MACHINE LEARNING LABORATORY
SEMESTER – VI

Subject Code	21AIL66	**CIE Marks**	50
Number of Contact Hours/Week	0:2:2	**SEE Marks**	50
Total Number of Lab Contact Hours	24	**Exam Hours**	3 Hrs

Credits – 1

Course Learning Objectives:
CLO 2. To learn and understand the Importance Machine learning Algorithms
CLO 3. Compare and contrast the learning techniques like ANN approach, Bayesian learning and reinforcement learning.
CLO 4. Able to solve and analyse the problems on ANN, Instance based learning and Reinforcement learning techniques.
CLO 5. To impart the knowledg

- Implement and evaluate ML algorithms in Python/Java programming language.

Descriptions (if any):
1. The programs can be implemented in either JAVA or Python.
2. Data sets can be taken from standard repository such as UCI

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

1.	Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.
2	Aim: Demonstrate the working model and principle of candidate elimination algorithm. Program: 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.
3	Aim: To construct the Decision tree using the training data sets under supervised learning concept. Program: 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 to classify a new sample.
4	Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle. Program: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
5	Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm. Program: 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
6	Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle. Program:- Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.
7	Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept. Program: 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 Python ML library classes/API in the program.
8	Aim: Demonstrate and analyse the results of classification based on KNN Algorithm. Program: 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	Aim: Understand and analyse the concept of Regression algorithm techniques. Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
10	Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm. Program: Implement and demonstrate the working of SVM algorithm for classification

1. **Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.**

```
import csv

# Load data from CSV
with open('lab1a.csv', 'r') as f:
    reader = csv.reader(f)
    your_list = list(reader)

if your_list:
    num_attributes = len(your_list[0]) - 1
    # Exclude the class label
    h = ['0'] * num_attributes
    print(h)

for i in your_list:
    print(i)

    # Check if the instance is positive
    if i[-1] == "Yes":
        j = 0

        # Iterate through attributes of the instance (excluding the class label)
        for x in i[:-1]:
            # Update hypothesis
            if x != h[j] and h[j] == '0':
                h[j] = x
            elif x != h[j] and h[j] != '0':
                h[j] = '?'
            j += 1
        print(h)

print("Most specific hypothesis is")
print(h)
```

Datasets:

example	sky	airtemp	humidity	wind	water	forecast	enjoy
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Output :

```
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast', 'enjoy']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
Most specific hypothesis is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

2. Aim: Demonstrate the working model and principle of candidate elimination algorithm. Program: 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 csv

# Open and read the CSV file
with open("lab1a.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

# Initialize specific and general hypotheses
specific = data[1][:-1]  # Initial specific hypothesis from the first positive example
general = [['?' for _ in range(len(specific))] for _ in range(len(specific))]  # General hypothesis

# Process each example in the data
for i in data:
    if i[-1] == "Yes":  # Positive example
        for j in range(len(specific)):
            if i[j] != specific[j]:
                specific[j] = "?"
                general[j][j] = "?"
    elif i[-1] == "No":  # Negative example
        for j in range(len(specific)):
            if i[j] != specific[j]:
                general[j][j] = specific[j]
            else:
                general[j][j] = "?"

    # Print steps of the Candidate Elimination Algorithm
    print("\nStep " + str(data.index(i) + 1) + " of Candidate Elimination Algorithm")
    print("Specific hypothesis: ", specific)
    print("General hypothesis: ", general)

# Collect the general hypotheses
gh = []  # gh = general Hypothesis
for i in general:
    for j in i:
        if j != '?':
            gh.append(i)
            break

# Final hypotheses
print("\nFinal Specific hypothesis:\n", specific)
print("\nFinal General hypothesis:\n", gh)
```

Datasets :

example	sky	airtemp	humidity	wind	water	forecast	enjoy
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Output:

```
Step 1 of Candidate Elimination Algorithm
Specific hypothesis:  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
General hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
Specific hypothesis:  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
General hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
Specific hypothesis:  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm
Specific hypothesis:  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General hypothesis:  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 5 of Candidate Elimination Algorithm
Specific hypothesis:  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General hypothesis:  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

3. Aim: To construct the Decision tree using the training data sets under supervised learning concept. Program: 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 to classify a new sample.

