

Machine Learning Lab Manual

INDEX

S.No	Title	Page No																														
1.	The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)																															
2.	Extract the data from database using python																															
3.	Implement k-nearest neighbours classification using python																															
4.	<p>Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans clustering with 3 means (i.e., 3 centroids) periments</p> <table> <thead> <tr> <th>VAR1</th> <th>VAR2</th> <th>CLASS</th> </tr> </thead> <tbody> <tr><td>1.713</td><td>1.586</td><td>0</td></tr> <tr><td>0.180</td><td>1.786</td><td>1</td></tr> <tr><td>0.353</td><td>1.240</td><td>1</td></tr> <tr><td>0.940</td><td>1.566</td><td>0</td></tr> <tr><td>1.486</td><td>0.759</td><td>1</td></tr> <tr><td>1.266</td><td>1.106</td><td>0</td></tr> <tr><td>1.540</td><td>0.419</td><td>1</td></tr> <tr><td>0.459</td><td>1.799</td><td>1</td></tr> <tr><td>0.773</td><td>0.186</td><td>1</td></tr> </tbody> </table>	VAR1	VAR2	CLASS	1.713	1.586	0	0.180	1.786	1	0.353	1.240	1	0.940	1.566	0	1.486	0.759	1	1.266	1.106	0	1.540	0.419	1	0.459	1.799	1	0.773	0.186	1	
VAR1	VAR2	CLASS																														
1.713	1.586	0																														
0.180	1.786	1																														
0.353	1.240	1																														
0.940	1.566	0																														
1.486	0.759	1																														
1.266	1.106	0																														
1.540	0.419	1																														
0.459	1.799	1																														
0.773	0.186	1																														
5.	<p>The following training examples map descriptions of individuals onto high, medium and low credit-worthiness.</p> <p>medium skiing design single twenties no -> highRisk</p> <p>high golf trading married forties yes -> lowRisk</p> <p>low speedway transport married thirties yes -> medRisk</p> <p>medium football banking single thirties yes -> lowRisk</p> <p>high flying media married fifties yes -> highRisk</p> <p>low football security single twenties no -> medRisk</p> <p>medium golf media single thirties yes -> medRisk</p> <p>medium golf transport married forties yes -> lowRisk</p> <p>high skiing banking single thirties yes -> highRisk</p> <p>low golf unemployed married forties yes -> highRisk</p> <p>Input attributes are (from left to right) income, recreation, job, status, age-</p>																															

	group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset?	
6.	Implement linear regression using python.	
7.	Implement Naïve Bayes theorem to classify the English text	
8.	Implement an algorithm to demonstrate the significance of genetic algorithm	
9.	Implement the finite words classification system using Back-propagation algorithm	
10.	Find-S Algorithm	
11.	Candidate Elimination Algorithm	
12.	K-Means Clustering Algorithm	

Experiment :1

1. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

ALGORITHM:

- Step 1: Calculate probability for each word in a text and filter the words which have a probability less than threshold probability. Words with probability less than threshold probability are irrelevant.
- Step 2: Then for each word in the dictionary, create a probability of that word being in insincere questions and its probability insincere questions. Then finding the conditional probability to use in naive Bayes classifier.
- Step 3: Prediction using conditional probabilities.
- Step 4: End.

PROGRAM:

```
PFIA=float(input("Enter probability that it is Friday and that a student is absent="))

PF=float(input(" probability that it is Friday="))

PABF=PFIA / PF

print("probability that a student is absent given that today is Friday using conditional
probabilities=",PABF)
```

OUTPUT:

Enter probability that it is Friday and that a student is
absent= 0.03 probability that it is Friday= 0.2

probability that a student is absent given that today is Friday using conditional probabilities= 0.15

Experiment:2

2. Extract the data from database using python

Extracting data from Excel sheet using Python

Step1: First convert dataset present in excel to CSV file using online resources, then execute following program:

consider dataset excel consists of 14 input columns and 3 output columns (C1, C2, C3)as follows:

Python Souce Code:

```
import pandas as pd

Import csv

dataset=pd.read_csv("Sample_Dataset.csv", delimiter=',')

print(dataset)                                #Print entire dataset

X = dataset[['AA','BB','CC','DD','EE','FF']].values

Y = dataset[['C1','C2','C3']].values

print(Y)                                     #Prints output values

print(X)                                     #Prints intput values

X1 = dataset[['AA','BB','CC']].values

print(X1)                                    #Prints first 5 columns of intput values

print(X[0:5])                               # Prints only first 5 rows of input values
```

OUTPUT SCREENS:

AA	BB	CC	DD	EE	FF	C1	C2	C3
1	2	3	1	2	3	0	1	0
3	2	1	4	1	2	1	1	1
1	2	3	1	2	3	1	0	1
3	2	1	4	1	2	1	1	0
1	2	3	1	2	3	1	1	1
3	2	1	4	1	2	0	1	0
1	2	3	1	2	3	1	1	1
3	2	1	4	1	2	1	0	1
1	2	3	1	2	3	1	1	0
3	2	1	4	1	2	1	1	1

Untitled8.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Files

- sample_data
- Sample_Dataset.csv

```
import pandas as pd
import csv
dataset=pd.read_csv("Sample_Dataset.csv", delimiter=',')
print(dataset) #Print entire dataset
X = dataset[['AA','BB','CC','DD','EE','FF']].values
Y = dataset[['C1','C2','C3']].values
print(Y) #Prints output values
print(X) #Prints input values
X1 = dataset[['AA','BB','CC']].values
print(X1) #Prints first 5 columns of input values
print(X[0:5])
```

	AA	BB	CC	DD	EE	FF	C1	C2	C3
0	1	2	3	1	2	3	0	1	0
1	3	2	1	4	1	2	1	1	1
2	1	2	3	1	2	3	1	0	1
3	3	2	1	4	1	2	1	1	0
4	1	2	3	1	2	3	1	1	1
5	3	2	1	4	1	2	0	1	0
6	1	2	3	1	2	3	1	1	1
7	3	2	1	4	1	2	1	0	1
8	1	2	3	1	2	3	1	1	0
9	3	2	1	4	1	2	1	1	1

