ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

(18CSL76)

LAB MANUAL

For

VII SEMESTER

ALVA'S
Education Foundation®

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MIJAR MOODUBIDIRE

VISION OF THE INSTITUTE

"Transformative education by pursuing excellence in Engineering and Management through enhancing skills to meet the evolving needs of the community"

MISSION OF THE INSTITUTE

1. To bestow quality technical education to imbibe knowledge, creativity and ethos to students community.
2. To inculcate the best engineering practices through transformative education.
3. To develop a knowledgeable individual for a dynamic industrial scenario.
4. To inculcate research, entrepreneurial skills and human values in order to cater the needs of the society

VISION OF THE DEPARTMENT

"Engendering competent, excellent professionals by transforming the knowledge and computing skills to individuals through modern innovative tools and techniques"

MISSION OF THE DEPARTMENT

1. To produce skilled, creative software developers through rigorous training.
2. To conduct specific technical courses to keep abreast to the latest technological developments and transformations in the domain.
3. To implement the ideas of research and innovations in interdisciplinary domains.
4. To establish Industry-Institute Interaction programs to enhance the skills of employability and entrepreneurship.

General Lab Guidelines:

- Conduct yourself in a responsible manner at all times in the laboratory. Intentional misconduct will lead to the exclusion from the lab.
- Do not wander around, or distract other students, or interfere with the laboratory experiments of other students.
- Read the handout and procedures before starting the experiments. Follow all written and verbal instructions carefully. If you do not understand the procedures, ask the instructor or teaching assistant.
- Attendance in all the labs is mandatory, absence permitted only with prior permission from Class teacher.
- The workplace has to be tidy before, during and after the experiment.
- Do not eat food, drink beverages or chew gum in the laboratory.

DO'S:-

- Uniform and ID card are must.
- Strictly follow the procedures for conduction of experiments.
- Records have to be submitted every week for evaluation.
- Chairs and stools should be kept under the workbenches when not in use.
- After the lab session, switch off the systems and every supply, .
- Keep your belongings in designated area.
- Sign the log book when you enter/leave the laboratory.

DONT'S:-

- Don't touch open wires unless you are sure that there is no voltage. Always disconnect the plug by pulling on the connector body not by the cable. Switch off the supply while you make changes in connections of wires.
- Students are not allowed to work in laboratory alone or without presence of the teaching staff/ instructor.
- No additional material should be carried by the students during regular labs.
- Avoid stepping on electrical wires or any other computer cables.
- Without permission no downloads or installations

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY
18CSL76

(Effective from the academic year 2018 -2019)

SEMESTER – VII

SYLLABUS

Course Code	18CSL76	CIE Marks	40
Number of Contact Hours/Week	0:0:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03
Credits – 2			

1. Implement A* Search algorithm.
2. Implement AO* Search algorithm.
3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.
6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
8. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
9. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Laboratory Outcomes: The student should be able to:

- Implement and demonstrate AI and ML algorithms.
- Evaluate different algorithms.

Conduct of Practical Examination:

Experiment distribution

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

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

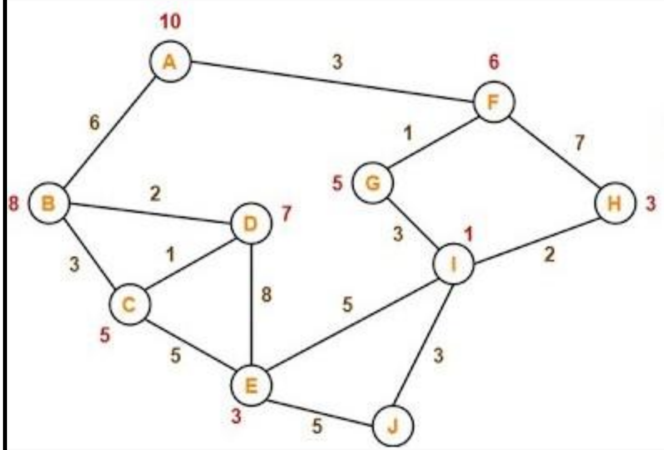
1. Implementation of A Star Search Algorithm

Program :

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}              #store distance from starting node
    parents = {}        # parents contains an adjacency map of all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                #for each node m,compare its distance from start i.e g(m) to the
                #from start through n node
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        g[m] = g[n] + weight
                        #change parent of m to n
                        parents[m] = n
                        #if m in closed set,remove and add to open
```

```
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None

        # if the current node is the stop_node
        # then we begin reconstruct in the path from it to the start_node
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

#define function to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
```

EXAMPLE 1

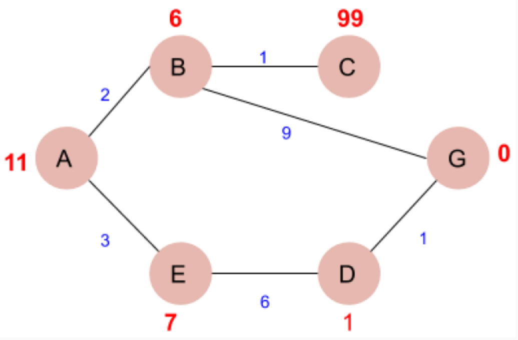



```
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]
#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
```

OUTPUT

Path found: ['A', 'F', 'G', 'I', 'J']

EXAMPLE 2




