MACHINE LEARNING LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

SEMESTER – VII **Subject Code 15CSL76**

Description (If any):

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

Lab Experiments:

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
4. Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets.
5. Write a program to implement the 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.

Program 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.

Find-S **Algorithm**

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

     If the constraint a_i in h is satisfied by x

     Then do nothing

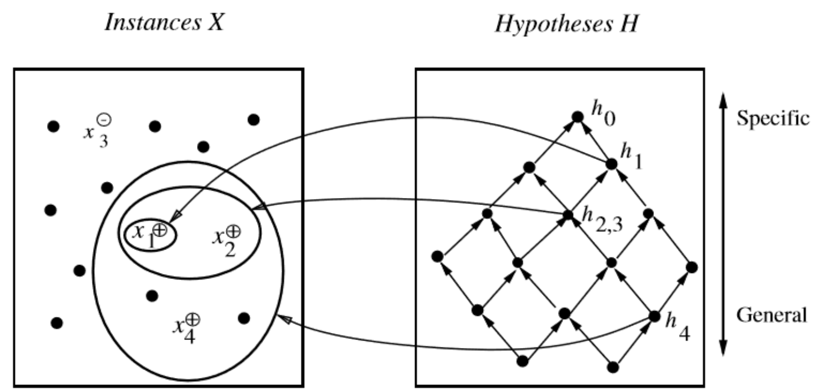
     Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Training Examples for EnjoySport

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
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

What is the general concept?

Hypothesis Space Search by Find-S



h_0 = <∅, ∅, ∅, ∅, ∅, ∅>

x_1 = <*Sunny Warm Normal Strong Warm Same*>, +	h_1 = <*Sunny Warm Normal Strong Warm Sam*
x_2 = <*Sunny Warm High Strong Warm Same*>, +	h_2 = <*Sunny Warm ? Strong Warm Same*>
x_3 = <*Rainy Cold High Strong Warm Change*>, -	h_3 = <*Sunny Warm ? Strong Warm Same*>
x_4 = <*Sunny Warm High Strong Cool Change*>, +	h_4 = <*Sunny Warm ? Strong ? ?* >

Find-s.py

```python
import csv
hypo=['%','%','%','%','%','%'];
with open('Training_examples.csv') as csv_file:
    readcsv = csv.reader(csv_file, delimiter=',')
    print(readcsv)
    data = []
    print("\nThe given training examples are:")
    for row in readcsv:
        print(row)
        if row[len(row)-1].upper() == "YES":
            data.append(row)
print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");

TotalExamples = len(data);
i=0;
j=0;
k=0;
print("The steps of the Find-s algorithm are\n",hypo);
list = [];
p=0;
d=len(data[p])-1;
for j in range(d):
    list.append(data[i][j]);
hypo=list;
i=1;
for i in range(TotalExamples):
    for k in range(d):
        if hypo[k]!=data[i][k]:
            hypo[k]='?';
            k=k+1;

        else:
            hypo[k];
    print(hypo);
i=i+1;
print("\nThe maximally specific Find-s hypothesis for the given training examples is");
list=[];
for i in range(d):
    list.append(hypo[i]);
print(list);
```

Input

	A	B	C	D	E	F	G
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes
5							

Output

```
The given training examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The positive examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']


The steps of the Find-s algorithm are
 ['%', '%', '%', '%', '%', '%']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']

The maximally specific Find-s hypothesis for the given training examples is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Program 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.

$G \leftarrow$ maximally general hypotheses in H
S $\leftarrow$ maximally specific hypotheses in H
For each training example d=<x,c(x)>
Case 1 : If d is a positive example

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

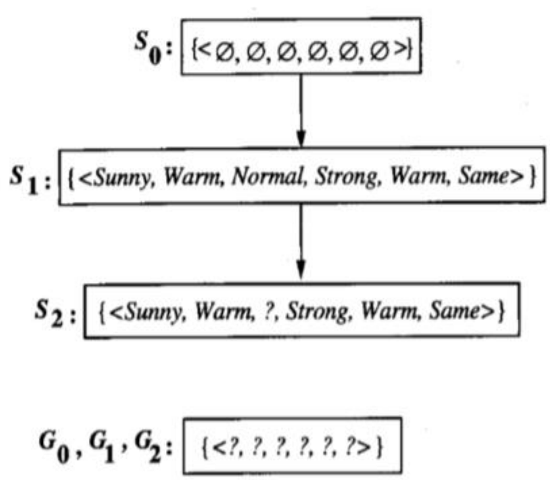
- *Remove s from S.*
- *Add to S all minimal generalizations h of s such that*
    - *h consistent with d*
    - *Some member of G is more general than h*
- *Remove from S any hypothesis that is more general than another hypothesis in S*

Case 2: If d is a negative example

Remove from S any hypothesis that is inconsistent with d
For each hypothesis g in G that is not consistent with d

- *Remove g from G.*
- *Add to G all minimal specializations h of g such that*
    - 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*

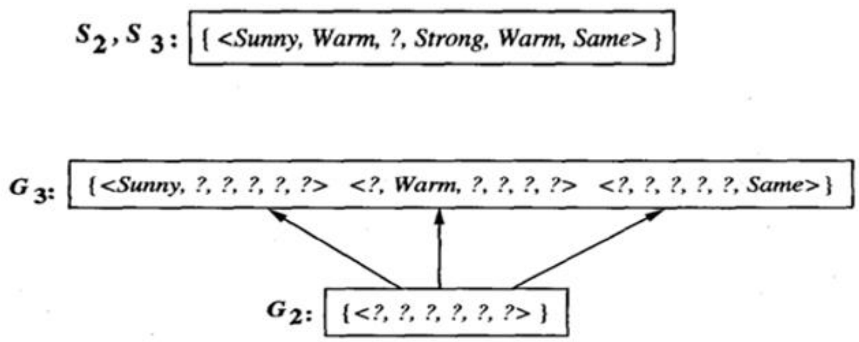
Iteration - 1



Training examples:

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

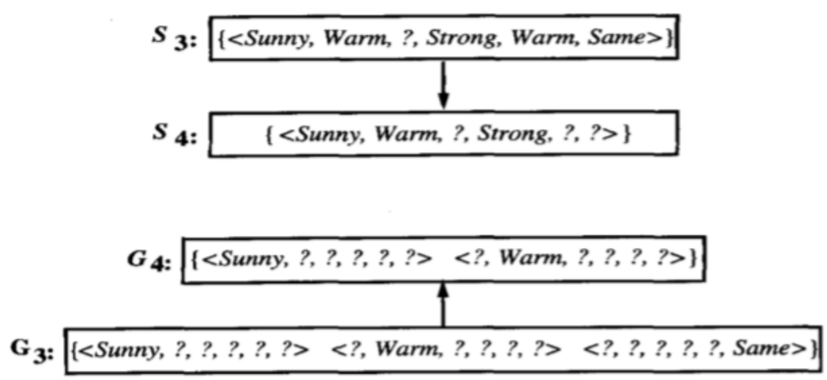
Iteration – 2



Training Example:

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

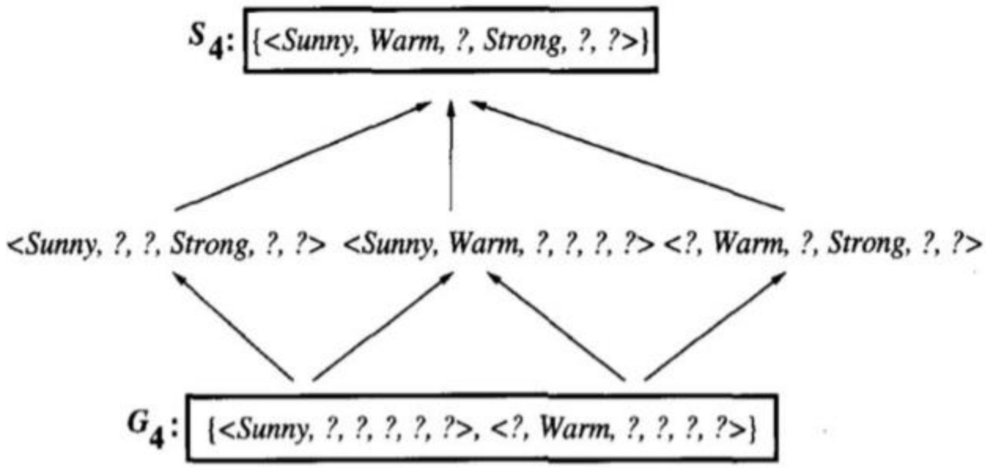
Iteration – 3



Training Example:

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

Final Version Space



