Geetha Shishu Shikshana Sangha(R)

GSSS INSTITUTE OF ENGINEERING & TECHNOLOGY FOR WOMEN

(Affiliated to VTU, Belagavi ,Approved by AICTE, New Delhi Govt. of Karnataka)

K R S Road, Metagalli, Mysuru-570016.

LAB MANUAL

MACHINE LEARNING LABORATORY 15CSL76

As per Choice Based Credit System (CBCS) scheme
(Effective from the academic year 2016 -2017)
VII SEMESTER

Prepared by

MANJUNATH S, Assistant Professor
MANJUPRASAD B, Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA, New Delhi,(Validity : 01.07.2017 - 30.06.2020))

July 2018

DEPARTMENT VISION

Knowledge dissemination with development of future leaders in Information Technology having a research blend.

****

DEPARTMENT MISSION

M1: Equip students with continuous learning process to acquire Hardware, Software and Computing knowledge to face new challenges.
M2: Inculcate the core Computer Science and Engineering components with discipline among the students by providing the state-of-the-art learner centric environment.
M3: To impart essential knowledge through quality and value based education to mould them as a complete Computer Science Engineer with ethical values, leadership roles by possessing good communication skills and ability to work effectively as a team member.
M4: Provide a platform to collaborate with successful people from entrepreneurial and research domains to learn and accomplish.

****

Program Educational Objectives (PEOs)

PEO1: To produce graduates satisfying Computer Science Engineering challenges.
PEO2: To meet dynamic requirements of IT industries professionally and ethically along with social responsibilities.
PEO3: To provide Computer Science and Engineering graduates to support nationŠs self employment growth with women entrepreneurial skills.
PEO4: To equip Graduates with minimum research blend for further career challenges internationally.

****

PROGRAM OUTCOMES- POs

PO1-Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2-Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3-Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4-Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5-Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6-The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7-Environment and sustainability: Understand theimpact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8-Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO9-Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10-Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11-Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to oneŠs own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12-Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

****

PROGRAM SPECIFIC OUTCOMES- PSO

PSO1: Enable students to design system and system architecture, inculcating software, computing and analytical ability.
PSO2: Enhance skills to be successful in National, International level competition like GATE, GRE, GMAT.

****

Acknowledgement

First, we would like to express our sincere thank to the **Dr. M Shivakumar M, Principal, GSSSIETW, Mysuru and Dr. S Meenakshi Sundaram, Professor & Head, Dept. of CSE, GSSSIETW, Mysuru** for providing their continuous support for attending various Machine Learning Workshops and Faculty Development Programs which helped us to prepare this Lab Manual.

We would like to sincerely owe gratitude to for Speckbit, Google, Dept. of CSE, SJBIT, Bangalore, Github and UCI Repository for providing us materials, data sets, programs and Python related information, without these getting this Lab Manual would have been difficult.

Last but not the least, We would like to thank **all the Staff of Dept.of CSE, GSSSIETW, Mysuru** for motivating and helped in preparing this Lab Manual

Manjunath S & Manjuprasad B
Assistant Professor
Dept. of CSE, GSSSIETW, Mysuru
e-mail:manjunath@gssss.edu.in, manjuprasad32@gsss.edu.in

VTU Syllabus
MACHINE LEARNING LABORATORY

As per Choice Based Credit System (CBCS) scheme

(Effective from the academic year 2016 -2017)

SEMESTER VII

Subject Code	15CSL76	IA Marks	20
Number of Lecture HoursWeek	01I + 02P	Exam Marks	80
Total Number of Lecture Hours	40	Exam Hours	03
CREDITS:02			

Course objectives: This course will enable students to

1. Make use of Data sets in implementing the machine learning algorithms
2. Implement the machine learning concepts and algorithms in any suitable language of choice.

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 toclassify a new sample.
4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
5. 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.

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 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 and draw graphs.

Study Experiment / Project: NIL

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:20 + 50 +10 (80)

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

Lab Cycle

COURSE OUTCOME: 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

Week	NO.	Experiment	COs
01		Introduction to Machine Learning Tools	.
02	01	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.	1,2,3,4
03	02	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.	1,2,3,4
04	03	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.	1,2,3,4
05	04	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	1,2,3,4
06	05	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.	1,2,3,4
07	06	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.	1,2,3,4
08	07	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.	1,2,3,4
09	08	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.	1,2,3,4
10	09	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.	1,2,3,4
11	10	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	1,2,3,4
12	.	Lab Internals	.

