# 

# B.V.V. Sangha’s

**Biluru Gurubasava Mahaswamiji Institute of Technology, Mudhol**

# Department of Computer Science and Engineering

# Machine Learning

# Laboratory Manual

# Course Code:17CSL76

# Semester: 7th

# Prepared by

# Dr.Niranjan R Chougala Mr.Sandeep S Kopp

# Professor Programmer

# Dept. of CSE,BGMIT Dept. of CSE,BGMIT

# AY: 2020-21

# Machine learning Course objectives:

This course will enable students to

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

# Description:

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

# Syllabus-

Course Code: 15CSL76 IA Marks: 20

Number of Lecture Hours/Week: 01I+02P Exam Marks: 80

Total Number of Lecture Hours: 40 Exam Hours: 03

# Laboratory Experiments:

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
4. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.
5. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test datasets.
6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using *k*-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
9. Write a program to implement *k*-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

# Course outcomes:

The students should be able to:

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

# Conduction of Practical Examination:

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

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

**Revision to few Concepts in Python**

IF statement: The if statement contains a logical expression using which the data is compared and a decision is made based on the result.

Syntax:

if expression:

statement(s)

**Sample: e.g**. a = 10

if a:

print("1-got a true exp") print(a)

b = 0 if b:

print("2 -got a true exp") print(b) print("exiting...")

IF..ELSE**:** An else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The else statement is an optional statement and there could be at the most only one **else**

statement following **if.**

# ´Syntax:

if expression: statement(s) else:

statement(s)

# Sample: e.g.

score = int(input("enter the input: ")) if score<50: print("fail grade:")

else:

print("Pass grade:")

Elif :The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

# Syntax:

if expression1:

statement(s) elifexpression2:

statement(s) elifexpression3:

else:

statement(s)statement(s)

# Sample: e.g

score = float(input("enter the input: ")) if score<=40.5: grade = score \* 0.05

print("fail grade:", grade) elif score<70.0:

grade = score \* 0.10 print("Pass grade:", grade) else:

grade = score \* 0.15 print("outstanding grade:",grade) print("end")

Similar to the else, the elif statement is optional. However, unlike else, for which there can be at the most one statement, there can be an arbitrary number of elif statements following an if.

Core Python does not provide switch or case statements as in other languages, but we can use if..elif...statements to simulate switch case.

Nested If-

# Syntax:

if expression1

statement(s)

if expression2: statement(s)

elif expression3: statement(s) else statement(s)

elif expression4: statement(s) else: statement(s)

# Sample: e.g

num=int(input("enter a number")) if num%2==0: if num%3==0:

print("number is divisibe by 3 and2") else:

print("number divisible by 2 not by 3") else:

if num%3==0:

print("number divisible by 3 not by 2") else:

print("number not divisible by 2 and not by 3")

while loop- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition istrue.

# Syntax:

while expression: statement(s)

# Sample: e.g.

count=0 while(count<10):

print("count is :",count) count=count+1

Infinite loop-**Sample: e.g.** var = 1

while var == 1 : # This constructs an infinite loop num = int(input("Enter a number :"))

print ("You entered: ", num) print ("Good bye!")

Using else statements with loops-

If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

While loop can be terminated with a break statement. In such a case, the else part is ignored. Thus a while loop‟s else part runs if no break occurs and condition is false.

# Sample: e.g.

count=1 while(count<=3): print("loop inside:") count=count+1 else:

print("false part") print("exit")

for loop- The for statement in Python has the ability to iterate over the items of any sequence.

# ´Syntax:

for iterating\_var in sequence: statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating\_var*.

Next, the statements block is executed.

Each item in the list is assigned to *iterating\_var*, and the statement(s) block is executed until the entire sequence is exhausted.

# Sample: e.g.

for letter in 'Python':

print ('Current Letter :', letter)

Nested for- We need to place a loop inside another loop. It can be used on for & while construct.

# Syntax:

for iterating\_var in sequence:

for iterating\_var in sequence: statements(s)

statement(s)

# Sample: e.g.

for i in range(1,5): for j in range(1,5): k=i\*j

print (k,end='\n') #print ("\n",k) print()

The print() function inner loop has end=' ' which appends a space instead of default newline. Hence, the numbers will appear in one row.

Nested while-

# Syntax:

while expression: while expression: statements(s) statement(s)

# Sample: e.g.

n=int(input("enter a value:")) i=1; while i<=n: k=round(n) j=4

while j<=k:

if ( I % j==0): j+=1

else: print (i) i+=1 Lists

Lists are the most versatile of Python's compound data types. Lists are ordered sequence of items.

A list contains items separated by commas and enclosed within square brackets ([]).

To some extent, lists are similar to arrays in C but are different by allowing the items to be of different types. (integer, float, string etc.).

# List creation:

1. Empty list : my\_list = []
2. List of integers: my\_list = [ 1,2,3 ]
3. List of floating type : my\_list = [ 2.2, 4.0, 99.99, .45]
4. List of mixed data types: my\_list = [ 1, “hai”, 9.99] 5. Nested list : [ “ hello” , [ 3, 5 , 7.7 ] , [ „z‟ ] ] **Accessing ListElements**

Lists can be accessed in two ways:

# List Index

Negative Indexing

In case of List Index technique use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

# E.g.

my\_list = [ 1 , 2, 3, 4, 5 ]

Now to access 3rdelement : print (my\_list[2])

In case of nested lists we use nested indexing as shown in example.

# E.g .

nes\_list = [ “welcome” , [ 2 , 0 , 1, 8 ] ]

Now to print 0 , we have to write : print (nes\_list [1][1] )

# Negative Indexing

In case of Negative Indexing technique use the index operator [] to access an item in a list. Index starts from -1. So, a list having 5 elements will have index from -1 to -5.

# E.g.

my\_list = [ „d‟,‟j‟,‟k‟,‟I‟,‟n‟,‟g‟ ]. Now to print k: print (my\_list[-4])

In case of nested lists we use nested indexing as shown in example.

# E.g .

my\_list = ['d007','j','k','I','n','g'], [1,2,3,8.8,5.5]

Now to print 8.8 , we have to write : print(my\_list[-1][3]) Now to print d007, we have to write print(my\_list[-2][-6])

# Slicing Lists-

It is an operation that extracts certain elements from a list and forms a sub list possibly with a different number of indices and different index ranges.

The syntax for list slicing is :**[start:end:step]**

where start, end, step parts of the syntax are integers and optional. They can be both positive and negative.

Note that the value having the end index is not included in the slice.

# E.g:

**n = [1, 2, 3, 4, 5, 6, 7, 8]**

**print(n[1:5])** has values with indexes 1,2,3,and 4. so output is : [2,3,4,5]

**print(n[:5])** has values starting from the default index i.e. 0 so output is: [1,2,3,4,5] **print(n[1:])** has values starting from index 1 till the default end index i.e. -1 so the output Is : [2,3,4,5,6,7,8] excluding 1 since it is a value at index 0.

**print(n[:])** has all the values starting from default start index i.e. 0 to default end index i.e. -1 so the output is : [1,2,3,4,5,6,7,8].

The third index in a slice syntax is the step. It allows us to take every n-th value from a list.

# E.g: n = [1, 2, 3, 4, 5, 6, 7,8]

**print(n[1:9:2]) à** a slice having every 2nd elementfrom the n list starting from 2nd element

,ending at 8th element is created. So output is [2,4,6,8].

**print(n[::2]) à** a slice is built taking every 2nd value from the beginning till end of list. So the output is [ 1,3,5,7].

**print(n[::1]) à** this is just creating a copy of the list. So output is [1,2,3,4,5,6,7,8]. **print(n[1::3]) à** a slice is built for every 3rd element starting from 2nd element till the end of list. So output of the list is[2,5,8].

Indexes can be negative numbers. Indexes with lower negative numbers must come first in the syntax.

E.g**: print(n[-1:-4])à** returns an empty list. Now change index to **print(n[-4:-1])** to get the output [5,6,7].

**print(n[::-1])à** creates a reversed list so output is [8,7,6,5,4,3,2,1].

# Adding List Elements-

The elements can be added to the list by using built-in functions and methods or by specifying index value or also by using slicing.

# E.g:

1. even=[2,4,6,8] Now to insert a value 1 instead of 2 we have to just write:**even[0]=1**.
2. Now to change the list to contain only odd numbers using slice operation I canwrite

**even[1:4]=[3,5,7]** so that now print(even) will output[1,3,5,7].

1. Now to add an item 9 to the list using built-in method I have towrite:

**even.append(9)** so the value 9 is added to the end of list automatically.

1. Suppose I want to add 11, 13,15 also to the list then I have to use another built-in method extend to add all items together to the list and not individually. So **even.extend([11,13,15])** adds 3 more elements to the list. On printing the list finally we have **print(even) ->**[1,3,5,7,9,11,13,15] as output.
2. We can insert an item at a desired location by using a built-in method insert(). E.g: even=[2,10] so we write **even.insert(1,4)** to get sublist as[2,4,10].

