ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY

A Unit of Alva's Education Foundation, Moodbidri.

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

Laboratory Manual of

MACHINE LEARNING LABORATORY

SUBJECT CODE: 15CSL76

SEMESTER – VII (2018 – 2019)

Prepared by

Dr. Roopalakshmi. R

Professor

Department of Information Science and Engineering

Reviewed & Approved By

Mr. Jayantkumar A. Rathod

Associate Professor & Head

Department of Information Science and Engineering

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

AIET – MIJAR

MACHINE LEARNING LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2016 -2017)
SEMESTER – VII

Course objectives: This course will enable students to

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

Description (If any):

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

Lab Experiments:

1. Implement and demonstrate the **FIND-S Algorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a *.CSV* file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.
3. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5. Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a *.CSV* file. Compute the accuracy of the classifier, considering few test data sets.
6. Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

7. Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. Apply **EM algorithm** to cluster a set of data stored in a *.CSV* file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
9. Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
10. Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Course outcomes: The students should be able to:

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

Conduction of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script
- Marks distribution: Procedure + Conduction + Viva:**20 + 50 +10 (80)**

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

INDEX

Department of Information Science & Engineering

SEMESTER - VII

Course Code: **15CSL76** | Course Name: **MACHINE LEARNING LABORATORY**

Course Teacher: **Dr. Roopalakshmi.R**

Course Outcomes: After studying this course, students will be able to,

CO Numbers	Course Outcomes	Blooms Level	Target Level
CO1	Understand the implementation procedures for the machine learning algorithms.	L2	2
CO2	Design Java/Python programs for various Learning algorithms.	L4	2
CO3	Apply appropriate data sets to the Machine Learning algorithms.	L3	2
CO4	Identify and apply Machine Learning algorithms to solve real world problems	L3	2

CO-PO/PSO Mapping Matrix:

CO Numbers	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	L	L	L	L	L				L	L			L	L	L
CO2	L	M	M	L	M				L	L			L	-	L
CO3	M	M	M	M	M				L	L	L	L	L	M	L
CO4	M	L	L	L	M		L		L	L			L	L	L
AVG	M	M	M	L	M				L	L			L	L	L

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.

Algorithm :

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

Illustration:

Step1: Find S

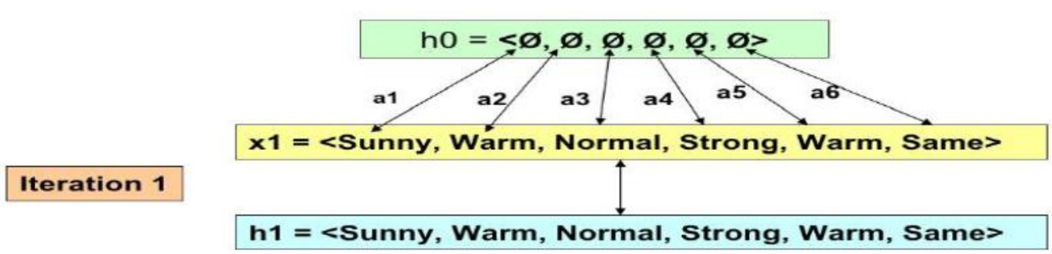
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

1. Initialize *h* to the most specific hypothesis in *H*

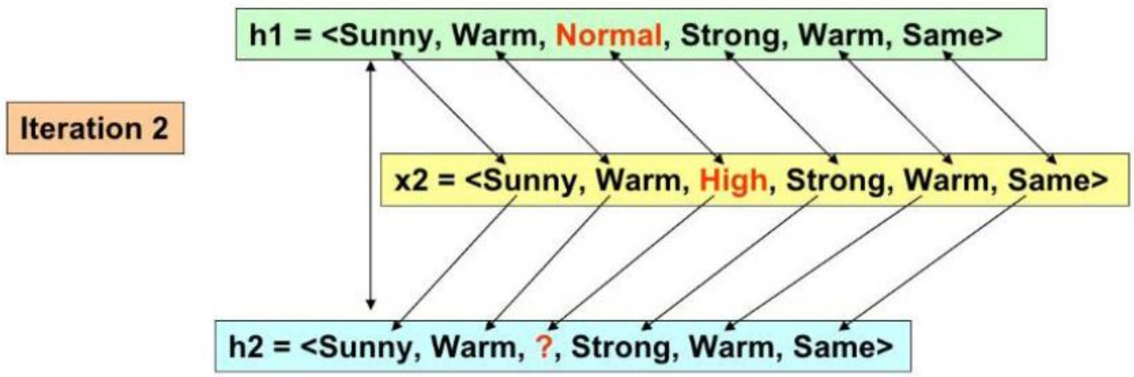
h0 = <Ø, Ø, Ø, Ø, Ø, Ø>

Step2 : Find S

2. For each positive training instance *x*
- **For each attribute constraint a_i in *h***
  If the constraint a_i is satisfied by *x*
  Then do nothing
  Else replace a_i in *h* by the next more general constraint that is satisfied by *x*

