SDM INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

VII SEMESTER

LAB MANUAL

SUBJECT: MACHINE LEARNING LABORATORY

SUBJECT CODE: 17CSL76

2020-2021

SYLLABUS

MACHINE LEARNING LABORATORY [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2017 - 2018) SEMESTER – VII			
Subject Code	17CSL76	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03
CREDITS – 02			

Description (If any):

1. The programs can be implemented in either JAVA or Python.
2. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.
3. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

Lab Experiments:

1. Implement and demonstrate the **FIND-S algorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.
3. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
4. Build an Artificial Neural Network by implementing the **Back propagation algorithm** and test the same using appropriate data sets.
5. Write a program to implement the **naive Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
6. Assuming a set of documents that need to be classified, use the **naive Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7. Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. 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.
9. 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.
10. Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment.

Course outcomes: The students should be able to:

1. Understand the implementation procedures for the machine learning algorithms.
2. Design Java/Python programs for various Learning algorithms.
3. Apply appropriate data sets to the Machine Learning algorithms.
4. Identify and apply Machine Learning algorithms to solve real world problems.

Conduction of Practical Examination:

All laboratory experiments are to be included for practical examination.

- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script
- Marks distribution: Procedure + Conduction + Viva:**15 + 70 +15 (100)**

Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

CONTENTS

INTRODUCTION TO MACHINE LEARNING

1) DEFINITION

Machine learning is a broad subfield of **artificial intelligence** which is concerned with the design and development of algorithms and techniques that allow computers to "learn". It can also be defined as the process by which a machine uses a sample training set to learn and then to generalize the data that it receives based on experience.

2) EXAMPLE

Handwriting analysis: Machine learning would involve the development of a computer algorithm to recognize and interpret a person's handwriting based on a particular sample set. Although this can be done with relative ease in the human brain, this form of artificial intelligence is very difficult to program in computers.

3) APPLICATIONS

Machine learning has a wide spectrum of applications which includes:

1) Natural language processing
2) Syntactic pattern recognition
3) Search engines
4) Medical diagnosis
5) Bioinformatics and cheminformatics
6) Detecting credit card fraud
7) Stock market analysis
8) Classifying DNA sequences
9) Speech and handwriting recognition
10) Object recognition in computer vision
11) Game playing and robot locomotion.

EXPERIMENT 1

a) Problem Statement: Implement and demonstrate the **FIND-S algorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

b) Description

Find-S algorithm is used to find maximally specific hypothesis which satisfies all given hypothesis. A model that approximates the target function and performs mappings of inputs to outputs is called hypothesis. Find-S algorithm considers only positive data sample & ignores negative data sample. To implement this algorithm the enjoy sport data set has been considered. This consists of 6 attributes namely: sky, air temperature, humidity, wind, water & forecast. The attribute sky has two possible values: sunny & rainy. The attribute air temperature has two possible values: warm & cold. The attribute humidity has two possible values: normal & high. The attribute wind has one possible value: strong. The attribute water has two possible values: warm & cool. The attribute forecast has two possible values: same & change. First column in the dataset is example that indicates day 1, 2, 3 & 4. EnjoySport ( ) function is the concept to be learnt which is denoted as follows: C(X) = {Yes, No} where Yes & No are EnjoySport values. From the given dataset, 4 hypothesis h1, h2, h3, h4 are obtained.

c) Algorithm: The FIND-S algorithm is discussed below:

1. Initialize **h** to the most specific hypothesis in **H**
2. **For** each positive training instance **x**
   - **For** each attribute constraint $\mathbf{a_i}$ in **h**
     If the constraint $\mathbf{a_i}$ in **h** is satisfied by **x** then do nothing
     else replace $\mathbf{a_i}$ in h by the next more general constraint that is satisfied by **x**
3. Output hypothesis **h**

Illustration

Step 1: Initialize h to the most specific hypothesis in H

$$h0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

Step 2:

