentals of logistic regression as binary classifier
- Gain insights into how logistic regression fits into the broader landscape of Machine Learning models
- Able to evaluate the performance of logistic regression model using appropriate metrics
- Extend the possibilities to handle multi-class classification problems with logistic regression

Theory:

Logistic Regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable based on one or more predictor variables (independent variables). Unlike linear regression, which predicts continuous outcomes, logistic regression predicts a probability that the dependent variable belongs to a particular category.

Logistic regression uses a logistic function called a sigmoid function to map predictions and their probabilities. The sigmoid function refers to an S-shaped curve that converts any real value to a range between 0 and 1.

If the output of the sigmoid function (estimated probability) is greater than a predefined threshold on the graph, the model predicts that the instance belongs to that class. If the estimated probability is less than the predefined threshold, the model predicts that the instance does not belong to the class.

For binary classification, the outcome is often coded as 0 or 1, where 1 typically represents the presence of an event (e.g., success, yes) and 0 represents its absence (e.g., failure, no).

The sigmoid function is referred to as an activation function for logistic regression and is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Where :

e = base of natural logarithms

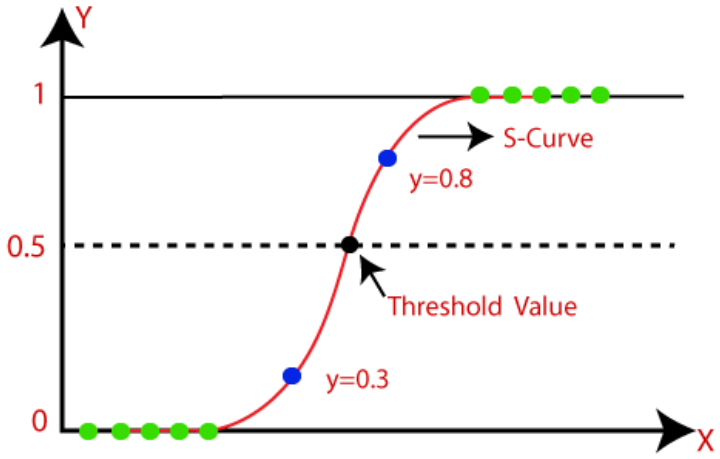
The logistic regression model uses the following logistic function (also known as the sigmoid function) to map predicted values to probabilities:

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Where:

- P(y=1) is the probability that the dependent variable y equals 1
- β_0 is the intercept
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the independent variables $x_1, x_2, \ldots, x_n$

The model aims to find the best-fit coefficients that maximize the likelihood of observing the given data.



Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

1. Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
2. Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
3. Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High"

Real-World Applications of Logistic Regression

Here are four real-world applications of logistic regression:

1. Medical Diagnosis: Predicting the presence or absence of diseases, such as cancer or diabetes, based on patient data (e.g., age, test results, lifestyle factors).
2. Credit Scoring: Assessing the likelihood that a borrower will default on a loan based on financial history, income, and other personal factors.
3. Spam Detection: Classifying emails as spam or not spam by analyzing features like sender, content, and frequency of certain words.
4. Customer Churn Prediction: Predicting whether a customer will leave a service based on usage patterns, customer service interactions, and other factors.

1. Problem Statement

The problem is to predict whether a passenger survived or not on the Titanic. Using various features such as age, sex, passenger class, etc., we aim to build a model that classifies a passenger as either survived (1) or not survived (0).

2. Analytical Approach

This problem is a classification problem, where the dependent variable (Survived) is categorical. The task is to predict whether a passenger survived (1) or did not survive (0) based on various independent features in the dataset.

- Dependent Variable: Survived (binary classification: 0 = not survived, 1 = survived)
- Independent Variables: Age, Sex, Pclass, SibSp, Parch, Fare, Embarked, etc.

The modeling approach will involve:

1. Data cleaning and preprocessing.
2. Feature extraction and encoding (one-hot encoding for categorical variables).
3. Splitting the data into training and testing sets.
4. Applying Logistic Regression as the classification algorithm.

3. Data Collection

The data used for this analysis is from the Titanic dataset, which contains the details of passengers aboard the Titanic. This dataset includes information such as:

1. PassengerId: Unique identifier for each passenger.
2. Survived: Whether the passenger survived (1) or did not survive (0).
3. Pclass: Passenger class (1 = First class, 2 = Second class, 3 = Third class).
4. Name: Passenger name.
5. Sex: Gender of the passenger.
6. Age: Age of the passenger.
7. SibSp: Number of siblings or spouse aboard the Titanic.
8. Parch: Number of parents or children aboard the Titanic.
9. Ticket: Ticket number.
10. Fare: Fare paid by the passenger.
11. Cabin: Cabin number.

