JSS Academy of Technical Education, Bangalore
Department Of Computer Science & Engineering

Academic Year: 2021-22

LAB MANUAL

Artificial Intelligence Machine Learning Laboratory

Prepared by

Dr. P B Mallikarjuna
Mrs. Bhavani B H
Mr. Sreenatha M

Subject Code: 18CSL76
(CBCS Scheme)

Dept. of Computer Science and Engineering,
JSSATE, Bengaluru

Co No.	CO Description	Bloom's Level
C406.1	Apply optimal method for Searching and to find hypothesis version space for any concept learning task	L3
C406.2	Develop real world applications using non-parametric base Machine learning models	L3
C406.3	Build real world applications using parametric based Machine learning models	L3
C406.4	Apply clustering techniques to solve real world problems	L3

Justification of CO – PO Mapping:

Cos/POs		POs											
		1	2	3	4	5	6	7	8	9	10	11	12
Artificial Intelligence and Machine Learning Laboratory	C408.1	3	3	3	2	2	-	-	-	-	-	-	1
	C408.2	3	3	3	2	2	-	-	-	-	-	-	1
	C408.3	3	3	3	2	2	-	-	-	-	-	-	1
	C408.4	3	3	3	2	2	-	-	-	-	-	-	1
	All Cos	3	3	3	2	2	-	-	-	-	-	-	1

CO	PO	Value	Justification
C408.1	PO1	3	The basic knowledge on mathematics, science, and engineering fundamentals is required while understanding optimal method to find hypothesis version space for any concept learning task
	PO2	3	By understanding optimal method to find hypothesis version space for any concept learning task and while solving problems using machine learning technique the analysis of problems are to be done.
	PO3	3	After understanding optimal method to find hypothesis version space for any concept learning task the development of solution has to be done.
	PO4	2	Detailed investigation of the problem is required before development of solution using the optimal method to find hypothesis version space for any concept learning task
	PO5	2	By understanding optimal method to find hypothesis version space for any concept learning task the usage of modern tools is required.
	PO12	1	Understanding optimal method to find hypothesis version space for any concept learning task should to continues learning to solve real world problems.
C408.2	PO1	3	The basic knowledge on mathematics, science, and engineering fundamentals is required for real world applications using non-parametric base Machine learning models
	PO2	3	By understanding real world applications using non-parametric

			base Machine learning models for classification and while solving problems using machine learning technique the analysis of problems are to be done.
	PO3	3	After understanding real world applications using non-parametric base Machine learning models using machine learning the development of solution has to be done.
	PO4	2	Detailed investigation of the problem is required before development of solution using the real world applications using non-parametric base Machine learning models
	PO5	2	By understanding real world applications using non-parametric base Machine learning models and while solving problems using machine learning technique the usage of modern tools is required.
	PO12	1	Understanding real world applications using non-parametric base Machine learning models using machine learning technique should to continues learning to solve real world problems.
C408.3	PO1	3	The basic knowledge on mathematics, science, and engineering fundamentals is required while understanding neural network models to solve linear and non-linear classification.
	PO2	3	By understanding real world applications using parametric based Machine learning models and while solving problems using machine learning technique the analysis of problems are to be done.
	PO3	3	After understanding real world applications using parametric based Machine learning models using machine learning the development of solution has to be done.
	PO4	2	Detailed investigation of the problem is required before development of solution using the neural network real world applications using parametric based Machine learning models
	PO5	2	By understanding neural network real world applications using parametric based Machine learning models and while solving problems using machine learning technique the usage of modern tools is required.
	PO12	1	Understanding real world applications using parametric based Machine learning models using machine learning technique should to continues learning to real world problems.
C408.4	PO1	3	The basic knowledge on mathematics, science, and engineering fundamentals is required while understanding clustering techniques to solve real world problems
	PO2	3	By understanding clustering techniques to solve real world problems. and while solving problems using machine learning technique the analysis of problems are to be done.
	PO3	3	After understanding clustering techniques to solve real world problems using machine learning the development of solution has to be done.
	PO4	2	Detailed investigation of the problem is required before development of solution using the clustering techniques to solve real world problems
	PO5	2	By understanding clustering techniques to solve real world problems and while solving problems using machine learning technique the usage of modern tools is required.
	PO12	1	Understanding clustering techniques to solve real world problems using machine learning technique should to continues

			learning to solve real world problems.