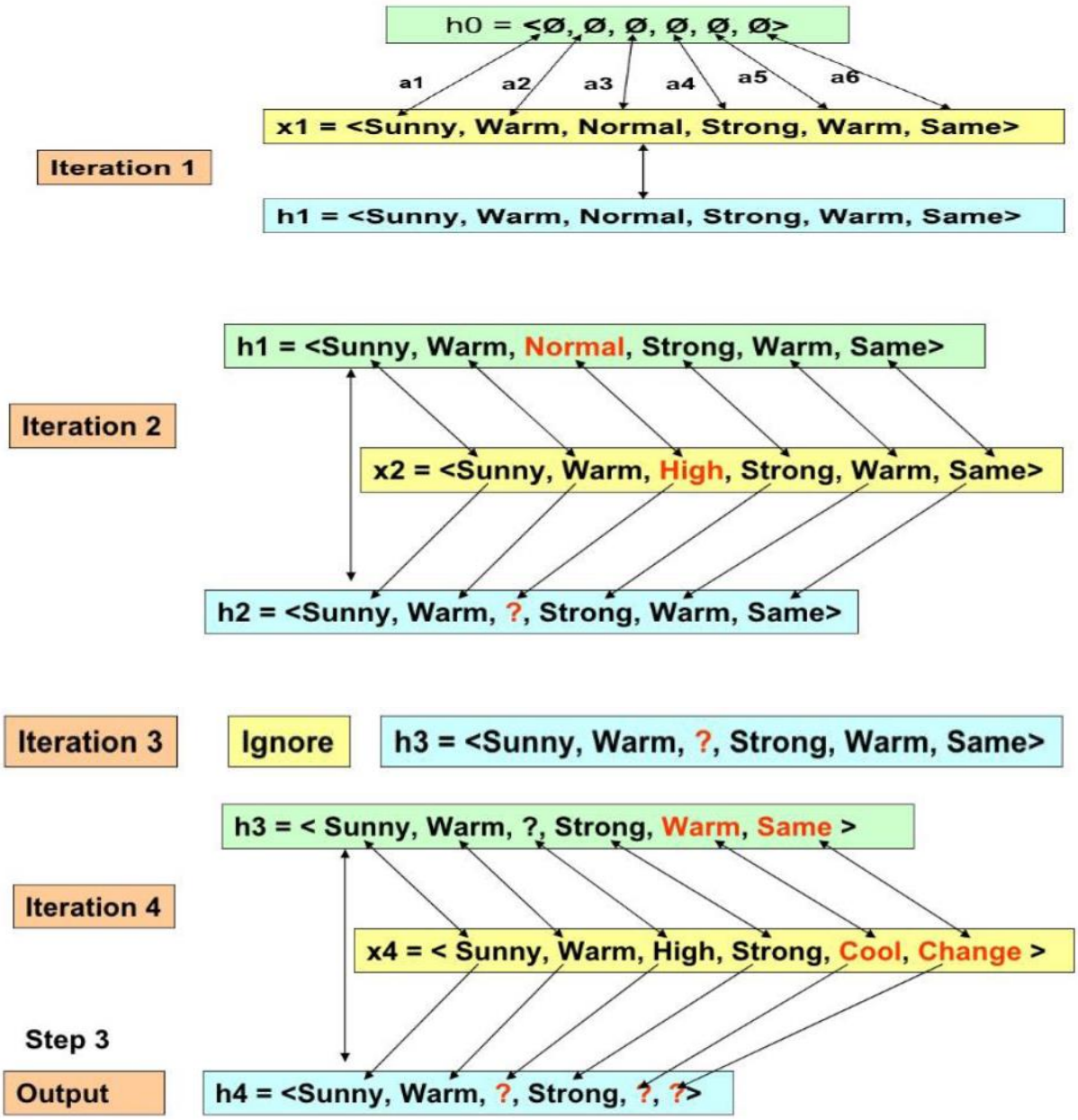
For each positive training instance x

For each attribute constraint a_i in h

If the constraint a_i is satisfied by x

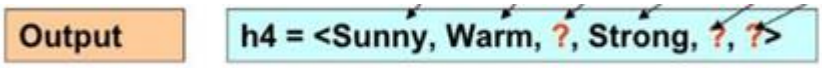
Then do nothing

Else replace a_i in h by the next more general constraint that is satisfied by x.



Step 3:

Output the hypothesis h



d) Program

```
import random
import csv
attributes = [['Sunny','Rainy'],
              ['Warm','Cold'],
              ['Normal','High'],
              ['Strong','Weak'],
              ['Warm','Cool'],
              ['Same','Change']]
num_attributes = len(attributes)
print (" \n The most general hypothesis : ['?','?','?','?','?','?']\n")
print ("\n The most specific hypothesis : ['0','0','0','0','0','0']\n")
a = []
print("\n The Given Training Data Set \n")
```

```
with open('C:\\Users\\rakshith\\Machine Learning Lab\\ws.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)
print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)
# Comparing with First Training Example
for j in range(0,num_attributes):
        hypothesis[j] = a[0][j];
# Comparing with Remaining Training Examples of Given Data Set
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
    if a[i][num_attributes]=='Yes':
            for j in range(0,num_attributes):
                if a[i][j]!=hypothesis[j]:
                    hypothesis[j]='?'
                else :
                    hypothesis[j]= a[i][j]
    print(" For Training Example No :{0} the hypothesis is ".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

e) Input

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

f) Output