Contents

Acknowledgement iii

VTU Syllabus iv

Lab Cycle vi

1 FIND-S Algorithm 1

2 Candidate-Elimination Algorithm 3

3 ID3 Algorithm 6

4 Backpropagation Algorithm 12

5 Naive Bayesian Classifier 16

6 Naive Bayesian Classifier 19

7 Bayesian Network 23

8 EM Algorithm and k-Means Algorithm 25

9 k-Nearest Neighbour Algorithm 30

10 Locally Weighted Regression Algorithm 36

Date Sets and Useful Links 40

Experiment No. 1

FIND-S Algorithm

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.

```
import numpy as np
import pandas as pd
data = pd.read_csv('finds.csv')
def train(concepts, target):
    for i, val in enumerate(target):
        if val == "Yes":
            specific_h = concepts[i]
            break

    for i,h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] == specific_h[x]:
                    pass
                else:
                    specific_h[x] = "?"
                    return specific_h

#slicing rows and column, : means begining to end of row
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print(train(concepts,target))
```

==

Data Set: finds.csv

Sky	Airtemp	Humidity	Wind	Water	Forecast	WaterSport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Cloudy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

==

OUTPUT:

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

Experiment No. 2

Candidate-Elimination Algorithm

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.

```
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):
    '''
    learn() function implements the learning method of the
    Candidate elimination algorithm.
    Arguments:
    concepts - a data frame with all the features
    target - a data frame with corresponding output values

    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing
    to the same memory location '''
    specific_h = concepts[0].copy()

    # Initialises G0 using list comprehension
    # Creates as many lists inside as there are arguments,
```

```python
    # that which later will be replaced with actual parameters
    # G0 = [['?', '?', '?', '?', '?', '?'],
    # ['?', '?', '?', '?', '?', '?'],
    # ['?', '?', '?', '?', '?', '?'],
    # ['?', '?', '?', '?', '?', '?'],
    # ['?', '?', '?', '?', '?', '?'],
    # ['?', '?', '?', '?', '?', '?']]
    general_h = [["?" for i in range(len(specific_h))] for i in
        ↪ range(len(specific_h))]

    # The learning iterations
    for i, h in enumerate(concepts):

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

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

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

                # For negative 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(['?', '?', '?', '?', '?', '?'])

    # Return final values
    return specific_h, general_h
```

```
s_final, g_final = learn(concepts, target)
print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
data.head()
```

==

Data Set: finds.csv

Sky	Airtemp	Humidity	Wind	Water	Forecast	WaterSport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Cloudy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

==

OUTPUT:

```
Final G:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Final G:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Experiment No. 3

ID3 Algorithm

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.

```
import csv
import math
import random
#Function 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 )
        print(tree)

        #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)):
                child=[]
                nodeVal=next(iter(tempDict))
                child=tempDict[next(iter(tempDict))].keys()
                tempDict = tempDict[next(iter(tempDict))]
                index = attributes.index(nodeVal)
```

```
                value = entry[index]
                if(value in tempDict.keys()):
                    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()
```

==

Data Set: weather.csv

id	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rainy	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rainy	mild	high	strong	no

==

OUTPUT:

```
Number of records: 15
{'outlook':
{'id': 'wind',
'1': 'weak',
'2': 'strong',
'3': 'weak',
'4': 'weak',
'5': 'weak',
```

```
'6': 'strong',
'7': 'strong',
'8': 'weak',
'9': 'weak',
'10': 'weak',
'11': 'strong',
'12': 'strong',
'13': 'weak',
'14': 'strong'}
}
Null
```

Experiment No. 4

Backpropagation Algorithm

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

```python
from math import exp
from random import seed
from random import random
# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for
            ↪ i in range(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for
            ↪ i in range(n_outputs)]
        network.append(output_layer)
        return network
# Calculate neuron activation for an input
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation
# Transfer neuron activation
def transfer(activation):
        return 1.0 / (1.0 + exp(-activation))
# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
```

```
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)
# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
```

```
                sum_error = 0
                for row in train:
                        outputs = forward_propagate(network, row)
                        expected = [0 for i in range(n_outputs)]
                        expected[row[-1]] = 1
                        sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
                        backward_propagate_error(network, expected)
                        update_weights(network, row, l_rate)
                print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
        [1.465489372,2.362125076,0],
        [3.396561688,4.400293529,0],
        [1.38807019,1.850220317,0],
        [3.06407232,3.005305973,0],
        [7.627531214,2.759262235,1],
        [5.332441248,2.088626775,1],
        [6.922596716,1.77106367,1],
        [8.675418651,-0.242068655,1],
        [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
print(network)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
        print(layer)
```

OUTPUT:

```
[[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]},
 {'weights': [0.2550690257394217, 0.49543508709194095, 0.4494910647887381]}],
[{'weights': [0.651592972722763, 0.7887233511355132, 0.0938595867742349]},
{'weights': [0.02834747652200631, 0.8357651039198697, 0.43276706790505337]}]]
>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
```

```
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297],
'output': 0.029980305604426185, 'delta': -0.0059546604162323625},
{'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716],
'output': 0.9456229000211323, 'delta': 0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275],
 'output': 0.23648794202357587, 'delta': -0.04270059278364587},
{'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715],
'output': 0.7790535202438367, 'delta': 0.03803132596437354}]
```

Experiment No. 5

Naive Bayesian Classifier

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.

