

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING



MACHINE LEARNING LABORATORY

Subject Code : KG23ACM302

Regulation : KGR23

Academic Yea : 2025-26

IV B. TECH I SEMESTER

COMPUTER SCIENCE AND ENGINEERING
KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

- To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with government organizations and collaboration with industries.
- To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.

PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.

PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.

PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

PROGRAM OUTCOMES

<p>PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.</p>
<p>PO II: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.</p>
<p>PO III: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.</p>
<p>PO IV: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.</p>
<p>PO V: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.</p>
<p>PO VI: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.</p>
<p>PO VII: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.</p>
<p>PO VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.</p>
<p>PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.</p>
<p>PO X: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.</p>
<p>PO XI: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.</p>
<p>PO XII: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.</p>

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype

PSO2: The graduating student will be analyze the contemporary research issues in different areas of computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.

PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi- disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

Course Objective: The objectives of this course for the students are to:

1. Implement and apply machine learning algorithms for classification, regression, and clustering tasks
2. Work with real-world datasets and apply data preprocessing techniques
3. Use machine learning frameworks and libraries such as Scikit-learn, TensorFlow, and Keras
4. Evaluate and fine-tune machine learning models for optimal performance
5. Analyze and interpret the results of machine learning experiments

Course Outcomes:

CO1: Understand modern notions in predictive data analysis(K2)

CO2: Select data, model selection, model complexity and identify the trends(K2)

CO3: Apply a range of machine learning algorithms along with their strengths and weaknesses(K3)

CO4. Build predictive models from data and analyze their performance (K2)

COS: Analyze and interpret the results from machine learning experiments and communicate findings effectively.(K4)

List of Experiments

1. Data Preprocessing Techniques

Aim: To understand and apply data preprocessing techniques such as handling missing values, encoding categorical variables, and feature scaling, which are essential steps before building machine learning models.

Algorithm:

1. Create a sample dataset with missing and categorical values.
 2. Handle missing values using mean imputation with `SimpleImputer`.
 3. Encode categorical variables using `LabelEncoder`.
 4. Standardize numerical features using `StandardScaler`.
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.
 3. 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. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

5. Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file and compute the accuracy with a few test data sets.
6. Implement naive Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.
7. Write a program to construct a Bayesian network to diagnose CORONA infection using standard WHO Data Set.
8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms.
9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.
10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select an appropriate data set for your experiment and draw graphs.

List of Experiments with Solutions

1. Data Preprocessing Techniques

Aim: To understand and apply data preprocessing techniques such as handling missing values, encoding categorical variables, and feature scaling, which are essential steps before building machine learning models.

Algorithm:

1. Create a sample dataset with missing and categorical values.
2. Handle missing values using mean imputation with `SimpleImputer`.
3. Encode categorical variables using `LabelEncoder`.
4. Standardize numerical features using `StandardScaler`.

CODE:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Sample dataset
data = {'Age': [25, 30, None, 35, 40],
        'Salary': [50000, 60000, 65000, None, 80000],
        'City': ['Delhi', 'Mumbai', 'Delhi', 'Chennai', 'Mumbai']}
df = pd.DataFrame(data)

# Handle missing values
imputer = SimpleImputer(strategy='mean')
df[['Age', 'Salary']] = imputer.fit_transform(df[['Age', 'Salary']])
```

```
# Encode categorical data
label = LabelEncoder()
df['City'] = label.fit_transform(df['City'])

# Feature Scaling
scaler = StandardScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])

print(df)
```

Output (Example):

	Age	Salary	City
0	-1.414214	-1.414214	1
1	-0.707107	-0.707107	2
2	0.000000	0.000000	1
3	0.707107	0.707107	0
4	1.414214	1.414214	2

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.

Code:

```
import numpy as np
import pandas as pd

In [2]:
# Loading Data from a CSV File
```

```
data = pd.DataFrame(data=pd.read_csv('trainingdata.csv'))
print(data)
```

```
   sky airTemp humidity  wind water forecast enjoySport
0 Sunny   Warm  Normal  Strong  Warm   Same       Yes
1 Sunny   Warm   High  Strong  Warm   Same       Yes
2 Rainy  Cold   High  Strong  Warm  Change       No
3 Sunny   Warm   High  Strong  Cool  Change       Yes
```

In [3]:

```
# Separating concept features from Target
```

```
concepts = np.array(data.iloc[:,0:-1])
```

```
print(concepts)
```

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

```
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
```

```
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
```

```
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

In [4]:

```
# Isolating target into a separate DataFrame
```

```
# copying last column to target array
```

```
target = np.array(data.iloc[:,-1])
```

```
print(target)
```

```
['Yes' 'Yes' 'No' 'Yes']
```

In [5]:

```
def learn(concepts, target):
```

```
    """
```

```
    learn() function implements the learning method of the Candidate elimination algorithm.
```

```
    Arguments:
```

```
        concepts - a data frame with all the features
```

```
        target - a data frame with corresponding output values
```

```
    """
```

```
# Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing to the same memory
location
specific_h = concepts[0].copy()
print("\nInitialization of specific_h and general_h")
print(specific_h)
#h=["#" for i in range(0,5)]
#print(h)

general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
print(general_h)
# The learning iterations
for i, h in enumerate(concepts):

    # Checking if the hypothesis has a positive target
    if target[i] == "Yes":
        for x in range(len(specific_h)):

            # Change values in S & G only if values change
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x][x] = '?'

    # Checking if the hypothesis has a positive target
    if target[i] == "No":
        for x in range(len(specific_h)):

            # For negative hypothesis change values only in G
            if h[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
```

else:

```
general_h[x][x] = '?'
```

```
print("\nSteps of Candidate Elimination Algorithm",i+1)
```

```
print(specific_h)
```

```
print(general_h)
```

find indices where we have empty rows, meaning those that are unchanged

```
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
```

for i in indices:

remove those rows from general_h

```
general_h.remove(['?', '?', '?', '?', '?', '?'])
```

Return final values

return specific_h, general_h

In [6]:

```
s_final, g_final = learn(concepts, target)
```

```
print("\nFinal Specific_h:", s_final, sep="\n")
```

```
print("\nFinal General_h:", g_final, sep="\n")
```

Initialization of specific_h and general_h

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

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

Steps of Candidate Elimination Algorithm 1

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

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

Steps of Candidate Elimination Algorithm 2

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



```
with open(filename, 'r') as csvfile:  
    datareader = csv.reader(csvfile, delimiter=',')  
    headers = next(datareader)  
    metadata = []  
    traindata = []  
    for name in headers:  
        metadata.append(name)  
    for row in datareader:  
        traindata.append(row)  
  
    return (metadata, traindata)
```

In [3]:

```
class Node:
```

```
    def __init__(self, attribute):  
        self.attribute = attribute  
        self.children = []  
        self.answer = ""
```

```
    def __str__(self):  
        return self.attribute
```

In [4]:

```
def subtables(data, col, delete):  
    dict = {}  
    items = np.unique(data[:, col])  
    count = np.zeros((items.shape[0], 1), dtype=np.int32)  
  
    for x in range(items.shape[0]):  
        for y in range(data.shape[0]):  
            if data[y, col] == items[x]:  
                count[x] += 1
```

```
for x in range(items.shape[0]):  
    dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")  
    pos = 0  
    for y in range(data.shape[0]):  
        if data[y, col] == items[x]:  
            dict[items[x]][pos] = data[y]  
            pos += 1  
    if delete:  
        dict[items[x]] = np.delete(dict[items[x]], col, 1)  
  
    return items, dict
```

In [5]:

```
def entropy(S):  
    items = np.unique(S)  
  
    if items.size == 1:  
        return 0  
  
    counts = np.zeros((items.shape[0], 1))  
    sums = 0  
  
    for x in range(items.shape[0]):  
        counts[x] = sum(S == items[x]) / (S.size * 1.0)  
  
    for count in counts:  
        sums += -1 * count * math.log(count, 2)  
    return sums
```