# Deleting List elements-

The elements can be removed from the list by using built-in functionsandmethods or by specifying index value using a **keyword del**or also by usingslicing.

# E.g:

1. my\_list=[„d‟,‟j‟,‟007‟,‟k‟,‟I‟,‟n‟,‟g‟]sotodeleteanitem„007‟wewrite**delmy\_list[2]**.
2. Nowtodeletemultipleitemsfromthelistwewrite**delmy\_list[2:7]**toget[„d‟,‟j‟].
3. Now to delete entire list using del keyword we write **delmy\_list**.
4. Now if you want an empty list then we write **delmy\_list[:].**
5. Nowtoremoveanitemfromthelistusingbuilt-inmethodwewrite**my\_list.remove(„007‟)**

´ We can use another method pop() to remove an item from the list. The pop()method removes or returns the last item if index is not provided.

1. **print(my\_list.pop(-2))** will remove „n‟ from thelist.
2. **print(my\_list.pop(1))** will remove the 2nd element from the list with index being1.
3. **print(my\_list.pop())** will remove the last item from thelist.

**my\_list.clear()** will clear the contents and output an empty list.

# Lists and Functions-

There are a number of built-in functions that can be used on lists that allow you to quickly look through a list without writing your own loops:

nums = [3, 41, 12, 9, 74, 15]

print(len(nums)) 6

print(max(nums)) 74

print(min(nums)) 3

print(sum(nums)) 154

print(sum(nums)/len(nums)) 25

# Lists and Strings-

A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as astring.

To convert from a string to a list of characters, you can use list: s = 'spam'

t = list(s) print(t)

['s', 'p', 'a', 'm']

The list function breaks a string into individual letters.

If you want to break a string into words, you can use the split method: s = 'pining for the fjords'

t = s.split() print(t)

['pining', 'for', 'the', 'fjords'] print(t[2]) the

# Aliasing-

If a refers to an object and you assign b = a, then both variables refer to the same object: a = [1, 2, 3]

b = a b is a True

The association of a variable with an object is called a *reference.*

An object with more than one reference has more than one name, so we say that the object is

*aliased.*

If the aliased object is mutable, changes made with one alias affect the other: b[0] = 17

print(a) [17, 2, 3]

Although this behavior can be useful, it is error-prone.

In general, it is safer to avoid aliasing when you are working with mutable objects.

# Dictionary-

Python's dictionaries are collection of key-value pairs. (also called as items)

A dictionary key can be of almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }).

Values can be assigned and accessed using square braces ([]). Duplicate keys are not allowed and keys are immutable.

dict = {}

dict['one'] = "This is one" dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'} print (dict['one']) # Prints value for 'one' key print (dict[2]) # Prints value for 2 key

The order of the key-value pairs is not the same.

In general, the order of items in a dictionary is unpredictable.

To see whether something appears as a value in a dictionary, you can use the method values, which returns the values as a list, and then use the in operator:

vals = list(tinydict.values()) john' in vals

True

vals = list(tinydict.values()) 9999' in vals

False

The in operator uses different algorithms for lists and dictionaries. For lists, it uses a linear search algorithm.

As the list gets longer, the search time gets longer in direct proportion to the length of the list. For dictionaries, Python uses an algorithm called a *hash table that has a remarkable property.* The in operator takes about the same amount of time no matter how many items there are in a dictionary.

# Break Statement-

Example-

for letter in 'python': if letter == 'h': break

print("current letter is :",letter)

# Continue Statement-

The continue statement in Python returns the control to the beginning of the current loop.

When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

The break statement can be used in both *while and for loops.*

# Sample: e.g.

for letter in 'python': if letter == 'h': continue

print("current letter is :",letter)

# Pass Statement-

In python, pass statement is a null operation; nothing happens when it executes. It is used as a place holder.

Whenever we want to implement a loop or a function in future but eventually now you do not want to code anything then use a pass statement since the python interpreter will not allow you to have an emptybody.

# Sample: e.g.

for letter in 'python': if letter == 'h': pass

print (“this is pass block”) print("current letter is :",letter) print(“good bye!”)

# CSV File Reading and Writing-

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. There is no “CSV standard”, so the format is operationally defined by the many applications which read and writeit.

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used byExcel.

The csv module‟s reader and writer objects read and write sequences.

Programmers can also read and write data in the DictReader and DictWriter classes.

# Reading a csv file-

In python, we use csv.reader() module to read the csv file. Here, we will show you how to read different types of csv files with different delimiter

like quotes(""), pipe(|) and comma(,).

We have a csv file called **pwd.csv** having default delimiter comma(,) with following data:

SN, Name, City

1, John, Washington 2, Eric, Los Angeles 3, Brad, Texas

# Reading a csv file, where the delimiter is a comma

import csv

with open('pwd.csv', 'r') as csvFile: reader = csv.reader(csvFile) for row in reader:

print(row) csvFile.close()

# Reading a csv file, where the delimiter is |

import csv

csv.register\_dialect('myDialect', delimiter = '|') with open('rwd.csv', 'r') as f:

reader = csv.reader(f, dialect='myDialect') for row in reader: print(row)

# Extract specific data from the spreadsheet into lists

import csv

with open('Example.csv') as csvfile:

readCSV = csv.reader(csvfile, delimiter=',') dates = [ ] colors = [ ]

for row in readCSV: color = row[3] date = row[0] dates.append(date) colors.append(color) print(dates) print(colors)

# Writing to a csv file-

In python, we use csv.writer() module to write the csv file.

A csv file called some.csv having default delimiter comma(,) with following data will be created. 2

M,a,r,i,e C,a,l,i,f,o,r,n,i,a

# Writing a csv file, where the delimiter is a comma

import csv

row = ['2', ' Marie', ' California']

with open('some.csv', 'w') as f: writer = csv.writer(f) writer.writerows(row) f.close()

# Writing a csv file, where the delimiter is |

import csv

person = [['SN', 'Person', 'DOB'],

['1', 'John', '18/1/2017'],

['2', 'Marie','19/2/2018'],

['3', 'Simon','20/3/2009']]

csv.register\_dialect('myDialect',delimiter = '|') with open('dob.csv', 'w') as f: writer = csv.writer(f, dialect='myDialect')

for row in person: writer.writerow(row) f.close()

# Machine Learning-

Machine learning is a subset of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) in the field of [computer science](https://en.wikipedia.org/wiki/Computer_science) that often uses statistical techniques to give [computers](https://en.wikipedia.org/wiki/Computer) the ability to "learn" (i.e., progressively improve performance on a specific task) with [data,](https://en.wikipedia.org/wiki/Data) without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the humangenome.

Data Output

Data

Output

Computer

Program(Model)

Computer

Program

**Fig a)**TraditionalProgramming **Fig b)** MachineLearning.

# Machine learning tasks

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

[**Supervised learning:**](https://en.wikipedia.org/wiki/Supervised_learning) The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback:

[**Semi-supervised learning:**](https://en.wikipedia.org/wiki/Semi-supervised_learning)The computer is given only an incomplete training signal: a training set with some (often many) of the target outputsmissing.

[**Active learning:**](https://en.wikipedia.org/wiki/Active_learning_(machine_learning))the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.

[**Reinforcement learning:**](https://en.wikipedia.org/wiki/Reinforcement_learning) training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

[**Unsupervised learning:**](https://en.wikipedia.org/wiki/Unsupervised_learning) No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

|  |  |  |
| --- | --- | --- |
| **Supervised learning** | **Un Supervised learning** | **Instance based** |
| **learning** |
| Find-s algorithm | EM algorithm |  |
| Candidate elimination  Algorithm | K means algorithm |  |
| Decision tree algorithm |  |
| Back propagation Algorithm | Locally weighted |
| Naïve Bayes Algorithm | Regression algorithm |
| K nearest neighbor |  |
| algorithm(lazy learning |  |
| algorithm) |  |

# Machine learning applications

In [**classification,**](https://en.wikipedia.org/wiki/Statistical_classification)inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more [(multi-label classification)](https://en.wikipedia.org/wiki/Multi-label_classification) of these classes. This is typically tackled in a supervised manner. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".

In [**regression,**](https://en.wikipedia.org/wiki/Regression_analysis)also a supervised problem, the outputs are continuous rather than discrete.

