CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING L T P C

COURSE OBJECTIVES:

The main objectives of this course are to:

- Study about Uninformed and Heuristic search techniques.
- Learn techniques for Reasoning under uncertainty
- Introduce Machine Learning and Supervised learning algorithms
- Study about Ensembling and Unsupervised learning algorithms
- Learn the basics of Deep learning using Neural networks

PRACTICAL EXERCISES: 30 PERIODS

1. Implementation of Uninformed search algorithms (BFS, DFS)
2. Implementation of Informed search algorithms (A*, memory-bounded A*)
3. Implement naïve Bayes models
4. Implement Bayesian Networks
5. Build Regression models
6. Build decision trees and random forests
7. Build SVM models
8. Implement ensembling techniques
9. Implement clustering algorithms
10. Implement EM for Bayesian networks
11. Build simple NN models
12. Build deep learning NN models

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Use Appropriate Search Algorithms for Problem solving

CO2: Apply Reasoning under Uncertainty

CO3: Build Supervised Learning models

CO4: Build Ensembling and Unsupervised models

CO5: Build Deep Learning Neural Network models

LIST OF EXPERIMENTS

S.NO	NAME OF EXPERIMENTS	BLOOMS LEVEL	COs
1	Implementation of Uninformed search algorithms (BFS, DFS)	**L3**	**CO1**
2	Implementation of Informed search algorithms (A*, memory-bounded A*)	**L3**	**CO1**
3	Implement naïve Bayes models	**L3**	**CO2**
4	Implement Bayesian Networks	**L3**	**CO2**
5	Build Regression models	**L4**	**CO3**
6	Build decision trees and random forests	**L4**	**CO3**
7	Build SVM models	**L4**	**CO3**
8	Implement ensembling techniques	**L3**	**CO4**
9	Implement clustering algorithms	**L3**	**CO4**
10	Implement EM for Bayesian networks	**L3**	**CO4**
11	Build simple NN models	**L4**	**CO5**
12	Build deep learning NN models	**L4**	**CO5**
Content Beyond Experiment			
12	Implement clustering techniques	**L3**	**CO1**
13	Visualize data using any plotting framework	**L3**	**CO2**

EX. NO: 1 IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHMS
DATE: (BFS, DFS)

AIM

To Implement the various Uninformed search algorithms (BFS, DFS) using Python.

DEPTH FIRST SEARCH:

ALGORITHM

1. Start by putting source vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. Next, create a list of that adjacent node of the vertex.
4. Add the ones which aren't in the visited list of vertexes to the top of the stack.
5. Lastly, keep repeating steps 2 and 3 until the stack is empty.

PROGRAM

```
Graph={
‘5’ : [‘3’,’7’],
‘3’ : [‘2’,’4’],
‘7’ : [‘8’],
‘2’ :[],
‘4’ : [‘8’],
‘8’ :[]
}
Visited = set()
def dfs(visited,graph,node):
If node not in visited:
Print (node)
Visited.add(node)
For neighbour in graph[node]:
```

```
dfs(visited,grapg,neighbour)
print(:Following is the Depth-First Search")
dfs(visited,graph,'5')
```

OUTPUT:

```
Following is the Depth-First Search
5
3
2
4
8
```

BREADTH FIRST SEARCH: ALGORITHM

- Start by putting any one of the graph's vertices at the back of the queue.
- Take the front item of the queue and add it to the visited list.
- Create a list of that vertex's adjacent nodes.
- Add those which are not within the visited list to the rear of the queue.
- Keep continuing steps two and three till the queue is empty.

PROGRAM

```
Graph = {
'5' : ['3','7'],
'3' : ['2','4'],
'7' : ['8'],
'2' :[],
'4' : ['8'],
'8' :[]
}
Visited = []
Queue = []
def bfs(visited,graph,node):
visited.append(node)
while queue:
m= queue.pop(0)
print(m,end = " ")
for neighbour in graph[m]:
if neighbour not in visited:
visited.append(neighbor)
queue.append(neighbor)
```

OUTPUT:

Following is the Breadth-First Search

5 3 7 2 4 8

RESULT:

Thus the Implementation of the various Uninformed search algorithms such as BFS and DFS using Python was performed and output was verified.

EX. NO: 2 IMPLEMENTATION OF INFORMED SEARCH ALGORITHMS

DATE: (A* SEARCH)

AIM

To Implement the various informed search algorithms such as A*, memory-bounded A* using Python.