In [6]:

```
def gain_ratio(data, col):
```

```
items, dict = subtables(data, col, delete=False)
```

```
total_size = data.shape[0]
```

```
entropies = np.zeros((items.shape[0], 1))
```

```
intrinsic = np.zeros((items.shape[0], 1))
```

```
for x in range(items.shape[0]):
```

```
    ratio = dict[items[x]].shape[0]/(total_size * 1.0)
```

```
    entropies[x] = ratio * entropy(dict[items[x]][:, -1])
```

```
    intrinsic[x] = ratio * math.log(ratio, 2)
```

```
total_entropy = entropy(data[:, -1])
```

```
iv = -1 * sum(intrinsic)
```

```
for x in range(entropies.shape[0]):
```

```
    total_entropy -= entropies[x]
```

```
return total_entropy / iv
```

In [7]:

```
def create_node(data, metadata):
```

```
    if (np.unique(data[:, -1])).shape[0] == 1:
```

```
        node = Node("")
```

```
        node.answer = np.unique(data[:, -1])[0]
```

```
    return node
```

```
gains = np.zeros((data.shape[1] - 1, 1))
```

```
for col in range(data.shape[1] - 1):
```

```
    gains[col] = gain_ratio(data, col)
```

```
split = np.argmax(gains)
```

```
node = Node(metadata[split])
```

```
metadata = np.delete(metadata, split, 0)
```

```
items, dict = subtables(data, split, delete=True)
```

```
for x in range(items.shape[0]):
```

```
    child = create_node(dict[items[x]], metadata)
```

```
    node.children.append((items[x], child))
```

```
return node
```

In [8]:

```
def empty(size):
```

```
    s = ""
```

```
    for x in range(size):
```

```
        s += " "
```

```
    return s
```

```
def print_tree(node, level):
```

```
    if node.answer != "":
```

```
        print(empty(level), node.answer)
```

```
        return
```

```
    print(empty(level), node.attribute)
```

```
    for value, n in node.children:
```

```
        print(empty(level + 1), value)
```

```
        print_tree(n, level + 2)
```

In [9]:

```
metadata, traindata = read_data("tennisdata.csv")
```

```
data = np.array(traindata)
```

```
node = create_node(data, metadata)
```

```
print_tree(node, 0)
```

```
Outlook
```

```
Overcast
```

```
  b'Yes'
```

```
Rainy
```

```
Windy
```

```
  b'False'
```

```
    b'Yes'
```

```
    b'True'
```

```
  b'No'
```

```
Sunny
```

```
Humidity
```

```
  b'High'
```

```
    b'No'
```

```
    b'Normal' b'Yes'
```

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

CODE:

```
import numpy as np
```

```
X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)  # X = (hours sleeping, hours studying)
```

```
y = np.array([92, 86, 89]), dtype=float)          # y = score on test
```

```
# scale units
```

```
X = X/np.amax(X, axis=0)    # maximum of X array
```

```
y = y/100                  # max test score is 100
```