```
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('A', 2), ('C', 1), ('G', 9)],
    'C': [('B', 1)],
    'D': [('E', 6), ('G', 1)],
    'E': [('A', 3), ('D', 6)],
    'G': [('B', 9), ('D', 1)]
}
aStarAlgo('A', 'G')
```

Output:

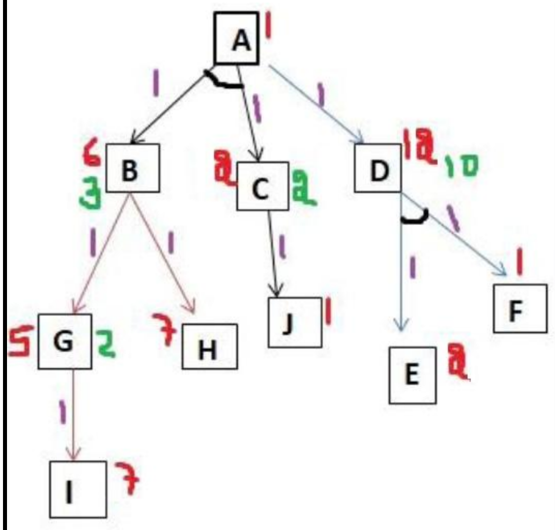
Path found: ['A', 'E', 'D', 'G']

2. Implementation of AO Star Search Algorithm

```
# Cost to find the AND and OR path
def calc_cost(H, condition, weight = 1):
        cost = {}
        if 'AND' in condition:
                AND_nodes = condition['AND']
                Path_A = ' AND '.join(AND_nodes)
                PathA = sum(H[node]+weight for node in AND_nodes)
                cost[Path_A] = PathA

        if 'OR' in condition:
                OR_nodes = condition['OR']
                Path_B =' OR '.join(OR_nodes)
                PathB = min(H[node]+weight for node in OR_nodes)
                cost[Path_B] = PathB
        return cost
# Update the cost
def update_cost(H, Conditions, weight=1):
        Main_nodes = list(Conditions.keys())
        Main_nodes.reverse()
        least_cost= {}
        for key in Main_nodes:
                condition = Conditions[key]
                print(key,':', Conditions[key],'>>>', calc_cost(H, condition, weight))
                c = calc_cost(H, condition, weight)
                H[key] = min(c.values())
                least_cost[key] = calc_cost(H, condition, weight)
        return least_cost
# Print the shortest path
def shortest_path(Start,Updated_cost, H):
        Path = Start
        if Start in Updated_cost.keys():
                Min_cost = min(Updated_cost[Start].values())
                key = list(Updated_cost[Start].keys())
                values = list(Updated_cost[Start].values())
                Index = values.index(Min_cost)

                # FIND MINIMIMUM PATH KEY
                Next = key[Index].split()
                # ADD TO PATH FOR OR PATH
                if len(Next) == 1:

                        Start =Next[0]
                        Path += '<--' +shortest_path(Start, Updated_cost, H)
                # ADD TO PATH FOR AND PATH
                else:
                        Path +='<--('+key[Index]+') '

                        Start = Next[0]
                        Path += '[' +shortest_path(Start, Updated_cost, H) + ' + '

                        Start = Next[-1]
                        Path += shortest_path(Start, Updated_cost, H) + ']'

        return Path
```

```
H = {'A': -1, 'B': 5, 'C': 2, 'D': 4, 'E': 7, 'F': 9, 'G': 3, 'H': 0, 'I':0, 'J':0}

Conditions = {
'A': {'OR': ['B'], 'AND': ['C', 'D']},
'B': {'OR': ['E', 'F']},
'C': {'OR': ['G'], 'AND': ['H', 'I']},
'D': {'OR': ['J']}
}
# weight
weight = 1
# Updated cost
print('Updated Cost :')
Updated_cost = update_cost(H, Conditions, weight=1)
print('*'*75)
print('Shortest Path :\n',shortest_path('A', Updated_cost,H))
```

#Graph – 1 as Input to AO Star Search Algorithm



#OUTPUT

```
Updated Cost :
D : {'OR': ['J']} >>> {'J': 1}
C : {'OR': ['G'], 'AND': ['H', 'I']} >>> {'H AND I': 2, 'G': 4}
B : {'OR': ['E', 'F']} >>> {'E OR F': 8}
A : {'OR': ['B'], 'AND': ['C', 'D']} >>> {'C AND D': 5, 'B': 9}
***************************************************************************
Shortest Path :
 A<--(C AND D) [C<--(H AND I) [H + I] + D<--J]
```

3. Implement Candidate Elimination Algorithm to get Consistent Version Space

Problem Statement: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Candidate-Elimination Algorithm:

1. Load data set
2. *G* <-maximally general hypotheses in H
3. S <- maximally specific hypotheses in H
4. For each training example d=<x,c(x)>Case 1 : If d is a positive example

Remove from G any hypothesis that is inconsistent with dFor each hypothesis s in S that is not consistent with d

- *Remove s from S.*
- *Add to S all minimal generalizations h of s such that*
  - *h consistent with d*
  - *Some member of G is more general than h*
- *Remove from S any hypothesis that is more general than another hypothesis in S*

Case 2: If d is a negative example

Remove from S any hypothesis that is inconsistent with dFor 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*
  - o *h consistent with d*
  - o *Some member of S is more specific than h*
- *Remove from G any hypothesis that is less general than another hypothesis in G*

```
import numpy as np
import pandas as pd

data = pd.read_csv(path+'/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
```

```
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Dataset:

EnjoySport Dataset is saved as .csv (comma separated values) file in the current working directory otherwise use the complete path of the dataset set in the program:

sky	airtemp	humidity	wind	water	forecast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

OUTPUT

Final Specific_h: [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]

Final General_h: [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’]]

4. Decision Tree ID3 Algorithm

Problem Statement: 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.

```
ID3(Examples, Target_attribute, Attributes)

Examples are the training examples.
Target_attribute 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.
Returns a decision tree that correctly classifies the given Examples.

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

Otherwise Begin
  A ← the attribute from Attributes that best* classifies Examples
  The decision attribute for Root ← A
  For each possible value, vi, of A,
    Add a new tree branch below Root, corresponding to the test A = vi
    Let Examples vi, be the subset of Examples that have value vi for A
    If Examples vi , is empty
      Then below this new branch add a leaf node with
      label = most common value of Target_attribute in Examples
    Else
      below this new branch add the subtree
      ID3(Examples vi, Targe_tattribute, Attributes – {A}))
End
Return Root
```

Import libraries and read data using read_csv() function. Remove the target from the data and store attributes in the features variable.

