

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

Sub Code : 18CSL76

Hrs / Week : 01I + 02P

Total Hrs : 40

IA Marks : 40

Exam Hours : 03

Exam Marks : 60

### 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
3. Implement and evaluate AI and ML algorithms in and Python programming language.

### Experiment List :

1. Implement A\* Search algorithm.
2. Implement AO\* Search algorithm.
3. 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.
4. 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.
5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
6. 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.
7. 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.

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

At the end of the course students will be able to

CO1- Understand the implementation procedures for the Artificial Intelligence and Machine Learning algorithms

CO2- Design Python programs for various Learning algorithms and apply appropriate data sets to the Machine Learning algorithms

CO3- Identify and apply Machine Learning algorithms to solve real world problems

**Conduct of Practical Examination:**

1. All laboratory experiments are to be included for practical examination.
2. Students are allowed to pick one experiment from the lot.
3. Marks distribution: Procedure + Conduction + Viva: 15 + 70 + 18 = 100 Marks

## INTRODUCTION

### Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with 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.

### 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:

1. **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:
2. **Semi-supervised learning:** the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
3. **Active 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.
4. **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.
5. **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).

### **Machine Learning Applications**

In 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) 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, also a supervised problem, the outputs are continuous rather than discrete. In clustering, 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 finds the distribution of inputs in some space.

Dimensionality reduction simplifies inputs by mapping them into a lower dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked with finding out which documents cover similar topics.

### **Machine learning Approaches**

#### **1. Decision tree learning**

Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

#### **2. Association rule learning**

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

#### **3. Artificial neural networks**

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

#### **4. Deep learning**

Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of 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 and speech Recognition.

#### **5. Inductive logic programming**

Inductive logic programming (ILP) is an approach to rule learning using 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 all positive and no negative examples. 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.

#### **6. Support vector machines**

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. 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.

#### **7. 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, and a common technique for statistical data analysis.

#### **8. Bayesian networks**

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic

graphical model that represents a set of random variables and their conditional independencies via a 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 and learning.

#### **9. 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 problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

#### **10. 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 systems.

#### **11. Genetic algorithms**

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype 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.

#### **12. Rule-based machine learning**

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 classifier systems, association rule learning, and artificial immune systems.

### 13.Feature selection approach

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.

## CONTENT BEYOND SYLLABUS

### PYTHON

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry.

Below are some facts about Python Programming Language:

1. Python is currently the most widely used multi-purpose, high-level programming language.
2. Python allows programming in Object-Oriented and Procedural paradigms.
3. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
4. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc.
5. The biggest strength of Python is huge collection of standard library which can be used for the following:
  - Machine Learning
  - GUI Applications (like Kivy, Tkinter, PyQt etc. )
  - Web frameworks like Django (used by YouTube, Instagram, Dropbox)
  - Image processing (like OpenCV, Pillow)
  - Web scraping (like Scrapy, BeautifulSoup, Selenium)
  - Test frameworks
  - Multimedia
  - Scientific computing



## Python Basic Programs:

### 1. Print Statement

```
# Python code for "Hello World"
# nothing else to type...see how simple is the syntax.
print("Hello World")
```

### 2. Input and Output

```
# Python program to illustrate
# getting input from user
name = input("Enter your name: ")
# user entered the name 'harssh'
print("hello", name)
```

### 3. Variables in python

```
# Python program to declare variables
myNumber = 3
print(myNumber)

myNumber2 = 4.5
print(myNumber2)

myNumber = "helloworld"
print(myNumber)
```

### 4. List in python

```
# Python program to illustrate a list
# creates a empty list
nums = []

# appending data in list
nums.append(21)
```

```
nums.append(40.5)
nums.append("String")
print(nums)
```

### **5. Selection in python**

```
# Python program to illustrate
# selection statement
```

```
num1 = 34
if(num1>12):
    print("Num1 is good")
elif(num1>35):
    print("Num2 is not gooooo....")
else:
    print("Num2 is great")
```