```
The most general hypothesis : ['?','?','?','?','?','?']
The most specific hypothesis : ['0','0','0','0','0','0']
The Given Training Data Set
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'Yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'No']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'Yes']
 The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
 Find S: Finding a Maximally Specific Hypothesis
 For Training Example No :0 the hypothesis is  ['sunny', 'warm', 'normal',
'strong', 'warm', 'same']
 For Training Example No :1 the hypothesis is  ['sunny', 'warm', '?', 'str
ong', 'warm', 'same']
 For Training Example No :2 the hypothesis is  ['sunny', 'warm', '?', 'str
ong', 'warm', 'same']
 For Training Example No :3 the hypothesis is  ['sunny', 'warm', '?', 'str
ong', '?', '?']
 The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', '?', 'strong', '?', '?']
```

EXPERIMENT 2

a) Problem Statement: 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.

b) Description

Candidate-Elimination algorithm is used to find all the set of hypothesis consistent with the given data sample. It uses version spaces & considers both positive & negative results. Consider the dataset same as that of the Find-S algorithm for implementation.

c) Algorithm: Candidate-Elimination algorithm is discussed below:

G ← maximally general hypotheses in H
S ← maximally specific hypotheses in H
For each training example d=<x,c(x)>
Case 1 : If d is a positive example
Remove from G any hypothesis that is inconsistent with d
For each hypothesis s in S that is not consistent with d
- *Remove s from S.*
- *Add to S all minimal generalizations h of s such that*
  - *h consistent with d*
  - *Some member of G is more general than h*
- *Remove from S any hypothesis that is more general than another hypothesis in S*

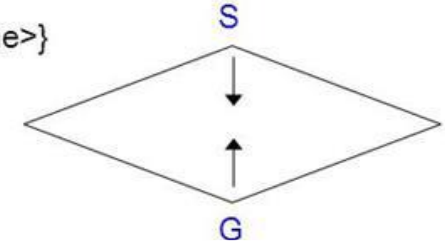
Case 2: If d is a negative example
Remove from S any hypothesis that is inconsistent with d
For each hypothesis g in G that is not consistent with d
- *Remove g from G.*
- *Add to G all minimal specializations h of g such that*
  - *h consistent with d*
  - *Some member of S is more specific than h*
- *Remove from G any hypothesis that is less general than another hypothesis in G*

Illustration

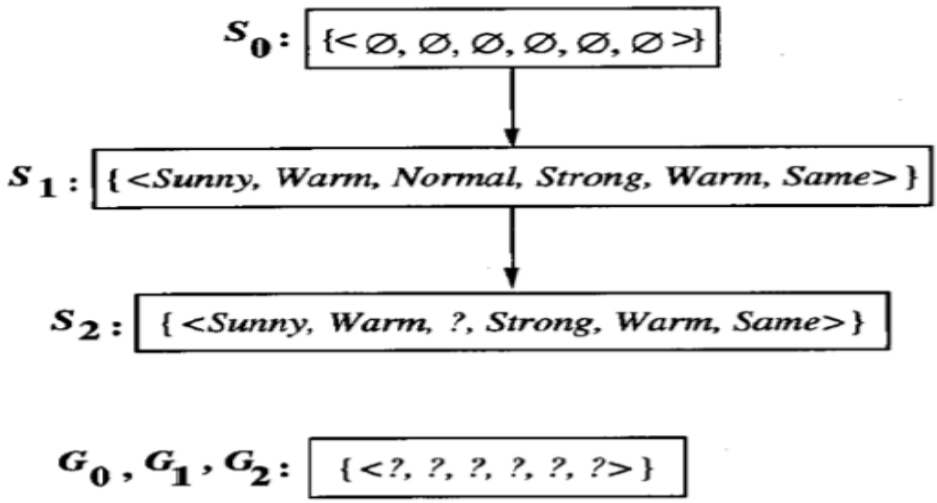
S_0 = {<∅, ∅, ∅, ∅, ∅, ∅>}
G_0 = {<?, ?, ?, ?, ?, ?>}

S_1 = {<Sunny, Warm, Normal, Strong, Warm, Same>}
G_1 = {<?, ?, ?, ?, ?, ?>}

S_2 = {<Sunny, Warm, ?, Strong, Warm, Same>}
G_2 = {<?, ?, ?, ?, ?, ?>}



Trace 1



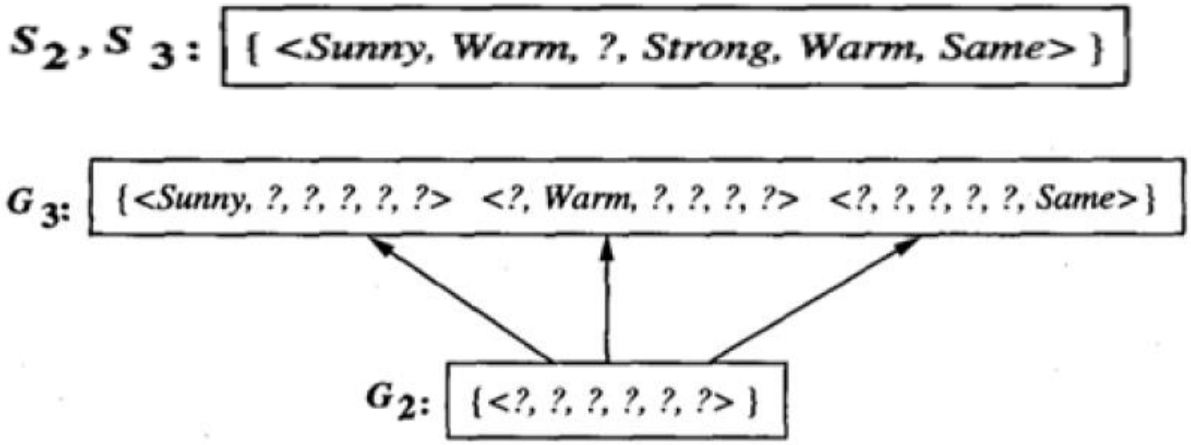
Training examples:

1. *<Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes*

2. *<Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes*

CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.

Trace 2

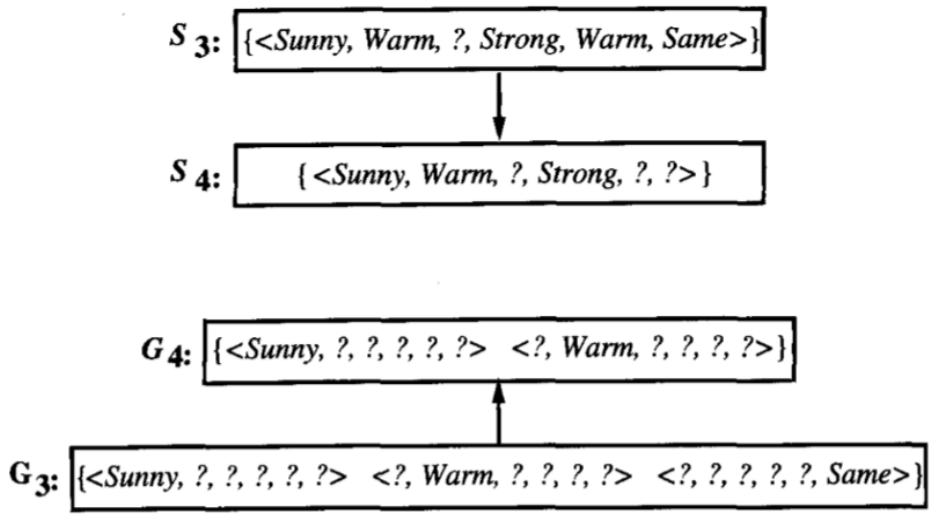


Training Example:

3. *<Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No*

CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.

Trace 3

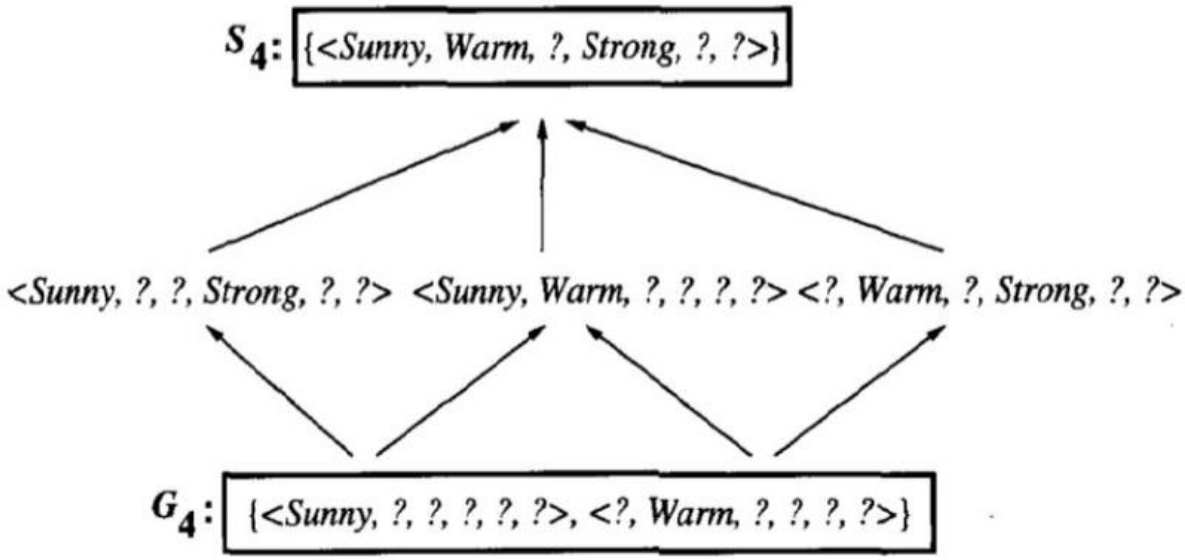


Training Example:

4. <*Sunny, Warm, High, Strong, Cool, Change*>, *EnjoySport = Yes*

CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S boundary, from S_3 to S_4. One member of G_3 must also be deleted, because it is no longer more general than the S_4 boundary.

Final Version Space



d) Program

```
import random
import csv
def g_0(n):
    return ("?",)*n
def s_0(n):
    return ('0',)*n
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)
l1 = [1, 2, 3]
l2 = [3, 4, 5]
list(zip(l1, l2))
```

```
Out[1]: [(1, 3), (2, 4), (3, 5)]
```

```
# min_generalizations
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)]
min_generalizations(h=('0', '0'  , 'sunny'),
                    x=('rainy', 'windy', 'cloudy'))
```

```
Out[2]: [('rainy', 'windy', '?')]
```

```
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
min_specializations(h=('?', 'x',),
                    domains=[['a', 'b', 'c'], ['x', 'y']],
                    x=('b', 'x'))
```

```
Out[3]: [('a', 'x'), ('c', 'x'), ('?', '0')]
```

```
with open('C:\\Users\\rakshith\\Machine Learning Lab\\wsce.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]
examples
```