```python
# Example of Naive Bayes implemented from Scratch in 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)):
        #print(testSet[i][-1]," ",predictions[i])
        if testSet[i][-1] == predictions[i]:
            correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        filename = 'pima-indians-diabetes.data.csv'
        splitRatio = 0.67
        dataset = loadCsv(filename)
        trainingSet,testSet=splitDataset(dataset, splitRatio) #dividing into training and test data
        #trainingSet = dataset #passing entire dataset as training data
        #testSet=[[8.0,183.0,64.0,0.0,0.0,23.3,0.672,32.0]]
        print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))
        # prepare model
        summaries = summarizeByClass(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: {0}%'.format(accuracy))

main()
```

OUTPUT:

```
Split 768 rows into train=514 and test=254 rows
1.0   1.0
Accuracy: 0.39370078740157477%
```

Experiment No. 6

Naive Bayesian Classifier

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.

```python
# Example of Naive Bayes implemented from Scratch in 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
```

```python

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)]

        print('sa',summaries)
        del summaries[-1]
        print('ss',summaries)
        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 (correct/float(len(testSet))) * 100.0

def main():
        filename = 'weather1.csv'
        splitRatio = 0.67
        dataset = loadCsv(filename)
        trainingSet=dataset
        #testSet = splitDataset(dataset, splitRatio)
        #[[8.0,183.0,64.0,0.0,0.0,23.3,0.672,32.0]]
        testSet=loadCsv('weathertest1.csv')
        print(testSet)
        print('Split {0} rows into train={1} and test={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: {0}%'.format(accuracy))

```

```
main()
```

==================================

OUTPUT:

```
Accurancy:
0.966666666667
-------------------------------------------------
Prediction
[1 2]
```

Experiment No. 7

Bayesian Network

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.

```python
# This example could be simplified a little bit by using Bernoulli instead of
# Categorical, but Categorical makes it possible to use more categories than
# just TRUE and FALSE.
from bayespy.nodes import Categorical, Mixture
from bayespy.inference import VB
import numpy as np
# NOTE: Python's built-in booleans don't work nicely for indexing, thus define
# own variables:
FALSE = 0
TRUE = 1

def _or(p_false, p_true):
    """
    Build probability table for OR-operation of two parents
    p_false: Probability table to use if both are FALSE
    p_true: Probability table to use if one or both is TRUE
    """
    return np.take([p_false, p_true], [[FALSE, TRUE], [TRUE, TRUE]], axis=0)

asia = Categorical([0.5, 0.5])
tuberculosis = Mixture(asia, Categorical, [[0.99, 0.01], [0.8, 0.2]])
smoking = Categorical([0.5, 0.5])
lung = Mixture(smoking, Categorical, [[0.98, 0.02], [0.25, 0.75]])
bronchitis = Mixture(smoking, Categorical, [[0.97, 0.03], [0.08, 0.92]])
xray = Mixture(tuberculosis, Mixture, lung, Categorical,
               _or([0.96, 0.04], [0.115, 0.885]))

dyspnea = Mixture(bronchitis, Mixture, tuberculosis, Mixture, lung, Categorical,
```

```
                    [_or([0.6, 0.4], [0.18, 0.82]),
                     _or([0.11, 0.89], [0.04, 0.96])])

# Mark observations
tuberculosis.observe(TRUE)
smoking.observe(FALSE)
bronchitis.observe(TRUE) # not a "chance" observation as in the original example

# Run inference
Q = VB(dyspnea, xray, bronchitis, lung, smoking, tuberculosis, asia)
Q.update(repeat=100)

# Show results
print("P(asia):", asia.get_moments()[0][TRUE])
print("P(tuberculosis):", tuberculosis.get_moments()[0][TRUE])
print("P(smoking):", smoking.get_moments()[0][TRUE])
print("P(lung):", lung.get_moments()[0][TRUE])
print("P(bronchitis):", bronchitis.get_moments()[0][TRUE])
print("P(xray):", xray.get_moments()[0][TRUE])
print("P(dyspnea):", dyspnea.get_moments()[0][TRUE])
```

==

OUTPUT:

```
Iteration 1: loglike=-6.453500e+00 (0.004 seconds)
Iteration 2: loglike=-6.453500e+00 (0.004 seconds)
Converged at iteration 2.
P(asia): 0.952380952381
P(tuberculosis): 1.0
P(smoking): 0.0
P(lung): 0.02
P(bronchitis): 1.0
P(xray): 0.885
P(dyspnea): 0.96
```

Experiment No. 8

EM Algorithm and k-Means Algorithm

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.