ALGORITHM FOR A*

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

PROGRAM:

```
Def aStarAlgo(start_node,stop_node):
open_set = set(start_node)
        closed_set = set()
        g = {}
        parents = {}
        g[start_node] = 0
parents[start_node] = start_node
while len(open_set)>0:
n=None
for v in open_set:
if n = = None or g[v] + heuristic(v) < g[n] + heuristic(n):
n = v
if n = = stop_node or Graph_nodes[n] = = None:
pass
else:
        for(m,weight) in get_neighbors(n):
          if m not in open_set and m not in
                closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
          else:
```

```
                    if g[m] > g[n] + weight:
                       g[m] = g[n] + weight
                       parents[m] = n
                    if m in closed_set:

                    closed_set.remove(m)
                       open_set.add(m)
            if n = = None:
               print('Path does not exist!')
               return None
            if n = = stop_node:
               path = [ ]
               while parents [n] !=n:
              path.append(n)
              n = parents [n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
 open_set . remove(n)
 closed_set . add(n)
 print ('Path does not exist !')
 return None
 def get_neighbors(v):
 if v in Graph_nodes:
            return Graph_nodes [v]
 else:
            return None
 def get_neighbors(v):
 if v in Graph_nodes:
 return Graph_nodes [v]
 else:
            return None
def heuristic(n):
            H_dist = {
            'A' : 11,
            'B' : 6,
            'C' : 99,
            'D' : 1,
            'E' : 7,
            'G' : 0,
 }
 return H_dist[n]
 Graph_nodes = {
 'A' : [('B',2) , ('E',3)],
 'B' : [('C',1) , ('G',9)],
 'C' : None,
 'E' : [('D',6)],
 'D' : [('G',1)],
 }
 print ('Search by A* Algorithm:')
 aStarAlgo('A','G')
```

OUTPUT:

Search by A* Algorithm:

Path Found: ['A','E','D','G']

RESULT:

Thus the Implementation of the informed search algorithm such as A* search using Python was performed and output was verified.

EX.NO:3 **IMPLEMENT NAIVE BAYES MODELS**

DATE:

AIM

To Implement the Naïve Bayes Model using Python.

ALGORITHM

Step 1 : Loading Initial Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

Step 2 : Importing Dataset

Step 3 : Exploring Dataset

Step 4 : Visualizing Dataset

Step 5: Preprocessing the Dataset

Step 6 : Data Normalization

Step 7 : Test Train Split

Step 8 : Sklearn Gaussian Naïve Bayes Model

Step 9 : Find the Accuracy of Naïve Bayes Model

PROGRAM:

```
import math
import random
import csv
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata
def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    test = list(mydata)
```

```
    while len(train) < train_num:
        index = random.randrange(len(test))
            train.append(test.pop(index))
    return train, test
def groupUnderClass(mydata):
      dict = {}
      for i in range(len(mydata)):
          if (mydata[i][-1] not in dict):
              dict[mydata[i][-1]] = []
          dict[mydata[i][-1]].append(mydata[i])
      return dict
def mean(numbers):
    return sum(numbers) / float(len(numbers))
def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)
 def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    del info[-1]
    return info
def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info
def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions
def accuracy_rate(test, predictions):
```

```
correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0
 filename =r'E:\user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv'
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
    mydata[i] = [float(x) for x in mydata[i]]
 ratio = 0.7

train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))
info =MeanAndStdDevForClass(train_data)
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

OUTPUT:

Total number of examples are: 200

Out of these, training examples are: 140

Test examples are: 60

Accuracy of your model is: 71.2376788

RESULT:

Thus the Implementation of Naïve Bayes classification using Python was performed and output was verified.

EX. NO: 4 **IMPLEMENT BAYESIAN NETWORKS**

DATE:

AIM

To implement the Bayesian Networks using Python.

ALGORITHM

Step 1 : Loading Initial Libraries

import numpy as np

import pandas as pd

import csv

Step 2 : Download datasets

Step 3 : Creation the Bayesian Network

Step 4 : Inference from the Bayesian Network

Step 5: Prediction using Bayesian Network

PROGRAM:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
form pgmpy.inference import VariableElimation
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan')
print('sample instances from the dataset are given below')
print(heartDisease.dtypes)
model = BayesianModel([('age',heartdisease'),('sex', heartdisease'),('exang',heartdisease'),('cp',
heartdisease'),(' heartdisease',restecg'),(' heartdisease','chol')])
print ('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator = MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
Print('\n1.Probability of HeartDisease given evidence = restecg')
q1 = HeartDiseasetest_infer.query(variables = ['heartdisease'],evidence={'restecg':1})
print(q1).
print('\n2. Probability of HeartDisease given evidence = cp')
q2 = HeartDiseasetest_infer.query(variables = ['heartdisease'],evidence= {'cp:2}) print(
```

OUTPUT:

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence = restecg

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

2. Probability of HeartDisease given evidence = cp

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

RESULT:

Thus the Implementation of Bayesian Network using Python was performed and output was verified.