Program

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("Dataset/4-dataset.csv")
features = [feat for feat in data]
features.remove("answer")
```

Create a class named Node with four members children, value, isLeaf and pred.

```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

Define a function called entropy to find the entropy oof the dataset.

```
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
```

```
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

Define a function named info_gain to find the gain of the attribute

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
```

Define a function named ID3 to get the decision tree for the given dataset

```
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root
```

Define a function named printTree to draw the decision tree

```
def printTree(root: Node, depth=0):
```

```
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
```

Define a function named classify to classify the new example

```
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)
```

Finally, call the ID3, printTree and classify functions

```
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

Output:

```
Outlook
  rain
        Wind
                strong
                        no
                weak
                        yes
  overcast
        yes

  sunny
        Humidity
                normal
                        yes
                high
                        no
```

```
Decision Tree is:
```

```
outlook
	overcast ->  ['yes']

	rain
		wind
			strong ->  ['no']

			weak ->  ['yes']

	sunny
		humidity
			high ->  ['no']

			normal ->  ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny',
'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'}  is:
['yes']
```

5. Back propagation Algorithm

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

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #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=5 #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
    wh += X.T.dot(d_hiddenlayer) *lr

    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends----------\n")
```

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

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Output

```
———Epoch- 1 Starts———-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.81951208]
[0.8007242 ]
[0.82485744]]
———Epoch- 1 Ends———-

———Epoch- 2 Starts———-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
```

```
Predicted Output:
[[0.82033938]
[0.80153634]
[0.82568134]]
———–Epoch- 2 Ends———-

———–Epoch- 3 Starts———-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82115226]
[0.80233463]
[0.82649072]]
———–Epoch- 3 Ends———-

———–Epoch- 4 Starts———-
Input:
[[0.66666667 1. ]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82195108]
[0.80311943]
[0.82728598]]
———–Epoch- 4 Ends———-
```

6. Naïve Bayesian Classifier

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

Bayes' Theorem is stated as:

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

Where,

P(h|D) is the probability of hypothesis h given the data D. This is called the **posterior probability**.

P(D|h) is the probability of data d given that the hypothesis h was true.

P(h) is the probability of hypothesis h being true. This is called the **prior probability of h.**

P(D) is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is ***hMAP*** is a MAP hypothesis provided.

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution

Representation for Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real- valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Gaussian Naive Bayes Model from Data

The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5}$$

Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distribution

Examples:
The data set used in this program is the ***Pima Indians Diabetes problem***.
This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.

The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome

Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BP	Skin Thickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35		33.6	0.627	50	1
2	1	85	66	29		26.6	0.351	31	
3	8	183	64			23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	
5		137	40	35	168	43.1	2.288	33	1
6	5	116	74			25.6	0.201	30	
7	3	78	50	32	88	31	0.248	26	1
8	10	115				35.3	0.134	29	

9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96				0.232	54	1

Python Program to Implement and Demonstrate Naïve Bayesian Classifier Machine Learning

```
import csv
import random
import math

def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));
        dataset = list(lines)
        for i in range(len(dataset)):
    #converting strings into numbers for processing
                dataset[i] = [float(x) for x in dataset[i]]
        return dataset
 def splitdataset(dataset, splitratio):
   #67% training size
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]
 def separatebyclass(dataset):
        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
        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): #creates a dictionary of classes
		summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
		del summaries[-1] #excluding labels +ve or -ve
		return summaries
 def summarizebyclass(dataset):
		separated = separatebyclass(dataset);
    #print(separated)
		summaries = {}
		for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
				summaries[classvalue] = summarize(instances) #summarize is used to cal to
mean and std
		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 = {} # probabilities contains the all prob of all class of test data
		for classvalue, classsummaries in summaries.items():#class and attribute information as
mean and sd
				probabilities[classvalue] = 1
				for i in range(len(classsummaries)):
						mean, stdev = classsummaries[i] #take mean and sd of every attribute
for class 0 and 1 seperaely
						x = inputvector[i] #testvector's first attribute
						probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use
normal dist
		return probabilities

def predict(summaries, inputvector): #training and test data is passed
		probabilities = calculateclassprobabilities(summaries, inputvector)
		bestLabel, bestProb = None, -1
		for classvalue, probability in probabilities.items():#assigns that class which has he
highest prob
				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 = 'naivedata.csv'
            splitratio = 0.67
            dataset = loadcsv(filename);

            trainingset, testset = splitdataset(dataset, splitratio)
            print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))
            # prepare model
            summaries = summarizebyclass(trainingset);
            #print(summaries)
   # test model
            predictions = getpredictions(summaries, testset) #find the predictions of test data with
the training data
            accuracy = getaccuracy(testset, predictions)
            print('Accuracy of the classifier is : {0}%'.format(accuracy))
 main()
```

Output

Split 768 rows into train=514 and test=254
Rows Accuracy of the classifier is : 71.65354330708661%

7. Implement the K-Means and Estimation & Maximization Algorithm

Problem Statement: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the 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.

