Experiment. 1.

Perform Data Pre-processing on a Dataset

Aim: Perform data pre-processing on a dataset

Hardware/Software requirements: python

Course Outcomes: Understand the basic concepts of machine learning

Theory : Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data pre-processing task.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model. It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Splitting dataset into training and test set
- Feature scaling

Get the Dataset

The collected data for a particular problem in a proper format is known as the **dataset**.

Here we will use a demo dataset for data preprocessing, and for practice, it can be downloaded from here, https://www.superdatascience.com/pages/machine-learning

For real-world problems, we can download datasets online from various sources such as https://www.kaggle.com/uciml/datasets, https://archive.ics.uci.edu/ml/index.php etc.

Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries.

```
# import numpy as nm

# import matplotlib.pyplot as mpt

# import padas as pd
```

Importing the Datasets

Save the dataset with .csv format and then import the data with read_csv() function of pandas library.

We can use read_csv function as below:

```
# data_set= pd.read_csv('Dataset.csv')
```

Extracting independent variable:

```
# x= data_set.iloc[:,:-1].values
```

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

```
# y= data_set.iloc[:,3].values
```

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

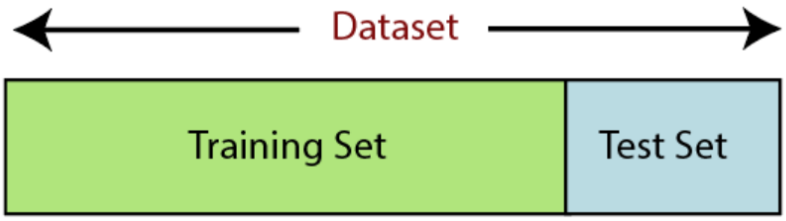
Handling Missing data: Calculating the mean we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

```
#handling missing data (Replacing missing data with the mean value)

from sklearn.preprocessing import Imputer

imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)

#Fitting imputer object to the independent variables x.

imputerimputer= imputer.fit(x[:, 1:3])

#Replacing missing data with the calculated mean value

x[:, 1:3]= imputer.transform(x[:, 1:3])
```

Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

```
# from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
```

Feature Scaling: Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

```
# st_x= StandardScaler()

# x_train= st_x.fit_transform(x_train)
```

For test dataset, we will directly apply **transform()** function instead of **fit_transform()**

because it is already done in training set.

```
# x_test= st_x.transform(x_test)
```

Combining all the steps:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('Dataset.csv')
8.
9. #Extracting Independent Variable
10. x= data_set.iloc[:, :-1].values
```

```
11.
12. #Extracting Dependent variable
13. y= data_set.iloc[:, 3].values
14.
15. #handling missing data(Replacing missing data with the mean value)
16. from sklearn.preprocessing import Imputer
17. imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)
18.
19. #Fitting imputer object to the independent varibles x.
20. imputerimputer= imputer.fit(x[:, 1:3])
21.
22. #Replacing missing data with the calculated mean value
23. x[:, 1:3]= imputer.transform(x[:, 1:3])
24.
25. #for Country Variable
26. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
27. label_encoder_x= LabelEncoder()
28. x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
29.
30. #Encoding for dummy variables
31. onehot_encoder= OneHotEncoder(categorical_features= [0])
32. x= onehot_encoder.fit_transform(x).toarray()
33.
34. #encoding for purchased variable
35. labelencoder_y= LabelEncoder()
36. y= labelencoder_y.fit_transform(y)
37.
38. # Splitting the dataset into training and test set.
39. from sklearn.model_selection import train_test_split
40. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
41.
42. #Feature Scaling of datasets
43. from sklearn.preprocessing import StandardScaler
44. st_x= StandardScaler()
45. x_train= st_x.fit_transform(x_train)
46. x_test= st_x.transform(x_test)
```

Conclusion : Students are able to understand the need of pre-processing steps and implemented same with python on the available dataset.

Viva Questions:

- What is Data Preprocessing?
- What do you mean by feature scaling or data normalization? Explain some techniques for feature scaling?
- What are the missing values? and How do you handle missing values?

Reference link :

- **https://www.youtube.com/watch?v=NSxEiohAH5o**
- **https://towardsdatascience.com/data-preprocessing-in-python-b52b652e37d5**

Experiment No.2

1. **Aim:** Implementation of Principal Component Analysis
2. **Objectives:**
    - To understand the basic concepts and importance of pre-processing of data analysis
    - To apply simple PCA reduction technique on actual dataset
3. **Outcomes**: Students will be able to implement PCA reduction technique.
4. **Hardware / Software Required:** Python 3.10
5. **Theory:** Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modelling. It is a technique to draw strong patterns from the given dataset by reducing the variances.
   Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

   The PCA algorithm is based on some mathematical concepts such as:
    - Variance and Covariance
    - Eigenvalues and Eigen factors

   Some common terms used in PCA algorithm:
   Dimensionality: It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
   Correlation: It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
   Orthogonal: It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
   Eigenvectors: If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
   Covariance Matrix: A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

   The transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:
   The principal component must be the linear combination of the original features.
   These components are orthogonal, i.e., the correlation between a pair of variables is zero.

The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Applications of Principal Component Analysis:

- PCA is mainly used as the dimensionality reduction technique in various AI applications such as computer vision, image compression, etc.
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

6. Steps to implement PCA in python

I. Import the python libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

II Import dataset

Load the breast_cancer dataset from sklearn.datasets. It is clear that the dataset has 569 data items with 30 input attributes. There are two output classes-benign and malignant. Due to 30 input features, it is impossible to visualize this data.