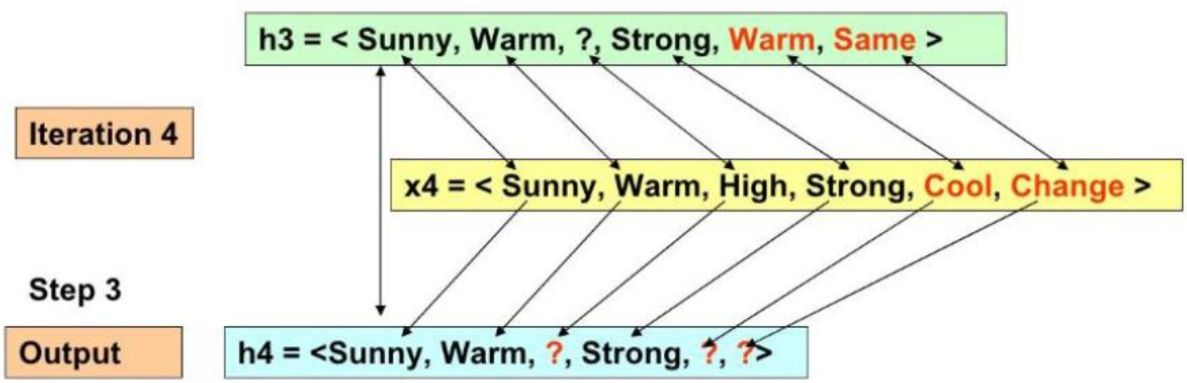


Step2 : Find S



Iteration 3 | Ignore | h3 = <Sunny, Warm, ?, Strong, Warm, Same>

Iteration 4 and Step 3 : Find S



Source Code of the Program :

```
import random
import csv

attributes =    [['Sunny','Rainy'],
                ['Warm','Cold'],
                ['Normal','High'],
                ['Strong','Weak'],
                ['Warm','Cool'],
                ['Same','Change']]
num_attributes = len(attributes)
print (" \n The most general hypothesis : ['?','?','?','?','?','?']\n")
print ("\n The most specific hypothesis : ['0','0','0','0','0','0']\n")
a = []
print("\n The Given Training Data Set \n")
with open('C:\\Users\\thyagaragu\\Desktop\\Data\\ws.csv', 'r') as csvFile:
reader = csv.reader(csvFile)
for row in reader:
a.append (row)
print(row)
print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

# Comparing with First Training Example
for j in range(0,num_attributes):
hypothesis[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
if a[i][num_attributes]=='Yes':
```

```
for j in range(0,num_attributes):
if a[i][j]!=hypothesis[j]:
hypothesis[j]='?'
else :
hypothesis[j]= a[i][j]
print(" For Training Example No :{0} the hypothesis is ".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

Output :

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

Algorithm:

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

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

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

Case 2: If d is a negative example

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

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

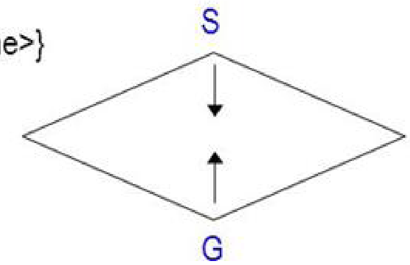
Illustration :

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

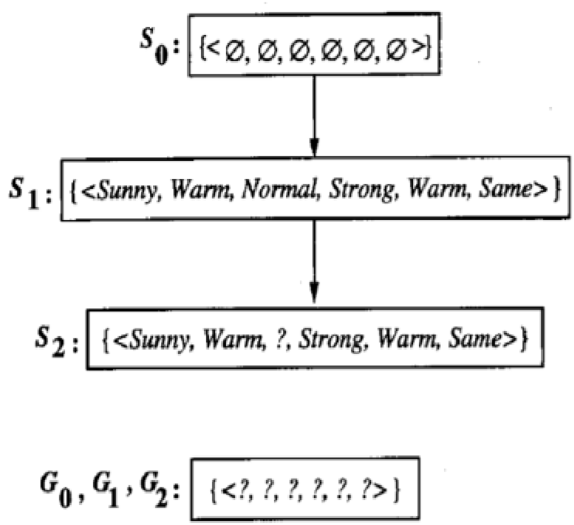
$S_0 = \{<\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing>\}$
$G_0 = \{<?, ?, ?, ?, ?, ?>\}$

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

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



Trace1 :

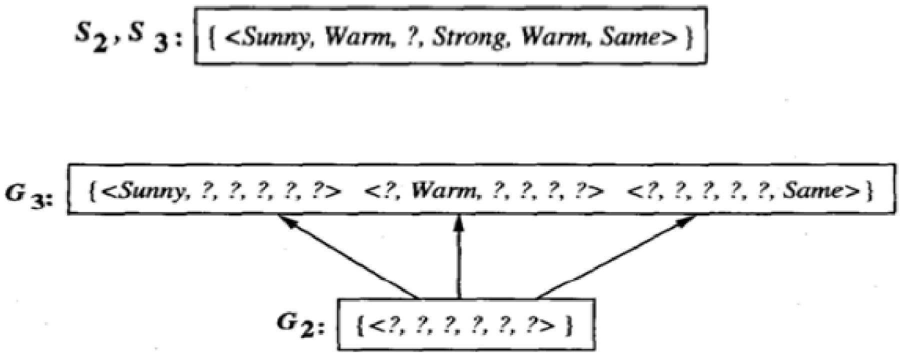


Training examples:

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

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

Trace 2:

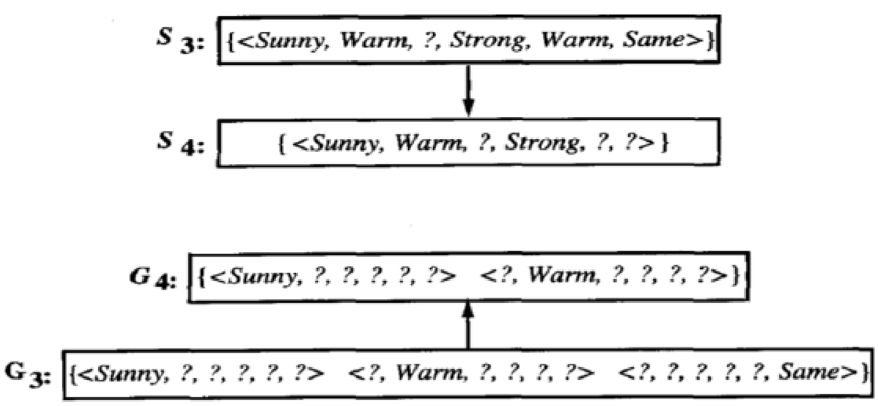


Training Example:

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

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

Trace3 :

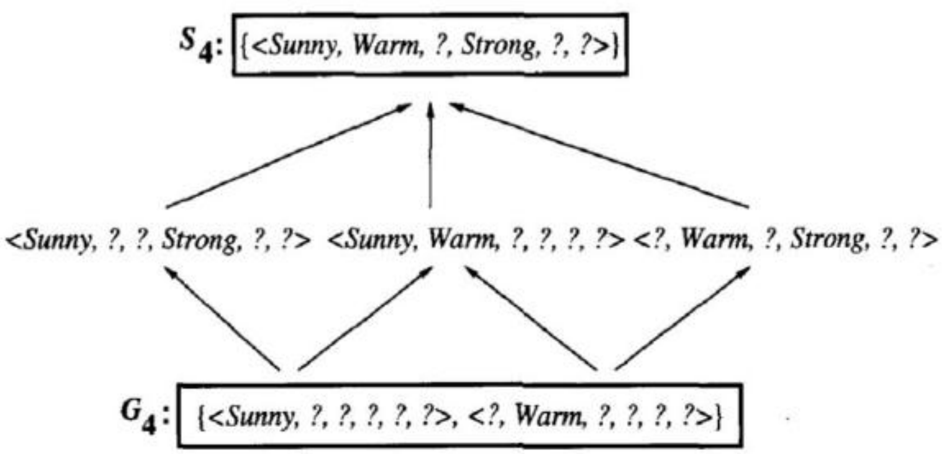


Training Example:

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

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

Final Version Space:



The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

Source Code :

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

```
list(zip(l1, l2))
[(1, 3), (2, 4), (3, 5)]

# min_generalizations
def fulfills(example, hypothesis):
### the implementation is the same as for hypotheses:
return more_general(hypothesis, example)
def min_generalizations(h, x):
h_new = list(h)
for i in range(len(h)):
if not fulfills(x[i:i+1], h[i:i+1]):
h_new[i] = '?' if h[i] != '0' else x[i]
return [tuple(h_new)]
min_generalizations(h=('0', '0' , 'sunny'),
x=('rainy', 'windy', 'cloudy'))
[('rainy', 'windy', '?')]
def min_specializations(h, domains, x):
results = []
for i in range(len(h)):
if h[i] == "?":
for val in domains[i]:
if x[i] != val:
h_new = h[:i] + (val,) + h[i+1:]
results.append(h_new)
elif h[i] != "0":
h_new = h[:i] + ('0',) + h[i+1:]
results.append(h_new)
return results
min_specializations(h=('?', 'x',),
domains=[['a', 'b', 'c'], ['x', 'y']],
x=('b', 'x'))
[('a', 'x'), ('c', 'x'), ('?', '0')]
with open('C:\\Users\\thyagaragu\\Desktop\\Data\\c1.csv') as csvFile:
examples = [tuple(line) for line in csv.reader(csvFile)]

#examples = [('sunny', 'warm', 'normal', 'strong', 'warm', 'same',True),
# ('sunny', 'warm', 'high', 'strong', 'warm', 'same',True),
# ('rainy', 'cold', 'high', 'strong', 'warm', 'change',False),
```