Python Program to Implement and Demonstrate K-Means and EM Algorithm Machine Learning

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("8-dataset.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output

The accuracy score of K-Mean: 0.24

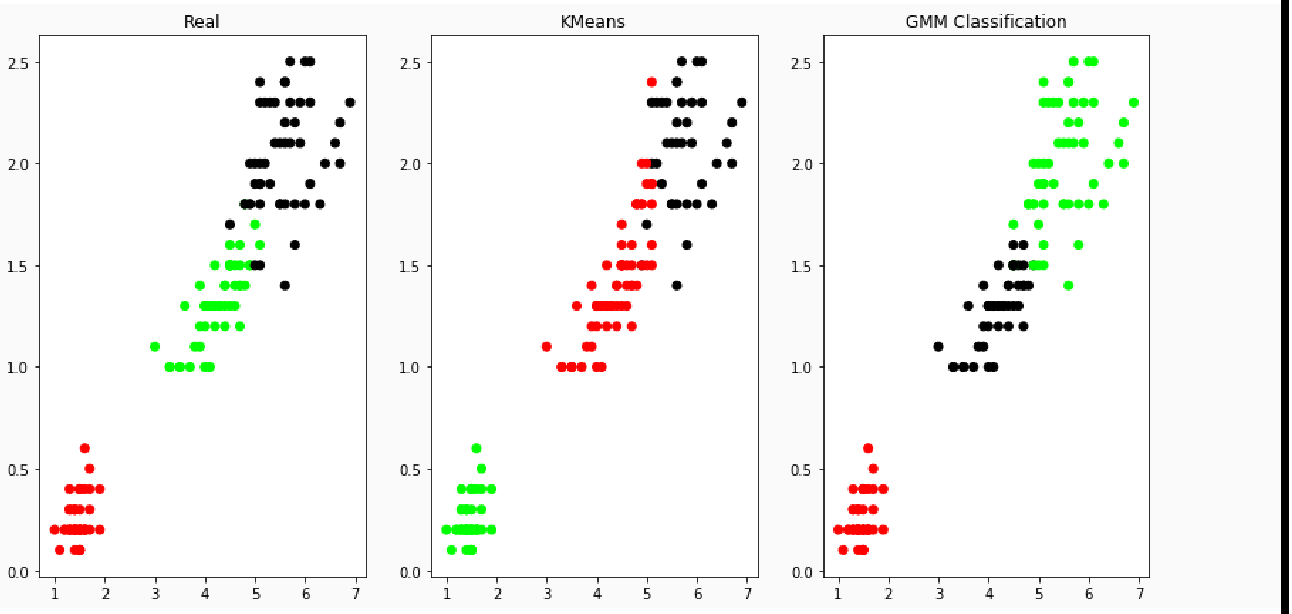
The Confusion matrixof K-Mean:

[[0 50 0]
[48 0 2]
[14 0 36]]

The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]
[0 5 45]
[0 50 0]]



The data set

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa
5.1	3.8	1.5	0.3	Iris-setosa
5.4	3.4	1.7	0.2	Iris-setosa
5.1	3.7	1.5	0.4	Iris-setosa
4.6	3.6	1	0.2	Iris-setosa
5.1	3.3	1.7	0.5	Iris-setosa
4.8	3.4	1.9	0.2	Iris-setosa
5	3	1.6	0.2	Iris-setosa
5	3.4	1.6	0.4	Iris-setosa
5.2	3.5	1.5	0.2	Iris-setosa
5.2	3.4	1.4	0.2	Iris-setosa
4.7	3.2	1.6	0.2	Iris-setosa
4.8	3.1	1.6	0.2	Iris-setosa
5.4	3.4	1.5	0.4	Iris-setosa
5.2	4.1	1.5	0.1	Iris-setosa
5.5	4.2	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5	3.2	1.2	0.2	Iris-setosa
5.5	3.5	1.3	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
4.4	3	1.3	0.2	Iris-setosa
5.1	3.4	1.5	0.2	Iris-setosa
5	3.5	1.3	0.3	Iris-setosa
4.5	2.3	1.3	0.3	Iris-setosa
4.4	3.2	1.3	0.2	Iris-setosa
5	3.5	1.6	0.6	Iris-setosa
5.1	3.8	1.9	0.4	Iris-setosa
4.8	3	1.4	0.3	Iris-setosa
5.1	3.8	1.6	0.2	Iris-setosa
4.6	3.2	1.4	0.2	Iris-setosa
5.3	3.7	1.5	0.2	Iris-setosa
5	3.3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
5.7	2.8	4.5	1.3	Iris-versicolor
6.3	3.3	4.7	1.6	Iris-versicolor

4.9	2.4	3.3	1	Iris-versicolor
6.6	2.9	4.6	1.3	Iris-versicolor
5.2	2.7	3.9	1.4	Iris-versicolor
5	2	3.5	1	Iris-versicolor
5.9	3	4.2	1.5	Iris-versicolor
6	2.2	4	1	Iris-versicolor
6.1	2.9	4.7	1.4	Iris-versicolor
5.6	2.9	3.6	1.3	Iris-versicolor
6.7	3.1	4.4	1.4	Iris-versicolor
5.6	3	4.5	1.5	Iris-versicolor
5.8	2.7	4.1	1	Iris-versicolor
6.2	2.2	4.5	1.5	Iris-versicolor
5.6	2.5	3.9	1.1	Iris-versicolor
5.9	3.2	4.8	1.8	Iris-versicolor
6.1	2.8	4	1.3	Iris-versicolor
6.3	2.5	4.9	1.5	Iris-versicolor
6.1	2.8	4.7	1.2	Iris-versicolor
6.4	2.9	4.3	1.3	Iris-versicolor
6.6	3	4.4	1.4	Iris-versicolor
6.8	2.8	4.8	1.4	Iris-versicolor
6.7	3	5	1.7	Iris-versicolor
6	2.9	4.5	1.5	Iris-versicolor
5.7	2.6	3.5	1	Iris-versicolor
5.5	2.4	3.8	1.1	Iris-versicolor
5.5	2.4	3.7	1	Iris-versicolor
5.8	2.7	3.9	1.2	Iris-versicolor
6	2.7	5.1	1.6	Iris-versicolor
5.4	3	4.5	1.5	Iris-versicolor
6	3.4	4.5	1.6	Iris-versicolor
6.7	3.1	4.7	1.5	Iris-versicolor
6.3	2.3	4.4	1.3	Iris-versicolor
5.6	3	4.1	1.3	Iris-versicolor
5.5	2.5	4	1.3	Iris-versicolor
5.5	2.6	4.4	1.2	Iris-versicolor
6.1	3	4.6	1.4	Iris-versicolor
5.8	2.6	4	1.2	Iris-versicolor
5	2.3	3.3	1	Iris-versicolor
5.6	2.7	4.2	1.3	Iris-versicolor
5.7	3	4.2	1.2	Iris-versicolor
5.7	2.9	4.2	1.3	Iris-versicolor
6.2	2.9	4.3	1.3	Iris-versicolor
5.1	2.5	3	1.1	Iris-versicolor
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica
7.6	3	6.6	2.1	Iris-virginica
4.9	2.5	4.5	1.7	Iris-virginica
7.3	2.9	6.3	1.8	Iris-virginica
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3	5.5	2.1	Iris-virginica
5.7	2.5	5	2	Iris-virginica
5.8	2.8	5.1	2.4	Iris-virginica
6.4	3.2	5.3	2.3	Iris-virginica
6.5	3	5.5	1.8	Iris-virginica