EX. NO: 5 IMPLEMENTATION OF REGRESSION MODELS

DATE:

AIM

To Implement the Linear Regression Model using Python.

ALGORITHM

Step 1 : Import the modules.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Step 1 : Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Step 2 : Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Step 3 : Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed: def myfunc(x):

```
return slope * x + intercept
```

Step 4 : Run each value of x through function. This result in new array with new values for y-axis:

```
mymodel = list(map(myfunc, x))
```

Step 5 : Draw the original scatter plot:

```
plt.scatter(x, y)
```

Step 6 : Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Step 7 : Display the diagram:

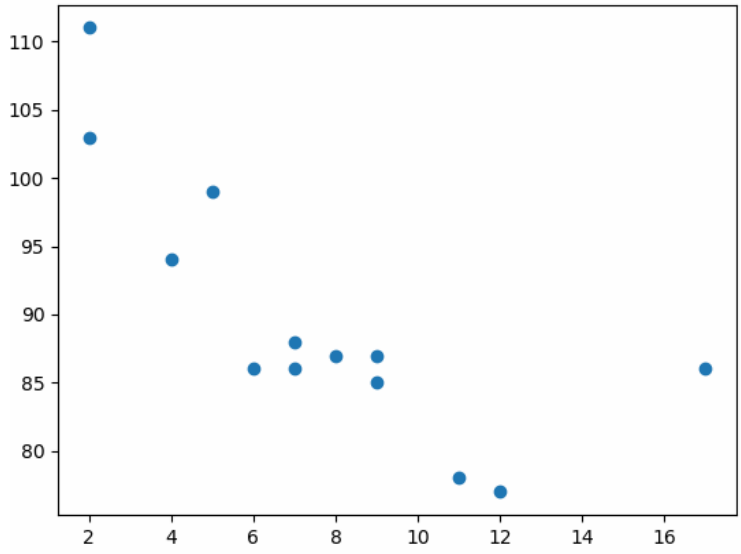
```
plt.show()
```

PROGRAM:

Start by drawing a scatter plot:

```
import matplotlib.pyplot
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)

plt.show()
```

OUTPUT:

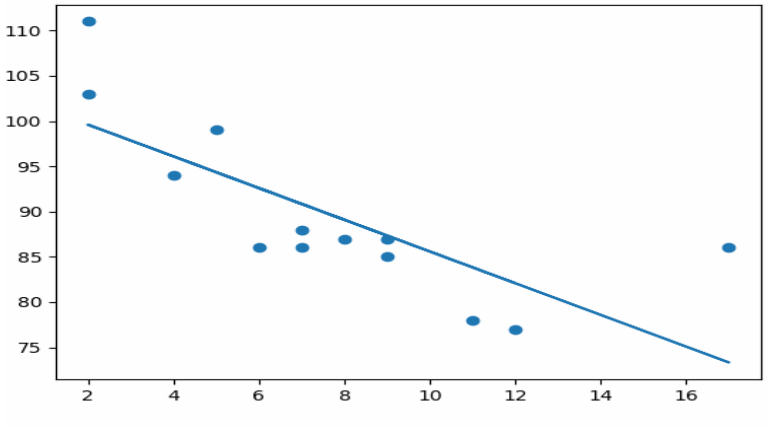


PROGRAM:

Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt

from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):

        return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

OUTPUT:



RESULT:

Thus the Implementation of Regression Models using Python was performed and output was verified.

EX. NO: 6 BUILDING OF DECISION TREES AND RANDOM FORESTS

DATE:

AIM

To Implement the Decision Trees and Random Forests using Python.

ALGORITHM

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure

(ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3.

Continue this process until a stage is reached where you cannot further classify the nodes

and called the final node as a leaf node.

