MACHINE LEARNING LAB MANUAL

Problem1 : 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

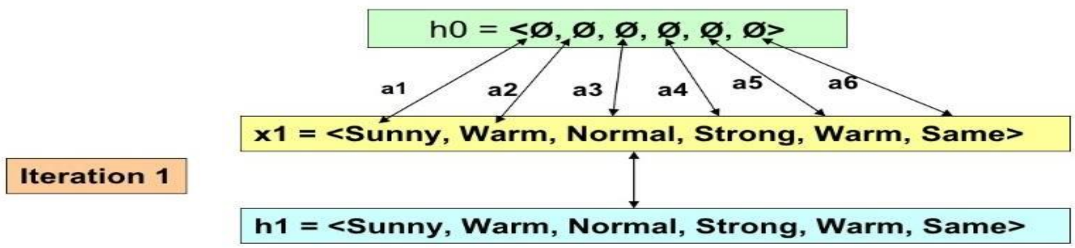
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

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

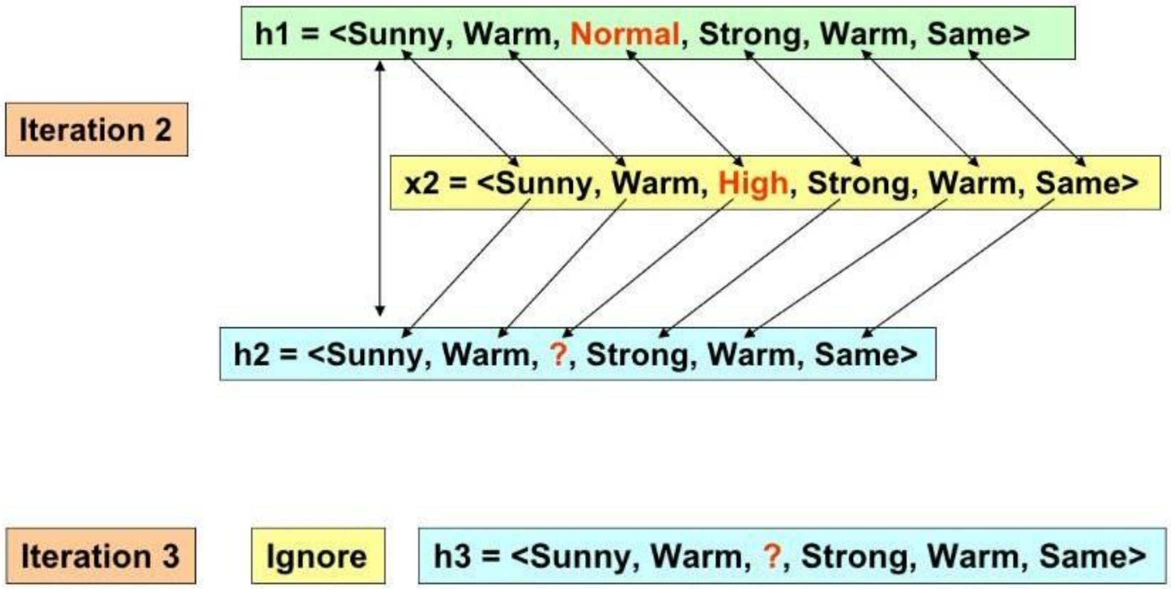
h0 = **<Ø, Ø, Ø, Ø, Ø, Ø>**

Step2 : Find S

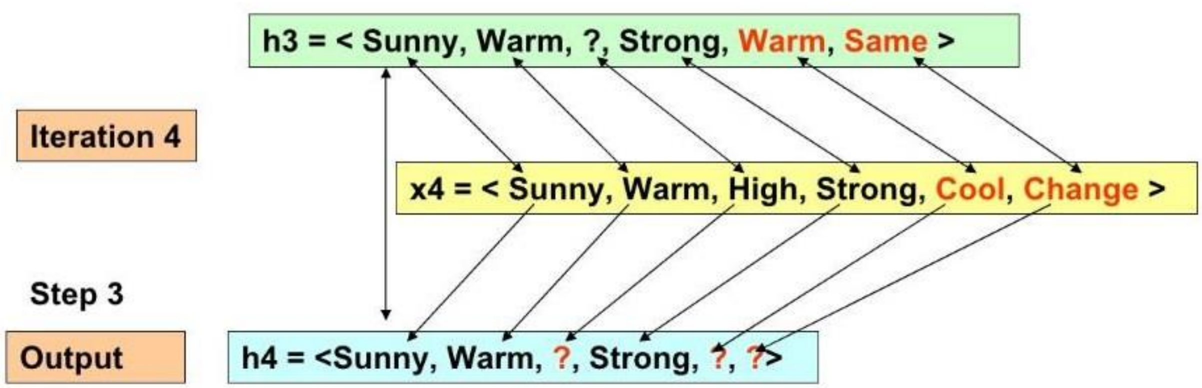
2. For each positive training instance x
   - For each attribute constraint a_i in h
     If the constraint a_i is satisfied by x
     Then do nothing
     Else replace a_i in h by the next more general constraint that is satisfied by x



Step2 : Find S



Iteration 4 and Step 3 : Find S



Source Code of the Program :

```
import csv
#!usr/bin/python
#list creatin
hypo=['ϕ','ϕ','ϕ','ϕ','ϕ','ϕ'];

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);
```

Training_examples.csv

Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Output :

```
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
 ['ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ']
```

```
['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', '?', '?']
>>>
```

Program2 : 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*

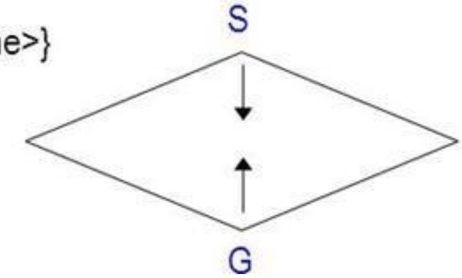
Illustration :

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

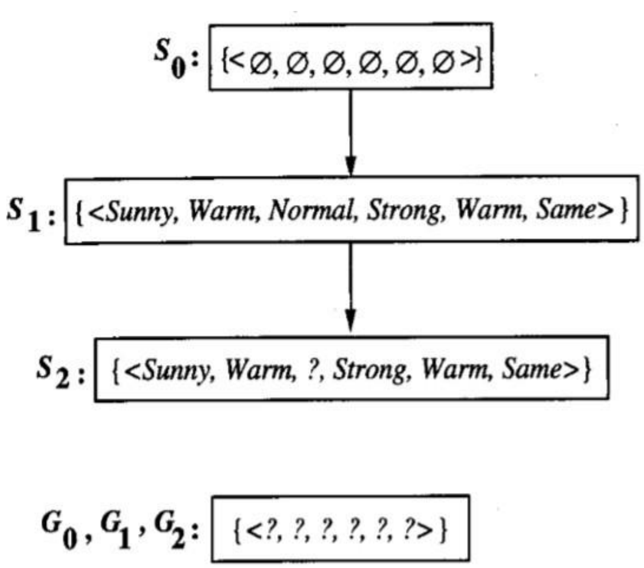
$S_0 = \{<\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing>\}$
$G_0 = \{<?, ?, ?, ?, ?, ?>\}$

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

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



Trace1 :



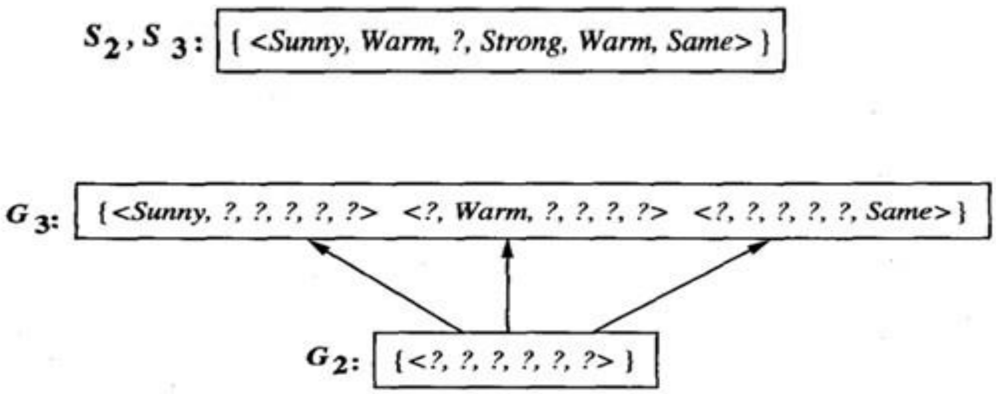
Training examples:

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:

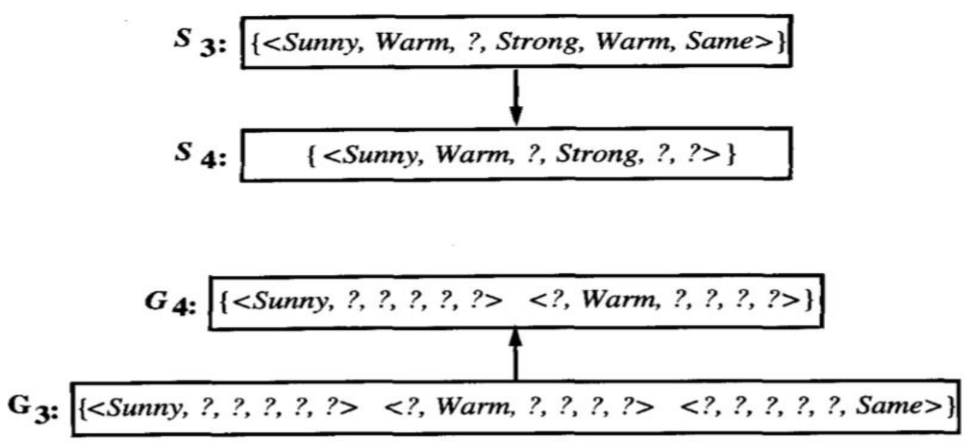


Training Example:

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 :

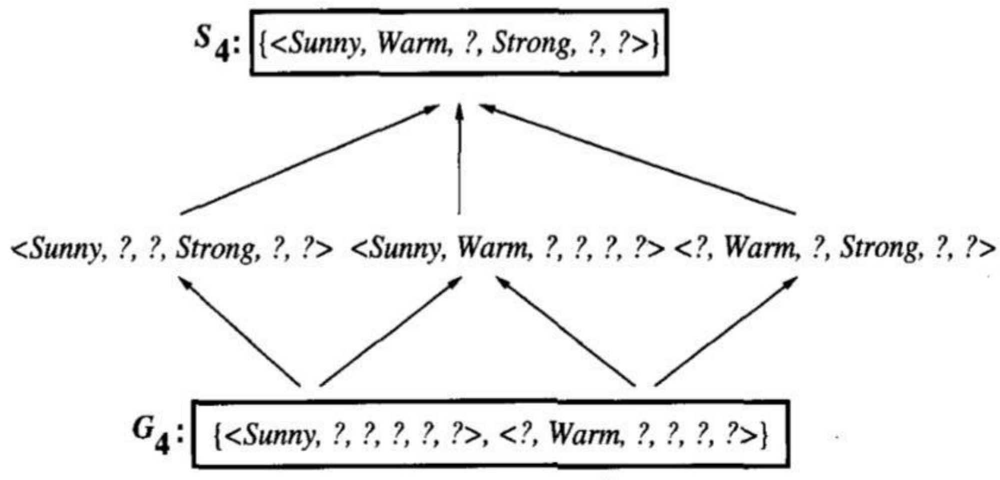


Training Example:

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 ('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('trainingexamples.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]
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):
    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)
```

```
trainingexamples.csv
```

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

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*

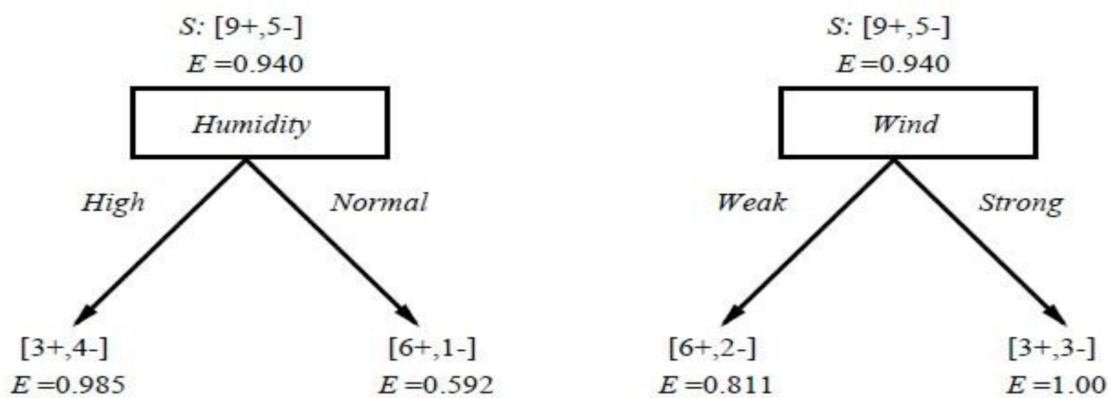
Illustration:

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?



Gain (S, Humidity)

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

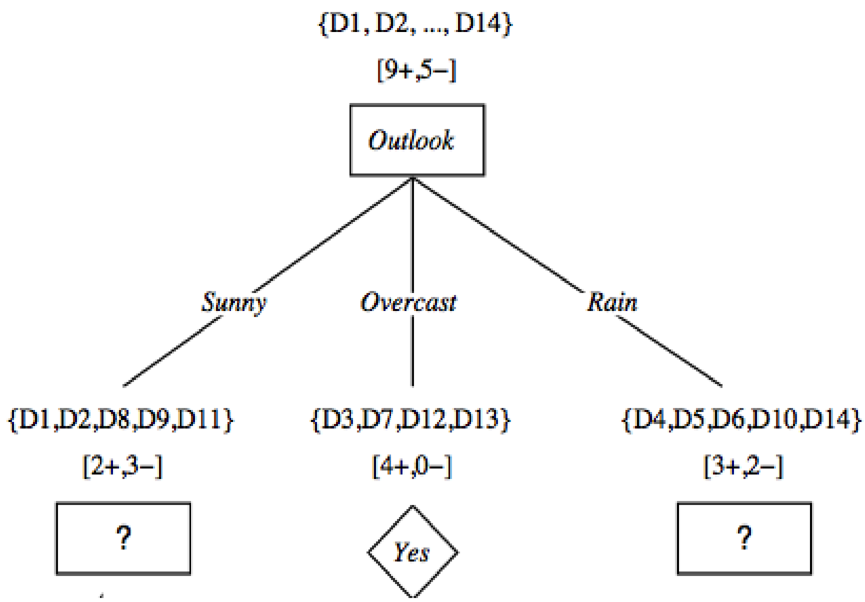
Gain (S, Wind)

= .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



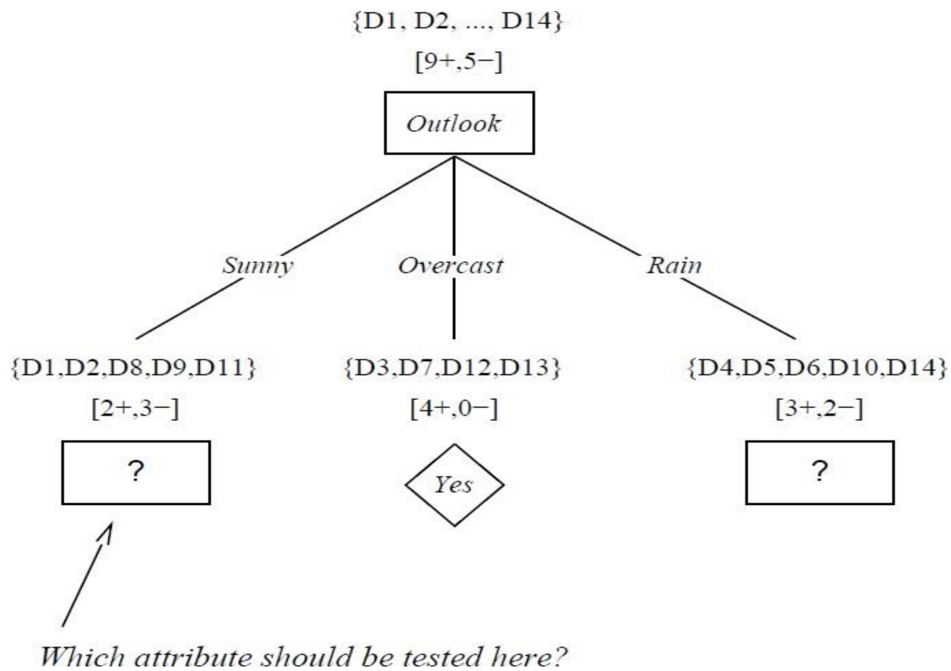
Second step

- 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}

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

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

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

Source Code:

Import Play Tennis Data

```
import pandas as pd
from pandas import DataFrame
df_tennis = DataFrame.from_csv('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 val
ues (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, trac
e=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, l
ambda 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['PropObservation
s'] )
    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, 'Outlo
ok', '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

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})
```

Info-gain for Humidity is: 0.151835501362

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})
```

```
Info-gain for Wind is:0.0481270304083

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})

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 valu
e of target attribute in dataset

        # Choose Best Attribute to split on:
        gainz = [information_gain(df, attr, target_attribute_name) for att
r 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 != be
st_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

In [85]:

```
# 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'}}}}

Classification Accuracy

```
def classify(instance, tree, default=None):
    attribute = next(iter(tree))#tree.keys()[0]
    if instance[attribute] in tree[attribute].keys():
        result = tree[attribute][instance[attribute]]
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default
```

```
df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No'
) )
    # classify func allows for a default arg: when tree doesn't have answe
r for a particular
    # combitation of attribute-values, we can use 'no' as the default gues
s

print('Accuracy is:' + str( sum(df_tennis['PlayTennis']==df_tennis['predic
ted'] ) / (1.0*len(df_tennis.index)) ))

df_tennis[['PlayTennis', 'predicted']]
```

Output :

```
Accuracy is:1.0
```

	PlayTennis	predicted
0	No	No
1	No	No
2	Yes	Yes
3	Yes	Yes
4	Yes	Yes
5	No	No
6	Yes	Yes
7	No	No
8	Yes	Yes
9	Yes	Yes
10	Yes	Yes
11	Yes	Yes
12	Yes	Yes
13	No	No