```
import math
import pandas as pd
from operator import itemgetter

class DecisionTree:
    def __init__(self, df, target, positive, parent_val=None, parent=None):
        self.data = df
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = None

    def _get_entropy(self, data):
        p = sum(data[self.target] == self.positive)
        n = data.shape[0] - p
        p_ratio = p / (p + n)
        n_ratio = 1 - p_ratio
        entropy_p = -p_ratio * math.log2(p_ratio) if p_ratio != 0 else 0
        entropy_n = -n_ratio * math.log2(n_ratio) if n_ratio != 0 else 0
        return entropy_p + entropy_n

    def _get_gain(self, feat):
        avg_info = 0
        for val in self.data[feat].unique():
            subset = self.data[self.data[feat] == val]
            avg_info += self._get_entropy(subset) * len(subset) / self.data.shape[0]
        return self._get_entropy(self.data) - avg_info

    def _get_splitter(self):
        self.splitter = max(self.gains, key=itemgetter(1))[0]
    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features]
            self._get_splitter()
            residual_columns = [k for k in self.data.columns if k != self.splitter]
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter] == val][residual_columns]
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val, self.splitter)
                tmp_node.update_nodes()
                self.childs.append(tmp_node)
        else:
            self.decision = self.data[self.target].iloc[0]

def print_tree(n, level=0):
    if n.parent is not None:
        indent = "  " * level
        decision = f"(Decision: {n.decision})" if n.decision is not None else ""
```

```
        print(f"{indent}{n.parent}: {n.parent_val} {decision}")
    for child in n.childs:
        if child:
            print_tree(child, level + 1)

# Load the dataset

# Example usage
df = pd.read_csv('Play Tennis.csv')  # Read the dataset from CSV
dt = DecisionTree(df, 'Play', 'Yes')  # Initialize the DecisionTree object
dt.update_nodes()  # Build the decision tree
print_tree(dt)  # Print the decision tree
```

Datasets :

Outlook	Tempratu	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Output :

```
Outlook: Sunny
  Humidity: High (Decision: No)
  Humidity: Normal (Decision: Yes)
Outlook: Overcast (Decision: Yes)
Outlook: Rain
  Wind: Weak (Decision: Yes)
  Wind: Strong (Decision: No)
```

4. Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle. Program: 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) # 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):
   sig = sigmoid(x)
   return sig * (1 - sig)

#Variable initialization
epoch=5000       #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))

#Forward Propogation
hinpl=np.dot (X, wh)
hinp=hinpl + bh
hlayer_act=sigmoid (hinp)
outinpl=np.dot (hlayer_act,wout)
outinp =outinpl+ bout
output= sigmoid (outinp)

#Backpropagation
EO= y-output
outgrad =derivatives_sigmoid(output)
d_output= EO*outgrad
EH = d_output.dot(wout.T)

#how much hidden layer wts contributed to error
hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad
```

```
#dotproduct of nextlayererror and currentlayerop
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 ("Predicted Output: \n",output)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.88493478]
 [0.87493145]
 [0.8812267 ]]
```

5. **Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm. Program: 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 import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
data = pd.read_csv('Play Tennis.csv')
print("The first 5 values of data is :\n",data.head())
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)
le_temperature = LabelEncoder()
X.Temperature = le_temperature.fit_transform(X.Temperature)
le_humidity = LabelEncoder()
X.Humidity = le_humidity.fit_transform(X.Humidity)
le_windy = LabelEncoder()
X.Windy = le_windy.fit_transform(X.Windy)
print("\nNow the Train data is :\n",X.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
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))
```

DATASET:

Outlook	Tempratui	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

OUTPUT:

```
The first 5 values of data is :
     Outlook Temperature Humidity   Windy 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

The First 5 values of train data is
     Outlook Temperature Humidity   Windy
0     Sunny         Hot     High    Weak
1     Sunny         Hot     High  Strong
2  Overcast         Hot     High    Weak
3      Rain        Mild     High    Weak
4      Rain        Cool   Normal    Weak

The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: Play, dtype: object

Now the Train data is :
    Outlook  Temperature  Humidity  Windy
0        2            1         0      1
1        2            1         0      0
2        0            1         0      1
3        1            2         0      1
4        1            0         1      1

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666
```

6. **Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle. Program:- Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.**

```
import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data
heartDisease = pd.read_csv('Heart_disease_cleveland_new.csv')
heartDisease = heartDisease.replace('?',np.nan)

#display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())

print(heartDisease.columns)

# Rename columns to match model variable names if necessary
heartDisease.rename(columns={
   'Heartdisease': 'heartdisease'
}, inplace=True)