PROGRAM:

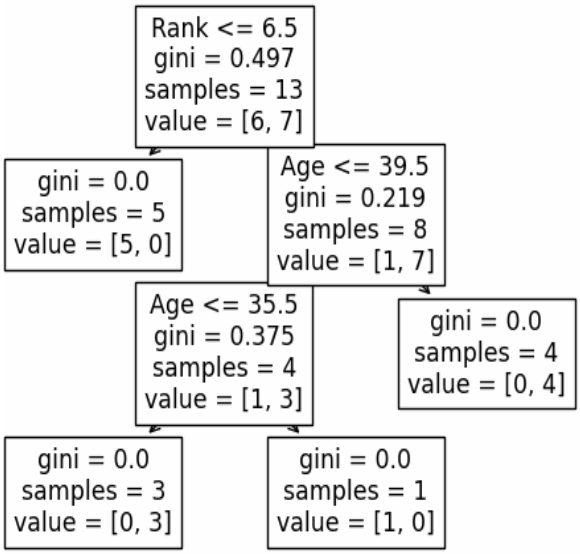
```
import sys
import matplotlib
matplotlib.use('Agg')
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
df = pandas.read_csv("data.csv")
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
```

```
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
features = ['Age', 'Experience', 'Rank', 'Nationality']


X = df[features]
y = df['Go']
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)
tree.plot_tree(dtree, feature_names=features)
#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

INPUT: data.csv

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

OUTPUT:



RESULT:

Thus the Building of Decision Trees and Random Forests using Python was performed and output was verified.

EX. NO: 7 **BUILD SVM MODELS**

DATE:

AIM:

To build SVM models using phython.

ALGORITHM:

Step 1: Import Libraries

Step 2: Load the Dataset

Step 3: Split the datasets into x and y , where x is Independent variables and y is dependent variables

Step 4: Split the x and y datasets into the training set and test set

Step 5: Perform feature Scaling

Step 6: Fit SVM to the training set

Step 7: Predict the test set results

Step 8: Make the confusion matrix

Step 9: Visualize the test set results.

PROGRAM:

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
x=dataset.iloc[:,[0,1]].values
y=dataset.iloc[:,2].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 = 'rbf',random_state = 0)
classifier.fit(x_train,y_train)
y_pred = classifier.predict(x_test)
print(y_pred)
accuracy_score(y_test,y_pred)
```

OUTPUT:

[00000001010000010010010101000000010000
00100001000101100011100100101010000010001
00001111001001100100000111]

RESULT:

Thus the Building of SVM using Python was performed and output was verified.

EX. NO: 8 **IMPLEMENT ENSEMBLE TECHNIQUES**

DATE:

AIM:

To implement ensemble techniques using python.

ALGORITHM:

Step 1: Split the train dataset into n parts

Step 2: A base model (say linear regression) is fitted on n-1 parts and predictions are made for the nth part. This is done for each one of the n part of the train set.

Step 3:The base model is then fitted on the whole train dataset.

Step 4:This model is used to predict the test dataset.

Step 5:The Steps 2 to 4 are repeated for another base model which results in another set of predictions for the train and test dataset.

Step 6:The predictions on train data set are used as a feature to build the new model.

Step 7:This final model is used to make the predictions on test dataset

PROGRAM:

```
importing utility modules

importpandas as pd

fromsklearn.model_selection importtrain_test_split

fromsklearn.metrics importmean_squared_error

fromsklearn.ensemble importRandomForestRegressor

importxgboost as xgb

fromsklearn.linear_model importLinearRegression

fromvecstack importstacking

df =pd.read_csv("train_data.csv")

target =df["target"]

train =df.drop("target")

X_train, X_test, y_train, y_test =train_test_split(train, target, test_size=0.20)

model_1 =LinearRegression()

model_2 =xgb.XGBRegressor()

model_3 =RandomForestRegressor()

all_models =[model_1, model_2, model_3]

s_train, s_test =stacking(all_models, X_train, X_test,y_train, regression=True, n_folds=4)

final_model =model_1
```

final_model =final_model.fit(s_train, y_train) pred_final =final_model.predict(X_test)
print(mean_squared_error(y_test, pred_final))

OUTPUT:

4510

RESULT:

Thus the implementation of ensemble techniques was performed and output was verified.

EX. NO: 9 **IMPLEMENTING CLUSTERING ALGORITHMS**
DATE:

AIM:

To implementing the clustering algorithm using python.

ALGORITHM:

Step 1: Select the value of K to decide the number of clusters to be formed.

Step 2: Select random K points which will act as centroids.

Step 3: Assign each data point, based on their distance from the randomly selected points(centroid), to the nearest cluster/closet centroid which will form the predefined clusters.

Step 4: Place a new centroid of each cluster.

Step 5: Repeat step 3, which reassign each datapoint to the new closet centroid of each cluster.

Step 6: If any reassignment occurs, then go to step 4 else go to step 7.