CO- PSO MAPPING

Cos/PSOs		PSOs			
		1	2	3	4
Artificial Intelligence and Machine Learning Laboratory	C408.1	2	3	3	-
	C408.2	2	3	3	-
	C408.3	2	3	3	-
	C408.4	2	3	3	-
	All Cos	2	3	3	-

Justification of CO – PSO Mapping:

CO	PSO	Value	Justification
C408.1	PSO1	2	To solve the real world problems using optimal method to find hypothesis version space for any concept learning task the principles of basic engineering science are necessary.
	PSO2	3	Analyze the real world applications using algorithmic models before solving the problems using optimal method to find hypothesis version space for any concept learning task
	PSO3	3	While solving the problems using optimal method to find hypothesis version space for any concept learning task the design and development of applications using various software and hardware tools is necessary.
C408.2	PSO1	2	To solve the real world problems using non-parametric base Machine learning models for classification the principles of basic engineering science are necessary.
	PSO2	3	Analyze the real world applications using algorithmic models before solving the problems using non-parametric base Machine learning models.
	PSO3	3	While solving the problems using non-parametric base Machine learning models the design and development of applications using various software and hardware tools is necessary.
C408.3	PSO1	2	To solve the real world problems using parametric based Machine learning models the principles of basic engineering science are necessary.
	PSO2	3	Analyze the real world applications using algorithmic models before solving the problems using parametric based Machine learning models
	PSO3	3	While solving the problems using parametric based Machine learning models the design and development of applications using various software and hardware tools is necessary.
C408.4	PSO1	2	To solve the real world problems using clustering techniques to solve real world problems the principles of basic engineering science are necessary.
	PSO2	3	Analyze the real world applications using algorithmic models before solving the problems using clustering techniques to solve real world problems.
	PSO3	3	While solving the problems using clustering techniques to solve

			real world problems. the design and development of applications using various software and hardware tools is necessary.

CONTENTS:

Expt No.	Title of the Experiments	RBT	CO
1	Implement A* Search algorithm	L3	CO 1,2,3,4
2	Implement AO* Search algorithm.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
5	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
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.	L3	CO 1,2,3,4
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		CO 1,2,3,4

REFERENCE:

1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, 2_{nd} edition, Springer series in statistics.
3. Ethem Alpaydın, Introduction to machine learning, second edition, MIT press.

-

C. EVALUATION SCHEME

Experiment distribution:

For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.

For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (Course to change in accordance with university regulations) q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks r) For laboratories having PART A and PART B i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

---

1. EXPERIMENT NO: 1
2. TITLE: **A* SEARCHING**
3. LEARNING OBJECTIVES:
   - Make use of algorithm for Optimized Searching.
   - Implement ML concepts and algorithms in Python
4. AIM:
   - Implement and demonstrate the A* Searching algorithm for finding the most optimized searching.

5. PROCEDURE / PROGRAMME :

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


def aStarAlgo(start_node, stop_node):

        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {} # parents contains an adjacency map of all nodes

        #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node


        while len(open_set) > 0:
           n = None

           #node with lowest f() is found
           for v in open_set:
              if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                 n = v
```

```
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
              #nodes 'm' not in first and last set are added to first
               #n is set its parent
               if m not in open_set and m not in closed_set:
                  open_set.add(m)
                  parents[m] = n
                  g[m] = g[n] + weight


               #for each node m,compare its distance from start i.e g(m) to the
               #from start through n node
               else:
                  if g[m] > g[n] + weight:
                     #update g(m)
                     g[m] = g[n] + weight
                     #change parent of m to n
                     parents[m] = n

                     #if m in closed set,remove and add to open
                     if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)

        if n == None:
            print('Path does not exist!')
            return None

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
        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


        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)

    print('Path does not exist!')
    return None

#define fuction to return neighbor and its distance
#from the passed node
```

```
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
        H_dist = {
            'A': 10,
            'B': 8,
            'C': 5,
            'D': 7,
            'E': 3,
            'F': 6,
            'G': 5,
            'H': 3,
            'I': 1,
            'J': 0
        }

        return H_dist[n]

#Describe your graph here
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')
```

9. APPLICATION AREAS:

- Classification based problems.

1. EXPERIMENT NO: 2
TITLE: Implement AO* Search algorithm.
2. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

PROGRAME :

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):  #instantiate graph object with graph topology,
heuristic values, start node
```

```
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self):        # starts a recursive AO* algorithm
        self.aoStar(self.start, False)

    def getNeighbors(self, v):    # gets the Neighbors of a given node
        return self.graph.get(v,'')

    def getStatus(self,v):        # return the status of a given node
        return self.status.get(v,0)

    def setStatus(self,v, val):   # set the status of a given node
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)    # always return the heuristic value of a given node

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value           # set the revised heuristic value of a given node


    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:",self.start)
        print("------------------------------------------------------------")
        print(self.solutionGraph)
        print("------------------------------------------------------------")

    def computeMinimumCostChildNodes(self, v):  # Computes the Minimum Cost of child nodes of a given
node v
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v):  # iterate over all the set of child node/s
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)


            if flag==True:                  # initialize Minimum Cost with the cost of first set of child node/s
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList      # set the Minimum Cost child node/s
                flag=False

                #print("cost to child node",costToChildNodeListDict)
            else:                           # checking the Minimum Cost nodes with the current Minimum Cost
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList  # set the Minimum Cost child node/s


        return minimumCost, costToChildNodeListDict[minimumCost]   # return Minimum Cost and Minimum
Cost child node/s


    def aoStar(self, v, backTracking):    # AO* algorithm for a start node and backTracking status flag

        print("HEURISTIC VALUES  :", self.H)
        print("SOLUTION GRAPH    :", self.solutionGraph)
        print("PROCESSING NODE   :", v)
        print("-------------------------------------------------------------------------------------------")

        if self.getStatus(v) >= 0:        # if status node v >= 0, compute Minimum Cost nodes of v
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
```

```
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v,len(childNodeList))

        solved=True             # check the Minimum Cost nodes of v are solved
        for childNode in childNodeList:
            self.parent[childNode]=v
            if self.getStatus(childNode)!=-1:
                solved=solved & False

        if solved==True:        # if the Minimum Cost nodes of v are solved, set the current node status
as solved(-1)
            self.setStatus(v,-1)
            self.solutionGraph[v]=childNodeList # update the solution graph with the solved nodes which
may be a part of solution


        if v != self.start:     # check the current node is the start node for backtracking the current node
value
            self.aoStar(self.parent[v], True)   # backtracking the current node value with backtracking status
set to true

        if backTracking==False:     # check the current call is not for backtracking
            for childNode in childNodeList:   # for each Minimum Cost child node
                self.setStatus(childNode,0)   # set the status of child node to 0(needs exploration)
                self.aoStar(childNode, False) # Minimum Cost child node is further explored with
backtracking status as false



# h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
# graph1 = {
#     'A': [[('B', 1), ('C', 1)], [('D', 1)]],
#     'B': [[('G', 1)], [('H', 1)]],
#     'C': [[('J', 1)]],
#     'D': [[('E', 1), ('F', 1)]],
#     'G': [[('I', 1)]]
# }
# G1= Graph(graph1, h1, 'A')
# G1.applyAOStar()
# G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}  # Heuristic values of Nodes
graph2 = {                                  # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],      # Neighbors of Node 'A', B, C & D with repective weights
    'B': [[('G', 1)], [('H', 1)]],              # Neighbors are included in a list of lists
    'D': [[('E', 1), ('F', 1)]]                 # Each sublist indicate a "OR" node or "AND" nodes
}

G2 = Graph(graph2, h2, 'A')                     # Instantiate Graph object with graph, heuristic values and start
Node
G2.applyAOStar()                            # Run the AO* algorithm
G2.printSolution()                          # Print the solution graph as output of the AO* algorithm searchdef
candidate_elimination(examples):
    domains = get_domains(examples)[:-1]
    n = len(domains)
    G = set([("?",)*n])
    S = set([("0",)*n])

    print("Maximally specific hypotheses - S ")
    print("Maximally general hypotheses - G ")

    i=0
    print("\nS[0]:",str(S),"\nG[0]:",str(G))
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions
        if cx=='Y': # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not fulfills(x, s)}
```

```
            G = specialize_G(x, domains, G, S)
        print("\nS[{0}]:".format(i),S)
        print("G[{0}]:".format(i),G)
    return

with open('data22_sports.csv') as csvFile:
    examples = [tuple(line) for line in csv.reader(csvFile)]

candidate_elimination(examples)
```

8. APPLICATION AREAS:
   - Classification based problems.

-

1. EXPERIMENT NO: 3
2. TITLE: Candidate Elimination
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

Program:

```
import csv
def get_domains(examples):
    d = [set() for i in examples[0]]

    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]

def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        #print("More:",x,y)
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
# the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != '0' else x[i]
    return [tuple(h_new)]

def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
```

```
                results.append(h_new)
        elif h[i] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results

def generalize_S(x, G, S):
    S_prev = list(S)
    #print("we are in",S_prev)
    for s in S_prev:
        #print(s)
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h) for g in G])])
            #print("we are", typeof(S))
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if any([more_general(h, h1) for h1 in S if h !=
h1])])

    return S

def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s) for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if any([more_general(g1, h) for g1 in G if h !=
g1])])

    return G

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1] # excluding last column
    print("It's me:",domains)
    n = len(domains)
    G = set([("?",)*n])
    S = set([("0",)*n])
    print("Maximally specific hypotheses - S ")
    print("Maximally general hypotheses - G ")
    i=0
    print("\nS[0]:",str(S),"\nG[0]:",str(G))
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1]
        if cx=='Y': # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else:
            S = {s for s in S if not fulfills(x, s)}
```

```
            G = specialize_G(x, domains, G, S)
        print("\nS[{0}]:".format(i),S)
        print("G[{0}]:".format(i),G)
    return

with open('3.csv') as csvFile:
    examples = [tuple(line) for line in csv.reader(csvFile)]
#print(examples)

candidate_elimination(examples)
```

1. EXPERIMENT NO: 4
2. TITLE: ID3 algorithm

4. AIM:

- Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

5. THEORY:

- ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".
- To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described below.

Algorithm: ID3(Examples, TargetAttribute, Attributes)
Input: Examples are the training examples.
Targetattribute is the attribute whose value is to be predicted by the tree.
Attributes is a list of other attributes that may be tested by the learned decision tree.
Output: Returns a decision tree that correctly classiJies the given
Examples Method:

1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,
   Return the single-node tree Root, with label = most common value of TargetAttribute in Examples
   Else
   A ← the attribute from Attributes that best classifies Examples The decision attribute for Root ←A For each possible value, vi, of A,
   Add a new tree branch below Root, corresponding to the test A = vi Let $Examples_{vi}$ be the subset of Examples that have value vi for A
   If $Examples_{vi}$ is empty Then below this new branch add a leaf node with label = most common value of TargetAttribute in Examples
   Else
   below this new branch add the subtree ID3($Examples_{vi}$, TargetAttribute, Attributes–{A})

End
Return Root

PROCEDURE / PROGRAM :

```
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
            self.answer = ""   # NULL indicates children exists. # Not
                                        Null indicates this is a Leaf Node

def subtables(data, col, delete):
    dic = {}
    coldata = [ row[col] for row in data]
    attr = list(set(coldata)) # All values of attribute retrived

    for k in attr:
         dic[k] = []

    for y in range(len(data)):
          key = data[y][col]
          if delete:
                del data[y][col]
          dic[key].append(data[y])

    return attr, dic

def entropy(S):
    attr = list(set(S))
    if len(attr) == 1: #if all are +ve/-ve then entropy = 0 return 0


    counts = [0,0] # Only two values possible 'yes' or 'no' for i in
    range(2):
        counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums

def compute_gain(data, col):
    attValues, dic = subtables(data, col, delete=False)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attValues)):
        ratio = len(dic[attValues[x]]) / ( len(data) * 1.0) entro =
        entropy([row[-1] for row in dic[attValues[x]]]) total_entropy -=
        ratio*entro

    return total_entropy
```

```
def build_tree(data, features):

    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1: # If all samples have same labels return that label
        node=Node("")

        node.answer = lastcol[0]
        return node

    n = len(data[0])-1
```

```
    gains = [compute_gain(data, col) for col in range(n) ]

    split = gains.index(max(gains)) # Find max gains and returns index node =
    Node(features[split]) # 'node' stores attribute selected #del (features[split])

    fea = features[:split]+features[split+1:]

    attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtables for x in
    range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))

    return node

def print_tree(node, level):
    if node.answer != "":
        print(" "*level, node.answer) # Displays leaf node yes/no return


    print(" "*level, node.attribute) # Displays attribute Name for value, n
    in node.children:
        print("     "*(level+1), value)
        print_tree(n, level + 2)

def classify(node,x_test,features):
    if node.answer != "":
        print(node.answer)
        return

    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

''' Main program '''
dataset, features = load_csv("data3.csv") # Read Tennis data node =
build_tree(dataset, features) # Build decision tree

print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)

testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance : ",xtest)
    print("The predicted label : ", end="")
    classify(node,xtest,features)
```

---

7. RESULTS & CONCLUSIONS:

Training instances: data3.csv

```
Outlook,Temperature,Humidity,Wind,Target
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
```

Testing instances: data3_test.csv

```
Outlook,Temperature,Humidity,Wind
rain,cool,normal,strong
sunny,mild,normal,strong
```

Output

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
     overcast
         yes
     rain
         Wind
             weak
                 yes
              strong
                  no
     sunny
         Humidity
             normal
                 yes
             high
                 no
The test instance       :  ['rain', 'cool', 'normal', 'strong']
The predicted label :  no
The test instance       :  ['sunny', 'mild', 'normal', 'strong']
The predicted label :       yes
```

8. LEARNING OUTCOMES :
   - The student will be able to demonstrate the working of the decision tree based ID3 algorithm, use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

9. APPLICATION AREAS:
   - Classification related prblem areas

4. TITLE: **BACKPROPAGATION ALGORITHM**
5. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python
6. AIM:
   - Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
7. THEORY:
   - Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
   - Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.
   - ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies.

Backpropogation algorithm

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do

For each training example <(x_1,…x_n),t>, do

// Propagate the input forward through the network:

a. Input the instance (x_1, ..,x_n) to the n/w & compute the n/w outputs o_k for every unit // Propagate the errors backward through the network:

```
import numpy as np # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)
y = np.array(([92], [86], [89]), dtype=float)                # Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize
y = y/100

def sigmoid(x):
    return 1/(1 + np.exp(-x))
def sigmoid_grad(x):
    return x * (1 - x)
```

```
# Variable initialization
epoch=1000                  #Setting training iterations
eta =0.2                    #Setting learning rate (eta)
input_neurons = 2           #number of features in data set
hidden_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
print("Normalized Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

7. RESULTS & CONCLUSIONS:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.89427812]
 [0.88503667]
 [0.89099058]]
```

8. LEARNING OUTCOMES :

- The student will be able to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

9. APPLICATION AREAS:

- Speech recognition, Character recognition, Human Face recognition

---

1. EXPERIMENT NO: 6
2. TITLE: **NAÏVE BAYESIAN CLASSIFIER**
3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.

- Implement ML concepts and algorithms in Python

4. AIM:

- 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.

5. THEORY:

Naive Bayes algorithm : Naive Bayes algorithm is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value,

---

v_{MAP}, given the attribute values (a_1, a_2, ..., a_n) that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Now we could attempt to estimate the two terms in Equation (19) based on the training data. It is easy to estimate each of the P(vj) simply by counting the frequency with which each target value vj occurs in the training data.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $a_1, a_2, \ldots, a_n$, is just the product of the probabilities for the individual attributes: $P(a_1, a_2, \ldots, a_n \mid v_j) = \Pi_i P(a_i \mid v_j)$.
Substituting this, we have the approach used by the naive Bayes classifier.

where v_{NB} denotes the target value output by the naive Bayes classifier.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class.

Let μ be the mean of the values in x associated with class Ck, and let σ^2_k be the variance of the values in x associated with class Ck. Suppose we have collected some observation value v. Then, the probability distribution of v given a class Ck, p(x=v|Ck) can be computed by plugging v into the equation for a Normal distribution parameterized by μ and σ^2_k . *That is*

Above method is adopted in our implementation of the program.

Pima Indian diabetis dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

---

6. PROCEDURE / PROGRAM :

```
import csv, random, math
import statistics as st

def loadCsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitDataset(dataset, splitRatio):
    testSize = int(len(dataset) * splitRatio);
    trainSet = list(dataset);
    testSet = []
    while len(testSet) < testSize:
   #randomly pick an instance from training data index =
        random.randrange(len(trainSet));
        testSet.append(trainSet.pop(index))
    return [trainSet, testSet]

#Create a dictionary of classes 1 and 0 where the values are the
#instacnes belonging to each class

def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        x = dataset[i]
        if (x[-1] not in separated):
            separated[x[-1]] = []
        separated[x[-1]].append(x)
    return separated

def compute_mean_std(dataset):
```

```
    mean_std = [ (st.mean(attribute), st.stdev(attribute))
                for attribute in zip(*dataset)]; #zip(*res) transposes a matrix (2-d array/list) del
    mean_std[-1] # Exclude label
    return mean_std

def summarizeByClass(dataset):
    separated = separateByClass(dataset);
    summary = {} # to store mean and std of +ve and -ve instances for
    classValue, instances in separated.items():
            #summaries is a dictionary of tuples(mean,std) for each class value
            summary[classValue] = compute_mean_std(instances)
    return summary

#For continuous attributes p is estimated using Gaussion distribution def
estimateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
```

```
def calculateClassProbabilities(summaries, testVector):
    p = {}
    #class and attribute information as mean and sd
    for classValue, classSummaries in summaries.items():
            p[classValue] = 1
            for i in range(len(classSummaries)):
                    mean, stdev = classSummaries[i]
                    x = testVector[i] #testvector's first attribute
                    #use normal distribution
                        p[classValue] *= estimateProbability(x, mean, stdev);
    return p

def predict(summaries, testVector):
    all_p = calculateClassProbabilities(summaries, testVector)
    bestLabel, bestProb = None, -1
    for lbl, p in all_p.items():#assigns that class which has he highest prob if
            bestLabel is None or p > bestProb:
                    bestProb = p
                    bestLabel = lbl
    return bestLabel

def perform_classification(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
            result = predict(summaries, testSet[i])
            predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
            if testSet[i][-1] == predictions[i]:
                    correct += 1
    return (correct/float(len(testSet))) * 100.0

dataset = loadCsv('data51.csv');
print('Pima Indian Diabetes Dataset loaded...')
print('Total instances available :',len(dataset))
print('Total attributes present :',len(dataset[0])-1)

print("First Five instances of dataset:")
for i in range(5):
   print(i+1 , ':' , dataset[i])

splitRatio = 0.2
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('\nDataset is split into training and testing set.')
print('Training examples = {0} \nTesting examples = {1}'.format(len(trainingSet), len(testSet)))
```

```
summaries = summarizeByClass(trainingSet); predictions =
perform_classification(summaries, testSet)
accuracy = getAccuracy(testSet, predictions)
print('\nAccuracy of the Naive Baysian Classifier is :', accuracy)
```

---

7. RESULTS & CONCLUSIONS:

Sample Result

```
Pima Indian Diabetes Dataset loaded...
Total instances available : 768
Total attributes present : 8
First Five instances of dataset:
1 : [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
2 : [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]
3 : [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]
4 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]
5 : [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set.
Training examples = 615
Testing examples = 153

Accuracy of the Naive Baysian Classifier is : 73.85
```

8. LEARNING OUTCOMES :

- The student will be able to apply naive baysian classifier for the relevent problem and analyse the results.

9. APPLICATION AREAS:

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

1. EXPERIMENT NO: 7
2. TITLE: **CLUSTERING BASED ON EM ALGORITHM AND K-MEANS**
3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM: 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.

5. THEORY:

Expectation Maximization algorithm

- The basic approach and logic of this clustering method is as follows.
- Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
- The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
- Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
- The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.
- In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.

K means Clustering

- The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
- The algorithm works as follows:
    1. First we initialize k points, called means, randomly.
    2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
    3. We repeat the process for a given number of iterations and at the end, we have our clusters.
- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).

- Pseudocode:
    1. Initialize k means with random values
    2. For a given number of iterations:
        Iterate through items:
            Find the mean closest to the item
            Assign item to mean
            Update mean

6. PROCEDURE / PROGRAMME :

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

```
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'] y =
pd.DataFrame(iris.target)
y.columns = ['Targets']

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

#  # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
#  Plot the Original Classifications using Petal features
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')
#  Plot the Models Classifications
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')

# General EM for GMM
from sklearn import preprocessing
#  transform your data such that its distribution will have a
#  mean value 0 and standard deviation of 1.
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 Kmeans.')
```

7. RESULTS & CONCLUSIONS:

Sample Output

Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.

8. LEARNING OUTCOMES :

- The students will be apble to apply EM algorithm and k-Means algorithm for clustering and anayse the results.

9. APPLICATION AREAS:

- Text mining
- Pattern recognition
- Image analysis
- Web cluster engines

---

1. EXPERIMENT NO: 8

2. TITLE: **K-NEAREST NEIGHBOUR**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

- 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.

5. THEORY:

- K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
- It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.
- Algorithm
  Input: Let m be the number of training data samples. Let p be an unknown point. Method:
  1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
  2. for i=0 to m
     Calculate Euclidean distance d(arr[i], p).
  3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
  4. Return the majority label among S.

6. PROCEDURE / PROGRAMME :

```
# import the required packages
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")

# Split the data into train and test samples
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names
for i in range(len(iris.target_names)):

    print("Label", i , "-",str(iris.target_names[i]))

# Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)

# Perform Training
classifier.fit(x_train, y_train)
# Perform testing
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
        print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-
                                                                label:", str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));

#from sklearn.metrics import classification_report, confusion_matrix
#print('Confusion Matrix')
#print(confusion_matrix(y_test,y_pred))
#print('Accuracy Metrics')
#print(classification_report(y_test,y_pred))
```

7. RESULTS & CONCLUSIONS:

Result-1

```
Iris Data set loaded...
Dataset is split into training and testing samples...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample:  [4.4   3. 1.3 0.2] Actual-label: 0   Predicted-label:  0
 Sample:  [5.1   2.5 3. 1.1] Actual-label: 1   Predicted-label:  1
 Sample:  [6.1   2.8 4. 1.3] Actual-label: 1   Predicted-label:  1
 Sample:  [6.   2.7 5.1 1.6] Actual-label: 1   Predicted-label:  2
 Sample:   [6.7 2.5 5.8 1.8] Actual-label: 2   Predicted-label:  2
 Sample:   [5.1 3.8 1.5 0.3] Actual-label: 0   Predicted-label:  0
```

Sample: [6.7 3.1 4.4 1.4] Actual-label: 1 Predicted-label: 1
Sample: [4.8 3.4 1.6 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.1 3.5 1.4 0.3] Actual-label: 0 Predicted-label: 0
Sample: [5.4 3.7 1.5 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.7 2.8 4.1 1.3] Actual-label: 1 Predicted-label: 1
Sample: [4.5 2.3 1.3 0.3] Actual-label: 0 Predicted-label: 0
Sample: [4.4 2.9 1.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.1 3.5 1.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [6.2 3.4 5.4 2.3] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 0.93

Result-2

Iris Data set loaded...
Dataset is split into training and testing samples...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
Sample: [6.5 3. 5.5 1.8] Actual-label: 2 Predicted-label: 2
4.1
Sample: [5.7 2.8 1.3] Actual-label: 1 Predicted-label: 1
Sample: [6.6 3. 4.4 1.4] Actual-label: 1 Predicted-label: 1
5.1
Sample: [6.9 3.1 2.3] Actual-label: 2 Predicted-label: 2
1.9
Sample: [5.1 3.8 0.4] Actual-label: 0 Predicted-label: 0
Sample: [7.2 3.2 6. 1.8] Actual-label: 2 Predicted-label: 2
4.4
Sample: [5.5 2.6 1.2] Actual-label: 1 Predicted-label: 1
Sample: [6. 2.9 4.5 1.5] Actual-label: 1 Predicted-label: 1
1.5
Sample: [5.1 3.7 0.4] Actual-label: 0 Predicted-label: 0
1.4
Sample: [5.2 3.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5. 3.5 1.6 0.6] Actual-label: 0 Predicted-label: 0
1.5
Sample: [4.9 3.1 0.1] Actual-label: 0 Predicted-label: 0
Sample: [5. 3. 1.6 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.7 3. 4.2 1.2] Actual-label: 1 Predicted-label: 1
5.1
Sample: [5.8 2.7 1.9] Actual-label: 2 Predicted-label: 2
Classification Accuracy : 1.0

8. LEARNING OUTCOMES :

- The student will be able to implement k-Nearest Neighbour algorithm to classify the iris data set and Print both correct and wrong predictions.

9. APPLICATION AREAS:

- Recommender systems
- Classification problems

10. REMARKS:

1. EXPERIMENT NO: 9
2. TITLE: **LOCALLY WEIGHTED REGRESSION ALGORITHM**
3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

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

5. THEORY:
   - Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
   - Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
   - Alternate approach is given by locally weighted regression.
   - Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
   - The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
   - The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.

   Algorithm

```
def local_regression(x0, X, Y, tau):
        # add bias term
        x0 = np.r_[1, x0]
        X = np.c_[np.ones(len(X)), X]

        # fit model: normal equations with kernel
        xw = X.T * radial_kernel(x0, X, tau) beta
        = np.linalg.pinv(xw @ X) @ xw @ Y

        # predict value
        return x0 @ beta
def radial_kernel(x0, X, tau):
        return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
```

6. PROCEDURE / PROGRAM :

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

```
def kernel(point,xmat, k):
    m,n = np.shape(xmat)
```

```
    weights = np.mat(np.eye((m))) # eye - identity matrix for j in
    range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest xsort =
    X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();

# load data points
data = pd.read_csv('data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data tip =

np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array mtip =
np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols #

increase k to get smooth curves

ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

8. LEARNING OUTCOMES :
    - To understand and implement linear regression and analyse the results with change in the parameters
9. APPLICATION AREAS:
    - Demand anaysis in business
    - Optimization of business processes
    - Forecasting
