

# Fundamentals of Machine Learning

## Lab Assignment – 5

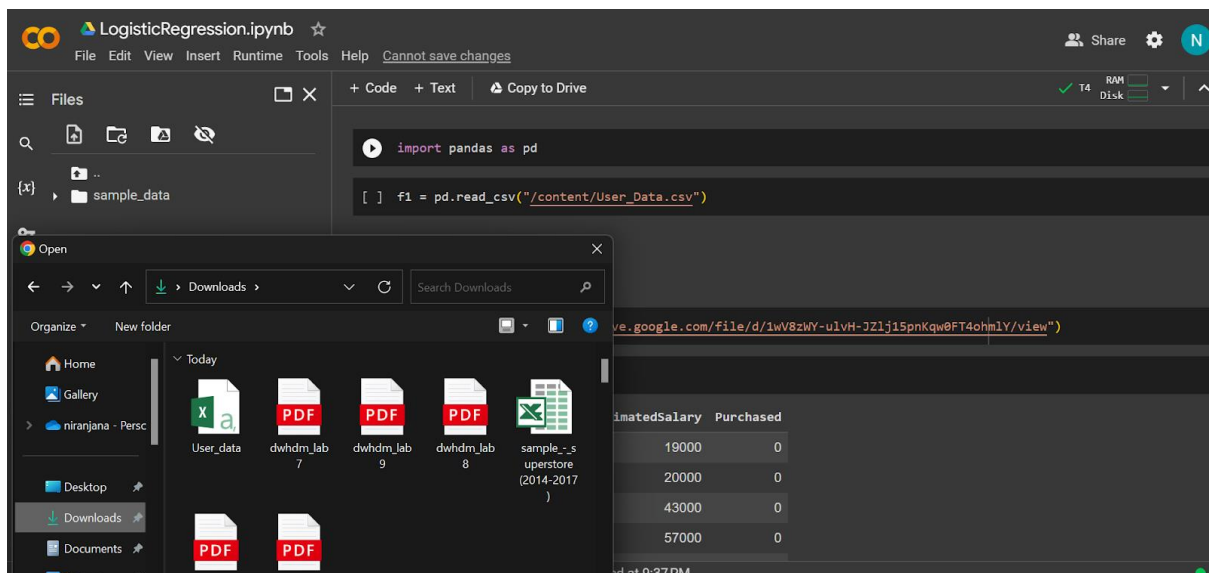
Name: Niranjana J  
USN: 22BTRAD027  
Branch: CSE- AI & DE

### Implementing LogisticRegression algorithm

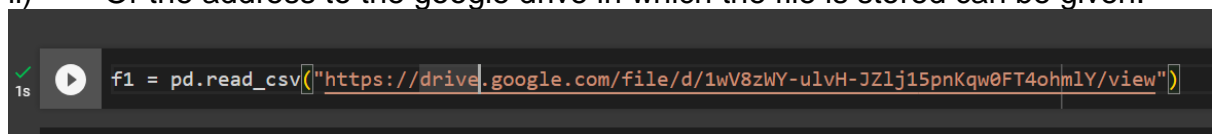
Logistic regression is a popular classification algorithm used to predict the probability of a binary outcome based on input variables.

Since we are performing the logistic regression on the User\_data csv file, we have to upload the file to the colab file section.

- i) Either the data file can be uploaded from the device.



- ii) Or the address to the google drive in which the file is stored can be given.



Once the file is uploaded, import pandas.

Pandas is a python library used for data analysis and manipulation. After importing the pandas library, the contents in the User\_data csv file is stored in a dataframe named f1.

```
✓ 0s [2] import pandas as pd
```

```
✓ 0s f1 = pd.read_csv("/content/User_data.csv")
```

Dataframe\_name.head() is a built-in function in pandas library that gives the top 5 rows in the dataframe.

```
✓ 0s f1.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Using the get\_dummies() function, categorical variables can be converted into dummy or indicator variables. Here, the field gender is converted into female and male. New fields named Gender\_Female and Gender\_Male are added as a result. The new data set is stored as a new data frame f2.

