**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Practical Name** | **Date** | **Remarks/Sign** |
| 1 | Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file. | 20/01/2022 |  |
| 2 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples. | 27/01/2022 |  |
| 3 | Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | 03/02/2022 |  |
| 4 | Write a program to implement the K nearest neighbour classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | 10/02/2022 |  |
| 5 | Write a program to implement the Decision Tree classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Also generate the random forest classifier. | 17/02/2022 |  |
| 6 | Write a program to implement the Linear Regression for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | 24/02/2022 |  |
| 7 | Write a program to implement the Logistic Regression for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | 17/03/2022 |  |
| 8 | Write a program to implement the support vector machine for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. | 24/03/2022 |  |
| 9 | Write a program to implement the K-mean clustering for a sample training data set stored as a .CSV file. | 31/03/2022 |  |
| 10 | Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. | 07/04/2022 |  |

**U19EC046 | ML | LAB 1**

**AIM**

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

**THEORY:**

In Machine Learning, concept learning can be termed as “*a problem of searching through a predefined space of potential hypothesis for the hypothesis that best fits the training examples”* ​

In order to understand Find-S algorithm, you need to have a basic idea of the following concepts as well:​

* Concept Learning​
* General Hypothesis​
* Specific Hypothesis​

**Concept Learning​**

Machines can also learn from concepts to identify whether an object belongs to a specific category or not. Any algorithm that supports concept learning requires the following:​

* Training Data​
* Target Concept​
* Actual Data Objects

**General Hypothesis​**

Hypothesis, in general, is an explanation for something. The general hypothesis basically states the general relationship between the major variables. For example, a general hypothesis for ordering food would be *I want a burger.*​

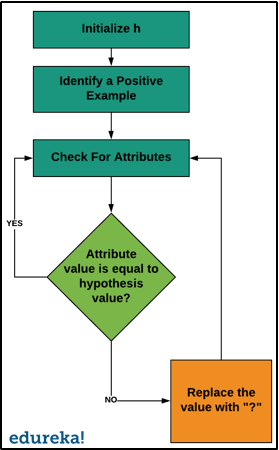
* G = { ‘?’, ‘?’, ‘?’, …..’?’}

**Specific Hypothesis​**

The specific hypothesis fills in all the important details about the variables given in the general hypothesis. The more specific details into the example given above would be *I want a cheeseburger with a lot of lettuce.* ​

* S = {‘Φ’,’Φ’,’Φ’, ……,’Φ’}

**ALGORIHM**



**CODE**

|  |
| --- |
| **import numpy as np**  **import pandas as pd**  **data = pd.read\_csv('./Data.csv')**  **def seriesToList(series):**  **return [value for \_, value in series.items()]**  **def initialHypothesis(df):**  **for i, r in df.iterrows():**  **if r[-1] == 'Yes':**  **return seriesToList(r)**  **def updateHypothisis(h, curr):**  **currList = seriesToList(curr)**  **if currList[-1] == 'Yes':**  **for i in range(len(h)):**  **if h[i] != '?' and h[i] != currList[i]:**  **h[i] = '?'**  **h = initialHypothesis(data)**  **print(f"initial Hypothesis : {h}")**  **for i, r in data.iterrows():**  **updateHypothisis(h, r)**  **print(f"final hypothesis : {h[:-1]}")** |

**OUTPUT**

|  |
| --- |
| initial Hypothesis :  ['Morning', 'Sunny', 'Warm', 'Yes', 'Mild ', 'Strong', 'Yes']  final hypothesis :  ['?', 'Sunny', '?', 'Yes', '?', '?'] |

**CONCLUSION**

In this practical we have implemented Python code for FIND-S algorithm for finding most specific hypothesis from given training samples.

**U19EC046 | ML | LAB 2**

**AIM**

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

**THEORY**

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

* You can consider this as an extended form of Find-S algorithm.
* Consider both positive and negative examples.
* Actually, positive examples are used here as Find-S algorithm (Basically they are generalizing from the specification).
* While the negative example is specified from generalize form.

**ALGORITHM**

Step1: Load Data set

Step2: Initialize General Hypothesis  and Specific  Hypothesis.

Step3: For each training example

Step4: If example is positive example

if attribute value == hypothesis value:

Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

Make generalize hypothesis more specific.