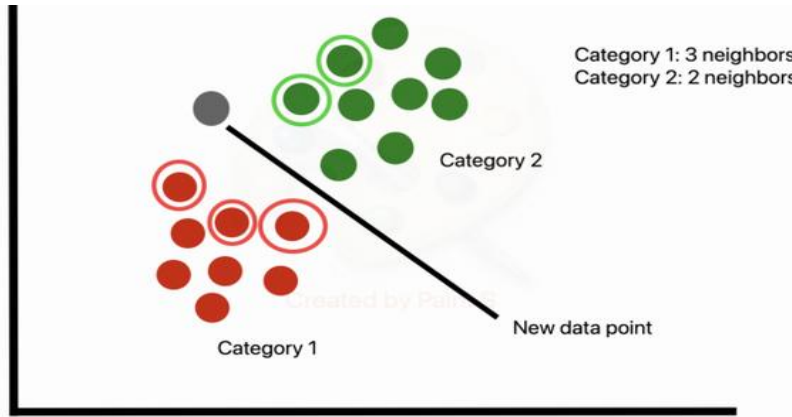
```
[[0 1 0]
 [1 1 1]
 [1 0 1]]
```

Disk 69.37 GB available

Experiment:3

3. Implement k-nearest neighbours classification using python

K-Nearest Neighbors, or KNN for short, is one of the simplest machine learning algorithms and is used in a wide array of institutions. KNN is a **non-parametric, lazy** learning algorithm. When we say a technique is non-parametric, it means that it does not make any assumptions about the underlying data.



Pros:

- No assumptions about data
- Simple algorithm — easy to understand
- Can be used for classification and regression

Cons:

- High memory requirement — All of the training data must be present in memory in order to calculate the closest K neighbors
- Sensitive to irrelevant features
- Sensitive to the scale of the data since we're computing the distance to the closest K points

ALGORITHM:

Step 1: Load the data

Step 2: Initialize the value of k

Step 3: For getting the predicted class, iterate from 1 to total number of training data points

- i) Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
 - ii) Sort the calculated distances in ascending order based on distance values. Get top k rows from the sorted array
 - iii) Get the most frequent class of these rows i.e. Get the labels of the selected K entries
 - iv) Return the predicted class If regression, return the mean of the K labels If classification, return the mode of the K labels
- If regression, return the mean of the K labels
 - If classification, return the mode of the K labels

Step 4: End.

PROGRAM

```
import numpy as np
```

```
from sklearn import datasets
```

```
iris = datasets.load_iris()
```

```
data = iris.data
```

```
labels = iris.target
```

```
for i in [0, 79, 99, 101]:
```

```
    print(f"index: {i:3}, features: {data[i]}, label: {labels[i]}")
```

```
np.random.seed(42)
```

```
indices = np.random.permutation(len(data))
```

```
n_training_samples = 12
```

```
learn_data = data[indices[:-n_training_samples]]
```

```
learn_labels = labels[indices[:-n_training_samples]]
```

```
test_data = data[indices[-n_training_samples:]]
```

```
test_labels = labels[indices[-n_training_samples:]]
```

```
print("The first samples of our learn set:")
```

```

print(f'{'index':7s}{'data':20s}{'label':3s}')
for i in range(5):
    print(f'{i:4d} {learn_data[i]} {learn_labels[i]:3}')

print("The first samples of our test set:")
print(f'{'index':7s}{'data':20s}{'label':3s}')
for i in range(5):
    print(f'{i:4d} {learn_data[i]} {learn_labels[i]:3}')

```

#The following code is only necessary to visualize the data of our learnset

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
colours = ("r", "b")
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(learn_data)):
        if learn_labels[i] == iclass:
            X[iclass][0].append(learn_data[i][0])
            X[iclass][1].append(learn_data[i][1])
            X[iclass][2].append(sum(learn_data[i][2:]))

colours = ("r", "g", "y")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

```



```

for iclass in range(3):
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])
plt.show()

#-----

def distance(instance1, instance2):
    """ Calculates the Euclidian distance between two instances"""
    return np.linalg.norm(np.subtract(instance1, instance2))

def get_neighbors(training_set, labels, test_instance, k, distance):
    """
    get_neighors calculates a list of the k nearest neighbors of an instance 'test_instance'.
    The function returns a list of k 3-tuples. Each 3-tuples consists of (index, dist, label)
    """

    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append((training_set[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return neighbors

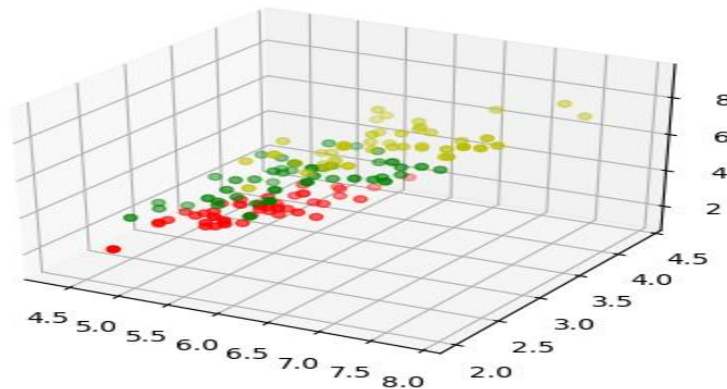
for i in range(5):
    neighbors = get_neighbors(learn_data, learn_labels, test_data[i], 3,
                             distance=distance)
    print("Index:      ", i, '\n',
          "Testset Data: ", test_data[i], '\n',
          "Testset Label: ", test_labels[i], '\n',

```

"Neighbors: ",neighbors,'\n')

OUTPUT:

```
(base) dohathi@dohathi-Compaq-15-Notebook-PC:~/ML_LAB$ python KNN.py
index: 0, features: [5.1 3.5 1.4 0.2], label: 0
index: 79, features: [5.7 2.6 3.5 1. ], label: 1
index: 99, features: [5.7 2.8 4.1 1.3], label: 1
index: 101, features: [5.8 2.7 5.1 1.9], label: 2
The first samples of our learn set:
index  data                      label
0      [6.1 2.8 4.7 1.2]          1
1      [5.7 3.8 1.7 0.3]          0
2      [7.7 2.6 6.9 2.3]          2
3      [6.  2.9 4.5 1.5]          1
4      [6.8 2.8 4.8 1.4]          1
The first samples of our test set:
index  data                      label
0      [6.1 2.8 4.7 1.2]          1
1      [5.7 3.8 1.7 0.3]          0
2      [7.7 2.6 6.9 2.3]          2
3      [6.  2.9 4.5 1.5]          1
4      [6.8 2.8 4.8 1.4]          1
```