```
#import the breast _cancer dataset

from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
data.keys()

# Check the output classes
print(data['target_names'])

# Check the input attributes
print(data['feature_names'])
```

Output:

In [2]:

```
#import the breast _cancer dataset
from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
data.keys()

# Check the output classes
print(data['target_names'])

# Check the input attributes
print(data['feature_names'])
```

```
['malignant' 'benign']
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

III.Apply PCA

- Standardize the dataset prior to PCA.
- Import PCA from sklearn.decomposition.
- Choose the number of principal components.

After executing this code, we get to know that the dimensions of x are (569,3) while the dimension of actual data is (569,30). Thus, it is clear that with PCA, the number of dimensions has reduced to 3 from 30. If we choose n_components=2, the dimensions would be reduced to 2.

```
# construct a dataframe using pandas
df1=pd.DataFrame(data['data'],columns=data['feature_names'])

# Scale data before applying PCA
scaling=StandardScaler()

# Use fit and transform method
scaling.fit(df1)
Scaled_data=scaling.transform(df1)

# Set the n_components=3
principal=PCA(n_components=3)
principal.fit(Scaled_data)
x=principal.transform(Scaled_data)

# Check the dimensions of data after PCA
print(x.shape)
```

Output:

In [3]:
```
# construct a dataframe using pandas
df1=pd.DataFrame(data['data'],columns=data['feature_names'])

# Scale data before applying PCA
scaling=StandardScaler()

# Use fit and transform method
scaling.fit(df1)
Scaled_data=scaling.transform(df1)

# Set the n_components=3
principal=PCA(n_components=3)
principal.fit(Scaled_data)
x=principal.transform(Scaled_data)

# Check the dimensions of data after PCA
print(x.shape)
```

```
(569, 3)
```

IV. Check Components

The principal.components_ provide an array in which the number of rows tells the number of principal components while the number of columns is equal to the number of features in actual data. We can easily see that there are three rows as n_components was chosen to be 3. However, each row has 30 columns as in actual data.

```
# Check the values of eigen vectors
# prodeced by principal components
principal.components_
```

In [4]:
```
# Check the values of eigen vectors
# prodeced by principal components
principal.components_
```

Out[4]:
```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611303,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947],
       [-0.00853121,  0.06454991, -0.00931419,  0.02869954, -0.10429169,
        -0.07409158,  0.00273377, -0.02556363, -0.04023992, -0.02257429,
         0.26848139,  0.37463365,  0.26664534,  0.21600657,  0.30883889,
         0.15477992,  0.17646379,  0.22465747,  0.28858428,  0.21150386,
        -0.04750699, -0.0422978 , -0.04854652, -0.01190234, -0.25979755,
        -0.23607555, -0.17305735, -0.17034419, -0.27131265, -0.23279139]])
```