```
import numpy as np
import math
import matplotlib.pyplot as plt
import csv
def get_binomial_log_likelihood(obs,probs):
    """ Return the (log)likelihood of obs, given the probs"""
    # Binomial Distribution Log PDF
    # ln (pdf) = Binomial Coeff * product of probabilities
    # ln[f(x|n, p)] = comb(N,k)* num_heads*ln(pH) + (N-num_heads) * ln(1-pH)
    N = sum(obs);#number of trials
    k = obs[0] # number of heads
    binomial_coeff = math.factorial(N) / (math.factorial(N-k) *
        math.factorial(k))
    prod_probs = obs[0]*math.log(probs[0]) + obs[1]*math.log(1-probs[0])
    log_lik = binomial_coeff + prod_probs
    return log_lik
# 1st: Coin B, {HTTTHHTHTH}, 5H,5T
# 2nd: Coin A, {HHHHTHHHHH}, 9H,1T
# 3rd: Coin A, {HTHHHHHTHH}, 8H,2T
# 4th: Coin B, {HTHTTTHHTT}, 4H,6T
# 5th: Coin A, {THHHTHHHTH}, 7H,3T
# so, from MLE: pA(heads) = 0.80 and pB(heads)=0.45

data=[]
with open("cluster.csv") as tsv:
    for line in csv.reader(tsv):
        data=[int(i) for i in line]

```

```
# represent the experiments
head_counts = np.array(data)
tail_counts = 10-head_counts
experiments = list(zip(head_counts,tail_counts))

# initialise the pA(heads) and pB(heads)
pA_heads = np.zeros(100); pA_heads[0] = 0.60
pB_heads = np.zeros(100); pB_heads[0] = 0.50

# E-M begins!
delta = 0.001
j = 0 # iteration counter
improvement = float('inf')
while (improvement>delta):
    expectation_A = np.zeros((len(experiments),2), dtype=float)
    expectation_B = np.zeros((len(experiments),2), dtype=float)
    for i in range(0,len(experiments)):
        e = experiments[i] # i'th experiment
          # loglikelihood of e given coin A:
        ll_A =
          ↪ get_binomial_log_likelihood(e,np.array([pA_heads[j],1-pA_heads[j]]))
          # loglikelihood of e given coin B
        ll_B =
          ↪ get_binomial_log_likelihood(e,np.array([pB_heads[j],1-pB_heads[j]]))

          # corresponding weight of A proportional to likelihood of A , ex. .45
        weightA = math.exp(ll_A) / ( math.exp(ll_A) + math.exp(ll_B) )

          # corresponding weight of B proportional to likelihood of B , ex. .55
        weightB = math.exp(ll_B) / ( math.exp(ll_A) + math.exp(ll_B) )
                #multiply weightA * e .45xNo. of heads and 45xNo. of tails for coin A
        expectation_A[i] = np.dot(weightA, e)
        #multiply weightB * e .45xNo. of heads and 45xNo. of Tails for coin B
        expectation_B[i] = np.dot(weightB, e)

    #summing up the data no. of heads and tails for coin A
    pA_heads[j+1] = sum(expectation_A)[0] / sum(sum(expectation_A));
    #summing up the data no. of heads and tails for coin B
    pB_heads[j+1] = sum(expectation_B)[0] / sum(sum(expectation_B));

    #checking the improvement to maximise the accuracy.
```

```
    improvement = ( max( abs(np.array([pA_heads[j+1],pB_heads[j+1]]) -
                    np.array([pA_heads[j],pB_heads[j]]) )) )
    print(np.array([pA_heads[j+1],pB_heads[j+1]]) -
                    np.array([pA_heads[j],pB_heads[j]]) )
    j = j+1

