RAJARAJESWARI COLLEGE OF ENGINEERING

No:14, Ramohalli Cross, Kumbalgodu, Mysore Road, Bangalore-560 074

A MANUAL FOR

Artificial Intelligence and Machine Learning Laboratory

(18CSL76)

VII SEMESTER

PREPARED BY

Mr. SHASHIDHAR V
Asst. Professor
Dept. of Computer Science & Engineering
R.R.C.E, Bangalore

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2022-23

18CSL76-Artificial Intelligence and Machine Learning Laboratory

1. Implement A* Search algorithm.
2. Implement AO* Search algorithm.
3. 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.
4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.
5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
6. Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
8. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Program-1: Implement A* Search algorithm.

```
from heuristicsearch.a_star_search import AStar
aj_list={'A': [('B', 6), ('F', 3)],
         'B': [('C', 3), ('D', 2)],
         'C': [('D', 1), ('E', 5)],
         'D': [('C', 1),('E', 8)],
         'E': [('I', 5), ('J', 5)],
         'F': [('G', 1),('H', 7)] ,
         'G': [('I', 3)],
         'H': [('I', 2)],
         'I': [('E', 5), ('J', 3)],}

heuristics={'A': 10,'B': 8,'C': 5,'D': 7,'E': 3,'F': 6,'G':
5,'H': 3,'I': 1,'J': 0}
graph=AStar(aj_list,heuristics)
graph.apply_a_star(start='A',stop='J')
```

Output:

```
Path
A -> F -> G -> I -> J
Cost
0 -> 3 -> 4 -> 7 -> 10
```

Program-2: Implement AO* Search algorithm.

```python
from heuristicsearch.ao_star import AOStar
print("Graphs-1")
heuristic={'A':1,'B':6,'C':2,'D':12,'E':2,'F':1,'G':5,'H':7,'J':
1,'T':3}
aj_list={'A':[[('B',1),('C',1)],[('D',1)]],
         'B':[[('G',1)],[('H',1)]],
         'C':[[('J',1)]],
         'D':[[('E',1),('F',1)]],
         'G':[[('I',1)]]
         }
graph=AOStar(aj_list,heuristic,'A')
graph.applyAOStar()
```

Output:

```
Graphs-1
PROCESSING NODE : A
-----------------------------------------------------------------
10 ['B', 'C']

PROCESSING NODE : B
-----------------------------------------------------------------
6 ['G']

PROCESSING NODE : A
-----------------------------------------------------------------
10 ['B', 'C']

PROCESSING NODE : G
-----------------------------------------------------------------
4 ['I']

PROCESSING NODE : B
-----------------------------------------------------------------
5 ['G']

PROCESSING NODE : A
-----------------------------------------------------------------
9 ['B', 'C']

PROCESSING NODE : I
-----------------------------------------------------------------
0 []

PROCESSING NODE : G
```

```
-----------------------------------------------------------------
1 ['I']

PROCESSING NODE : B
-----------------------------------------------------------------
2 ['G']

PROCESSING NODE : A
-----------------------------------------------------------------
6 ['B', 'C']

PROCESSING NODE : C
-----------------------------------------------------------------
2 ['J']

PROCESSING NODE : A
-----------------------------------------------------------------
6 ['B', 'C']

PROCESSING NODE : J
-----------------------------------------------------------------
0 []

PROCESSING NODE : C
-----------------------------------------------------------------
1 ['J']

PROCESSING NODE : A
-----------------------------------------------------------------
5 ['B', 'C']

FOR THE SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
-------------------------------------------------------------
{'I':  [],  'G':  ['I'],  'B':  ['G'],  'J':  [],  'C':  ['J'],  'A':
['B', 'C']}
-------------------------------------------------------------
```

Program-3: 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.

