MACHINE LEARNING

LAB ASSIGNMENT-5

NAME – CHAITHANYA GOWDA L

USN – 22BTRAD010

Implementing LogisticRegression algorithm.

CODE:

import pandas as pd

f1 = pd.read\_csv("/content/User\_Data.csv")

f1.head()

f2 = pd.get\_dummies(f1, columns=["Gender"])

f2.head()

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

X\_data = f2.iloc[:,[1,2,4,5]].values

Y\_data = f1.iloc[:, 3].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_data, Y\_data, test\_size = 0.2)

ss = StandardScaler()

x\_train\_ss =ss.fit\_transform(X\_train)

x\_test\_ss = ss.fit\_transform(X\_test)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(f1[["Age","EstimatedSalary"]], f1.Purchased, test\_size = 0.2)

X\_train.size

model = LogisticRegression()

#model.fit(X\_train, Y\_train)

model.fit(x\_train\_ss, Y\_train)

model.predict(x\_test\_ss)

**Code:**

import pandas as pd

**Explanation:**

**import pandas as pd**: This line imports the pandas library into the code. The "as pd" part of the code allows us to refer to the pandas library using the abbreviation "pd" instead of typing out "pandas" each time

**Code:**

f1 = pd.read\_csv("E:/Coding notebook/7th Term/ML/User\_Data.csv")

**Explanation:**

**pd.read\_csv('E:/Coding notebook/7th Term/ML/User\_Data.csv')**: This line uses the pandas library to read the data from a csv file named 'User\_Data.csv'. The data from the csv file is then stored in a pandas DataFrame, which is a 2-dimensional, size-mutable, heterogeneous tabular data structure with labeled axes (rows and columns)

**Code:**

f1.head()

**Explanation:**

The code f1.head() is used to display the first few rows of a pandas DataFrame, in this case 'f1'.

By calling f1.head(), you're essentially asking pandas to show you the first few rows of the DataFrame. This can be helpful when you want to quickly take a look at your data and see if everything looks correct.

**Code:**

f2 = pd.get\_dummies(f1,*columns*=["Gender"])

**Explanation:**

In this code, the pandas function get\_dummies() is being used to convert categorical variable(s) into dummy/indicator variables.

f1 is the pandas DataFrame that was read from the csv file 'User\_Data.csv'. f2 is a new DataFrame that is created by calling get\_dummies() on f1.

The columns parameter is used to specify the columns in the DataFrame that need to be converted. In this case, the "Gender" column is specified.

As a result of this operation, f2 will contain new dummy columns for each unique value in the "Gender" column of f1. For example, if 'Male' and 'Female' are the unique values in the "Gender" column, f2 will contain new columns named 'Gender\_Male' and 'Gender\_Female'. The values in these new columns will be 1 if the corresponding row in f1 has 'Male' or 'Female' in the "Gender" column, and 0 otherwise.

**Code:**

f2.head()

**Explanation:**

The code f2.head() is used to display the first few rows of a pandas DataFrame, in this case 'f2'.

This can be useful for getting a quick look at your data and checking if everything looks correct before moving on to the next step in your data analysis

**Code:**

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

**Explanation:**

**StandardScaler**: The StandardScaler module in sklearn.preprocessing is used to standardize features by removing the mean and scaling to unit variance.

This module is important because many machine learning algorithms, especially those involving a large number of features, may perform poorly if the features are not standardized. Standardization helps these algorithms to work correctly and improves their performance.

**train\_test\_split**: The train\_test\_split module in sklearn.model\_selection is used to split the dataset into training and testing subsets.

This module is important because it is crucial to evaluate the performance of a machine learning model on unseen data. Therefore, splitting the dataset into a training set (used to train the model) and a testing set (used to evaluate the model) is a basic step in the machine learning pipeline.

**LogisticRegression**: The LogisticRegression module in sklearn.linear\_model is a class for logistic regression.

This module is important because logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. It is widely used in binary classification problems, such as predicting whether a customer will make a purchase or not.

In summary, the three modules, StandardScaler, train\_test\_split, and LogisticRegression, are essential building blocks for implementing machine learning models in sklearn. They help in standardizing features, splitting datasets into training and testing subsets, and implementing logistic regression models, respectively.

**Code:**

X\_data = f2.iloc[:,[1,2,4,5]].values

Y\_data = f1.iloc[:, 3].values

**Explanation:**

Feature Extraction: This line of code extracts features from the f2 dataframe by selecting only specific columns. Here, f2.iloc[:,[1,2,4,5]] means to select all rows but only the 1st, 2nd, 4th, and 5th columns from f2. The .values attribute at the end is used to get the numpy array of the selected features.