```
Index: 2
Testset Data: [6.3 2.3 4.4 1.3]
Testset Label: 1
Neighbors: [(array([6.2, 2.2, 4.5, 1.5]), 0.26457513110645864, 1),
(array([6.3, 2.5, 4.9, 1.5]), 0.574456264653803, 1), (array([6. , 2.2, 4.
. , 1. ]), 0.5916079783099617, 1)]

Index: 3
Testset Data: [6.4 2.9 4.3 1.3]
Testset Label: 1
Neighbors: [(array([6.2, 2.9, 4.3, 1.3]), 0.200000000000000018, 1),
(array([6.6, 3. , 4.4, 1.4]), 0.2645751311064587, 1), (array([6.6, 2.9,
4.6, 1.3]), 0.3605551275463984, 1)]

Index: 4
Testset Data: [5.6 2.8 4.9 2. ]
Testset Label: 2
Neighbors: [(array([5.8, 2.7, 5.1, 1.9]), 0.31622776601683755, 2),
(array([5.8, 2.7, 5.1, 1.9]), 0.31622776601683755, 2), (array([5.7, 2.5,
5. , 2. ]), 0.33166247903553986, 2)]
```

Experiment 4

4. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e., 3centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

ALGORITHM:

K means Clustering Algorithm:

K Means algorithm is a centroid-based clustering (unsupervised) technique. This technique groups the dataset into k different clusters having an almost equal number of points. Each of the clusters has a centroid point which represents the mean of the data points lying in that cluster.

The idea of the K-Means algorithm is to find k-centroid points and every point in the dataset will belong to either of the k-sets having minimum Euclidean distance.

Step 1: Create X array with [var1,var2] as each element from the given input.

Step 2: Create y array with Class attribute from the given input.

Step 3: Training the KMeans model by providing (X,y) as training data.

Step 4: Predict the model by giving input.

Step 5: End

PROGRAM:

```
from sklearn.cluster import KMeans
```

```
import numpy as np
```

```
X = np.array([[1.713,1.586], [0.180,1.786], [0.353,1.240], [0.940,1.566], [1.486,0.759],  
[1.266,1.106],[1.540,0.419],[0.459,1.799],[0.773,0.186]])
```

```
y=np.array([0,1,1,0,1,0,1,1,1])
```

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(X,y)
```

```
kmeans.predict([[0.906, 0.606]])
```

OUTPUT:

```
array([0], dtype=int32)           // Given Input belongs to 0th cluster
```

Experiment 5

5. The following training examples map descriptions of individuals onto high, medium and low credit-worthiness.

medium skiing design single twenties no -> highRisk
high golf trading married forties yes -> lowRisk
low speedway transport married thirties yes -> medRisk
medium football banking single thirties yes -> lowRisk
high flying media married fifties yes -> highRisk
low football security single twenties no -> medRisk
medium golf media single thirties yes -> medRisk
medium golf transport married forties yes -> lowRisk
high skiing banking single thirties yes -> highRisk
low golf unemployed married forties yes -> highRisk

Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner.
Find the unconditional probability of 'golf' and the conditional probability of 'single' given 'medRisk' in the dataset?

PROGRAM:

```
totalRecords=10
numberGolfRecreation=4
probGolf=numberGolfRecreation/totalRecords
print("Unconditional probability of golf: ={}".format(probGolf))
#conditional probability of 'single' given 'medRisk'
# bayes Formula
# $p(\text{single}|\text{medRisk}) = \frac{p(\text{medRisk}|\text{single})p(\text{single})}{p(\text{medRisk})}$ 
# $p(\text{medRisk}|\text{single}) = \frac{p(\text{medRisk} \cap \text{single})}{p(\text{single})}$ 
numberMedRiskSingle=2
numberMedRisk=3
probMedRiskSingle=numberMedRiskSingle/totalRecords
probMedRisk=numberMedRisk/totalRecords
conditionalProbability=(probMedRiskSingle/probMedRisk)
print("Conditional probability of single given medRisk: = {}".format(conditionalProbability))
```

OUTPUT:

Unconditional probability of golf: =0.4
Conditional probability of single given medRisk: = 0.6666666666666667

Experiment 6

6. Implement linear regression using python

ALGORITHM:

Step 1: Create Database for Linear Regression

Step 2: Finding Hypothesis of Linear Regression

Step 3: Training a Linear Regression model

Step 4: Evaluating the model

Step 5: Scikit-learn implementation

Step 6: End

PROGRAM:

Importing Necessary Libraries

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# generate random data-set

np.random.seed(0)

x = np.random.rand(100, 1) #Generate a 2-D array with 100 rows, each row containing 1
random numbers:

y = 2 + 3 * x + np.random.rand(100, 1)

regression_model = LinearRegression() # Model initialization

regression_model.fit(x, y) # Fit the data(train the model)

y_predicted = regression_model.predict(x) # Predict

# model evaluation

rmse = mean_squared_error(y, y_predicted)

r2 = r2_score(y, y_predicted)
```

```

# printing values

print('Slope:', regression_model.coef_)

print('Intercept:', regression_model.intercept_)

print('Root mean squared error: ', rmse)

print('R2 score: ', r2)

# plotting values # data points

plt.scatter(x, y, s=10)

plt.xlabel('x-Values from 0-1')

plt.ylabel('y-values from 2-5')