plt.figure();
plt.plot(range(0,j),pA_heads[0:j])#for plotting the graph coin A
plt.plot(range(0,j),pB_heads[0:j])#for plotting the graph coin B
plt.show()
```

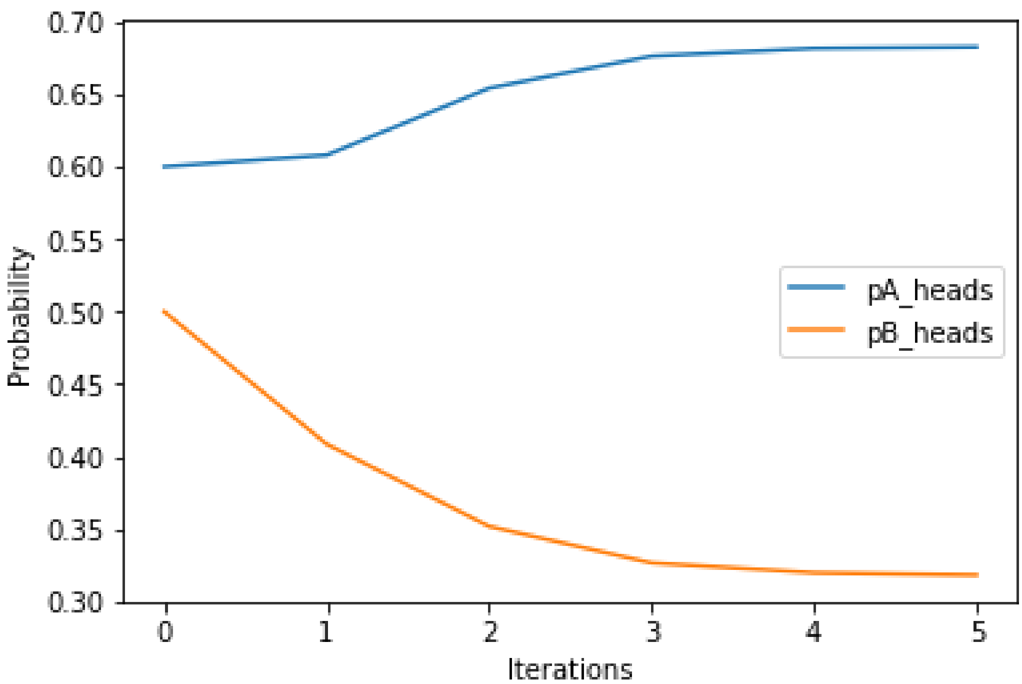
==

Data Set: cluster.csv

5	9	8	4	7	2	3	1	6	4	6

OUTPUT:

```
[ 0.00796672 -0.09125939]
[ 0.04620638 -0.05680878]
[ 0.02203957 -0.02519619]
[ 0.00533685 -0.00675812]
[ 0.00090446 -0.00162885]
[  6.34794565e-05  -4.42987679e-04]
```



k-Means algorithm

```
 # clustering dataset
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

data=[]
ydata=[]
with open("cluster.csv") as tsv:
    for line in csv.reader(tsv):
        data=[int(i) for i in line]
        ydata=[10-int(i) for i in line]

#np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x1 = np.array(data)
#np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
x2 = np.array(ydata)
print(x1)
plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)

plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 10])
    plt.ylim([0, 10])
```

```
plt.show()
```

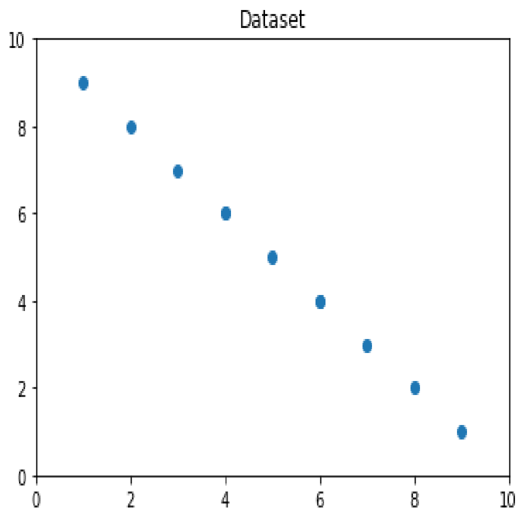
===

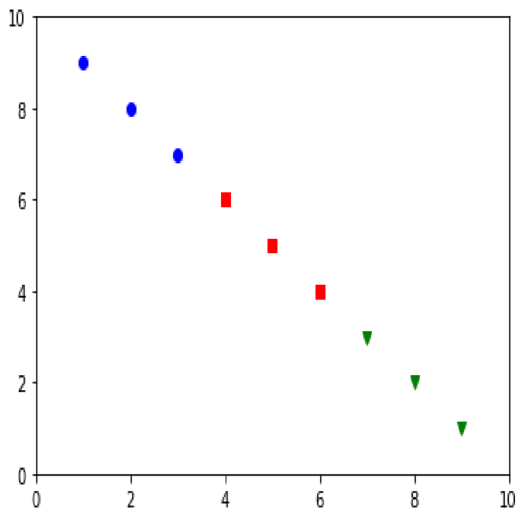
Data Set: cluster.csv

5	9	8	4	7	2	3	1	6	4	6

OUTPUT:

```
[5 9 8 4 7 2 3 1 6 4 6]
```





Experiment No. 9

k-Nearest Neighbour Algorithm

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.