In [**clustering,**](https://en.wikipedia.org/wiki/Cluster_analysis) a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

[**Density estimation**](https://en.wikipedia.org/wiki/Density_estimation) finds the [distribution](https://en.wikipedia.org/wiki/Probability_distribution) of inputs in some space.

[**Dimensionality reduction**](https://en.wikipedia.org/wiki/Dimensionality_reduction) simplifies inputs by mapping them into a lower-dimensional space. [Topic modelling](https://en.wikipedia.org/wiki/Topic_modeling) is a related problem, where a program is given a list of [human language](https://en.wikipedia.org/wiki/Natural_language) documents and is tasked with finding out which documents cover similar topics.

# Machine learning Approaches Decision tree learning

Decision tree learning uses a [decision tree](https://en.wikipedia.org/wiki/Decision_tree) as a [predictive model,](https://en.wikipedia.org/wiki/Predictive_modelling) which maps observations about an item to conclusions about the item's target value.

# Association rule learning

Association rule learning is a method for discovering interesting relations between variables in large databases.

# Artificial neural networks

An [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by [biological neural networks.](https://en.wikipedia.org/wiki/Biological_neural_networks) Computations are structured in terms of an interconnected group of [artificial neurons,](https://en.wikipedia.org/wiki/Artificial_neuron) processing information using a [connectionist](https://en.wikipedia.org/wiki/Connectionism) approach to [computation.](https://en.wikipedia.org/wiki/Computation) Modern neural networks are [non-linear statistical](https://en.wikipedia.org/wiki/Non-linear) [data](https://en.wikipedia.org/wiki/Data_modeling) [modelling](https://en.wikipedia.org/wiki/Data_modeling) tools. They are usually used to model complex relationships between inputs and outputs, to [find patterns](https://en.wikipedia.org/wiki/Pattern_recognition) in data, or to capture the statistical structure in an unknown [joint](https://en.wikipedia.org/wiki/Joint_probability_distribution) [probability distribution](https://en.wikipedia.org/wiki/Joint_probability_distribution) between observed variables.

# Deep learning

Falling hardware prices and the development of [GPUs](https://en.wikipedia.org/wiki/GPU) for personal use in the last few years have contributed to the development of the concept of [deep learning](https://en.wikipedia.org/wiki/Deep_learning) which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing.

Some successful applications of deep learning are [computer vision](https://en.wikipedia.org/wiki/Computer_vision) and [speech recognition.](https://en.wikipedia.org/wiki/Speech_recognition)

# Inductive logic programming

Inductive logic programming (ILP) is an approach to rule learning using [logic programming](https://en.wikipedia.org/wiki/Logic_programming) as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical

database of facts, an ILP system will derive a hypothesized logic program that [entails](https://en.wikipedia.org/wiki/Entailment) all positive and no negative examples. [Inductive programming](https://en.wikipedia.org/wiki/Inductive_programming) is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as [functional programs.](https://en.wikipedia.org/wiki/Functional_programming)

# Support vector machines

Support vector machines (SVMs) are a set of related [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) methods used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression.](https://en.wikipedia.org/wiki/Regression_analysis) Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

# Clustering

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some pre designated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| of the data, | often defined by | some similarity | metric and | evaluated | | for | example |
| by internal | compactness (similarity between | | members | of | The same | | cluster) |
| and separation between different | | clusters. Other methods | | are | based | on estimated | |

density and graph connectivity. Clustering is a method of [unsupervised learning,](https://en.wikipedia.org/wiki/Unsupervised_learning) and a common technique for [statistical data analysis.](https://en.wikipedia.org/wiki/Statistics)

# Bayesian networks

A Bayesian network, belief network or directed acyclic graphical model is a [probabilistic graphical model](https://en.wikipedia.org/wiki/Graphical_model) that represents a set of [random variables](https://en.wikipedia.org/wiki/Random_variables) and their [conditional independencies](https://en.wikipedia.org/wiki/Conditional_independence) via a [directed acyclic graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph) (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform [inference](https://en.wikipedia.org/wiki/Inference) andlearning.

# Reinforcement learning

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

# Similarity and metric learning

In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric function) that can predict if new objects are similar. It is sometimes used in [Recommendation](https://en.wikipedia.org/wiki/Recommendation_systems) [systems.](https://en.wikipedia.org/wiki/Recommendation_systems)

# Genetic algorithms

A genetic algorithm (GA) is a [search heuristic](https://en.wikipedia.org/wiki/Search_algorithm) that mimics the process of [natural selection,](https://en.wikipedia.org/wiki/Natural_selection) and uses methods such as [mutation](https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)) and [crossover](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)) to generate new [genotype](https://en.wikipedia.org/wiki/Chromosome_(genetic_algorithm)) in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and [evolutionary algorithms.](https://en.wikipedia.org/wiki/Evolutionary_algorithm)

# Rule-based machine learning

[Rule-based machine learning](https://en.wikipedia.org/wiki/Rule-based_machine_learning) is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply, knowledge. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include [learning](https://en.wikipedia.org/wiki/Learning_classifier_system) [classifier systems, association rule learning,](https://en.wikipedia.org/wiki/Learning_classifier_system) and [artificial immune systems.](https://en.wikipedia.org/wiki/Artificial_immune_system)

# Feature selection approach

[Feature selection](https://en.wikipedia.org/wiki/Feature_selection) is the process of selecting an optimal subset of relevant features for use in model construction. It is assumed the data contains some features that are either redundant or irrelevant, and can thus be removed to reduce calculation cost without incurring much loss of information. Common optimality criteria include accuracy, similarity and information measures.

# Software Requirements-

1. Python 3.5 version and above.
2. Machine Learning Packages-
   * Scikit- Learn
   * Numpy- matrices and linear algebra.
   * Scipy- many numerical routines.
   * Matplotlib- creating plots of data
   * Pandas- facilitates structured / tabular data manipulation and visualisations.
   * Pomegranate- for fast and flexible probabilistic models.
3. An Integrated Development Environment (IDE) for Python Programming.

# Anaconda-

Together with a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages.

# Program 1.Implement and demonstrate the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data froma.CSV file.

**Task: Find-S algorithm is used to find a maximally specific hypothesis.**

Positive and negative training examples for the target concept EnjoySport.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

Given:**Instances X**: Possible days, each described by the attributes

**Sky** (with possible values Sunny, Cloudy, and Rainy),

**AirTemp** (with values Warm and Cold), **Humidity** (with values Normal and High), **Wind** (with values Strong and Weak), **Water** (with values Warm and Cool), and **Forecast** (with values Same and Change).

**Hypotheses H**: Each hypothesis is described by a conjunction of constraints on the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast. The constraints may be "?" (any value is acceptable), "0 (no value is acceptable), or a specific value.

**Target concept c**: EnjoySport : X (0,l)

Training examples D: Positive and negative examples of the target function

**Code:-**

import csv

def loadCsv(filename):

lines = csv.reader(open(filename, "rt"))

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = dataset[i]

return dataset

attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']

print(attributes)

num\_attributes = len(attributes)

filename = "Weather.csv"

dataset = loadCsv(filename)

print(dataset)

target=['Yes','Yes','No','Yes']

print(target)

hypothesis=['0'] \* num\_attributes

print(hypothesis)

print("The Hypothesis are")

for i in range(len(target)):

if(target[i] == 'Yes'):

for j in range(num\_attributes):

if(hypothesis[j]=='0'):

hypothesis[j] = dataset[i][j]

if(hypothesis[j]!= dataset[i][j]):

hypothesis[j]='?'

print(i+1,'=',hypothesis)

print("Final Hypothesis")

print(hypothesis)

**Input:**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

**OUPUT:**

['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']

[['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']]

Intial Hypothesis

['0', '0', '0', '0', '0', '0']

The Hypothesis are

1 = ['Sunny ', 'Warm', 'Normal', 'Strong ', 'Warm', 'Same']

2 = ['Sunny ', 'Warm', '?', 'Strong ', 'Warm', 'Same']

3 = ['Sunny ', 'Warm', '?', 'Strong ', 'Warm', 'Same']

4 = ['Sunny ', 'Warm', '?', 'Strong ', '?', '?']

Final Hypothesis

['Sunny ', 'Warm', '?', 'Strong ', '?', '?']