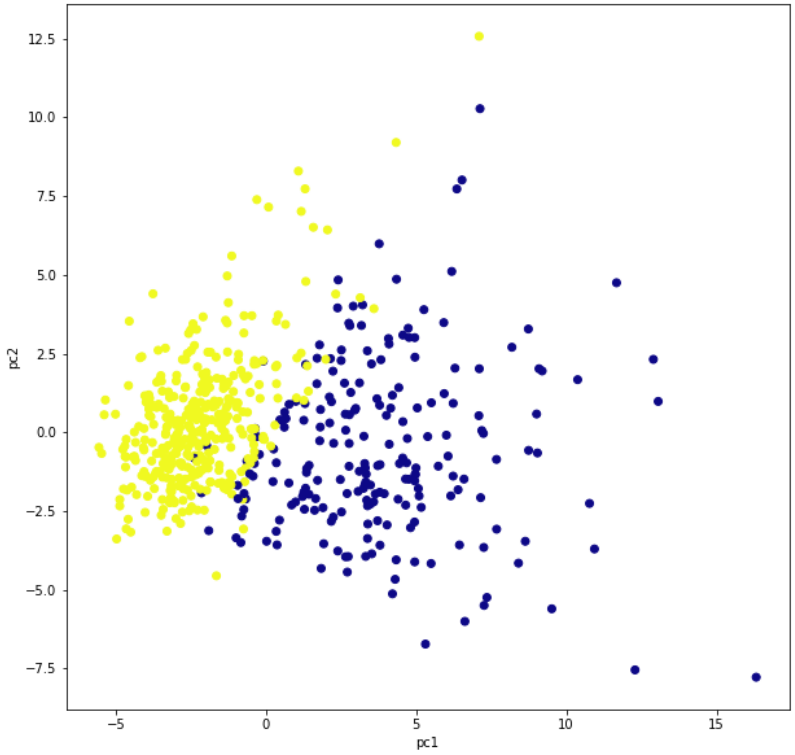
V. Plot the components (Visualization)

Plot the principal components for better data visualization. Though we had taken n_components =3, here we are plotting a 2d graph as well as 3d using first two principal components and 3 principal components respectively. For three principal components, we need to plot a 3d graph. The colors show the 2 output classes of the original dataset-benign and malignant. It is clear that principal components show clear separation between two output classes.

```
plt.figure(figsize=(10,10))
plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma')
plt.xlabel('pc1')
plt.ylabel('pc2')
```

Output:

```
In [5]:
1 plt.figure(figsize=(10,10))
2 plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma')
3 plt.xlabel('pc1')
4 plt.ylabel('pc2')

Out[5]: Text(0, 0.5, 'pc2')
```



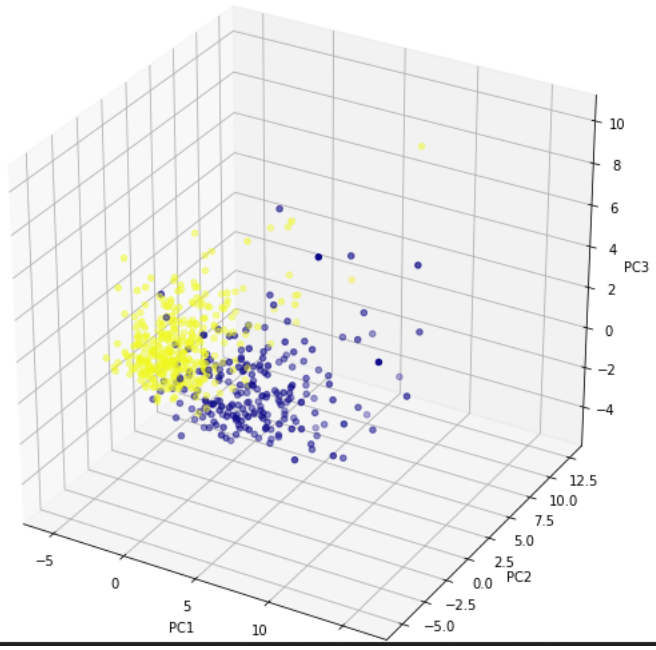
For three principal components, we need to plot a 3d graph. x[:,0] signifies the first principal component. Similarly, x[:,1] and x[:,2] represent the second and the third principal component.

```
# import relevant libraries for 3d graph
from mpl_toolkits.mplot3d import Axes3D
fig =plt.figure(figsize=(10,10))
```

```
# choose projection 3d for creating a 3d graph
axis =fig.add_subplot(111, projection='3d')

# x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3
axis.scatter(x[:,0],x[:,1],x[:,2], c=data['target'],cmap='plasma')
axis.set_xlabel("PC1", fontsize=10)
axis.set_ylabel("PC2", fontsize=10)
axis.set_zlabel("PC3", fontsize=10)
```