**CODE**

|  |
| --- |
| **import numpy as np**  **import pandas as pd**  **class CandiateElimination:**  **def \_\_init\_\_(self, csvPath):**  **self.csvPath = csvPath**  **self.data = pd.read\_csv(self.csvPath)**  **self.specificHypothesis = []**  **self.generalHypothesis = []**  **self.initailizeSpecificHypothesis()**  **self.hypothesisLength = len(self.specificHypothesis)**  **self.initalizeGeneralHypothesis()**    **def seriesToList(self, series):**  **return [value for \_, value in series.items()]**    **def initailizeSpecificHypothesis(self):**  **for i, r in self.data.iterrows():**  **if r[-1] == 'Yes':**  **self.specificHypothesis =  self.seriesToList(r)[:-1]**  **break**    **def initalizeGeneralHypothesis(self):**  **for i in range(self.hypothesisLength):**  **self.generalHypothesis.append(**  **['?' for \_ in range(self.hypothesisLength)])**    **def updateSpecificHypothisis(self, instance):**  **for i in range(len(self.specificHypothesis)):**  **if self.specificHypothesis[i] != '?'**  **and self.specificHypothesis[i] != instance[i]:**  **self.specificHypothesis[i] = '?'**  **def updateGeneralHypothisis(self, instance):**  **for i in range(self.hypothesisLength):**  **if self.specificHypothesis[i] != '?'**  **and self.specificHypothesis[i] != instance[i]:**  **self.generalHypothesis[i][i] = self.specificHypothesis[i]**    **def discardContradicting(self):**  **\_generalHypothesis = []**  **for i in range(self.hypothesisLength):**  **if self.generalHypothesis[i][i] == self.specificHypothesis[i] and self.specificHypothesis[i] != '?':**  **\_generalHypothesis.append(self.generalHypothesis[i])**  **self.generalHypothesis = \_generalHypothesis**    **def fit(self, instance):**  **currInstance = self.seriesToList(instance)**  **if currInstance[-1]=='Yes':**  **self.updateSpecificHypothisis(currInstance)**  **else:**  **self.updateGeneralHypothisis(currInstance)**  **def getHypothesis(self):**  **for i, r in self.data.iterrows():**  **self.fit(r)**  **self.discardContradicting()**  **return (self.specificHypothesis, self.generalHypothesis)**    **myHypothesis = CandiateElimination('cea.csv')**  **(specific, general) = myHypothesis.getHypothesis()**  **print(f"\**  **Specific Hypothesis : {specific}\n\**  **General Hypothesis : {general}\**  **")** |

**OUTPUT**

Specific Hypothesis :

['Sunny', 'Warm ', '?', 'Strong', '?', '?']

General Hypothesis :

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm ', '?', '?', '?', '?']]

**CONCLUSION**

In this practical we have implemented python code for Candidate elimination algorithm and found specific and general hypothesis for given dataset.

**U19EC046 | ML | LAB 3**

**AIM**

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

**ALGORITHM**

1. Import necessary libraries
2. Read the dataset using pandas
3. Split the dataset into features and results
4. Split the features and results into training and testing dataset using scikit learn

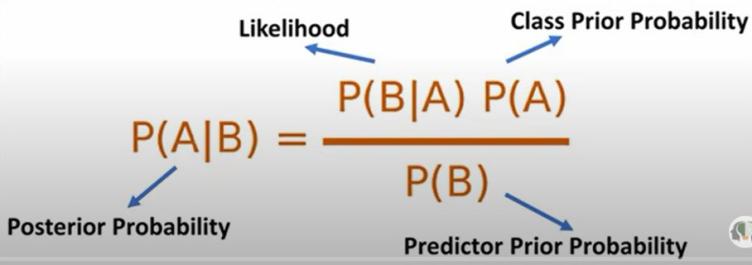
test\_train\_split

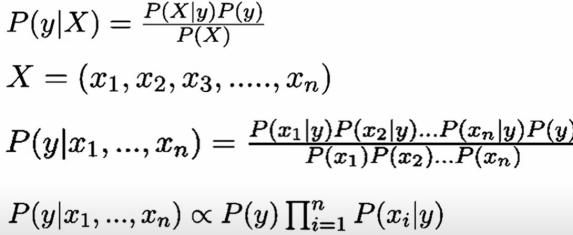
1. Apply pre-processing if needed, use scikit-learn preprocessing class
2. Create an instance of gaussian naive bais classifier
3. Train the model using fit method
4. Predict the result using predict method on model
5. Find the accuracy of model using score method on tesing dataset

**THEORY:**

Naive Bayes classifiers are a collection of classification algorithms based on Bayes’ Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

**Formulas involved:**







**CODE**

1. Social Network Ads Database

|  |
| --- |
| **import pandas as pd**  **from sklearn.naive\_bayes import GaussianNB**  **from sklearn.model\_selection import train\_test\_split**  **df = pd.read\_csv('Social\_Network\_Ads.csv')**  **X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=3020)**  **from sklearn.preprocessing import StandardScaler**  **sc = StandardScaler()**  **X\_train = sc.fit\_transform(X\_train)**  **X\_test = sc.transform(X\_test)**  **classifier = GaussianNB()**  **classifier.fit(X\_train, Y\_train)**  **print(classifier.predict([X\_test[0]]))**  **classifier.score(X\_test, Y\_test)** |