**Program 2- For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples**

**Task**: The CANDIDATE-ELIMINATION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

**Dataset**: EnjoySport training examples

**Code:-**

import numpy as np

import pandas as pd

# Loading Data from a CSV File

data = pd.DataFrame(data=pd.read\_csv('Training\_examples.csv'))

# Separating concept features from Target

concepts = np.array(data.iloc[:,0:-1])

# Isolating target into a separate DataFrame

#copying last column to target array

target = np.array(data.iloc[:,-1])

def learn(concepts, target):

'''

learn() function implements the learning method of the Candidate elimination algorithm.

Arguments:

concepts - a data frame with all the features

target - a data frame with corresponding output values

'''

# Initialise S0 with the first instance from concepts

# .copy() makes sure a new list is created instead of just pointing to the same memory location

specific\_h = concepts[0].copy()

print("initialization of specific\_h and general\_h")

print(specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print(general\_h)

# The learning iterations

for i, h in enumerate(concepts):

# Checking if the hypothesis has a positive target

if target[i] == "Yes":

for x in range(len(specific\_h)):

# Change values in S & G only if values change

if h[x] != specific\_h[x]:

specific\_h[x] = '?'

general\_h[x][x] = '?'

# Checking if the hypothesis has a positive target

if target[i] == "No":

for x in range(len(specific\_h)):

# For negative hyposthesis change values only in G

if h[x] != specific\_h[x]:

general\_h[x][x] = specific\_h[x]

else:

general\_h[x][x] = '?'

print(" steps of Candidate Elimination Algorithm",i+1)

print(specific\_h)

print(general\_h)

# find indices where we have empty rows, meaning those that are unchanged

indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

# remove those rows from general\_h

general\_h.remove(['?', '?', '?', '?', '?', '?'])

# Return final values

return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

**Input:**

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:**

initialization of specific\_h and general\_h

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 1

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of Candidate Elimination Algorithm 2

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

steps of Candidate Elimination Algorithm 3

['Sunny' 'Warm' 'High' 'Strong' '?' '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

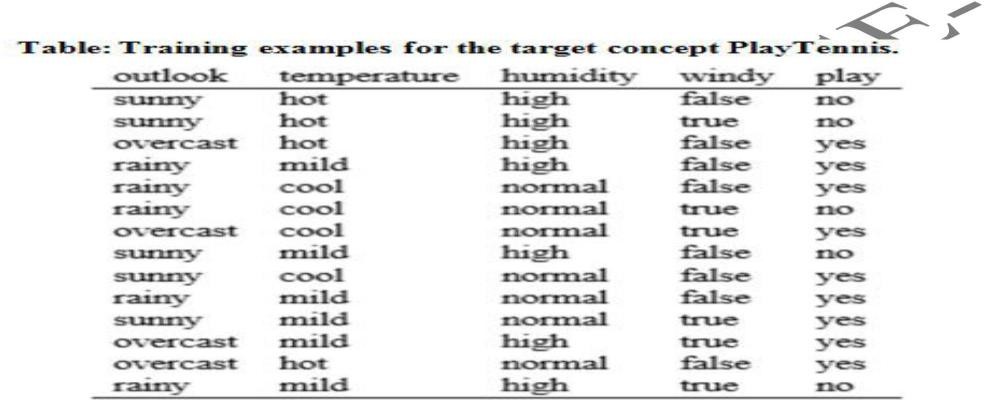
['Sunny' 'Warm' 'High' 'Strong' '?' '?']

Final General\_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

# Program 3.Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

**Task**: ID3determinesthe information gain for each candidate attribute (i.e.,Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain as the root node of the tree. The information gain values for all four attributes are calculated using the following formula:

Entropy(S)=∑- P(I).log2P(I)

Gain(S,A)=Entropy(S)-∑ [ P(S/A).Entropy(S/A)] Dataset-tennis.csv

# Sunny outlook on decision:

5 instances of sunny : In that 3 instances are NO and 2 instances are YES Gain( Outlook = Sunny/Temp)= 0.570 Gain ( Outlook Sunny/Humidity) =0.970

Gain ( Outlook = Sunny/ Wind) = 0.019

Since humidity produces the highest score, if outlook were Sunny.

# Overcast outlook on decision:

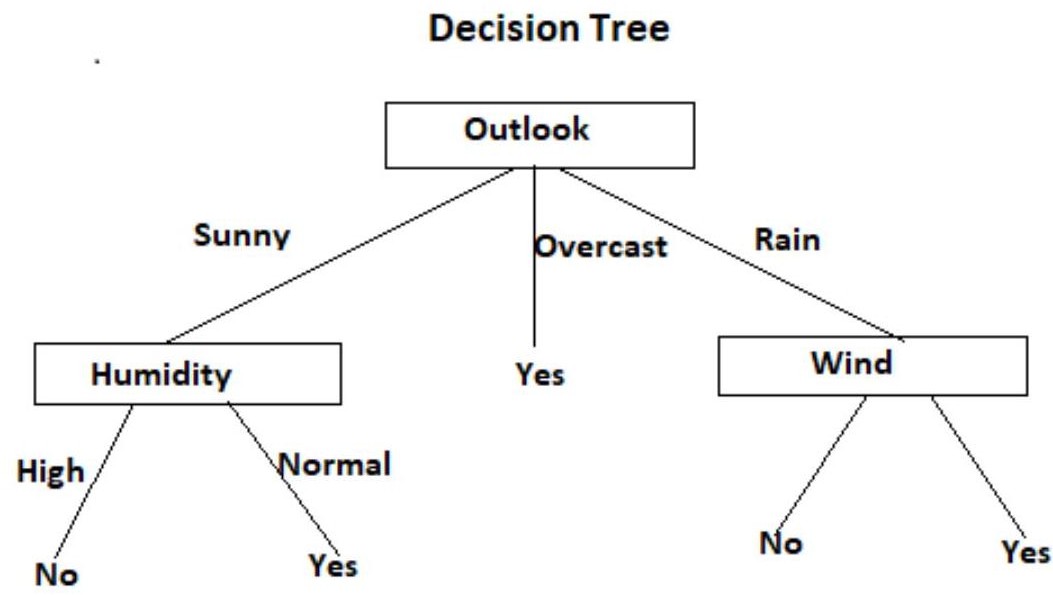
Decision will always be yes, if outlook were overcast.

# Rain outlook on decision:

5 instances of rain :In that 3 instances are YES and 2 instances are NO. Gain( Outlook= Rain/Temp) Gain( Outlook=Rain/Humidity)

Gain( Outlook= Rain/Wind)

Here, wind produces the highest score . And wind has two attributes namely strong and weak.



**Code:-**

import pandas as pd

import numpy as np

#Import the dataset and define the feature as well as the target datasets / columns#

dataset = pd.read\_csv('playtennis.csv',

names=['outlook','temperature','humidity','wind','class',])

#Import all columns omitting the fist which consists the names of the animals

#We drop the animal names since this is not a good feature to split the data on

attributes =('Outlook','Temperature','Humidity','Wind','PlayTennis')

def entropy(target\_col):

"""

Calculate the entropy of a dataset.

The only parameter of this function is the target\_col parameter which specifies the target column

"""

elements,counts = np.unique(target\_col,return\_counts = True)

entropy = np.sum([(-counts[i]/np.sum(counts))\*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])

#print('Entropy =', entropy)

return entropy

def InfoGain(data,split\_attribute\_name,target\_name="class"):

#Calculate the entropy of the total dataset

total\_entropy = entropy(data[target\_name])

##Calculate the entropy of the dataset

#Calculate the values and the corresponding counts for the split attribute

vals,counts= np.unique(data[split\_attribute\_name],return\_counts=True)

#Calculate the weighted entropy

Weighted\_Entropy = np.sum([(counts[i]/np.sum(counts))\*entropy(data.where(data[split\_attribute\_name]==vals[i]).dropna()[target\_name]) for i in range(len(vals))])

#Calculate the information gain

Information\_Gain = total\_entropy - Weighted\_Entropy

return Information\_Gain

def ID3(data,originaldata,features,target\_attribute\_name="class",parent\_node\_class = None): #Define the stopping criteria --> If one of this is satisfied, we want to return a leaf node#

#If all target\_values have the same value, return this value

if len(np.unique(data[target\_attribute\_name])) <= 1:

return np.unique(data[target\_attribute\_name])[0]

#If the dataset is empty, return the mode target feature value in the original dataset

elif len(data)==0:

return np.unique(originaldata[target\_attribute\_name])[np.argmax(np.unique(originaldata[target\_attribute\_name],return\_counts=True)[1])]

elif len(features) ==0:

return parent\_node\_class

#If none of the above holds true, grow the tree!

else:

#Set the default value for this node --> The mode target feature value of the current node

parent\_node\_class = np.unique(data[target\_attribute\_name])[np.argmax(np.unique(data[target\_attribute\_name],return\_counts=True)[1])]

