**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**BELAGAVI**



**VII Semester**

**LAB MANUAL**

***“Machine Learning Laboratory”***

**(Sub. Code: 15CSL76)**

****

Prepared By

|  |  |  |
| --- | --- | --- |
| **Dr. Naveena C** | **Mrs. Roopa M J** | **Mrs. Chaitra M** |
| Professor, | Associate Professor, | Assistant Professor, |
| **Dr. Ajay Prakash B.V** | **Mr. Ajay B.N** | **Chandrashekar D K** |
| Associate Professor, | Assistant Professor, | Assistant Professor, |

**Department of COMPUTER science and engineering**

**SJB Institute of Technology**

**BGS HEALTH AND EDUCATION CITY KENGERI,**

**BANGALORE-560060**

**(Affiliated to Visvesvaraya Technological University, Belagavi &**

**Approved by AICTE, New Delhi)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DESCRIPTION** | |  |  |  |  |  |  |  |  |
| 1. PREREQUISITES: | |  |  |  |  |  |  |  |  |
| • | Creative thinking, sound mathematical insight and programming skills. | | | |  |  |  |  |  |
| • | Design and Analysis of Algorithms (15CS43) | | |  |  |  |  |  |  |
| • | Design and Analysis of Algorithms Laboratory (15CSL47) | | | |  |  |  |  |  |
| • | Fundamentals of Data Structures (15CS33) | | |  |  |  |  |  |  |
| • | Data Structures Laboratary (15CSL37) | | |  |  |  |  |  |  |
| • | Computer Programming Laboratory (15CPL16/26) | | |  |  |  |  |  |  |
| 2. BASE COURSE: | |  |  |  |  |  |  |  |  |
| • | Machine Learning (15CS73) | |  |  |  |  |  |  |  |
| 3. COURSE OUTCOMES: | | |  |  |  |  |  |  |  |
| At the end of the course, the student will be able to; | | | |  |  |  |  |  |  |
| 1. | Understand the implementation procedures for the machine learning algorithms. | | | | | | | |  |
| 2. | Design 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. | | | | | |  |  |  |
| 4. RESOURSES REQUIRED: | | |  |  |  |  |  |  |  |
| • | Hardware resources | |  |  |  |  |  |  |  |
|  | ◦ Desktop PC |  |  |  |  |  |  |  |  |
|  | ◦ Windows / Linux operating system | | |  |  |  |  |  |  |
| • | Software resources | |  |  |  |  |  |  |  |
|  | ◦ Python |  |  |  |  |  |  |  |  |
|  | ◦ Anaconda IDE with Spider | |  |  |  |  |  |  |  |
| • | Datasets from standard repositories (Ex: https://archive.ics.uci.edu/ml/datasets.html) | | | | | | | |  |
|  | | |  | |  |  |  |  |  |

1. RELEVANCE OF THE COURSE:
   * Project work (15CSP78, 15CSP85)
2. GENERAL INSTRUCTIONS:
   * Implement the program in Python editor like Spider and demosnstrate the same.
   * External practical examination.
     + All laboratory experiments are to be included
     + Students are allowed to pick one experiment from the lot.
     + Marks distribution: Procedure + Conduction + Viva: 20 + 50 +10 (80)
     + Change of experiment is allowed only once and marks allotted to the procedure part to be made zero

1. CONTENTS:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Expt | Title of the Experiments | BT | CO |  |
| No. |  |
| 1 | Implement and demonstratethe **FIND-S algorithm** for finding the most |  | CO |  |
|  | specific hypothesis based on a given set of training data samples. Read the | L3 |  |
|  | 1,2,3,4 |  |
|  | training data from a .CSV file. |  |  |
|  |  |  |  |
| 2 | For a given set of training data examples stored in a .CSV file, implement |  | CO |  |
|  | and demonstrate the **Candidate-Elimination algorithm** to output a | L3 |  |
|  | 1,2,3,4 |  |
|  | 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** |  | CO |  |
|  | **algorithm.** Use an appropriate data set for building the decision tree and | L3 |  |
|  | 1,2,3,4 |  |
|  | apply this knowledge toclassify a new sample. |  |  |
|  |  |  |  |
| 4 | Build an Artificial Neural Network by implementing the **Backpropagation** | L3 | CO |  |
|  | **algorithm** and test the same using appropriate data sets. | 1,2,3,4 |  |
|  |  |  |
| 5 | Write a program to implement the **naïve Bayesian classifier** for a sample |  | CO |  |
|  | training data set stored as a .CSV file. Compute the accuracy of the classifier, | L3 |  |
|  | 1,2,3,4 |  |
|  | considering few test data sets. |  |  |
|  |  |  |  |
| 6 | 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 | L3 | CO |  |
|  | can be used to write the program. Calculate the accuracy, precision, and | 1,2,3,4 |  |
|  |  |  |
|  | recall for your data set. |  |  |  |
| 7 | Write a program to construct a **Bayesian network** considering medical data. |  | CO |  |
|  | Use this model to demonstrate the diagnosis of heart patients using standard | L3 |  |
|  | 1,2,3,4 |  |
|  | 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 | L3 | CO |  |
|  | of these two algorithms and comment on the quality of clustering. You can | 1,2,3,4 |  |
|  |  |  |
|  | add Java/Python ML library classes/API in the program. |  |  |  |
| 9 | Write a program to implement **k-Nearest Neighbour algorithm** to classify |  | CO |  |
|  | the iris data set. Print both correct and wrong predictions. Java/Python ML | L3 |  |
|  | 1,2,3,4 |  |
|  | library classes can be used for this problem. |  |  |
|  |  |  |  |
| 10 | Implement the non-parametric **Locally Weighted Regression algorithm** in |  | CO |  |
|  | order to fit data points. Select appropriate data set for your experiment and draw graphs. | L3 |  |
|  | 1,2,3,4 |  |

1. 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, 2nd edition, Springer series in statistics.
   3. Ethem Alpaydın, Introduction to machine learning, second edition, MIT press.

**C. EVALUATION SCHEME**

For CBCS 2015 scheme:

1. Laboratory Components : 12 Marks

(Record writing, Laboratory performance and Viva-voce)

1. Laboratory IA tests: 8 Marks

(Minimum 2 IAs are mandatory. For the final IA test marks, average of the 2 IA test marks shall be considered and converted to maximum of 8)

1. Continuous Internal Evaluation (CIE) = 12+ 8 = 20 Marks
2. *SEE* : 80 Marks

1. EXPERIMENT NO: 1
2. TITLE: **FIND-S 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 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.**
5. THEORY:
   * + The concept learning approach in machine learning, can be formulated as “Problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples”.
     + Find-S algorithm for concept learning is one of the most basic algorithms of machine learning.

Find-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x

For each attribute constraint a i in h :

If the constraint a i in h is satisfied by x then do nothing

Else replace a i in h by the next more general constraint that is satisfied by x

* 1. Output hypothesis h
* It is Guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
* Also Notice that negative examples are ignored.

Limitations of the Find-S algorithm:

* + No way to determine if the only final hypothesis (found by Find-S) is consistent with data or there are more hypothesis that is consistent with data.
  + Inconsistent sets of training data can mislead the finds algorithm as it ignores negative data samples.
  + A good concept learning algorithm should be able to backtrack the choice of hypothesis found so that the resulting hypothesis can be improved over time. Unfortunately, Find-S provide no such method.

1. PROCEDURE / PROGRAMME :

**FindS.py**

import numpy as np

import pandas as pd

data=pd.read\_csv('finds.csv')

print('Data',data)

def train (concepts,target):

specific\_h=concepts[0]

print('specific1',specific\_h)

for i,h in enumerate(concepts):

print('i',i)

print('h',h)

if target[i]=="Yes":

for x in range(len(specific\_h)):

print('x',x)

print('specific',specific\_h)

if h[x]==specific\_h[x]:

pass

else:

specific\_h[x]="?"

return specific\_h

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

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

print('Concept',concepts)

print('Target',target)

print(train(concepts,target))

**Output:**

Data Sky Airtemp Humidity Wind Water Forecast WaterSport

0 Sunny Warm Normal Strong Warm Same Yes

1 Sunny Warm High Strong Warm Same Yes

2 Cloudy Cold High Strong Warm Change No

3 Sunny Warm High Strong Cool Change Yes

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

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

['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

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

Target ['Yes' 'Yes' 'No' 'Yes']

specific ['Sunny' 'Warm' '?' 'Strong' '?' '?']

7. LEARNING OUTCOMES :

• Students will be able to apply Find-S algorithm to the real world problem and find the most specific hypothesis from the training data.

8. APPLICATION AREAS:

• Classification based problem

1. EXPERIMENT NO: 2
2. TITLE: **CANDIDATE-ELIMINATION ALGORITHM**
3. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
4. AIM:
   * **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.**
5. THEORY:
   * The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.
   * It computes the description of this set without explicitly enumerating all of its members.
   * This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.
   * The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.
   * A version space can be represented with its general and specific boundary sets.
   * The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
   * Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries

**Algorithm**

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
   1. If d is a positive example

Remove from G any hypothesis 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 is consistent with d, and some member of G is more general than h

Remove from S, hypothesis that is more general than another hypothesis in S 3.2. If d is a negative example

Remove from S any hypothesis 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 is consistent with d, and some member of S is more specific than h

Remove from G any hypothesis that is less general than another hypothesis in G

6. PROCEDURE / PROGRAMME :

import numpy as np

import pandas as pd

**# Loading Data from a CSV File**

data = pd.DataFrame(data=pd.read\_csv('finds.csv'))

**# Separating concept features from Target**

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

**# Isolating target into a separate DataFrame**

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

def learn(concepts, target):

specific\_h=[0,0,0,0,0,0]

import numpy as np

import pandas as pd

**# Loading Data from a CSV File**

data = pd.DataFrame(data=pd.read\_csv('finds.csv'))

**# Separating concept features from Target**

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

**# Isolating target into a separate DataFrame**

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

def learn(concepts, target):

specific\_h=[0,0,0,0,0,0]

print('s0',specific\_h)

specific\_h = concepts[0].copy()

print('s1',specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print('g0',general\_h)

for i, h in enumerate(concepts):

if target[i] == "Yes":

for x in range(len(specific\_h)):

**# Change values in S & G only if values change**

if h[x] != specific\_h[x]:

specific\_h[x] = '?'

general\_h[x][x] = '?'

print(f"s{x}",specific\_h)

print(f"g{x}",general\_h)

**# Checking if the hypothesis has a positive target**

if target[i] == "No":

for x in range(len(specific\_h)):

**# For negative hyposthesis change values only in G**

if h[x] != specific\_h[x]:

general\_h[x][x] = specific\_h[x]

else:

general\_h[x][x] = '?'

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

indices = [i for i,val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

**# remove those rows from general\_h**

general\_h.remove(['?', '?', '?', '?', '?', '?'])

print('i',indices)

**# Return final values**

return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

**OUTPUT:**

s0 [0, 0, 0, 0, 0, 0]

s1 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

g0 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

g2 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

g2 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

s4 ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']

g4 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

s5 ['Sunny' 'Warm' '?' 'Strong' '?' '?']

g5 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

i [2, 3, 4, 5]

print("Final S:", s\_final, sep="\n")

Final S:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

print("Final G:", g\_final, sep="\n")

Final G:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

1. LEARNING OUTCOMES :
   * The students will be able to apply candidate elimination algorithm and output a description of the set of all hypotheses consistent with the training examples
2. APPLICATION AREAS:
   * Classification based problems.
3. EXPERIMENT NO: 3
4. TITLE: **ID3 ALGORITHM**
5. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
6. 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 to classify a new sample.**
7. 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 Examplesvi be the subset of Examples that have value vi for A

If Examplesvi 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(Examplesvi, TargetAttribute, Attributes–{A})

End

1. Return Root

6. PROCEDURE / PROGRAMME :

**Dataset set used – playtennis.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **outlook** | **temp** | **humidity** | **wind** | **play** |
| Sunny | Hot | High | Weak | No |
| Sunny | Hot | High | Strong | No |
| Overcast | Hot | High | Weak | Yes |
| Rain | Mild | High | Weak | Yes |
| Rain | Cool | Normal | Weak | Yes |
| Rain | Cool | Normal | Strong | No |
| Overcast | Cool | Normal | Strong | Yes |
| Sunny | Mild | High | Weak | No |
| Sunny | Cool | Normal | Weak | Yes |
| Rain | Mild | Normal | Weak | Yes |
| Sunny | Mild | Normal | Strong | Yes |
| Overcast | Mild | High | Strong | Yes |
| Overcast | Hot | Normal | Weak | Yes |
| Rain | Mild | High | Strong | No |

import csv

import math

import random

# Class Node which will be used while classify a test-instance using the tree which was built earlier

class Node():

    value = ""

    children = []

    def \_\_init\_\_(self, val, dictionary):

        self.value = val

        if (isinstance(dictionary, dict)):

            self.children = dictionary.keys()

# Majority Function which tells which class has more entries in given data-set

def majorClass(attributes, data, target):

    freq = {}

    index = attributes.index(target)

    for tuple in data:

        if tuple[index] in freq:

            freq[tuple[index]] += 1

        else:

            freq[tuple[index]] = 1

    max = 0

    major = ""

    for key in freq.keys():

        if freq[key]>max:

            max = freq[key]

            major = key

    return major

# Calculates the entropy of the data given the target attribute

def entropy(attributes, data, targetAttr):

    freq = {}

    dataEntropy = 0.0

    i = 0

    for entry in attributes:

        if (targetAttr == entry):

            break

        i = i + 1

    i = i - 1

    for entry in data:

        if entry[i] in freq:

            freq[entry[i]] += 1.0

        else:

            freq[entry[i]]  = 1.0

    for freq in freq.values():

        dataEntropy += (-freq/len(data)) \* math.log(freq/len(data), 2)

    return dataEntropy

# Calculates the information gain (reduction in entropy) in the data when a particular attribute is chosen for splitting the data.

def info\_gain(attributes, data, attr, targetAttr):

    freq = {}

    subsetEntropy = 0.0

    i = attributes.index(attr)

    for entry in data:

        if entry[i] in freq:

            freq[entry[i]] += 1.0

        else:

            freq[entry[i]]  = 1.0

    for val in freq.keys():

        valProb        = freq[val] / sum(freq.values())

        dataSubset     = [entry for entry in data if entry[i] == val]

        subsetEntropy += valProb \* entropy(attributes, dataSubset, targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)

# This function chooses the attribute among the remaining attributes which has the maximum information gain.

def attr\_choose(data, attributes, target):

    best = attributes[0]

    maxGain = 0;

    for attr in attributes:

        newGain = info\_gain(attributes, data, attr, target)

        if newGain>maxGain:

            maxGain = newGain

            best = attr

    return best

# This function will get unique values for that particular attribute from the given data

def get\_values(data, attributes, attr):

    index = attributes.index(attr)

    values = []

    for entry in data:

        if entry[index] not in values:

            values.append(entry[index])

    return values

# This function will get all the rows of the data where the chosen "best" attribute has a value "val"

def get\_data(data, attributes, best, val):

    new\_data = [[]]

    index = attributes.index(best)

    for entry in data:

        if (entry[index] == val):

            newEntry = []

            for i in range(0,len(entry)):

                if(i != index):

                    newEntry.append(entry[i])

            new\_data.append(newEntry)

    new\_data.remove([])

    return new\_data

# This function is used to build the decision tree using the given data, attributes and the target attributes. It returns the decision tree in the end.

def build\_tree(data, attributes, target):

    data = data[:]

    vals = [record[attributes.index(target)] for record in data]

    default = majorClass(attributes, data, target)

    if not data or (len(attributes) - 1) <= 0:

        return default

    elif vals.count(vals[0]) == len(vals):

        return vals[0]

    else:

        best = attr\_choose(data, attributes, target)

        tree = {best:{}}

        for val in get\_values(data, attributes, best):

            new\_data = get\_data(data, attributes, best, val)

            newAttr = attributes[:]

            newAttr.remove(best)

            subtree = build\_tree(new\_data, newAttr, target)

            tree[best][val] = subtree

    return tree

#Main function

def execute\_decision\_tree():

    data = []

    #load file

    with open("weather.csv") as tsv:

        for line in csv.reader(tsv):

            data.append(tuple(line))

        print("Number of records:",len(data))

        #set attributes

        attributes=['outlook','temperature','humidity','wind','play']

        target = attributes[-1]

        #set training data

        acc = []

        training\_set = [x for i, x in enumerate(data)]

        tree = build\_tree( training\_set, attributes, target )

        #execute algorithm on test data

        results = []

        test\_set = [('rainy','mild','high','strong')]

        for entry in test\_set:

            tempDict = tree.copy()

            result = ""

            while(isinstance(tempDict, dict)):

                root = Node(next(iter(tempDict)), tempDict[next(iter(tempDict))])

                tempDict = tempDict[next(iter(tempDict))]

                index = attributes.index(root.value)

                value = entry[index]

                if(value in tempDict.keys()):

                    child = Node(value, tempDict[value])

                    result = tempDict[value]

                    tempDict = tempDict[value]

                else:

                    result = "Null"

                    break

            if result != "Null":

                results.append(result == entry[-1])

        print(result)

if \_\_name\_\_ == "\_\_main\_\_":

    execute\_decision\_tree()

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| for the input   |  |  |  |  | | --- | --- | --- | --- | | Rain | Mild | High | Strong | | **Output 1:**  **Number of records: 15**  **No** |
| for the input   |  |  |  |  | | --- | --- | --- | --- | | Rain | Cool | Normal | Weak | | Output 2:  Number of records: 15  Yes |
| for the input   |  |  |  |  | | --- | --- | --- | --- | | Overcast | Hot | Normal | strong | | Output 3:  Number of records: 15  Null |

1. 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 to classify a new sample.
2. APPLICATION AREAS:
   * Classification related problem areas
3. EXPERIMENT NO: 4
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 <(x1,…xn),t>, do

// Propagate the input forward through the network:

a. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every unit // Propagate the errors backward through the network:

b. For each output unit k, calculate its error term k ; k = ok(1-ok)(tk-ok)

c. For each hidden unit h, calculate its error term h; h=oh(1-oh) k wh,k kj xi,j

6. PROCEDURE / PROGRAMME :

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

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

y = y/100

**#Sigmoid Function**

def sigmoid (x):

return 1/(1 + np.exp(-x))

**#Derivative of Sigmoid Function**

def derivatives\_sigmoid (x):

return x\*(1-x)

**#Variable initialization**

epoch=7000 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

**#weight and bias initialization**

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

**#draws a random range of numbers uniformly of dim x\*y**

for i in range(epoch):

**#Forward Propogation**

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+ bout

output = sigmoid(outinp)

**#Backpropagation**

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr# dotproduct of nextlayererror and currentlayerop

# bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**//output**

Input:

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

Predicted Output:

[[ 0.90224607]

[ 0.87341151]

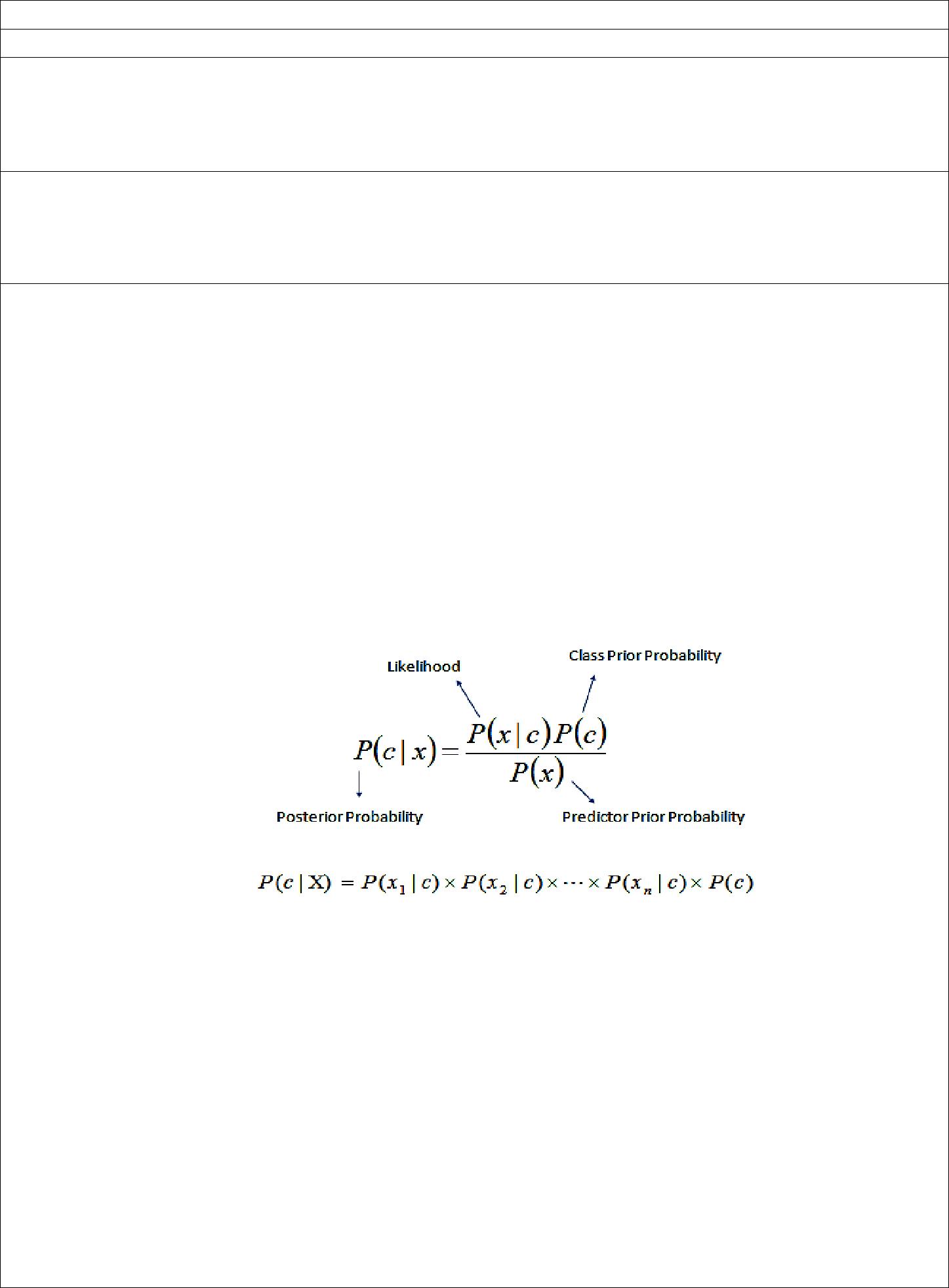
[ 0.8925683 ]]

1. 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.
2. APPLICATION AREAS:
   * Speech recognition, Character recognition, Human Face recognition
3. EXPERIMENT NO: 5
4. TITLE: **NAÏVE BAYESIAN CLASSIFIER**
5. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
6. 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.
7. 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.

1. PROCEDURE / PROGRAMME :

**Pima Indians Diabetes Database**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

* Pregnancies(P): Number of times pregnant
* Glucose(G): Plasma glucose concentration a 2 hours in an oral glucose tolerance test
* BloodPressure(BP): Diastolic blood pressure (mm Hg)
* SkinThickness(ST): Triceps skin fold thickness (mm)
* Insulin(I): 2-Hour serum insulin (mu U/ml)
* BMI: Body mass index (weight in kg/(height in m)^2)
* DiabetesPedigreeFunction(DPF): Diabetes pedigree function
* Age(A): Age (years)
* Outcome(O): Class variable (0 or 1)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | G | BP | ST | I | BMI | DPF | A | O |
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |

# Example of Naive Bayes implemented from Scratch in Python

# Online Resource: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

import csv

import random

import math

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):

trainSize = int(len(dataset) \* splitRatio)

trainSet = []

copy = list(dataset)

while len(trainSet) < trainSize:

index = random.randrange(len(copy))

trainSet.append(copy.pop(index))

return [trainSet, copy]

def separateByClass(dataset):

separated = {}

for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated):

separated[vector[-1]] = []

separated[vector[-1]].append(vector)

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 summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]

del summaries[-1]

return summaries

def summarizeByClass(dataset):

separated = separateByClass(dataset)

summaries = {}

for classValue, instances in separated.items():

summaries[classValue] = summarize(instances)

return summaries

def calculateProbability(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, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.items():

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i]

x = inputVector[i]

probabilities[classValue] \*= calculateProbability(x, mean, stdev)

return probabilities

def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel

def getPredictions(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 (float(correct)/float(len(testSet))) \* 100.0

def main():

filename = 'pima-indians-diabetes.csv'

dataset = loadCsv(filename)

trainingSet=dataset

testSet=loadCsv('pima-indians-diabetes-test-1.csv')

print('Records in training data={1} and test data={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))

# prepare model

summaries = summarizeByClass(trainingSet)

# test model

predictions = getPredictions(summaries, testSet)

print(predictions)

accuracy = getAccuracy(testSet, predictions)

print("Accuracy:",accuracy,"%")

main()

Output:

Records in training data=768 and test data=11 rows

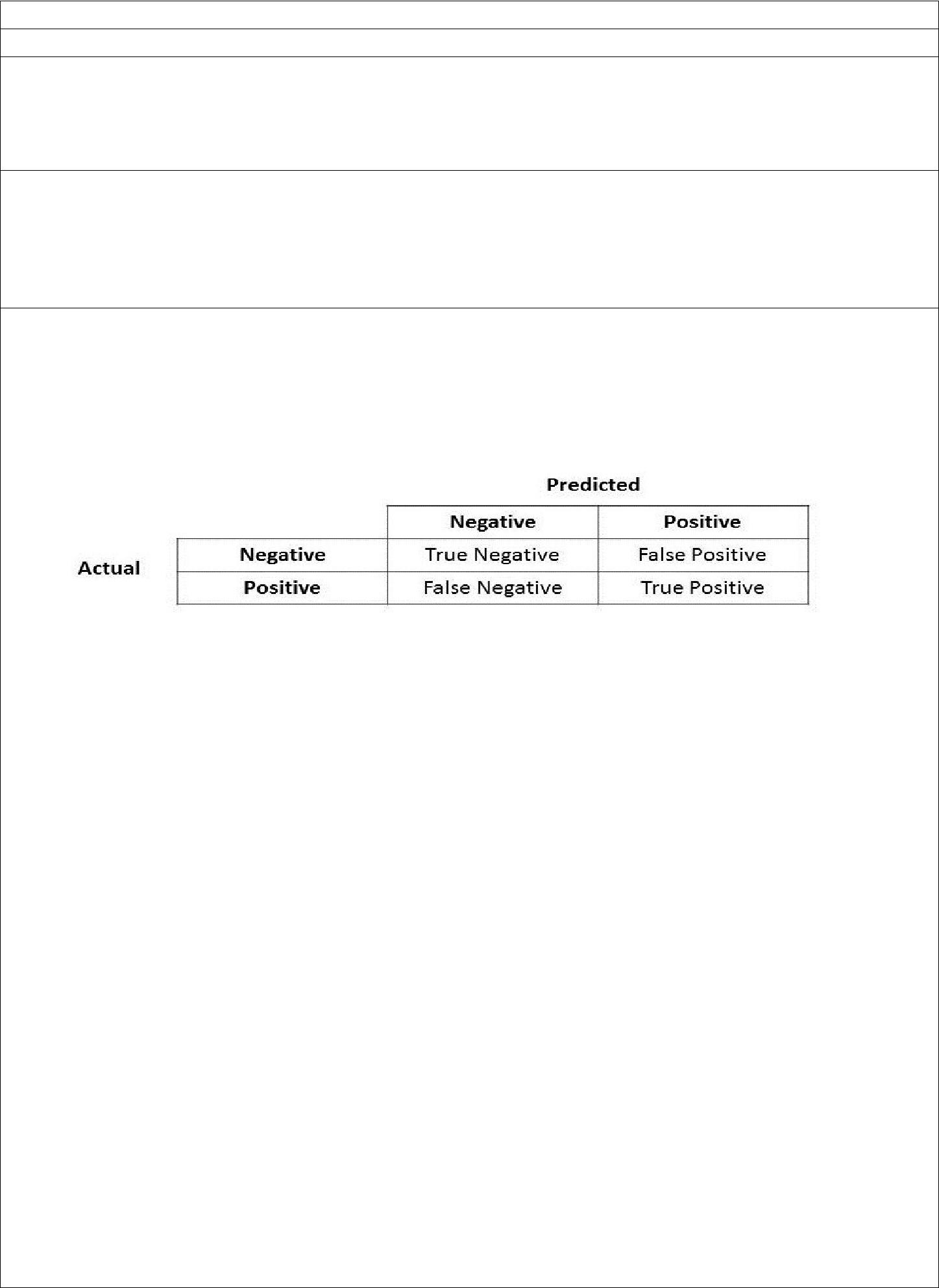
[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0]

Accuracy: 81.81818181818183 %

1. LEARNING OUTCOMES :
   * The student will be able to apply naive baysian classifier for the relevent problem and analyse the results.
2. 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
3. EXPERIMENT NO: 6
4. TITLE: **DOCUMENT CLASSIFICATION USING NAÏVE BAYESIAN CLASSIFIER**
5. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
6. AIM:
   * 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.
7. THEORY:

For the theoey of the naive bayesian classifier refer Experiment No. 5. Theory of performance anaysis analysis is ellaborated here.

Analysis of Document Classification



* For classification tasks, the terms true positives, true negatives, false positives, and false negatives compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the observation).
* Precision - Precision is the ratio of correctly predicted positive documents to the total predicted positive documents. High precision relates to the low false positive rate.

Precision = (Σ True positive ) / ( Σ True positive + Σ False positive)

* Recall (Sensitivity) - Recall is the ratio of correctly predicted positive documents to the all observations in actual class.

Recall = (Σ True positive ) / ( Σ True positive + Σ False negative)

* Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

Accuracy = (Σ True positive + Σ True negative) / Σ Total population



1. PROCEDURE / PROGRAMME :

**Data set**

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

I do not like this restaurant,

neg I am tired of this stuff,neg

I can't deal with this,neg

He is my sworn enemy,neg

My boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this juice,neg

I love to dance,pos

I am sick and tired of this place,neg

What a great holiday,pos

That is a bad locality to stay,neg

We will have good fun tomorrow,pos

I went to my enemy's house today,neg

**import** pandas **as** pd

msg=pd.read\_csv('data61.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)

***# 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])

***# 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*

*#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))

***#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))

Output:

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

Dataset is split into Training and Testing samples

Total training instances : 13

Total testing instances : 5

Total features extracted using CountVectorizer: 48

Features for first 5 training instances are listed below

about am an and awesome bad beers boss can deal ... this \

0 0 1 0 1 0 0 0 0 0 0 ... 1

1 0 0 1 0 1 0 0 0 0 0 ... 0

2 0 1 0 0 0 0 0 0 0 0 ... 1

3 0 0 0 0 0 1 0 0 0 0 ... 0

4 1 0 0 0 0 0 1 0 0 0 ... 0

tired to tomorrow very view we what will with

0 1 0 0 0 0 0 0 0 0

1 0 0 0 0 1 0 1 0 0

2 1 0 0 0 0 0 0 0 0

3 0 1 0 0 0 0 0 0 0

4 0 0 0 1 0 0 0 0 0

[5 rows x 48 columns]

Classstification results of testing samples are given below

This is my best work -> neg

I went to my enemy's house today -> neg

I love to dance -> pos

I do not like this restaurant -> neg

This is an amazing place -> pos

Accuracy metrics

Accuracy of the classifer is 0.8

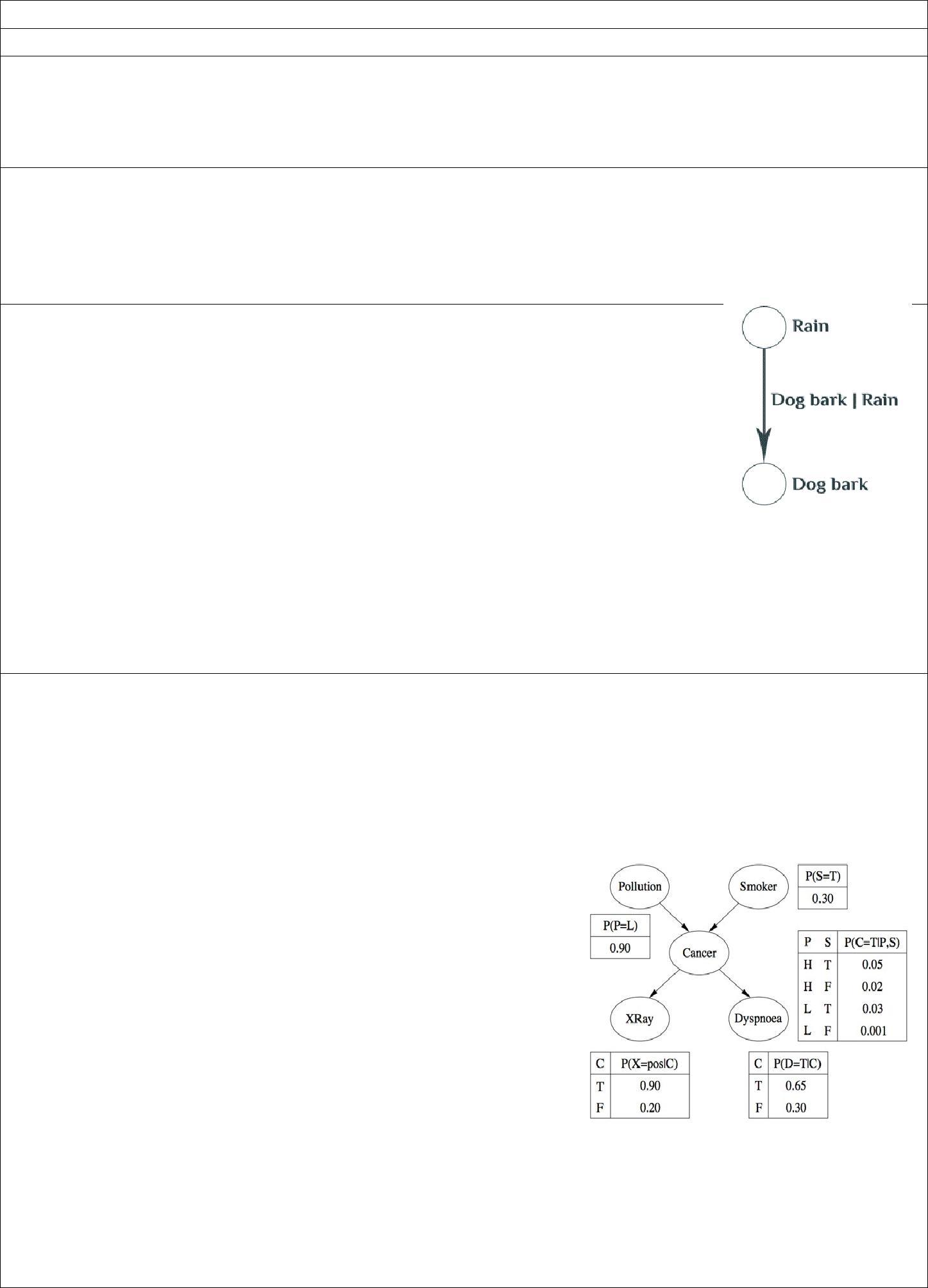
Recall : 0.666666666667

Precison : 1.0

Confusion matrix

[[2 0]

[1 2]]

1. LEARNING OUTCOMES :
   * The student will be able to apply naive baysian classifier for document classification and analyse the results.
2. APPLICATION AREAS:
   * Applicable in document classification
3. EXPERIMENT NO: 7
4. TITLE: **BAYESIAN NETWORK**
5. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
6. AIM:
   * 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.
7. THEORY:
   * Bayesian networks are very convenient for representing similar probabilistic relationships between multiple events.
   * Bayesian networks as graphs - People usually represent Bayesian networks as directed graphs in which each node is a hypothesis or a random process. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc.
   * The arrows between nodes represent the conditional probabilities between them — how information about the state of one node changes the probability distribution of another node it’s connected to.
8. PROCEDURE / PROGRAMME :

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]

#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang',

# 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']

heartDisease = pd.read\_csv('data7\_heart.csv', names = attributes)

heartDisease = heartDisease.replace('?', np.nan)

# Display the data

print('Few examples from the dataset are given below')

print(heartDisease.head())

print('\nAttributes and datatypes')

print(heartDisease.dtypes)

# Model Baysian Network

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),

('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),

('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

# Learning CPDs using Maximum Likelihood Estimators

print('\nLearning CPDs using Maximum Likelihood Estimators...');

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network

print('\nInferencing with Bayesian Network:')

HeartDisease\_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.

print('\n1.Probability of HeartDisease given Age=20')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'age': 28})

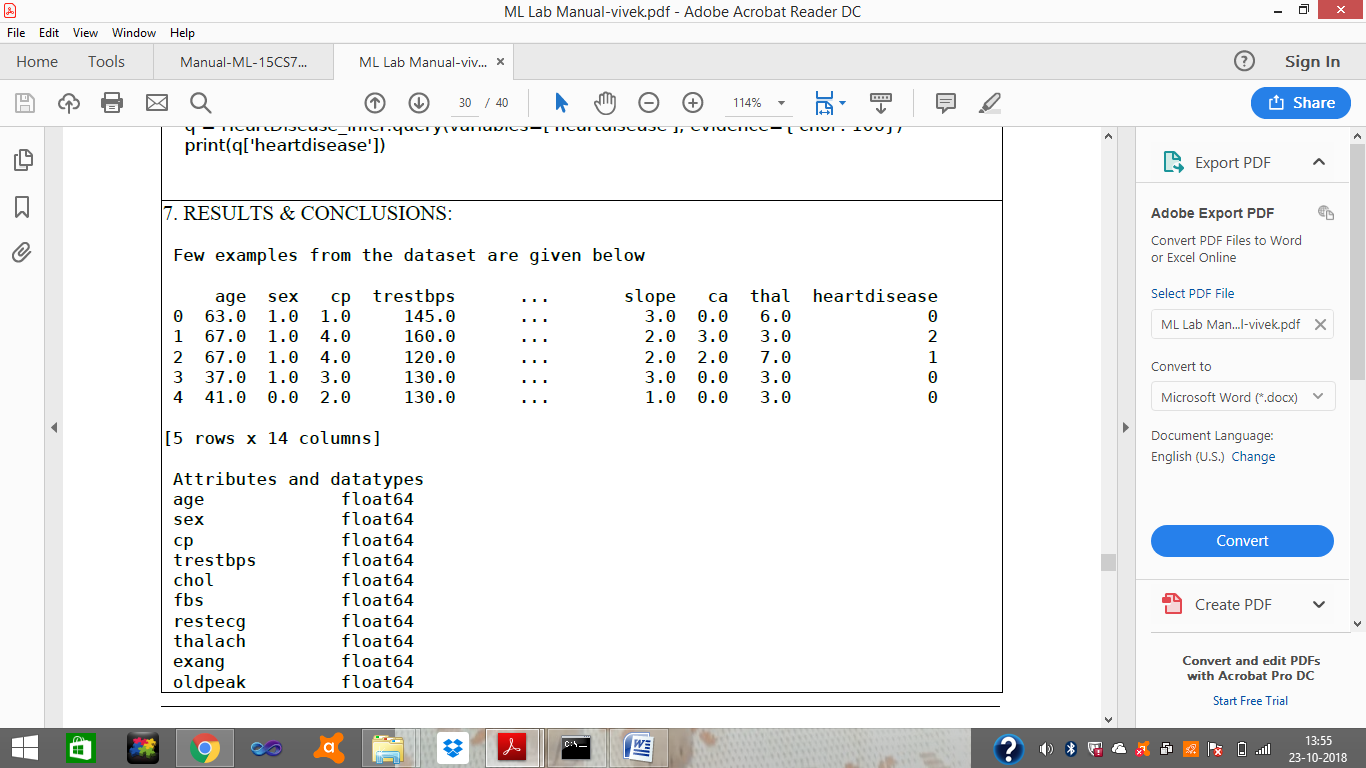
print(q['heartdisease'])

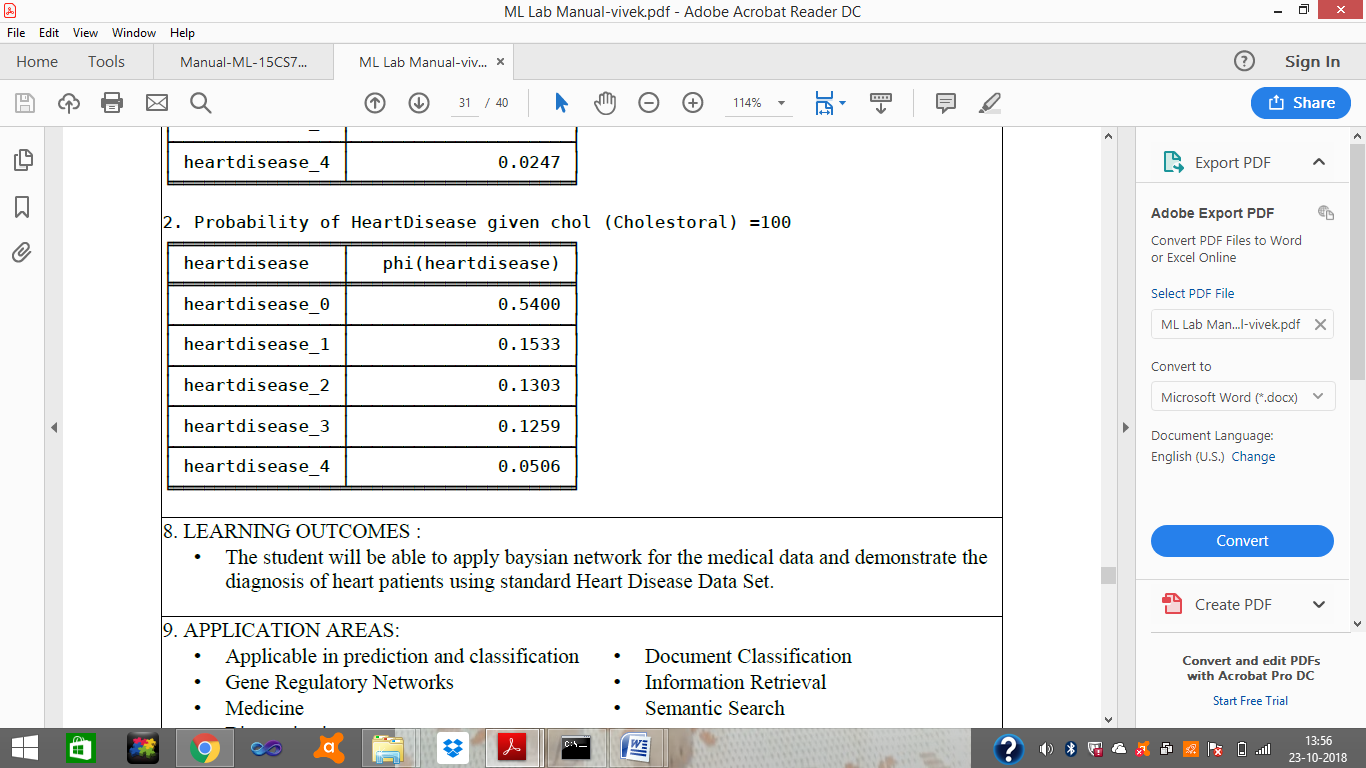
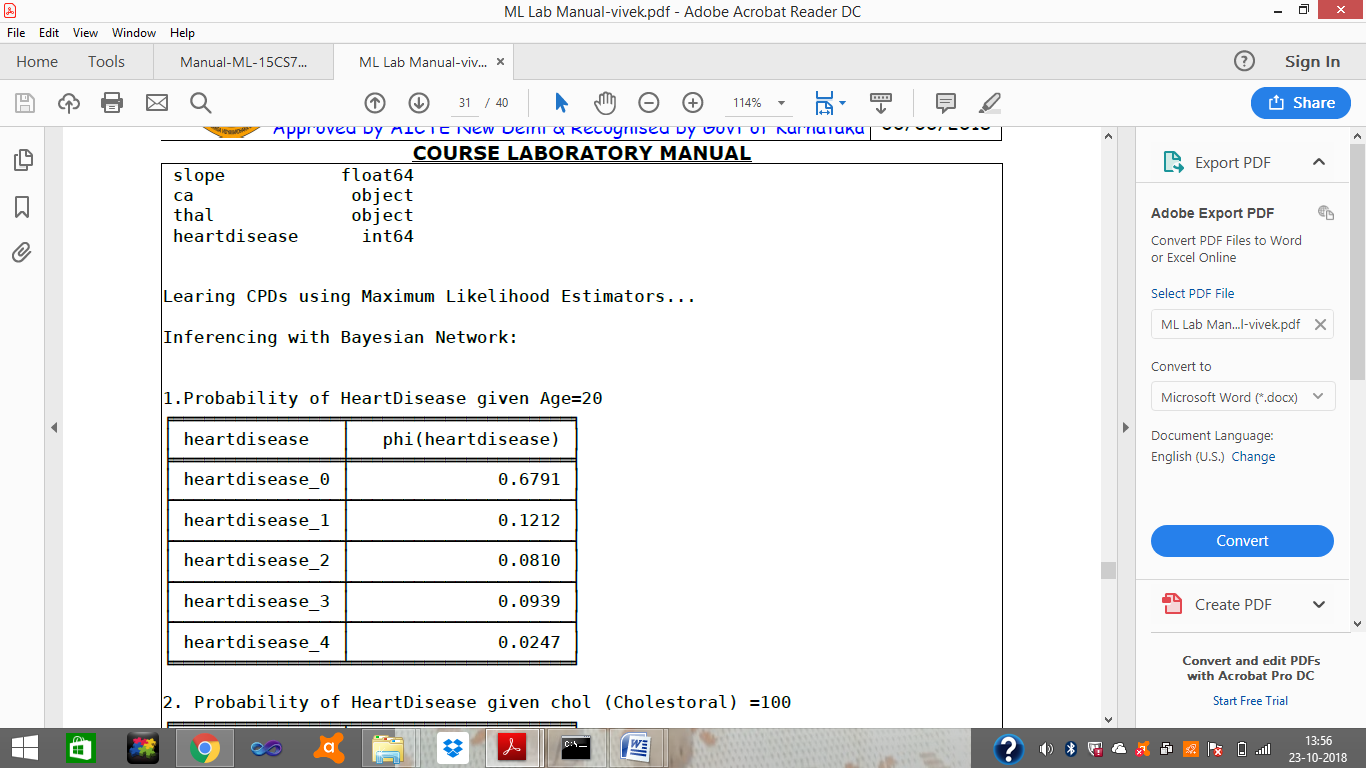
print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'chol': 100})

print(q['heartdisease'])

**Output:**





1. LEARNING OUTCOMES :
   * The student will be able to apply baysian network for the medical data and demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.
2. APPLICATION AREAS:

|  |  |  |  |
| --- | --- | --- | --- |
| • Applicable in prediction and classification | | • | Document Classification |
| • | Gene Regulatory Networks | • | Information Retrieval |
| • | Medicine | • | Semantic Search |

• Biomonitoring

1. EXPERIMENT NO: 8
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

1. 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.labels\_ : Gives cluster no for which samples belongs to**

model.fit(X)

**# 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')

plt.show()

**# 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')

plt.show()

**# 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')

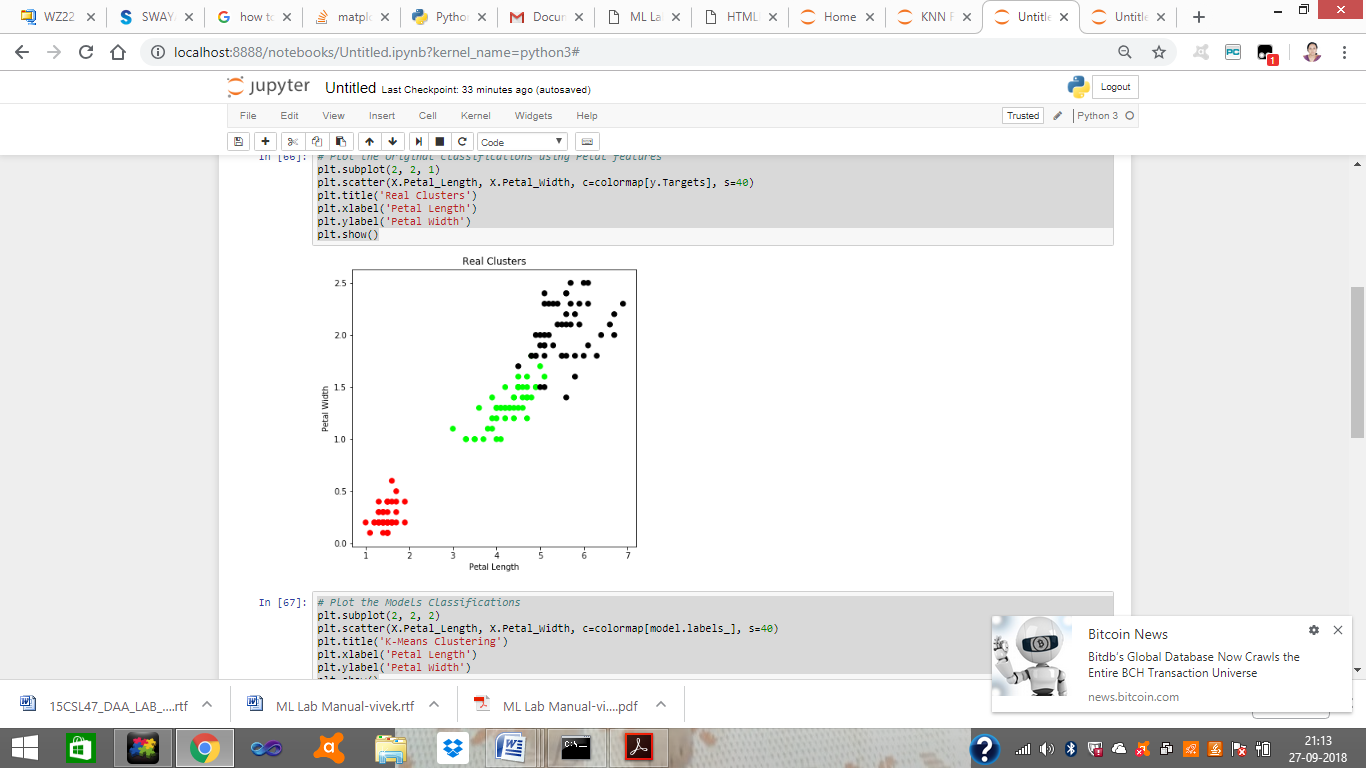
plt.xlabel('Petal Length')

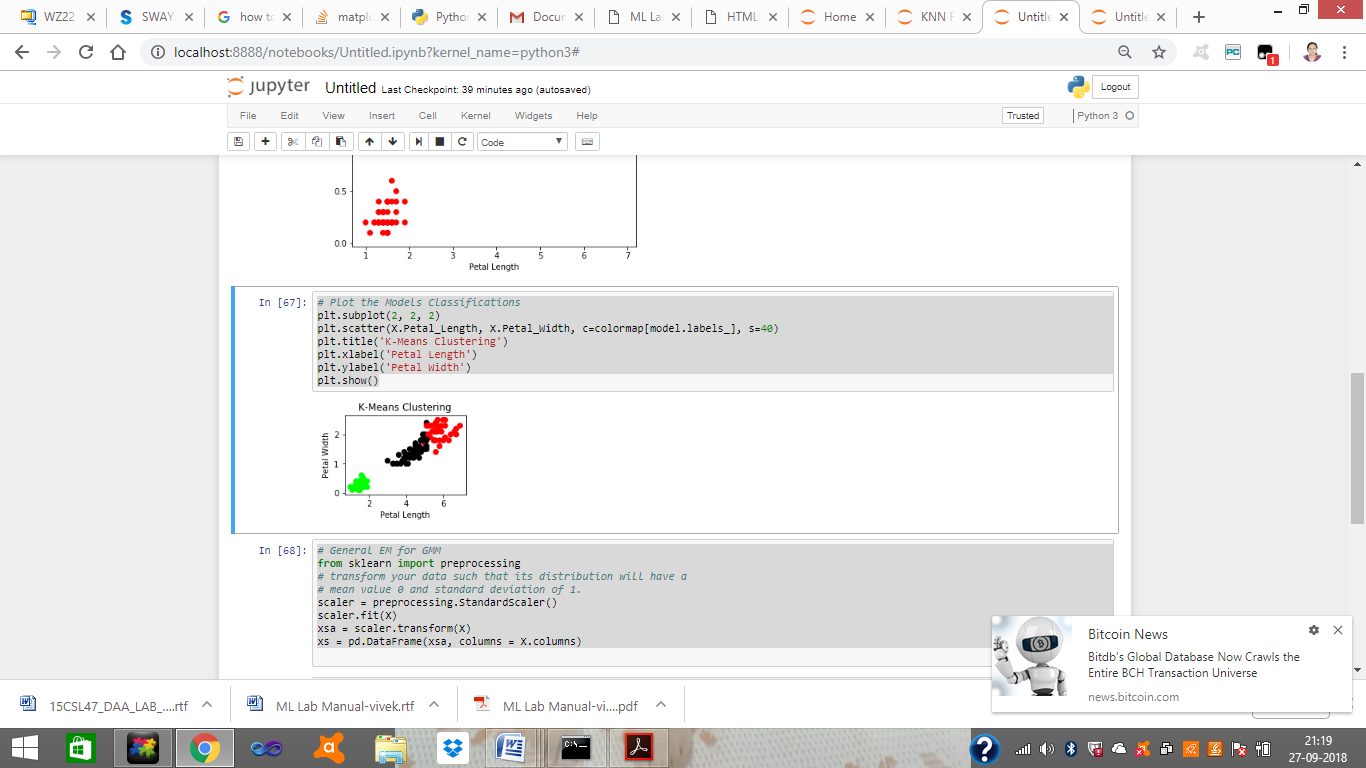
plt.ylabel('Petal Width')

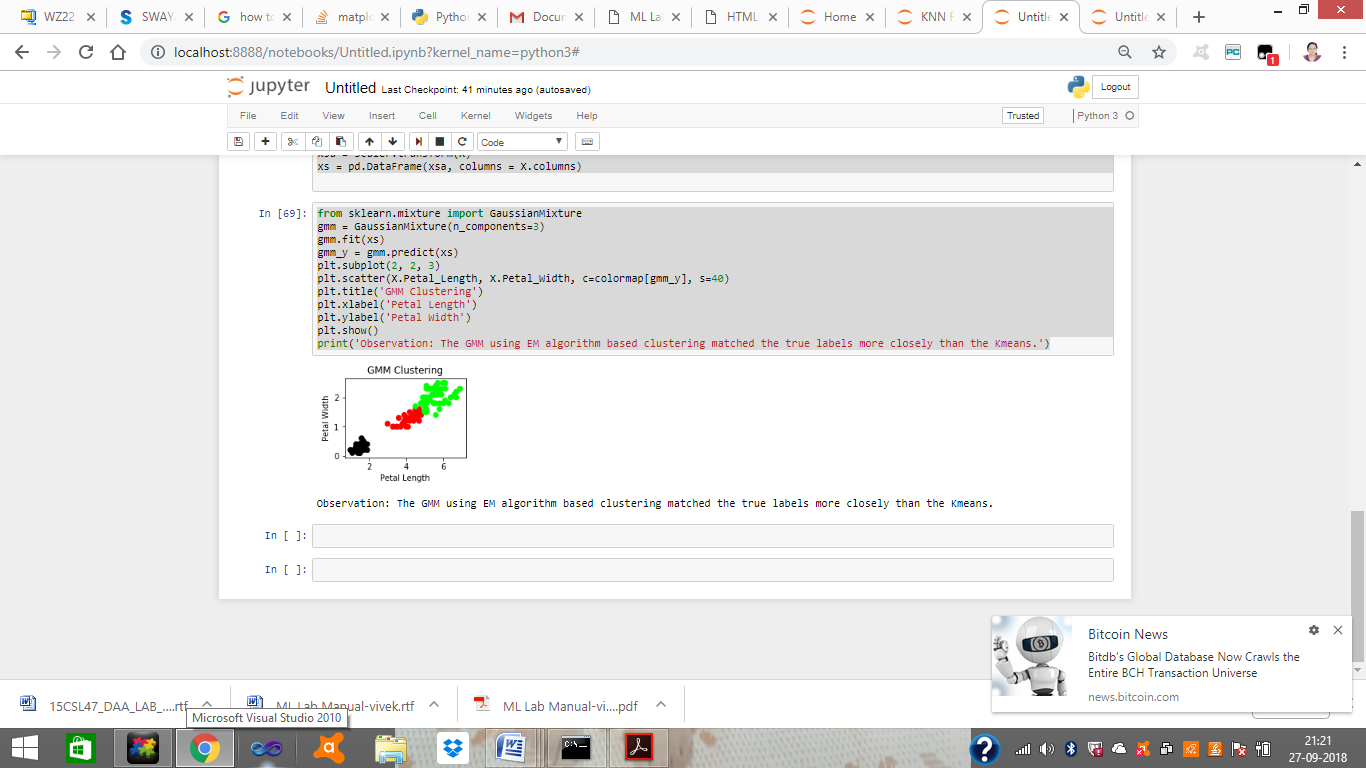
plt.show()

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

Output:







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

1. LEARNING OUTCOMES :
   * The students will be apble to apply EM algorithm and k-Means algorithm for clustering and anayse the results.
2. APPLICATION AREAS:

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

1. EXPERIMENT NO: 9
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:

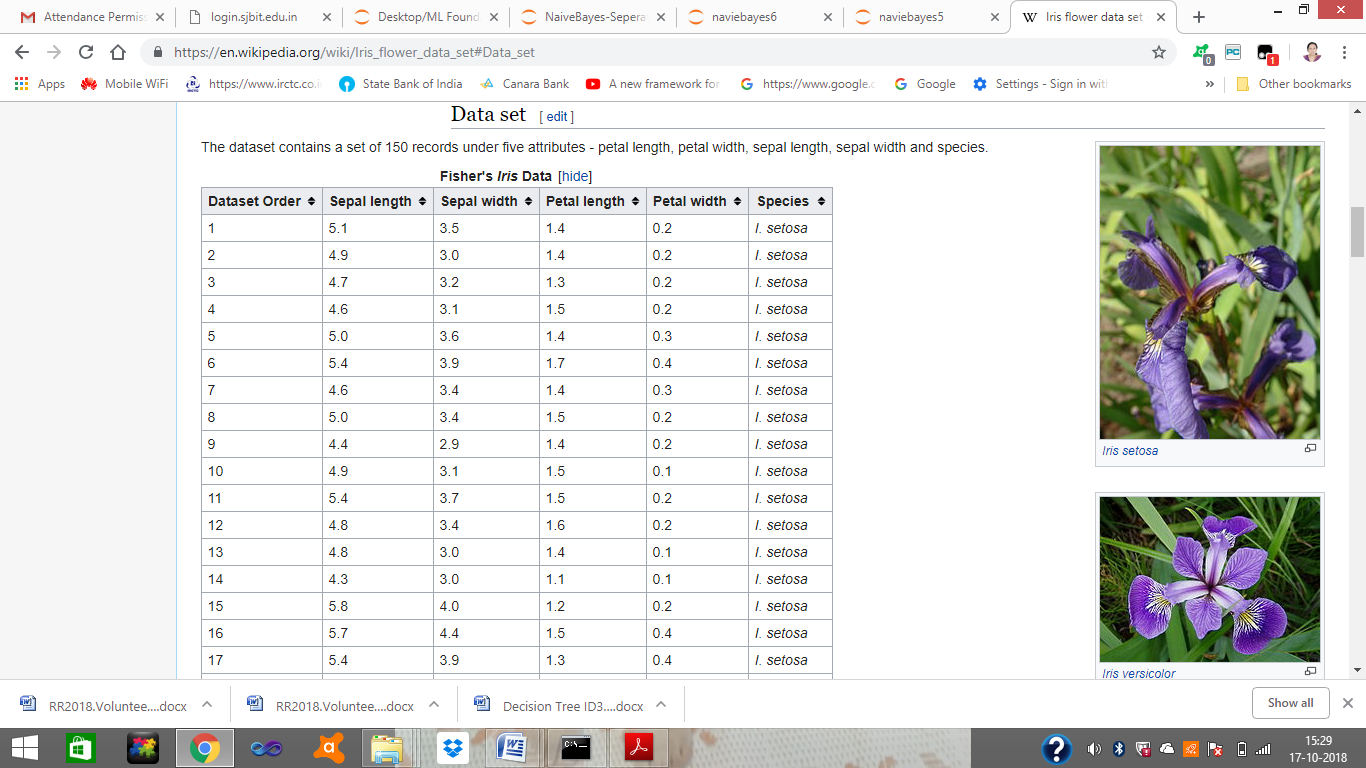
* + 1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
    2. for i=0 to m

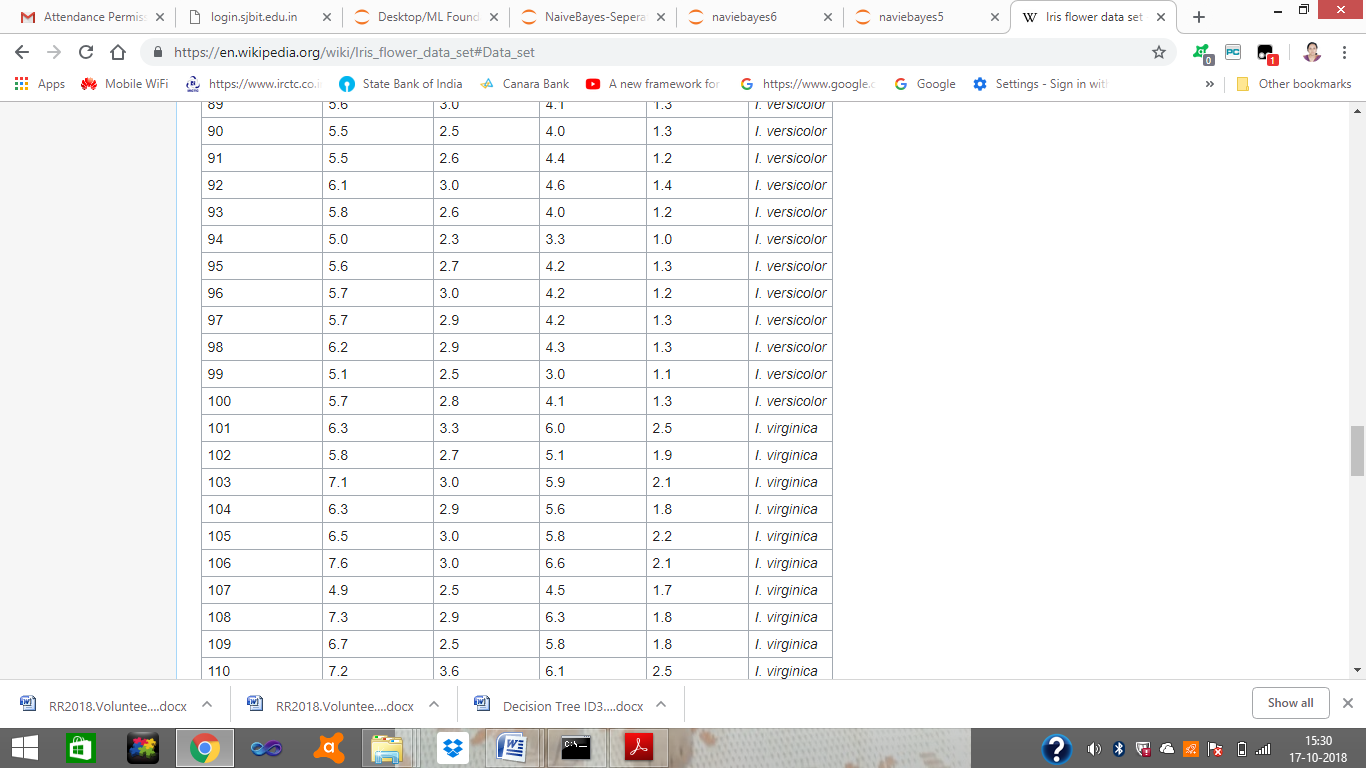
Calculate Euclidean distance d(arr[i], p).

* + 1. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
    2. Return the majority label among S.

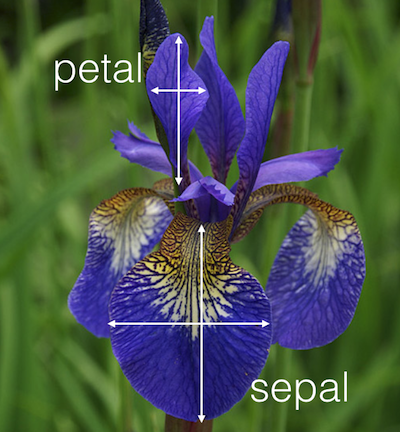
1. PROCEDURE / PROGRAMME :

Dataset: (150 Rows)





Iris flower sample image



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))

**Output:**

**Iris Data set loaded...**

**Dataset is split into training and testing...**

**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. 2.2 4. 1. ] Actual-label: 1 Predicted-label: 1**

**Sample: [ 4.8 3. 1.4 0.3] Actual-label: 0 Predicted-label: 0**

**Sample: [ 5.8 2.6 4. 1.2] Actual-label: 1 Predicted-label: 1**

**Sample: [ 5.8 2.7 3.9 1.2] Actual-label: 1 Predicted-label: 1**

**Sample: [ 5.1 3.5 1.4 0.2] Actual-label: 0 Predicted-label: 0**

**Sample: [ 6.3 2.3 4.4 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 3.1 5.6 2.4] Actual-label: 2 Predicted-label: 2**

**Sample: [ 5.5 4.2 1.4 0.2] Actual-label: 0 Predicted-label: 0**

**Sample: [ 5.6 3. 4.5 1.5] Actual-label: 1 Predicted-label: 1**

**Sample: [ 6.1 3. 4.9 1.8] Actual-label: 2 Predicted-label: 2**

**Sample: [ 6. 3.4 4.5 1.6] Actual-label: 1 Predicted-label: 1**

**Sample: [ 6.1 2.6 5.6 1.4] Actual-label: 2 Predicted-label: 2**

**Sample: [ 5.5 2.4 3.8 1.1] Actual-label: 1 Predicted-label: 1**

**Sample: [ 5.6 2.9 3.6 1.3] Actual-label: 1 Predicted-label: 1**

**Classification Accuracy : 0.933333333333**

**Confusion Matrix**

**[[3 0 0]**

**[0 8 1]**

**[0 0 3]]**

**Accuracy Metrics**

**precision recall f1-score support**

**0 1.00 1.00 1.00 3**

**1 1.00 0.89 0.94 9**

**2 0.75 1.00 0.86 3**

**avg / total 0.95 0.93 0.94 15**

1. 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.
2. APPLICATION AREAS:
   * Recommender systems
   * Classification problems
3. EXPERIMENT NO: 10
4. TITLE: **LOCALLY WEIGHTED REGRESSION ALGORITHM**
5. LEARNING OBJECTIVES:
   * Make use of Data sets in implementing the machine learning algorithms.
   * Implement ML concepts and algorithms in Python
6. 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.
7. 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.