In [2]:

```
class Neural_Network(object):
```

```
    def __init__(self):
```

```
        # Parameters
```

```
        self.inputSize = 2
```

```
        self.outputSize = 1
```

```
        self.hiddenSize = 3
```

```
        # Weights
```

```
        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)    # (3x2) weight matrix from  
input to hidden layer
```

```
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)    # (3x1) weight matrix  
from hidden to output layer
```

```
    def forward(self, X):
```

```
        #forward propagation through our network
```

```
        self.z = np.dot(X, self.W1)    # dot product of X (input) and first set of 3x2 weights
```

```
        self.z2 = self.sigmoid(self.z)    # activation function
```

```
        self.z3 = np.dot(self.z2, self.W2)    # dot product of hidden layer (z2) and second set of  
3x1 weights
```

```
        o = self.sigmoid(self.z3)    # final activation function
```

```
    return o
```

```
    def sigmoid(self, s):
```

```
        return 1/(1+np.exp(-s))    # activation function
```

```
def sigmoidPrime(self, s):
```

```
    return s * (1 - s)    # derivative of sigmoid
```

```
def backward(self, X, y, o):
```

```
    # backward propagate through the network
```

```
    self.o_error = y - o    # error in output
```

```
    self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to
```

```
        self.z2_error = self.o_delta.dot(self.W2.T)    # z2 error: how much our hidden layer  
weights contributed to output error
```

```
        self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to  
z2 error
```

```
        self.W1 += X.T.dot(self.z2_delta)    # adjusting first set (input --> hidden) weights
```

```
        self.W2 += self.z2.T.dot(self.o_delta) # adjusting second set (hidden --> output) weights
```

```
def train (self, X, y):
```

```
    o = self.forward(X)
```

```
    self.backward(X, y, o)
```

In [3]:

```
NN = Neural_Network()
```

```
for i in range(1000): # trains the NN 1,000 times
```

```
    print ("\nInput: \n" + str(X))
```

```
    print ("\nActual Output: \n" + str(y))
```

```
    print ("\nPredicted Output: \n" + str(NN.forward(X)))
```

```
        print ("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(X))))))    # mean sum squared  
loss)
```

```
NN.train(X, y)
```

Input:

```
[[0.66666667 1.    ]  
 [0.33333333 0.55555556]  
 [1.    0.66666667]]
```

Actual Output:

```
[[0.92]  
 [0.86]  
 [0.89]]
```

Predicted Output:

```
[[0.47212874]  
 [0.42728946]  
 [0.40891365]]
```

Loss:

```
0.20642371917499927
```

5. Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file and compute the accuracy with a few test data sets.

code

```
# import necessary libarities  
import pandas as pd  
from sklearn import tree  
from sklearn.preprocessing import LabelEncoder  
from sklearn.naive_bayes import GaussianNB
```

```
# load data from CSV
data = pd.read_csv('tennisdata.csv')
print("The first 5 values of data is :\n",data.head())
The first 5 values of data is :
    Outlook Temperature Humidity Windy PlayTennis
0  Sunny      Hot    High False    No
1  Sunny      Hot    High  True    No
2  Overcast   Hot    High False    Yes
3  Rainy     Mild    High False    Yes
4  Rainy     Cool   Normal False    Yes
In [2]:
# obtain Train data and Train output
X = data.iloc[:, :-1]
print("\n\nThe First 5 values of train data is\n\n",X.head())
The First 5 values of train data is
    Outlook Temperature Humidity Windy
0  Sunny      Hot    High False
1  Sunny      Hot    High  True
2  Overcast   Hot    High False
3  Rainy     Mild    High False
4  Rainy     Cool   Normal False
In [3]:
y = data.iloc[:, -1]
print("\n\nThe first 5 values of Train output is\n\n",y.head())
The first 5 values of Train output is
0  No
1  No
2  Yes
3  Yes
```

4 Yes

Name: PlayTennis, dtype: object

In [4]:

```
# Convert then in numbers
```

```
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())
```

Now the Train data is :

	Outlook	Temperature	Humidity	Windy
0	2	1	0	0
1	2	1	0	1
2	0	1	0	0
3	1	2	0	0
4	1	0	1	0

In [5]:

```
le_PlayTennis = LabelEncoder()
```

```
y = le_PlayTennis.fit_transform(y)
```

```
print("\nNow the Train output is\n",y)
```

Now the Train output is

```
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

In [6]:

```
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))
Accuracy is: 0.6666666666666666
```

6. Implement naive Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.

CODE:

```
import pandas as pd
msg = pd.read_csv('document.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
Total Instances of Dataset: 18

In [2]:
X = msg.message
y = msg.labelnum

from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)

from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
```

In [6]:

```
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
```

```
print(df[0:5])
```

```

about am an and awesome bad beers best boss can ... tired to \
0  0  1  0  1    0  0  0  0  0  0 ...  1  0
1  0  0  0  0    0  0  0  0  0  0 ...  0  0
2  0  0  0  0    0  0  0  0  0  0 ...  0  0
3  0  0  0  0    0  0  0  0  0  1 ...  0  0
4  0  0  0  0    0  0  0  0  0  0 ...  0  0

```

```
today tomorrow very we went will with work
```

```

0  0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0
3  0  0  0  0  0  0  1  0
4  0  0  0  0  0  0  0  0

```

[5 rows x 49 columns]

In [4]:

```
from sklearn.naive_bayes import MultinomialNB
```

```
clf = MultinomialNB()
```

```
clf.fit(Xtrain_dm, ytrain)
```

```
pred = clf.predict(Xtest_dm)
```

In [5]:

```
for doc, p in zip(Xtrain, pred):
```

```
    p = 'pos' if p == 1 else 'neg'
```

```
    print("%s -> %s" % (doc, p))
```

I am sick and tired of this place -> pos

I do not like the taste of this juice -> neg

I love this sandwich -> neg

I can't deal with this -> pos

I do not like this restaurant -> neg

In [6]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
```

```
print('Accuracy Metrics: \n')
```

```
print('Accuracy: ', accuracy_score(ytest, pred))
```

```
print('Recall: ', recall_score(ytest, pred))
```

```
print('Precision: ', precision_score(ytest, pred))
```

```
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

Accuracy Metrics:

Accuracy: 0.6

Recall: 0.5

Precision: 1.0

Confusion Matrix:

```
[[1 0]
```

```
[2 2]]
```

7. Write a program to construct a Bayesian network to diagnose CORONA infection using standard WHO Data Set.

```
import pandas as pd
```

```
data=pd.read_csv("heartdisease.csv")
```

```
heart_disease=pd.DataFrame(data)
```

```
print(heart_disease)
```

```
   age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0    0     0     1     1     3         0         1
1    1     0     1     1     3         0         1
2    2     1     0     0     2         1         1
3    3     4     0     1     1     3         2         0
4    4     3     1     1     0     0         2         0
```

```

5 2 0 1 1 1 0 1
6 4 0 1 0 2 0 1
7 0 0 1 1 3 0 1
8 3 1 1 0 0 2 0
9 1 1 0 0 0 2 1
10 4 1 0 1 2 0 1
11 4 0 1 1 3 2 0
12 2 1 0 0 0 0 0
13 2 0 1 1 1 0 1
14 3 1 1 0 0 1 0
15 0 0 1 0 0 2 1
16 1 1 0 1 2 1 1
17 3 1 1 1 0 1 0
18 4 0 1 1 3 2 0

```

In [2]:

```
from pgmpy.models import BayesianModel
```

```

model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholesterol'),
('Lifestyle','diet'),
('cholesterol','heartdisease'),
('diet','cholesterol')
])

```

```
from pgmpy.estimate import MaximumLikelihoodEstimator
```

```
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
```

```
from pgmpy.inference import VariableElimination
```

```
HeartDisease_infer = VariableElimination(model)
```

In [3]:

```
print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
```

```
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')
```

```
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age':int(input('Enter age :')),
    'Gender':int(input('Enter Gender :')),
    'Family':int(input('Enter Family history :')),
    'diet':int(input('Enter diet :')),
    'Lifestyle':int(input('Enter Lifestyle :')),
    'cholestrol':int(input('Enter cholestrol :'))
})
```

```
print(q['heartdisease'])
```

```
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
```

```
For Gender Enter { Male:0, Female:1 }
```

```
For Family History Enter { yes:1, No:0 }
```

```
For diet Enter { High:0, Medium:1 }
```

```
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
```

```
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
```

```
Enter age :1
```

```
Enter Gender :1
```

```
Enter Family history :0
```

```
Enter diet :1
```

```
Enter Lifestyle :0
```

```
Enter cholestrol :1
```

```
+-----+-----+
```

```
| heartdisease | phi(heartdisease) |
```

```
+=====+
```

```
| heartdisease_0 | 0.0000 |
```

```
+-----+-----+
```

```
| heartdisease_1 |      1.0000|  
+-----+-----+
```

8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms.

CODE:

```
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  
In [2]:  
dataset=load_iris()  
# print(dataset)  
In [3]:  
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)  
In [4]:  
plt.figure(figsize=(14,7))  
colormap=np.array(['red','lime','black'])  
  
# REAL PLOT  
plt.subplot(1,3,1)
```

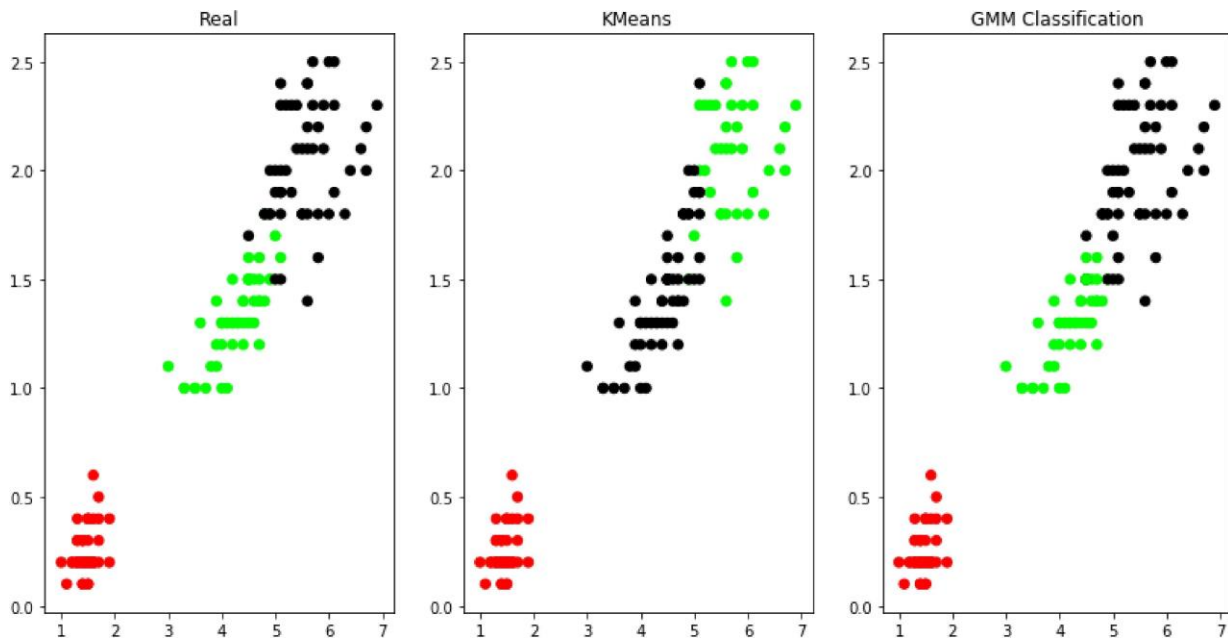
```
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)  
plt.title('Real')
```

```
# 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')  
Out[4]:  
Text(0.5, 1.0, 'GMM Classification')
```



9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

CODE:

```

from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

In [2]:
dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)

In [3]:
kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

Out[3]:

```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,  
                    weights='uniform')
```

In [4]:

```
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,datas  
et["target_names"][prediction])
```

```
print(kn.score(X_test,y_test))
```

```
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

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

CODE:

```
from math import ceil
import numpy as np
from scipy import linalg
In [2]:
```

```
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
```

```
In [4]:
```

```
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-")
```

Out[4]:

[<matplotlib.lines.Line2D at 0x37459696d8>]