# Verify the renaming
print(heartDisease.columns)

# Define the Bayesian Network structure
model = BayesianNetwork([
   ('age', 'trestbps'),
   ('age', 'fbs'),
   ('sex', 'trestbps'),
   ('exang', 'trestbps'),
   ('trestbps', 'heartdisease'),
   ('fbs', 'heartdisease'),
   ('heartdisease', 'restecg'),
   ('heartdisease', 'thalach'),
   ('heartdisease', 'chol')
])

# Fit the model using Maximum Likelihood Estimators
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)
```

```
print('\nAvailable states for age:')
age_states = heartDisease['age'].unique()
print(age_states)

#computing the Probability of HeartDisease given Age
print('\n 1. Probability of HeartDisease given Age=37')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence={'age':37})
print(q)

print('\nAvailable states for age:')
chol_states = heartDisease['chol'].unique()
print(chol_states)

#computing the Probability of HeartDisease given cholesterol
print('\n 2. Probability of HeartDisease given cholesterol=250')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence={'chol':250})
print(q)
```

Datasets :

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	0	145	233	1	2	150	0	2.3	2	0	2	0
67	1	3	160	286	0	2	108	1	1.5	1	3	1	1
67	1	3	120	229	0	2	129	1	2.6	1	2	3	1
37	1	2	130	250	0	0	187	0	3.5	2	0	1	0
41	0	1	130	204	0	2	172	0	1.4	0	0	1	0
56	1	1	120	236	0	0	178	0	0.8	0	0	1	0
62	0	3	140	268	0	2	160	0	3.6	2	2	1	1
57	0	3	120	354	0	0	163	1	0.6	0	0	1	0
63	1	3	130	254	0	2	147	0	1.4	1	1	3	1
53	1	3	140	203	1	2	155	1	3.1	2	0	3	1
57	1	3	140	192	0	0	148	0	0.4	1	0	2	0
56	0	1	140	294	0	2	153	0	1.3	1	0	1	0
56	1	2	130	256	1	2	142	1	0.6	1	1	2	1
44	1	1	120	263	0	0	173	0	0	0	0	3	0
52	1	2	172	199	1	0	162	0	0.5	0	0	3	0
57	1	2	150	168	0	0	174	0	1.6	0	0	1	0
48	1	1	110	229	0	0	168	0	1	2	0	3	1
54	1	3	140	239	0	0	160	0	1.2	0	0	1	0
48	0	2	130	275	0	0	139	0	0.2	0	0	1	0
49	1	1	130	266	0	0	171	0	0.6	0	0	1	0
64	1	0	110	211	0	2	144	1	1.8	1	0	1	0
58	0	0	150	283	1	2	162	0	1	0	0	1	0
58	1	1	120	284	0	2	160	0	1.8	1	0	1	1
58	1	2	132	224	0	2	173	0	3.2	0	2	3	1
60	1	3	130	206	0	2	132	1	2.4	1	2	3	1
50	0	2	120	219	0	0	158	0	1.6	1	0	1	0
58	0	2	120	340	0	0	172	0	0	0	0	1	0
66	0	0	150	226	0	0	114	0	2.6	2	0	1	0
43	1	3	150	247	0	0	171	0	1.5	0	0	1	0
40	1	3	110	167	0	2	114	1	2	1	0	3	1

OUTPUT:

```
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   0       145   233    1        2      150      0      2.3      2
1   67    1   3       160   286    0        2      108      1      1.5      1
2   67    1   3       120   229    0        2      129      1      2.6      1
3   37    1   2       130   250    0        0      187      0      3.5      2
4   41    0   1       130   204    0        2      172      0      1.4      0

   ca  thal  Heartdisease
0   0     2             0
1   3     1             1
2   2     3             1
3   0     1             0
4   0     1             0

Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'Heartdisease'],
      dtype='object')
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease'],
      dtype='object')

Learning CPD using Maximum likelihood estimators


Available states for age:
[63 67 37 41 56 62 57 53 44 52 48 54 49 64 58 60 50 66 43 40 69 59 42 55
 61 65 71 51 46 45 39 68 47 34 35 29 70 77 38 74 76]


 1. Probability of HeartDisease given Age=37
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.6221 |
+-----------------+---------------------+
| heartdisease(1) |              0.3779 |
+-----------------+---------------------+