#Select the feature which best splits the dataset

item\_values = [InfoGain(data,feature,target\_attribute\_name) for feature in features] #Return the information gain values for the features in the dataset

best\_feature\_index = np.argmax(item\_values)

best\_feature = features[best\_feature\_index]

#Create the tree structure. The root gets the name of the feature (best\_feature) with the maximum information

#gain in the first run

tree = {best\_feature:{}}

#Remove the feature with the best inforamtion gain from the feature space

features = [i for i in features if i != best\_feature]

#Grow a branch under the root node for each possible value of the root node feature

for value in np.unique(data[best\_feature]):

value = value

#Split the dataset along the value of the feature with the largest information gain and therwith create sub\_datasets

sub\_data = data.where(data[best\_feature] == value).dropna()

#Call the ID3 algorithm for each of those sub\_datasets with the new parameters --> Here the recursion comes in!

subtree = ID3(sub\_data,dataset,features,target\_attribute\_name,parent\_node\_class)

#Add the sub tree, grown from the sub\_dataset to the tree under the root node

tree[best\_feature][value] = subtree

return(tree)

def predict(query,tree,default = 1):

#1.

for key in list(query.keys()):

if key in list(tree.keys()):

#2.

try:

result = tree[key][query[key]]

except:

return default

#3.

result = tree[key][query[key]]

#4.

if isinstance(result,dict):

return predict(query,result)

else:

return result

def train\_test\_split(dataset):

training\_data = dataset.iloc[:14].reset\_index(drop=True)

#We drop the index respectively relabel the index

#starting form 0, because we do not want to run into errors regarding the row labels / indexes

#testing\_data = dataset.iloc[10:].reset\_index(drop=True)

return training\_data #,testing\_data

def test(data,tree):

#Create new query instances by simply removing the target feature column from the original dataset and

#convert it to a dictionary

queries = data.iloc[:,:-1].to\_dict(orient = "records")

#Create a empty DataFrame in whose columns the prediction of the tree are stored

predicted = pd.DataFrame(columns=["predicted"])

#Calculate the prediction accuracy

for i in range(len(data)):

predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)

print('The prediction accuracy is: ',(np.sum(predicted["predicted"] == data["class"])/len(data))\*100,'%')

"""

Train the tree, Print the tree and predict the accuracy

"""

XX = train\_test\_split(dataset)

training\_data=XX

#testing\_data=XX[1]

tree = ID3(training\_data,training\_data,training\_data.columns[:-1])

print(' Display Tree',tree)

print('len=',len(training\_data))

test(training\_data,tree)

**Input: Playtennis dataset**

Outlook,Temperature,Humidity,Windy,PlayTennis

Sunny,Hot,High,False,No

Sunny,Hot,High,True,No

Overcast,Hot,High,False,Yes

Rainy,Mild,High,False,Yes

Rainy,Cool,Normal,False,Yes

Rainy,Cool,Normal,True,No

Overcast,Cool,Normal,True,Yes

Sunny,Mild,High,False,No

Sunny,Cool,Normal,False,Yes

Rainy,Mild,Normal,False,Yes

Sunny,Mild,Normal,True,Yes

Overcast,Mild,High,True,Yes

Overcast,Hot,Normal,False,Yes

Rainy,Mild,High,True,No

**Output:**

Display Tree {'outlook': {'Outlook': 'PlayTennis', 'Overcast': 'Yes', 'Rainy': {'wind': {'False': 'Yes', 'True': 'No'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}

len= 14

The prediction accuracy is: 100.0 %

# Program 4-Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate dataset

Back propagation Algorithm

The Back propagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks, the back propagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. Back propagation can be used for both classification and regression problems.

Working of algorithm:

* 1. Initialize Network.
  2. Forward Propagate.
  3. Back Propagate Error.
  4. Train the network

**Code:-**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=7000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer #weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons)) #draws a random range of numbers uniformly of dim x\*y for i inrange(epoch):

#Forward Propogation hinp1=np.dot(X,wh)

hinp=hinp1 + bh hlayer\_act =sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp)

#Backpropagation EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr# dotproduct of nextlayererror and currentlayerop

# bout += np.sum(d\_output, axis=0,keepdims=True) \*lr wh += X.T.dot(d\_hiddenlayer)\*lr

#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr print("Input: \n" + str(X))

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

**Output-**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.75415399]

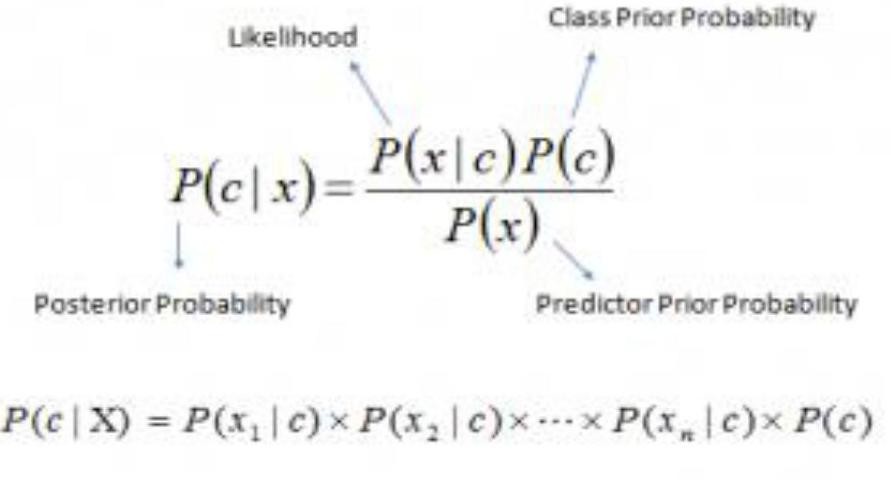
[0.74085499]

[0.7545631 ]]

# Program 5.Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

# Task:It is a classification technique based on [Bayes‟ Theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as „Naive‟.

Data Set : PlayTennis example



**Code:-**

# Naive Bayes impementation in Python

import csv

import random

import math

#1.Load Data

def loadCsv(filename):

lines = csv.reader(open(filename, "rt"))

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [float(x) for x in dataset[i]]

return dataset

#Split the data into Training and Testing randomly

def splitDataset(dataset, splitRatio):

trainSize = int(len(dataset) \* splitRatio)

trainSet = []

copy = list(dataset)

while len(trainSet) < trainSize:

index = random.randrange(len(copy))

trainSet.append(copy.pop(index))

return [trainSet, copy]

#Seperatedata by Class

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

#Calculate Mean

def mean(numbers):

return sum(numbers)/float(len(numbers))

#Calculate Standard Deviation

def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)

#Summarize the data

def summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]

del summaries[-1]

return summaries

#Summarize Attributes by Class

def summarizeByClass(dataset):

separated = separateByClass(dataset)

print(len(separated))

summaries = {}

for classValue, instances in separated.items():

summaries[classValue] = summarize(instances)

print(summaries)

return summaries

#Calculate Gaussian Probability Density Function

def calculateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

#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

#Make a Prediction

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

#return a list of predictions for each test instance.

def getPredictions(summaries, testSet):

predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i])

predictions.append(result)

return predictions

#calculate accuracy ratio.

def getAccuracy(testSet, predictions):

correct = 0

for i in range(len(testSet)):

if testSet[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

filename = 'DBetes.csv'

splitRatio = 0.70

dataset = loadCsv(filename)

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))

# prepare model

summaries = summarizeByClass(trainingSet)

# test model

predictions = getPredictions(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)

print('Accuracy: {0}%'.format(accuracy))

**Input:** DBetes.csv—250 lines

**Output:**

Split 250 rows into train=175 and test=75 rows

2

{0.0: [(3.311926605504587, 2.977375328442019), (110.40366972477064, 27.37331738987706), (68.34862385321101, 15.89571113061458), (19.94495412844037, 14.761583563541949), (68.54128440366972, 125.06631811948658)], 1.0: [(5.121212121212121, 3.9594683074106394), (137.16666666666666, 30.429531877050135), (72.57575757575758, 16.471944078491212), (21.71212121212121, 17.22320898494029), (85.66666666666667, 124.22006939712128)]}

Accuracy: 78.66666666666666%

# 6.Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your dataset.

**Code:-**

#1 Loading the data set

from sklearn.datasets import fetch\_20newsgroups

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True)

print("lenth of the twenty\_train--------->", len(twenty\_train))

#print(twenty\_train.target\_names) #prints all the categories

print("\*\*\*First Line of the First Data File\*\*\*")

#print("\n".join(twenty\_train.data[0].split("\n")[:5]))#prints first line of the first data file

#2 Extracting features from text files

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

print('dim=',X\_train\_counts.shape)

#3 TF-IDF

from sklearn.feature\_extraction.text import TfidfTransformer

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

print(X\_train\_tfidf.shape)

# Machine Learning

#4 Training Naive Bayes (NB) classifier on training data.