```
Out[9]: [('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Y'),
         ('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Y'),
         ('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'N'),
         ('Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Y')]
```

```
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]
```

```
get_domains(examples)
```

```
Out[10]: [['Rainy', 'Sunny'],
          ['Cold', 'Warm'],
          ['High', 'Normal'],
          ['Strong'],
          ['Cool', 'Warm'],
          ['Change', 'Same'],
          ['N', 'Y']]
```

```
def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]
    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])
    i=0
    print("\n G[{0}]:".format(i),G)
    print("\n S[{0}]:".format(i),S)
    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("\n G[{0}]:".format(i),G)
        print("\n S[{0}]:".format(i),S)
    return
```

```
def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        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])])
            ## 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
```

```
candidate_elimination(examples)
```

e) Input

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

f) Output

```
G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('0', '0', '0', '0', '0', '0')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[3]: {('Sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('?', 'Warm', '?', '?', '?', '?')}

S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
```

EXPERIMENT 3

a) Problem Statement: Write a program to demonstrate the working of decision tree based ID3 algorithm. Use an appropriate dataset to build the decision tree & apply this knowledge to test the new sample.

b) Description: Iterative Dichotomiser (ID3) Heuristic algorithm is used for building the decision tree. It considers the original set of attributes as the root node. On each iteration of the algorithm, we iterate through every unused attribute of the remaining set and calculates the entropy (or information gain) of that attribute. Then, we select the attribute which has the smallest entropy (or largest information gain) value. The set of remaining attributes is then split by the selected attribute to produce subsets of the data. The algorithm continues to recurse on each subset, considering only attributes never selected before. In testing phase at runtime, we will use trained decision tree to classify the new unseen test cases by working down the decision tree using the values of this test case to arrive at a terminal node that tells us what class this test case belongs to.

c) Algorithm: Iterative Dichotomiser (ID3) Heuristic algorithm is discussed below:

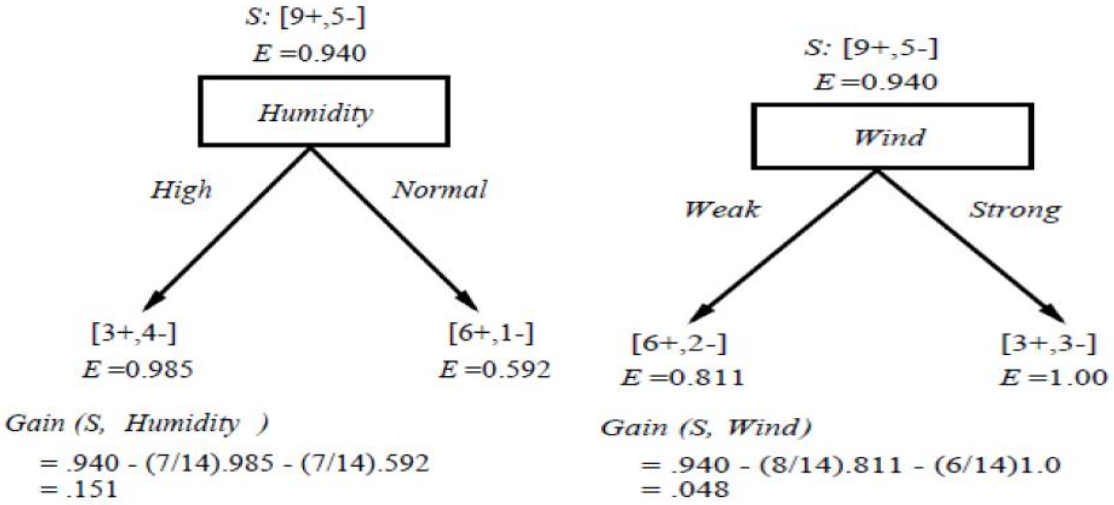
ID3 - Algorithm

ID3*(Examples, TargetAttribute, Attributes)*

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = -
- If *Attributes* is empty, Return the single-node tree Root, with label = most common value of *TargetAttribute* in *Examples*
- Otherwise Begin
  - 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*

Illustration

Step 1: Compute the Gain and identify which attribute is the best classifier as illustrated below



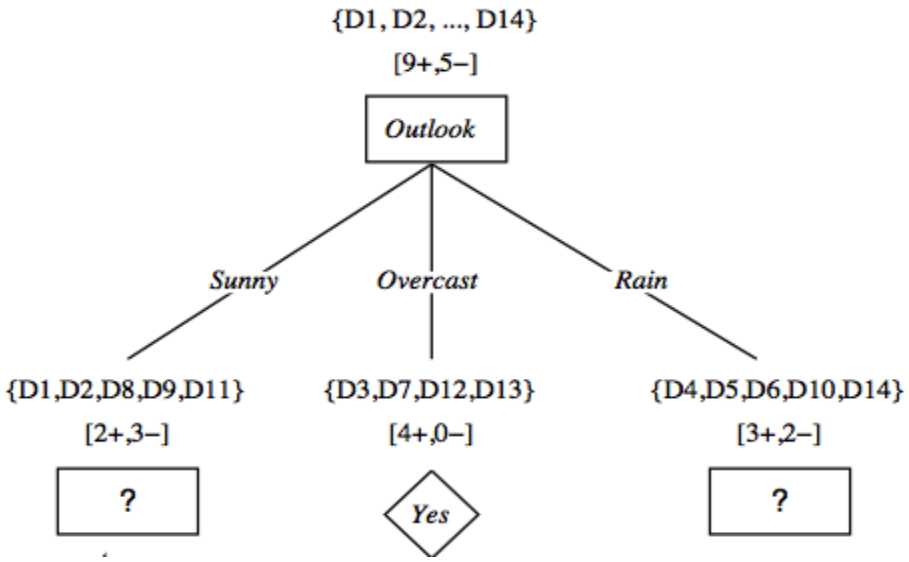
Which attribute to test at the root?

- *Gain*(*S*, *Outlook*) = 0.246
- *Gain*(*S*, *Humidity*) = 0.151
- *Gain*(*S*, *Wind*) = 0.048
- *Gain*(*S*, *Temperature*) = 0.029

Outlook provides the best prediction for the target
Let's grow the tree:

- add to the tree a successor for each possible value of **Outlook**
- partition the training samples according to the value of **Outlook**.

Step 2



d) 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 retrieved
    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)
```

e) Input

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	**No**
D2	Sunny	Hot	High	Strong	**No**
D3	Overcast	Hot	High	Weak	**Yes**
D4	Rain	Mild	High	Weak	**Yes**
D5	Rain	Cool	Normal	Weak	**Yes**
D6	Rain	Cool	Normal	Strong	**No**
D7	Overcast	Cool	Normal	Strong	**Yes**
D8	Sunny	Mild	High	Weak	**No**
D9	Sunny	Cool	Normal	Weak	**Yes**
D10	Rain	Mild	Normal	Weak	**Yes**
D11	Sunny	Mild	Normal	Strong	**Yes**
D12	Overcast	Mild	High	Strong	**Yes**
D13	Overcast	Hot	Normal	Weak	**Yes**
D14	Rain	Mild	High	Strong	**No**

f) Output

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

EXPERIMENT 4

a) Problem Statement: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

b) Description: The BACKPROPAGATION algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs. The learning problem faced by BACKPROPAGATION algorithm is to search a large hypothesis space defined by all possible weight values for all the units in the network.