Classification Accuracy: Training/Testing Set

```
training data = df tennis.iloc[1:-4] # all but last thousand instances
test data  = df tennis.iloc[-4:] # just the last thousand
train_tree = id3(training_data, 'PlayTennis', attribute_names)

test data['predicted2'] = test data.apply(
# <---- test_data source
                                          classify,
                                          axis=1,
                                          args=(train tree,'Yes') ) # <---
- train_data tree


print ('\n\n Accuracy is : ' + str( sum(test data['PlayTennis']==test data
['predicted2'] ) / (1.0*len(test_data.index)) ))
```

Output :

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

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

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

Strong
PlayTennis Outlook Temperature Humidity Wind predicted
1 No Sunny Hot High Strong No
Weak
PlayTennis Outlook Temperature Humidity Wind predicted
7 No Sunny Mild High Weak No
8 Yes Sunny Cool Normal Weak Yes
No and Yes Classes: PlayTennis Counter({'No': 1})
No and Yes Classes: PlayTennis Counter({'No': 1, 'Yes': 1})
No and Yes Classes: PlayTennis Counter({'No': 2, 'Yes': 1})

```
Accuracy is : 0.75
```

Lab Exercise : Apply above Program to clasify the new sample /new data set.

Program4: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets

Algorithm:

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

- 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)

The Backpropagation Algorithm for a feed-forward 2-layer network of sigmoid units, the stochastic version

Idea: Gradient descent over the entire vector of network weights.

Initialize all weights to small random numbers.

Until satisfied, // *stopping criterion* to be (later) defined

for each training example,

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:
   $\delta_h \leftarrow o_h(1 - o_h)\sum_{k \in outputs} w_{kh}\delta_k$
4. update each network weight w_{ji}:
   $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where $\Delta w_{ji} = \eta\delta_j x_{ji}$,
   and x_{ji} is the ith input to unit j.

Source Code:

Below is a small contrived dataset that we can use to test out training our neural network.

X1	X2	Y
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

Below is the complete example. We will use 2 neurons in the hidden layer. It is a binary classification problem (2 classes) so there will be two neurons in the output layer. The network will be trained for 20 epochs with a learning rate of 0.5, which is high because we are training for so few iterations.

```
import random
from math import exp
from random import seed
# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_
inputs + 1)]} for i in range(n_hidden)]
    network.append(hidden layer)
    output_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_
hidden + 1)]} for i in range(n_outputs)]
    network.append(output layer)
    return network

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation
```

```
# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['outp
ut'])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[
j]
            neuron['weights'][-1] += l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
```

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

#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)

i= 1
for layer in network:
    j=1
    for sub in layer:
        print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)
        j=j+1
    i=i+1
```

Output :

```
>epoch=0, lrate=0.500, error=4.763
>epoch=1, lrate=0.500, error=4.558
>epoch=2, lrate=0.500, error=4.316
>epoch=3, lrate=0.500, error=4.035
>epoch=4, lrate=0.500, error=3.733
>epoch=5, lrate=0.500, error=3.428
>epoch=6, lrate=0.500, error=3.132
>epoch=7, lrate=0.500, error=2.850
>epoch=8, lrate=0.500, error=2.588
>epoch=9, lrate=0.500, error=2.348
```

```
>epoch=10, lrate=0.500, error=2.128
>epoch=11, lrate=0.500, error=1.931
>epoch=12, lrate=0.500, error=1.753
>epoch=13, lrate=0.500, error=1.595
>epoch=14, lrate=0.500, error=1.454
>epoch=15, lrate=0.500, error=1.329
>epoch=16, lrate=0.500, error=1.218
>epoch=17, lrate=0.500, error=1.120
>epoch=18, lrate=0.500, error=1.033
>epoch=19, lrate=0.500, error=0.956

 Layer[1] Node[1]:
 {'weights': [-1.435239043819221, 1.8587338175173547, 0.7917644224148094],
'output': 0.029795197360175857, 'delta': -0.006018730117768358}

 Layer[1] Node[2]:
 {'weights': [-0.7704959899742789, 0.8257894037467045, 0.21154103288579731
], 'output': 0.06771641538441577, 'delta': -0.005025585510232048}

 Layer[2] Node[1]:
 {'weights': [2.223584933362892, 1.2428928053374768, -1.3519548925527454],
'output': 0.23499833662766154, 'delta': -0.0422466187950293306}

 Layer[2] Node[2]:
 {'weights': [-2.509732251870173, -0.5925943219491905, 1.259965727484093],
'output': 0.7543931062537561, 'delta': 0.04550706392557862}
```

Predict

Making predictions with a trained neural network is easy enough. We have already seen how to forward-propagate an input pattern to get an output. This is all we need to do to make a prediction. We can use the output values themselves directly as the probability of a pattern belonging to each output class. It may be more useful to turn this output back into a crisp class prediction. We can do this by selecting the class value with the larger probability. This is also called the arg max function. Below is a function named predict() that implements this procedure. It returns the index in the network output that has the largest probability. It assumes that class values have been converted to integers starting at 0.

```python
from math import exp

# 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
```

```python
# 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

# Make a prediction with a network
def predict(network, row):
    outputs = forward propagate(network, row)
    return outputs.index(max(outputs))

# Test making predictions with the network
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]]
network = [[{'weights': [-1.482313569067226, 1.8308790073202204, 1.078381922048799]}, {'weights': [0.23244990332399884, 0.3621998343835864, 0.40289821191094327]}],
    [{'weights': [2.5001872433501404, 0.7887233511355132, -1.1026649757805829]}, {'weights': [-2.429350576245497, 0.8357651039198697, 1.0699217181280656]}]]
for row in dataset:
    prediction = predict(network, row)
    print('Expected=%d, Got=%d' % (row[-1], prediction))
```

```
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
```

Program5: 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

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

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

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

Bayes' rule:

Given a set of variables, X = {x1,x2,x...,xd}, we want to construct the posterior probability for the event Cj among a set of possible outcomes C = {c1,c2,c...,cd} , the Bayes Rule is

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j)\, p(C_j)$$

Since Naive Bayes assumes that the conditional probabilities of the independent variables are statistically independent we can decompose the likelihood to a product of terms:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$

and rewrite the posterior as:

$$p(C_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j)$$

Using Bayes' rule above, we label a new case X with a class level Cj that achieves the highest posterior probability.

Naive Bayes can be modeled in several different ways including normal, lognormal, gamma and Poisson density functions:

$$p(x_k \mid C_j) = \begin{cases} \dfrac{1}{\sigma_{kj}\sqrt{2\pi}} \exp\left(\dfrac{-(x-\mu_{kj})^2}{2\sigma_{kj}}\right), \quad -\infty < x < \infty, -\infty < \mu_{kj} <, \sigma_{kj} > 0 & \text{Normal} \\ \mu_{kj} : \text{mean}, \ \sigma_{kj} : \text{standard deviation} & \\ \dfrac{1}{x\sigma_{kj}(2\pi)^{1/2}} \exp\left\{\dfrac{-[\log(x/m_{kj})]^2}{2\sigma_{kj}^{\,2}}\right\}, \quad 0 < x < \infty, m_{kj} > 0, \sigma_{kj} > 0 & \text{Lognormal} \\ m_{kj} : \text{scale parameter}, \ \sigma_{kj} : \text{shape parameter} & \\ \dfrac{\left(\dfrac{x}{b_{kj}}\right)^{c_{kj}-1}}{b_{kj}\Gamma(c_{kj})} \exp\left(\dfrac{-x}{b_{kj}}\right), \quad 0 \le x < \infty, b_{kj} > 0, c_{kj} > 0 & \text{Gamma} \\ b_{kj} : \text{scale parameter}, \ c_{kj} : \text{shape parameter} & \\ \dfrac{\lambda_{kj} \exp(-\lambda_{kj})}{x!}, \quad 0 \le x < \infty, \lambda_{kj} > 0, x = 0,1,2,\ldots & \text{Poisson} \\ \lambda_{kj} : \text{mean} & \end{cases}$$

Types

- 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.

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

- Multinomial Naive Bayes : Its is used when we have discrete data (e.g. movie ratings ranging 1 and 5 as each rating will have certain frequency to represent). In text learning we have the count of each word to predict the class or label

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}{}^{x_i}$$

$$\hat{P}(x_i \mid \omega_j) = \frac{\sum tf(x_i, d \in \omega_j) + \alpha}{\sum N_{d \in \omega j} + \alpha \cdot V}$$

- **Bernoulli Naive Bayes :** It assumes that all our features are binary such that they take only two values. Means 0s can represent "word does not occur in the document" and 1s as "word occurs in the document"

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Source Code:

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

# 1.Data Handling
# 1.1 Loading the Data from csv file of ConceptLearning dataset.
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

#1.2 Splitting the Data set into Training Set
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]

#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 safe div(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 = safe div(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(-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

#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
	accuracy = safe div(correct,float(len(testSet))) * 100.0
	return accuracy

def main():
	filename = 'ConceptLearning.csv'
	splitRatio = 0.90
	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:

```
Naive Bayes Classifier for concept learning problem
Split 16 rows into
Number of Training data: 14
Number of Test Data: 2

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]
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]

The Test data set are:
[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]%
Predictions: [5.0, 5.0]%
Accuracy: 50.0%
```

Program6: 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* →{+, −}

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 Iv,), 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

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

   - o ***docs*$_j$ ← subset of *Examples* for which the target value is v_j**
   - o *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	

- Naive Bayes: 89% classification accuracy

Learning Curve for 20 Newsgroups

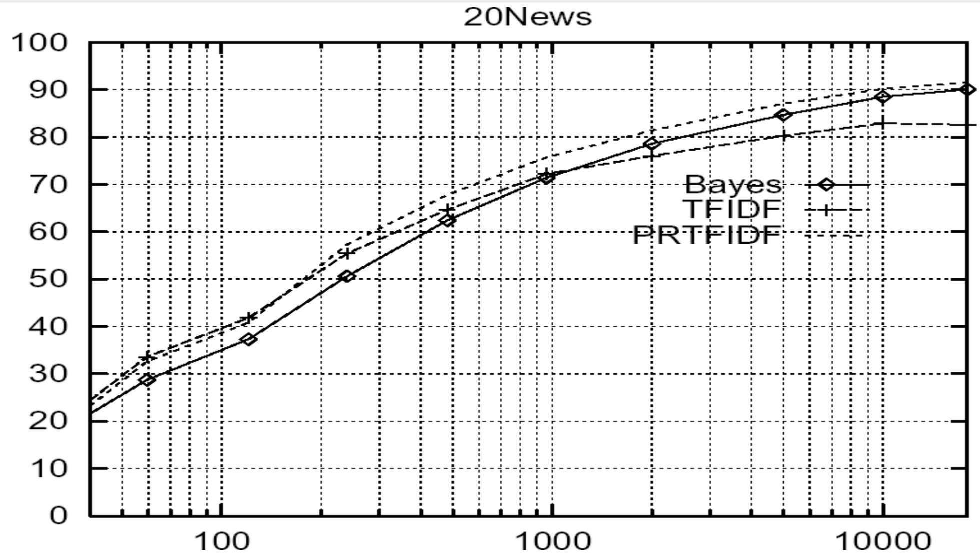


Fig: Accuracy vs. Training set size (1/3 withheld for test

Example :

- In the example, we are given a sentence " A very close game", a training set of five sentences (as shown below), and their corresponding category (Sports or Not Sports).
- The goal is to build a Naive Bayes classifier that will tell us which category the sentence " A very close game" belongs to. applying a Naive Bayes classifier, thus the strategy would be calculating the probability of both "A very close game is Sports", as well as it's *Not Sports*. The one with the higher probability will be the result.
- to calculate P(*Sports* **|** *A very close game*), i.e. the probability that the category of the sentence is *Sports*given that the sentence is "A very close game".

Text	Category
"A great game"	Sports
"The election was over"	Not sports
"Very clean match"	Sports
"A clean but forgettable game"	Sports
"It was a close election"	Not sports

Step 1: Feature Engineering

- word frequencies, i.e., counting the occurrence of every word in the document.
- *P(a very close game)* = P(a)XP(very)XP(close)XP(game)
- *P(a very close game | Sports)* = P(a|Sports) X P(Very|Sports) X P(close|Sports) X P(game|Sports)
- P(*a very close game* | *Not Sports*)=P(*a* | *Not Sports*)xP(*very* | *Not Sports*)xP(*close* | *Not Sports*) x P(*game* | *Not Sports*)

Step 2: Calculating the probabilities

- Here , the word "close" does not exist in the category Sports, thus P(*close* |*Sports*) = 0, leading to P(*a very close game* | *Sport*s)=0.
- Given an observation x = (x1, …, xd) from a multinomial distribution with N trials and parameter vector θ = (θ1, …, θd), a "smoothed" version of the data gives the estimator.

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

- where the pseudo count α > 0 is the smoothing parameter (α = 0 corresponds to no smoothing)

Word	P(word | Sports)	P(word | Not Sports)
a	$\frac{2+1}{11+14}$	$\frac{1+1}{9+14}$
very	$\frac{1+1}{11+14}$	$\frac{0+1}{9+14}$
close	$\frac{0+1}{11+14}$	$\frac{1+1}{9+14}$
game	$\frac{2+1}{11+14}$	$\frac{0+1}{9+14}$

$$P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times P(Sports)$$
$$= 4.61 \times 10^{-5}$$
$$= 0.0000461$$

$$\text{P(a — Not Sports)} \times P(very|Not\ Sports) \times P(close|Not\ Sports) \times P(game|Not\ Sports) \times P(Not\ Sports)$$
$$= 1.43 \times 10^{-5}$$
$$= 0.0000143$$

As seen from the results shown below, P(a very close game | Sports) gives a higher probability, suggesting that the sentence belongs to the Sports category.

Multinomial Naive Bayes

Term Frequency

A alternative approach to characterize text documents — rather than binary values — is the *term frequency (tf(t, d))*. The term frequency is typically defined as the number of times a given term *t* (i.e., word or token) appears in a document *d* (this approach is sometimes also called *raw frequency*). In practice, the term frequency is often normalized by dividing the raw term frequency by the document length.

$$\text{normalized term frequency} = \frac{tf(t,d)}{n_d}$$

where

- $tf(t,d)$: Raw term frequency (the count of term t in document d).
- n_d: The total number of terms in document d.

The term frequencies can then be used to compute the maximum-likelihood estimate based on the training data to estimate the class-conditional probabilities in the multinomial model:

$$\hat{P}(x_i \mid \omega_j) = \frac{\sum tf(x_i, d \in \omega_j) + \alpha}{\sum N_{d \in \omega j} + \alpha \cdot V}$$

where

- x_i: A word from the feature vector $\mathbf{x}$ of a particular sample.
- $\sum tf(x_i, d \in \omega_j)$: The sum of raw term frequencies of word x_i from all documents in the training sample that belong to class ω_j.
- $\sum N_{d \in \omega j}$: The sum of all term frequencies in the training dataset for class ω_j.
- α: An additive smoothing parameter ($\alpha = 1$ for Laplace smoothing).
- V: The size of the vocabulary (number of different words in the training set).

The class-conditional probability of encountering the text $\mathbf{x}$ can be calculated as the product from the likelihoods of the individual words (under the *naive* assumption of conditional independence).

$$P(\mathbf{x} \mid \omega_j) = P(x_1 \mid \omega_j) \cdot P(x_2 \mid \omega_j) \cdot \ldots \cdot P(x_n \mid \omega_j) = \prod_{i=1}^{m} P(x_i \mid \omega_j)$$

Confusion Matrix

- A confusion matrix is a summary of prediction results on a classification problem.
- The number of correct and incorrect predictions are summarized with count values and broken down by each class.

- Positive (P) : Observation is positive.
- Negative (N): Observation is not positive.
- True Positive (TP) :

Observation is positive, and is predicted to be positive.

- False Negative (FN) :

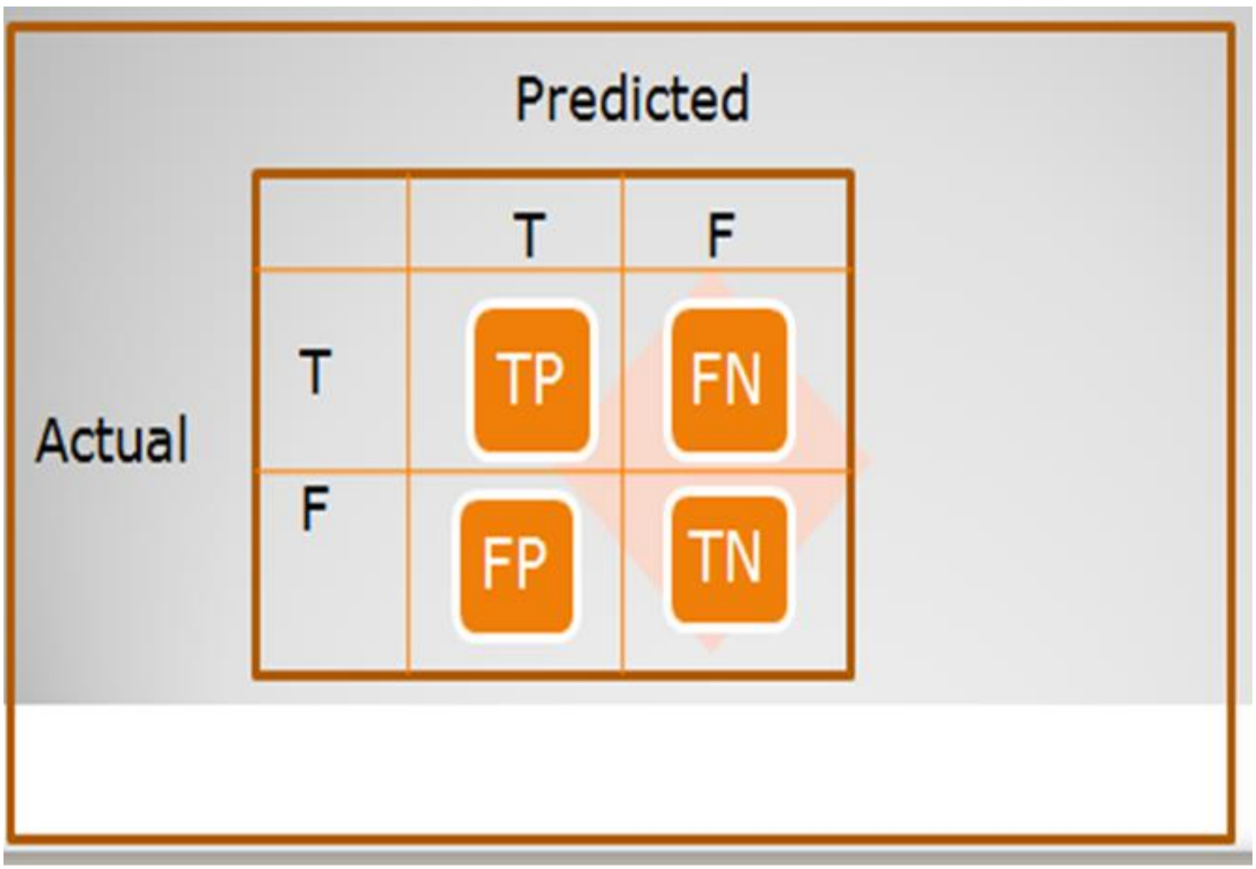
Observation is positive, but is predicted negative.

- True Negative (TN) :

Observation is negative, and is predicted to be negative.

- False Positive (FP) :

Observation is negative, but is predicted positive.