```python
import csv
def g_0(n):
    return ("?",)*n
def s_0(n):
    return ('ϕ',)*n
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "ϕ"
                          and (x == y or y == "ϕ"))
        more_general_parts.append(mg)
    return all(more_general_parts)
def fulfills(example, hypothesis):
    ### the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != 'ϕ' else x[i]
    return [tuple(h_new)]
def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != "ϕ":
            h_new = h[:i] + ('ϕ',) + h[i+1:]
            results.append(h_new)
    return results
with open('training.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]
```

```python
def get_domains(examples):
    d = [set() for i in examples[0]]
    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]
get_domains(examples)
def candidate_elimination(examples):
    """
   :rtype: object
    """
    domains = get_domains(examples)[:-1]

    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])
    i = 0
    print("\n G[{0}]:".format(i), G)
    print("\n S[{0}]:".format(i), S)
    for xcx in examples:
        i = i + 1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions
        if cx == 'Y':  # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else:  # x is negative example
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
        print("\n G[{0}]:".format(i), G)
        print("\n S[{0}]:".format(i), S)
    return
```

```python
def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                               for g in G])])
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if
                                 any([more_general(h, h1)
                                      for h1 in S if h != h1])])
    return S
def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        #if g not in G:
           # continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                                for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if
                                 any([more_general(g1, h)
                                      for g1 in G if h != g1])])
    return G
candidate_elimination(examples)
```

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

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

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

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}

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

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[3]: {('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?',
'?', 'same')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[4]: {('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}
```

Program 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.

ID3 - Algorithm

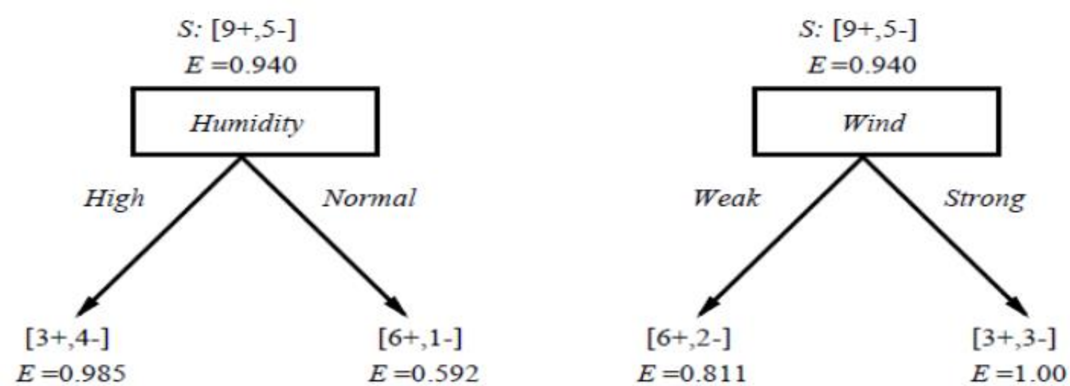
ID3(*Examples, TargetAttribute, Attributes*)

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

1	Outlook	Temperature	Humidity	Windy	Play Tennis
2	Sunny	Hot	High	F	N
3	Sunny	Hot	High	T	N
4	Overcast	Hot	High	F	Y
5	Rain	Mild	High	F	Y
6	Rain	Cool	Normal	F	Y
7	Rain	Cool	Normal	T	N
8	Overcast	Cool	Normal	T	Y
9	Sunny	Mild	High	F	N
10	Sunny	Cool	Normal	F	Y
11	Rain	Mild	Normal	F	Y
12	Sunny	Mild	Normal	T	Y
13	Overcast	Mild	High	T	Y
14	Overcast	Hot	Normal	F	Y
15	Rain	Mild	High	T	N

Compute the Gain and identify which attribute is the best as illustrated below

Which attribute is the best classifier?



Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

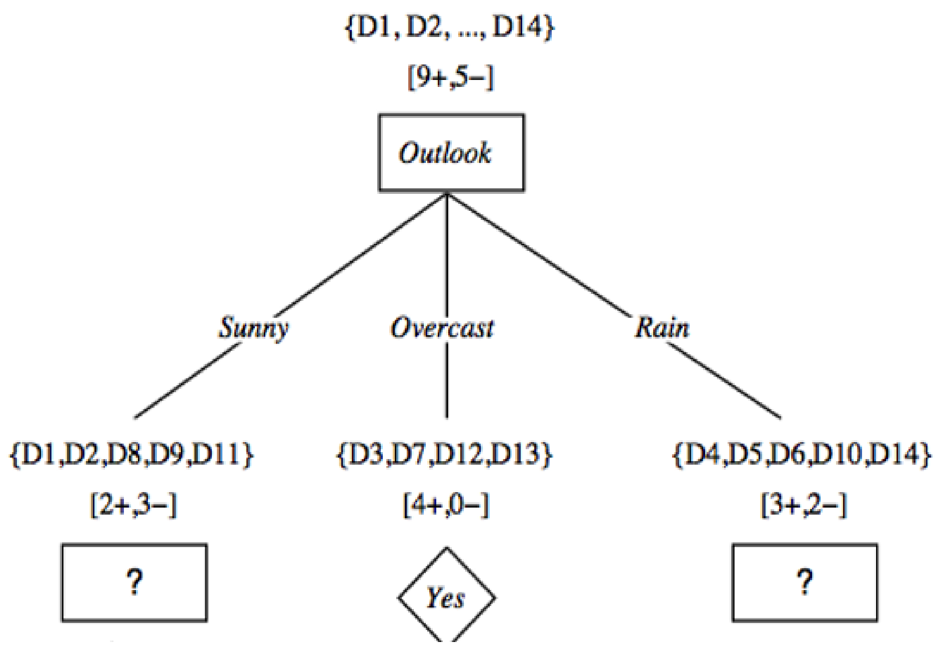
Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048

Which attribute to test at the root?