7.7	3.8	6.7	2.2	Iris-virginica
7.7	2.6	6.9	2.3	Iris-virginica
6	2.2	5	1.5	Iris-virginica
6.9	3.2	5.7	2.3	Iris-virginica
5.6	2.8	4.9	2	Iris-virginica
7.7	2.8	6.7	2	Iris-virginica
6.3	2.7	4.9	1.8	Iris-virginica
6.7	3.3	5.7	2.1	Iris-virginica
7.2	3.2	6	1.8	Iris-virginica
6.2	2.8	4.8	1.8	Iris-virginica
6.1	3	4.9	1.8	Iris-virginica
6.4	2.8	5.6	2.1	Iris-virginica
7.2	3	5.8	1.6	Iris-virginica
7.4	2.8	6.1	1.9	Iris-virginica
7.9	3.8	6.4	2	Iris-virginica
6.4	2.8	5.6	2.2	Iris-virginica
6.3	2.8	5.1	1.5	Iris-virginica
6.1	2.6	5.6	1.4	Iris-virginica
7.7	3	6.1	2.3	Iris-virginica
6.3	3.4	5.6	2.4	Iris-virginica
6.4	3.1	5.5	1.8	Iris-virginica
6	3	4.8	1.8	Iris-virginica
6.9	3.1	5.4	2.1	Iris-virginica
6.7	3.1	5.6	2.4	Iris-virginica
6.9	3.1	5.1	2.3	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.8	3.2	5.9	2.3	Iris-virginica
6.7	3.3	5.7	2.5	Iris-virginica
6.7	3	5.2	2.3	Iris-virginica
6.3	2.5	5	1.9	Iris-virginica
6.5	3	5.2	2	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3	5.1	1.8	Iris-virginica

8. k-Nearest Neighbour Algorithm

Problem Statement: Write a program to implement the 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

Python Program to Implement the k-Nearest Neighbour Algorithm

K-Nearest Neighbor Algorithm

Training algorithm:

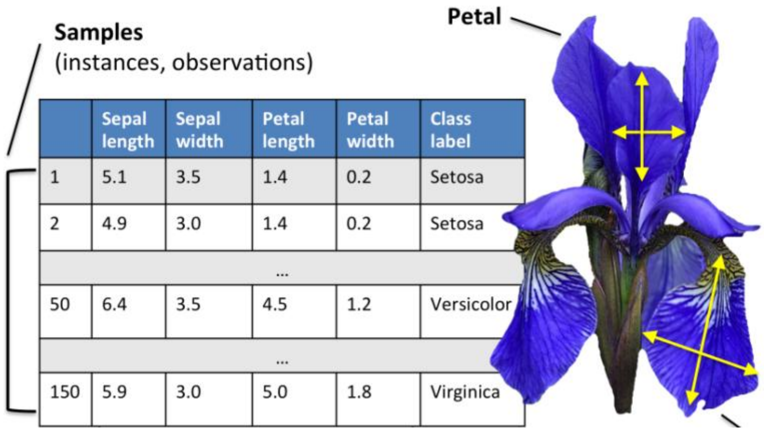
- For each training example (x, f (x)), add the example to the list training examples
  Classification algorithm:
  - Given a query instance x_q to be classified,
    - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
    - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class.



Sample Data

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

****Use the same data set as previous one**

Program

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("9-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

Output

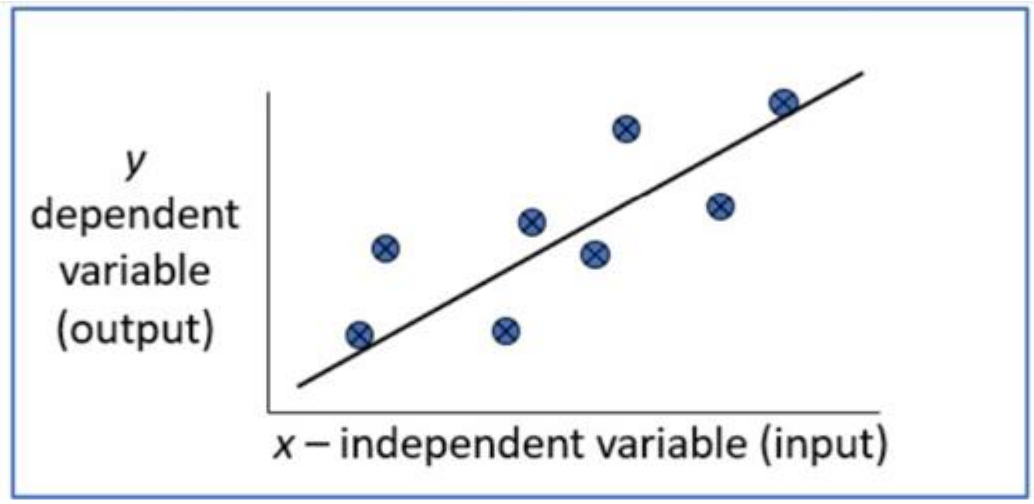
```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
-------------------------------------------------------------------------
Iris-versicolor          Iris-versicolor          Correct
Iris-virginica           Iris-versicolor          Wrong
Iris-virginica           Iris-virginica           Correct
Iris-versicolor          Iris-versicolor          Correct
Iris-setosa              Iris-setosa              Correct
Iris-versicolor          Iris-versicolor          Correct
Iris-setosa              Iris-setosa              Correct
Iris-setosa              Iris-setosa              Correct
Iris-virginica           Iris-virginica           Correct
Iris-virginica           Iris-versicolor          Wrong
Iris-virginica           Iris-virginica           Correct
Iris-setosa              Iris-setosa              Correct
Iris-virginica           Iris-virginica           Correct
Iris-virginica           Iris-virginica           Correct
Iris-versicolor          Iris-versicolor          Correct
-------------------------------------------------------------------------

Confusion Matrix:
 [[4 0 0]
 [0 4 0]
 [0 2 5]]
-------------------------------------------------------------------------

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         4
Iris-versicolor       0.67      1.00      0.80         4
 Iris-virginica       1.00      0.71      0.83         7

    avg / total       0.91      0.87      0.87        15

-------------------------------------------------------------------------
Accuracy of the classifier is 0.87
-------------------------------------------------------------------------
```