c) Algorithm: The BACKPROPAGATION algorithm is discussed below:

function BackProp (D, η, n_{in}, n_{hidden}, n_{out})

- D is the training set consists of m pairs: $\{(x_i, y_i)^m\}$
- η is the learning rate as an example (0.1)
- n_{in}, n_{hidden} e n_{out} are the numbero of imput hidden and output unit of neural network

Make a feed-forward network with n_{in}, n_{hidden} e n_{out} units
Initialize all the weight to short randomly number (es. [-0.05 0.05])
Repeat until termination condition are verifyed:
For any sample in D:

Forward propagate the network computing the output o_u of every unit u of the network

Back propagate the errors onto the network:

– For every output unit k, compute the error δ_k: $\delta_k = o_k(1-o_k)(t_k-o_k)$

– For every hidden unit h compute the error δ_h: $\delta_h = o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$

– Update the network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}, \quad where \ \Delta w_{ji} = \eta \delta_j x_{ji}$

(x_{ji} is the input of unit j from coming from unit i)

The Back propagation Algorithm for a feed-forward 2-layer network of sigmoid units, the stochastic version

Idea: Gradient descent over the entire vector of network weights.

Initialize all weights to small random numbers.
Until satisfied, // *stopping criterion* to be (later) defined
for each training example,

1. input the training example to the network, and compute the network outputs
2. for each output unit k:
   $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$

3. for each hidden unit h:
   $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
4. update each network weight w_{ji}:
   $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where $\Delta w_{ji} = \eta \delta_j x_{ji}$,
   and x_{ji} is the ith input to unit j.

d) Program

```
import numpy as np
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 leaning 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
```

```
#Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons))
bh=np.random.uniform(size=(1,hidden_neurons))
wout=np.random.uniform(size=(hidden_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

```
for i in range(epoch):
#forward propogation
    h_ip=np.dot(X,wh)+bh
    h_act=sigmoid(h_ip)
    o_ip=np.dot(h_act,wout)
    output =sigmoid(o_ip)
    #Back propogation
    #Error at Outuput layer
    Eo = y-output
    outgrad=sigmoid_grad(output)
    d_output=Eo*outgrad
    #Error at Hidden layer
    Eh=d_output.dot(wout.T)
    hiddengrad=sigmoid_grad(h_act)
    d_hidden=Eh*hiddengrad
    wout += h_act.T.dot(d_output)*eta
    wh += X.T.dot(d_hidden)*eta
print("Normalized Input: \n"+ str(X))
print("Actual Output: \n"+ str(y))
print("Predicted Output : \n", output)
```

e) Input

Hours Slept	Hours Studied	Marks Obtained
2	9	92
1	5	86
3	6	89

f) Output

```
Normalized Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.8969548 ]
 [0.87694914]
 [0.89512028]]
```

EXPERIMENT 5

a) Problem Statement: 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.

b) Description: Naive Bayes classification algorithm can be extremely fast relative to other classification algorithms. It works on Bayes theorem of probability to predict the class of unknown data set. 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.

c) Algorithm/Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h|D)$ = probability of h given D
- $P(D|h)$ = probability of D given h

For the Bayesian Rule above, we have to extend it so that we have

$$P(C|X_1, X_2, \ldots, X_n) = \frac{P(X_1, X_2, \ldots, X_n|C)\,P(C)}{P(X_1, X_2, \ldots, X_n)}$$

c) Program

```
import csv, random, math
import statistics as st
from statistics import stdev
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 #instances 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 mean(numbers):
    return sum(numbers)/float(len(numbers))
def stdev(numbers):
    avg=mean(numbers)
    variance=sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)
```

```
def compute_mean_std(dataset):
    mean_std = [(st.mean(attribute), st.stdev(attribute))for attribute in zip(*dataset)];
    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('pima-indians-diabetes.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)
```

d) Input

Pima-indians-diabetes.csv

e) Output

```
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 : 72.54901960784314
```

EXPERIMENT 6

a) Problem Statement: Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

b) Description: 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 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).

c) Algorithm

LEARN_NAIVE_BAYES_TEXT($Examples$, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - $Vocabulary \leftarrow$ the set of all distinct words and other tokens occurring in any text document from $Examples$
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*
   - For each target value v_j in V do
     - $docs_j \leftarrow$ the subset of documents from $Examples$ for which the target value is v_j
     - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
     - $n \leftarrow$ total number of distinct word positions in $Text_j$
     - for each word w_k in $Vocabulary$
       - $n_k \leftarrow$ number of times word w_k occurs in $Text_j$
       - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT(Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the ith position within Doc.

- $positions \leftarrow$ all word positions in Doc that contain tokens found in $Vocabulary$
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

d) Program

```
import pandas as pd
msg=pd.read_csv('data6.csv',names=['message','label']) #Tabular form data
print('Total instances in the dataset:',msg.shape[0])
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
Y=msg.labelnum
print('\nThe message and its label of first 5 instances are listed below')
X5, Y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
    print(x,',',y)
```

```
Total instances in the dataset: 18

The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos
I feel very good about these beers , pos
This is my best work , pos
What an awesome view , pos
```

```
# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)
print('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing instances :', xtest.shape[0])
```

```
Dataset is split into Training and Testing samples
Total training instances : 13
Total testing instances : 5
```

```
#Output of count vectoriser is a sparse matrix
#CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix xtest_dtm = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])
print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
```

```
Total features extracted using CountVectorizer: 42

Features for first 5 training instances are listed below
   am  amazing  an  awesome  best  boss  can  dance  deal  do  ...  this  \
0   0        0   0        0     0     0    0      0     0   1  ...     1
1   0        0   0        0     0     1    0      0     0   0  ...     0
2   0        0   0        0     0     0    0      0     0   1  ...     1
3   0        0   0        0     0     0    0      0     0   0  ...     1
4   0        0   0        0     0     0    1      0     1   0  ...     1

   tired  to  tomorrow  view  we  what  will  with  work
0      0   0         0     0   0     0     0     0     0
1      0   0         0     0   0     0     0     0     0
2      0   0         0     0   0     0     0     0     0
3      0   0         0     0   0     0     0     0     0
4      0   0         0     0   0     0     0     1     0

[5 rows x 42 columns]
```

```
#print(xtrain_dtm) #Same as above but sparse matrix representation
#Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
print('\nClassstification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
    pred = 'pos' if p==1 else 'neg'
    print('%s -> %s ' % (doc, pred))
```

```
Classstification results of testing samples are given below
I am tired of this stuff -> pos
My boss is horrible -> neg
I do not like the taste of this juice -> neg
I went to my enemy's house today -> pos
We will have good fun tomorrow -> pos
```

```
#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Recall :',metrics.recall_score(ytest,predicted),
'\nPrecison :',metrics.precision_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

e) Input

data6.csv (This consists of sentiment analysis data)

f) Output

```
Accuracy metrics
Accuracy of the classifer is 0.6
Recall : 1.0
Precison : 0.3333333333333333
Confusion matrix
[[2 2]
 [0 1]]
```

EXPERIMENT 7

a) Problem Statement: Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

b) Problem Description: A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities. In contrast to the naive Bayes classifier, which assumes that ***all*** the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to ***subsets*** of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether.

c) Algorithm

Bayesian Belief networks describe conditional independence among *subsets* of variables → allows combining prior knowledge about (in) dependencies among variables with observed training data (also called Bayes Nets).

- **Definition: *X* is *conditionally independent* of *Y* given *Z* if the probability distribution governing *X* is independent of the value of *Y* given the value of *Z*; that is, if**
  $$(\forall x_i, y_j, z_k)\ P(X= x_i | Y= y_j, Z= z_k) = P(X= x_i | Z= z_k)$$
  more compactly, we write
  $$P(X | Y, Z) = P(X | Z)$$
- **Example: *Thunder* is conditionally independent of *Rain*, given *Lightning***
  $$P(Thunder | Rain, Lightning) = P(Thunder | Lightning)$$
- **Naive Bayes uses cond. indep. to justify**
  $$P(X, Y | Z) = P(X | Y, Z)\ P(Y | Z) = P(X | Z)\ P(Y | Z)$$

d) Program

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

```
#Read the attributes
lines=list(csv.reader(open('data7_names.csv','r')));
attributes=lines[0]
```

```
#Read Cleveland Heart Disease data
heartDisease = pd.read_csv('data7.csv',names=attributes)
#heartDisease = heartDisease.replace('?',np.nan)
#Display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())
```

```
#Model Baysian Network
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'),('sex', 'trestbps'), ('exang', 'trestbps'),('trestbps','heartdisease'),
('fbs','heartdisease'), ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])
```

```
#Learing CPDs using Maximum Likelihood Estimators
print('\nLearing CPDs using Maximum Likelihood Estimators....')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
#Doing exact inference using Variable Elimination
print('\n Inferencing with Bayesian Network :')
HeartDisease_infer = VariableElimination(model)
```

```
#Computing the probability of bronc given smoke.
print('\n1.Probability of HeartDisease given Age=28')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
print('\n2.Probability of HeartDisease given chol(Cholestrol)=100')
q= HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

e) Input

data7.csv (This file consists of Heart disease data set)

f) Output

1.Probability of HeartDisease given Age=28

heartdisease	phi(heartdisease)
heartdisease_0	0.5936
heartdisease_1	0.1661
heartdisease_2	0.0904
heartdisease_3	0.1103
heartdisease_4	0.0395

2.Probability of HeartDisease given chol(Cholestrol)=100

heartdisease	phi(heartdisease)
heartdisease_0	1.0000
heartdisease_1	0.0000
heartdisease_2	0.0000
heartdisease_3	0.0000
heartdisease_4	0.0000

EXPERIMENT 8

a) Problem Statement: 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.

b) Description: EM algorithm is a widely used approach to learning in the presence of unobserved variables. The EM algorithm can be used even for variables whose value is never directly observed, provided the general form of the probability distribution governing these variables is known. The EM algorithm has been used to train Bayesian belief networks as well as radial basis function networks. The EM algorithm is also the basis for many unsupervised clustering algorithms and it is the basis for the widely used Baum-Welch forward-backward algorithm for learning Partially Observable Markov Models.

c) Algorithm: The EM algorithm first initializes the hypothesis to h = $\langle \mu_1, \mu_2 \rangle$, where μ_1 and μ_2 are arbitrary initial values. It then iteratively re-estimates h by repeating the following two steps until the procedure converges to a stationary value for h.

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu_1', \mu_2' \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu_1', \mu_2' \rangle$ and iterate.

K Means Algorithm

Step 1: The sample space is initially partitioned into K clusters and the observations are randomly assigned to the clusters.

Step 2: For each sample:

1) Calculate the distance from the observation to the centroid of the cluster.

2) IF the sample is closest to its own cluster THEN leave it. ELSE select another cluster.

Step 3: Repeat steps 1 and 2 until no observations are moved from one cluster to another.

d) Program

```
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')
```

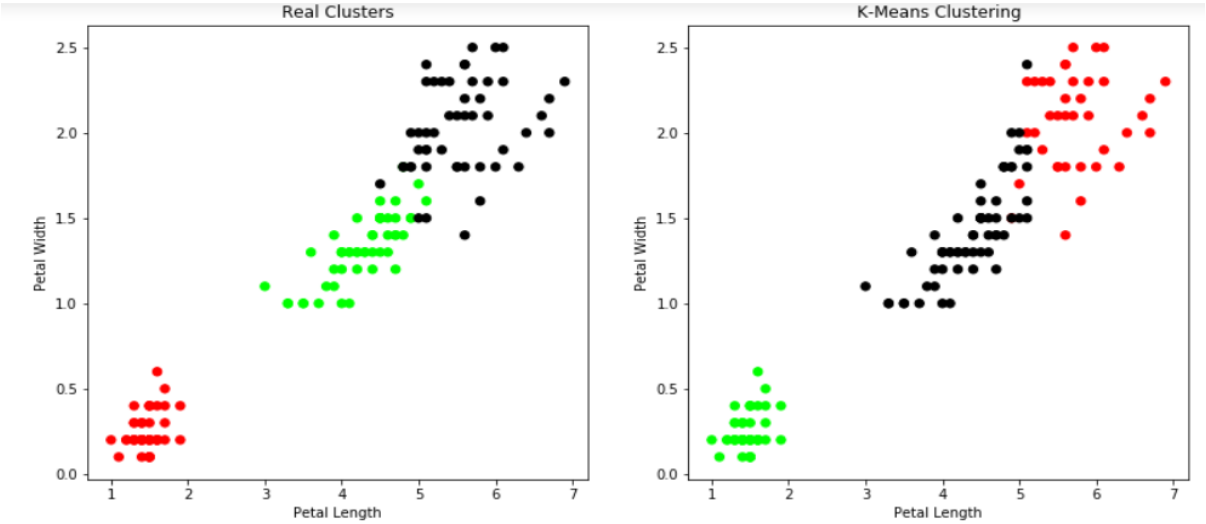
```
# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

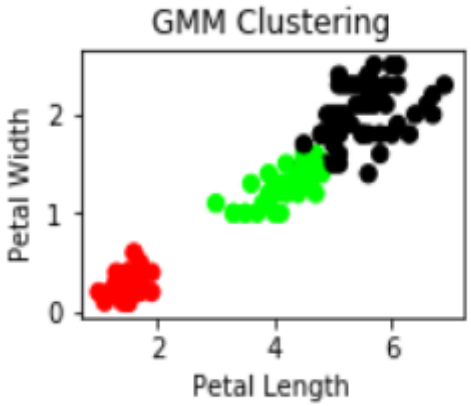
```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

e) Input

Iris data set (Imported from sklearn library)

f) Output:





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

The above figure illustrates the comparison between K-Means clustering & GMM clustering. There are three clusters each corresponds to the three species namely setosa, versicolour & virginica. 40 data points has been considered and classified into different clusters based on the petal length and petal width features. The clusters are indicated by using 3 different colors namely red, lime & black.

EXPERIMENT 9

a) Problem Statement: 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.

b) Problem Description: The most basic instance-based method is the k-NEAREST NEIGHBOR algorithm. This algorithm assumes all instances correspond to points in the n-dimensional space R^n. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. In nearest-neighbor learning the target function may be either discrete-valued or real-valued.Let us first consider learning discrete-valued target functions of the form f: $R^n \rightarrow V$, where V is the finite set {vl, . . . vs}.

c) Algorithm: The k-NEAREST NEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

d) Program

```
# 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))
```

e) Input

Iris data set (Imported from sklearn library)

f) Output

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: [7.2 3. 5.8 1.6]	Actual-label: 2	Predicted-label: 2
Sample: [5.8 2.7 3.9 1.2]	Actual-label: 1	Predicted-label: 1
Sample: [5.8 2.7 4.1 1.0]	Actual-label: 1	Predicted-label: 1
Sample: [6.3 2.3 4.4 1.3]	Actual-label: 1	Predicted-label: 1
Sample: [5.1 3.4 1.5 0.2]	Actual-label: 0	Predicted-label: 0
Sample: [5.1 3.8 1.6 0.2]	Actual-label: 0	Predicted-label: 0
Sample: [5.8 2.6 4. 1.2]	Actual-label: 1	Predicted-label: 1
Sample: [6.7 2.5 5.8 1.8]	Actual-label: 2	Predicted-label: 2
Sample: [5.7 3.8 1.7 0.3]	Actual-label: 0	Predicted-label: 0
Sample: [5.9 3. 5.1 1.8]	Actual-label: 2	Predicted-label: 2
Sample: [4.8 3. 1.4 0.3]	Actual-label: 0	Predicted-label: 0
Sample: [5.5 2.3 4. 1.3]	Actual-label: 1	Predicted-label: 1
Sample: [6.8 2.8 4.8 1.4]	Actual-label: 1	Predicted-label: 1
Sample: [6.3 2.5 5. 1.9]	Actual-label: 2	Predicted-label: 2
Sample: [7.1 3. 5.9 2.1]	Actual-label: 2	Predicted-label: 2

Classification Accuracy: 1.0

EXPERIMENT 10

a) Problem Statement: Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

b) Description: Locally weighted regression is a generalization of k-nearest neighbour approach. It constructs an explicit approximation to f over a local region surrounding x_q. Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f. For example, we might approximate the target function in the neighborhood surrounding x_q. Using a linear function, a quadratic function, a multilayer neural network, or some other functional form. The phrase "locally weighted regression" is called ***local*** because the function is approximated based only on data near the query point, ***weighted*** because the contribution of each training example is weighted by its distance from the query point, and ***regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

c) Algorithm

1. Consider the case of locally weighted regression in which the target function f is approximated near ***xq,*** using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

Where $a_i(x)$ denotes the value of the i^{th} attribute of the instance ***x.***

2. Choose weights that minimize the squared error summed over the set D of training examples

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

The above equation leads to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

3. Redefine the error criterion ***E*** to emphasize fitting the local training examples. The error ***E(xq)*** is used to emphasize the fact that now the error is being defined as a function of the query point ***xq.*** The following 3 criteria are specified below.

1. Minimize the squared error over just the k nearest neighbours:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from ***xq.***

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \; k \; nearest \; nbrs \; of \; x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

d) 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
```

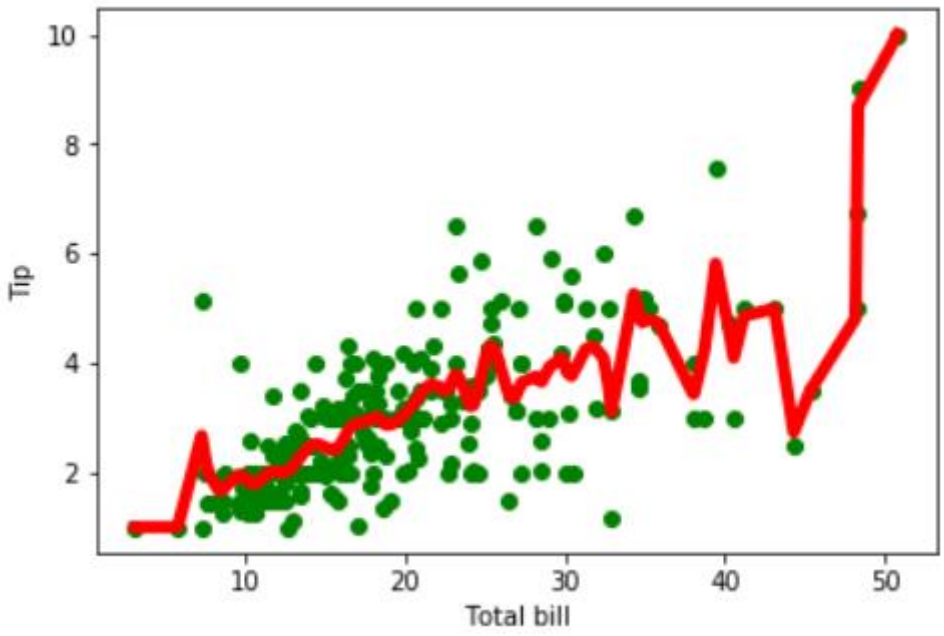
```
def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total 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
ypred = localWeightRegression(X,mtip,0.5) # increase k to get smooth curves
graphPlot(X,ypred)
```

e) Input

data10_tips.csv (bill amount & tips data)

f) Output



Viva Questions & Answers

1. **What is Machine learning? Give Example**

   Machine learning is a branch of computer science which deals with system programming in order to automatically learn and improve with experience. For example: Robots are programmed so that they can perform the task based on data they gather from sensors.

2. **Mention the difference between Data Mining and Machine learning?**

   Machine learning relates with the study, design and development of the algorithms that give computers the capability to learn without being explicitly programmed.

   Data mining can be defined as the process in which the unstructured data tries to extract knowledge or unknown interesting patterns. During this process machine learning algorithms are used.

3. **What is 'Overfitting' in Machine learning?**

   In machine learning, when a statistical model describes random error or noise instead of underlying relationship 'overfitting' occurs.

4. **Why overfitting happens?**

   The possibility of overfitting exists as the criteria used for training the model is not the same as the criteria used to judge the efficacy of a model.

5. **How can you avoid overfitting?**

   By using a lot of data overfitting can be avoided, overfitting happens relatively as you have a small dataset, and you try to learn from it.

6. **What is inductive machine learning?**

   The inductive machine learning involves the process of learning by examples, where a system, from a set of observed instances tries to induce a general rule.

7. **What are the five popular algorithms of Machine Learning?**

   a) Decision Trees

   b) Neural Networks (back propagation)

   c) Probabilistic networks

   d) Nearest Neighbour

   e) Support vector machines

8. **What are the different Algorithm techniques in Machine Learning?**

The different types of techniques in Machine Learning are

   a) Supervised Learning

   b) Unsupervised Learning

   c) Semi-supervised Learning

   d) Reinforcement Learning

9. What are the three stages to build the hypotheses or model in machine learning?

a) Model building

b) Model testing

c) Applying the model

10. What is the standard approach to supervised learning?

The standard approach to supervised learning is to split the set of example into the training set and the test.

11. What is 'Training set' and 'Test set'?

Training set is an example given to the learner, while Test set is used to test the accuracy of the hypotheses generated by the learner, and it is the set of example held back from the learner. Training set is distinct from Test set.

12. List down various approaches for machine learning?

a) Concept Vs Classification Learning

b) Symbolic Vs Statistical Learning

c) Inductive Vs Analytical Learning

13. List functions of 'Unsupervised Learning'?

a) Find clusters of the data

b) Find low-dimensional representations of the data

c) Find interesting directions of data

d) Find novel observations/database cleaning

14. List any two functions of 'Unsupervised Learning'?

a) Classifications

b) Speech recognition

c) Regression

d) Predict time series

15. What is algorithm independent machine learning?

Machine learning in where mathematical foundations are independent of any particular classifier or learning algorithm is referred as algorithm independent machine learning.

16. What is the difference between artificial learning and machine learning?

Designing and developing algorithms according to the behaviours based on empirical data are known as Machine Learning. While artificial intelligence in addition to machine learning, it also covers other aspects like knowledge representation, natural language processing, planning, robotics etc.

17. What is classifier in machine learning?

A classifier in a Machine Learning is a system that inputs a vector of discrete or continuous feature values and outputs a single discrete value, the class.

18. What are the advantages of Naive Bayes?

In Naïve Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data. The main advantage is that it can't learn interactions between features.

19. In what areas Pattern Recognition is used?

a) Computer Vision

b) Speech Recognition

c) Data Mining

d) Statistics

20. What is Inductive Logic Programming in Machine Learning?

Inductive Logic Programming (ILP) is a subfield of machine learning which uses logical programming representing background knowledge and examples.

21. What are the two methods used for the calibration in Supervised Learning?

The two methods used for predicting good probabilities in Supervised Learning are

a) Platt Calibration

b) Isotonic Regression

22. Which method is frequently used to prevent overfitting?

When there is sufficient data 'Isotonic Regression' is used to prevent an overfitting issue.

23. What is Perceptron in Machine Learning?

In Machine Learning, Perceptron is an algorithm for supervised classification of the input into one of several possible non-binary outputs.

24. What are Bayesian Networks (BN)?

Bayesian Network is used to represent the graphical model for probability relationship among a set of variables.

25. Why instance based learning algorithm sometimes referred as Lazy learning algorithm?

Instance based learning algorithm is also referred as Lazy learning algorithm as they delay the induction or generalization process until classification is performed.

26. What are the two classification methods that SVM (Support Vector Machine) can handle?

a) Combining binary classifiers

b) Modifying binary to incorporate multiclass learning

27. What is ensemble learning?

To solve a particular computational program, multiple models such as classifiers or experts are strategically generated and combined. This process is known as ensemble learning.

28. Why ensemble learning is used?

Ensemble learning is used to improve the classification, prediction, function approximation etc of a model.

29. When to use ensemble learning?

Ensemble learning is used when you build component classifiers that are more accurate and independent from each other.

30. What are the two paradigms of ensemble methods?

The two paradigms of ensemble methods are

a) Sequential ensemble methods

b) Parallel ensemble methods

31. What is the general principle of an ensemble method?

The general principle of an ensemble method is to combine the predictions of several models built with a given learning algorithm in order to improve robustness over a single model.

32. What is bagging and boosting in ensemble method?

Bagging is a method in ensemble for improving unstable estimation or classification schemes. While boosting method are used sequentially to reduce the bias of the combined model. Boosting and Bagging both can reduce errors by reducing the variance term.

33. What is PCA, KPCA and ICA used for?

PCA (Principal Components Analysis), KPCA (Kernel based Principal Component Analysis) and ICA (Independent Component Analysis) are important feature extraction techniques used for dimensionality reduction.

34. What is dimension reduction in Machine Learning?

In Machine Learning and statistics, dimension reduction is the process of reducing the number of random variables under considerations and can be divided into feature selection and feature extraction.

35. What are support vector machines?

Support vector machines are supervised learning algorithms used for classification and regression analysis.

36. What are the different methods for Sequential Supervised Learning?

The different methods to solve Sequential Supervised Learning problems are

a) Sliding-window methods

b) Recurrent sliding windows

c) Hidden Markov models

d) Maximum entropy Markov models

e) Conditional random fields

f) Graph transformer networks

37. What is batch statistical learning?

Statistical learning techniques allow learning a function or predictor from a set of observed data that can make predictions about unseen or future data.

38. What is PAC Learning?

PAC (Probably Approximately Correct) learning is a learning framework that has been introduced to analyze learning algorithms and their statistical efficiency.

39. What are the different categories you can categorized the sequence learning process?

a) Sequence prediction

b) Sequence generation

c) Sequence recognition

d) Sequential decision

40. Give a popular application of machine learning that you see on day to day basis?

The recommendation engine implemented by major ecommerce websites uses Machine Learning.