```
# ('sunny', 'warm', 'high', 'strong', 'cool', 'change',True)]
examples
[('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Y'),
('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Y'),
('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'N'),
('Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Y')]
def get_domains(examples):
d = [set() for i in examples[0]]
for x in examples:
for i, xi in enumerate(x):
d[i].add(xi)
return [list(sorted(x)) for x in d]
get_domains(examples)
[['Rainy', 'Sunny'],
['Cold', 'Warm'],
['High', 'Normal'],
['Strong'],
['Cool', 'Warm'],
['Change', 'Same'],
['N', 'Y']]
def candidate_elimination(examples):
domains = get_domains(examples)[:-1]
G = set([g_0(len(domains))])
S = set([s_0(len(domains))])
i=0
print("\n G[{0}]:".format(i),G)
print("\n S[{0}]:".format(i),S)
for xcx in examples:
i=i+1
x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions
if cx=='Y': # x is positive example
G = {g for g in G if fulfills(x, g)}
S = generalize_S(x, G, S)
else: # x is negative example
S = {s for s in S if not fulfills(x, s)}
G = specialize_G(x, domains, G, S)
print("\n G[{0}]:".format(i),G)
print("\n S[{0}]:".format(i),S)
```

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

output :

```
G[0]: {('?', '?', '?', '?', '?', '?')}
S[0]: {('0', '0', '0', '0', '0', '0')}
G[1]: {('?', '?', '?', '?', '?', '?')}
S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}
 G[2]: {('?', '?', '?', '?', '?', '?')}
S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}
G[3]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same')}
S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}
G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}
S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
```

****

Program3: 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.

Algorithm :

ID3 - Algorithm

ID3(*Examples*, *TargetAttribute*, *Attributes*)

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

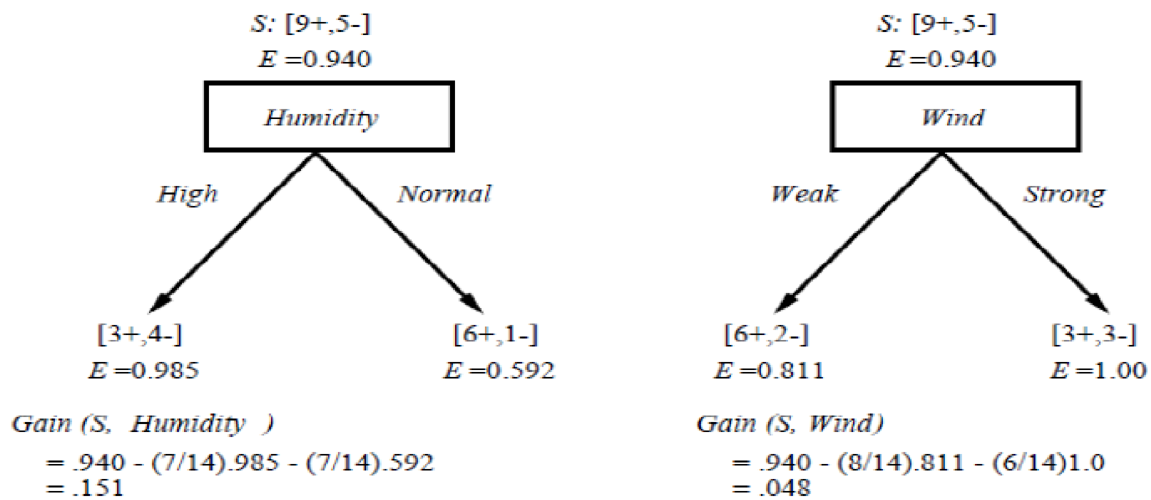
Illustration:

To illustrate the operation of ID3, let's consider the learning task represented by the below examples

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

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

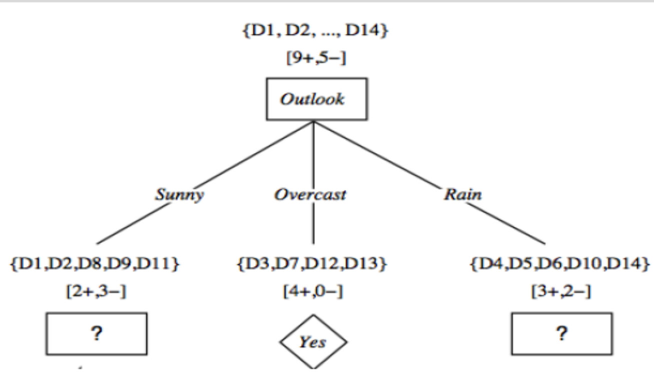
Which attribute is the best classifier?



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

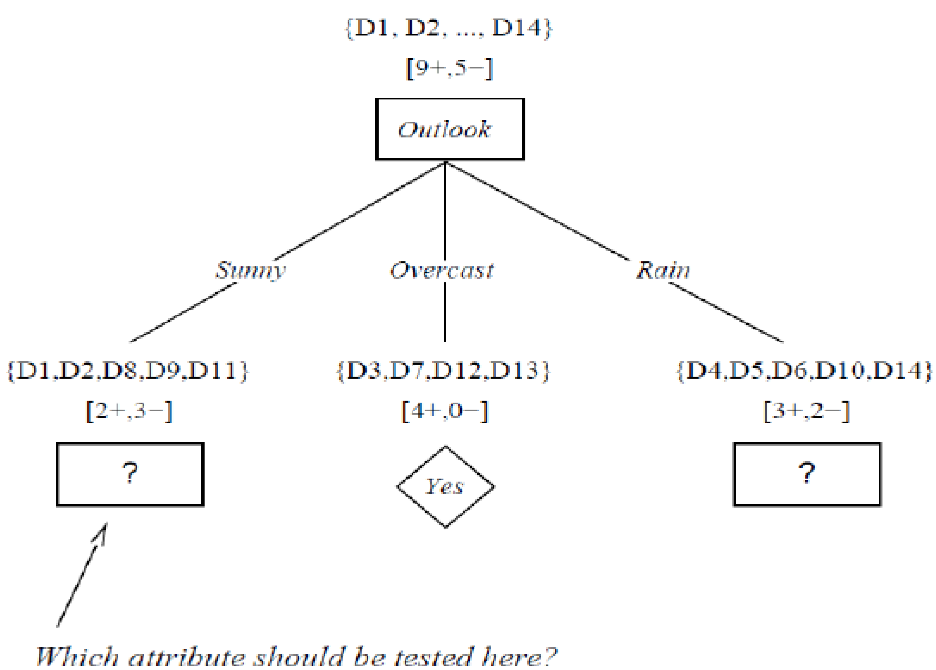
- Working on *Outlook=Sunny* node:

  $Gain(S_{Sunny}, Humidity) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$

  $Gain(S_{Sunny}, Wind) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0.019$

  $Gain(S_{Sunny}, Temp.) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$
- *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 Play Tennis Data

```
import pandas as pd
from pandas import DataFrame
df_tennis = DataFrame.from_csv('C:\\Users\\Dr.Thyagaraju\\Desktop\\Data\\PlayTennis.csv')
df_tennis
```

Output :

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong
7	No	Sunny	Mild	High	Weak
8	Yes	Sunny	Cool	Normal	Weak
9	Yes	Rain	Mild	Normal	Weak
10	Yes	Sunny	Mild	Normal	Strong
11	Yes	Overcast	Mild	High	Strong
12	Yes	Overcast	Hot	Normal	Weak
13	No	Rain	Mild	High	Strong

Entropy of the Training Data Set