Target Variable: This line of code extracts the target variable (or label) from the f1 dataframe. Here, f1.iloc[:, 3] means to select all rows but only the 3rd column from f1. The .values attribute at the end is used to get the numpy array of the target variable.

So, the overall goal of this code is to separate the features from the target variable from the original datasets. This is a common practice in machine learning where you have a set of features and a target variable, and you want to use the features to predict the target variable.

**Code:**

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_data, Y\_data, *test\_size* = 0.2)

**Explanation:**

**Train-Test Split**: The line of code performs a train-test split of the input features and target variable.

The function train\_test\_split() splits the specified input data into train and test subsets. It takes the following parameters:

**X\_data**: This is a pandas dataframe that contains the features to be used for prediction.

**Y\_data**: This is a pandas series that contains the target variable or the label to be predicted.

**test\_size**: This is the proportion of the dataset to include in the test split. The default value is 0.25 (25%). In this case, 20% of the data is used for testing and 80% for training.

**Split Data**: After the split, the original data is divided into two parts: a training set and a test set. The training set is used to train the model, while the test set is used to evaluate the model's performance.

**Code:**

ss = StandardScaler()

x\_train\_ss =ss.fit\_transform(X\_train)

x\_test\_ss = ss.fit\_transform(X\_test)

**Explanation:**

**StandardScaler**: This line of code initializes a StandardScaler object, which is a tool provided by the sklearn library for preprocessing the input features by removing the mean and scaling to unit variance.

**Fit and Transform**: These lines of code fit the StandardScaler object to the training data and then apply the scaling transformation to both the training and testing data.

Here, ss.fit\_transform(X\_train) first fits the scaler to the training data and then applies the scaling transformation to the training data. Similarly, ss.fit\_transform(X\_test) fits the scaler to the training data and then applies the scaling transformation to the testing data.

**Code:**

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(f1[["Age","EstimatedSalary"]], f1.Purchased, *test\_size* = 0.2)

**Explanation:**

**Train Test Split**: This line of code uses the train\_test\_split function from the sklearn.model\_selection module to split the input features and the target variable into training and testing sets.

**Feature and Target Variable**: The features (X) in this case are the Age and EstimatedSalary columns of the f1 dataframe.

**Test Size**: The test\_size parameter specifies the proportion of the dataset to include in the test split. In this case, the test size is set to 0.2, which means that 20% of the dataset will be used for testing and 80% for training

**Code:**

X\_train.size

**Explanation:**

The line of code X\_train.size is used to find the number of elements (features) in the X\_train dataset. This information is useful when understanding the size of the data being worked with, or when creating a new array with the same dimensions as X\_train.

**Code:**

model = LogisticRegression()

model.fit(x\_train\_ss, Y\_train)

**Explanation:**

a. LogisticRegression() creates an instance of the Logistic Regression model.

b. model.fit(X\_train, Y\_train) trains the model using the x\_train\_ss features and the Y\_train labels. This step may take some time depending on the size of the dataset and the complexity of the model.

**Code:**

model.predict(x\_test\_ss)

**Explanation:**

The line of code X\_train.size is used to find the number of elements (features) in the X\_train dataset. This information is useful when understanding the size of the data being worked with, or when creating a new array with the same dimensions as X\_train.

The line of code model.fit(X\_train, Y\_train) is used to train the logistic regression model. In this case, the model is being trained on the x\_train\_ss features and the Y\_train labels.

**Code:**

model.predict\_proba(x\_test\_ss)

**Explanation:**

a. LogisticRegression() creates an instance of the Logistic Regression model.

b. model.fit(X\_train, Y\_train) trains the model using the x\_train\_ss features and the Y\_train labels. This step may take some time depending on the size of the dataset and the complexity of the model.

**Code:**

model.score(x\_test\_ss, Y\_test)

**Explanation:**

a. LogisticRegression() creates an instance of the Logistic Regression model.

b. model.fit(X\_train, Y\_train) trains the model using the x\_train\_ss features and the Y\_train labels. This step may take some time depending on the size of the dataset and the complexity of the model.

After executing these lines of code, the model is ready to make predictions on new data. You can use the model's score() method to determine the accuracy of the model. The score() method returns the mean accuracy of the model, with 1.0 indicating a perfect model and values below 1.0 indicating an inaccurate model.

It's important to note that the predict\_proba() and score() methods should not be called in the same pipeline. predict\_proba() should be used to generate probabilities for each class in a multi-class classification problem, while score() should be used to calculate the accuracy of the model.

**Screenshots of the implementation of the code and the output:**