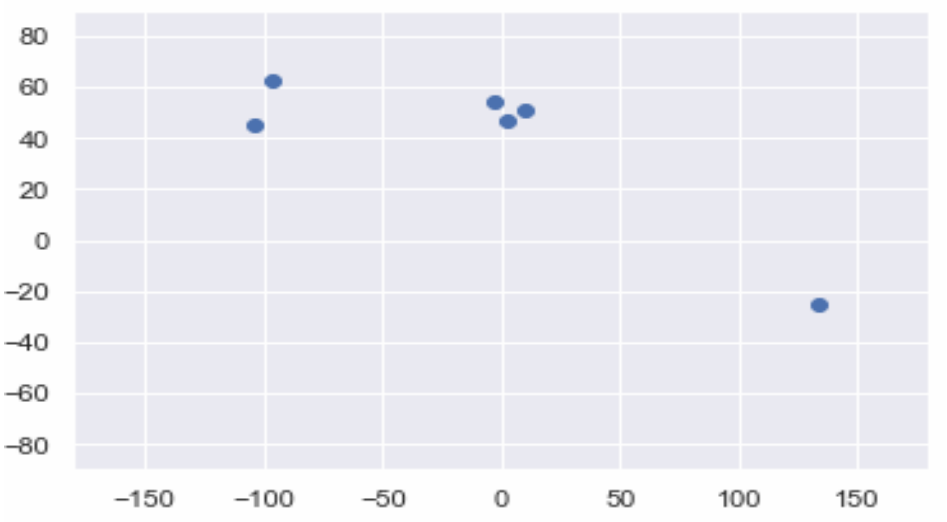
PROGRAM:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import Kmeans
```

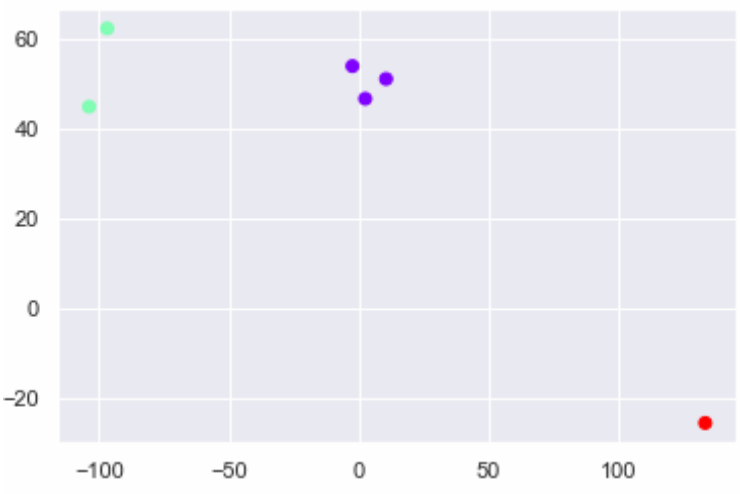
```
data = pd.read_CSV('Countryclusters.CSV')

data

x = data.iloc[:,1:3]

x

kmeans = KMeans(3)

means.fit(x)

identified_clusters = kmeans.fit_predict(x)

identified_clusters

array([1,1,0,0,0,2])

data_with_clusters = data.copy()

data_with_clusters['Clusters'] = identified_clusters

plt.scatter(data_with_clusters['Longitude'],data_with_clusters['Latitude'],c =
data_with_clusters['Clusters'],cmap = 'rainbow')

WCSS=[]

for i in range (1,7):

kmeans = Kmeans(i)

kmeans.fit(x)

WCSS_iter = kmeans.inertia_WCSS.append(WCSS_iter)

number_clusters = range(1,7)

plt.plot(number_clusters,WCSS)

plt.tittle('The Elbow tittle')

plt.xlabel('Number of clusters')