```
import numpy as np
from sklearn import preprocessing,cross_validation,neighbors
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from matplotlib import style
#dataset=[[1,4,1],[2,4.5,1],[5,2,2],[2,'?',1],[3,6,1],
#[6,3,2],[5,2,2],[5,1,2],[3,6,1],[6,3,2]]

#labels=['height','paw','class']

df=pd.read_csv("irisdata.csv")

df.replace('setosa',1, inplace=True)
df.replace('versicolor',2, inplace=True)
df.replace('virginica',3, inplace=True)

#Missing Data Handling
df.replace('?',-9999,inplace=True)

#Define Attributes and Classes
X=np.array(df.drop(['species'],1))
Y=np.array(df['species'])

X_train,X_test,Y_train,Y_test=
    cross_validation.train_test_split(X,Y,test_size=0.2)

plt.plot(X_train,Y_train,'b.')

```

```python
# Define the classifier using panda library

clf=neighbors.KNeighborsClassifier()

# Save the model with the fit method
clf.fit(X_train,Y_train)

# use the test set and calculate the accuracy of the model
accuracy=clf.score(X_test, Y_test)

print("Accurancy:")
print(accuracy)

print("---------------------------------------------------")
example_measures = np.array([[4.7,3.2,2,0.2],[5.1,2.4,4.3,1.3]])
example_measures = example_measures.reshape(2,-1)

prediction = clf.predict(example_measures)

print("Prediction")
print(prediction)
```

==

Data Set: irisdata.csv

sepal$_{l}ength$	sepal$_{w}idth$	petal$_{l}ength$	petal$_{w}idth$	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa
5.8	4	1.2	0.2	setosa

5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5	3	1.6	0.2	setosa
5	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.1	1.5	0.1	setosa
4.4	3	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa
5	3.5	1.6	0.6	setosa
5.1	3.8	1.9	0.4	setosa
4.8	3	1.4	0.3	setosa
5.1	3.8	1.6	0.2	setosa
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5	3.3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor

6.3	3.3	4.7	1.6	versicolor
4.9	2.4	3.3	1	versicolor
6.6	2.9	4.6	1.3	versicolor
5.2	2.7	3.9	1.4	versicolor
5	2	3.5	1	versicolor
5.9	3	4.2	1.5	versicolor
6	2.2	4	1	versicolor
6.1	2.9	4.7	1.4	versicolor
5.6	2.9	3.6	1.3	versicolor
6.7	3.1	4.4	1.4	versicolor
5.6	3	4.5	1.5	versicolor