Available states for age:
[233 286 229 250 204 236 268 354 254 203 192 294 256 263 199 168 239 275
 266 211 283 284 224 206 219 340 226 247 167 230 335 234 177 276 353 243
 225 302 212 330 175 417 197 198 290 253 172 273 213 305 216 304 188 282
 185 232 326 231 269 267 248 360 258 308 245 270 208 264 321 274 325 235
 257 164 141 252 255 201 222 260 182 303 265 309 307 249 186 341 183 407
 217 288 220 209 227 261 174 281 221 205 240 289 318 298 564 246 322 299
 300 293 277 214 207 223 160 394 184 315 409 244 195 196 126 313 259 200
 262 215 228 193 271 210 327 149 295 306 178 237 218 242 319 166 180 311
 278 342 169 187 157 176 241 131]


 2. Probability of HeartDisease given cholesterol=250
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              1.0000 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
```

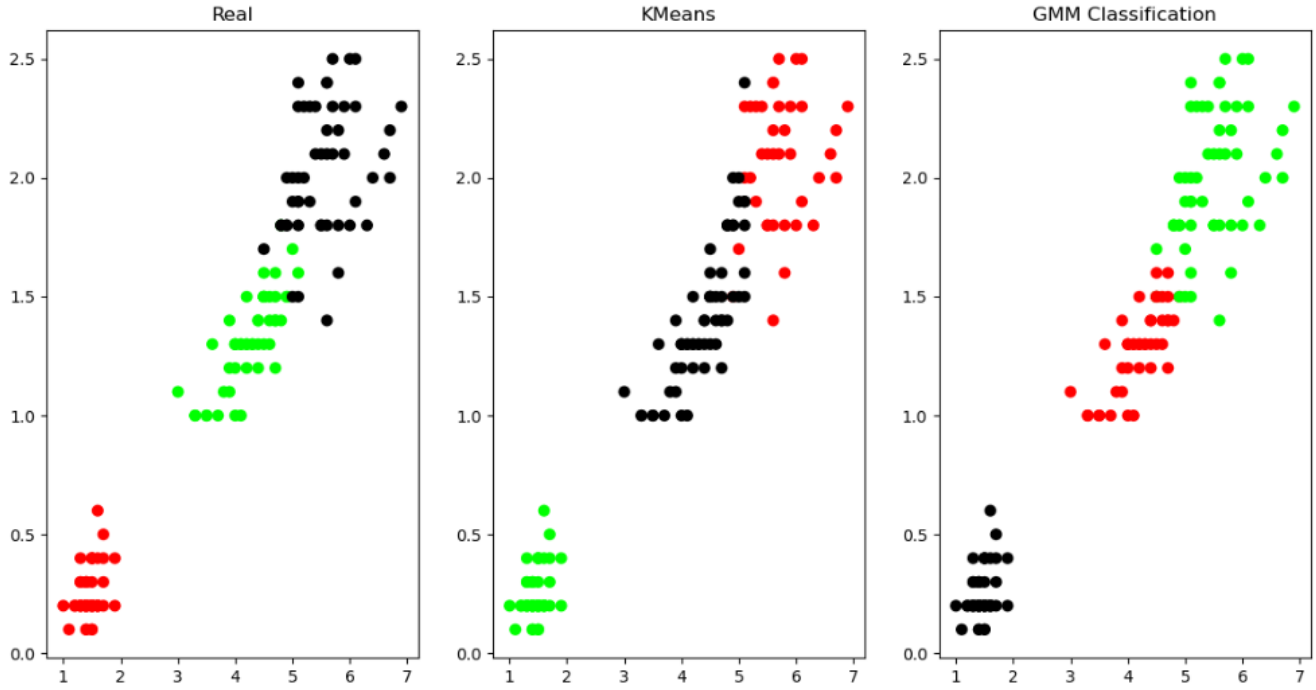
7. **Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept. Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.**

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

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

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

DATASET:

Id	SepalLeng	SepalWidt	PetalLengt	PetalWidt	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa

OUTPUT:



8. **Aim: Demonstrate and analyse the results of classification based on KNN Algorithm. Program: 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.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)

kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

for i in range(len(X_test)):
   x=X_test[i]
   x_new=np.array([x])
   prediction=kn.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_names"][predicti
on])
print(kn.score(X_test,y_test))
```