### **6. Functions in python**

```
a)# Python program to illustrate
# functions
def hello():
    print("hello")
    print("hello again")
hello()

# calling function
hello()
```

```
b)# Python program to illustrate
# function with main
def getInteger():
    result = int(input("Enter integer: "))
    return result

def Main():
    print("Started")

    # calling the getInteger function and
    # storing its returned value in the output variable
    output = getInteger()
    print(output)

# now we are required to tell Python
# for 'Main' function existence
if __name__=="__main__":
    Main()
```

## 7. For, while, break, continue keyword

```
# Using for loop
for i in range(10):
    print(i, end = " ")
    # break the loop as soon it sees 6
    if i == 6:
        break
    print()
# loop from 1 to 10
i = 0
while i < 10:
```

```
# If i is equals to 6,  
# continue to next iteration  
# without printing  
if i == 6:  
    i+= 1  
    continue  
else:  
    # otherwise print the value  
    # of i  
    print(i, end = " ")  
    i += 1
```

## 8. Python NumPy

It is a general-purpose array processing package which provides tools for handling the n-dimensional arrays. It provides various computing tools such as comprehensive mathematical functions, linear algebra routines.

a)# importing numpy module

```
import numpy as np
```

```
# creating list
```

```
list = [1, 2, 3, 4]
```

```
# creating numpy array
```

```
sample_array = np.array(list1)
```

```
print("List in python : ", list)
```

```
print("Numpy Array in python :",sample_array)
```

b)# importing numpy module

**import** numpy as np

# creating list

list\_1 = [1, 2, 3, 4]

list\_2 = [5, 6, 7, 8]

list\_3 = [9, 10, 11, 12]

# creating numpy array

sample\_array = np.array([list\_1,list\_2,list\_3])

print("Numpy multi dimensional array in python\n",sample\_array)

## 9. Pandas

It is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analysing data much easier. Pandas is fast and it has high-performance & productivity for users.

# Import pandas

**import** pandas as pd

# reading csv file

pd.read\_csv("filename.csv")

a)**import** pandas as pd

# Import the excel file and call it xls\_file

excel\_file = pd.ExcelFile('pandasEx.xlsx')

# View the excel\_file's sheet names

print(excel\_file.sheet\_names)

# Load the excel\_file's Sheet1 as a dataframe

df = excel\_file.parse('Sheet1')

print(df)

```
weather_data={
    'day':['1//12017','1/2/2017','1/3/2017','1/4/2017','1/5/2017','1/6/2017'],
    'temperature':[32,35,28,24,32,31],
    'windspeed':[6,7,2,7,4,2],
    'event':['Rain','Sunny','Snow','Snow','Rain','Sunny']
}
df=pd.DataFrame(weather_data)
df
import pandas as pd
df=pd.read_excel('D:\personal\ML progrm\DataFlair\weather.xlsx','Sheet1')
df
def convert_cell(cell):
    if cell=="Rainy":
        return 'Sunny'
    return cell
df=pd.read_excel('D:\personal\ML
progrm\DataFlair\weather.xlsx','Sheet1',converters={'event':convert_cell})
df
df_stocks=pd.DataFrame({
    'ticket':['GOOGL','WMT','MSFT'],
    'price':[845,65,64],
    'pe':[30.37,14.26,2.12]
})

df_weather=pd.DataFrame({
    'day':['1//12017','1/2/2017','1/3/2017','1/4/2017','1/5/2017','1/6/2017'],
    'temperature':[32,35,28,24,32,31],
    'windspeed':[6,7,2,7,4,2],
    'event':['Rain','Sunny','Snow','Snow','Rain','Sunny']
})
```

## 10. Matplotlib

It is a low level graph plotting library in python that serves as a visualization utility.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
```

```
plt.bar(x,y)
plt.show()

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

## 11. Scikit-learn

Sklearn is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Installation : `conda install scikit-learn`

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.3, random_state = 1
)
```



```
print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.4, random_state=1
)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
classifier_knn = KNeighborsClassifier(n_neighbors = 3)
classifier_knn.fit(X_train, y_train)
y_pred = classifier_knn.predict(X_test)
# Finding accuracy by comparing actual response values(y_test)with predicted response
value(y_pred)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
# Providing sample data and the model will make prediction out of that data