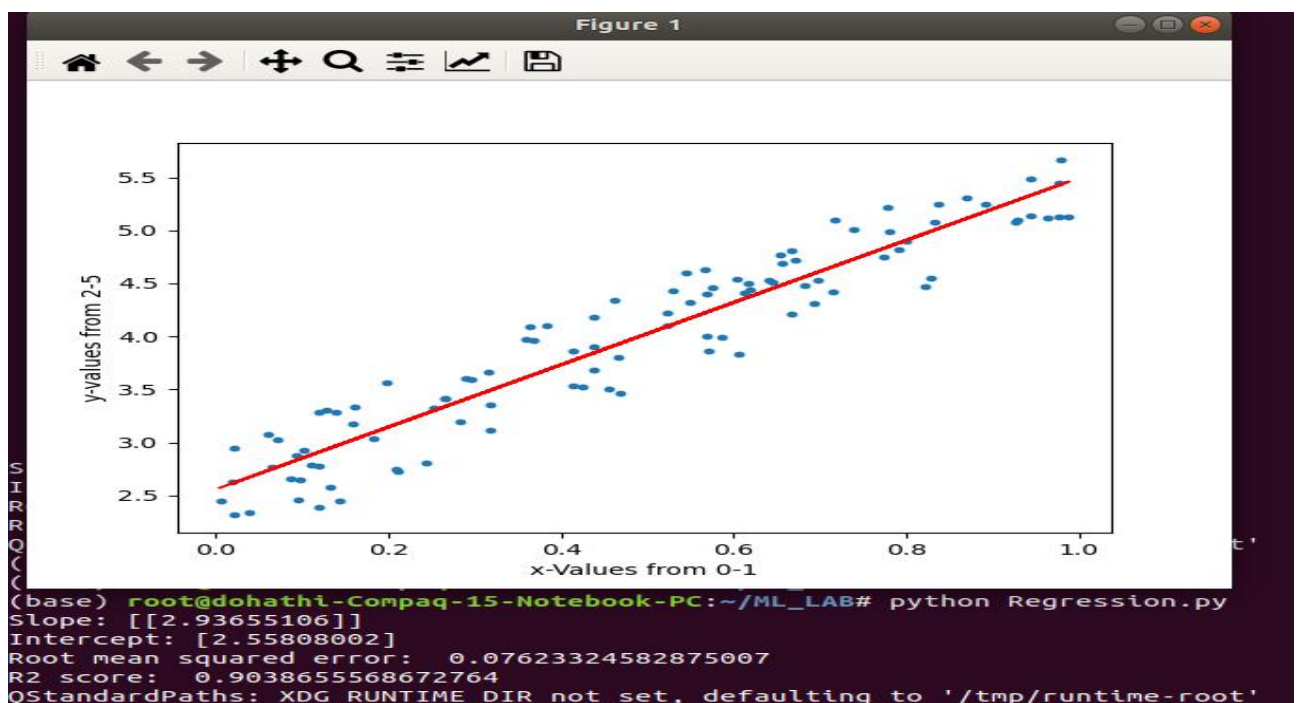
# predicted values

plt.plot(x, y_predicted, color='r')

plt.show()

```

OUTPUT:



Experiment 7

7. Implement Naïve Bayes Theorem to Classify the English Text using python

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of [Bayes' Theorem](#)

Dealing with text data

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
'this']

print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

The values 0,1,2, encode the frequency of a word that appeared in the initial text data.

E.g. The first transformed row is [0 1 1 1 0 0 1 0 1] and the **unique vocabulary** is ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this'], thus this means that the words “document”, “first”, “is”, “the” and “this” appeared 1 time each in the initial text string (i.e. ‘This is the first document.’).

In our example, we will convert the collection of text documents (train and test sets) into a matrix of token counts.

To implement that text transformation we will use the `make_pipeline` function. This will internally transform the text data and then the model will be fitted using the transformed data.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

So to solve the text classification problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.

Now, use Bayes theorem to calculate the posterior probability.

Source Code

```
print("NAIVE BAYES ENGLISH TEST CLASSIFICATION")

import numpy as np, pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score

sns.set() # use seaborn plotting style
```

```

# Load the dataset
data = fetch_20newsgroups()# Get the text categories
text_categories = data.target_names# define the training set
train_data = fetch_20newsgroups(subset="train", categories=text_categories)# define
the test set
test_data = fetch_20newsgroups(subset="test", categories=text_categories)

print("We have {} unique classes".format(len(text_categories)))
print("We have {} training samples".format(len(train_data.data)))
print("We have {} test samples".format(len(test_data.data)))

# let's have a look as some training data let it 5th only
#print(test_data.data[5])

# Build the model
model = make_pipeline(TfidfVectorizer(), MultinomialNB())# Train the model using the
training data
model.fit(train_data.data, train_data.target)# Predict the categories of the test data
predicted_categories = model.predict(test_data.data)

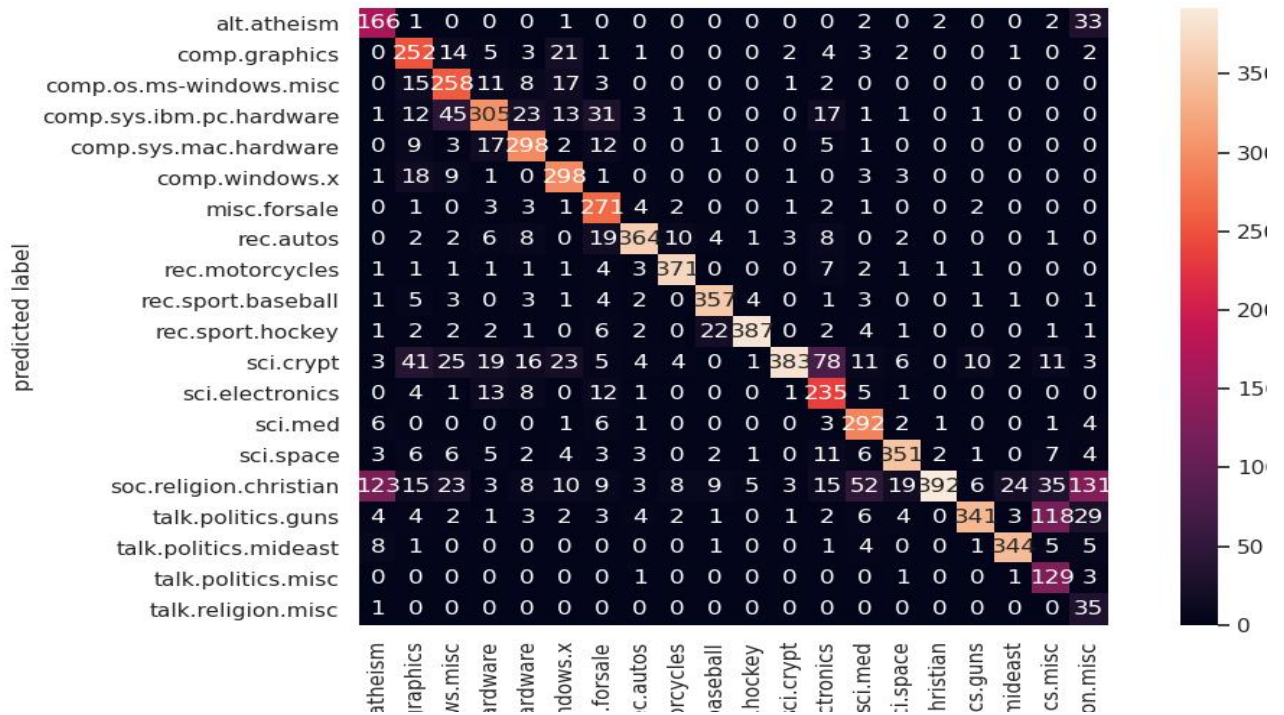
print(np.array(test_data.target_names)[predicted_categories])