1. Iris Dataset

|  |
| --- |
| **from sklearn.datasets import load\_iris**  **import numpy as np**  **data2=load\_iris()**  **df1=pd.DataFrame(np.c\_[data2.data, data2.target],columns=[list(data2.feature\_names)+['target']])**  **X2=df1.iloc[:,0:-1]**  **Y2=df1.iloc[:,-1]**  **from sklearn.model\_selection import train\_test\_split**  **X2\_train,X2\_test,Y2\_train,Y2\_test=train\_test\_split(X2,Y2,test\_size=0.15,random\_state=1000)**  **classifier.fit(X2\_train, Y2\_train)**  **classifier.score(X2\_test,Y2\_test)** |

1. Breast Cancer Dataset

|  |
| --- |
| **from sklearn.datasets import load\_breast\_cancer**  **import numpy as np**  **data2=load\_breast\_cancer()**  **df1=pd.DataFrame(np.c\_[data2.data, data2.target],columns=[list(data2.feature\_names)+['target']])**  **X2=df1.iloc[:,0:-1]**  **Y2=df1.iloc[:,-1]**  **from sklearn.model\_selection import train\_test\_split**  **X2\_train,X2\_test,Y2\_train,Y2\_test=train\_test\_split(X2,Y2,test\_size=0.15,random\_state=2020)**  **classifier.fit(X2\_train, Y2\_train)**  **classifier.score(X2\_test,Y2\_test)** |

**OUTPUT**

1. Social Network Ads database

* Accuracy : 91.25%

1. iris database

* Accuracy : 91.3%

1. Breast cancer database

* Accuracy : 97.37%

**CONCLUSION**

In this practical we have implemented code for naive baies Classifier and tested the prepared model’s accuracy on various datasets. It was observed that the accuracy of classifier increases if we perform pre-processing on dataset.

**U19EC046 | ML | LAB 4**

**AIM**

Write a program to implement the K nearest neighbour classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

**ALGORITHM**

1. Import necessary libraries
2. Read the dataset using pandas
3. Split the dataset into features and results
4. Split the features and results into training and testing dataset using scikit learn

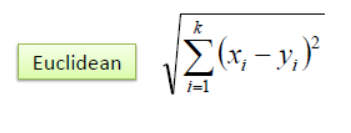
test\_train\_split

1. Apply pre-processing if needed, use scikit-learn preprocessing class
2. Create an instance of KNN classifier, tune properly taking k the numer of neighbors
3. Train the model using fit method
4. Predict the result using predict method on model
5. Find the accuracy and confusion matrix using scikit-learn matrices

**THEORY:**

K Nearest Neighbour is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure. It is mostly used to classifies a data point based on how its neighbours are classified.

In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given “unseen” observation. Similarity is defined according to a distance metric between two data points. A popular one is the Euclidean distance method



**Pros of KNN**

1. Simple to implement
2. Flexible to feature/distance choices
3. Naturally handles multi-class cases
4. Can do well in practice with enough representative data

**Cons of KNN**

1. Need to determine the value of parameter K (number of nearest neighbours)
2. Computation cost is quite high because we need to compute the distance of each query instance to all training samples.
3. Storage of data
4. Must know we have a meaningful distance function.

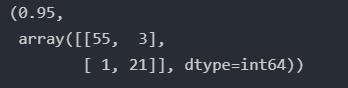
**CODE**

1. KNN classifier

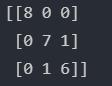
|  |
| --- |
| **from sklearn.datasets import load\_iris**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn.neighbors import KNeighborsClassifier**  **from sklearn.metrics import confusion\_matrix, accuracy\_score**  **from sklearn.preprocessing import StandardScaler, RobustScaler**  **import numpy as np**  **import pandas as pd**  **df = pd.read\_csv('Social\_Network\_Ads.csv')**  **X = df.iloc[:,:-1]**  **Y = df.iloc[:,-1]**  **x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)**  **sc = StandardScaler()**  **x\_train = sc.fit\_transform(x\_train)**  **x\_test = sc.transform(x\_test)**  **model = KNeighborsClassifier(n\_neighbors=3, metric='euclidean')**  **model.fit(x\_train, y\_train)**  **model.score(x\_test, y\_test), confusion\_matrix(y\_pred=model.predict(x\_test), y\_true=y\_test)** |

**OUTPUT**

1. KNN classifier
2. Social networking Ads



1. Iris Dataset





**CONCLUSION**

In this practical we have studied and implements KNN classifier using scikit-learn library.

**U19EC046 | ML | LAB 5**

**AIM**

Write a program to implement the Decision Tree classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Also generate the random forest classifier.

**THEORY:**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.​

The decisions or the test are performed on the basis of features of the given dataset.​ It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.**By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:​

* Information Gain​
* Gini Index​

**Information Gain**​:

* Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.​
* It calculates how much information a feature provides us about a class.​
* According to the value of information gain, we split the node and build the decision tree.​
* A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first.

**Gini Index:**

* Gini index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.​
* An attribute with the low Gini index should be preferred as compared to the high Gini index.​
* It only creates binary splits, and the CART (Classification and Regression Tree) algorithm uses the Gini index to create binary splits.