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]
preds = classifier_knn.predict(sample)
pred_species = [iris.target_names[p] for p in preds] print("Predictions:", pred_species)
```

## 12. Steps in Machine learning projects

### 1. Installing the Python:

Python can be downloaded from **Python Software Foundation** website at python.org.

<https://www.python.org/downloads/windows/>

# Check the versions of libraries

```
# Python version
```

```
import sys
```

```
print('Python: {}'.format(sys.version))
```

```
# scipy
```

```
import scipy
```

```
print('scipy: {}'.format(scipy.__version__))
```

```
# numpy
```

```
import numpy
```

```
print('numpy: {}'.format(numpy.__version__))
```

```
# matplotlib
```

```
import matplotlib
```

```
print('matplotlib: {}'.format(matplotlib.__version__))
```

```
# pandas
```

```
import pandas
```

```
print('pandas: {}'.format(pandas.__version__))
```

```
# scikit-learn
```

```
import sklearn
```

```
print('sklearn: {}'.format(sklearn.__version__))
```

## 2. Loading the dataset.

import all of the modules, functions and objects

```
from pandas import read_csv
```

```
from pandas.plotting import scatter_matrix
```

```
from matplotlib import pyplot
```

```
# Load dataset
```

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
```

```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  
dataset = read_csv(url, names=names)
```

### 3. Summarizing the dataset.

In this step we are going to take a look at the data a few different ways:

1. Dimensions of the dataset.: We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the shape property.

```
# shape  
print(dataset.shape)
```

### 2. Peek at the data itself.

```
# head  
print(dataset.head(20))
```

3. Statistical summary of all attributes: This includes the count, mean, the min and max values as well as some percentiles.

```
# descriptions  
print(dataset.describe())
```

4. Breakdown of the data by the class variable.: number of instances (rows) that belong to each class. We can view this as an absolute count.

```
class distribution  
print(dataset.groupby('class').size())
```

```
from pandas import read_csv  
# Load dataset
```

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
# shape
print(dataset.shape)
# head
print(dataset.head(20))
# descriptions
print(dataset.describe())
# class distribution
print(dataset.groupby('class').size())
```

4. Visualizing the dataset.: Given that the input variables are numeric, we can create box and whisker plots of each.

```
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
# histograms
dataset.hist()
pyplot.show()
# scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```

5. Evaluating some algorithms.
6. Making some predictions.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
random_state=1)
# Make predictions on validation dataset
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
# Evaluate predictions
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

## Experiment No 1

### Implement A\* Search algorithm

**Task: find the shortest path between an initial and a final point.**

A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. This algorithm is the advanced form of BFS algorithm (Breadth-first search), which searches for shorter path first than the longer paths. A\* search algorithms used in many games and web-based maps to find the shortest path very efficiently.

Explanation

Basic concepts of A\*

$$F(n) = g(n) + h(n)$$

Where

$g(n)$  : The actual cost path from start node to the current node

$h(n)$  : The actual cost path from the current node to goal node

$f(n)$  : The actual cost path from the start node to the goal node

For implementation of A\* algorithm we have to use two arrays namely Open and CLOSE.

OPEN: An array that contains the nodes that have been generated but have not been yet examined till yet.

CLOSE: an array which contains the nodes which are examined.