- Which attribute should be tested at the root?
  - *Gain*(*S*, *Outlook*) = 0.246
  - *Gain*(*S*, *Humidity*) = 0.151
  - *Gain*(*S*, *Wind*) = 0.048
  - *Gain*(*S*, *Temperature*) = 0.029
- *Outlook* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Outlook*
  - partition the training samples according to the value of *Outlook*

After first step

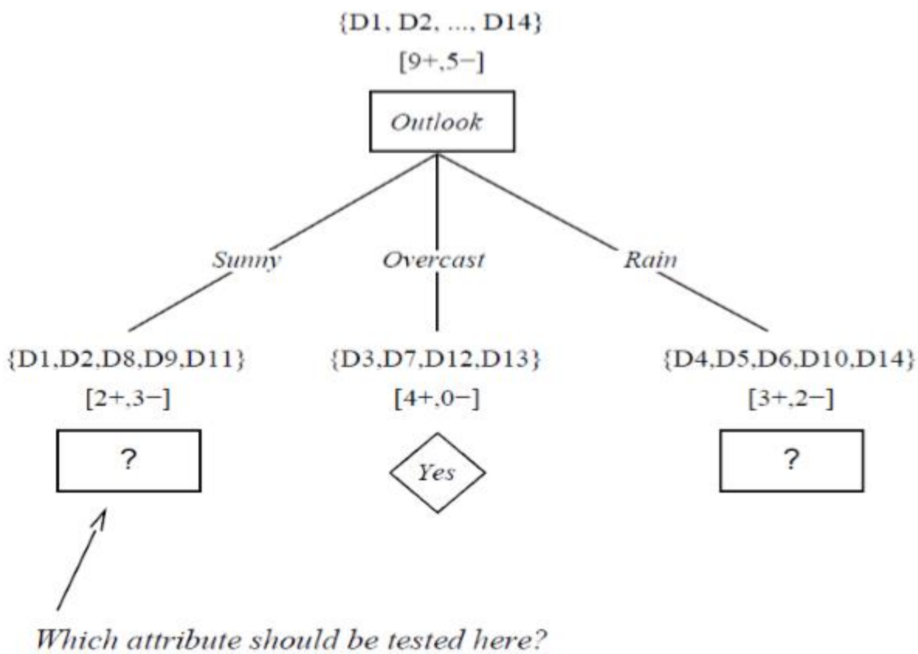


Second step

- **Working on *Outlook=Sunny* node:**

  $Gain(S_{Sunny}, Humidity) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$
  $Gain(S_{Sunny}, Wind) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0\ .019$
  $Gain(S_{Sunny}, Temp.) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$
- ***Humidity* provides the best prediction for the target**
- **Lets grow the tree:**
  - **add to the tree a successor for each possible value of *Humidity***
  - **partition the training samples according to the value of *Humidity***

Second and third steps



$S_{sunny} = \{D1,D2,D8,D9,D11\}$

$$Gain\ (S_{sunny}\ , Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$$

$$Gain\ (S_{sunny}\ , Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$$

$$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$$

```
import pandas as pd
import math

class Node:
    def __init__(self,l):
        self.label=l
        self.branches = {}

def entropy(data):
    total_ex = len(data)
    positive_ex = len(data.loc[data["Play Tennis"] == 'Y'])
    negative_ex = len(data.loc[data["Play Tennis"] == 'N'])
    entropy = 0
    if(positive_ex > 0):
        entropy += (-1)*(positive_ex/float(total_ex))*(math.log(positive_ex,2)-math.log(total_ex,2))
    if(negative_ex > 0):
        entropy += (-1)*(negative_ex/float(total_ex))*(math.log(negative_ex,2)-math.log(total_ex,2))
    return entropy

def gain(s,data,attrib):
    values = set(data[attrib])
    print(values)
    gain = s
    for val in values:
        gain -= len(data.loc[data[attrib] == val])/float(len(data))*entropy(data.loc[data[attrib] == val])
    return gain

def get_attrib(data):
    entropy_s = entropy(data)
    attribute =""
    max_gain = 0
    for attr in data.columns[:len(data.columns)-1]:
        g = gain(entropy_s,data,attr)

        if g > max_gain:
            max_gain = g
            attribute = attr

    return attribute
```

```python

def decision_tree(data):

   root = Node("NULL")

   if(entropy(data) == 0):
      if(len(data.loc[data[data.columns[-1]] == 'Y']) == len(data)):
         root.label = "Y"
         return root
      else:
         root.label = "N"
         return root

   if(len(data.columns) == 1):
      return
   else:
      attrib = get_attrib(data)
      root.label = attrib
      values = set(data[attrib])

      for val in values:
         root.branches[val] = decision_tree(data.loc[data[attrib] == val].drop(attrib,axis = 1))
      return root

def get_rules(root,rule,rules):
   if not root.branches:
      rules.append(rule[:-2]+" => "+root.label)
      return rules

   for i in root.branches:
      get_rules(root.branches[i],rule+root.label+"="+i+" ^ ",rules)
   return rules
```

```python

def test(tree,test_str):
    if not tree.branches:
        return tree.label
    return test(tree.branches[test_str[tree.label]],test_str)


data = pd.read_csv('Data_3.csv')

entropy_s = entropy(data)

attrib_count = 0
cols = len(data.columns)-1

tree = decision_tree(data)

rules = get_rules(tree,"",[])
print(rules)

test_str = {}
print("Enter test case input")
for i in data.columns[:-1]:
    test_str[i] = input(i+": ")

print(test_str)
print(test(tree,test_str))
```