```
Accuracy: 0.8348868175765646
                          precision    recall  f1-score

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

           avg / total       0.88      0.83      0.84

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

Classifier Model Evaluation Metrics

- **Accuracy** :

Accuracy=TP+TN / (TP+FP+TN+FN)

- **Precision : Class labeled as positive indeed positive**

Precision = TP/(TP + FP)

- **Recall :Class is correctly recognized**

Recall = TP/(TP+FN)

- **F_1 score:** It is the harmonic average of the Precision and Recall, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

F1 = 2 x (precision x recall)/(precision + recall)

Confusion Matrix

192	2	6	119
2	347	4	36
2	11	322	61
2	2	1	393

- **TP = 192**
- **FN = Sum of values in the corresponding row (exclude TP) 2+6+119=127**
- **FP = Sum of values in the corresponding column (exclude TP) 2+2+2=6**
- **TN = Sum of all the columns and rows excluding that class column and row**

347	4	36	387
11	322	61	394
2	1	393	396
360	327	490	**1177**

TN=1177

- **Accuracy=TP+TN / (TP+FP+TN+FN)**
  Accuracy = 192+1177/(192+127+6+1177)

- **Precision = TP/(TP + FP)**
  Precision = 192/(192+6)

- **Recall = TP/(TP+FN)**
  Recall=192/(192+127)

- **F1 = 2 x (precision x recall)/(precision + recall)**
  F1=2*[(0.97*0.60)/(0.97+0.60)]

Source Code:

Loading the 20 newsgroups dataset : The dataset is called "Twenty Newsgroups". Here is the official description, quoted from the website:http://qwone.com/~jason/20Newsgroups/

The 20 Newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. To the best of our knowledge, it was originally collected by Ken Lang, probably for his paper "Newsweeder: Learning to filter netnews," though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

```
Source Code:

from sklearn.datasets import fetch_20newsgroups

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

import numpy as np

twenty_train = fetch_20newsgroups(subset='train',

shuffle=True)

x = len(twenty_train.target_names)

print("\n The number of categories:",x)

print("\n The %d Different Categories of

20Newsgroups\n" %x)

i=1

for cat in twenty_train.target_names:

 print("Category[%d]:" %i,cat)

 i=i+1

print("\n Length of train data

is",len(twenty_train.data))

print("\n Length of file names is

",len(twenty_train.filenames))

#Considering only four Categories
```

```
categories = ['alt.atheism',
'soc.religion.christian','comp.graphics',
'sci.med']
twenty_train =
fetch_20newsgroups(subset='train',categories=cate
gories,shuffle=True)
twenty_test =
fetch_20newsgroups(subset='test',categories=categ
ories,shuffle=True)
print("Reduced length of train
data",len(twenty_train.data))
print("length of test
data",len(twenty_test.data))
print("Target Names",twenty_train.target_names)
#print("\n".join(twenty_train.data[0].split("\n")
))
#print(twenty_train.target[0])
#Extracting features from text files
from sklearn.feature_extraction.text import
CountVectorizer
count_vect = CountVectorizer()
#Term Frequencies(tf): Divide the number of
occurrences of each word in a document by the
total number of words in the document
X_train_tf =
count_vect.fit_transform(twenty_train.data)
X_train_tf.shape
```

```
print("tf train count",X_train_tf.shape)
#another refinement for tf is called tf-idf for
“Term Frequency times Inverse Document
Frequency”.
from sklearn.feature_extraction.text import
TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf =
tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
print("tfidf train count",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)
print("tf test count",X_test_tf.shape)
X_test_tfidf =
tfidf_transformer.transform(X_test_tf)
print("tfidf test count",X_test_tfidf.shape)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:",
accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,pr
```

```
edicted,target_names=twenty_test.target_names))



print("confusion matrix is

\n",metrics.confusion_matrix(twenty_test.target,

predicted))
```

Output:

```
The number of categories: 20

 The 20 Different Categories of 20Newsgroups

Category[1]: alt.atheism
Category[2]: comp.graphics
Category[3]: comp.os.ms-windows.misc
Category[4]: comp.sys.ibm.pc.hardware
Category[5]: comp.sys.mac.hardware
Category[6]: comp.windows.x
Category[7]: misc.forsale
Category[8]: rec.autos
Category[9]: rec.motorcycles
Category[10]: rec.sport.baseball
Category[11]: rec.sport.hockey
Category[12]: sci.crypt
Category[13]: sci.electronics
Category[14]: sci.med
Category[15]: sci.space
Category[16]: soc.religion.christian
Category[17]: talk.politics.guns
Category[18]: talk.politics.mideast
Category[19]: talk.politics.misc
Category[20]: talk.religion.misc

 Length of train data is 11314

 Length of file names is  11314
Reduced length of train data 2257
length of test data 1502
Target Names ['alt.atheism', 'comp.graphics', 'sci.med',
'soc.religion.christian']
tf train count (2257, 35788)
tfidf train count (2257, 35788)
tf test count (1502, 35788)
tfidf test count (1502, 35788)
Accuracy: 0.8348868175765646
                        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]]
>>>
```

Program7 : 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 :

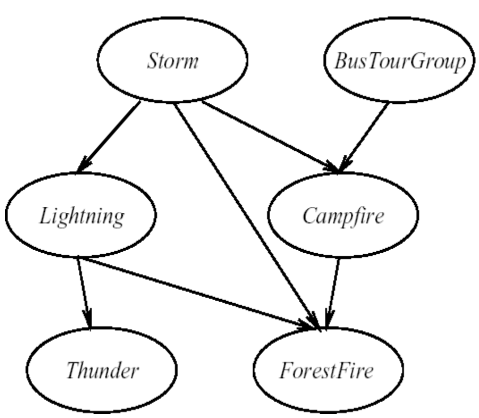
Bayesian Network (BAYESIAN BELIEF NETWORKS

- 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*, . . . , *ForestFire*)
  - in general,

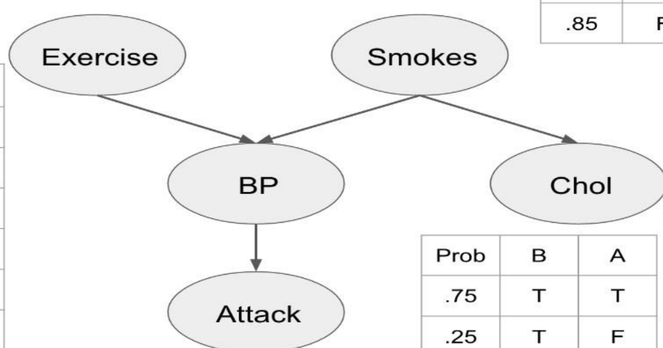
$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

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

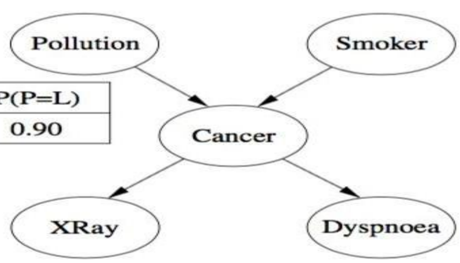


Prob	E	S	B
.45	T	T	T
.55	T	T	F
.05	T	F	T
.95	T	F	F
.95	F	T	T
.05	F	T	F
.55	F	F	T
.45	F	F	F

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:

Constructing a Bayesian Network considering Medical Data

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()
```

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])
```

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'))
```

Determining the Localindependencies

```
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,'Pollut
ion': 1})
print(q['Cancer'])
```

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
```