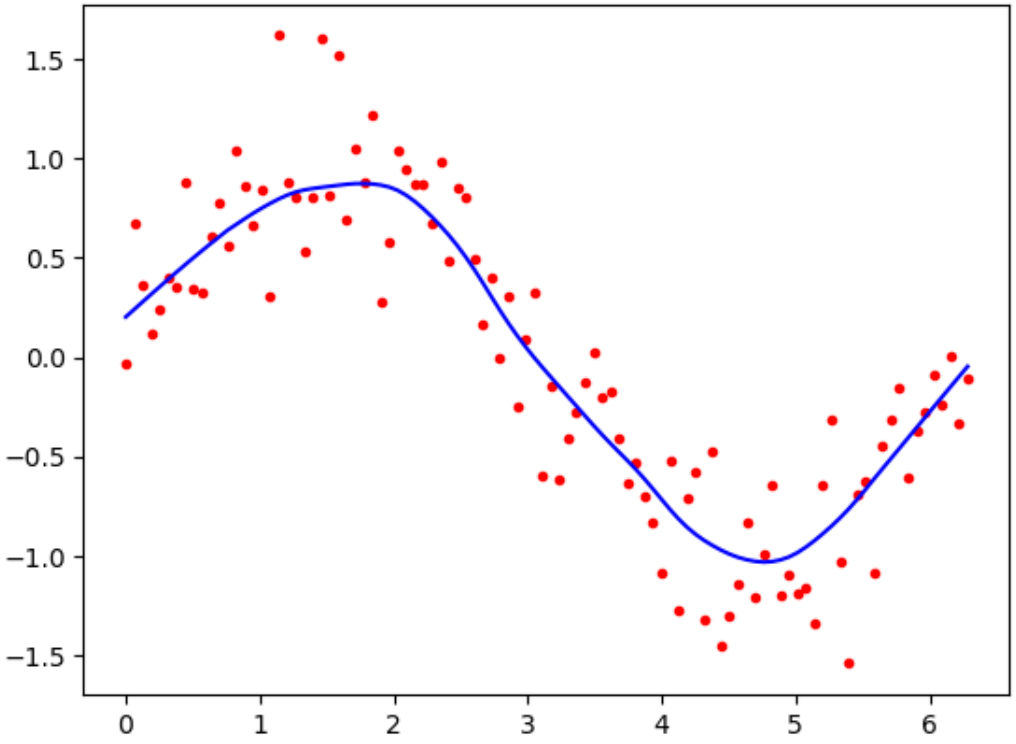
OUTPUT:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

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

```
from math import ceil
import numpy as np
from scipy import linalg

def lowess(x, y, f, iterations):
   n = len(x)
   r = int(ceil(f * n))
   h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
   w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
   w = (1 - w ** 3) ** 3
   yest = np.zeros(n)
   delta = np.ones(n)
   for iteration in range(iterations):
      for i in range(n):
         weights = delta * w[:, i]
         b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
         A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x), np.sum(weights * x * x)]])
         beta = linalg.solve(A, b)
         yest[i] = beta[0] + beta[1] * x[i]

      residuals = y - yest
      s = np.median(np.abs(residuals))
      delta = np.clip(residuals / (6.0 * s), -1, 1)
      delta = (1 - delta ** 2) ** 2

   return yest

import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f =0.25
iterations=3
yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

OUTPUT:



10. Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm. Program: Implement and demonstrate the working of SVM algorithm for classification

```
import pandas as pd
from sklearn.model_selection import train_test_split from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Step 1: Dataset Preparation data = {
'Weight': [150, 200, 250, 180, 300, 220],
'Color': ['Red', 'Red', 'Orange', 'Orange', 'Red', 'Orange'],
'Label': ['Apple', 'Apple', 'Orange', 'Orange', 'Apple', 'Orange']
}

df = pd.DataFrame(data)

# Step 2: Feature Extraction
df['Color'] = df['Color'].map({'Red': 0, 'Orange': 1})

# Step 3: Data Split
X = df[['Weight', 'Color']] y = df['Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Model Training svm = SVC(kernel='linear') svm.fit(X_train, y_train)

# Step 5: Model Evaluation y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:", accuracy)

# Step 6: Prediction new_data = {
'Weight': [190],
'Color': [0]
}

new_df = pd.DataFrame(new_data) new_prediction = svm.predict(new_df) print("New prediction:", new_prediction)
```

Output :

```
Accuracy: 0.0

New prediction: ['Orange']
```