### Algorithm

1. Firstly place the starting node into OPEN and find its  $f(n)$  value.
2. Then remove the node from OPEN, having the smallest  $f(n)$  value. If it is goal node, then stop and return to success.
3. Else remove the node from OPEN, and find all its successors.
4. Find the  $f(n)$  value of all successors, place them into OPEN, and place the removed node in close.
5. Goto Step -2
6. Exit

**A\* search in Python programming:**

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)

    closed_set = set()

    g = {} #store distance from starting node

    parents = {} # parents contains an adjacency map of all nodes

    #distance of starting node from itself is zero
    g[start_node] = 0

    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None

        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
```

```
        open_set.add(m)

        parents[m] = n

        g[m] = g[n] + weight

        #for each node m,compare its distance from start i.e g(m) to the
        #from start through n node
        else:

            if g[m] > g[n] + weight:

                #update g(m)

                g[m] = g[n] + weight

                #change parent of m to n

                parents[m] = n

                #if m in closed set,remove and add to open

                if m in closed_set:

                    closed_set.remove(m)

                    open_set.add(m)

    if n == None:

        print('Path does not exist!')

        return None

    # if the current node is the stop_node

    # then we begin reconstructin the path from it to the start_node

    if n == stop_node:

        path = []
```



```
while parents[n] != n:
    path.append(n)
    n = parents[n]
path.append(start_node)
path.reverse()
print('Path found: {}'.format(path))
return path

# remove n from the open_list, and add it to closed_list
# because all of his neighbors were inspected
open_set.remove(n)
closed_set.add(n)

print('Path does not exist!')
return None

#define function to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
```

```
def heuristic(n):  
    H_dist = {  
        'A': 11,  
        'B': 6,  
        'C': 99,  
        'D': 1,  
        'E': 7,  
        'G': 0,  
    }  
    return H_dist[n]  
  
#Describe your graph here  
Graph_nodes = {  
    'A': [('B', 2), ('E', 3)],  
    'B': [('C', 1), ('G', 9)],  
    'C': None,  
    'E': [('D', 6)],  
    'D': [('G', 1)],  
}  
aStarAlgo('A', 'G')
```

**Output:**