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target)

# Building a pipeline: We can write less code and do all of the above, by building a pipeline as follows:

# The names ‘vect’ , ‘tfidf’ and ‘clf’ are arbitrary but will be used later.

# We will be using the 'text\_clf' going forward.

from sklearn.pipeline import Pipeline

text\_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])

text\_clf = text\_clf.fit(twenty\_train.data, twenty\_train.target)

# Performance of NB Classifier

import numpy as np

twenty\_test = fetch\_20newsgroups(subset='test', shuffle=True)

predicted = text\_clf.predict(twenty\_test.data)

accuracy=np.mean(predicted == twenty\_test.target)

print("Predicted Accuracy = ",accuracy)

#To Calculate Accuracy,Precision,Recall

from sklearn import metrics

print("Accuracy= ",metrics.accuracy\_score(twenty\_test.target,predicted))

print("Precision=",metrics.precision\_score(twenty\_test.target,predicted,average=None))

print("Recall=",metrics.recall\_score(twenty\_test.target,predicted,average=None))

print(metrics.classification\_report(twenty\_test.target, predicted,target\_names=twenty\_test.target\_names))

# Output:-

# lenth of the twenty\_train---------> 6

# \*\*\*First Line of the First Data File\*\*\*

# dim= (11314, 130107)

# (11314, 130107)

# Predicted Accuracy = 0.7738980350504514

# Accuracy= 0.7738980350504514

# Precision= [0.80193237 0.81028939 0.81904762 0.67180617 0.85632184 0.88955224

# 0.93127148 0.84651163 0.93686869 0.92248062 0.89170507 0.59379845

# 0.83629893 0.92113565 0.84172662 0.43896976 0.64339623 0.92972973

# 0.95555556 0.97222222]

# Recall= [0.52037618 0.64781491 0.65482234 0.77806122 0.77402597 0.75443038

# 0.69487179 0.91919192 0.9321608 0.89924433 0.96992481 0.96717172

# 0.59796438 0.73737374 0.89086294 0.98492462 0.93681319 0.91489362

# 0.41612903 0.13944223]

# precision recall f1-score support

# alt.atheism 0.80 0.52 0.63 319

# comp.graphics 0.81 0.65 0.72 389

# comp.os.ms-windows.misc 0.82 0.65 0.73 394

# comp.sys.ibm.pc.hardware 0.67 0.78 0.72 392

# comp.sys.mac.hardware 0.86 0.77 0.81 385

# comp.windows.x 0.89 0.75 0.82 395

# misc.forsale 0.93 0.69 0.80 390

# rec.autos 0.85 0.92 0.88 396

# rec.motorcycles 0.94 0.93 0.93 398

# rec.sport.baseball 0.92 0.90 0.91 397

# rec.sport.hockey 0.89 0.97 0.93 399

# sci.crypt 0.59 0.97 0.74 396

# sci.electronics 0.84 0.60 0.70 393

# sci.med 0.92 0.74 0.82 396

# sci.space 0.84 0.89 0.87 394

# soc.religion.christian 0.44 0.98 0.61 398

# talk.politics.guns 0.64 0.94 0.76 364

# talk.politics.mideast 0.93 0.91 0.92 376

# talk.politics.misc 0.96 0.42 0.58 310

# talk.religion.misc 0.97 0.14 0.24 251

# avg / total 0.82 0.77 0.77 7532

**Program 7.Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML libraryclasses/API.**

A Bayesian belief network describes the probability distribution over a set of variables. Probability

P(A) is used to denote the probability of A. For example if A is discrete with states {True, False} then P(A) might equal [0.2, 0.8]. I.e. 20% chance of being True, 80% chance of being False.

Joint probability

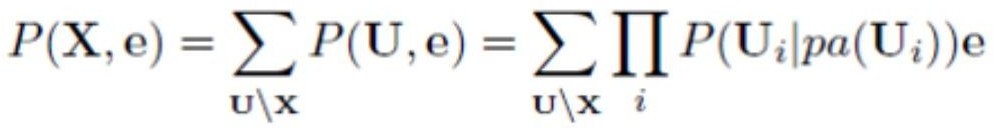
A joint probability refers to the probability of more than one variable occurring together, such as the probability of A and B, denoted P(A,B).

Conditional probability

Conditional probability is the probability of a variable (or set of variables) given another variable (or set of variables), denoted P(A|B).For example, the probability of Windy being True, given that Raining is True might equal 50%.This would be denoted P(Windy = True | Raining = True) =50%.

Once the structure has been defined (i.e. nodes and links), a Bayesian network requires a probability distribution to be assigned to each node. Each node X in a Bayesian network requires a probability distribution P(X | pa(X)).Note that if a node X has no parents pa(X) is empty, and the required distribution is just P(X) sometimes referred to as the prior. This is the probability of itself given its parent nodes.

If U = {A1,...,An} is the universe of variables (all the variables) in a Bayesian network, and pa(Ai) are the parents of Ai then the joint probability distribution P(U) is the simply the product of all the probability distributions (prior and conditional) in the network, as shown in the equation below. This equation is known as the chain rule.



**Code:-**

import bayespy as bp

import numpy as np

import csv

from colorama import init

from colorama import Fore, Back, Style

init()

# Define Parameter Enum values

#Age

ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4}

# Gender

genderEnum = {'Male':0, 'Female':1}

# FamilyHistory

familyHistoryEnum = {'Yes':0, 'No':1}

# Diet(Calorie Intake)

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

# LifeStyle

lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}

# Cholesterol

cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}

# HeartDisease

heartDiseaseEnum = {'Yes':0, 'No':1}

#heart\_disease\_data.csv

with open('heart\_disease\_data.csv') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

data = []

for x in dataset:

data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])

# Training data for machine learning todo: should import from csv

data = np.array(data)

N = len(data)

# Input data column assignment

p\_age = bp.nodes.Dirichlet(1.0\*np.ones(5))

age = bp.nodes.Categorical(p\_age, plates=(N,))

age.observe(data[:,0])

p\_gender = bp.nodes.Dirichlet(1.0\*np.ones(2))

gender = bp.nodes.Categorical(p\_gender, plates=(N,))

gender.observe(data[:,1])

p\_familyhistory = bp.nodes.Dirichlet(1.0\*np.ones(2))

familyhistory = bp.nodes.Categorical(p\_familyhistory, plates=(N,))

familyhistory.observe(data[:,2])

p\_diet = bp.nodes.Dirichlet(1.0\*np.ones(3))

diet = bp.nodes.Categorical(p\_diet, plates=(N,))

diet.observe(data[:,3])

p\_lifestyle = bp.nodes.Dirichlet(1.0\*np.ones(4))

lifestyle = bp.nodes.Categorical(p\_lifestyle, plates=(N,))

lifestyle.observe(data[:,4])

p\_cholesterol = bp.nodes.Dirichlet(1.0\*np.ones(3))

cholesterol = bp.nodes.Categorical(p\_cholesterol, plates=(N,))

cholesterol.observe(data[:,5])

# Prepare nodes and establish edges

# np.ones(2) -> HeartDisease has 2 options Yes/No

# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values

p\_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))

heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p\_heartdisease)

heartdisease.observe(data[:,6])

p\_heartdisease.update()

# Sample Test with hardcoded values

#print("Sample Probability")

#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")

#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'], cholesterolEnum['High']], bp.nodes.Categorical, p\_heartdisease).get\_moments()[0][heartDiseaseEnum['Yes']])

# Interactive Test

m = 0

while m == 0:

print("\n")

res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, p\_heartdisease).get\_moments()[0][heartDiseaseEnum['Yes']]

print("Probability(HeartDisease) = " + str(res))

#print(Style.RESET\_ALL)

m = int(input("Enter for Continue:0, Exit :1 "))

**Input:**

SuperSeniorCitizen,Male,Yes,Medium,Sedetary,High,Yes

SuperSeniorCitizen,Female,Yes,Medium,Sedetary,High,Yes

SeniorCitizen,Male,No,High,Moderate,BorderLine,Yes

Teen,Male,Yes,Medium,Sedetary,Normal,No

Youth,Female,Yes,High,Athlete,Normal,No

MiddleAged,Male,Yes,Medium,Active,High,Yes

Teen,Male,Yes,High,Moderate,High,Yes

SuperSeniorCitizen,Male,Yes,Medium,Sedetary,High,Yes

Youth,Female,Yes,High,Athlete,Normal,No

SeniorCitizen,Female,No,High,Athlete,Normal,Yes

Teen,Female,No,Medium,Moderate,High,Yes

Teen,Male,Yes,Medium,Sedetary,Normal,No

MiddleAged,Female,No,High,Athlete,High,No

MiddleAged,Male,Yes,Medium,Active,High,Yes

Youth,Female,Yes,High,Athlete,BorderLine,No

SuperSeniorCitizen,Male,Yes,High,Athlete,Normal,Yes

SeniorCitizen,Female,No,Medium,Moderate,BorderLine,Yes

Youth,Female,Yes,Medium,Athlete,BorderLine,No

Teen,Male,Yes,Medium,Sedetary,Normal,No

**Output:**

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

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

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

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

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

Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2

Probability(HeartDisease) = 0.13784165696493575

Enter for Continue:0, Exit :1

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

# Expectation-Maximization (EM)

The EM algorithm tends to get stuck less than K-means algorithm. The idea is to assign data points partially to different clusters instead of assigning to only one cluster. To do this partial assignment, we model each cluster using a probabilistic distribution So a data point associates with a cluster with certain probability and it belongs to the cluster with the highest probability in the final assignment.