Output:

```
In [6]:
# import relevant libraries for 3d graph
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,10))

# choose projection 3d for creating a 3d graph
axis = fig.add_subplot(111, projection='3d')

# x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3
axis.scatter(x[:,0],x[:,1],x[:,2], c=data['target'],cmap='plasma')
axis.set_xlabel("PC1", fontsize=10)
axis.set_ylabel("PC2", fontsize=10)
axis.set_zlabel("PC3", fontsize=10)

Out[6]: Text(0.5, 0, 'PC3')
```



VI. Calculate variance ratio

Explained_variance_ratio provides an idea of how much variation is explained by principal components.

```
# check how much variance is explained by each principal component
print(principal.explained_variance_ratio_)
```

Output:

```
In [7]:
1 # check how much variance is explained by each principal component
2 print(principal.explained_variance_ratio_)
```

```
[0.44272026 0.18971182 0.09393163]
```

7. Conclusion: Implemented PCA and understand the use and its applications.

8. Viva Questions:

- What is a PCA?
- Can you list out the critical assumptions of linear regression?
- What is the primary difference between R square and adjusted R square?
- How Principal Component Analysis (PCA) is used for Dimensionality Reduction?

Experiment. 3

Aim: To implement Linear Regression.

SoftwareRequired:(Studentsshouldwrite**Softwarerequired** basedonsoftware used for implementing program)**e.g.,** python

Theory: Linear regression is a basic predictive analytics technique that uses historical data to predict an output variable. It is popular for predictive modeling because it is easily understood. Linear regression models have many real-world applications in an array of industries such as economics (e.g., predicting growth), business (e.g., predicting product sales, employee performance), social science (e.g., predicting political leanings from gender or race), healthcare (e.g., predicting blood pressure levels from weight, disease onset from biological factors), and more.

The basic idea is that if we can fit a linear regression model to observed data, we can then use the model to predict any future values. There are two kinds of variables in a linear regression model:

- The input or predictor variable is the variable(s) that help predict the value of the output variable. It is commonly referred to as X.
- The output variable is the variable that we want to predict. It is commonly referred to as Y.

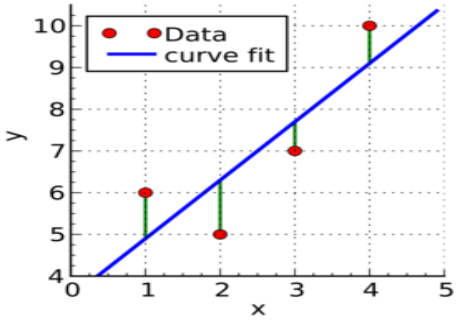
To estimate Y using linear regression, we assume the equation:

$$Y_e = \alpha + \beta X$$

where $\mathbf{Y_e}$ is the estimated or predicted value of **Y** based on our linear equation.

Our goal is to find statistically significant values of the parameters α and β that minimize the difference between Y and Y_e. If we are able to determine the optimum values of these two parameters, then we will have the line of best fit that we can use to predict the values of Y, given the value of X. How do we estimate α and β? We can use a method called ordinary least squares.