```
Path found: ['A', 'E', 'D', 'G']  
['A', 'E', 'D', 'G']
```

## Experiment No 2

### Implement AO\* Search algorithm.

**Task: find more than one solution by ANDing more than one branch.**

AO\* Algorithm basically based on problem decomposition (Breakdown problem into small pieces). When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs** or **AND - OR trees** are used for representing the solution.

Our real-life situations can't be exactly decomposed into either AND tree or OR tree but is always a combination of both. So, we need an AO\* algorithm where O stands for 'ordered'. AO\* algorithm represents a part of the search graph that has been explicitly generated so far.

### Algorithm

**AO\* algorithm is given as follows:**

**Step-1:** Create an initial graph with a single node (start node).

**Step-2:** Transverse the graph following the current path, accumulating node that has not yet been expanded or solved.

**Step-3:** Select any of these nodes and explore it. If it has no successors then call this value - **FUTILITY** else calculate  $f'(n)$  for each of the successors.

**Step-4:** If  $f'(n)=0$ , then mark the node as **SOLVED**.

**Step-5:** Change the value of  $f'(n)$  for the newly created node to reflect its successors by backpropagation.

**Step-6:** Whenever possible use the most promising routes, If a node is marked as **SOLVED** then mark the parent node as **SOLVED**.

**Step-7:** If the starting node is **SOLVED** or value is greater than **FUTILITY** then stop else repeat from Step-2.

## Experiment No 3

**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

The candidate elimination algorithm incrementally builds the version space given a hypothesis space  $H$  and a set  $E$  of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

The version space, denoted hypothesis space  $H$  and training examples  $D$ , is the subset of hypotheses from  $H$  consistent with the training examples in  $D$ .

$$VS_{H,D} = \{h \in H \mid \text{Consistent}(h, D)\}$$

**Dataset:** "Days on which my friend Aldo enjoys his favorite water sport."

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

**Candidate-Elimination Algorithm:**

1. Load data set
2.  $G \leftarrow$  maximally general hypotheses in  $H$
3.  $S \leftarrow$  maximally specific hypotheses in  $H$

4. For each training

example  $d = \langle x, c(x) \rangle$

Case 1 : If  $d$  is a

positive example

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

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

Case 2: If  $d$  is a negative example

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

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

$S_0$   $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$G_0$   $\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle +$

$S_0$   $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$S_1$   $\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle$

$G_0, G_1$

$\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle +$

$S_1$   $\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle$

$S_2$   $\langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

$G_1, G_2$

$\langle ?, ?, ?, ?, ?, ? \rangle$

For training example d,

$\langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle -$

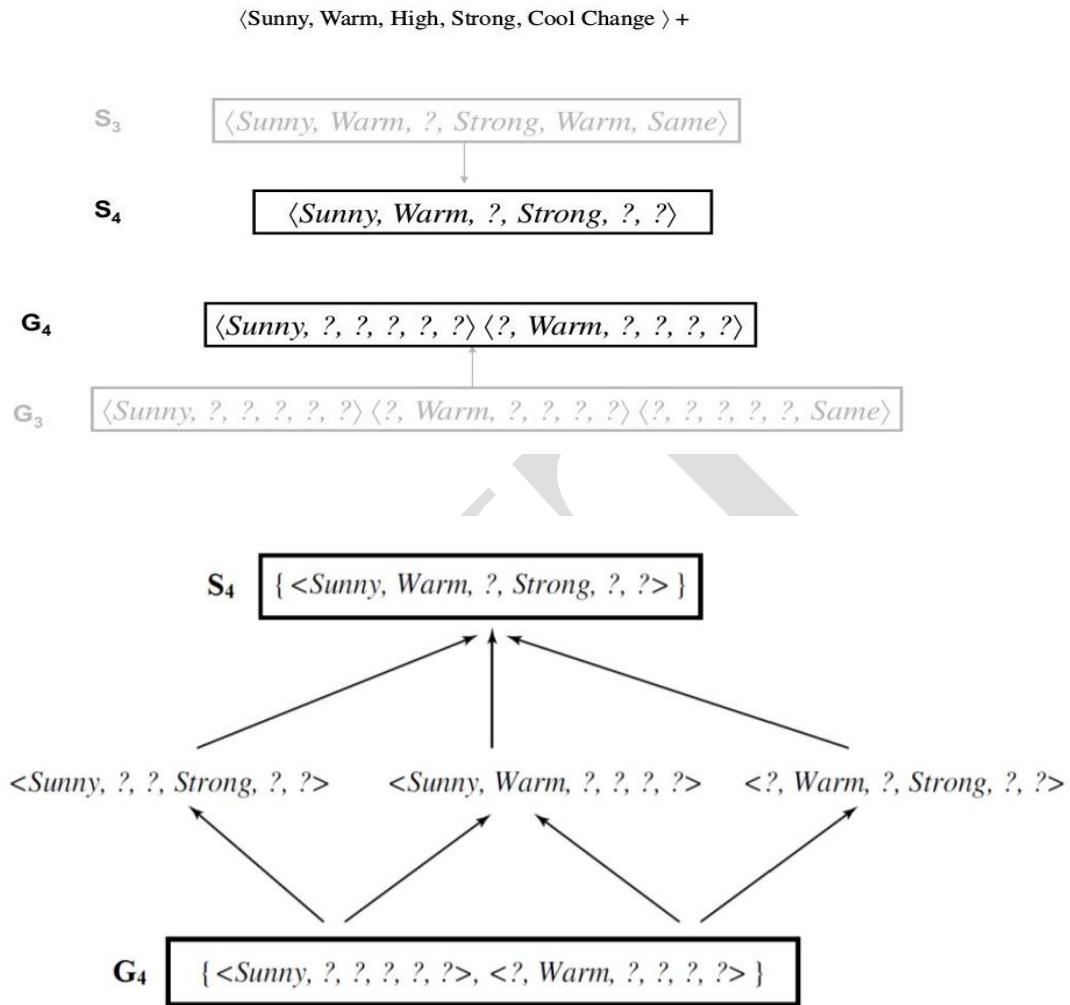
$S_2, S_3$   $\langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

$G_3$   $\langle \text{Sunny, ?, ?, ?, ?, ?} \rangle \langle \text{?, Warm, ?, ?, ?, ?} \rangle \langle \text{?, ?, ?, ?, ?, Same} \rangle$

$G_2$

$\langle ?, ?, ?, ?, ?, ? \rangle$

for training example d,



### Candidate Elimination algorithm in python programming:

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv("D:\\20-21\\17CSL76-ML-LAB\\dataset\\enjoysport (pg-2).csv"))
#data = pd.read_csv("D:\\20-21\\17CSL76-ML-LAB\\dataset\\enjoysport (pg-2).csv")
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
```

```
def learn(concepts, target):
    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)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        if target[i] == "no":
            print("If instance is Negative ")
            for x in range(len(specific_h)):
                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)
    print("\n")
    print("\n")
```

```
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
```



```
for i in indices:
```

```
    general_h.remove(['?', '?', '?', '?', '?', '?'])
```

```
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")
```