# Expectation-Maximization (EM) algorithm

Step 1: An initial guess is made for the model’s parameters and a probability distribution is created. This is sometimes called the “E-Step” for the “Expected” distribution.

Step 2: Newly observed data is fed into the model.

Step 3: The probability distribution from the E-step is drawn to include the new data. This is sometimes called the “M-step.”

Step 4: Steps 2 through 4 are repeated until stability.

**Code:-**

import numpy as np

from scipy import stats

np.random.seed(110) # for reproducible random results

# set parameters

red\_mean = 3

red\_std = 0.8

blue\_mean = 7

blue\_std = 1

# draw 40 samples from normal distributions with red/blue parameters

red = np.random.normal(red\_mean, red\_std, size=40)

blue = np.random.normal(blue\_mean, blue\_std, size=40)

both\_colours = np.sort(np.concatenate((red, blue)))

#Since the colours are hidden from us, we will start the EM process

#Starting guesses are very critical because the EM Algorithm converges to

# a local maxima. Hence we can get different answers with different starting points

#One reasonably good guess would be to take the value from a different but less

#robust algorithm

# estimates for the mean

red\_mean\_guess = 2.1

blue\_mean\_guess = 6

# estimates for the standard deviation

red\_std\_guess = 1.5

blue\_std\_guess = 0.8

#These are pretty bad guesses

#To continue with EM and improve these guesses, we compute the likelihood

#of each data point (regardless of its secret colour) appearing under

#these guesses for the mean and standard deviation

#The variable both\_colours holds each data point. The function stats.norm computes

#the probability of the point under a normal distribution with the given parameters:

for i in range(10):

likelihood\_of\_red = stats.norm(red\_mean\_guess, red\_std\_guess).pdf(both\_colours)

likelihood\_of\_blue = stats.norm(blue\_mean\_guess, blue\_std\_guess).pdf(both\_colours)

#Normalize these weights so that they can total 1

likelihood\_total = likelihood\_of\_red + likelihood\_of\_blue

red\_weight = likelihood\_of\_red / likelihood\_total

blue\_weight = likelihood\_of\_blue / likelihood\_total

#With our current estimates and our newly-computed weights, we can now compute new,

#probably better, estimates for the parameters (step 4). We need a function for the

#mean and a function for the standard deviation:

def estimate\_mean(data, weight):

return np.sum(data \* weight) / np.sum(weight)

def estimate\_std(data, weight, mean):

variance = np.sum(weight \* (data - mean)\*\*2) / np.sum(weight)

return np.sqrt(variance)

# new estimates for standard deviation

blue\_std\_guess = estimate\_std(both\_colours, blue\_weight, blue\_mean\_guess)

red\_std\_guess = estimate\_std(both\_colours, red\_weight, red\_mean\_guess)

# new estimates for mean

red\_mean\_guess = estimate\_mean(both\_colours, red\_weight)

blue\_mean\_guess = estimate\_mean(both\_colours, blue\_weight)

#Lets print the model parameters (The means and the std deviation in our case)

print("red mean:", red\_mean\_guess, ":::::::::", "blue mean:", blue\_mean\_guess)

print("red std:", red\_std\_guess, ":::::::::", "blue std:", blue\_std\_guess)

#plot the data

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.mlab as mlab

#The two Gaussian distributions

y = np.zeros(len(both\_colours))

mured = red\_mean\_guess

sigmared = red\_std\_guess

x = np.linspace(mured - 2.5\*sigmared, mured + 2.5\*sigmared, 100)

plt.plot(x,mlab.normpdf(x, mured, sigmared))

mublue = blue\_mean\_guess

sigmablue = blue\_std\_guess

y = np.linspace(mublue - 2.5\*sigmablue, mublue + 2.5\*sigmablue, 100)

plt.plot(y,mlab.normpdf(y, mublue, sigmablue))

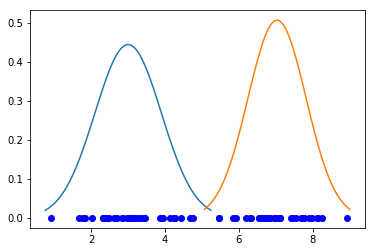
#The data points themselves

for i in range(len(both\_colours)):

plt.plot(both\_colours[i],0,"bo")

plt.show()

**Output:-**



# K-means

import pylab as pl

import numpy as np

from sklearn.cluster import KMeans

np.random.seed(110) # for reproducible random results

# set parameters

red\_mean = 3

red\_std = 0.8

blue\_mean = 7

blue\_std = 1

# draw 20 samples from normal distributions with red/blue parameters

red = np.random.normal(red\_mean, red\_std, size=40)

blue = np.random.normal(blue\_mean, blue\_std, size=40)

both\_colours = np.sort(np.concatenate((red, blue)))

y = np.zeros(len(both\_colours))

#We will need the elbow curve for calculating exact value of k

#But we will use 2 for now

kmeans=KMeans(n\_clusters=2)

kmeansoutput=kmeans.fit(both\_colours.reshape(-1,1))

#but what value of K was actually good?

Nc = range(1, 5)

kmeans = [KMeans(n\_clusters=i) for i in Nc]

score = [kmeans[i].fit(both\_colours.reshape(-1,1)).score(both\_colours.reshape(-1,1)) for i in range(len(kmeans))]

pl.plot(Nc,score)

pl.xlabel('Number of Clusters')

pl.ylabel('Score')

pl.title('Elbow Curve')

pl.show()

#plot the points themselves

pl.scatter(both\_colours,y,c=kmeansoutput.labels\_)

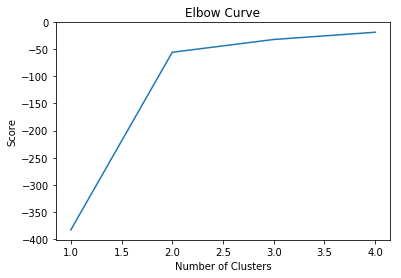
pl.xlabel('Data points')

pl.ylabel('None')

pl.title('2 Cluster K-Means')

pl.show()

**Output:-**



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

**ALGORITHM:**

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

Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

for i=0 to m:

Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.

Return the majority label among S.

**Code:-**

from sklearn.datasets import load\_iris

iris = load\_iris()

print("Feature Names:",iris.feature\_names,"Iris Data:",iris.data,"Target Names:",iris.target\_names,"Target:",iris.target)

#2. Split the data into Test and Data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

iris.data, iris.target, test\_size = .25)

#neighbors\_settings = range(1, 11)

#for n\_neighbors in neighbors\_settings:

#3.Build The Model

from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()

clf.fit(X\_train, y\_train)

#4.Calculate Accuracy of the Test data with the trained data

print(" Accuracy=",clf.score(X\_test, y\_test))

#5 Calculate the prediction with the labels of the test data

print("Predicted Data")r

print(clf.predict(X\_test))

prediction=clf.predict(X\_test)

print("Test data :")

print(y\_test)

#6 To identify the miss classification

diff=prediction-y\_test

print("Result is ")

print(diff)

print('Total no of samples misclassied =', sum(abs(diff)))

**Output:**

Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'] Iris Data: [[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.1 1.5 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1. ]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1. ]

[5.9 3. 4.2 1.5]

[6. 2.2 4. 1. ]

[6.1 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1. ]

[6.2 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

[6.3 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

[6.4 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3.5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1. ]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1. ]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

[6. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

[5. 2.3 3.3 1. ]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2. ]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5.2. ]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2. ]

[7.7 2.8 6.7 2. ]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2. ]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2. ]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]] Target Names: ['setosa' 'versicolor' 'virginica'] Target: [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 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 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]

Accuracy= 0.9736842105263158

Predicted Data

[0 1 2 1 2 0 1 2 2 0 2 0 2 2 2 0 2 1 2 1 1 0 2 2 2 0 1 2 0 0 2 1 1 0 1 0 0

0]

Test data :

[0 1 2 1 2 0 1 2 2 0 2 0 2 2 2 0 2 1 2 1 1 0 2 2 1 0 1 2 0 0 2 1 1 0 1 0 0

0]

Result is

[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 0 0 0 0 0 0 0 0 0 0 0 0

0]

Total no of samples misclassied = 1

**Program 10.Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and drawgraphs.**

**Nonparametric regression:** is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data. Nonparametric regression requires larger sample sizes than regression based on parametric models because the data must supply the model structure as well as the model estimates.