Importing Heart Disease Data Set and Customizing

```
Cleveland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-datab
ases/heart-disease/processed.hungarian.data'
```

```
#Hungarian_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-data
bases/heart-disease/processed.hungarian.data'
#Switzerland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-da
tabases/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', 'thalac
h', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']
heartDisease = pd.read_csv(urlopen(Cleveland_data_URL), names = names) #ge
ts Cleveland data
#HungarianHeartDisease = pd.read_csv(urlopen(Hungarian_data_URL), names =
names) #gets Hungary data
#SwitzerlandHeartDisease = pd.read_csv(urlopen(Switzerland_data_URL), name
s = names) #gets Switzerland data
#datatemp = [ClevelandHeartDisease, HungarianHeartDisease, SwitzerlandHear
tDisease] #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
```

Modeling Heart Disease Data

```
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

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

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

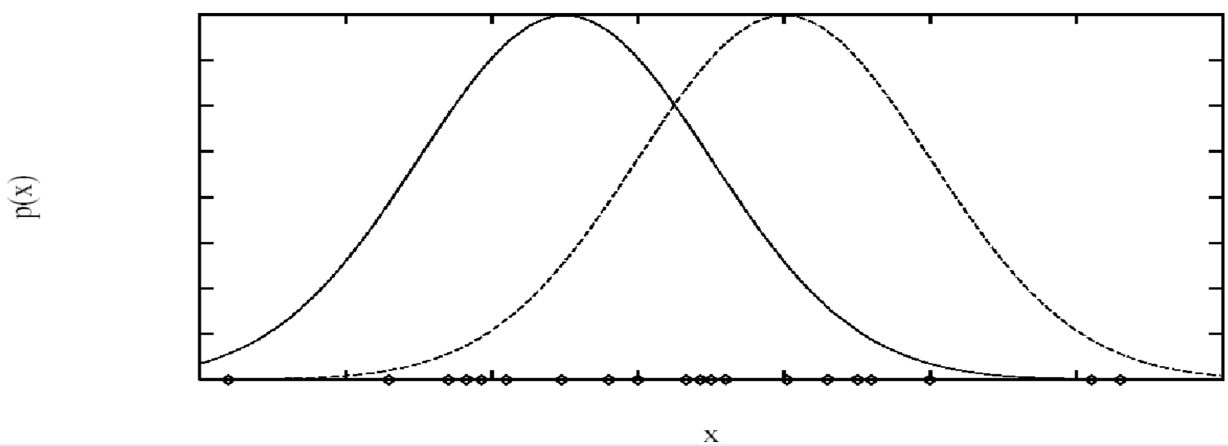
Program 8 : Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

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

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

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

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Basic Algorithm of K-means

Algorithm 1 Basic K-means Algorithm.

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

Details of K-means

1. Initial centroids are often chosen randomly.
   - *Clusters produced vary from one run to another*
2. The centroid is (typically) the mean of the points in the cluster.
3. 'Closeness' is measured by **Euclidean distance**, cosine similarity, correlation, etc.
4. K-means will converge for common similarity measures mentioned above.
5. Most of the convergence happens in the first few iterations.
   - *Often the stopping condition is changed to 'Until relatively few points change clusters'*

Euclidean Distance

$$d(i,j)=\sqrt{|x_{i_1}-x_{j_1}|^2+|x_{i_2}-x_{j_2}|^2+\ldots+|x_{i_p}-x_{j_p}|^2}$$

A simple example: Find the distance between two points, the original and the point (3,4)

$$d_E(O,A)=\sqrt{3^2+4^2}=5$$

Update Centroid

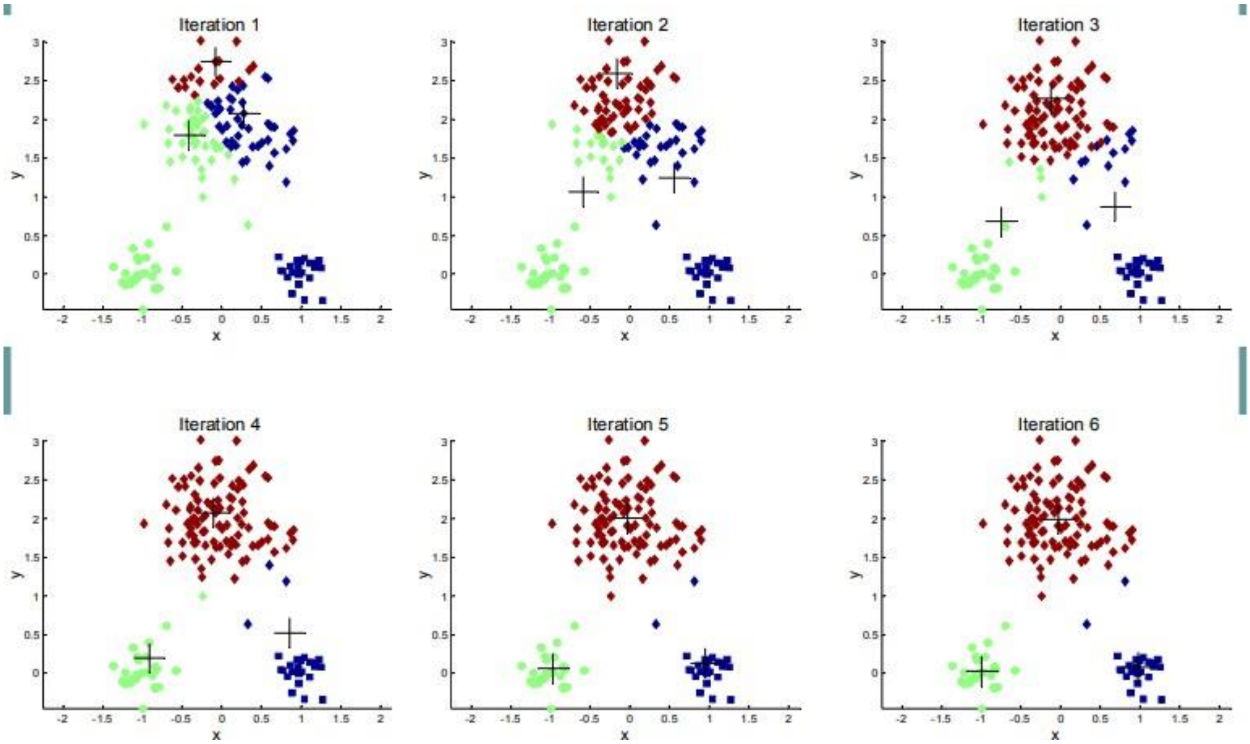
We use the following equation to calculate the n dimensional centroid point amid k n-dimensional points

$$CP(x_1,x_2,\ldots,x_k)=\left(\frac{\sum_{i=1}^{k}x1st_i}{k},\frac{\sum_{i=1}^{k}x2nd_i}{k},\ldots,\frac{\sum_{i=1}^{k}xnth_i}{k}\right)$$

Example: Find the centroid of 3 2D points, (2,4), (5,2) and (8,9)

$$CP=\left(\frac{2+5+8}{3},\frac{4+2+9}{3}\right)=(5,5)$$

Examples of K Means

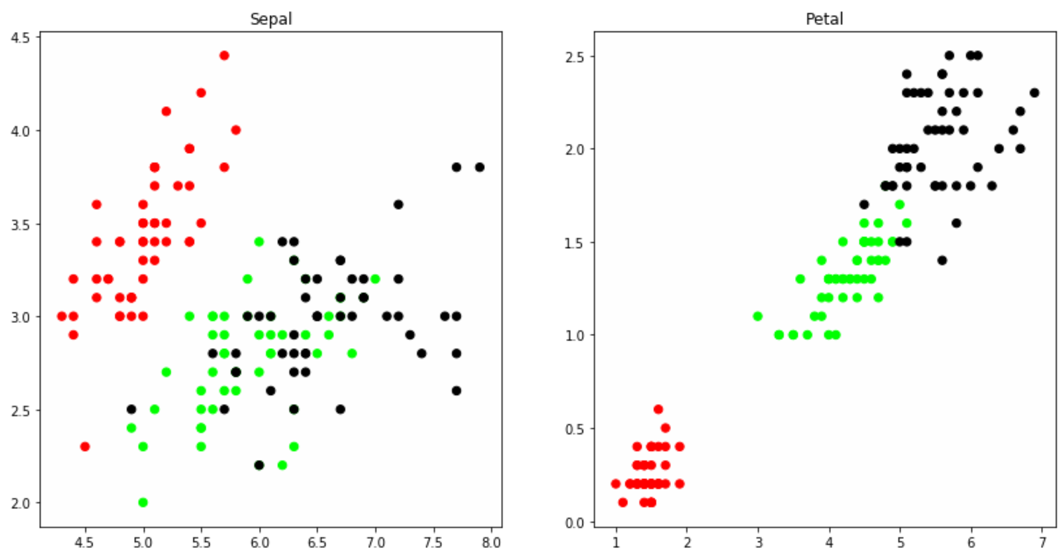


Source Code:

```
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()

#print("\n IRIS DATA :",iris.data);
#print("\n IRIS FEATURES :\n",iris.feature_names)
#print("\n IRIS TARGET  :\n",iris.target)
#print("\n IRIS TARGET NAMES:\n",iris.target_names)