### Output:

```
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
, '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
, '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
    steps of Candidate Elimination Algorithm 1
    ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
    [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
, '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
, '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts
If instance is Positive
    steps of Candidate Elimination Algorithm 2
    ['sunny' 'warm' '?' 'strong' 'warm' 'same']
    [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
, '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
, '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts
If instance is Negative
    steps of Candidate Elimination Algorithm 3
    ['sunny' 'warm' '?' 'strong' 'warm' 'same']
    [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['
?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '
?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
```

For Loop Starts

If instance is Positive

steps of Candidate Elimination Algorithm 4

```
['sunny' 'warm' '?' 'strong' '?' '?']  
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['  
?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '  
?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Final Specific\_h:

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

Final General\_h:

```
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

## Experiment No 4

**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:** ID3 determines the 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:

$$\text{Entropy}(S) = -\sum P(I) \cdot \log_2 P(I)$$

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum [P(S/A) \cdot \text{Entropy}(S/A)]$$

ID3 stands for Iterative Dichotomiser 3. Algorithm used to generate a decision tree. ID3 is a precursor to the C4.5 Algorithm. The ID3 algorithm was invented by Ross Quinlan.

Decision Tree Classifies data using the attributes, Tree consists of decision nodes and decision leafs. Nodes can have two or more branches which represents the value for the attribute tested. Leaf nodes produce a homogeneous result. The ID3 follows the Occam's razor principle. Attempts to create the smallest possible decision tree.

Entropy Formula to calculate : A complete homogeneous sample has an entropy of 0, An equally divided sample has an entropy of 1. **Entropy** =  $-p \log_2(p) - (1-p) \log_2(1-p)$  for a sample of negative and positive elements.

Information gain is based on the decrease in entropy after a dataset is split on an attribute. Looking for which attribute creates the most homogeneous branches

**Dataset:**

outlook	temperature	humidity	windy	play
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rainy	mild	high	weak	yes
rainy	cool	normal	weak	yes
rainy	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rainy	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rainy	mild	high	strong	no

**Calculation:**

Decision/play column consists of 14 instances and includes two labels: yes and no. There are 9 decisions labeled yes and 5 decisions labeled no.

**Find the entropy of the class variable.**

$$E(S) = -[(9/14)\log(9/14) + (5/14)\log(5/14)] = 0.94$$

**calculate average weighted entropy.** ie, we have found the total of weights of each feature multiplied by probabilities.

$$E(S, \text{outlook}) = (5/14)*E(3,2) + (4/14)*E(4,0) + (5/14)*E(2,3) = (5/14)*(-(3/5)\log(3/5) - (2/5)\log(2/5)) + (4/14)*(0) + (5/14)*((2/5)\log(2/5) - (3/5)\log(3/5)) = 0.693$$

**The next step is to find the information gain.** It is the difference between parent entropy and average weighted entropy we found above.

$$IG(S, \text{outlook}) = 0.94 - 0.693 = 0.247$$

Similarly find Information gain for Temperature, Humidity, and Windy.