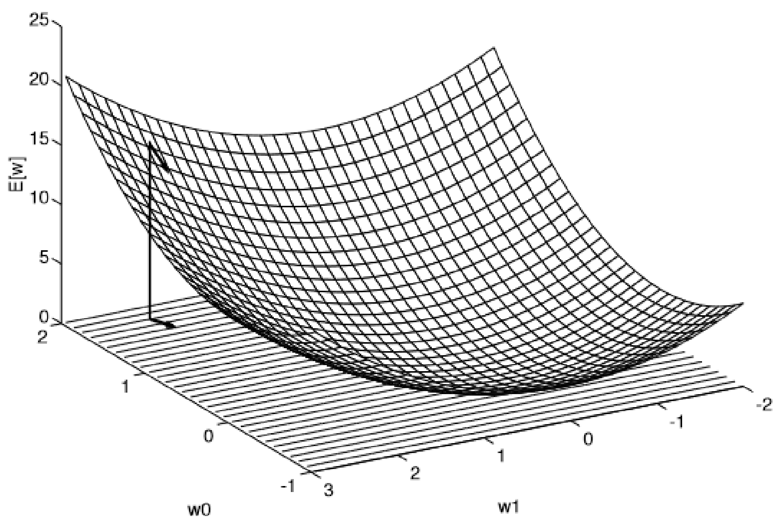
Output:

```
In [1]: runfile('C:/Users/Admin/Desktop/Program3.py', wdir='C:/Users/Admin/Desktop')
{'Sunny', 'Overcast', 'Rain'}
{'Hot', 'Cool', 'Mild'}
{'Normal', 'High'}
{'T', 'F'}
{'Hot', 'Cool', 'Mild'}
{'Normal', 'High'}
{'T', 'F'}
{'Cool', 'Mild'}
{'Normal', 'High'}
{'T', 'F'}
['Outlook=Sunny ^ Humidity=Normal  => Y', 'Outlook=Sunny ^ Humidity=High  => N', 'Outlook=Overcast  => Y', 'Outlook=Rain ^ Windy=T  => N',
'Outlook=Rain ^ Windy=F  => Y']
Enter test case input

Outlook: Rain

Temperature: Mild

Humidity: High

Windy: F
{'Outlook': 'Rain', 'Temperature': 'Mild', 'Humidity': 'High', 'Windy': 'F'}
Y
```

Program 4

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

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

     $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$

  4. Update each network weight $w_{i,j}$

     $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

     where

     $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

```
#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)
train_network(network, dataset, 0.5, 20, n_outputs)
```

```
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)
```

```python

# 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']

```

```python
# 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)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

Output:

```
In [2]: runfile('C:/Users/Admin/Desktop/ANN Backpropagation.py', wdir='C:/Users/Admin/Desktop')
>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}]
```

Program 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.

Bayes Theorem

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

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

Naive Bayes Classifier

Assume target function $f : X \rightarrow V$, where each instance x described by attributes $\langle a_1, a_2 \ldots a_n \rangle$. Most probable value of $f(x)$ is:

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

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

Naive Bayes assumption:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

which gives

Naive Bayes classifier: $v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$

- **Gaussian: It is used in classification and it assumes that features follow a normal distribution. Gaussian Naive Bayes is used in cases when all our features are continuous. For example in Iris dataset features are sepal width, petal width, sepal length, petal length.**

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

```python
print("\nNaive Bayes Classifier for concept learning problem")
import csv
import math
def safe_div(x,y):
    if y == 0:
        return 0
    return x / y

def loadCsv(filename):
   lines = csv.reader(open(filename))
   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)
   i=0
   while len(trainSet) < trainSize:
      #index = random.randrange(len(copy))

      trainSet.append(copy.pop(i))
   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 safe_div(sum(numbers),float(len(numbers)))

def stdev(numbers):
   avg = mean(numbers)
   variance = safe_div(sum([pow(x-avg,2) for x in numbers]),float(len(numbers)-1))
   return math.sqrt(variance)
```

```python

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(-safe_div(math.pow(x-mean,2),(2*math.pow(stdev,2))))
   final = safe_div(1 , (math.sqrt(2*math.pi) * stdev)) * exponent
   return final

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
```

```python

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
   accuracy = safe_div(correct,float(len(testSet))) * 100.0
   return accuracy
```

```python

def main():
    filename = 'ConceptLearning.csv'
    splitRatio = 0.75
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into'.format(len(dataset)))
    print('Number of Training data: ' + (repr(len(trainingSet))))
    print('Number of Test Data: ' + (repr(len(testSet))))
    print("\nThe values assumed for the concept learning attributes are\n")
    print("OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1 Mild=2 Cool=3\nHUMIDITY=> High=1 Normal=2\nWIND=> Weak=1 Strong=2")
    print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
    print("\nThe Training set are:")
    for x in trainingSet:
        print(x)
    print("\nThe Test data set are:")
    for x in testSet:
        print(x)
    print("\n")
    # prepare model
    summaries = summarizeByClass(trainingSet)
    # test model
    predictions = getPredictions(summaries, testSet)
    actual = []
    for i in range(len(testSet)):
        vector = testSet[i]
        actual.append(vector[-1])
    # Since there are five attribute values, each attribute constitutes to 20% accuracy. So if all attributes match with predictions then 100% accuracy
    print('Actual values: {0}%'.format(actual))
    print('Predictions: {0}%'.format(predictions))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: {0}%'.format(accuracy))

main()

```

Output:

```
In [7]: runfile('C:/Users/Admin/Desktop/NaiveBayes.py', wdir='C:/Users/Admin/Desktop')

Naive Bayes Classifier for concept learning problem
Split 16 rows into
Number of Training data: 12
Number of Test Data: 4

The values assumed for the concept learning attributes are

OUTLOOK=> Sunny=1 Overcast=2 Rain=3
TEMPERATURE=> Hot=1 Mild=2 Cool=3
HUMIDITY=> High=1 Normal=2
WIND=> Weak=1 Strong=2
TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5

The Training set are:
[1.0, 1.0, 1.0, 1.0, 5.0]
[1.0, 1.0, 1.0, 2.0, 5.0]
[2.0, 1.0, 1.0, 2.0, 10.0]
[3.0, 2.0, 1.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 2.0, 5.0]
[2.0, 3.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 1.0, 5.0]
[1.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 2.0, 2.0, 10.0]
[2.0, 2.0, 1.0, 2.0, 10.0]

The Test data set are:
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]
[1.0, 2.0, 1.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 2.0, 5.0]


Actual values: [10.0, 5.0, 10.0, 5.0]%
Predictions: [5.0, 10.0, 5.0, 5.0]%
Accuracy: 25.0%
```

Program 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.

Algorithm :

Learning to Classify Text: Preliminaries

Target concept Interesting? : $Document \rightarrow \{+, -\}$

1. Represent each document by vector of words

- one attribute per word position in document

2. Learning: Use training examples to estimate

- $P(+)$ – $P(-)$
- $P(doc|+)$ – $P(doc|-)$

Naive Bayes conditional independence assumption

$$P(doc|v_j) = \prod_{i=1}^{length(doc)} P(a_i = w_k|v_j)$$

where $P(a_i = w_k \mid v_j)$ is probability that word in position i is w_k, given v_j

one more assumption:

$$P(a_i = w_k|v_j) = P(a_m = w_k|v_j), \forall i, m$$

Learning to Classify Text: Algorithm

S1: LEARN_NAIVE_BAYES_TEXT (*Examples*, V)
S2: CLASSIFY_NAIVE_BAYES_TEXT (*Doc*)

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

S1: LEARN_NAIVE_BAYES_TEXT (*Examples*, V)

1. *collect all words and other tokens that occur in Examples*

- *Vocabulary* ← all distinct words and other tokens in *Examples*

2. *calculate the required* $P(v_j)$ *and* $P(w_k \mid v_j)$ *probability terms*

- For each target value v_j in V do

$$P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$$

  - ***$docs_j$* ← subset of *Examples* for which the target value is v_j**
  - $Text_j$ ← a single document created by concatenating all members of $docs_j$
  - n ← total number of words in $Text_j$ (counting duplicate words multiple times)
  - for each word w_k in *Vocabulary*
    - * n_k ← number of times word w_k occurs in $Text_j$

$$P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$$

S2: CLASSIFY_NAIVE_BAYES_TEXT (*Doc*)

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB} where

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

Twenty News Groups

- Given 1000 training documents from each group Learn to classify new documents according to which newsgroup it came from

comp.graphics	misc.forsale	alt.atheism	sci.space
comp.os.ms-windows.misc	rec.autos	soc.religion.christian	sci.crypt
comp.sys.ibm.pc.hardware	rec.motorcycles	talk.religion.misc	sci.electronics
comp.sys.mac.hardware	rec.sport.baseball	talk.politics.mideast	sci.med
comp.windows.x	rec.sport.hockey	talk.politics.misc	
		talk.politics.guns	

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import classification_report
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)
print("\n".join(twenty_train.data[0].split("\n")))
print(twenty_train.target[0])
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_names))
print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

Output:

```
Michael Collier (Programmer)                The Computer Unit,
Email: M.P.Collier@uk.ac.city               The City University,
Tel: 071 477-8000 x3769                     London,
Fax: 071 477-8565                           EC1V 0HB.

1
Accuracy: 0.834886817577
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

           avg / total       0.88      0.83      0.84      1502

confusion matrix is
 [[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

Program 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.

Algorithm :

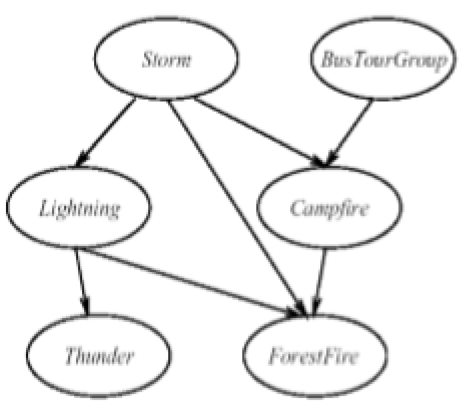
Bayesian Network (BAYESIAN BELIEF NETWORKS

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

Conditional Independence

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

Bayesian Belief Network



	S,B	$S,\neg B$	$\neg S,B$	$\neg S,\neg B$
C	0.4	0.1	0.8	0.2
$\neg C$	0.6	0.9	0.2	0.8

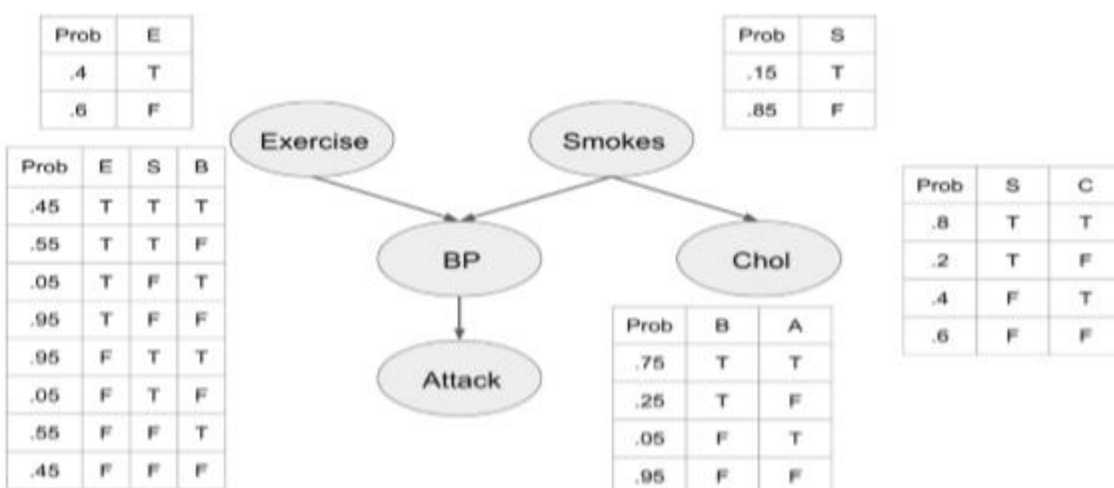
- Represents a set of conditional independence assertions:
    - Each node is asserted to be conditionally independent of its non descendants, given its immediate predecessors.
    - Directed acyclic graph

- Represents joint probability distribution over all variables
    - e.g., $P(Storm, BusTourGroup, \ldots, ForestFire)$
    - in general,

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

    where $Parents(Y_i)$ denotes immediate predecessors of Y_i in graph
    - so, joint distribution is fully defined by graph, plus the $P(y_i | Parents(Y_i))$

Example 1:



Example2 :

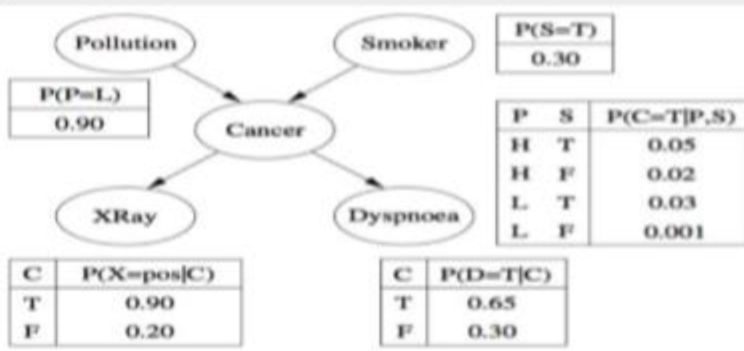


FIGURE 2.1
A BN for the lung cancer problem.

```python
import bayespy as bp
import numpy as np
import csv
from colorama import init
init()

# Define Parameter Enum values
#Age
ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4}
# Gender
genderEnum = {'Male':0, 'Female':1}
# FamilyHistory
familyHistoryEnum = {'Yes':0, 'No':1}
# Diet(Calorie Intake)
dietEnum = {'High':0, 'Medium':1, 'Low':2}
# LifeStyle
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}
# Cholesterol
cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}
# HeartDisease
heartDiseaseEnum = {'Yes':0, 'No':1}
#heart_disease_data.csv
with open('heart_disease_data.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:
        data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])
# Training data for machine learning todo: should import from csv
data = np.array(data)
N = len(data)
```

```python

# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:,4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:,5])

# Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:,6])
p_heartdisease.update()
```

```python

# Sample Test with hardcoded values
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'], cholesterolEnum['High

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryE
    print("Probability(HeartDisease) = " +  str(res))
    #print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1  "))

```

Output:

```
In [1]: runfile('C:/Users/TVK/Desktop/BBN.py', wdir='C:/Users/TVK/Desktop')


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2,
'Youth': 3, 'Teen': 4}4

Enter Gender: {'Male': 0, 'Female': 1}0

Enter FamilyHistory: {'Yes': 0, 'No': 1}1

Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0

Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}1

Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5

Enter for Continue:0, Exit :1
```