# 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')
```

```
Text(0.5,1,'Petal')
```



Build the KMeans Model

```
# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(X)
# This is what KMeans thought
model.labels_
```

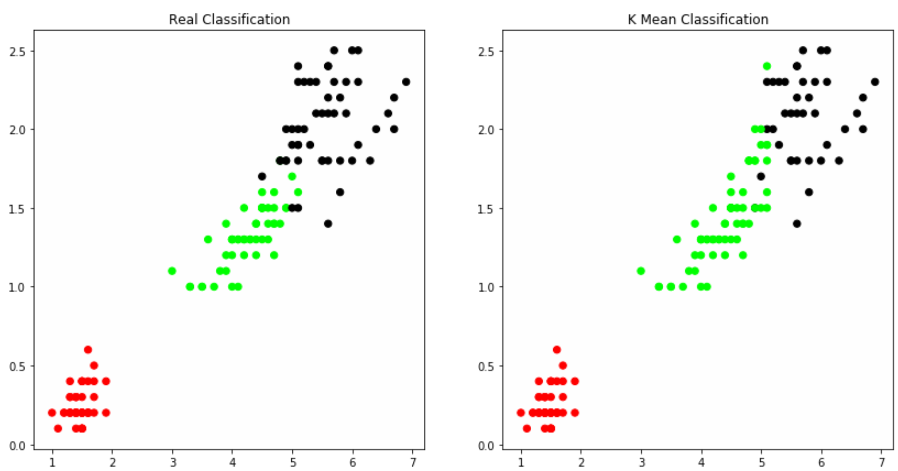
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
,

```
        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])
```

Visualise the classifier results

```
# 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')
```

```
Text(0.5,1,'K Mean Classification')
```



The Fix

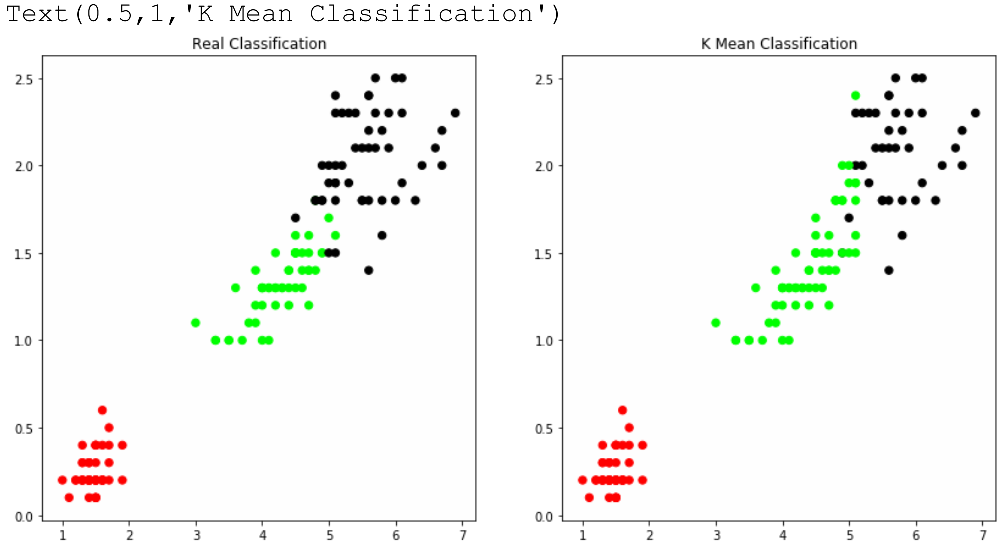
```
# The fix, we convert all the 1s to 0s and 0s to 1s.
predY = np.choose(model.labels , [0, 1, 2]).astype(np.int64)
print (predY)
```

```
[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 2 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]
```

Re-plot

```
# 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')
```

```
Text(0.5,1,'K Mean Classification')
```



Performance Measures

Accuracy

```
sm.accuracy_score(y, model.labels_)
```

```
0.89333333333333331
```

Confusion Matrix

```
# Confusion Matrix
sm.confusion_matrix(y, model.labels_)
```

```
array([[50,  0,  0],
       [ 0, 48,  2],
       [ 0, 14, 36]], dtype=int64)
```

GMM

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
xs.sample(5)
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
132	0.674501	-0.587764	1.047087	1.316483
110	0.795669	0.337848	0.762759	1.053537
93	-1.021849	-1.744778	-0.260824	-0.261193
24	-1.264185	0.800654	-1.056944	-1.312977
111	0.674501	-0.819166	0.876490	0.922064

```
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)
```

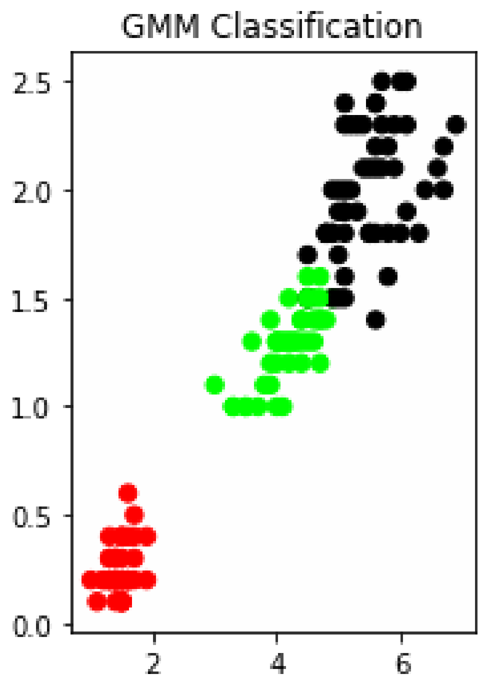
```
y_cluster_gmm = gmm.predict(xs)
y_cluster_gmm
```

```
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2
,
       1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

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

```
ext(0.5,1,'GMM Classification')
```




```
sm.accuracy_score(y, y_cluster_gmm)
```

```
0.9666666666666667
```

```
# Confusion Matrix
sm.confusion_matrix(y, y_cluster_gmm)
```

```
array([[50,  0,  0],
       [ 0, 45,  5],
       [ 0,  0, 50]], dtype=int64)
```

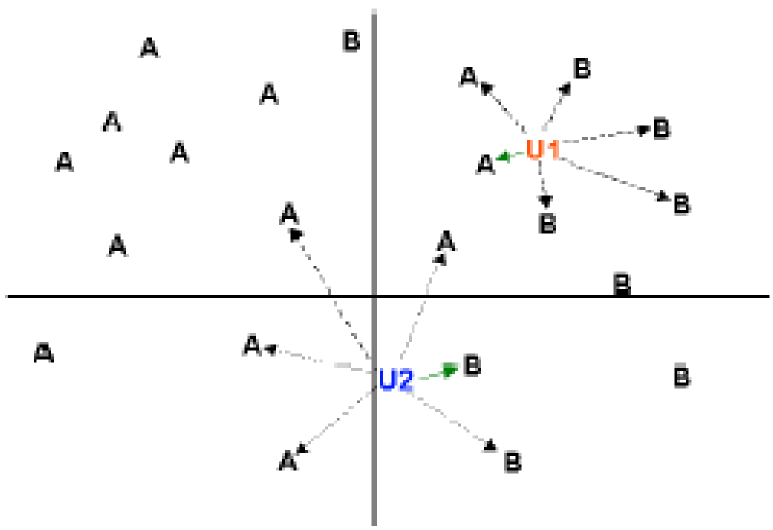
```
# so the GMM clustering matched the true labels more closely than the Kmea
ns,
# as expected from the plots.
```

Program9 : 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.

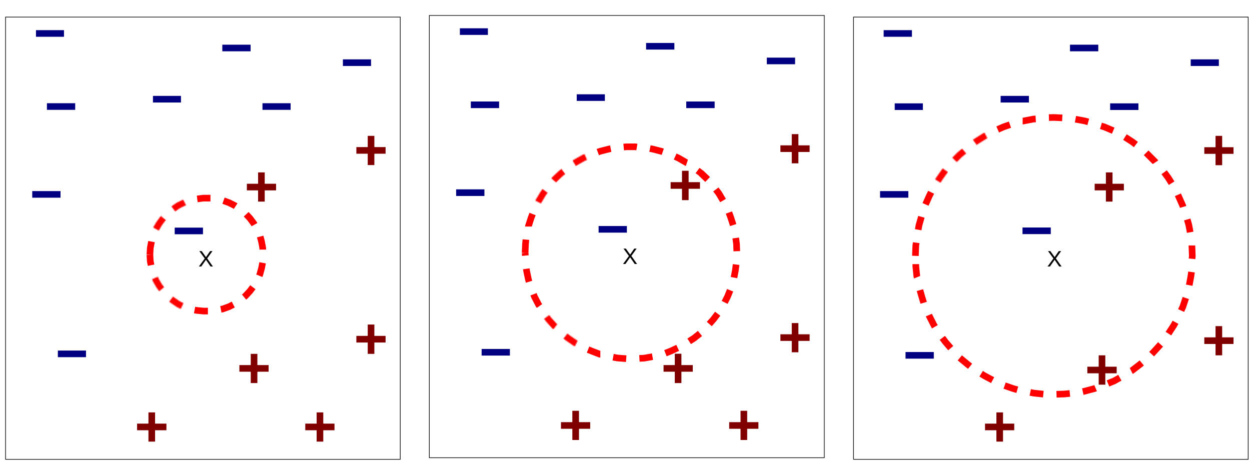
Algorithm :

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

Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_{\boldsymbol{x}}$ is the sum of all values in the vector $\boldsymbol{x}$.