12. Embarked: Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

You can download the dataset from the following link:
https://github.com/25-Madhuri/Dataset

Sample Code

Step 1: Import Libraries

 # Importing necessary libraries for data manipulation, visualization, and modeling

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report
```

Step 2 : Load the Titanic dataset

Load the Titanic dataset from a CSV file.

```
df = pd.read_csv('titanic-training-data.csv')
```

 # Display a random sample of 10 rows to understand the structure of the dataset

```
df.sample(10)
```

Step 3: Check Data Structure

Check the data types and general structure of the dataset.

```
# Check the data types of each column

df.dtypes
```

```
# Check detailed information about the DataFrame (columns, non-null count, data types)

df.info()
```

Step 4: Check for Missing Values
Check for missing values in the dataset.

```
# Check for missing values in each column

df.isnull().sum()
```

Step 5: Visualize Target Variable Distribution
Visualize the distribution of the target variable **'Survived'** using a count plot.

```
# Plot the distribution of the target variable 'Survived'

sns.countplot(x='Survived', data=df)

# Plot the distribution of 'Survived' by 'Sex'

sns.countplot(x='Survived', hue='Sex', data=df)
```

Step 6: Crosstab to Analyze Relationship between 'Survived' and 'Sex'

A **crosstab** is useful when you want to compare two categorical variables. In this case, we're using it to compare survival status (Survived) with gender (Sex).

```
# Create a crosstab to analyze the relationship between 'Survived' and 'Sex'

pd.crosstab(df['Survived'], df['Sex'])
```

Step 7: Visualize Target Variable Distribution by Passenger Class

You can visualize the distribution of the target variable **'Survived'** based on the **Pclass** (passenger class) using the **countplot**.

 # Plot the distribution of 'Survived' by 'Pclass' (passenger class)

```
sns.countplot(x='Survived', hue='Pclass', data=df)
```

Step 8: Crosstab to Analyze Relationship between 'Survived' and 'Pclass'

We can use a **crosstab** to analyze the relationship between survival status (Survived) and passenger class (Pclass).

 # Create a crosstab to analyze the relationship between 'Survived' and 'Pclass'

```
pd.crosstab(df['Survived'], df['Pclass'])
```

Step 9: Boxplot to Analyze Age Distribution by Passenger Class

You can visualize the distribution of **'Age'** across different **Pclass** values using a **boxplot**.

 # Plot the distribution of 'Age' across different 'Pclass' values

```
sns.boxplot(x='Pclass', y='Age', data=df)
```

Step 10: Drop the 'Cabin' Column

Since the **'Cabin'** column has many missing values, we will drop it from the dataset. The axis=1 argument specifies that we want to drop a **column**.

 # Drop the 'Cabin' column as it has too many missing values

```
df = df.drop("Cabin", axis=1, errors="ignore")
```

 # Display the first 5 rows of the updated DataFrame

```
df.head()
```

Step 11: Drop Rows with Missing Values

Remove rows that contain missing values and check the new shape of the DataFrame.

```
# Drop rows with missing values

df = df.dropna()

# Display the shape (number of rows and columns) of the updated DataFrame

df.shape
df.head()
```

Step 12: One-Hot Encoding of Categorical Columns

Convert the categorical columns **'Sex'**, **'Embarked'**, and **'Pclass'** into binary values using **one-hot encoding**.

```
# Apply one-hot encoding to the categorical columns: 'Sex', 'Embarked', 'Pclass'

df = pd.get_dummies(columns=['Sex', 'Embarked', 'Pclass'], data=df)

# Display the updated DataFrame with encoded columns

df.head()
```

Step 13: Drop Irrelevant Columns and Check Data Types

Remove the irrelevant columns **'PassengerId'**, **'Name'**, **'Ticket'**, and **'Fare'**, and check the data types of the remaining columns.

```
# Drop irrelevant columns: 'PassengerId', 'Name', 'Ticket', 'Fare'

df = df.drop(["PassengerId", "Name", "Ticket", "Fare"], axis=1, errors="ignore")

# Display the updated DataFrame

df.head()
```

 # Check the data types of the remaining columns

```
df.dtypes
```

Step 13: Split the Data and Train Logistic Regression Model

Define the feature columns (**X**) and target column (**Y**), then split the data into training and testing sets. After that, we initialize and train the **Logistic Regression** model.

 # Define features (X) and target variable (Y)

```
X = df.drop("Survived", axis=1)  # Features (all columns except 'Survived')

Y = df["Survived"]  # Target variable (Survived)
```

 # Split the data into training and test sets (70% train, 30% test)

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1)
```

 # Initialize and train the Logistic Regression model

```
model_1 = LogisticRegression()

model_1.fit(X_train, Y_train)
```

Step 14: Evaluate Model Performance

After training the model, evaluate its performance using the **score** method on both training and testing sets. Additionally, make predictions on the test set.