# plot the confusion matrix
mat = confusion_matrix(test_data.target, predicted_categories)
sns.heatmap(mat.T, square = True, annot=True, fmt = "d",
xticklabels=train_data.target_names,yticklabels=train_data.target_names)
plt.xlabel("true labels")
plt.ylabel("predicted label")
plt.show()
print("The accuracy is {}".format(accuracy_score(test_data.target,
predicted_categories)))

```

OUTPUT:

```
L_Programs$ python NB_NaiveBayes.py
Whether: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Features: [(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0),
(1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
Predicted Value for the input 0:Overcast, 2:Mild: [1]
NAIVE BAYES ENGLISH TEST CLASSIFICATION
We have 20 unique classes
We have 11314 training samples
We have 7532 test samples
```

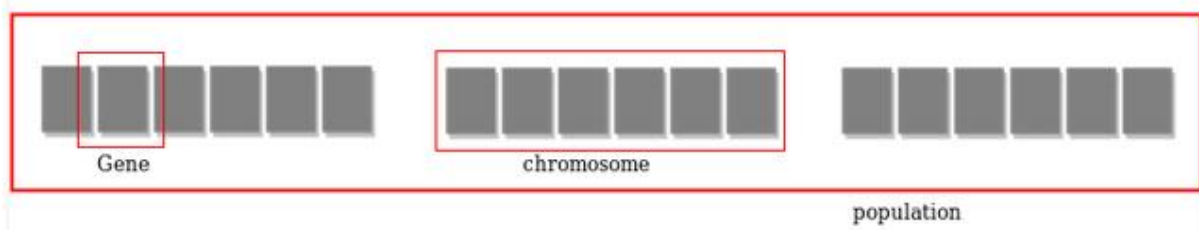


Experiment 8

8. Implement an algorithm to demonstrate the significance of Genetic Algorithm in python

ALGORITHM:

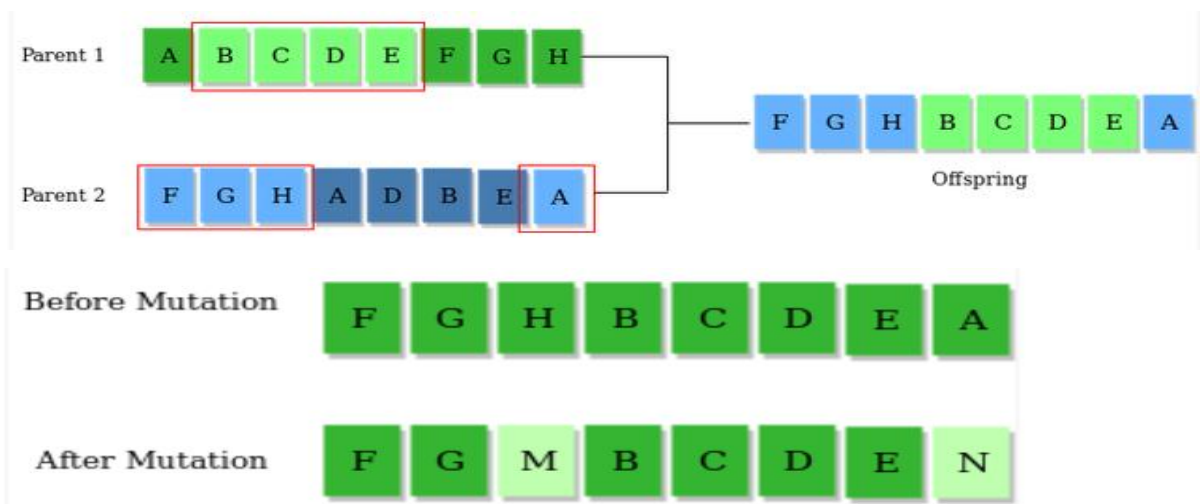
1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.



Operators of Genetic Algorithms

Once the initial generation is created, the algorithm evolve the generation using following operators –

- 1) Selection Operator:** The idea is to give preference to the individuals with good fitness scores and allow them to pass there genes to the successive generations.
- 2) Crossover Operator:** This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).
- 3) Mutation Operator:** The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence.



Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9 and other special symbols are considered as genes
- A string generated by these character is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

Source Code

Python3 program to create target string, starting from
random string using Genetic Algorithm

import random

Number of individuals in each generation

POPULATION_SIZE = 100

Valid genes

GENES = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
QRSTUVWXYZ 1234567890, .-:;_!"#%&/()=?@\${[]}"

Target string to be generated

TARGET = "I love GeeksforGeeks"

class Individual(object):

'''

Class representing individual in population '''

def __init__(self, chromosome):

self.chromosome = chromosome

self.fitness = self.cal_fitness()

@classmethod

def mutated_genes(self):

'''

create random genes for mutation

'''

global GENES

gene = random.choice(GENES)

return gene

@classmethod

def create_gnome(self):

```

'''
create chromosome or string of genes
'''

global TARGET
genome_len = len(TARGET)
return [self.mutated_genes() for _ in range(genome_len)]

def mate(self, par2):
    ''' Perform mating and produce new offspring '''

    # chromosome for offspring
    child_chromosome = []
    for gp1, gp2 in zip(self.chromosome, par2.chromosome):

        # random probability
        prob = random.random()

        # if prob is less than 0.45, insert gene
        # from parent 1
        if prob < 0.45:
            child_chromosome.append(gp1)

        # if prob is between 0.45 and 0.90, insert
        # gene from parent 2
        elif prob < 0.90:
            child_chromosome.append(gp2)

        # otherwise insert random gene(mutate),
        # for maintaining diversity
        else:
            child_chromosome.append(self.mutated_genes())

    # create new Individual(offspring) using
    # generated chromosome for offspring
    return Individual(child_chromosome)

def cal_fitness(self):
    ''' Calculate fitness score, it is the number of
    characters in string which differ from target string. '''
    global TARGET
    fitness = 0
    for gs, gt in zip(self.chromosome, TARGET):
        if gs != gt: fitness+= 1
    return fitness

# Driver code
def main():
    global POPULATION_SIZE

```

```

#current generation
generation = 1

found = False
population = []