Nonparametric regression is used for prediction and is reliable even if hypotheses of linear regression are not verified.

**Locally weighted Learning** also known as memory-based learning, instance-based learning, lazy- learning, and closely related to kernel density estimation, similarity searching and case-based reasoning.

LOWESS (Locally Weighted Scatterplot Smoothing), sometimes called LOESS (locally weighted smoothing), is a popular tool used in [regression analysis](http://www.statisticshowto.com/probability-and-statistics/regression-analysis/)that creates a smooth line through a [timeplot](http://www.statisticshowto.com/timeplot/)or [scatter plot](http://www.statisticshowto.com/probability-and-statistics/regression-analysis/scatter-plot-chart/#definition)to help you to see relationship between [variables](http://www.statisticshowto.com/variable/)and foresee trends.

[Locally weighted regression](https://en.wikipedia.org/wiki/Local_regression) is a very powerful non-parametric model used in statistical learning.

**Locally weighted regression**

**Local** means using nearby points (i.e. a nearest neighbors approach) **Weighted** means we value points based upon how far away they are. **Regression** means approximating afunction

This is **an instance-based learningmethod**

**Code:-**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#the Gaussian Kernel

def kernel(point,xmat, k):

m,n = np.shape(xmat)

weights = np.mat(np.eye((m)))

for j in range(m):

diff = point - X[j]

weights[j,j] = np.exp(diff\*diff.T/(-2.0\*k\*\*2))

return weights

#Weigh each point by its distance to the reference point. We are considering

# All points here. If KNN was the topic, we could restrict this to "K"

def localWeight(point,xmat,ymat,k):

wei = kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T))

return W

def localWeightRegression(xmat,ymat,k):

m,n = np.shape(xmat)

ypred = np.zeros(m)

for i in range(m):

# predicted value y = wx. Here w = weights we have computed.

# Remember that both w and x are vectors here (2\*1 and 1\*2 respectively)

# Resultant value of y is a scalar

ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k)

return ypred

# load data points

data = pd.read\_csv('LR.csv')

colA = np.array(data.colA)

colB = np.array(data.colB)

#preparing and add 1

#convert to matrix form

mcolA = np.mat(colA)

mcolB = np.mat(colB)

m= np.shape(mcolA)[1]

one = np.ones((1,m),dtype=int)

#horizontally stack

X= np.hstack((one.T,mcolA.T))

print(X.shape)

#set k here (0.5)

ypred = localWeightRegression(X,mcolB,0.5)

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

xsort = X[SortIndex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1)

ax.scatter(colA,colB, color='green')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('colA')

plt.ylabel('colB')

plt.show();

**Input:**

colA,colB

3,4

2.883,5

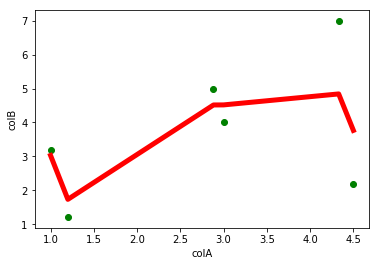
4.33,7

4.5,2.2

1.2,1.2

1,3.2

**Output:**



**Viva Questions and Answers-**

1. **What is machinelearning?**

Machine learning is a subset of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) in the field of [computer](https://en.wikipedia.org/wiki/Computer_science) [science](https://en.wikipedia.org/wiki/Computer_science) that often uses statistical techniques to give [computers](https://en.wikipedia.org/wiki/Computer) the ability to "learn" (i.e., progressively improve performance on a specific task) with [data,](https://en.wikipedia.org/wiki/Data) without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

1. **Define supervised learning**

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.

1. **Define unsupervised learning**

No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

1. **Define semi supervised learning**

The computer is given only an incomplete training signal: a training set with some (often many) of the target outputsmissing.

1. **Define reinforcementlearning**

Training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

1. **What do you mean byhypotheses**

In a machine learning problem where the input is denoted by x and the output is y.In order to do machine learning, there should exist a relationship (pattern) between the input and output values. Lets say that this is the function y=f(x) this known as the target function. However, f(.) is unknown function to us, so machine learning algorithms try to guess a ``hypothesis'' function h(x) that approximates the unknown f(.) the set of all possible hypotheses is known as the Hypothesis set H(.), the goal is the learning process is to find the final hypothesis that best approximates the unknown targetfunction.

1. **What is classification?**

In [classification,](https://en.wikipedia.org/wiki/Statistical_classification) inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more [(multi-label classification)](https://en.wikipedia.org/wiki/Multi-label_classification) of these classes. This is typically tackled in a supervised manner. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".

1. **What is clustering**

Clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis used in many fields.

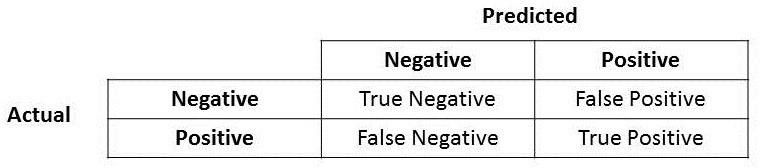
1. **Define precision, accuracy andrecall**

**Precision** attempts to answer the following question: What proportion of positive identifications was actually correct?

Precision= True Positive / (True Positive + False Positive).

**Recall** attempts to answer the following question: What proportion of actual positives was identified correctly?

Recall= True Positive / (True Positive + False Negative).



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

TP- True positive, TN- True negative, FP- False positive, FN- False negative.

1. **How KNN is different from k-meansclustering**

|  |  |
| --- | --- |
| KNN | K-Means |
| It is a **Supervised** learning  Technique | It is an **Unsupervised** learning  technique |
| It is used mostly for Classification, andsometimes  even for Regression | It is used for **Clustering** |
| „K‟ in KNN is the number of nearest neighbors used to classify or (predict in case of continuous variable/regression) a test sample | K‟ in K-Means is the number of clusters the algorithm is trying to identify/learn from the data. The clusters are often unknown since this is used with Unsupervised  learning. |
| K-NN doesn‟t haveatraining phase as such. But the prediction of a test observation is done based on the K-Nearest (often Euclidean distance) Neighbors (observations) based onweighted  averages/votes. | In training phase of K-Means, K observations are arbitrarily selected (known as centroids). Each point in the vector space is assigned to a cluster represented by nearest (Euclidean distance)  centroid. |

1. **Defineregression**

Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables(x).

1. **What is conceptlearning**

Concept learning also refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner simplifies what has been observed by condensing it in the form of an example.

1. **Define decisiontree**

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining and machine learning.

1. **What isANN**

Artificial Neural networks (ANN) or neural networks are computational algorithms. Itintended to simulate the behavior of biological systems composed of “neurons”. ANNs

are computational models inspired by an animal's central nervous systems. It is capable of machine learning as well as patternrecognition.

1. **Define Bayesian beliefnetworks**

A Bayesian network, Bayes network, belief network, Bayes model or probabilistic directed acyclic graphical model is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph.

1. **Differentiate hard and softclustering.**

A hard clustering means we have non-overlapping clusters, where each instance belongs to one and only one cluster. In a soft clustering method, a single individual can belong to multiple clusters, often with a confidence (belief) associated with each cluster.

1. **What is inductive machinelearning**

The inductive bias (also known as learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered. In machine learning, one aims to construct algorithms that are able to learn to predict a certain target output.

1. **Explain gradient descentapproximation**

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

1. **DefineBias**

The bias–variance dilemma or problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set: The bias is an error from erroneous assumptions in the learning algorithm.

1. **Definepruning**

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction ofoverfitting.