 # Evaluate the model's performance on the training data

```
train_accuracy = model_1.score(X_train, Y_train)
```

```
# Evaluate the model's performance on the testing data
test_accuracy = model_1.score(X_test, Y_test)
# Print the accuracy results
print(f"Training Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")

# Make predictions on the test set
predictions = model_1.predict(X_test)
```

Step 15: Calculate Accuracy Score and Classification Report

Use **accuracy score** to evaluate the model's predictions and print the **classification report** for more detailed performance metrics.

```
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report
# Calculate the accuracy score
accuracy_score = metrics.accuracy_score(Y_test, predictions)

# Print the accuracy score
print(f"Accuracy Score: {accuracy_score}")
```

```
# Generate and print the classification report

print("Classification Report:")
print(classification_report(Y_test, predictions))
```

Step 16: Visualize Confusion Matrix

A **confusion matrix** helps evaluate the performance of a classification model by comparing the actual versus predicted values. Here, we use **seaborn** to visualize it as a heatmap.

```
# Generate the confusion matrix

cm = metrics.confusion_matrix(Y_test, predictions, labels=[1, 0])

# Create a DataFrame for better visualization

df_cm = pd.DataFrame(cm, index=["1", "0"], columns=["Predict 1", "Predict 0"])


# Plot the confusion matrix as a heatmap

plt.figure(figsize=(7, 5))

sns.heatmap(df_cm, annot=True, fmt="g")

plt.title("Confusion Matrix")

plt.show()
```

Formulas for **Accuracy**, **Precision**, **Recall**, and **F1-score** based on the **confusion matrix**.

1. Accuracy

Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total instances in the dataset.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TPTPTP = True Positives (Predicted 1, Actual 1)
- TNTNTN = True Negatives (Predicted 0, Actual 0)
- FPFPFP = False Positives (Predicted 1, Actual 0)
- FNFNFN = False Negatives (Predicted 0, Actual 1)

2. Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Where:

- TPTPTP = True Positives
- FPFPFP = False Positives

3. Recall (Sensitivity or True Positive Rate)

Recall is the ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where:

- TPTPTP = True Positives
- FNFNFN = False Negatives

4. F1 Score

The F1 Score is the harmonic mean of **Precision** and **Recall**, providing a balance between the two metrics.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Conclusion –

Hence, the model summarizes the findings from the experiment, highlighting the relationship between passenger attributes (such as gender, class, and age) and the likelihood of survival on the Titanic. The logistic regression model effectively predicts survival, with key features like 'Sex' and 'Pclass' significantly influencing the outcomes

Viva Questions

Sr.No	Question	CO
1.	What is Logistic Regression and how does it differ from Linear Regression?	**1ICPC406_2**
2.	When would you use Logistic Regression instead of Linear Regression?	**1ICPC406_2**
3.	Explain the logistic function and how it is used in Logistic Regression?	**1ICPC406_2**
4.	What is the range of the output of a Logistic Regression model and what does it represent?	**1ICPC406_2**

5.	How do you handle the categorical predictors in Logistic Regression?	**1ICPC406_2**
6.	What are some methods to assess the performance of a Logistic Regression model?	**1ICPC406_2**
7.	What is the purpose of using a confusion matrix in the context of Logistic Regression?	**1ICPC406_2**
8.	How can you handle imbalanced datasets in Logistic Regression?	**1ICPC406_2**
9.	What is the difference between binary logistic regression and multinomial logistic regression?	**1ICPC406_2**
10.	State a basic example for Logisitc Regression?	**1ICPC406_2**

References –

a. Textbook –

i. Machine Learning with Python – An approach to Applied ML – Abhishek Vijayvargiya, BPB Publications, 1st Edition 2018

ii. Machine Learning, Tom Mitchell, McGraw Hill Education, 1st Edition 1997

b. Online references –

i. https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic- regression-for-data-science-beginners/

ii. https://www.simplilearn.com/tutorials/machine-learning-tutorial/logistic-regression- in-python

iii. https://www.kaggle.com/code/nargisbegum82/logistic-regression-in-machine- learning