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('play2.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    print(" steps of Candidate Elimination Algorithm",i+1)
    print("Specific_h ",i+1,"\n ")
    print(specific_h)
    print("general_h ", i+1, "\n ")
    print(general_h)
    indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Output:

Step 1 of Candidate Elimination Algorithm

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm

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

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm

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

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

Step 5 of Candidate Elimination Algorithm

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

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

Final Specific hypothesis:

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

Final General hypothesis:

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

Program-4: Decision Tree ID3 Algorithm Machine Learning

```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = 
df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy

def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = 
len(df[attribute][df[attribute]==variable][df[Class]
==target_variable])
            den = len(df[attribute][df[attribute]==variable])
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)

def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-
find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node,value):
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df,tree=None):
    Class = df.keys()[:-1]
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree={}
```

```
        tree[node] = {}
    for value in attValue:
        subtable = get_subtable(df,node,value)
        clValue,counts =
np.unique(subtable['play'],return_counts=True)
        if len(counts)==1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree

import pandas as pd
import numpy as np
eps = np.finfo(float).eps
from numpy import log2 as log
df = pd.read_csv('play2.csv')
print("\n Given Play Tennis Data Set:\n\n",df)
tree= buildTree(df)
import pprint
pprint.pprint(tree)

"""test={'Outlook':'Sunny','Temperature':'Hot','Humidity':'High'
,'Wind':'Weak'}
def func(test, tree, default=None):
    attribute = next(iter(tree))
    print(attribute)
    if test[attribute] in tree[attribute].keys():
        print(tree[attribute].keys())
        print(test[attribute])
        result = tree[attribute][test[attribute]]
        if isinstance(result, dict):
            return func(test, result)
        else:
            return result
    else:
        return default
ans = func(test, tree)
print(ans)
"""
```

Output:

```
Given Play Tennis Data Set:

      Outlook Temperature Humidity    Wind play

0      Sunny         Hot     High    Weak   No
```

```
1       Sunny          Hot      High  Strong   No
2    Overcast          Hot      High    Weak  Yes
3        Rain         Mild      High    Weak  Yes
4        Rain         Cool    Normal    Weak  Yes
5        Rain         Cool    Normal  Strong   No
6    Overcast         Cool    Normal  Strong  Yes
7       Sunny         Mild      High    Weak   No
8       Sunny         Cool    Normal    Weak  Yes
9        Rain         Mild    Normal    Weak  Yes
10      Sunny         Mild    Normal  Strong  Yes
11   Overcast         Mild      High  Strong  Yes
12   Overcast          Hot    Normal    Weak  Yes
13       Rain         Mild      High  Strong   No
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Program-5: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
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)
y=y/100
def sigmoid(x):
     return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
     return x*(1-x)
epoch=7000
lr=0.25
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neuron
s))
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))
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)
     EO=y-output
     outgrad=derivatives_sigmoid(output)
     d_output=EO*outgrad
     EH=d_output.dot(wout.T)
     hiddengrad=derivatives_sigmoid(hlayer_act)
     d_hiddenlayer=EH*hiddengrad
     wout+=hlayer_act.T.dot(d_output)*lr
     wh+=X.T.dot(d_hiddenlayer)*lr
print("Input=\n"+str(X))
print("Actual output:\n"+str(y))
print("predicated output:",output)
```

Output:

```
Input=
[[0.66666667 1.        ]
```

```
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output:
[[0.92]
 [0.86]
 [0.89]]
predicated output: [[0.89494013]
 [0.87827289]
 [0.896573  ]]
```

Program-6: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = pd.read_csv('tennis.csv')
print("The first 5 Values of data is :\n", data.head())

X = data.iloc[:, :-1]
print("\nThe First 5 values of the train attributes is\n",
X.head())