9. Locally Weighted Regression Algorithm

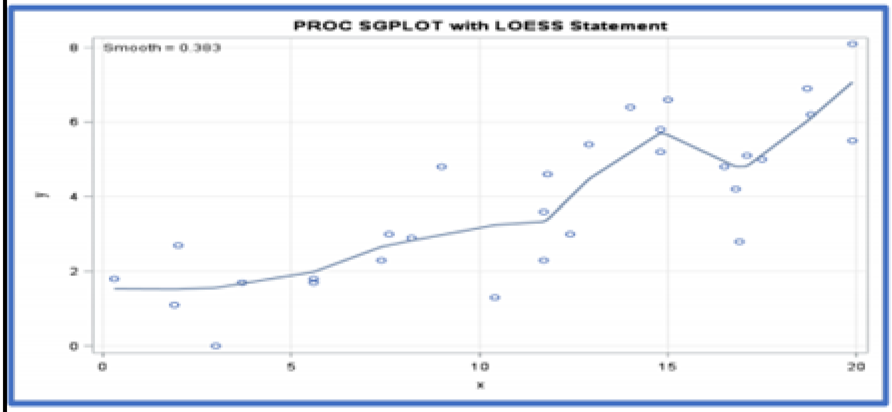
Problem Statement: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

Regression:

- Regression is a technique from statistics that are used to predict values of the desired target quantity when the target quantity is continuous.
    - In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
        - y is called the dependent variable.
        - x is called the independent variable.



Loess/Lowess Regression:
Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:
Locally weighted regression is a very powerful nonparametric model used in statistical learning.

Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
The weights are given by a kernel function (k or w) which can be chosen arbitrarily

Algorithm

1. Read the Given data Sample to X and the curve (linear or nonlinear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using:

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

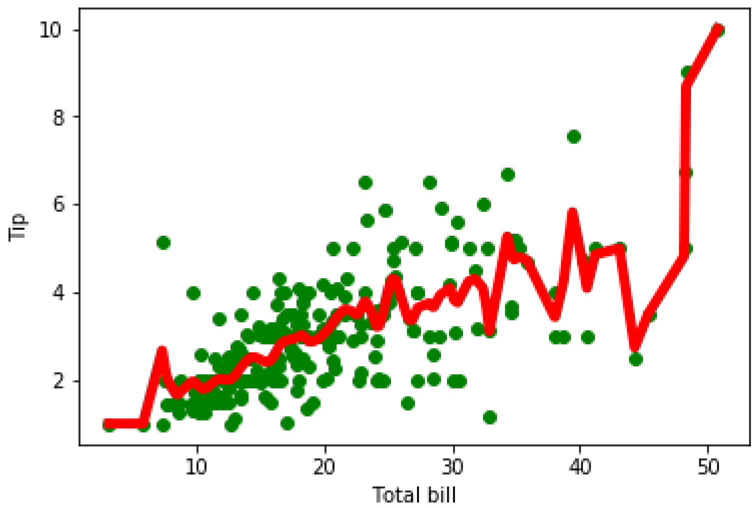
6. Prediction = x0*β

Program

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

def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np1.eye((m)))
    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

# load data points
data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np1.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
```