**ALGORITHM**

1. Import necessary libraries
2. Read the dataset using pandas
3. Split the dataset into features and results
4. Split the features and results into training and testing dataset using scikit learn

test\_train\_split

1. Apply pre-processing if needed, use scikit-learn preprocessing class
2. Create an instance of Decision Tree classifier using gini or entropy
3. Train the model using fit method
4. Predict the result using predict method on model
5. Find the accuracy and confusion matrix using scikit-learn matrices

**CODE**

1. Decision Tree Classification

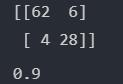
|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Social\_Network\_Ads.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)**  **from sklearn.preprocessing import StandardScaler**  **sc = StandardScaler()**  **X\_train = sc.fit\_transform(X\_train)**  **X\_test = sc.transform(X\_test)**  **from sklearn.tree import DecisionTreeClassifier**  **classifier = DecisionTreeClassifier(criterion = 'gini', random\_state = 0)    *# or use criterion = 'gini'***  **classifier.fit(X\_train, y\_train)**  **y\_pred = classifier.predict(X\_test)**  **y\_pred = classifier.predict(X\_test)**  **print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))**  **from sklearn.metrics import confusion\_matrix, accuracy\_score**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print(cm)**  **accuracy\_score(y\_test, y\_pred)** |

1. Random forest Classifier

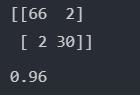
|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Social\_Network\_Ads.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)**  **from sklearn.preprocessing import StandardScaler**  **sc = StandardScaler()**  **X\_train = sc.fit\_transform(X\_train)**  **X\_test = sc.transform(X\_test)**  **from sklearn.ensemble import RandomForestClassifier**  **classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 2837)**  **classifier.fit(X\_train, y\_train)**  **y\_pred = classifier.predict(X\_test)**  **print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))**  **from sklearn.metrics import confusion\_matrix, accuracy\_score**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print(cm)**  **accuracy\_score(y\_test, y\_pred)** |

**OUTPUT**

1. Decision tree



1. Random Forest Classifier



**CONCLUSION**

In this practical we have studied and implements decision tree classifier using scikit-learn library. We also learn about different attribute selection measures and usually gini index is used as it considers both positive and negative sample.

**U19EC046 | ML | LAB 6**

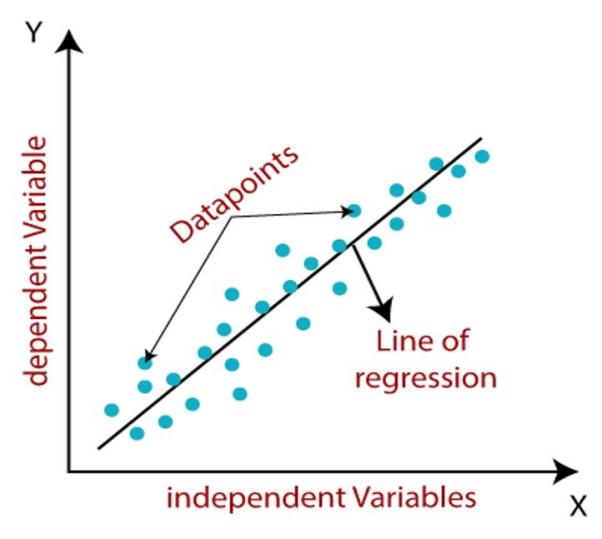
**AIM**

Write a program to implement the Linear Regression for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

**THEORY:**

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.​

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.



**Types:**

***Simple Linear Regression:****​*If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.​

**y=wx+b​**

​

***Multiple Linear regression:****​*If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.​

**y=wx1+wx2+wx3+b​**

**ALGORITHM**

1. Import necessary libraries
2. Read the dataset using pandas
3. Split the dataset into features and results
4. Split the features and results into training and testing dataset using scikit learn

test\_train\_split

1. Apply pre-processing if needed, use scikit-learn preprocessing class
2. If your dataset has categorical features encode it using scikit learn’s one hot encoder
3. Create an instance of regressor
4. Train the model using fit method
5. Predict the result using predict method on model
6. Find the r2 score and mean square error using scikit-learn matrices

**CODE**

1. Linear Regression

|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Salary\_Data.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **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 = 543)**  **from sklearn.linear\_model import LinearRegression**  **regressor = LinearRegression()**  **regressor.fit(X\_train, y\_train)**  **y\_pred = regressor.predict(X\_test)**  **from sklearn.metrics import r2\_score, mean\_squared\_error**  **score = r2\_score(y\_test, y\_pred)**  **print(f"score: {round(score\*100, 3)} %")**  **mse = mean\_squared\_error(y\_true=y\_test, y\_pred=y\_pred)**  **print(f"mean squared error: {round(mse, 3)}")** |

1. Multiple Linear Regression

|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('50\_Startups.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **from sklearn.compose import ColumnTransformer**  **from sklearn.preprocessing import OneHotEncoder**  **ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])], remainder='passthrough')**  **X = np.array(ct.fit\_transform(X))**  **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 = 755)**  **from sklearn.linear\_model import LinearRegression**  **regressor = LinearRegression()**  **regressor.fit(X\_train, y\_train)**  **y\_pred = regressor.predict(X\_test)**  **from sklearn.metrics import r2\_score, mean\_squared\_error**  **score = r2\_score(y\_test, y\_pred)**  **print(f"score: {round(score\*100, 3)} %")**  **mse = mean\_squared\_error(y\_true=y\_test, y\_pred=y\_pred)**  **print(f"mean squared error: {round(mse, 3)}")** |

1. Polynomial Regression

|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Position\_Salaries.csv')**  **X = dataset.iloc[:, 1:-1].values**  **y = dataset.iloc[:, -1].values**  **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 = 755)**  **from sklearn.linear\_model import LinearRegression**  **lin\_reg = LinearRegression()**  **lin\_reg.fit(X\_train, y\_train)**  **from sklearn.preprocessing import PolynomialFeatures**  **from sklearn.metrics import r2\_score, mean\_squared\_error**  ***# Polynomial Regression***  **poly\_reg = PolynomialFeatures(degree = 13)**  **X\_poly = poly\_reg.fit\_transform(X\_train)**  **lin\_reg\_2 = LinearRegression()**  **lin\_reg\_2.fit(X\_poly, y\_train)**  **y\_pred\_poly = lin\_reg\_2.predict(poly\_reg.fit\_transform(X\_test))**  **score\_poly = r2\_score(y\_test, y\_pred\_poly)**  **print(f"r2 score Poly: {round(score\_poly\*100, 3)} %")**  **mse\_poly = mean\_squared\_error(y\_true=y\_test, y\_pred=y\_pred\_poly)**  **print(f"mean squared error Poly: {round(mse\_poly, 3)}")**  ***# Linear Regression***  **y\_pred\_lin = lin\_reg.predict(X)**  **score\_lin = r2\_score(y, y\_pred\_lin)**  **print(f"r2 score Lin: {round(score\_lin\*100, 3)} %")**  **mse\_lin = mean\_squared\_error(y\_true=y, y\_pred=y\_pred\_lin)**  **print(f"mean squared error Lin: {round(mse\_lin, 3)}")** |

**OUTPUT**

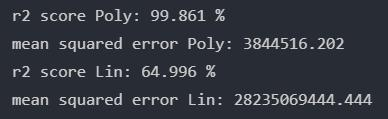
1. Linear Regression



1. Multiple Linear Regression



1. Polynomial Regression and comparing with linear regression



**CONCLUSION**

In This Practical, we have studied and implemented linear, multiple linear and polynomial regression using scikit learn library in python. It was observed the polynomial regression outperformed linear regression in non linear dataset.

**U19EC046 | ML | LAB 7**

**AIM**

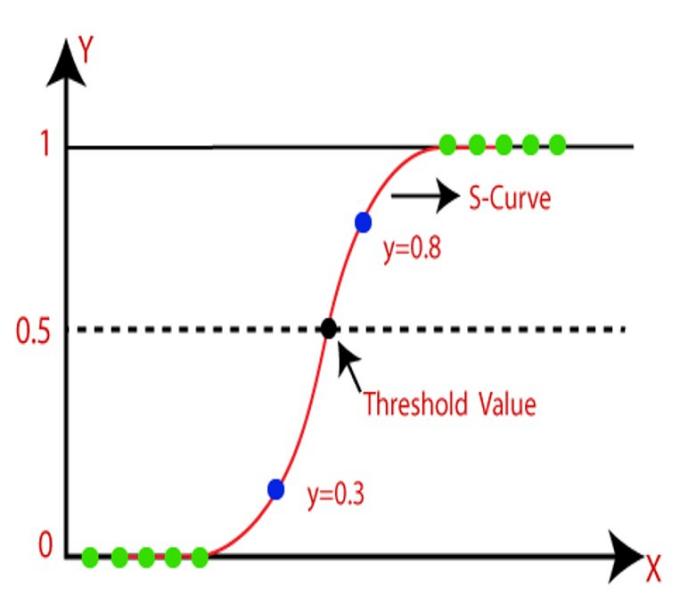
Write a program to implement the Logistic Regression for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. ​

**THEORY:**

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.​

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.​

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.



The sigmoid function is a mathematical function used to map the predicted values to probabilities.​ It maps any real value into another value within a range of 0 and 1.​

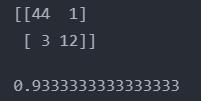
The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.​ In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

**CODE**

|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Social\_Network\_Ads.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.15, random\_state = 0)**  **from sklearn.preprocessing import StandardScaler**  **sc = StandardScaler()**  **X\_train = sc.fit\_transform(X\_train)**  **X\_test = sc.transform(X\_test)**  **from sklearn.linear\_model import LogisticRegression**  **classifier = LogisticRegression(random\_state = 100)**  **classifier.fit(X\_train, y\_train)**  **from sklearn.metrics import confusion\_matrix, accuracy\_score**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print(cm)**  **accuracy\_score(y\_test, y\_pred)** |

**OUTPUT**

1.



**CONCLUSION**

In This Practical, we have studied and implemented Logistic regression using scikit learn library in python.

**U19EC046 | ML | LAB 8**

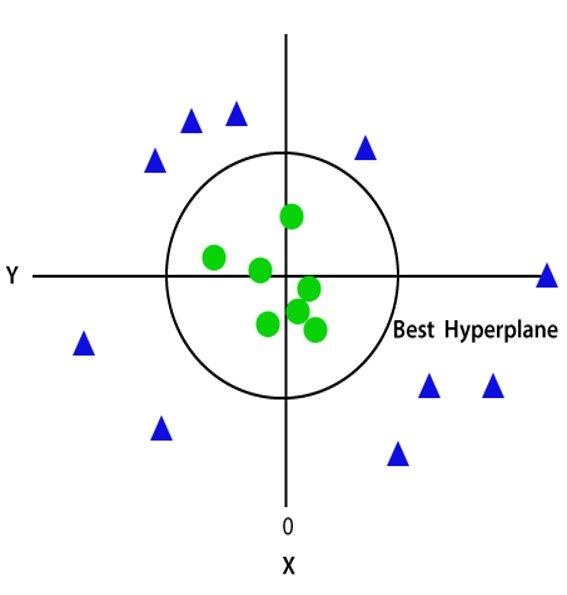
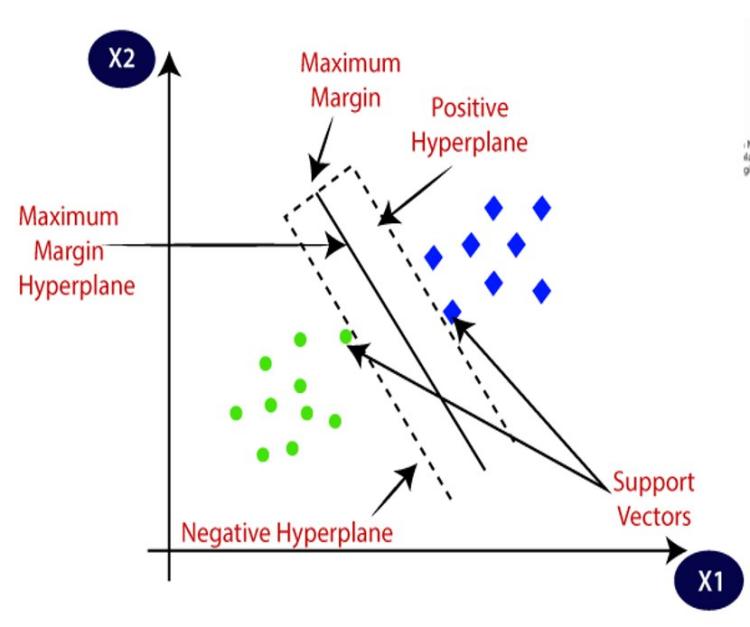
**AIM**

Write a program to implement the support vector machine for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

**THEORY:**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.​ The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.​

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.



**Types of SVM:**

***Linear SVM:*** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.​

***Non-linear SVM:*** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

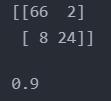
**CODE**

|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd** |
| **dataset = pd.read\_csv('Social\_Network\_Ads.csv')**  **X = dataset.iloc[:, :-1].values**  **y = dataset.iloc[:, -1].values**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)**  **from sklearn.preprocessing import StandardScaler**  **sc = StandardScaler()**  **X\_train = sc.fit\_transform(X\_train)**  **X\_test = sc.transform(X\_test)**  **from sklearn.svm import SVC**  **classifier = SVC(kernel = 'linear', random\_state = 0)**  **classifier.fit(X\_train, y\_train)**  **from sklearn.metrics import confusion\_matrix, accuracy\_score**  **cm = confusion\_matrix(y\_test, y\_pred)**  **print(cm)**  **accuracy\_score(y\_test, y\_pred)** |