```
def entropy(probs): # Calulate the Entropy of given probability
import math
return sum( [-prob*math.log(prob, 2) for prob in probs] )
def entropy_of_list(a_list): # Entropy calculation of list of discrete values (YES/NO)
from collections import Counter
cnt = Counter(x for x in a_list)
```

```
print("No and Yes Classes:",a_list.name,cnt)
num_instances = len(a_list)*1.0
probs = [x / num_instances for x in cnt.values()]
return entropy(probs) # Call Entropy:
# The initial entropy of the YES/NO attribute for our dataset.
#print(df_tennis['PlayTennis'])
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
print("Entropy of given PlayTennis Data Set:",total_entropy)
```

Output :

No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Entropy of given PlayTennis Data Set: 0.9402859586706309

Information Gain of Attributes

```
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
print("Information Gain Calculation of ",split_attribute_name)
'''
Takes a DataFrame of attributes,and quantifies the entropy of a target attribute after
performing a split along the values of another attribute.
'''
# Split Data by Possible Vals of Attribute:
df_split = df.groupby(split_attribute_name)
#print(df_split.groups)
for name,group in df_split:
print(name)
print(group)
# Calculate Entropy for Target Attribute, as well as
# Proportion of Obs in Each Data-Split
nobs = len(df.index) * 1.0
#print("NOBS",nobs)
df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs]
})[target_attribute_name]
#print("DFAGGENT",df_agg_ent)
df_agg_ent.columns = ['Entropy', 'PropObservations']
#if trace: # helps understand what fxn is doing:
# print(df_agg_ent)
# Calculate Information Gain:
new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
```

```
old_entropy = entropy_of_list(df[target_attribute_name])
 return old_entropy - new_entropy
print('Info-gain for Outlook is :'+str( information_gain(df_tennis, 'Outlook', 'PlayTennis')),"\n")
print('\n Info-gain for Humidity is: ' + str( information_gain(df_tennis, 'Humidity',
'PlayTennis')),"\n")
print('\n Info-gain for Wind is:' + str( information_gain(df_tennis, 'Wind', 'PlayTennis')),"\n")
print('\n Info-gain for Temperature is:' + str( information_gain(df_tennis,
'Temperature','PlayTennis')),"\n")
```

Output :

```
Information Gain Calculation of  Outlook
Overcast
   PlayTennis   Outlook Temperature Humidity    Wind
2          Yes  Overcast          Hot     High    Weak
6          Yes  Overcast         Cool   Normal  Strong
11         Yes  Overcast         Mild     High  Strong
12         Yes  Overcast          Hot   Normal    Weak
Rain
   PlayTennis Outlook Temperature Humidity    Wind
3          Yes    Rain        Mild     High    Weak
4          Yes    Rain        Cool   Normal    Weak
5           No    Rain        Cool   Normal  Strong
9          Yes    Rain        Mild   Normal    Weak
13          No    Rain        Mild     High  Strong
Sunny
   PlayTennis Outlook Temperature Humidity    Wind
0           No   Sunny         Hot     High    Weak
1           No   Sunny         Hot     High  Strong
7           No   Sunny        Mild     High    Weak
8          Yes   Sunny        Cool   Normal    Weak
10         Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 4})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Info-gain for Outlook is :0.246749819774
```

Info-gain for Humidity is: 0.151835501362

Info-gain for Wind is:0.0481270304083

Info-gain for Temperature is:0.029222565659

ID3 Algorithm

```
def id3(df, target_attribute_name, attribute_names, default_class=None):
## Tally target attribute:
from collections import Counter
cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO
## First check: Is this split of the dataset homogeneous?
if len(cnt) == 1:
return next(iter(cnt))
## Second check: Is this split of the dataset empty?
# if yes, return a default value
elif df.empty or (not attribute_names):
 return default_class
## Otherwise: This dataset is ready to be divvied up!
else:
# Get Default Value for next recursive call of this function:
default_class = max(cnt.keys()) #[index_of_max] # most common value of target attribute in
dataset
# Choose Best Attribute to split on:
gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names]
index_of_max = gainz.index(max(gainz))
best_attr = attribute_names[index_of_max]
# Create an empty tree, to be populated in a moment
tree = {best_attr:{}}
remaining_attribute_names = [i for i in attribute_names if i != best_attr]
# Split dataset
# On each split, recursively call this algorithm.
# populate the empty tree with subtrees, which
# are the result of the recursive call
for attr_val, data_subset in df.groupby(best_attr):
subtree = id3(data_subset,
target_attribute_name,
remaining_attribute_names,
default_class)
tree[best_attr][attr_val] = subtree
return tree
```

Predicting Attributes

```
# Get Predictor Names (all but 'class')
attribute_names = list(df_tennis.columns)
```

```
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis') #Remove the class attribute
print("Predicting Attributes:", attribute_names)
```

Output :

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']

Tree Construction

```
# Run Algorithm:
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
```

Output

```
Information Gain Calculation of  Outlook
Overcast
   PlayTennis   Outlook Temperature Humidity    Wind
2         Yes  Overcast         Hot     High    Weak
6         Yes  Overcast        Cool   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
12        Yes  Overcast         Hot   Normal    Weak
Rain
   PlayTennis Outlook Temperature Humidity    Wind
3         Yes    Rain        Mild     High    Weak
4         Yes    Rain        Cool   Normal    Weak
5          No    Rain        Cool   Normal  Strong
9         Yes    Rain        Mild   Normal    Weak
13         No    Rain        Mild     High  Strong
Sunny
   PlayTennis Outlook Temperature Humidity    Wind
0          No   Sunny         Hot     High    Weak
1          No   Sunny         Hot     High  Strong
7          No   Sunny        Mild     High    Weak
8         Yes   Sunny        Cool   Normal    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 4})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Information Gain Calculation of  Temperature
Cool
  PlayTennis   Outlook Temperature Humidity    Wind
4        Yes      Rain        Cool   Normal    Weak
5         No      Rain        Cool   Normal  Strong
6        Yes  Overcast        Cool   Normal  Strong
8        Yes     Sunny        Cool   Normal    Weak
Hot
   PlayTennis   Outlook Temperature Humidity    Wind
0          No     Sunny         Hot     High    Weak
1          No     Sunny         Hot     High  Strong
2         Yes  Overcast         Hot     High    Weak
12        Yes  Overcast         Hot   Normal    Weak
Mild
   PlayTennis   Outlook Temperature Humidity    Wind
3         Yes      Rain        Mild     High    Weak
```

```
7           No      Sunny        Mild      High     Weak
9          Yes       Rain        Mild    Normal     Weak
10         Yes      Sunny        Mild    Normal   Strong
11         Yes   Overcast        Mild      High   Strong
13          No       Rain        Mild      High   Strong
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 4, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Information Gain Calculation of  Humidity
High
   PlayTennis   Outlook Temperature Humidity    Wind
0           No      Sunny         Hot      High     Weak
1           No      Sunny         Hot      High   Strong
2          Yes   Overcast         Hot      High     Weak
3          Yes       Rain        Mild      High     Weak
7           No      Sunny        Mild      High     Weak
11         Yes   Overcast        Mild      High   Strong
13          No       Rain        Mild      High   Strong
Normal
   PlayTennis   Outlook Temperature Humidity    Wind
4          Yes       Rain        Cool    Normal     Weak
5           No       Rain        Cool    Normal   Strong
6          Yes   Overcast        Cool    Normal   Strong
8          Yes      Sunny        Cool    Normal     Weak
9          Yes       Rain        Mild    Normal     Weak
10         Yes      Sunny        Mild    Normal   Strong
12         Yes   Overcast         Hot    Normal     Weak
No and Yes Classes: PlayTennis Counter({'No': 4, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Information Gain Calculation of  Wind
Strong
   PlayTennis   Outlook Temperature Humidity    Wind
1           No      Sunny         Hot      High   Strong
5           No       Rain        Cool    Normal   Strong
6          Yes   Overcast        Cool    Normal   Strong
10         Yes      Sunny        Mild    Normal   Strong
11         Yes   Overcast        Mild      High   Strong
13          No       Rain        Mild      High   Strong
Weak
   PlayTennis   Outlook Temperature Humidity  Wind
0           No      Sunny         Hot      High  Weak
2          Yes   Overcast         Hot      High  Weak
3          Yes       Rain        Mild      High  Weak
```