6. PROCEDURE / PROGRAMME : Dataset: (244 Rows)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| total\_bill | tip | sex | smoker | day | time | size |
| 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 21.01 | 3.5 | Male | No | Sun | Dinner | 3 |
| 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |
| 25.29 | 4.71 | Male | No | Sun | Dinner | 4 |
| 8.77 | 2 | Male | No | Sun | Dinner | 2 |
| 26.88 | 3.12 | Male | No | Sun | Dinner | 4 |
| 15.04 | 1.96 | Male | No | Sun | Dinner | 2 |
| 14.78 | 3.23 | Male | No | Sun | Dinner | 2 |
| 10.27 | 1.71 | Male | No | Sun | Dinner | 2 |
| 35.26 | 5 | Female | No | Sun | Dinner | 4 |
| 15.42 | 1.57 | Male | No | Sun | Dinner | 2 |
| 18.43 | 3 | Male | No | Sun | Dinner | 4 |
| 14.83 | 3.02 | Female | No | Sun | Dinner | 2 |
| 21.58 | 3.92 | Male | No | Sun | Dinner | 2 |

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

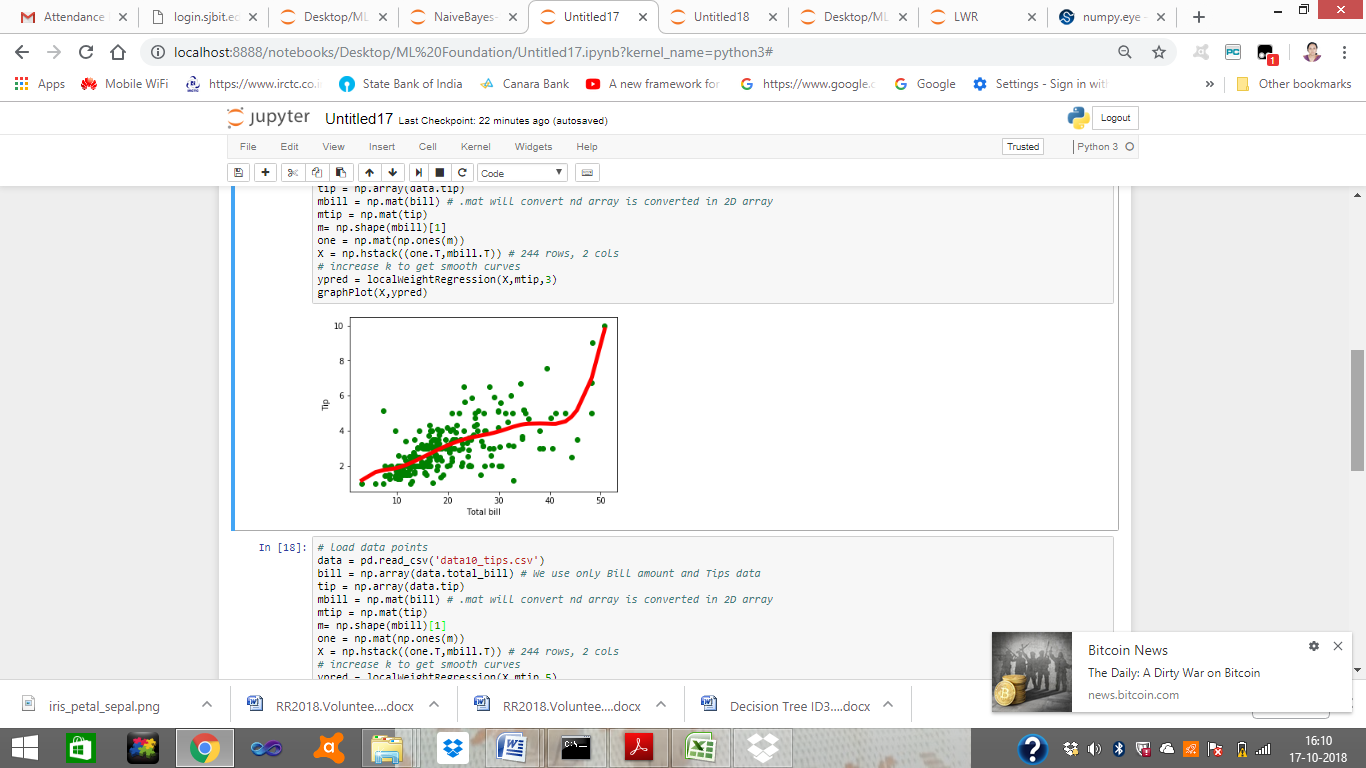
# increase k to get smooth curves

ypred = localWeightRegression(X,mtip,3)

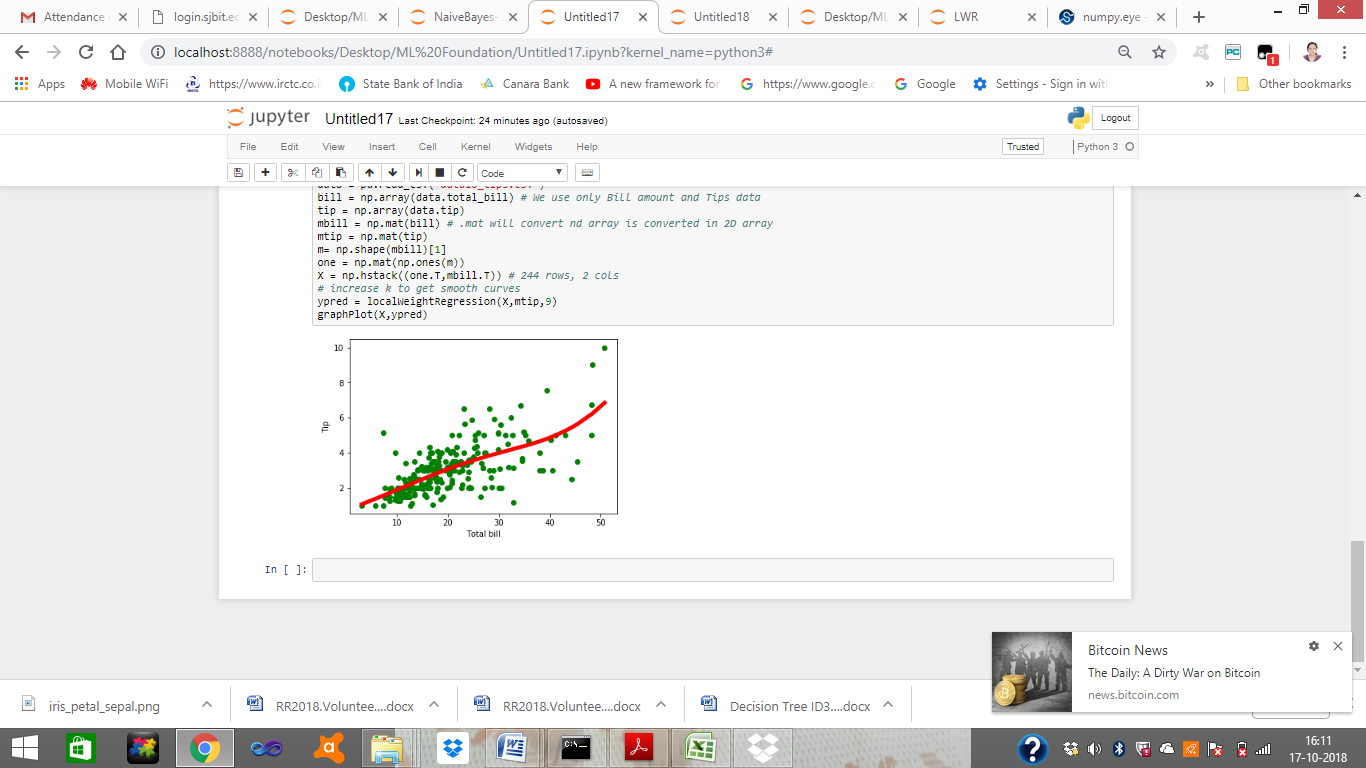
graphPlot(X,ypred)

Output:

|  |  |
| --- | --- |
| Regression with parameter k = 3 |  |



Regression with parameter k = 9



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

**ML viva Questions**

1. What’s the trade-off between bias and variance?
2. What is the difference between supervised and unsupervised machine learning?
3. How is KNN different from k-means clustering?
4. Define precision and recall.
5. What is Bayes’ Theorem? How is it useful in a machine learning context?
6. Why is “Naive” Bayes naive?
7. What’s the difference between probability and likelihood?
8. How is a decision tree pruned?
9. What’s the F1 score? How would you use it?
10. When should you use classification over regression?
11. How do you ensure you’re not over fitting with a model?
12. How would you evaluate a logistic regression model?
13. How do you handle missing or corrupted data in a dataset?
14. **How is True Positive Rate and Recall related? Write the equation.**
15. **What is the difference between supervised and unsupervised machine learning?**
16. Comparison between Machine Learning and Big Data
17. **What is deep learning?**
18. Compare kmeans and EM algorithm
19. Compare find s and candidate elimination algorithm
20. Compare candidate elimination algorithm with decision tree
21. Define numpy
22. Define pandas
23. What is matplotlib? Why it is used?
24. What is tensor flow
25. Define data frame
26. What is hypothesis
27. What is perceptron
28. What is back propagation
29. What is ANN
30. What is feed forward network
31. Define multilayer perceptron
32. Define concept learning.