```
✓ 0s f2 = pd.get_dummies(f1, columns=["Gender"])
```

```
[ ] f2.head()
```

	User ID	Age	EstimatedSalary	Purchased	Gender_Female	Gender_Male
0	15624510	19	19000	0	0	1
1	15810944	35	20000	0	0	1
2	15668575	26	43000	0	1	0
3	15603246	27	57000	0	1	0
4	15804002	19	76000	0	0	1

Top 5 rows of f2 include:

```
f2.head()
```

	User ID	Age	EstimatedSalary	Purchased	Gender_Female	Gender_Male
0	15624510	19	19000	0	0	1
1	15810944	35	20000	0	0	1
2	15668575	26	43000	0	1	0
3	15603246	27	57000	0	1	0
4	15804002	19	76000	0	0	1

Later on, import the LogisticRegression function from linear\_model module, train\_test\_split function from model-selection and StandardScaler function from preprocessing module .

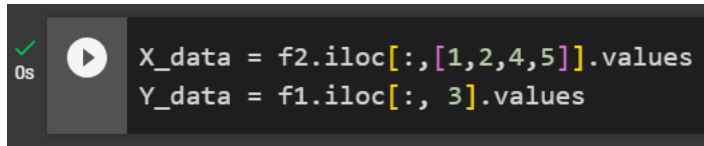
```
✓ [11] from sklearn.linear_model import LogisticRegression  
1s    from sklearn.model_selection import train_test_split  
      from sklearn.preprocessing import StandardScaler
```

The first import statement imports the LogisticRegression class from the sklearn.linear\_model module. This class provides the implementation of logistic regression.

The second import statement imports the train\_test\_split function from the sklearn.model\_selection module. This function is used to split the dataset into training and testing sets.

The third import statement imports the StandardScaler class from the sklearn.preprocessing module. This class is used to standardize the input variables by subtracting the mean and scaling to unit variance.

Now, the data should be classified into the input set(X\_data) and output set(Y\_data)



```
X_data = f2.iloc[:, [1, 2, 4, 5]].values
Y_data = f1.iloc[:, 3].values
```

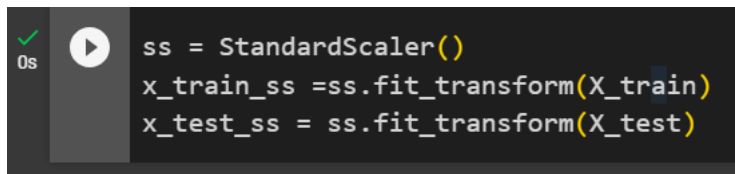
The code

```
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data,
test_size = 0.2)
```

Splits 80% of X\_data and Y\_data as the training data and rest 20% as the testing data set. This is stored in X\_train, X\_test, Y\_train, Y\_test respectively.

Afterwards, the following code does:

1. Importing the necessary libraries and modules.
2. Creating an instance of the StandardScaler class.
3. Scaling the training and testing data using the fit\_transform method.



```
ss = StandardScaler()
x_train_ss = ss.fit_transform(X_train)
x_test_ss = ss.fit_transform(X_test)
```

In the following code,

X\_train: The training set's input features (independent variables).

X\_test: The testing set's input features (independent variables).

Y\_train: The training set's target variable (dependent variable).

Y\_test: The testing set's target variable (dependent variable).

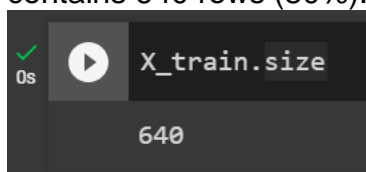
f1[["Age", "EstimatedSalary"]]: The input features used for train-test split. In this case, it includes the "Age" and "EstimatedSalary" columns from the dataset.

f1.Purchased: The target variable used for train-test split. It represents the "Purchased" column from the dataset.

test\_size = 0.2: This parameter specifies the proportion of the dataset that should be allocated for testing. In this case, 20% of the data will be used for testing, while the remaining 80% will be used for training.

```
X_train, X_test, Y_train, Y_test =
train_test_split(f1[["Age", "EstimatedSalary"]], f1.Purchased, test_size
= 0.2)
```

.size() function gives the size of the dataframe. In this case, the training dataset contains 640 rows (80%).



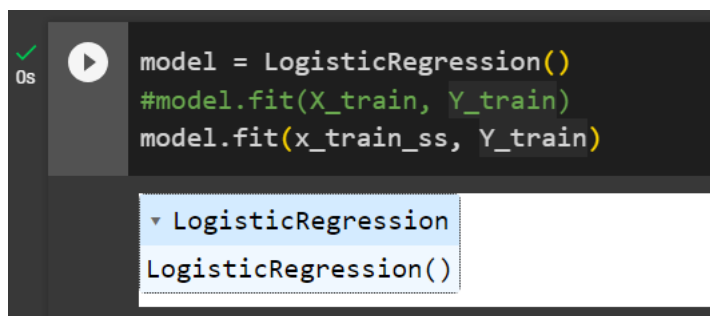
```
X_train.size
```

640

From this, it is understandable that the testing dataset contains 160 records and there are a total of 800 records in the dataset.

Here, an instance of the LogisticRegression class and assign it to the variable model. This class is part of the scikit-learn library, which provides a wide range of machine learning algorithms and tools.

The fit() method is then called on the model object to train the logistic regression model. The fit() method takes two arguments: x\_train\_ss and Y\_train. Here, x\_train\_ss represents the input features or independent variables, and Y\_train represents the target variable or dependent variable.