```
SortIndex = X[:,1].argsort(0)
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();
```

Output



Data Set:

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
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3

20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2
20.29	2.75	Female	No	Sat	Dinner	2
15.77	2.23	Female	No	Sat	Dinner	2
39.42	7.58	Male	No	Sat	Dinner	4
19.82	3.18	Male	No	Sat	Dinner	2
17.81	2.34	Male	No	Sat	Dinner	4
13.37	2	Male	No	Sat	Dinner	2
12.69	2	Male	No	Sat	Dinner	2
21.7	4.3	Male	No	Sat	Dinner	2
19.65	3	Female	No	Sat	Dinner	2
9.55	1.45	Male	No	Sat	Dinner	2
18.35	2.5	Male	No	Sat	Dinner	4
15.06	3	Female	No	Sat	Dinner	2
20.69	2.45	Female	No	Sat	Dinner	4
17.78	3.27	Male	No	Sat	Dinner	2
24.06	3.6	Male	No	Sat	Dinner	3
16.31	2	Male	No	Sat	Dinner	3
16.93	3.07	Female	No	Sat	Dinner	3
18.69	2.31	Male	No	Sat	Dinner	3
31.27	5	Male	No	Sat	Dinner	3
16.04	2.24	Male	No	Sat	Dinner	3
17.46	2.54	Male	No	Sun	Dinner	2
13.94	3.06	Male	No	Sun	Dinner	2
9.68	1.32	Male	No	Sun	Dinner	2
30.4	5.6	Male	No	Sun	Dinner	4
18.29	3	Male	No	Sun	Dinner	2
22.23	5	Male	No	Sun	Dinner	2
32.4	6	Male	No	Sun	Dinner	4
28.55	2.05	Male	No	Sun	Dinner	3
18.04	3	Male	No	Sun	Dinner	2
12.54	2.5	Male	No	Sun	Dinner	2
10.29	2.6	Female	No	Sun	Dinner	2
34.81	5.2	Female	No	Sun	Dinner	4
9.94	1.56	Male	No	Sun	Dinner	2
25.56	4.34	Male	No	Sun	Dinner	4
19.49	3.51	Male	No	Sun	Dinner	2
38.01	3	Male	Yes	Sat	Dinner	4
26.41	1.5	Female	No	Sat	Dinner	2
11.24	1.76	Male	Yes	Sat	Dinner	2
48.27	6.73	Male	No	Sat	Dinner	4
20.29	3.21	Male	Yes	Sat	Dinner	2
13.81	2	Male	Yes	Sat	Dinner	2
11.02	1.98	Male	Yes	Sat	Dinner	2
18.29	3.76	Male	Yes	Sat	Dinner	4
17.59	2.64	Male	No	Sat	Dinner	3
20.08	3.15	Male	No	Sat	Dinner	3
16.45	2.47	Female	No	Sat	Dinner	2

3.07	1	Female	Yes	Sat	Dinner	1
20.23	2.01	Male	No	Sat	Dinner	2
15.01	2.09	Male	Yes	Sat	Dinner	2
12.02	1.97	Male	No	Sat	Dinner	2
17.07	3	Female	No	Sat	Dinner	3
26.86	3.14	Female	Yes	Sat	Dinner	2
25.28	5	Female	Yes	Sat	Dinner	2
14.73	2.2	Female	No	Sat	Dinner	2
10.51	1.25	Male	No	Sat	Dinner	2
17.92	3.08	Male	Yes	Sat	Dinner	2
27.2	4	Male	No	Thur	Lunch	4
22.76	3	Male	No	Thur	Lunch	2
17.29	2.71	Male	No	Thur	Lunch	2
19.44	3	Male	Yes	Thur	Lunch	2
16.66	3.4	Male	No	Thur	Lunch	2
10.07	1.83	Female	No	Thur	Lunch	1
32.68	5	Male	Yes	Thur	Lunch	2
15.98	2.03	Male	No	Thur	Lunch	2
34.83	5.17	Female	No	Thur	Lunch	4
13.03	2	Male	No	Thur	Lunch	2
18.28	4	Male	No	Thur	Lunch	2
24.71	5.85	Male	No	Thur	Lunch	2
21.16	3	Male	No	Thur	Lunch	2
28.97	3	Male	Yes	Fri	Dinner	2
22.49	3.5	Male	No	Fri	Dinner	2
5.75	1	Female	Yes	Fri	Dinner	2
16.32	4.3	Female	Yes	Fri	Dinner	2
22.75	3.25	Female	No	Fri	Dinner	2
40.17	4.73	Male	Yes	Fri	Dinner	4
27.28	4	Male	Yes	Fri	Dinner	2
12.03	1.5	Male	Yes	Fri	Dinner	2
21.01	3	Male	Yes	Fri	Dinner	2
12.46	1.5	Male	No	Fri	Dinner	2
11.35	2.5	Female	Yes	Fri	Dinner	2
15.38	3	Female	Yes	Fri	Dinner	2
44.3	2.5	Female	Yes	Sat	Dinner	3
22.42	3.48	Female	Yes	Sat	Dinner	2
20.92	4.08	Female	No	Sat	Dinner	2
15.36	1.64	Male	Yes	Sat	Dinner	2
20.49	4.06	Male	Yes	Sat	Dinner	2
25.21	4.29	Male	Yes	Sat	Dinner	2
18.24	3.76	Male	No	Sat	Dinner	2
14.31	4	Female	Yes	Sat	Dinner	2
14	3	Male	No	Sat	Dinner	2
7.25	1	Female	No	Sat	Dinner	1
38.07	4	Male	No	Sun	Dinner	3
23.95	2.55	Male	No	Sun	Dinner	2
25.71	4	Female	No	Sun	Dinner	3

17.31	3.5	Female	No	Sun	Dinner	2
29.93	5.07	Male	No	Sun	Dinner	4
10.65	1.5	Female	No	Thur	Lunch	2
12.43	1.8	Female	No	Thur	Lunch	2
24.08	2.92	Female	No	Thur	Lunch	4
11.69	2.31	Male	No	Thur	Lunch	2
13.42	1.68	Female	No	Thur	Lunch	2
14.26	2.5	Male	No	Thur	Lunch	2
15.95	2	Male	No	Thur	Lunch	2
12.48	2.52	Female	No	Thur	Lunch	2
29.8	4.2	Female	No	Thur	Lunch	6
8.52	1.48	Male	No	Thur	Lunch	2
14.52	2	Female	No	Thur	Lunch	2
11.38	2	Female	No	Thur	Lunch	2