Ordinary Least Squares



Green lines show the difference between actual values **Y** and estimate values $\mathbf{Y_e}$

The objective of the least squares method is to find values of α and β that minimize the sum of the squared difference between Y and Y_e. We will not go through the derivation here, but using calculus we can show that the values of the unknown parameters are as follows:

$$\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\alpha = \bar{Y} - \beta * \bar{X}$$

where $\bar{X}$ is the mean of X values and $\bar{Y}$ is the mean of Y values.

If you are familiar with statistics, you may recognize β as simply Cov(X, Y) / Var(X).

Implementation Steps:

1. Generate data with nonzero mean and standard deviation (X). Also, create random data by multiplying a constant and adding residual which is an actual data (Y).
2. Calculate the mean of X and Y

   Mean_X = X1+X2+…X_N/ N

   Mean_Y = Y1+Y2+…Y_N/ N
3. Calculate *α* and *β using*

$$\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\alpha = \bar{Y} - \beta * \bar{X}$$

4. Compute the predicted output y

$$Y_e = \alpha + \beta X$$

5. Plot regression against actual data

OutputAnalysis:(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

AdditionalLearning:(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

- Regression analysis allows you to understand the strength of relationships between variables.
- Regression analysis tells you what predictors in a model are statistically significant and which are not.
- Regression analysis can give a confidence interval for each regression coefficient that it estimates
- The simple linear regression method tries to find the relationship between a single independent variable and a corresponding dependent variable. The independent variable is the input, and the corresponding dependent variable is the output.
- The multiple linear regression method tries to find the relationship between two or more independent variables and the corresponding dependent variable. There's also a special case of multiple linear regression called polynomial regression. There are other non-linear regression methods used for highly complicated data analysis.

Conclusion:(Students should write proper conclusion (in the form of discussion)on their own)

- Linear regression is a statistical method that tries to show a relationship between variables. It looks at different data points and plots a trend line. A simple example of linear regression is finding that the cost of repairing a piece of machinery increases with time.
- More precisely, linear regression is used to determine the character and strength of the association between a dependent variable and a series of other independent variables. It helps create models to make predictions.

Experiment. 4

Aim: To implement Logistic Regression.

Softwarerequired:(StudentsshouldwriteSoftwarerequired basedonsoftware used for implementing program)

Theory: In statistics logistic regression is used to model the probability of a certain class or event. Logistic regression is similar to linear regression because both of these involve estimating the values of parameters used in the prediction equation based on the given training data. Linear regression predicts the value of some continuous, dependent variable. Whereas logistic regression predicts the probability of an event or class that is dependent on other factors. Thus, the output of logistic regression always lies between 0 and 1. Because of this property it is commonly used for classification purpose.

Logistic Model

Consider a model with features *x1, x2, x3 … xn*. Let the binary output be denoted by *Y*, that can take the values 0 or 1. Let *p* be the probability of *Y = 1*, we can denote it as *p = P(Y=1)*. The mathematical relationship between these variables can be denoted as:

$$ln(\frac{p}{1-p}) = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 \ldots b_n x_n$$

Here the term *p/(1−p)* is known as the *odds* and denotes the likelihood of the event taking place. Thus *ln(p/(1−p))* is known as the *log odds* and is simply used to map the probability that lies between 0 and 1 to a range between (−∞,+∞). The terms *b0, b1, b2…* are parameters (or weights) that we will estimate during training.
So we simplify the equation to obtain the value of p:

1. The log term ln on the LHS can be removed by raising the RHS as a power of e:

$$\frac{p}{1-p} = e^{b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 \ldots b_n x_n}$$

2. Now we can easily simplify to obtain the value of p :

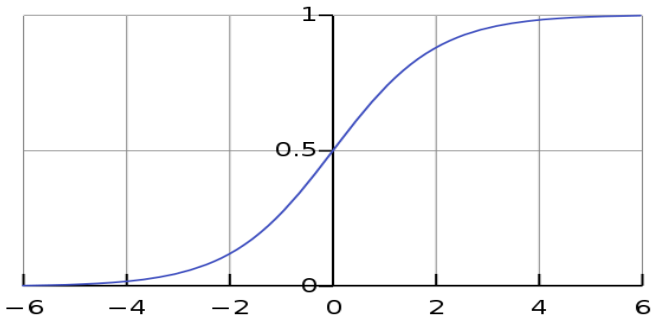
$$p = \frac{e^{b_0+b_1x_1+b_2x_2+b_3x_3...b_nx_n}}{1+e^{b_0+b_1x_1+b_2x_2+b_3x_3...b_nx_n}}$$

$$or$$

$$p = \frac{1}{1+e^{-(b_0+b_1x_1+b_2x_2+b_3x_3...b_nx_n)}}$$

3. This actually turns out to be the equation of the Sigmoid Function which is widely used in other machine learning applications. The Sigmoid Function is given by:

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



4. Now we will be using the above derived equation to make our predictions. Before that we will train our model to obtain the values of our parameters b0, b1, b2… that result in least error. This is where the error or loss function comes in.

Loss Function

The loss is basically the error in our predicted value. In other words, it is a difference between our predicted value and the actual value. We will be using the L2 Loss Function to calculate the error. Theoretically you can use any function to calculate the error. This function can be broken down as:

1. Let the actual value be y_i. Let the value predicted using our model be denoted as $\bar{y}_i$. Find the difference between the actual and predicted value.
2. Square this difference.
3. Find the sum across all the values in training data.

$$L = \sum_{i=1}^{n}(y_i - \bar{y_i})^2$$

Now that we have the error, we need to update the values of our parameters to minimize this error. This is where the “learning” actually happens, since our model is updating itself based on its previous output to obtain a more accurate output in the next step. Hence with each iteration our model becomes more and more accurate. We will be using the ***Gradient Descent Algorithm*** to estimate our parameters. Another commonly used algorithm is the Maximum Likelihood Estimation.

The Gradient Descent Algorithm

You might know that the partial derivative of a function at its minimum value is equal to 0. So gradient descent basically uses this concept to estimate the parameters or weights of our model by minimizing the loss function. For simplicity, assume that our output depends only on a single feature *x*. So, we can rewrite our equation as:

$$\bar{y_i} = p = \frac{1}{1 + e^{-(b_0 + b_1 x_i)}} = \frac{1}{1 + e^{-b_0 - b_1 x_i}} \quad (1)$$

Thus, we need to estimate the values of weights b0 and b1 using our given training data.

1. Initially let b0=0 and b1=0. Let L be the learning rate. The learning rate controls by how much the values of b0 and b1 are updated at each step in the learning process. Here let L=0.001.
2. Calculate the partial derivative with respect to b0 and b1. The value of the partial derivative will tell us how far the loss function is from its minimum value. It is a measure of how much our weights need to be updated to attain minimum or ideally 0 error. In case you have more than one feature, you need to calculate the partial derivative for each weight b0, b1 … bn where n is the number of features.

$$D_{b_0} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times \bar{y_i} \times (1 - \bar{y_i})$$
$$D_{b_1} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times \bar{y_i} \times (1 - \bar{y_i}) \times x_i \quad (2)$$

3. Next we update the values of b0 and b1:

$$b_0 = b_0 - L \times D_{b_0}$$
$$b_1 = b_1 - L \times D_{b_1} \quad (3)$$

4. We repeat this process until our loss function is a very small value or ideally reaches 0 (meaning no errors and 100% accuracy). The number of times we repeat this learning process is known as **iterations or epochs.**

Steps:

1. Read and visualize the database
2. Divide the data to training set and test set (80:20)
3. Perform data normalization
4. Compute the method to make predictions using equation (1)
5. Evaluate the method to train the model using equations (2) and (3). Obtain b0 and b1
6. Train the model using training data
7. Make the predictions
8. Calculate accuracy

OutputAnalysis:(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

AdditionalLearning:(Studentsshouldwriteadditionallearningontheirownbasedonwhatadditionally)

- Linear regression predicts the continuous dependent variable for a given set of independent variables, logistic regression predicts the categorical dependent variable.
- Linear regression is used to solve regression problems, logistic regression is used to solve classification problems.
-

Conclusion:(Students should write proper conclusion (in the form of discussion) on their own)

- Logistic regression can solve regression problems, but it's mainly used for classification problems. Its output can only be 0 or 1. It's valuable in situations where we need to determine the probabilities between two classes or, in other words, calculate the likelihood of an event.