# create initial population
for _ in range(POPULATION_SIZE):
    gnome = Individual.create_gnome()
    population.append(Individual(gnome))

while not found:

    # sort the population in increasing order of fitness score
    population = sorted(population, key = lambda x:x.fitness)

    # if the individual having lowest fitness score ie.
    # 0 then we know that we have reached to the target
    # and break the loop
    if population[0].fitness <= 0:
        found = True
        break

    # Otherwise generate new offsprings for new generation
    new_generation = []

    # Perform Elitism, that mean 10% of fittest population
    # goes to the next generation
    s = int((10*POPULATION_SIZE)/100)
    new_generation.extend(population[:s])

    # From 50% of fittest population, Individuals
    # will mate to produce offspring
    s = int((90*POPULATION_SIZE)/100)
    for _ in range(s):
        parent1 = random.choice(population[:50])
        parent2 = random.choice(population[:50])
        child = parent1.mate(parent2)
        new_generation.append(child)

    population = new_generation

    print("Generation: {}\\tString: {}\\tFitness: {}".\\
        format(generation,
            "".join(population[0].chromosome),
            population[0].fitness))

```

```
generation += 1
```

```
print("Generation: {}\\tString: {}\\tFitness: {}".\\  
      format(generation,  
            "".join(population[0].chromosome),  
            population[0].fitness))
```

```
if __name__ == '__main__':  
    main()
```

OUTPUT:

```
Generation: 1      String: t0{"-?=jH[k8=B4]0e@}      Fitness: 18  
Generation: 2      String: t0{"-?=jH[k8=B4]0e@}      Fitness: 18  
Generation: 3      String: .#lRWf9k_Ifslw #0$k_      Fitness: 17  
Generation: 4      String: .-1Rq?9mHqk3Wo]3rek_      Fitness: 16  
Generation: 5      String: .-1Rq?9mHqk3Wo]3rek_      Fitness: 16  
Generation: 6      String: A#ldW) #lIkslw cVek)      Fitness: 14  
Generation: 7      String: A#ldW) #lIkslw cVek)      Fitness: 14  
Generation: 8      String: (, o x _x%Rs=, 6Peek3      Fitness: 13  
                  .  
                  .  
                  .  
Generation: 29     String: I lope Geeks#o, Geeks      Fitness: 3  
Generation: 30     String: I loMe GeeksfoBGeeks      Fitness: 2  
Generation: 31     String: I love Geeksfo0Geeks      Fitness: 1  
Generation: 32     String: I love Geeksfo0Geeks      Fitness: 1  
Generation: 33     String: I love Geeksfo0Geeks      Fitness: 1  
Generation: 34     String: I love GeeksforGeeks      Fitness: 0
```


Experiment 9

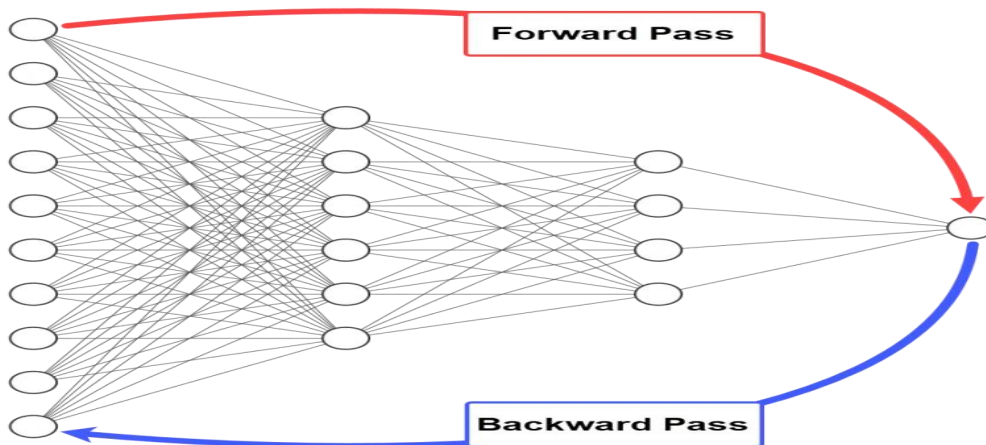
9. Implement an algorithm to demonstrate Back Propagation Algorithm in python

Back Propagation ALGORITHM:

It is the most widely used algorithm for training artificial neural networks.

In the simplest scenario, the architecture of a neural network consists of some sequential layers, where the layer numbered i is connected to the layer numbered $i+1$. The layers can be classified into 3 classes:

1. Input
2. Hidden
3. Output



Usually, each neuron in the hidden layer uses an activation function like sigmoid or rectified linear unit (ReLU). This helps to capture the non-linear relationship between the inputs and their outputs.

The neurons in the output layer also use activation functions like sigmoid (for regression) or SoftMax (for classification).

To train a neural network, there are 2 passes (phases):

- Forward
- Backward

The forward and backward phases are repeated from some epochs. In each epoch, the following occurs:

1. The inputs are propagated from the input to the output layer.
2. The network error is calculated.
3. The error is propagated from the output layer to the input layer.

Knowing that there's an error, what should we do? We should minimize it. To minimize network error, we must change something in the network.

Remember that the only parameters we can change are the weights and biases. We can try different weights and biases, and then test our network.

Advantages of Backpropagation in Python

It is relatively faster and simple algorithm to implement.

Extensively used in the field of face recognition and speech recognition.

Moreover, it is a flexible method as no prior knowledge of the neural network is needed.

Disadvantages of Backpropagation

The algorithm is not advantageous for noisy and irregular data.

The performance of the backpropagation highly depends on the input.

Source Code:

```
import numpy
import matplotlib.pyplot as plt

def sigmoid(sop):
    return 1.0/(1+numpy.exp(-1*sop))

def error(predicted, target):
    return numpy.power(predicted-target, 2)

def error_predicted_deriv(predicted, target):
    return 2*(predicted-target)

def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))
```

```
def sop_w_deriv(x):
```

```
    return x
```

```
def update_w(w, grad, learning_rate):
```

```
    return w - learning_rate*grad
```

```
x1=0.1
```

```
x2=0.4
```

```
target = 0.7
```

```
learning_rate = 0.01
```

```
w1=numpy.random.rand()
```

```
w2=numpy.random.rand()
```

```
print("Initial W : ", w1, w2)
```

```
predicted_output = []
```

```
network_error = []
```

```
old_err = 0
```

```
for k in range(80000):
```

```
    # Forward Pass
```

```
    y = w1*x1 + w2*x2
```

```
    predicted = sigmoid(y)
```

```
    err = error(predicted, target)
```

```
    predicted_output.append(predicted)
```

```
    network_error.append(err)
```

```
    # Backward Pass
```

```
    g1 = error_predicted_deriv(predicted, target)
```

```
g2 = sigmoid_sop_deriv(y)
```

```
g3w1 = sop_w_deriv(x1)
```

```
g3w2 = sop_w_deriv(x2)
```

```
gradw1 = g3w1*g2*g1
```

```
gradw2 = g3w2*g2*g1
```

```
w1 = update_w(w1, gradw1, learning_rate)
```

```
w2 = update_w(w2, gradw2, learning_rate)
```

```
#print(predicted)
```

```
plt.figure()
```

```
plt.plot(network_error)
```

```
plt.title("Iteration Number vs Error")
```

```
plt.xlabel("Iteration Number")
```

```
plt.ylabel("Error")
```

```
plt.show()
```

```
plt.figure()
```

```
plt.plot(predicted_output)
```

```
plt.title("Iteration Number vs Prediction")
```

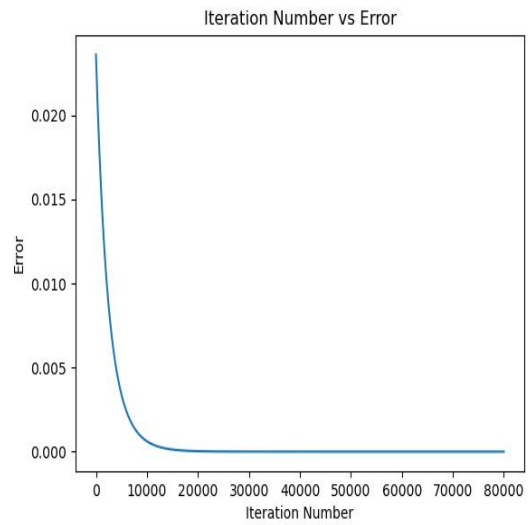
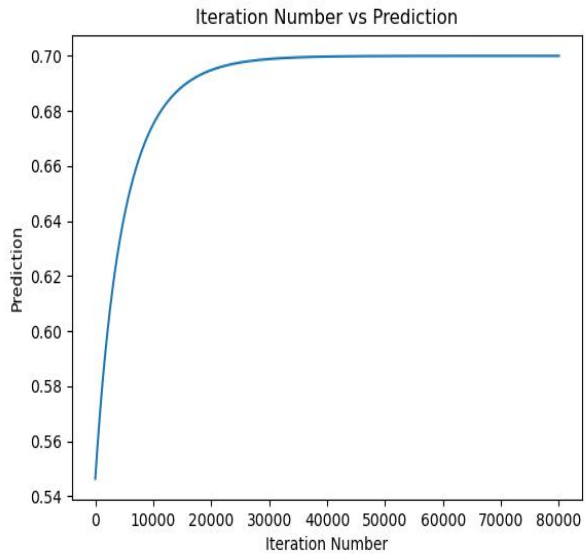
```
plt.xlabel("Iteration Number")
```

```
plt.ylabel("Prediction")
```

```
plt.show()
```

OUTPUT:

Initial W : 0.08698924153243281 0.4532713230157145



Experiment 10

10. Implementing FIND-S algorithm using python

Training Database

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

TABLE 2.1

Positive and negative training examples for the target concept *EnjoySport*.

Algorithm

1. Initialize h to the most specific hypothesis in H
 2. For each positive training instance x
 For each attribute constraint a , in h
 If the constraint a , is satisfied by x
 Then do nothing
 Else replace a , in h by the next more general constraint that is satisfied by x
 3. Output hypothesis h
-

Hypothesis Construction

$$\begin{array}{ll} h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle & \\ x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, + & h_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle \\ x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, + & h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle \\ x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, - & h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle \\ x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, + & h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle \end{array}$$

Source Code:

```
import csv
with open('enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
    if a[i][num_attribute] == 'Yes':          #for each positive example only
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is : \n".format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

OUTPUT:

```
[kln@localhost ML_Programs]$ python FindS.py
[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'TRUE'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'TRUE'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'FALSE'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'TRUE']]
(\n The total number of training instances are : ', 4)

The initial hypothesis is :
['0', '0', '0', '0', '0', '0']
(\n The hypothesis for the training instance 1 is : \n', ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'])
(\n The hypothesis for the training instance 2 is : \n', ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same'])
(\n The hypothesis for the training instance 3 is : \n', ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same'])
(\n The hypothesis for the training instance 4 is : \n', ['Sunny', 'Warm', '?', 'Strong', '?', '?'])

The Maximally specific hypothesis for the training instance is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Experiment 11

11. Implementing Candidate Elimination algorithm using python

Training Database

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

TABLE 2.1

Positive and negative training examples for the target concept *EnjoySport*.

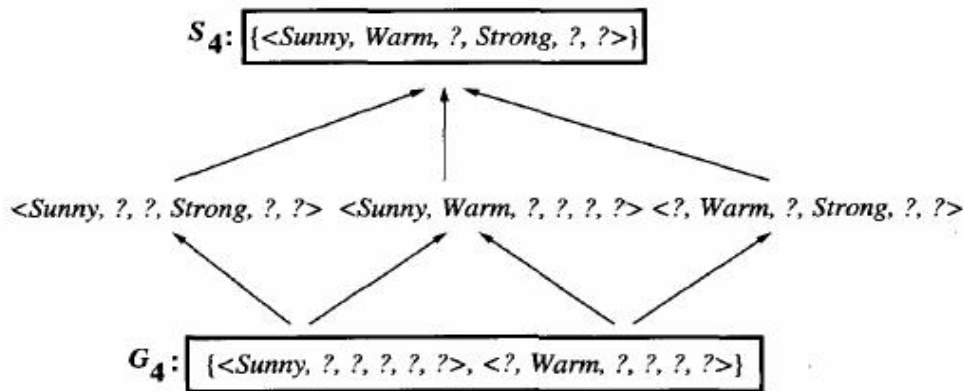
Algorithm

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d , do

- If d is a positive example
 - Remove from G any hypothesis 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 is consistent with d , and some member of G is more general than h
 - Remove from S any hypothesis that is more general than another hypothesis in S
 - If d is a negative example
 - Remove from S any hypothesis 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 is consistent with d , and some member of S is more specific than h
 - Remove from G any hypothesis that is less general than another hypothesis in G
-



Source
e

Code: **FIGURE 2.7**

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