```
4          Yes       Rain         Cool    Normal   Weak
7           No      Sunny         Mild      High   Weak
8          Yes      Sunny         Cool    Normal   Weak
9          Yes       Rain         Mild    Normal   Weak
12         Yes   Overcast          Hot    Normal   Weak
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 6, 'No': 2})
No and Yes Classes: PlayTennis Counter({'Yes': 9, 'No': 5})
Information Gain Calculation of  Temperature
Cool
  PlayTennis Outlook Temperature Humidity    Wind
4         Yes     Rain         Cool    Normal    Weak
5          No     Rain         Cool    Normal  Strong
Mild
   PlayTennis Outlook Temperature Humidity    Wind
3          Yes     Rain         Mild      High    Weak
9          Yes     Rain         Mild    Normal    Weak
13          No     Rain         Mild      High  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 1, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
Information Gain Calculation of  Humidity
High
   PlayTennis Outlook Temperature Humidity    Wind
3          Yes     Rain         Mild      High    Weak
13          No     Rain         Mild      High  Strong
Normal
  PlayTennis Outlook Temperature Humidity    Wind
4         Yes     Rain         Cool    Normal    Weak
5          No     Rain         Cool    Normal  Strong
9         Yes     Rain         Mild    Normal    Weak
No and Yes Classes: PlayTennis Counter({'Yes': 1, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 2, 'No': 1})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
Information Gain Calculation of  Wind
Strong
   PlayTennis Outlook Temperature Humidity    Wind
5           No     Rain         Cool    Normal  Strong
13          No     Rain         Mild      High  Strong
Weak
  PlayTennis Outlook Temperature Humidity  Wind
3         Yes     Rain         Mild      High  Weak
4         Yes     Rain         Cool    Normal  Weak
9         Yes     Rain         Mild    Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 2})
```

```
No and Yes Classes: PlayTennis Counter({'Yes': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 3, 'No': 2})
Information Gain Calculation of  Temperature
Cool
  PlayTennis Outlook Temperature Humidity  Wind
8        Yes   Sunny        Cool   Normal  Weak
Hot
  PlayTennis Outlook Temperature Humidity    Wind
0         No   Sunny         Hot     High    Weak
1         No   Sunny         Hot     High  Strong
Mild
   PlayTennis Outlook Temperature Humidity    Wind
7          No   Sunny        Mild     High    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2})
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
Information Gain Calculation of  Humidity
High
  PlayTennis Outlook Temperature Humidity    Wind
0         No   Sunny         Hot     High    Weak
1         No   Sunny         Hot     High  Strong
7         No   Sunny        Mild     High    Weak
Normal
   PlayTennis Outlook Temperature Humidity    Wind
8         Yes   Sunny        Cool   Normal    Weak
10        Yes   Sunny        Mild   Normal  Strong
No and Yes Classes: PlayTennis Counter({'No': 3})
No and Yes Classes: PlayTennis Counter({'Yes': 2})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
Information Gain Calculation of  Wind
Strong
   PlayTennis Outlook Temperature Humidity    Wind
1          No   Sunny         Hot     High  Strong
10        Yes   Sunny        Mild   Normal  Strong
Weak
  PlayTennis Outlook Temperature Humidity  Wind
0         No   Sunny         Hot     High  Weak
7         No   Sunny        Mild     High  Weak
8        Yes   Sunny        Cool   Normal  Weak
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 3, 'Yes': 2})
```

```
The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

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

Algorithm:

function BackProp $(D, \eta, n_{in}, n_{hidden}, n_{out})$

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

Make a feed-forward network with n_{in}, n_{hidden} e n_{out} units
Initialize all the weight to short randomly number (es. [-0.05 0.05])
Repeat until termination condition are verifyed:
For any sample in D:
Forward propagate the network computing the output o_u of every unit u of the network
Back propagate the errors onto the network:
– For every output unit k, compute the error δ_k: $\delta_k = o_k(1-o_k)(t_k - o_k)$
– For every hidden unit h compute the error δ_h: $\delta_h = o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$
– Update the network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}$, *where* $\Delta w_{ji} = \eta \delta_j x_{ji}$

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

Source Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
return x * (1 - x)

#Variable initialization
epoch=7000              #Setting training iterations
```

```
lr=0.1                          #Setting learning rate
inputlayer_neurons = 2          #number of features in data set
hiddenlayer_neurons = 3         #number of hidden layers neurons
output_neurons = 1              #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

#Forward Propogation
hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+ bout
output = sigmoid(outinp)

#Backpropagation
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)   #how much hidden layer wts contributed to
error
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr  #dotproduct of nextlayer error and current layer op
# bout += np.sum(d_output, axis=0,keepdims=True) *lr
wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

output

Input:

[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]

Actual Output:

[[0.92]
[0.86]
[0.89]]

Predicted Output:

[[0.89559591]
[0.88142069]
[0.8928407]]

****

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.