5.8	2.7	4.1	1	versicolor
6.2	2.2	4.5	1.5	versicolor
5.6	2.5	3.9	1.1	versicolor
5.9	3.2	4.8	1.8	versicolor
6.1	2.8	4	1.3	versicolor
6.3	2.5	4.9	1.5	versicolor
6.1	2.8	4.7	1.2	versicolor
6.4	2.9	4.3	1.3	versicolor
6.6	3	4.4	1.4	versicolor
6.8	2.8	4.8	1.4	versicolor
6.7	3	5	1.7	versicolor
6	2.9	4.5	1.5	versicolor
5.7	2.6	3.5	1	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1	versicolor
5.8	2.7	3.9	1.2	versicolor
6	2.7	5.1	1.6	versicolor
5.4	3	4.5	1.5	versicolor
6	3.4	4.5	1.6	versicolor
6.7	3.1	4.7	1.5	versicolor
6.3	2.3	4.4	1.3	versicolor
5.6	3	4.1	1.3	versicolor
5.5	2.5	4	1.3	versicolor
5.5	2.6	4.4	1.2	versicolor
6.1	3	4.6	1.4	versicolor
5.8	2.6	4	1.2	versicolor
5	2.3	3.3	1	versicolor
5.6	2.7	4.2	1.3	versicolor
5.7	3	4.2	1.2	versicolor
5.7	2.9	4.2	1.3	versicolor

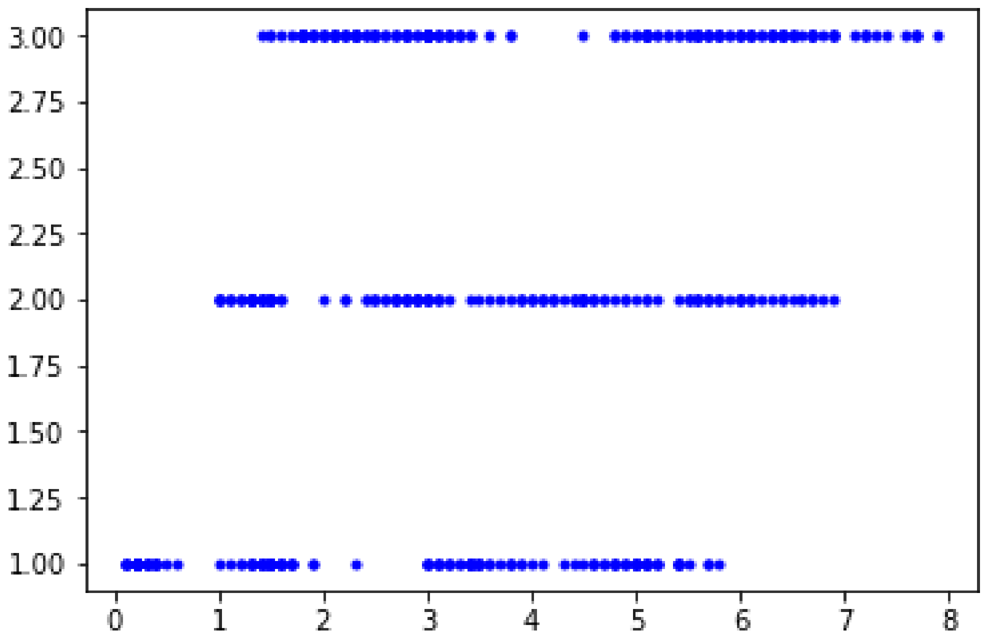
6.2	2.9	4.3	1.3	versicolor
5.1	2.5	3	1.1	versicolor
5.7	2.8	4.1	1.3	versicolor
6.3	3.3	6	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
6.5	3	5.8	2.2	virginica
7.6	3	6.6	2.1	virginica
4.9	2.5	4.5	1.7	virginica
7.3	2.9	6.3	1.8	virginica
6.7	2.5	5.8	1.8	virginica
7.2	3.6	6.1	2.5	virginica
6.5	3.2	5.1	2	virginica
6.4	2.7	5.3	1.9	virginica
6.8	3	5.5	2.1	virginica
5.7	2.5	5	2	virginica
5.8	2.8	5.1	2.4	virginica
6.4	3.2	5.3	2.3	virginica
6.5	3	5.5	1.8	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
6	2.2	5	1.5	virginica
6.9	3.2	5.7	2.3	virginica
5.6	2.8	4.9	2	virginica
7.7	2.8	6.7	2	virginica
6.3	2.7	4.9	1.8	virginica
6.7	3.3	5.7	2.1	virginica
7.2	3.2	6	1.8	virginica
6.2	2.8	4.8	1.8	virginica
6.1	3	4.9	1.8	virginica
6.4	2.8	5.6	2.1	virginica
7.2	3	5.8	1.6	virginica
7.4	2.8	6.1	1.9	virginica
7.9	3.8	6.4	2	virginica
6.4	2.8	5.6	2.2	virginica
6.3	2.8	5.1	1.5	virginica
6.1	2.6	5.6	1.4	virginica
7.7	3	6.1	2.3	virginica
6.3	3.4	5.6	2.4	virginica
6.4	3.1	5.5	1.8	virginica

6	3	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica
6.7	3.1	5.6	2.4	virginica
6.9	3.1	5.1	2.3	virginica
5.8	2.7	5.1	1.9	virginica
6.8	3.2	5.9	2.3	virginica
6.7	3.3	5.7	2.5	virginica
6.7	3	5.2	2.3	virginica
6.3	2.5	5	1.9	virginica
6.5	3	5.2	2	virginica
6.2	3.4	5.4	2.3	virginica
5.9	3	5.1	1.8	virginica

==

OUTPUT:

```
Accurancy:
0.966666666667
-------------------------------------------------
Prediction
[1 2]
```



Experiment No. 10

Locally Weighted Regression Algorithm

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

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
%matplotlib inline
import math
boston = load_boston()
features = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target,columns=['target'])
data = pd.concat([features,target],axis=1)
x = data['RM']

X1 = sorted(np.array(x/x.mean()))

#Extending the X dataset to obtain more samples
X=X1+[i+1 for i in X1]

#Applying sin to obtain a non-linear dataset
Y=np.sin(X)
plt.plot(X,Y)
n = int(0.8 * len(X))

x_train = X[:n]
y_train = Y[:n]

x_test = X[n:]
y_test = Y[n:]

w=np.exp([-(1.2-i)**2/(2*0.1) for i in x_train])
```

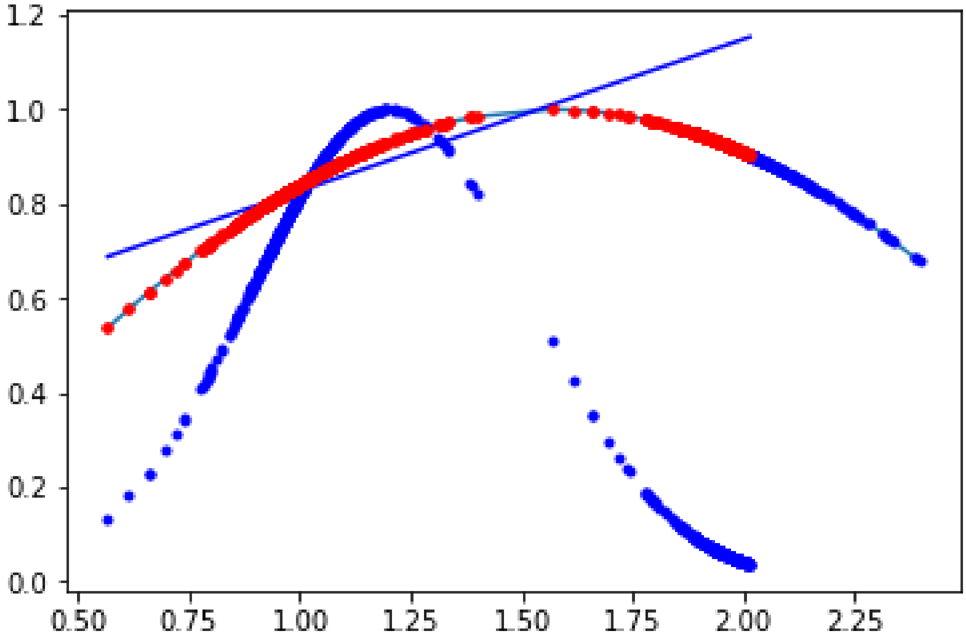
```

#print(w)

#print(x_train[:10])

plt.plot(x_train, y_train,'r.')

plt.plot(x_train,w,'b.')
def h(x,a,b):
    return a*x + b
    #cost function

def error(a,x,b,y,w):
    e = 0
    m = len(x)

    # Apply the weights multiplication for the cost function


    for i in range(m):
        e += np.power(h(x[i],a,b)-y[i],2)*w[i]

    return (1/(2*m)) * e

   #Calculating Gradient

def step_gradient(a,x,b,y,learning_rate,w):
    grad_a = 0
    grad_b = 0
    m = len(x)
    for i in range(m):
        grad_a += (2/m)*((h(x[i],a,b)-y[i])*x[i])*w[i]
        grad_b += (2/m)*(h(x[i],a,b)-y[i])*w[i]
    a = a - (grad_a * learning_rate)
    b = b - (grad_b * learning_rate)

    return a,b
    #Gradient Descent
def descend(initial_a, initial_b, x, y, learning_rate, iterations,w):
    a = initial_a
    b = initial_b
```

```
    for i in range(iterations):
        e = error(a,x,b,y,w)
        if i%1000 == 0:
            print(f"Error: {e}-- a:{a}, b:{b}")

        a, b = step_gradient(a,x,b,y, learning_rate,w)

    return a,b


a = 1.8600662368042573
b = -0.7962243178421666
learning_rate = 0.01
iterations = 10000
#Assign a Prediction Point 'P' for which we like to get the hypothesis
#p=1.0

final_a, final_b = descend(a,b,x_train,y_train, learning_rate, iterations,w)

#Calculate the final hypothesis
H=[i*final_a+final_b for i in x_train]

# Plot the Training Set and Final Hypothesis
plt.plot(x_train, y_train,'r.',x_train, H,'b')


print(error(a,x_test,b,y_test,w))
print(error(final_a,x_test, final_b,y_test,w))
plt.plot(x_test,y_test,'b.',x_train,y_train,'r.')
```

OUTPUT:

```
Error: 0.06614137226206705-- a:1.8600662368042573, b:-0.7962243178421666
Error: 0.01831248988715221-- a:1.3533605603913972, b:-0.6206735673234249
Error: 0.011422762970211432-- a:1.1032234861838637, b:-0.347590814908577
Error: 0.007176247674245229-- a:0.9068452261129998, b:-0.13319830250762849
Error: 0.004558888179990802-- a:0.7526720746347259, b:0.03511752470395558
Error: 0.0029456664570710407-- a:0.6316334187867452, b:0.16725934893398114
Error: 0.0019513497294632626-- a:0.536608078323685, b:0.2710015934995427
Error: 0.001338497980224941-- a:0.4620053386711435, b:0.3524478227325071
Error: 0.0009607639482851428-- a:0.4034360271954487, b:0.41638983867834906
Error: 0.0007279458172072266-- a:0.35745428091221954, b:0.4665896016596849
```

```
1.69309840122
0.0372197540025
```



Date Sets and Useful Links

Datasets

- https://archive.ics.uci.edu/ml/datasets.html
- https://github.com/awesomedata/awesome-public-datasets
- http://academictorrents.com/
- https://gengo.ai/articles/the-50-best-free-datasets-for-machine-learning/
- https://www.forbes.com/sites/bernardmarr/2018/02/26/big-data-and-ai

Machine Learning from Beginners

- kaggle.com/learn
- https://www.dataschool.io/machine-learning-with-scikit-learn/
- https://medium.com/learning-new-stuff/machine-learning-in-a-week

Machine Learning Intermediate

- https://developers.google.com/machine-learning/crash-course/

Deep Learning

- https://www.udemy.com/zero-to-deep-learning/
- https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/
- Fast.aipart1
- Fast.aipart2
- https://colab.research.google.com/drive/1pQ9QZ9V8bP99iRyv_oIkqTHRk3b9pC20

Papers for Knowledge Reading

- http://www.arxiv-sanity.com/
- https://github.com/dennybritz/deeplearning-papernotes
- https://medium.com/paper-club
- https://adeshpande3.github.io/adeshpande3.github.io/
- http://jmlr.org/papers/v15/delgado14a.html