22.82	2.18	Male	No	Thur	Lunch	3
19.08	1.5	Male	No	Thur	Lunch	2
20.27	2.83	Female	No	Thur	Lunch	2
11.17	1.5	Female	No	Thur	Lunch	2
12.26	2	Female	No	Thur	Lunch	2
18.26	3.25	Female	No	Thur	Lunch	2
8.51	1.25	Female	No	Thur	Lunch	2
10.33	2	Female	No	Thur	Lunch	2
14.15	2	Female	No	Thur	Lunch	2
16	2	Male	Yes	Thur	Lunch	2
13.16	2.75	Female	No	Thur	Lunch	2
17.47	3.5	Female	No	Thur	Lunch	2
34.3	6.7	Male	No	Thur	Lunch	6
41.19	5	Male	No	Thur	Lunch	5
27.05	5	Female	No	Thur	Lunch	6
16.43	2.3	Female	No	Thur	Lunch	2
8.35	1.5	Female	No	Thur	Lunch	2
18.64	1.36	Female	No	Thur	Lunch	3
11.87	1.63	Female	No	Thur	Lunch	2
9.78	1.73	Male	No	Thur	Lunch	2
7.51	2	Male	No	Thur	Lunch	2
14.07	2.5	Male	No	Sun	Dinner	2
13.13	2	Male	No	Sun	Dinner	2
17.26	2.74	Male	No	Sun	Dinner	3
24.55	2	Male	No	Sun	Dinner	4
19.77	2	Male	No	Sun	Dinner	4
29.85	5.14	Female	No	Sun	Dinner	5
48.17	5	Male	No	Sun	Dinner	6
25	3.75	Female	No	Sun	Dinner	4
13.39	2.61	Female	No	Sun	Dinner	2
16.49	2	Male	No	Sun	Dinner	4
21.5	3.5	Male	No	Sun	Dinner	4
12.66	2.5	Male	No	Sun	Dinner	2
16.21	2	Female	No	Sun	Dinner	3

13.81	2	Male	No	Sun	Dinner	2
17.51	3	Female	Yes	Sun	Dinner	2
24.52	3.48	Male	No	Sun	Dinner	3
20.76	2.24	Male	No	Sun	Dinner	2
31.71	4.5	Male	No	Sun	Dinner	4
10.59	1.61	Female	Yes	Sat	Dinner	2
10.63	2	Female	Yes	Sat	Dinner	2
50.81	10	Male	Yes	Sat	Dinner	3
15.81	3.16	Male	Yes	Sat	Dinner	2
7.25	5.15	Male	Yes	Sun	Dinner	2
31.85	3.18	Male	Yes	Sun	Dinner	2
16.82	4	Male	Yes	Sun	Dinner	2
32.9	3.11	Male	Yes	Sun	Dinner	2
17.89	2	Male	Yes	Sun	Dinner	2
14.48	2	Male	Yes	Sun	Dinner	2
9.6	4	Female	Yes	Sun	Dinner	2
34.63	3.55	Male	Yes	Sun	Dinner	2
34.65	3.68	Male	Yes	Sun	Dinner	4
23.33	5.65	Male	Yes	Sun	Dinner	2
45.35	3.5	Male	Yes	Sun	Dinner	3
23.17	6.5	Male	Yes	Sun	Dinner	4
40.55	3	Male	Yes	Sun	Dinner	2
20.69	5	Male	No	Sun	Dinner	5
20.9	3.5	Female	Yes	Sun	Dinner	3
30.46	2	Male	Yes	Sun	Dinner	5
18.15	3.5	Female	Yes	Sun	Dinner	3
23.1	4	Male	Yes	Sun	Dinner	3
15.69	1.5	Male	Yes	Sun	Dinner	2
19.81	4.19	Female	Yes	Thur	Lunch	2
28.44	2.56	Male	Yes	Thur	Lunch	2
15.48	2.02	Male	Yes	Thur	Lunch	2
16.58	4	Male	Yes	Thur	Lunch	2
7.56	1.44	Male	No	Thur	Lunch	2
10.34	2	Male	Yes	Thur	Lunch	2
43.11	5	Female	Yes	Thur	Lunch	4
13	2	Female	Yes	Thur	Lunch	2
13.51	2	Male	Yes	Thur	Lunch	2
18.71	4	Male	Yes	Thur	Lunch	3
12.74	2.01	Female	Yes	Thur	Lunch	2
13	2	Female	Yes	Thur	Lunch	2
16.4	2.5	Female	Yes	Thur	Lunch	2
20.53	4	Male	Yes	Thur	Lunch	4
16.47	3.23	Female	Yes	Thur	Lunch	3
26.59	3.41	Male	Yes	Sat	Dinner	3
38.73	3	Male	Yes	Sat	Dinner	4
24.27	2.03	Male	Yes	Sat	Dinner	2
12.76	2.23	Female	Yes	Sat	Dinner	2
30.06	2	Male	Yes	Sat	Dinner	3

25.89	5.16	Male	Yes	Sat	Dinner	4
48.33	9	Male	No	Sat	Dinner	4
13.27	2.5	Female	Yes	Sat	Dinner	2
28.17	6.5	Female	Yes	Sat	Dinner	3
12.9	1.1	Female	Yes	Sat	Dinner	2
28.15	3	Male	Yes	Sat	Dinner	5
11.59	1.5	Male	Yes	Sat	Dinner	2
7.74	1.44	Male	Yes	Sat	Dinner	2
30.14	3.09	Female	Yes	Sat	Dinner	4
12.16	2.2	Male	Yes	Fri	Lunch	2
13.42	3.48	Female	Yes	Fri	Lunch	2
8.58	1.92	Male	Yes	Fri	Lunch	1
15.98	3	Female	No	Fri	Lunch	3
13.42	1.58	Male	Yes	Fri	Lunch	2
16.27	2.5	Female	Yes	Fri	Lunch	2
10.09	2	Female	Yes	Fri	Lunch	2
20.45	3	Male	No	Sat	Dinner	4
13.28	2.72	Male	No	Sat	Dinner	2
22.12	2.88	Female	Yes	Sat	Dinner	2
24.01	2	Male	Yes	Sat	Dinner	4
15.69	3	Male	Yes	Sat	Dinner	3
11.61	3.39	Male	No	Sat	Dinner	2
10.77	1.47	Male	No	Sat	Dinner	2
15.53	3	Male	Yes	Sat	Dinner	2
10.07	1.25	Male	No	Sat	Dinner	2
12.6	1	Male	Yes	Sat	Dinner	2
32.83	1.17	Male	Yes	Sat	Dinner	2
35.83	4.67	Female	No	Sat	Dinner	3
29.03	5.92	Male	No	Sat	Dinner	3
27.18	2	Female	Yes	Sat	Dinner	2
22.67	2	Male	Yes	Sat	Dinner	2
17.82	1.75	Male	No	Sat	Dinner	2
18.78	3	Female	No	Thur	Dinner	2

**** Students have to work on different data sets to improve their knowledge**