Bayesian 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

$$\text{Posterior} = \frac{\text{Likelihood x Prior}}{\text{Evidence}}$$

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

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

Source Code:

```
import csv
import random
import math

# 1.Data Handling
# 1.1 Loading the Data from csv file of Pima indians diabetes dataset.
def loadcsv(filename):
lines = csv.reader(open(filename, "r"))
dataset = list(lines)
for i in range(len(dataset)):
# converting the attributes from string to floating point numbers
dataset[i] = [float(x) for x in dataset[i]]
return dataset

#1.2 Splitting the Data set into Training Set
def splitDataset(dataset, splitRatio):
trainSize = int(len(dataset) * splitRatio)
trainSet = []
copy = list(dataset)
while len(trainSet) < trainSize:
index = random.randrange(len(copy)) # random index
trainSet.append(copy.pop(index))
return [trainSet, copy]

#2.Summarize Data
#The naive bayes model is comprised of a
#summary of the data in the training dataset.
#This summary is then used when making predictions.
#involves the mean and the standard deviation for each attribute, by class value

#2.1: Separate Data By Class
#Function to categorize the dataset in terms of classes
#The function assumes that the last attribute (-1) is the class value.
#The function returns a map of class values to lists of data instances.
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

#The mean is the central middle or central tendency of the data,
# and we will use it as the middle of our gaussian distribution
# when calculating probabilities
#2.2 : Calculate Mean
 def mean(numbers):
return sum(numbers)/float(len(numbers))

#The standard deviation describes the variation of spread of the data,
#and we will use it to characterize the expected spread of each attribute
#in our Gaussian distribution when calculating probabilities.
#2.3 : Calculate Standard Deviation
def stdev(numbers):
avg = mean(numbers)
variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
return math.sqrt(variance)

#2.4 : Summarize Dataset
#Summarize Data Set for a list of instances (for a class value)
#The zip function groups the values for each attribute across our data instances
#into their own lists so that we can compute the mean and standard deviation values
#for the attribute.
def summarize(dataset):
summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
del summaries[-1]
return summaries

#2.5 : Summarize Attributes By Class
#We can pull it all together by first separating our training dataset into
#instances grouped by class.Then calculate the summaries for each attribute.
def summarizeByClass(dataset):
separated = separateByClass(dataset)
summaries = {}
```

```
for classValue, instances in separated.items():
summaries[classValue] = summarize(instances)
return summaries

#3.Make Prediction
#3.1 Calculate Probaility Density Function
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

#3.2 Calculate Class Probabilities
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

#3.3 Prediction : look for the largest probability and return the associated class
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

#4.Make Predictions
# Function which return predictions for list of predictions
# For each instance
def getPredictions(summaries, testSet):
predictions = []
for i in range(len(testSet)):
result = predict(summaries, testSet[i])
```

```
predictions.append(result)
return predictions

#5. Computing Accuracy
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

#Main Function
def main():
filename = 'C:\\Users\\Dr.Thyagaraju\\Desktop\\Data\\pima-indians-diabetes.csv'
splitRatio = 0.67
dataset = loadcsv(filename)
#print("\n The Data Set :\n",dataset)
print("\n The length of the Data Set : ",len(dataset))
print("\n The Data Set Splitting into Training and Testing \n")
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('\n Number of Rows in Training Set:{0} rows'.format(len(trainingSet)))
print('\n Number of Rows in Testing Set:{0} rows'.format(len(testSet)))
print("\n First Five Rows of Training Set:\n")
for i in range(0,5):
 print(trainingSet[i],"\n")
print("\n First Five Rows of Testing Set:\n")
for i in range(0,5):
print(testSet[i],"\n")
# prepare model
summaries = summarizeByClass(trainingSet)
print("\n Model Summaries:\n",summaries)
# test model
predictions = getPredictions(summaries, testSet)
print("\nPredictions:\n",predictions)
accuracy = getAccuracy(testSet, predictions)
print('\n Accuracy: {0}%'.format(accuracy))
main()
```

Output:

```
The length of the Data Set :  768

 The Data Set Splitting into Training and Testing

 Number of Rows in Training Set:514 rows
 Number of Rows in Testing Set:254 rows
 First Five Rows of Training Set:

[4.0, 116.0, 72.0, 12.0, 87.0, 22.1, 0.463, 37.0, 0.0]
[0.0, 84.0, 64.0, 22.0, 66.0, 35.8, 0.545, 21.0, 0.0]
[0.0, 162.0, 76.0, 36.0, 0.0, 49.6, 0.364, 26.0, 1.0]
[10.0, 101.0, 86.0, 37.0, 0.0, 45.6, 1.136, 38.0, 1.0]
[5.0, 78.0, 48.0, 0.0, 0.0, 33.7, 0.654, 25.0, 0.0]
```

First Five Rows of Testing Set:

```
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]
[4.0, 110.0, 92.0, 0.0, 0.0, 37.6, 0.191, 30.0, 0.0]
[10.0, 139.0, 80.0, 0.0, 0.0, 27.1, 1.441, 57.0, 0.0]
[7.0, 100.0, 0.0, 0.0, 0.0, 30.0, 0.484, 32.0, 1.0]
```

Predictions:

```
 [0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.
0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0,
1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0,
1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0]
```

Accuracy: 80.31496062992126%

****

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.

Procedure:

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

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

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

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

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

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

Source Code :

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

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

# Output of count vectoriser is a sparse matrix
# CountVectorizer - stands for 'feature extraction'

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])
print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
print('\n Classification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
```

```
pred = 'pos' if p==1 else 'neg'
 print('%s -> %s ' % (doc, pred))
```

#printing accuracy metrics

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

RESULTS :

Data set

I love this sandwich,pos
This is an amazing place,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,neg
I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy,neg
My boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg

Output

Total instances in the dataset: 18
The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos

I feel very good about these beers , pos
This is my best work , pos
What an awesome view , pos
Dataset is split into Training and Testing samples
Total training instances : 13
Total testing instances : 5
Total features extracted using CountVectorizer: 46
Features for first 5 training instances are listed below
am amazing an and awesome bad ... view we went what will with
0 1 0 0 1 0 0 ... 0 0 0 0 0 0
1 0 0 0 0 0 0 ... 0 0 0 0 0 0
2 0 0 1 0 1 0 ... 1 0 0 1 0 0
3 0 1 1 0 0 0 ... 0 0 0 0 0 0
4 0 0 0 0 0 1 ... 0 0 0 0 0 0

Classification results of testing samples are given below

This is an awesome place -> pos
I love this sandwich -> pos
I love to dance -> pos
This is my best work -> pos
I feel very good about these beers -> pos

Accuracy metrics
Accuracy of the classifer is 0.4
Recall : 0.4
Precison : 1.0
Confusion matrix
[[0 0]
[3 2]]
****

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 :

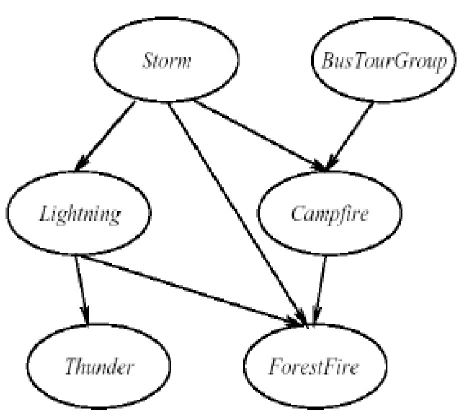
Bayesian Network (BAYESIAN BELIEF NETWORKS

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

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

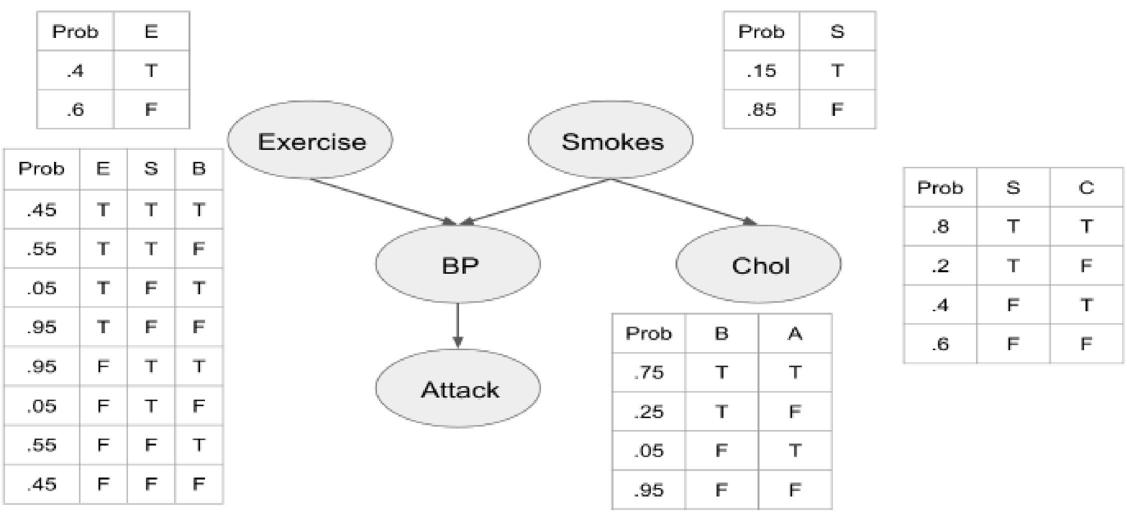
- Represents joint probability distribution over all variables
    - e.g., *P*(*Storm*, *BusTourGroup*, . . . , *ForestFire*)
    - in general,

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

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:



Prob	E
.4	T
.6	F

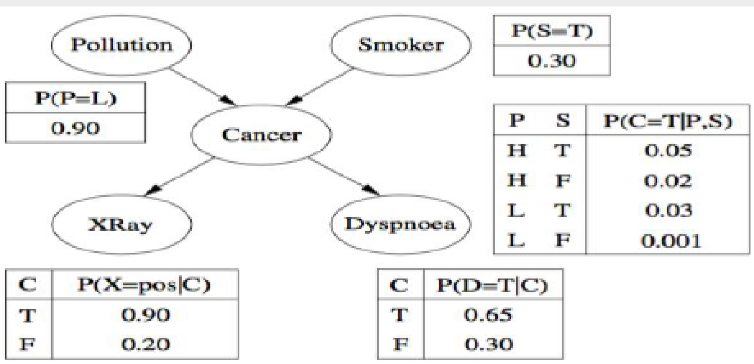
Prob	S
.15	T
.85	F

Prob	E	S	B
.45	T	T	T
.55	T	T	F
.05	T	F	T
.95	T	F	F
.95	F	T	T
.05	F	T	F
.55	F	F	T
.45	F	F	F

Prob	S	C
.8	T	T
.2	T	F
.4	F	T
.6	F	F

Prob	B	A
.75	T	T
.25	T	F
.05	F	T
.95	F	F

Example2 :



P(S=T)
0.30

P(P=L)
0.90

P	S	P(C=T|P,S)
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

C	P(X=pos|C)
T	0.90
F	0.20

C	P(D=T|C)
T	0.65
F	0.30

FIGURE 2.1
A BN for the lung cancer problem.

Source Code:

7.1. Constructing a Bayesian Network considering Medical Data

7.1.1 Defining a Structure with nodes and edges