```
model = LogisticRegression()  
#model.fit(X_train, Y_train)  
model.fit(x_train_ss, Y_train)
```

The screenshot shows a Jupyter Notebook interface. On the left, there is a play button icon and a '0s' execution time. The main area displays the code for creating a LogisticRegression model and fitting it to the training data. Below the code, a dropdown menu is open, showing 'LogisticRegression' and 'LogisticRegression()'.

The x\_test\_ss dataset is fed into the model and the following output are obtained.



```
model.predict(x_test_ss)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

The screenshot shows a Jupyter Notebook interface. The code cell contains the command to predict the test dataset. The output is a large array of zeros, indicating that the model predicted the class 0 for all test samples.

predict\_proba function that allows us to obtain the predicted probabilities for each class. This function takes the input data as an argument and returns an array of probabilities for each class.

Now, By calling the predict\_proba function on the model object with x\_test\_ss as the argument, we can obtain the predicted probabilities for each class.

```
[0.56087873, 0.43912127],
[0.64916843, 0.35083157],
[0.66997469, 0.33002531],
[0.65842763, 0.34157237],
[0.5118892 , 0.4881108 ],
[0.53083136, 0.46916864],
[0.67381333, 0.32618667],
[0.77905571, 0.22094429],
[0.67408955, 0.32591045],
[0.64787267, 0.35212733],
[0.69354884, 0.30645116],
[0.65539492, 0.34460508],
[0.66944154, 0.33055846],
[0.69330367, 0.30669633],
[0.64473921, 0.35526079],
[0.56268588, 0.43731412],
[0.69189461, 0.30810539],
[0.72602189, 0.27397811],
[0.69596792, 0.30403208],
[0.65208789, 0.34791211],
[0.75001609, 0.24998391],
[0.62559131, 0.37440869],
[0.66445417, 0.33554583],
[0.71243734, 0.28756266],
[0.66414092, 0.33585908],
[0.52229248, 0.47770752],
[0.51464871, 0.48535129],
[0.73954604, 0.26045396],
[0.66369215, 0.33630785],
[0.6825684 , 0.3174316 ],
[0.53478087, 0.46521913],
[0.55141213, 0.44858787],
[0.68013949, 0.31986051],
[0.61329426, 0.38670574],
[0.64027634, 0.35972366],
[0.72048436, 0.27951564],
[0.63977895, 0.36022105],
[0.59804262, 0.40195738],
```

```
model.predict_proba(x_test_ss)
array([[0.58136641, 0.41863359],
       [0.63131245, 0.36868755],
       [0.63684787, 0.36315213],
       [0.58297629, 0.41702371],
       [0.68146022, 0.31853978],
       [0.71170672, 0.28829328],
       [0.66744762, 0.33255238],
       [0.73437798, 0.26562202],
       [0.61881689, 0.38118311],
       [0.72268141, 0.27731859],
       [0.65547649, 0.34452351],
       [0.60849647, 0.39150353],
       [0.68820878, 0.31179122],
       [0.73166771, 0.26833229],
       [0.53430628, 0.46569372],
       [0.65528135, 0.34471865],
       [0.66747199, 0.33252801],
       [0.68286447, 0.31713553],
       [0.700833 , 0.299167 ],
       [0.68774534, 0.31225466],
       [0.62654328, 0.37345672],
       [0.58340407, 0.41659593],
       [0.67279995, 0.32720005],
       [0.6000564 , 0.3999436 ],
       [0.69464907, 0.30535093],
       [0.72680821, 0.27319179],
       [0.52418641, 0.47581359],
       [0.69311316, 0.30688684],
       [0.69526663, 0.30473337],
       [0.74464851, 0.25535149],
       [0.64121239, 0.35878761],
       [0.61130396, 0.38869604],
       [0.68287267, 0.31712733],
       [0.64654237, 0.35345763],
       [0.65899418, 0.34100582],
```

```
[0.55804202, 0.44195798],
[0.65825799, 0.34174201],
[0.57128965, 0.42871035],
[0.79741755, 0.20258245],
[0.61659955, 0.38340045],
[0.56733765, 0.43266235],
[0.70828453, 0.29171547],
[0.58494295, 0.41505705]])
```

The score() function is called on the model object to evaluate its performance on the test data.

According to this, the model is 61.25% accurate.

```
[21] model.score(x_test_ss, Y_test)

0.6125
```

After assessing the accuracy of a model, necessary steps can be taken to improve its accuracy such as add more data, treat missing and outlier values, feature engineering, algorithm tuning, cross validation etc.