plt.ylable('WCSS')
```

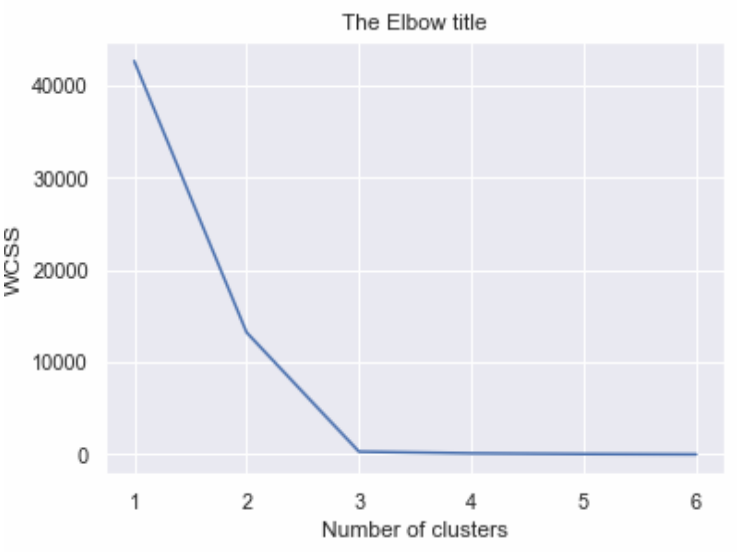
OUTPUT:

s.no	Country	Latitude	Longitude	Language
0	USA	44.97	-103.77	English
1	Canada	62.40	-96.80	English
2	France	46.75	2.40	French
3	UK	54.01	-2.53	English
4	Germany	51.15	10.40	German
5	Australia	-25.45	133.11	English



	Latitude	Longitude
0	44.97	-103.77
1	62.40	-96.80
2	46.75	2.40
3	54.01	-2.53
4	51.15	10.40
5	-25.45	133.11





This method shows that 3 is a good number of clusters.

RESULT:

Thus the implementation of clustering algorithm was performed and output was verified.

EX. NO: 10 IMPLEMENTATION OF EM FOR BAYESIAN NETWORKS

DATE:

AIM:

To implementing the EM for Bayesian Networks.

ALGORITHM:

Step 1: To estimate the missing or latent variables.

Step 2: To optimize the parameters of the models

Step 3: To initialize the parameter values.

Step 4: The E-step is used to estimate the values of the missing or incomplete data.

Step 5: The M-step is used to complete the data obtained from the parameter values present in Step 4.

Step 6: To check if the values of latent variables are covariance or not.

PROGRAM:

```
def GMM_sklearn(x, weights = None, means = None, covariances = None):

model = GaussianMixture(n_components=2,covariance_type = 'full',tol =0.01,max_iter =
1000,weights_init =weights,means_init = means,precisions_init = covariances)

model.fit(x)

print("\n scikit learn:\n \t phi:%s \n\tmu_0: %s \n\tmu_1: %s \n\tsigma_0: %s \n\tsigma_1: %s"
%(model.weights_[1],model.means_[0,:],model.means_[1,:],model.covariances_[0,:],model.covarianc
es_[1,:]))

return model.predict(x),model.predict_prob(x)[:,1]

learned_params = learn_params(x_labled,y_labled)

weights = [1 – learned_params["phi'],learned_params["phi']]

means = [learned_params["mu0'],learned_params["mu1']]

covariances = [learned_params["sigma0"],learned_params["sigma1"]]

sklearn_forecasts,posterior_sklearn = GMM_sklearn(x_unlabeled,weights,means,covariances)

output_df = pd.DataFrame({'semisupervised_forecasts':  semisupervised_forecasts,

                          'semisupervised_posterior':  semisupervised_posterior[:,1],

 'sklearn_forecasts': sklearn_forecasts, 'posterior_sklearn;:

 posterior_sklearn})

print("\n%s%% of forecasts matched." % (output_df["semisupervised_forecasts"] = =
output_df["sklearn_forecasts"]].shape[0]/output_df.shape[0]*100))
```

OUTPUT:

Semi-supervised:

phi:0.586349881794546

mu_0: [-1.04546727 -1.02704636]

mu_1: [0.98763329 0.99661118]

sigma_0: [[0.36018609 0.30853357][0.30853357 0.75384027]]

sigma_1: [[0.7196797 0.1437903][0.1437903 0.30853791]]

total steps: 4

scikit learn:

phi:0.59647894226803

mu_0: [-1.06169376 -1.0563389]

mu_1: [0.96408565 0.98206315]

sigma_0: [[0.35027155 0.29629092][0.29629092 0.73083581]]

sigma_1: [[0.74510804 0.16156928][0.16156928 0.32021029]]

99.4% of forecasts matched

RESULT:

Thus the implementation of EM for Bayesian Networks was performed and output was verified.

EX. NO: 11 **BUILD SIMPLE NN MODELS**

DATE:

AIM:

To Build simple NN Models using python.

ALGORITHM:

Step 1: Import NumPy, Scikit-learn and Matplotlib.

Step 2: Create a Training and Test Data Set.

Step 3: Scale the Data.

Step 4: Create a Neural Network Class.

Step 5: Create an Initialize Function.

Step 6: Create a Forward Propagation Function.

Step 7: Create a Backward Propagation Function.

Step 8: Create a Training Function.

Step 9: Plot the Mean Absolute Error Development.

Step 10: Calculate the Accuracy and its Components.

PROGRAM:

```
from numpy import exp, array, random, dot
class NeuralNetwork():
def __init__(self):
random.seed(1)
self.synaptic_weights = 2 * random.random((3, 1)) – 1
```