Program 8

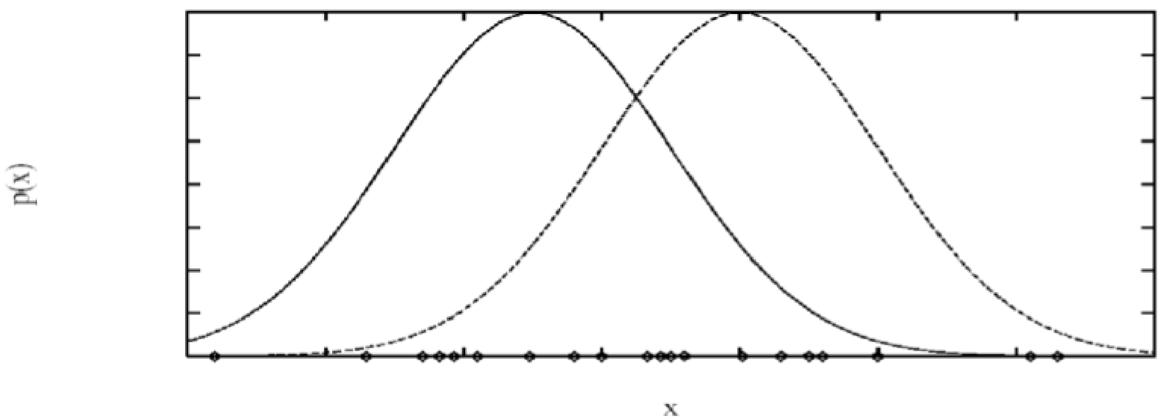
Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Algorithm :

Expectation Maximization (EM) Algorithm

- When to use:
  - Data is only partially observable
  - Unsupervised clustering (target value unobservable)
  - Supervised learning (some instance attributes unobservable)
- Some uses:
  - Train Bayesian Belief Networks
  - Unsupervised clustering (AUTOCLASS)
  - Learning Hidden Markov Models

Generating Data from Mixture of *k* Gaussians



- **Each instance x generated by**

1. Choosing one of the *k* Gaussians with uniform probability
2. Generating an instance at random according to that Gaussian

EM for Estimating k Means

- Given:
    - Instances from X generated by mixture of k Gaussian distributions
    - Unknown means $<\mu_1,\ldots,\mu_k>$ of the k Gaussians
    - Don't know which instance x_i was generated by which Gaussian
- Determine:
    - Maximum likelihood estimates of $<\mu_1,\ldots,\mu_k>$
- Think of full description of each instance as

  $y_i = < x_i, z_{i1}, z_{i2}>$ where

    - z_{ij} is 1 if x_i generated by jth Gaussian
    - x_i observable
    - z_{ij} unobservable

• EM Algorithm: Pick random initial $h = <\mu_1, \mu_2>$ then iterate

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

$$E[z_{ij}] = \frac{p(x = x_i|\mu = \mu_j)}{\Sigma_{n=1}^{2} p(x = x_i|\mu = \mu_n)} = \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\Sigma_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}}$$

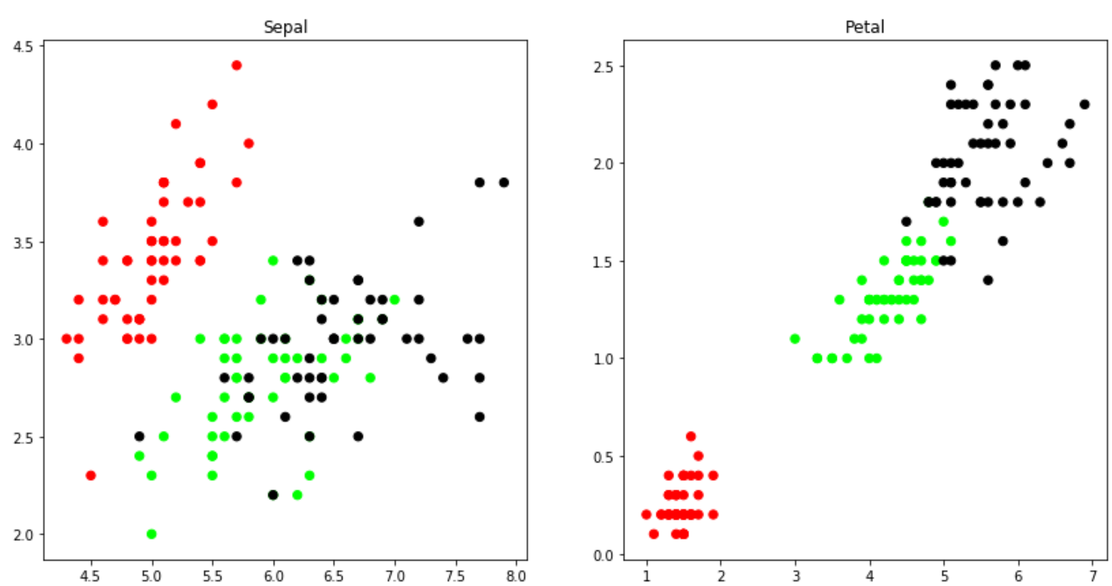
M step: Calculate a new maximum likelihood hypothesis $h' = <\mu'_1, \mu'_2>$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated above. Replace $h = <\mu_1, \mu_2>$ by $h' = <\mu'_1, \mu'_2>$.

$$\mu_j \leftarrow \frac{\Sigma_{i=1}^{m} E[z_{ij}]\ \ x_i}{\Sigma_{i=1}^{m} E[z_{ij}]}$$

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
%matplotlib inline
# import some data to play with
iris = datasets.load_iris()
# Store the inputs as a Pandas Dataframe and set the column names
X = pd.DataFrame(iris.data)
#print(X)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
```

```
#print(X.columns)
#print("X:",x)
#print("Y:",y)
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap = np.array(['red', 'lime', 'black'])
# Plot Sepal
plt.subplot(1, 2, 1)
plt.scatter(X.Sepal_Length,X.Sepal_Width, c=colormap[y.Targets], s=40)
plt.title('Sepal')
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Petal')
```

Out[6]: Text(0.5,1,'Petal')

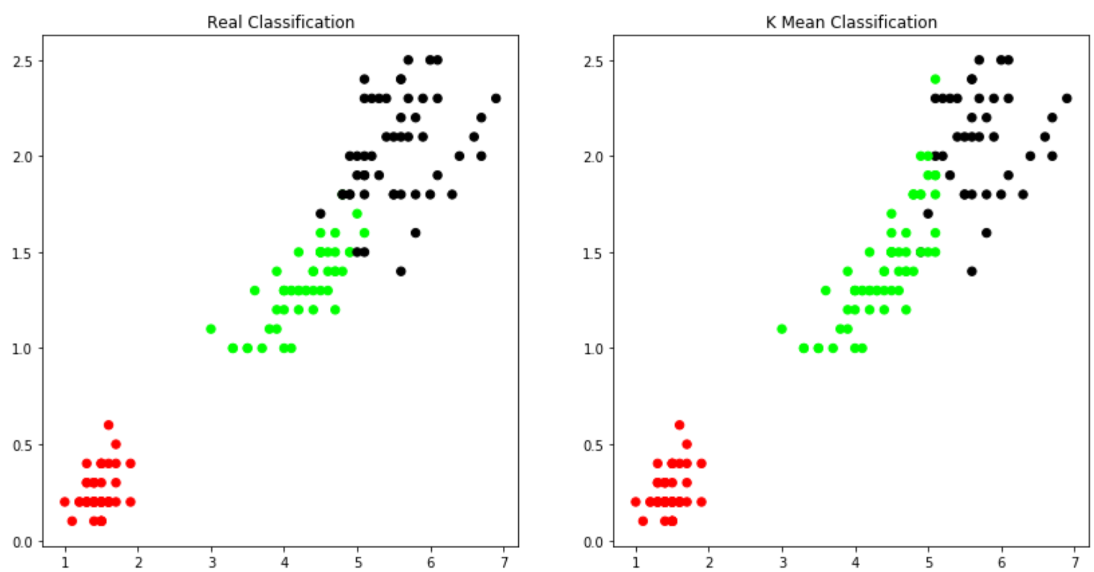


In [7]:
```python
# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(X) # This is what KMeans thought
model.labels_
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1,
       2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
       1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