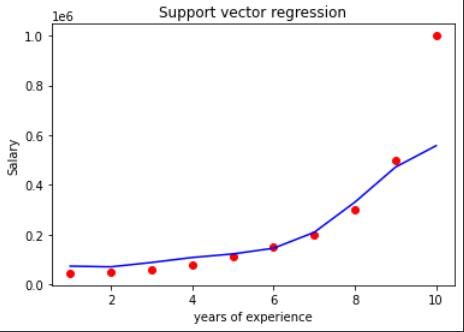
|  |
| --- |
| **import numpy as np**  **import matplotlib.pyplot as plt**  **import pandas as pd**  **dataset = pd.read\_csv('Position\_Salaries.csv')**  **X = dataset.iloc[:, 1:-1].values**  **y = dataset.iloc[:, -1].values**  **y = y.reshape(len(y),1)**  **from sklearn.preprocessing import StandardScaler**  **sc\_X = StandardScaler()**  **sc\_y = StandardScaler()**  **X = sc\_X.fit\_transform(X)**  **y = sc\_y.fit\_transform(y)**  **from sklearn.svm import SVR**  **regressor = SVR(kernel = 'rbf')**  **regressor.fit(X, y)**  **from sklearn.metrics import r2\_score, mean\_squared\_error**  **Y\_model = sc\_y.inverse\_transform(regressor.predict(X).reshape(-1,1))**  **score = r2\_score(Y\_model, sc\_y.inverse\_transform(y))**  **print(f"score: {round(score\*100, 3)} %")**  **mse = mean\_squared\_error(y\_true=y, y\_pred=Y\_model)**  **print(f"mean squared error: {round(mse, 3)}")*## Predicting a new result***  **plt.scatter(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(y), color = 'red')**  **plt.plot(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(regressor.predict(X).reshape(-1, 1)), color = 'blue')**  **plt.title('Support vector regression')**  **plt.xlabel('years of experience')**  **plt.ylabel('Salary')**  **plt.show()**  **X\_grid = np.arange(min(sc\_X.inverse\_transform(X)), max(sc\_X.inverse\_transform(X)), 0.1)**  **X\_grid = X\_grid.reshape((len(X\_grid), 1))**  **plt.scatter(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(y), color = 'red')**  **plt.plot(X\_grid, sc\_y.inverse\_transform(regressor.predict(sc\_X.transform(X\_grid)).reshape(-1,1)), color = 'blue')**  **plt.title('Truth or Bluff (SVR)')**  **plt.xlabel('Position level')**  **plt.ylabel('Salary')**  **plt.show()** |

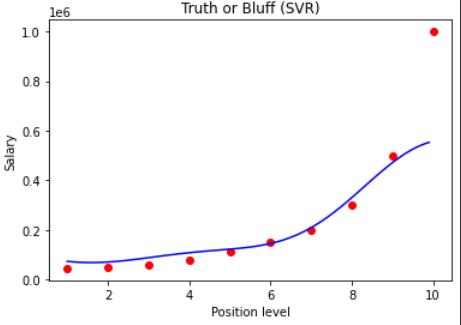
**OUTPUT**

1. SVM Classifier



1. SVM regressio





**CONCLUSION**

Hence, we have studied and implemented SVM classifier and regression using scikit learn library in python.

**U19EC046 | ML | LAB 9**

**AIM**

Write a program to implement the K-mean clustering for a sample training data set stored as a .CSV file.

**THEORY**

K- Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.

Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

* Determines the best value for K center points or centroids by an iterative process.
* Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

**STEPS**

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

**ELBOW METHOD**

This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

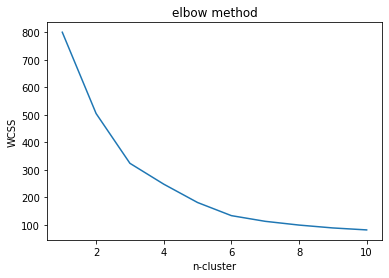
To find the optimal value of clusters, the elbow method follows the below steps:

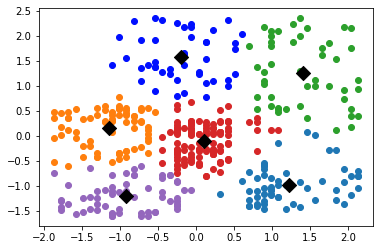
* It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
* For each value of K, calculates the WCSS value.
* Plots a curve between calculated WCSS values and the number of clusters K.
* The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

**CODE**

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  from sklearn.cluster import KMeans  from sklearn import preprocessing  data = pd.read\_csv('/content/Social\_Network\_Ads.csv')  X = data.iloc[:, 0:-1]  scaler = preprocessing.StandardScaler().fit(X)  X = scaler.transform(X)  wcss = []  for i in range(1, 11):    kmeans = KMeans(n\_clusters=i, init = 'k-means++', random\_state=42)    kmeans.fit(X)    wcss.append(kmeans.inertia\_)  plt.plot(range(1, 11), wcss)  plt.title('elbow method')  plt.xlabel('n-cluster')  plt.ylabel('WCSS')  plt.show()  kmeans = KMeans(n\_clusters=6, init = 'k-means++', random\_state=42)  kmeans.fit(X)  res = kmeans.predict(X)  centers = kmeans.cluster\_centers\_  colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#000fff']  for i, e in enumerate(X):    plt.scatter(e[0], e[1], c=colors[res[i]])  for [x, y] in centers:    plt.scatter(x, y, c='black', marker = 'D', s=100)  plt.show() |

**OUTPUT**





**CONCLUSION**

In this Practical we have studied and implemented K-means clustering algorithm and also understood how to choose the cluster size using the elbow method.