```
def __sigmoid(self, x):
return 1 / (1 + exp(-x))
def __sigmoid_derivative(self, x):
return x * (1 - x)
def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
for iteration in xrange(number_of_training_iterations):
output = self.think(training_set_inputs)
error = training_set_outputs – output
adjustment = dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))
self.synaptic_weights += adjustment
def think(self, inputs):
return self.__sigmoid(dot(inputs, self.synaptic_weights))
if __name___== "__main__":
neural_network = NeuralNetwork()
print "Random starting synaptic weights: "
print neural_network.synaptic_weights
training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

training_set_outputs = array([[0, 1, 1, 0]]).T
neural_network.train(training_set_inputs, training_set_outputs, 10000)
print "New synaptic weights after training: "
print neural_network.synaptic_weights
print "Considering new situation [1, 0, 0] -> ?: "
print neural_network.think(array([1, 0, 0]))
```

OUTPUT:

Random starting synaptic weights:[[-0.16595599][0.44064899][-0.99977125]]

New synaptic weights after training:[[9.67299303][-0.2078435][-4.62963669]]

Considering new situation [1, 0, 0] -> ?: [0.99993704]

RESULT:

Thus the Building of simple NN Models was performed and output was verified.

EX. NO: 12 **BUILD DEEPLEARNING NN MODELS**
DATE:

AIM:

To Build Deep learning NN Models using python.

ALGORITHM:

Step 1: Data Pre-processing.

Step 2: Separating your Training and Testing Datasets.

Step 3: Transforming the Data.

Step 4: Building the Artificial Neural Network.

Step 5: Running Predictions on the Test Set.

Step 6: Checking the Confusion Matrix.

Step 7: Making single Prediction.

Step 8: Improving the Model Accuracy.

Step 9: Adding Dropout Regularization to Fight Over-Fitting

Step 10: Hyperparameter Tuning.

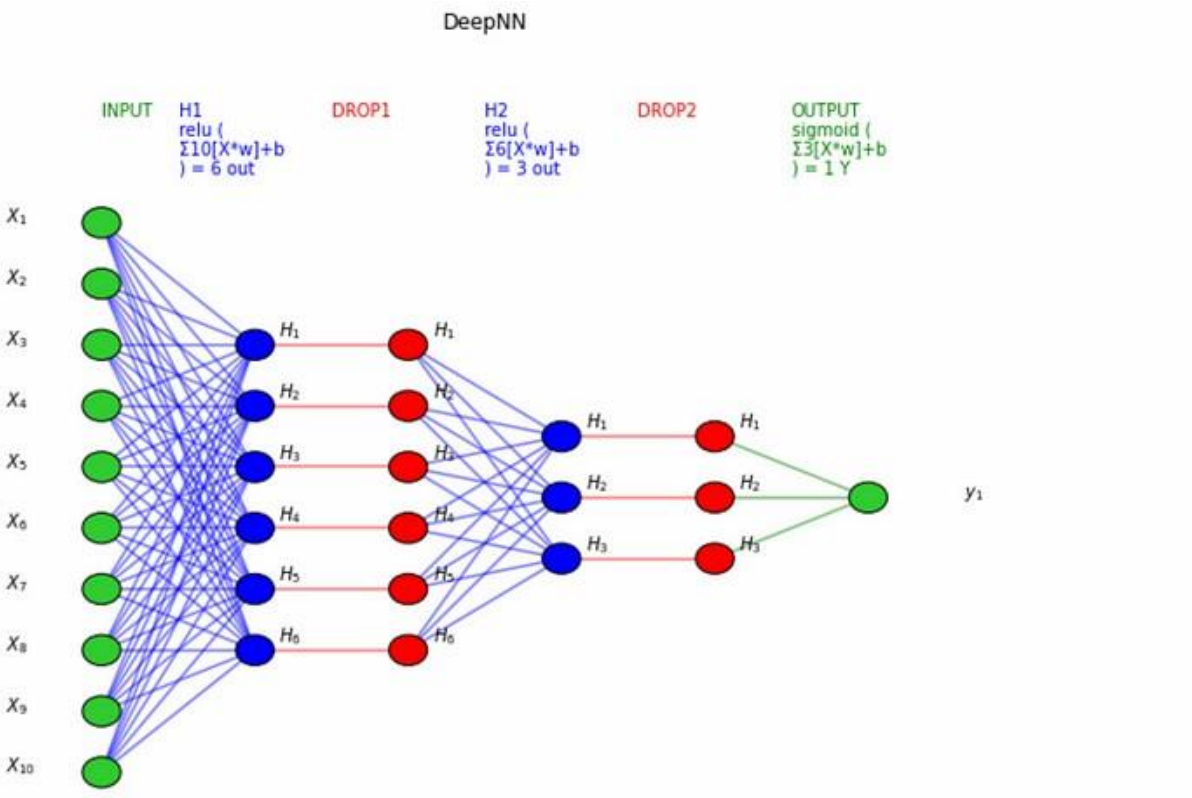
PROGRAM:

```
def utils_nn_config(model):
lst_layers = []
if "Sequential" in str(model):
layer = model.layers[0]
lst_layers.append({"name":"input", "in":int(layer.input.shape[-1]), "neurons":0,
"out":int(layer.input.shape[-1]), "activation":None, "params":0, "bias":0})
    for layer in model.layers:
        try:
            dic_layer = {"name":layer.name, "in":int(layer.input.shape[-1]),
"neurons":layer.units,"out":int(layer.output.shape[-1]), "activation":layer.get_config()["activation"],
"params":layer.get_weights()[0], "bias":layer.get_weights()[1]}
except:
            dic_layer = {"name":layer.name, "in":int(layer.input.shape[-1]), "neurons":0,
"out":int(layer.output.shape[-1]), "activation":None, "params":0, "bias":0}
lst_layers.append(dic_layer)
    return lst_layers
def visualize_nn(model, description=False, figsize=(10,8)):
lst_layers = utils_nn_config(model)
    layer_sizes = [layer["out"] for layer in lst_layers]
fig = plt.figure(figsize=figsize)
ax = fig.gca()
    ax.set(title=model.name)
ax.axis('off')
left, right, bottom, top = 0.1, 0.9, 0.1, 0.9
    x_space = (right-left) / float(len(layer_sizes)-1)
    y_space = (top-bottom) / float(max(layer_sizes))
    p = 0.025
    for i,n in enumerate(layer_sizes):
top_on_layer = y_space*(n-1)/2.0 + (top+bottom)/2.0
```

```
layer = lst_layers[i]
color = "green" if i in [0, len(layer_sizes)-1] else "blue"
color = "red" if (layer['neurons'] == 0) and (i > 0) else color
if (description is True):
d = i if i == 0 else i-0.5
if layer['activation'] is None:
plt.text(x=left+d*x_space, y=top, fontsize=10, color=color, s=layer["name"].upper())
else:
plt.text(x=left+d*x_space, y=top, fontsize=10, color=color, s=layer["name"].upper())
plt.text(x=left+d*x_space, y=top-p, fontsize=10, color=color, s=layer['activation']+" (")
plt.text(x=left+d*x_space, y=top-2*p, fontsize=10, color=color, s="Σ"+str(layer['in'])+"[X*w]+b")
out = " Y"  if i == len(layer_sizes)-1 else " out"
plt.text(x=left+d*x_space, y=top-3*p, fontsize=10, color=color, s=") = "+str(layer['neurons'])+out)
for m in range(n):
color = "limegreen" if color == "green" else color
        circle = plt.Circle(xy=(left+i*x_space, top_on_layer-m*y_space-4*p), radius=y_space/4.0, color=color,
ec='k', zorder=4)
if i == 0:
ax.add_artist(circle)
plt.text(x=left-4*p, y=top_on_layer-m*y_space-4*p, fontsize=10, s=r'$X_{'+str(m+1)+'}$')
elif i == len(layer_sizes)-1:
plt.text(x=right+4*p, y=top_on_layer-m*y_space-4*p, fontsize=10, s=r'$y_{'+str(m+1)+'}$')
else:
            plt.text(x=left+i*x_space+p, y=top_on_layer-m*y_space+(y_space/8.+0.01*y_space)-4*p,
fontsize=10, s=r'$H_{'+str(m+1)+'}$')
   for i, (n_a, n_b) in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
layer = lst_layers[i+1]
color = "green" if i == len(layer_sizes)-2 else "blue"
color = "red" if layer['neurons'] == 0 else color
layer_top_a = y_space*(n_a-1)/2. + (top+bottom)/2. -4*p
layer_top_b = y_space*(n_b-1)/2. + (top+bottom)/2. -4*p
for m in range(n_a):
for o in range(n_b):
line = plt.Line2D([i*x_space+left, (i+1)*x_space+left], [layer_top_a-m*y_space, layer_top_b-o*y_space],
c=color, alpha=0.5)
if layer['activation'] is None:
```

```
if o == m:
ax.add_artist(line)
else:
ax.add_artist(line)
plt.show()
```

OUTPUT:



RESULT:

Thus the Building of deep learning NN Model was performed and output was verified.