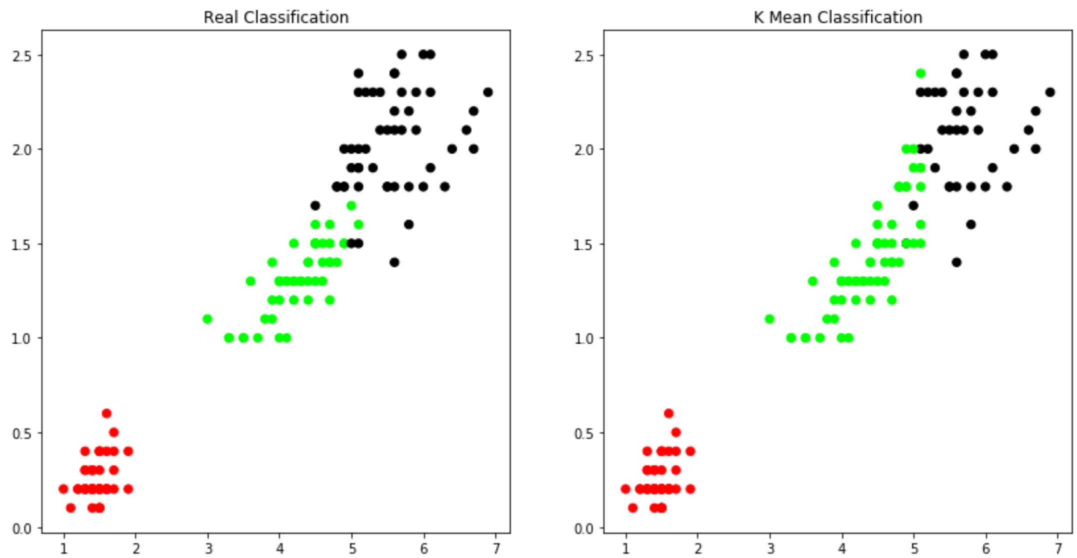
In [8]:
```python
# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
```

Out[8]: Text(0.5,1,'K Mean Classification')



In [11]:
```
predY = np.choose(model.labels_, [0, 1, 2]).astype(np.int64)
# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
colormap = np.array(['red', 'lime', 'black'])
# Plot Orginal
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
# Plot Predicted with corrected values
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[predY], s=40)
plt.title('K Mean Classification')
```

Out[11]: Text(0.5,1,'K Mean Classification')



In [13]:
```
sm.accuracy_score(y, model.labels_)
```

Out[13]: 0.89333333333333331

In [15]:
```
#Gaussian mixture - EM
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
xs.sample(5)
```

Out[15]:

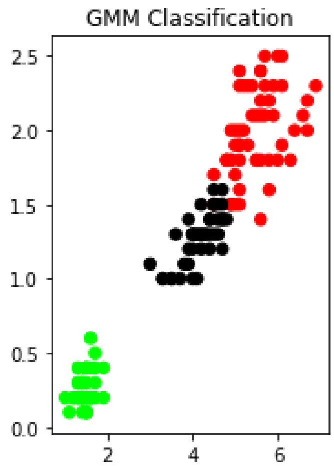
	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
11	-1.264185	0.800654	-1.227541	-1.312977
76	1.159173	-0.587764	0.592162	0.264699
130	1.886180	-0.587764	1.331416	0.922064
95	-0.173674	-0.124958	0.250967	0.001753
128	0.674501	-0.587764	1.047087	1.185010

In [16]:
```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=1, precisions_init=None,
                random_state=None, reg_covar=1e-06, tol=0.001, verbose=0, verbose_interval=10,
                warm_start=False, weights_init=None)
y_cluster_gmm = gmm.predict(xs)
y_cluster_gmm
```

Out[16]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0,
       2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [18]:
```
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm], s=40)
plt.title('GMM Classification')
```

Out[18]: `Text(0.5,1,'GMM Classification')`



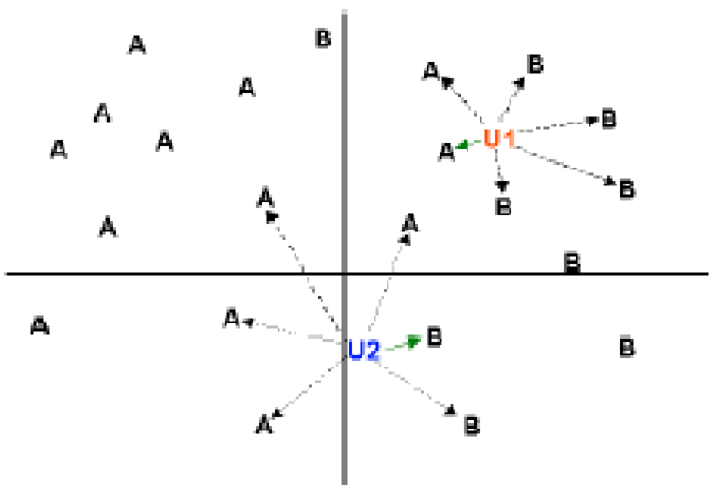
In [19]:
```
sm.accuracy_score(y, y_cluster_gmm)
```

Program 9

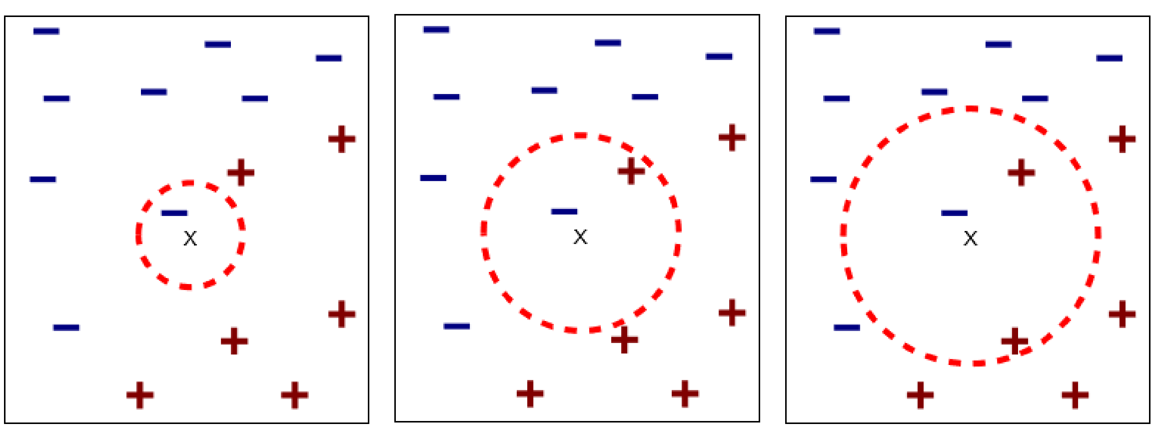
Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