Y = data.iloc[:, -1]
print("\nThe First 5 values of target values is\n", Y.head())

obj1= LabelEncoder()
X.Outlook = obj1.fit_transform(X.Outlook)
print("\n The Encoded and Transformed Data in Outlook
\n",X.Outlook)

obj2 = LabelEncoder()
X.Temperature = obj2.fit_transform(X.Temperature)

obj3 = LabelEncoder()
X.Humidity = obj3.fit_transform(X.Humidity)

obj4 = LabelEncoder()
X.Wind = obj4.fit_transform(X.Wind)
print("\n The Encoded and Transformed Training Examples \n",
X.head())

obj5 = LabelEncoder()
Y = obj5.fit_transform(Y)
print("The class Label encoded in numerical form is",Y)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
test_size = 0.20)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
from sklearn.metrics import accuracy_score
```

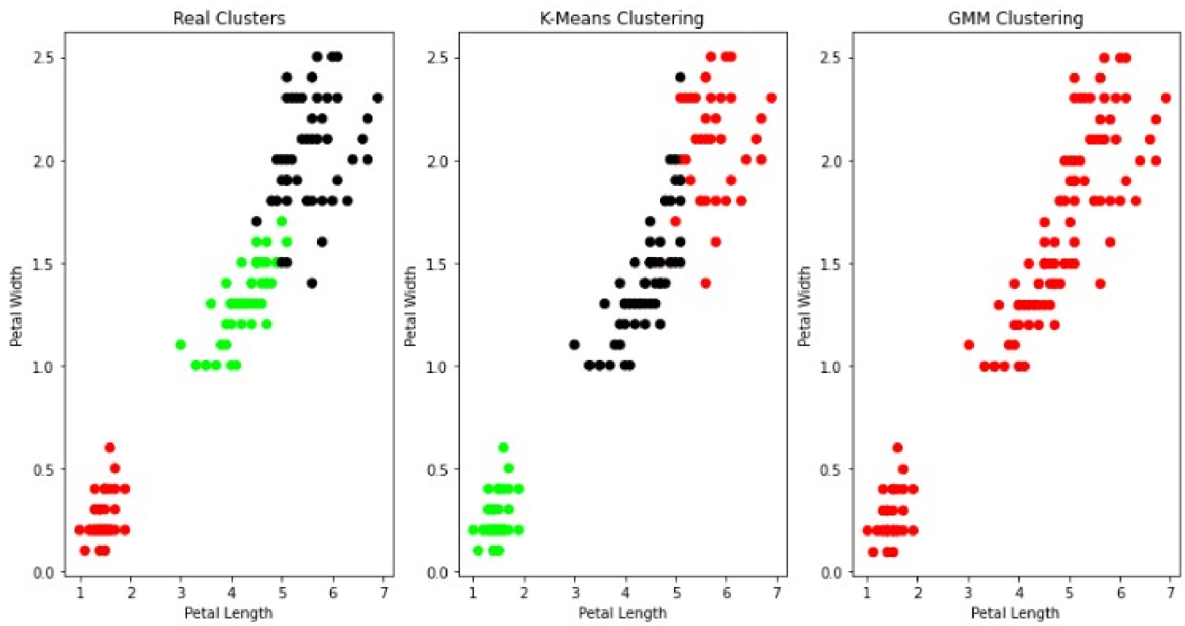
```
print("Accuracy is:", accuracy_score(classifier.predict(X_test),
Y_test))
```

Program-7: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which
samples belongs to

plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
```

```
gmm = GaussianMixture(n_components=40)
gmm.fit(xs)
plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering
matched the true labels more closely than the Kmeans.')
```

Output:



Program-8: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
iris=datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test =
train_test_split(iris.data,iris.target,test_size=0.1)
#random_state=0
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:",
str(y_test[r])," Predicted-label:", str(y_pred[r]))

    print("Classification Accuracy :" ,
classifier.score(x_test,y_test));
```

Output:

```
Iris Data set loaded...
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [5.  3.6 1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [4.5 2.3 1.3 0.3]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [5.1 3.5 1.4 0.3]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.1 2.6 5.6 1.4]  Actual-label: 2  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [4.4 2.9 1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [5.2 3.5 1.5 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.2 3.4 5.4 2.3]  Actual-label: 2  Predicted-label: 2
```

```
Classification Accuracy : 0.9333333333333333
 Sample: [4.8 3.4 1.9 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.9 3.1 5.4 2.1]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
 Sample: [5.6 3.  4.1 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [4.7 3.2 1.6 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.3 2.3 4.4 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.1 3.4 1.5 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
```

Program-9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 *
tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in
domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```

Output: