ARTIFICIAL INTELLIGENCE MACHINE LEARNING
18CSL76

LABORATORY MANUAL

VII Semester B.E.

(Academic Year: 2022-23)

Prepared By

Mrs. Suchetha G

Mrs. Madhura N Hegde
Assistant Professor, Department of ISE

Sahyadri college of Engineering and Management
An Autonomous institution

SAHYADRI
COLLEGE OF ENGINEERING & MANAGEMENT
An Autonomous Institution

Vision

To be a premier institution in Technology and Management by fostering excellence in education, innovation, incubation and values to inspire and empower the young minds.

Mission

M1. Creating an academic ambience to impart holistic education focusing on individual growth, integrity, ethical values and social responsibility.

M2. Develop skill based learning through industry-institution interaction to enhance competency and promote entrepreneurship.

M3. Fostering innovation and creativity through competitive environment with state-of-the-art infrastructure.

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Vision

To be a center of excellence in Information Science and Engineering through the interactive teaching-learning process, research, and innovation.

Mission

M1. Creating competitive ambience to enhance the innovative and experiential learning process through state of the art infrastructure.

M2. Grooming young minds through industry-institute interactions to solve societal issues and inculcate affinity towards research and entrepreneurship.

M3. Promoting teamwork and leadership qualities through inter-disciplinary activities in diversified areas of information science and engineering.

Program Educational Objectives (PEOs):

PEO1: Possess theoretical and practical knowledge to identify, scrutinize, formulate and solve challenging problems related to dynamically evolving information science.

PEO2: Inculcate core competency, professionalism and ethics to cater industrial needs and to solve societal problems.

PEO3: Engage in Lifelong learning and stay intact to the transformation in technologies and pursue research.

Program Outcomes:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. 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.

PO3. 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.

PO4. 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.

PO5. 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.

PO6. 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.

PO7. 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.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. 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.

PO11. 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.

PO12. 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.

Program Specific Outcomes (PSOs):

PSO1: Exhibit competency and skills in distributed computing, information security, cyber security, data analytics, and machine learning.

PSO2: Able to provide sustainable solution to implement and validate information science projects.

Subject Code	Co	Subject Name	CO No.	Course Outcomes	Blooms Level
18CSL76	C416	Artificial Intelligence & Machine Learning Laboratory	C416.1	Implement and Demonstrate AI search algorithms.	CL3
			C416.2	Demonstrate the Candidate Elimination Algorithm for finding the Hypothesis Space.	CL3
			C416.3	Demonstrate the implementation of Decision Tree Algorithm and use it to perform classification of a new data sample.	CL3
			C416.4	Demonstrate the implementation, training and testing of an Artificial Neural Network using Back propagation Algorithm.	CL3
			C416.5	Demonstrate the use of Python ML libraries to implement Naïve Bayes Classifier, EM Algorithm, k-Means Clustering Algorithm, K-Nearest Neighbors Algorithm and Locally Weighted Regression algorithm.	CL3

CO-PO-PSO MAPPING:

CO No.	**Porgramme Outcomes (PO)**												**Programme Specific Outcome (PSO)**		
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
C416.1	**3**	**3**	**2**	**1**	**2**			**2**		**2**	1	2	**3**	**2**	
C416.2	**3**	**3**	**2**	**1**	**2**			**2**		**2**	1	2	**3**	**2**	
C416.3	**3**	**3**	**2**	**1**	**2**			**2**		**2**	1	2	**3**	**2**	
C416.4	**3**	**3**	**2**	**1**	**2**			**2**		**2**	1	2	**3**	**2**	
C416.5	**3**	**3**	**2**	**1**	**2**			**2**		**2**	1	2	**3**	**2**	

Introduction

Machine Learning is making the computer learn from studying data and statistics.Machine Learning is a step into the direction of artificial intelligence (AI). Machine Learning is a program that analyses data and learns to predict the outcome.

NumPy

It is useful component that makes Python as one of the favorite languages for Data Science. It basically stands for Numerical Python and consists of multidimensional array objects. By using NumPy, we can perform the following important operations −

- Mathematical and logical operations on arrays.
- Fourier transformation
- Operations associated with linear algebra.

Pandas

It is another useful Python library that makes Python one of the favorite languages for Data Science. Pandas is basically used for data manipulation, wrangling and analysis. With the help of Pandas, in data processing we can accomplish the following five steps −

- Load
- Prepare
- Manipulate
- Model
- Analyze
-

Data representation in Pandas

The entire representation of data in Pandas is done with the help of following three data structures −

Series, Data frame, Panel

```
import pandas as pd

import numpy as np

data = np.array(['g','a','u','r','a','v'])

Is = pd.Series(data)

print (s)
```

Scikit-learn

Another useful and most important python library for Data Science and machine learning in Python is Scikit-learn. The following are some features of Scikit-learn that makes it so useful −

- It is built on NumPy, SciPy, and Matplotlib.
- It is an open source and can be reused under BSD license.
- It is accessible to everybody and can be reused in various contexts.

- Wide range of machine learning algorithms covering major areas of ML like classification, clustering, regression, dimensionality reduction, model selection etc. can be implemented with the help of it

In Machine Learning (and in mathematics) there are often three values that interests us:

- **Mean** - The average value
- **Median** - The mid point value
- **Mode** - The most common value

```
import numpy

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

x = numpy.mean(speed)

print(x)
```

```
import numpy

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

x = numpy.median(speed)

print(x)
```

```
import numpy

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

x = numpy.median(speed)

print(x)
```

Mode

The Mode value is the value that appears the most number of times:

The SciPy module has a method for this

```
from scipy import stats
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = stats.mode(speed)
print(x)
```

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range

```
import numpy

speed = [86,87,88,86,87,85,86]

x = numpy.std(speed)

print(x)
```

Variance

Variance is another number that indicates how spread out the values are.

If you take the square root of the variance, you get the standard deviation

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.var(speed)

print(x)
```

Data Distribution

```
import numpy

x = numpy.random.uniform(0.0, 5.0, 250)

print(x)
```

Methods to Load CSV Data File

While working with ML projects, the most crucial task is to load the data properly into it. The most common data format for ML projects is CSV and it comes in various flavors and varying difficulties to parse. In this section, we are going to discuss about three common approaches in Python to load CSV data file .

Load CSV with Python Standard Library

```
import csv
import numpy as np
path = r"c:\iris.csv"
with open(path,'r') as f:
  reader = csv.reader(f,delimiter = ',')
  headers = next(reader)
  data = list(reader)
  data = np.array(data).astype(float)
print(data.shape)
print(data[:3])
```

Load CSV with NumPy

```
from numpy import loadtxt
path = r"C:\pima-indians-diabetes.csv"
datapath= open(path, 'r')
data = loadtxt(datapath, delimiter=",")
print(data.shape)
print(data[:3])
```

Load CSV with Pandas

```
from pandas import read_csv
path = r"C:\iris.csv"
data = read_csv(path)
print(data.shape)
print(data[:3])
```

Histogram

To visualize the data set we can draw a histogram with the data we collected.

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 250)

plt.hist(x, 5)
plt.show()
```

Big Data Distributions

Create an array with 100000 random numbers,and display them using histogram with 100 bars

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 100000)

plt.hist(x, 100)
plt.show()
```

Normal Data Distribution

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```

Scatter Plot

A scatter plot is a diagram where each value in the data set is represented by a dot.

```
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()
```

Random Data Distributions

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

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

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()
```

Draw a polynomial regression line through the data points:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

1. Implement A* Search algorithm.

[ ]:
```
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_neighbours(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 Doesn\'t 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 Doesn\'t Exist!')
    return None
```

[ ]:
```
def get_neighbours(v):
    if v in graph_nodes:
        return graph_nodes[v]
    else:
        return None
```

[ ]:
```
def heuristic(n):
    H_dist = {
        'A' : 10,
        'B' : 8,
        'C' : 5,
        'D' : 7,
        'E' : 3,
        'F' : 6,
        'G' : 5,
        'H' : 4,
        'I' : 1,
        'J' : 0
    }
    return H_dist[n]
```

[ ]:
```
graph_nodes = {
    '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)]
}
```

[ ]:
```
astarAlgo('A','J')
```

```
Path Found :  ['A', 'F', 'G', 'I', 'J']
```

[ ]:
```
['A', 'F', 'G', 'I', 'J']
```

2. Implement AO* Search algorithm.

[ ]:
```
def recAOStar(n):
    global finalPath
    print("Expanding Node :", n)
    and_nodes = []
    or_nodes = []

    if (n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes = allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes = allNodes[n]['OR']
    if len(and_nodes) == 0 and len(or_nodes) == 0:
        return
    solvable = False
    marked = {}

    while not solvable:

        if len(marked) == len(and_nodes) + len(or_nodes):
            min_cost_least, min_cost_group_least = least_cost_group(and_nodes,␣
 ↪or_nodes, {})
            solvable = True
            change_heuristic(n, min_cost_least)
            optimal_child_group[n] = min_cost_group_least
            continue

        min_cost, min_cost_group = least_cost_group(and_nodes, or_nodes, marked)
        is_expanded = False


        if len(min_cost_group) > 1:
            if (min_cost_group[0] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[0])
            if (min_cost_group[1] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[1])
```

```
        else:
            if (min_cost_group in allNodes):
                is_expanded = True
                recAOStar(min_cost_group)

        if is_expanded:
            min_cost_verify, min_cost_group_verify =␣
↪least_cost_group(and_nodes, or_nodes, {})
            if min_cost_group == min_cost_group_verify:
                solvable = True
                change_heuristic(n, min_cost_verify)
                optimal_child_group[n] = min_cost_group

        else:
              solvable = True
              change_heuristic(n, min_cost)
              optimal_child_group[n] = min_cost_group

        marked[min_cost_group] = 1
    return heuristic(n)
```

[ ]:
```
def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = {}

    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
    for node in or_nodes:
        if not node in marked:
            cost = 0
            cost = cost + heuristic(node) + 1
            node_wise_cost[node] = cost

    min_cost = 999999
    min_cost_group = None

    for costKey in node_wise_cost:
        if node_wise_cost[costKey] < min_cost:
            min_cost = node_wise_cost[costKey]
            min_cost_group = costKey
    return [min_cost, min_cost_group]
```

[ ]:
```
def heuristic(n):
    return H_dist[n]
```

[ ]:
```
def change_heuristic(n, cost):
    H_dist[n] = cost
    return
```

[ ]:
```
def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]

    if len(node) > 1:
          if node[0] in optimal_child_group:
              print("->", end="")
              print_path(node[0])
          if node[1] in optimal_child_group:
              print("->", end="")
              print_path(node[1])
    else:
          if node in optimal_child_group:
              print("->", end="")
              print_path(node)
```

[ ]:
```
H_dist = {
    'A': -1,
    'B': 4,
    'C': 2,
    'D': 3,
    'E': 6,
    'F': 8,
    'G': 2,
    'H': 0,
    'I': 0,
    'J': 0
}
```

[ ]:
```
allNodes = {
      'A': {'AND': [('C', 'D')], 'OR': ['B']},
      'B': {'OR': ['E', 'F']},
      'C': {'AND': [('H', 'I')], 'OR': ['G']},
      'D': {'OR': ['J']}
}
```

[ ]:
```
optimal_child_group = {}
optimal_cost = recAOStar('A')

print('Nodes Which Gives Optimal Cost Are')
print_path('A')
print('\nOptimal Cost Is :', optimal_cost)
```

```
Expanding Node : A
Expanding Node : B
Expanding Node : C
Expanding Node : D
Nodes Which Gives Optimal Cost Are
CD->HI->J
Optimal Cost Is : 5
```

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 pandas as pd
df = pd.read_csv('Datasets/EnjoySports.csv')
# df = df.drop(['slno'], axis = 1)
concepts = df.values[:,:-1]
target = df.values[:,-1]
df.head()
```

[ ]:

	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

[ ]:
```
def learn(concepts,target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in␣
 ↪range(len(specific_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] = '?'

    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(f" Final S : {s_final}")
print(f" Final G : {g_final}")
```

```
 Final S : ['sunny' 'warm' '?' 'strong' '?' '?']
 Final G : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?']]
```

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 to classify a new sample.

[ ]:
```
def infoGain(P, N):

    import math
    return -P / (P + N) * math.log2(P / (P + N)) - N / (P + N) * math.log2(N /␣
   (P + N))
```

[ ]:
```
def insertNode(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:

        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node: 'None'}
    return tree
```

[ ]:
```
def insertConcept(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:

        tree[addTo] = Node
    return tree
```

[ ]:
```
def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:

        return tree

    countC = {}
    for cVal in conceptVals:
```

```
        dataCC = data[data[concept] == cVal]

        countC[cVal] = dataCC.shape[0]


    if countC[conceptVals[0]] == 0:


        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree
    if countC[conceptVals[1]] == 0:


        tree = insertConcept(tree, addTo, conceptVals[0])
        return tree

    ClassEntropy = infoGain(countC[conceptVals[0]], countC[conceptVals[1]])


    Attr = {}

    for a in AttributeList:
        Attr[a] = list(set(data[a]))

    AttrCount = {}

    EntropyAttr = {}

    for att in Attr:
        for vals in Attr[att]:
            for c in conceptVals:
                iData = data[data[att] == vals]

                dataAtt = iData[iData[concept] == c]

                AttrCount[c] = dataAtt.shape[0]

            TotalInfo = AttrCount[conceptVals[0]] + AttrCount[conceptVals[1]]

            if AttrCount[conceptVals[0]] == 0 or AttrCount[conceptVals[1]] == 0:
                InfoGain = 0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[0]],␣
↪AttrCount[conceptVals[1]])
```

```
            if att not in EntropyAttr:

                EntropyAttr[att] = (TotalInfo / Total) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + (TotalInfo / Total) *␣
 ↪InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]


    Node = max(Gain, key=Gain.get)


    tree = insertNode(tree, addTo, Node)

    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)

        newData = data[data[Node] == nD].drop(Node, axis=1)


        AttributeList = list(newData)[:-1]

        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree,␣
 ↪nD)
    return tree
```

[ ]:
```
import pandas as pd
def main():
    data = pd.read_csv('Datasets/PlayTennis.csv')
 ↪
    if 'Unnamed: 0' in data.columns:
        data = data.drop('Unnamed: 0', axis=1)
#       data = data.drop('slno', axis=1)
    AttributeList = list(data)[:-1]
 ↪
    concept = str(list(data)[-1])
 ↪
    conceptVals = list(set(data[concept]))
 ↪
    tree = getNextNode(data, AttributeList, concept, conceptVals,
                       {'root': 'None'}, 'root')
    return tree
 ↪
```

[ ]:
```
tree = main()['root']
```

[ ]:
```
df = pd.read_csv('Datasets/PlayTennis.csv')

def test(tree, d):
    for k in tree:
        for v in tree[k]:
            if (d[k] == v and isinstance(tree[k][v], dict)):
                test(tree[k][v], d)
            elif (d[k] == v):
                print("Classification: " + tree[k][v])
```

[ ]:
```
if 'Unnamed: 0' in df.columns:
    df = df.drop('Unnamed: 0', axis=1)
df.head()
```

[ ]:

	Outlook	Temperature	Humidity	Wind	PlayTennis
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

[ ]:
```
print(tree)
test(tree, df.loc[0, :])
```

```
{'Outlook': {'Overcast': 'Yes', 'Sunny': {'Humidity': {'Normal': 'Yes', 'High':
'No'}}, 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}}}
Classification: No
```

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 = 1000
learning_rate = 0.6
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

[ ]:
```
wh = np.random.uniform(size = (inputlayer_neurons,
                               hiddenlayer_neurons))

bh = np.random.uniform(size = (1, hiddenlayer_neurons))

wo = np.random.uniform(size = (hiddenlayer_neurons,
                               output_neurons))

bo = np.random.uniform(size = (1, output_neurons))
```

[ ]:
```
for i in range(epoch):


    net_h = np.dot(X, wh) + bh
```

```
    sigma_h = sigmoid(net_h)

    net_o = np.dot(sigma_h, wo) + bo

    output = sigmoid(net_o)



    deltaK = (y - output) * derivatives_sigmoid(output)
    deltaH = deltaK.dot(wo.T) * derivatives_sigmoid(sigma_h)
    wo = wo + sigma_h.T.dot(deltaK) * learning_rate
    wh = wh + X.T.dot(deltaH) * learning_rate
```

[ ]:
```
print ("Input: \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n", output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89347243]
 [0.88144644]
 [0.89533573]]
```

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

[ ]:
```
def probAttr(data, attr, val):
    Total = data.shape[0]
    cnt = len(data[data[attr] == val])

    return cnt, cnt / Total
```

[ ]:
```
def train(data, Attr, conceptVals, concept):
    conceptProbs = {}
    countConcept = {}
    for cVal in conceptVals:
        countConcept[cVal], conceptProbs[cVal] = probAttr(data, concept, cVal)

    AttrConcept = {}
    probability_list = {}
    for att in Attr:
        probability_list[att] = {}
        AttrConcept[att] = {}
        for val in Attr[att]:
            AttrConcept[att][val] = {}
            a,probability_list[att][val] = probAttr(data, att, val)
            for cVal in conceptVals:
                dataTemp = data[data[att]==val]

                AttrConcept[att][val][cVal] = len(dataTemp[dataTemp[concept] ==␣
↪cVal]) / countConcept[cVal]

    print("P(A) : ", conceptProbs,"\n")
    print("P(X/A) : ", AttrConcept,"\n")
```

```
    print("P(X) : ", probability_list,"\n")
    return conceptProbs, AttrConcept, probability_list
```

[ ]:
```
def test(examples, Attr, concept_list, conceptProbs, AttrConcept,␣
↪probability_list):
    misclassification_count = 0
    Total = len(examples)                                                    ␣
↪
    for ex in examples:
        px = {}                                                              ␣
↪
        for a in Attr:                                                       ␣
↪
            for x in ex:                                                     ␣
↪
                for c in concept_list:                                       ␣
↪
                    if x in AttrConcept[a]:                                  ␣
↪                                                                          ␣
↪
                        if c not in px:                                      ␣
↪                                                                         ␣
↪
                            px[c] = conceptProbs[c] * AttrConcept[a][x][c] /␣
↪probability_list[a][x]
                        else:                                                ␣
↪
                            px[c] = px[c] * AttrConcept[a][x][c] /␣
↪probability_list[a][x]
        print(px)
        classification = max(px, key = px.get)                               ␣
↪
        print("Classification :", classification, "Expected :", ex[-1])
        if(classification != ex[-1]):
            misclassification_count += 1
    misclassification_rate = misclassification_count * 100 / Total
    accuracy = 100 - misclassification_rate
    print("Misclassification Count = {}".format(misclassification_count))
    print("Misclassification Rate = {}%".format(misclassification_rate))
    print("Accuracy = {}%".format(accuracy))
```

[ ]:
```
import pandas as pd
data = pd.read_csv('Datasets/PlayTennis.csv')
data.drop(['Unnamed: 0'], axis = 1, inplace = True)
print(data)
concept = str(list(data)[-1])
```

```
print(concept)
concept_list = set(data[concept])
print(concept_list)
Attr = {}
for a in list(data)[:-1]:
    Attr[a] = set(data[a])
    print(Attr[a])
conceptProbs, AttrConcept, probability_list = train(data, Attr, concept_list,␣
 ↪concept)

examples = pd.read_csv('Datasets/PlayTennis.csv')
test(examples.values, Attr, concept_list, conceptProbs, AttrConcept,␣
 ↪probability_list)
```

```
     Outlook Temperature Humidity    Wind PlayTennis
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
PlayTennis
{'No', 'Yes'}
{'Overcast', 'Sunny', 'Rain'}
{'Hot', 'Mild', 'Cool'}
{'Normal', 'High'}
{'Strong', 'Weak'}
P(A) :  {'No': 0.35714285714285715, 'Yes': 0.6428571428571429}

P(X/A) :  {'Outlook': {'Overcast': {'No': 0.0, 'Yes': 0.4444444444444444},
'Sunny': {'No': 0.6, 'Yes': 0.2222222222222222}, 'Rain': {'No': 0.4, 'Yes':
0.3333333333333333}}, 'Temperature': {'Hot': {'No': 0.4, 'Yes':
0.2222222222222222}, 'Mild': {'No': 0.4, 'Yes': 0.4444444444444444}, 'Cool':
{'No': 0.2, 'Yes': 0.3333333333333333}}, 'Humidity': {'Normal': {'No': 0.2,
'Yes': 0.6666666666666666}, 'High': {'No': 0.8, 'Yes': 0.3333333333333333}},
'Wind': {'Strong': {'No': 0.6, 'Yes': 0.3333333333333333}, 'Weak': {'No': 0.4,
'Yes': 0.6666666666666666}}}

P(X) :  {'Outlook': {'Overcast': 0.2857142857142857, 'Sunny':
```

```
0.35714285714285715, 'Rain': 0.35714285714285715}, 'Temperature': {'Hot':
0.2857142857142857, 'Mild': 0.42857142857142855, 'Cool': 0.2857142857142857},
'Humidity': {'Normal': 0.5, 'High': 0.5}, 'Wind': {'Strong':
0.42857142857142855, 'Weak': 0.5714285714285714}}

{'No': 0.9408000000000002, 'Yes': 0.2419753086419753}
Classification : No Expected : No
{'No': 1.8816000000000002, 'Yes': 0.16131687242798354}
Classification : No Expected : No
{'No': 0.0, 'Yes': 0.6049382716049383}
Classification : Yes Expected : Yes
{'No': 0.4181333333333335, 'Yes': 0.4839506172839506}
Classification : Yes Expected : Yes
{'No': 0.07840000000000004, 'Yes': 1.0888888888888888}
Classification : Yes Expected : Yes
{'No': 0.15680000000000005, 'Yes': 0.7259259259259259}
Classification : Yes Expected : No
{'No': 0.0, 'Yes': 1.2098765432098766}
Classification : Yes Expected : Yes
{'No': 0.6272000000000001, 'Yes': 0.3226337448559671}
Classification : No Expected : No
{'No': 0.11760000000000002, 'Yes': 0.7259259259259256}
Classification : Yes Expected : Yes
{'No': 0.10453333333333338, 'Yes': 0.9679012345679012}
Classification : Yes Expected : Yes
{'No': 0.31360000000000005, 'Yes': 0.43017832647462273}
Classification : Yes Expected : Yes
{'No': 0.0, 'Yes': 0.5377229080932785}
Classification : Yes Expected : Yes
{'No': 0.0, 'Yes': 1.2098765432098766}
Classification : Yes Expected : Yes
{'No': 0.8362666666666669, 'Yes': 0.3226337448559671}
Classification : No Expected : No
Misclassification Count = 1
Misclassification Rate = 7.142857142857143%
Accuracy = 92.85714285714286%
```

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


model = KMeans(n_clusters = 3)

model.fit(X)
```

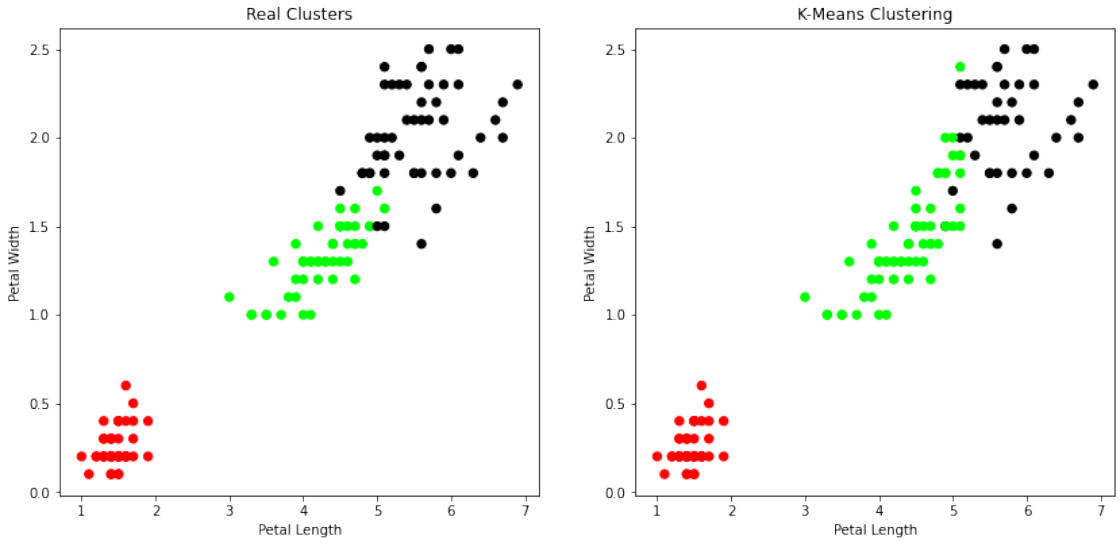
[ ]:
```
KMeans(n_clusters=3)
```

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

plt.subplot(2, 2, 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(2, 2, 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')
```

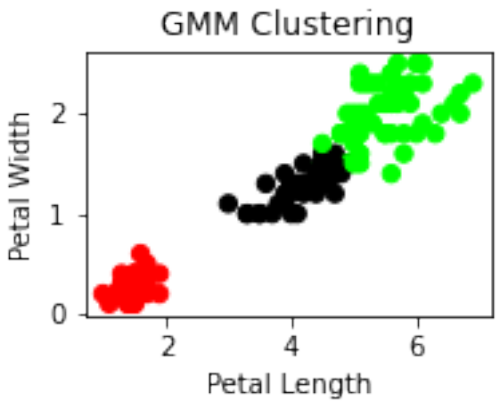
[ ]:
```
Text(0, 0.5, 'Petal Width')
```



[ ]:
```
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 = 3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c = colormap[gmm_y], 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 K-Means.')
```

```
Observation: The GMM Using EM Algorithm Based Clustering Matched The True Labels
More Closely Than The K-Means.
```


GMM Clustering
Petal Width
Petal Length
0
1
2
2
4
6

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

[ ]:
```
dataset = load_iris()
X_train,X_test,y_train,y_test =␣
↪train_test_split(dataset["data"],dataset["target"],random_state = 0)
```

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

[ ]:
```
KNeighborsClassifier(n_neighbors=3)
```

[ ]:
```
prediction = kn.predict(X_test)
confusion_matrix(y_test, prediction)
```

[ ]:
```
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  0,  9]], dtype=int64)
```

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

[ ]:
```
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
```

[ ]:
```
def main():
    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-")
```

[ ]:
```
main()
```

```
Matplotlib is building the font cache; this may take a moment.
```