```
# Starting with defining the network structure
from pgmpy.models import BayesianModel
cancer_model = BayesianModel([('Pollution', 'Cancer'),
('Smoker', 'Cancer'),
('Cancer', 'Xray'),
('Cancer', 'Dyspnoea')])
cancer_model.nodes()
cancer_model.edges()
cancer_model.get_cpds()
```

7.1.2 Creation of Conditional Probability Table

```
# Now defining the parameters.
from pgmpy.factors.discrete import TabularCPD
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
values=[[0.03, 0.05, 0.001, 0.02],
[0.97, 0.95, 0.999, 0.98]],
evidence=['Smoker', 'Pollution'],
evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
values=[[0.9, 0.2], [0.1, 0.8]],
evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
values=[[0.65, 0.3], [0.35, 0.7]],
evidence=['Cancer'], evidence_card=[2])
```

7.1.3 Associating Conditional probabilities with the Bayesian Structure

```
# Associating the parameters with the model structure.
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
# Checking if the cpds are valid for the model.
```

```
cancer_model.check_model()

# Doing some simple queries on the network
cancer_model.is_active_trail('Pollution', 'Smoker')
cancer_model.is_active_trail('Pollution', 'Smoker', observed=['Cancer'])
cancer_model.get_cpds()
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
print(cancer_model.get_cpds('Cancer'))
```

7.1.4 Determining the Local independencies

```
cancer_model.local_independencies('Xray')
cancer_model.local_independencies('Pollution')
cancer_model.local_independencies('Smoker')
cancer_model.local_independencies('Dyspnoea')
cancer_model.local_independencies('Cancer')
cancer_model.get_independencies()
```

7.1.5.Inferencing with Bayesian Network

```
# Doing exact inference using Variable Elimination
from pgmpy.inference import VariableElimination
cancer_infer = VariableElimination(cancer_model)
# Computing the probability of bronc given smoke.
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q['Cancer'])
# Computing the probability of bronc given smoke.
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q['Cancer'])
# Computing the probability of bronc given smoke.
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})
print(q['Cancer'])
```

7.2 Diagnosis of heart patients using standard Heart Disease Data Set

```
import numpy as np
from urllib.request import urlopen
import urllib
```

```
import matplotlib.pyplot as plt # Visuals
import seaborn as sns
import sklearn as skl
import pandas as pd
```

7.2.1 Importing Heart Disease Data Set and Customizing

```
Cleveland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hungarian.data'
 #Hungarian_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hungarian.data'
#Switzerland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.switzerland.data'
np.set_printoptions(threshold=np.nan) #see a whole array when we output it
names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope',
'ca', 'thal', 'heartdisease']
heartDisease = pd.read_csv(urlopen(Cleveland_data_URL), names = names) #gets Cleveland data
#HungarianHeartDisease = pd.read_csv(urlopen(Hungarian_data_URL), names = names) #gets Hungary data
#SwitzerlandHeartDisease = pd.read_csv(urlopen(Switzerland_data_URL), names = names) #gets Switzerland data
#datatemp = [ClevelandHeartDisease, HungarianHeartDisease, SwitzerlandHeartDisease] #combines all arrays into a list
#heartDisease = pd.concat(datatemp)#combines list into one array
heartDisease.head()
del heartDisease['ca']
del heartDisease['slope']
del heartDisease['thal']
del heartDisease['oldpeak']
heartDisease = heartDisease.replace('?', np.nan)
heartDisease.dtypes
heartDisease.columns
```

7.2.2 Modeling Heart Disease Data

```
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
```

```
('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])
# Learing CPDs using Maximum Likelihood Estimators
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
#for cpd in model.get_cpds():
# print("CPD of {variable}:".format(variable=cpd.variable))
# print(cpd)
print(model.get_cpds('age'))
print(model.get_cpds('chol'))
print(model.get_cpds('sex'))
model.get_independencies()
```

7.2.3.Inferencing with Bayesian Network

```
# Doing exact inference using Variable Elimination
from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)
# Computing the probability of bronc given smoke.
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
```

heartdisease	phi(heartdisease)
heartdisease_0	0.6333
heartdisease_1	0.3667

In [35]:

```
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 10
0})
print(q['heartdisease'])
```

heartdisease	phi(heartdisease)
heartdisease_0	1.0000
heartdisease_1	0.0000

****

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.

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

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 $h' = <\mu'_1, \mu'_2>$.

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

K Means Algorithm

- 1. The sample space is initially partitioned into K clusters and the observations are randomly assigned to the clusters.
- 2. For each sample:
    - Calculate the distance from the observation to the centroid of the cluster.
    - IF the sample is closest to its own cluster THEN leave it ELSE select another cluster.
- 3. Repeat steps 1 and 2 untill no observations are moved from one cluster to another

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

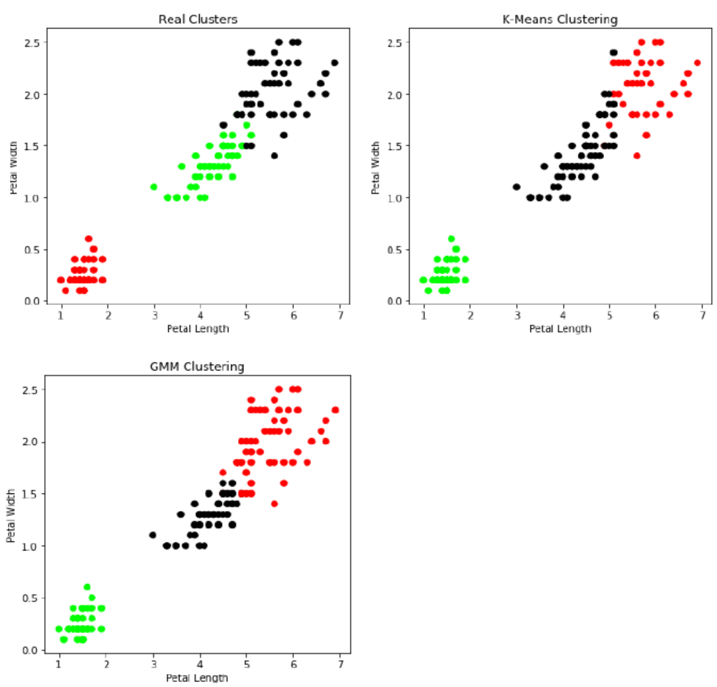
Source Code:

```
import matplotlib.pyplot as plt
 from sklearn import datasets
 from sklearn.cluster import KMeans
 import pandas as pd
 import numpy as np

 # import some data to play with
 iris = datasets.load_iris()
 X = pd.DataFrame(iris.data)
 X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
 y = pd.DataFrame(iris.target)
 y.columns = ['Targets']

 # Build the K Means Model
 model = KMeans(n_clusters=3)
 model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

 # # Visualise the clustering results
 plt.figure(figsize=(14,14))
 colormap = np.array(['red', 'lime', 'black'])
 # Plot the Original Classifications using Petal features
 plt.subplot(2, 2, 1)
 plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
 plt.title('Real Clusters')
 plt.xlabel('Petal Length')
 plt.ylabel('Petal Width')

 # Plot the Models Classifications
 plt.subplot(2, 2, 2)
 plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
 plt.title('K-Means Clustering')
 plt.xlabel('Petal Length')
 plt.ylabel('Petal Width')

 # General EM for GMM
 from sklearn import preprocessing
 # transform your data such that its distribution will have a
```

```
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Observation: The GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.')
```

Output:

Sample Output



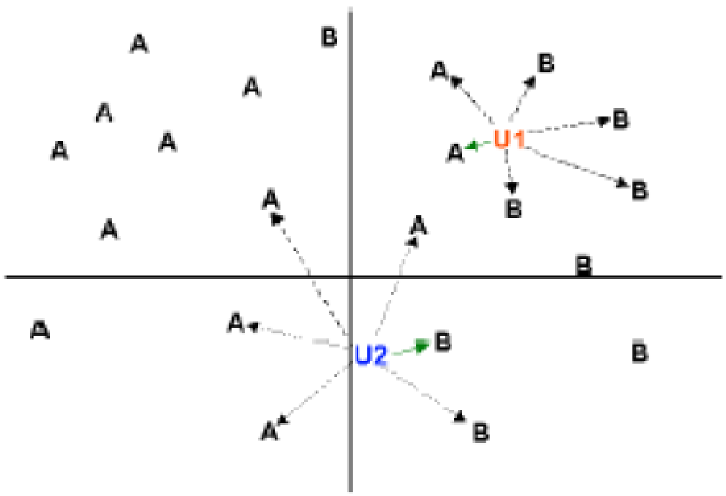
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.

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.

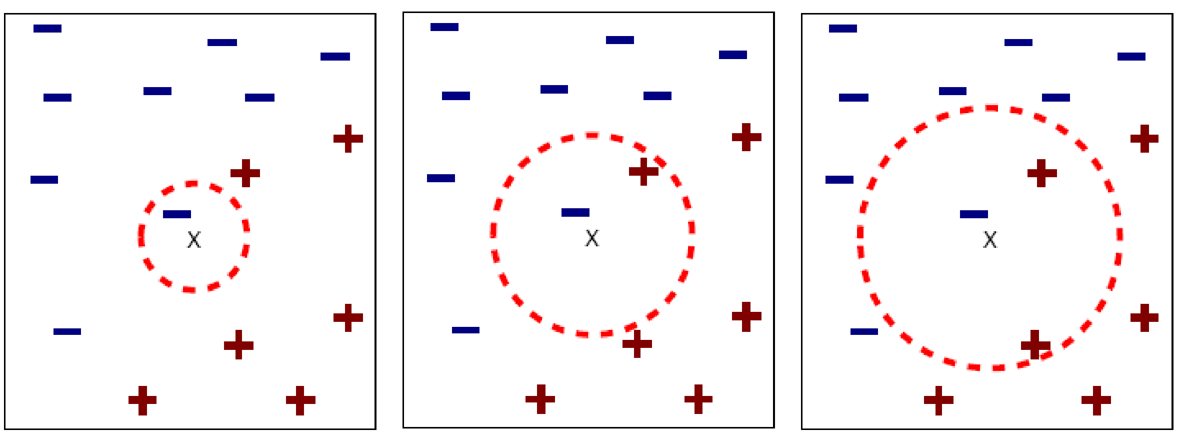
Algorithm :

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

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.

Algorithm:

Input: Let m be the number of training data samples. Let p be an unknown point.

Method:

1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
4. Return the majority label among S.

Source Code :

```
# Python program to demonstrate # KNN classification algorithm # on IRIS dataset
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))
print("\n IRIS DATA :\n",iris_dataset["data"])
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],
random_state=0)
print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)
x_new = np.array([[5, 2.9, 1, 0.2]])
```

```
print("\n XNEW \n",x_new)
prediction = kn.predict(x_new)
print("\n Predicted target value: {}\n".format(prediction))
print("\n Predicted feature name: {}\n".format
(iris_dataset["target_names"][prediction]))
i=1
x= X_test[i]
x_new = np.array([x])
print("\n XNEW \n",x_new)
for i in range(len(X_test)):
x = X_test[i]
x_new = np.array([x])
prediction = kn.predict(x_new)
print("\n Actual : {0} {1}, Predicted
:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset["target_
names"][prediction]))
print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

<u>Output :</u>

```
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 0 setosa, Predicted :[0]['setosa']
--------
Actual : 1 versicolor, Predicted :[2]['virginica']

TEST SCORE[ACCURACY]: 0.97
```

****

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.

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 .

Algorithm

1. **Read the Given data Sample to X and the curve (linear or non linear) 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**

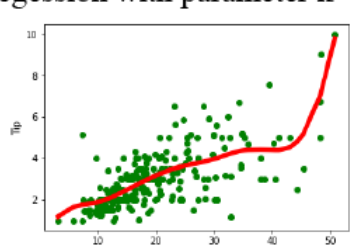
Source Code:

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
m,n = np1.shape(xmat)
weights = np1.mat(np1.eye((m)))
for j in range(m):
diff = point - X[j]
```

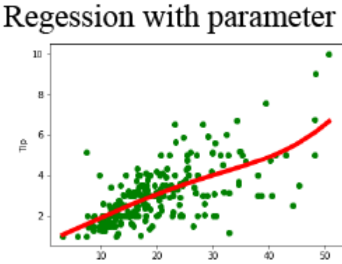
```
weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
return weights
def localWeight(point,xmat,ymat,k):
wei = kernel(point,xmat,k)
W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))
return W
def localWeightRegression(xmat,ymat,k):
m,n = np1.shape(xmat)
ypred = np1.zeros(m)
for i in range(m):
ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
return ypred

# load data points
data = pd.read_csv('data10.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))
 #set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

Output:

Regession with parameter k = 3



Regession with parameter k = 9



****