```
import csv

with open("enjoysport.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)

    print(data)
    print("-----")
    s=data[1][:-1]    #extracting one row or instance or record
    g=[['?' for i in range(len(s))] for j in range(len(s))]

    print(s)
    print("-----")
    print(g)
    print("-----")

    for i in data:
        if i[-1]=="Yes": # For each positive training record or instance
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'

        elif i[-1]=="No": # For each negative training record or example
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]='?'
    print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
    print(s)
```

```

print(g)
gh=[]
for i in g:
    for j in i:
        if j!='?':
            gh.append(i)
            break
print("\nFinal specific hypothesis:\n",s)
print("\nFinal general hypothesis:\n",gh)

```

OUTPUT:

```

[kln@localhost ML_Programs]$ python CandidateElimination.py
[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'TRUE'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'TRUE'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'FALSE'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'TRUE']]
-----
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same']
-----
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
-----
('\nSteps of Candidate Elimination Algorithm', 1)
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
-----
('\nSteps of Candidate Elimination Algorithm', 2)
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
-----
('\nSteps of Candidate Elimination Algorithm', 3)
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
-----
('\nSteps of Candidate Elimination Algorithm', 4)
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
-----
('\nFinal specific hypothesis:\n', ['Sunny', 'Warm', '?', 'Strong', '?', '?'])
('\nFinal general hypothesis:\n', [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']])

```

Experiment 12

Implement K-Means_Clustering using python

ALGORITHM:

Step 1: Read the Given data Sample to X

Step 2: Train Dataset with K=5

Step 3: Find optimal number of clusters(k) in a dataset using Elbow method

Step 4: Train Dataset with K=3 (optimal K-Value)

Step 4: Compare results

Step 6: End

PROGRAM:

#Import libraries

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

#Read DataSet

```
df = datasets.load_iris()
x = df.data
y = df.target
```

```
print(x)
print(y)
```

#Lets try with k=5 initially

```
kmeans5 = KMeans(n_clusters=5)
y_kmeans5 = kmeans5.fit_predict(x)
print(y_kmeans5)
```

```
array([[5.006      , 3.418      , 1.464      , 0.244      ],
       [6.52916667, 3.05833333, 5.50833333, 2.1625      ],
       [5.508      , 2.6        , 3.908      , 1.204      ],
       [7.475      , 3.125      , 6.3        , 2.05       ],
       [6.20769231, 2.85384615, 4.74615385, 1.56410256]])
```



```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.418      , 1.464      , 0.244      ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

Experiment: 13

13. Implement Decision-Tree Learning algorithm using python.

Reference: <https://www.geeksforgeeks.org/decision-tree-implementation-python/>

Decision Tree Applications

1. A decision tree is used to determine whether an applicant is likely to default on a loan.
2. It can be used to determine the odds of an individual developing a specific disease.
3. It can help ecommerce companies in predicting whether a consumer is likely to purchase a specific product.
4. Decision trees can also be used to find customer churn rates.

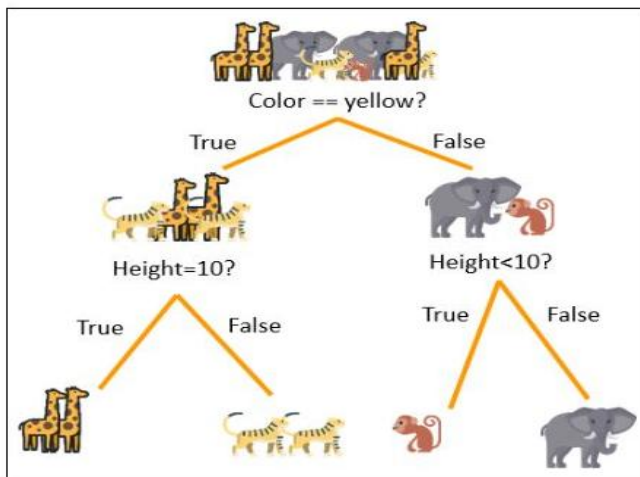


Fig: Decision tree to classify animals

Training Dataset		
Color	Height	Label
Grey	10	Elephant
Yellow	10	Giraffe
Brown	3	Monkey
Grey	10	Elephant
Yellow	4	Tiger

ALGORITHM:

Assumptions we make while using Decision tree :

- At the beginning, we consider the whole training set as the root.
- Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or internal node.

Pseudocode :

1. Find the best attribute and place it on the root node of the tree.
2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

PROGRAM:

```
# Importing the required packages

import numpy as np
import pandas as pd

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
//from sklearn import tree

# Function importing Dataset

def importdata():

    balance_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/balance-scale/balance-scale.data',sep= ',', header = None)

    # Printing the dataset shape
    print ("Dataset Length: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)

    # Printing the dataset observations
    print ("Dataset: ",balance_data.head())
```

```

    return balance_data

# Function to split the dataset
def splitdataset(balance_data):

    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size = 0.3, random_state = 100)

    return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
                                     random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)

    return clf_gini

# Function to perform training with entropy.

```



```

def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy

    clf_entropy = DecisionTreeClassifier(
        criterion = "entropy", random_state = 100,
        max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)

    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)

    return y_pred


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
          confusion_matrix(y_test, y_pred))

```

```

print ("Accuracy : ",
accuracy_score(y_test,y_pred)*100)

print("Report : ",
classification_report(y_test, y_pred))

# Driver code
def main():

    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)

```

```
# Calling main function

if __name__=="__main__":

    main()
```

OUTPUT:

```
Dataset Length: 625
Dataset Shape: (625, 5)
Dataset:      0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix: [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy : 73.40425531914893
Report :
           precision    recall  f1-score   support

      B         0.00         0.00         0.00         13
      L         0.73         0.79         0.76         85
      R         0.74         0.79         0.76         90

 accuracy          0.73         188
 macro avg         0.49         0.53         0.51         188
weighted avg         0.68         0.73         0.71         188
```

```
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R']
```

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]

[0 63 22]

[0 20 70]]

Accuracy : 70.74468085106383

Report : precision recall f1-score support

B 0.00 0.00 0.00 13

L 0.71 0.74 0.72 85

R 0.71 0.78 0.74 90

accuracy 0.71 188

macro avg 0.47 0.51 0.49 188

weighted avg 0.66 0.71 0.68 188