**U19EC046 | ML | LAB 10**

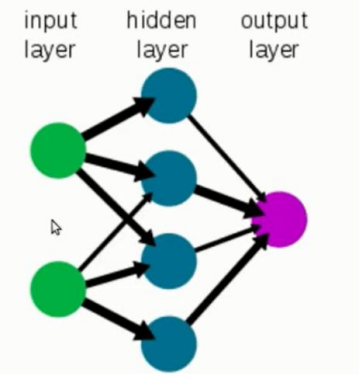
**AIM**

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

**THEORY**

*What is Artificial Neural Networks?*

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.



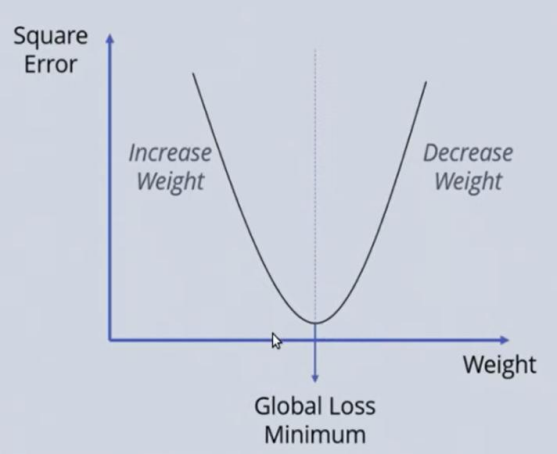
*What is Backpropagation?*

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

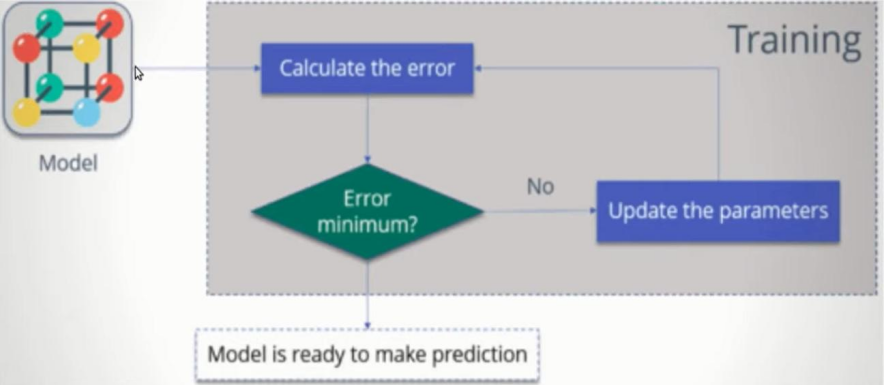
*How Backpropagation Algorithm Works*

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.



*Consider the following Back propagation neural network example diagram to understand:*

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs
5. ErrorB= Actual Output – Desired Output
6. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
7. Keep repeating the process until the desired output is achieved

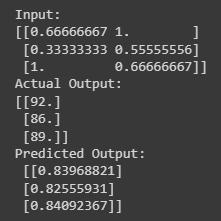


**CODE**

|  |
| --- |
| **import numpy as np**  **X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)**  **y = np.array(([92], [86], [89]), dtype=float)**  **X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100**  **#Sigmoid Function**  **def sigmoid (x):**  **return (1/(1 + np.exp(-x)))**  **#Derivative of Sigmoid Function**  **def derivatives\_sigmoid(x):**  **return x \* (1 - x)**  **#Variable initialization**  **epoch=7000 #Setting training iterations**  **lr=0.1 #Setting learning rate**  **inputlayer\_neurons = 2 #number of features in data set**  **hiddenlayer\_neurons = 3 #number of hidden layers neurons**  **output\_neurons = 1 #number of neurons at output layer**  **#weight and bias initialization**  **wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))**  **bh=np.random.uniform(size=(1,hiddenlayer\_neurons))**  **wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))**  **bout=np.random.uniform(size=(1,output\_neurons))**  **# draws a random range of numbers uniformly of dim x\*y**  **#Forward Propagation**  **for i in range(epoch):**  **hinp1=np.dot(X,wh)**  **hinp=hinp1 + bh**  **hlayer\_act = sigmoid(hinp)**  **outinp1=np.dot(hlayer\_act,wout)**  **outinp= outinp1+ bout**  **output = sigmoid(outinp)**    **#Backpropagation**  **EO = y-output**  **outgrad = derivatives\_sigmoid(output)**  **d\_output = EO\* outgrad**  **EH = d\_output.dot(wout.T)**  **hiddengrad = derivatives\_sigmoid(hlayer\_act)**  **#how much hidden layer wts contributed to error**  **d\_hiddenlayer = EH \* hiddengrad 'true', ':', 'true',**  **wout += hlayer\_act.T.dot(d\_output) \*lr**  **# dotproduct of nextlayererror and currentlayerop**  **bout += np.sum(d\_output, axis=0,keepdims=True) \*lr**  **wh += X.T.dot(d\_hiddenlayer) \*lr**  **#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr**  **print("Input: \n" + str(X))**  **print("Actual Output: \n" + str(y))**  **print("Predicted Output: \n" ,output)** |

**OUTPUT**

1.



**CONCLUSION**

In this experiment we have implemented ANN with backpropagation in python.