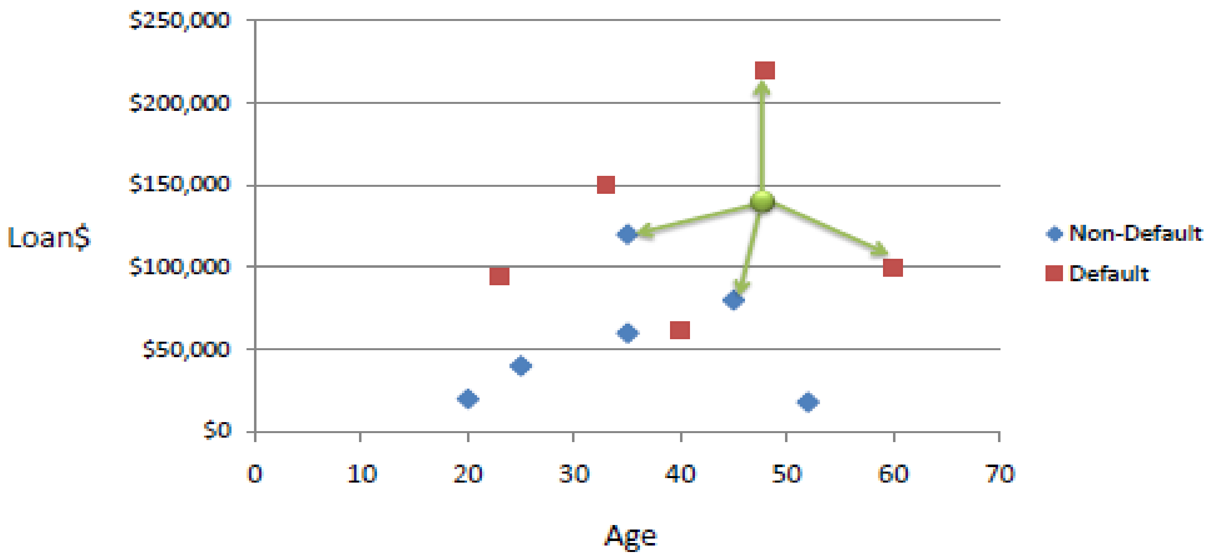
Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\,\mathrm{sign}(y_i - y_j)$$

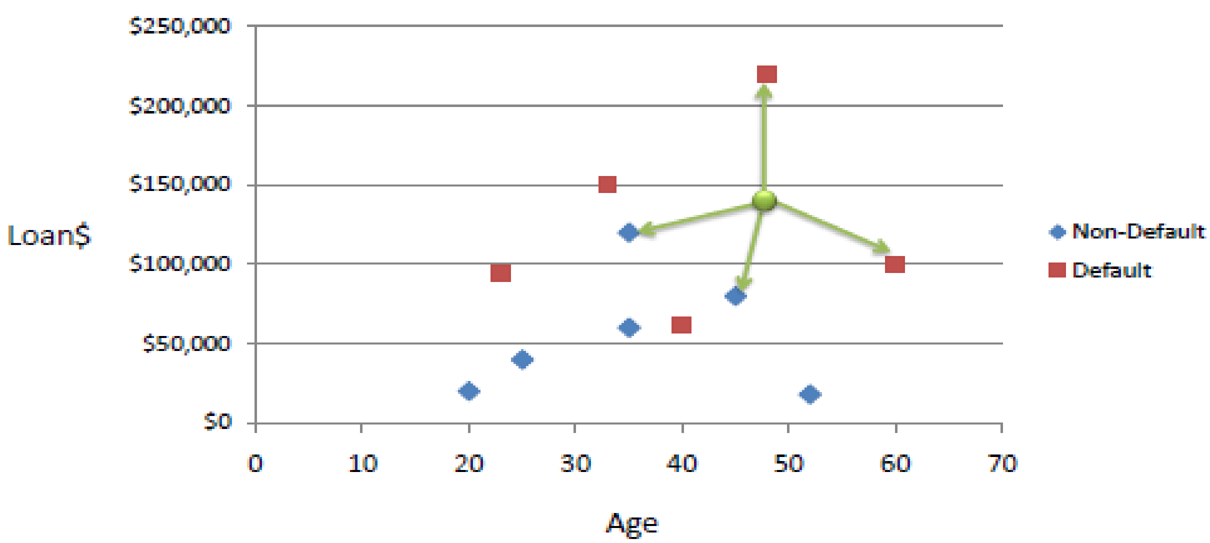
$\mathrm{sign}(x)$=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions.
($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

Example: Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.



D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y

Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	$142,000	?		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

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_d
ataset["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))
)
```

Output :

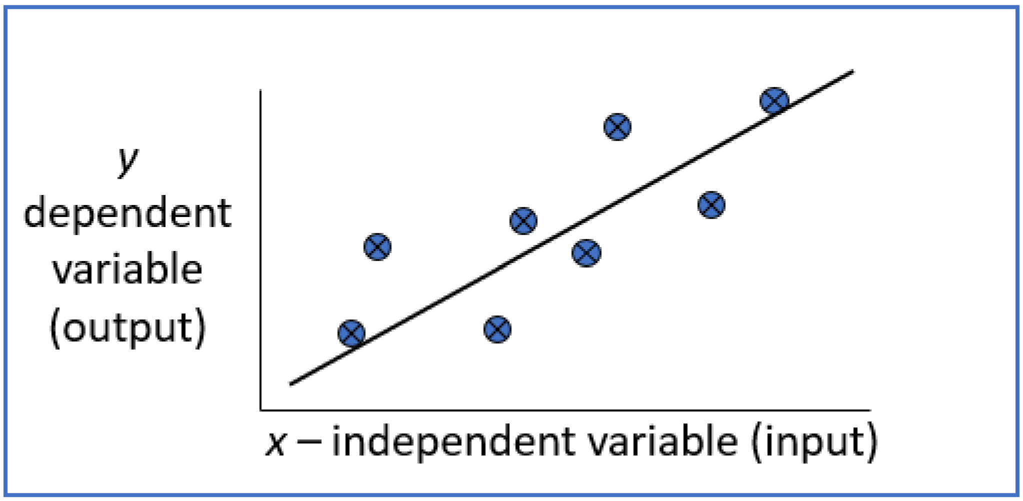
```
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 1 versicolor, Predicted :[1]['versicolor']
Actual : 0 setosa, Predicted :[0]['setosa']
Actual : 2 virginica, Predicted :[2]['virginica']
Actual : 0 setosa, Predicted :[0]['setosa']
--------
Actual : 1 versicolor, Predicted :[2]['virginica']

TEST SCORE[ACCURACY]: 0.97
```

Program10 : Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

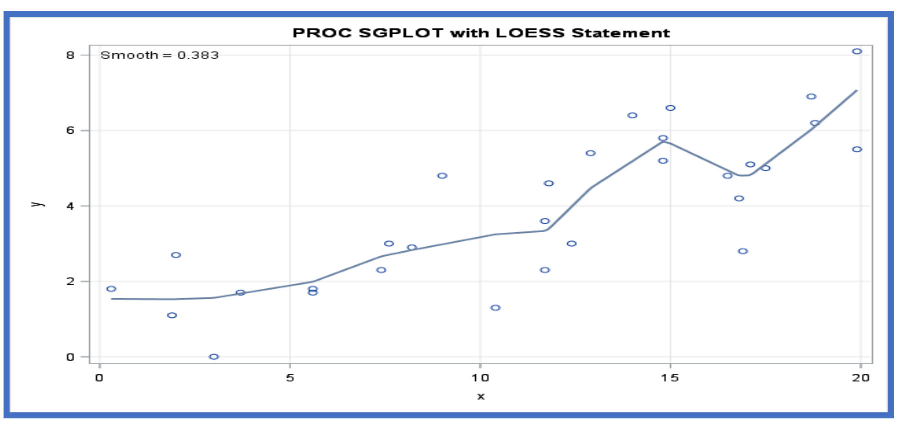
Algorithm :

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.



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



Lowess Algorithm: Locally weighted regression is a very powerful 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:

```
import numpy as np
from bokeh.plotting import figure, show, output notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

output_notebook()
```

BokehJS 0.12.10 successfully loaded.

```
import numpy as np
```

```
def local_regression(x0, X, Y, tau):
    # add bias term
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
    X = np.c_[np.ones(len(X)), X]

    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau)    # XTranspose * W

    beta = np.linalg.pinv(xw @ X) @ xw @ Y    # @ Matrix Multiplication or
Dot Product

    # predict value
    return x0 @ beta    # @ Matrix Multiplication or Dot Product for predi
ction

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function
```

```
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y  :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
```

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y  :
 [ 2.13582188  2.13156806  2.12730467  2.12303166  2.11874898  2.11445659
  2.11015444  2.10584249  2.10152068]
Normalised (10 Samples) X :
 [-3.17013248 -2.87908581 -3.37488159 -2.90743352 -2.93640374 -2.97978828
 -3.0549104  -3.0735006  -2.88552749]
```

```
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples)  :\n",domain[1:10])

def plot_lwr(tau):
    # prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot
```

```
Xo Domain Space(10 Samples)   :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

```
# Plotting the curves with different tau
show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]
]))
```

Output :