K-Nearest-Neighbor Algorithm

- **Principle: points (documents) that are close in the space belong to the same class**



Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

```python
import csv
import random
import math
import operator
def loadDataset(filename, split, trainingSet=[], testSet=[]):
    with open(filename) as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]
```

```python

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

def main():
   # prepare data
   trainingSet=[]
   testSet=[]
   split = 0.67
   loadDataset('iris_data.csv', split, trainingSet, testSet)
   print ('\n Number of Training data: ' + (repr(len(trainingSet))))
   print (' Number of Test Data: ' + (repr(len(testSet))))
   # generate predictions
   predictions=[]
   k = 3
   print('\n The predictions are: ')
   for x in range(len(testSet)):
      neighbors = getNeighbors(trainingSet, testSet[x], k)
      result = getResponse(neighbors)
      predictions.append(result)
      print(' predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
   accuracy = getAccuracy(testSet, predictions)
   print('\n The Accuracy is: ' + repr(accuracy) + '%')

main()
```

Output:

```
Number of Training data: 107
Number of Test Data: 42

The predictions are:
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
```

```
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
```

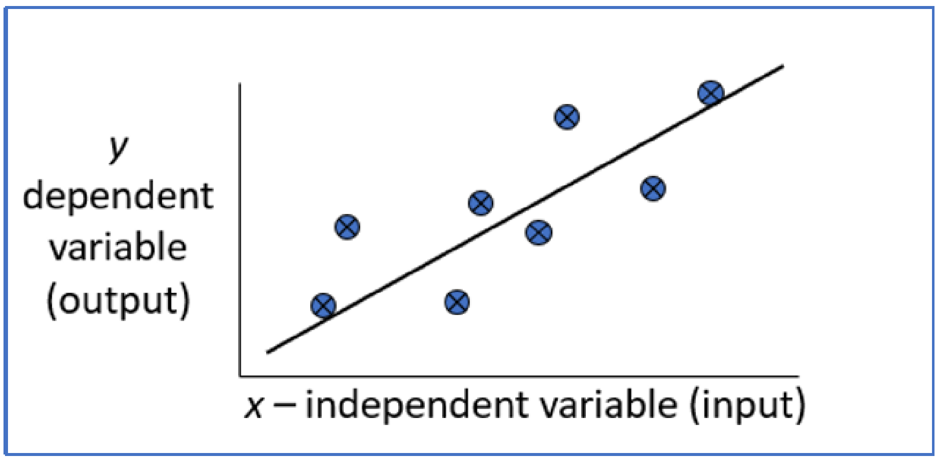
Program 10

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

Algorithm :

Regression:

- Regression is a technique from statistics that is used to predict values of a 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.



Lowess Algorithm: **Locally weighted regression** is a very powerful non-parametric 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 .

```python
from math import ceil
import numpy as np
from scipy import linalg

def lowers(x, y, f=2./3., iter=3):
    n = len(x)
    r = int(ceil(f*n))
    h = [np.sort(np.abs(x-x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:,None] - x[None,:])/h),0.0,1.0)
    w = (1 - w**3)**3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iter):
        for i in range(n):
            weights = delta * w[:,i]
            b = np.array([np.sum(weights*y),np.sum(weights*y*x)])
            A = np.array([[np.sum(weights),np.sum(weights*x)],[np.sum(weights*x),np.sum(weights*x*x)]])
            beta = linalg.solve(A,b)
            yest[i] = beta[0] + beta[1]*x[i]
        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals/(6.0 * s),-1,1)
        delta = (1 - delta ** 2)**2
    return yest

if __name__ == '__main__':
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    print("====================== value of x =========================")
    print(x)
    y = np.sin(x) + 0.3 * np.random.randn(n)
    print("====================== value of y =========================")
    print(y)
    f = 0.25
    yest = lowers(x,y, f=f, iter = 3)
    import pylab as pl
    pl.clf()
    pl.plot(x,y, label = 'y noisy')
    pl.plot(x,yest, label = 'y pred')
    pl.legend()
    pl.show()
```

Output:

```
In [2]: runfile('C:/Users/Admin/Desktop/Reg.py', wdir='C:/Users/Admin/Desktop')
====================== value of x ========================
[0.         0.06346652 0.12693304 0.19039955 0.25386607 0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
 3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
 4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
 4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
 4.95038842 5.01385494 5.07732146 5.14078798 5.2042545  5.26772102
 5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
 5.71198664 5.77545316 5.83891968 5.9023862  5.96585272 6.02931923
 6.09278575 6.15625227 6.21971879 6.28318531]
```

```
===================== value of y ========================
[ 0.1266644  -0.03244146  0.50361126  0.42495004  0.63350245  0.10235533
  0.62906337  0.62899943  0.54260171  0.46671203  0.37989349  0.59253887
  1.20004319  0.71552505  0.54649986  0.77124862  0.89694616  0.8722321
  0.9004438   0.76715746  0.81631107  0.69056155  1.15803992  0.9797445
  1.03228202  1.0936951   1.22108739  1.06986031  0.99444149  1.20076387
  1.32703772  1.25791332  1.13683781  1.06678327  0.94863286  0.17945764
  1.07616552  0.68863536  0.60433969  0.57382534  1.01666485  0.49003627
  0.77793754  0.42030935 -0.25304554  0.45142546  0.14481713 -0.31577034
  0.129159    0.19477737  0.2308672  -0.22417117 -0.15971811 -0.56899249
 -0.21710665 -0.80061264 -0.37023792 -0.21745574 -0.72636135 -0.6348439
 -0.61895226 -0.48990179 -0.38482874 -0.68576498 -0.44594557 -0.52611693
 -0.6976268  -0.55668743 -1.11335092 -0.74359858 -0.62046239 -1.34282407
 -0.4937161  -1.33600267 -1.24113564 -0.83742224 -1.24943033 -1.12400449
 -0.95820833 -1.01412379 -0.6189642  -1.28402795 -1.0600953  -0.82065032
 -1.05378546 -0.38008588 -0.42376466 -1.11587338 -0.23794601 -0.28969893
 -0.48702635 -0.28412428 -0.57085416 -0.28177065  0.16293896  0.00919529
 -0.00896556  0.25115278 -0.15466518 -0.1977382 ]
```