$$IG(S, \text{Temperature}) = 0.940 - 0.911 = 0.029$$

$$IG(S, \text{Humidity}) = 0.940 - 0.788 = 0.152$$

$$IG(S, \text{Windy}) = 0.940 - 0.8932 = 0.048$$

**Calculate parent entropy  $E(\text{sunny})$**

$$E(\text{sunny}) = -(3/5)\log(3/5) - (2/5)\log(2/5) = 0.971.$$

$$E(\text{sunny}, \text{Temperature}) = (2/5)*E(0,2) + (2/5)*E(1,1) + (1/5)*E(1,0) = 2/5 = 0.4$$

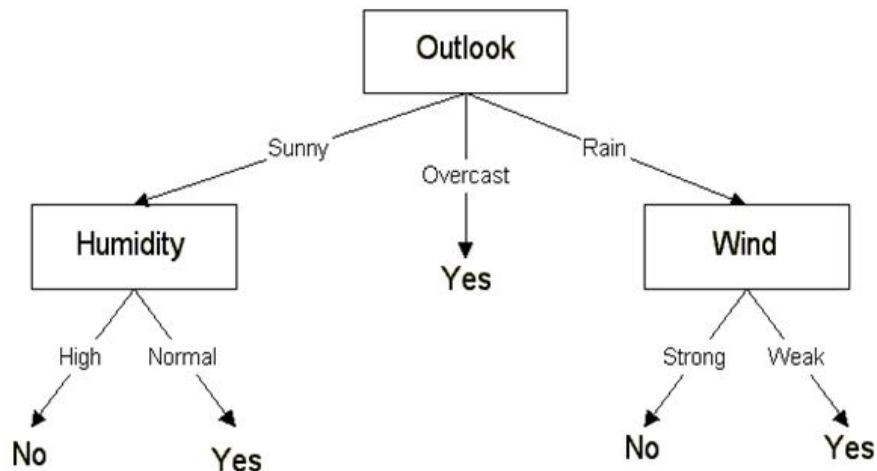
**calculate information gain.**

$$IG(\text{sunny}, \text{Temperature}) = 0.971 - 0.4 = 0.571$$

Similarly we get

$$IG(\text{sunny}, \text{Humidity}) = 0.971$$

$$IG(\text{sunny}, \text{Windy}) = 0.020$$



### Algorithm:

ID3(Examples, Target\_attribute, Attributes)

Examples are the training examples. Target\_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree.

Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -

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

### **ID3 program in Python programming:**

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3-dataset.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
```

```
self.children = []  
self.value = ""  
self.isLeaf = False  
self.pred = ""
```

```
def entropy(examples):  
    pos = 0.0  
    neg = 0.0  
    for _, row in examples.iterrows():  
        if row["answer"] == "yes":  
            pos += 1  
        else:  
            neg += 1  
    if pos == 0.0 or neg == 0.0:  
        return 0.0  
    else:  
        p = pos / (pos + neg)  
        n = neg / (pos + neg)  
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```
def info_gain(examples, attr):  
    uniq = np.unique(examples[attr])  
    #print ("\n",uniq)  
    gain = entropy(examples)  
    #print ("\n",gain)  
    for u in uniq:  
        subdata = examples[examples[attr] == u]  
        #print ("\n",subdata)  
        sub_e = entropy(subdata)  
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e  
        #print ("\n",gain)
```

```
    return gain
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
```



```
        child = ID3(subdata, new_attrs)
        dummyNode.children.append(child)
        root.children.append(dummyNode)

    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
root = ID3(data, features)
printTree(root)
```

**Output:**

```
outlook
    overcast ->  ['yes']
    rain
        wind
            strong ->  ['no']
            weak ->  ['yes']
    sunny
        humidity
            high ->  ['no']
            normal ->  ['yes']
```

### **Experiment No 4**

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

**Task: implement multilayer feed-forward networks and train the weights**

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks, the backpropagation 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. Backpropagation 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

**BACKPROPAGATION (training\_example,  $\eta$ ,  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$ )**

Each training example is a pair of the form  $(x, t)$ , where  $(x)$  is the vector of network input values, and  $(t)$  is the vector of target network output values.

$\eta$  is the learning rate (e.g., 0.05).

$n_i$ , is the number of network inputs,

$n_{hidden}$  the number of units in the hidden layer, and

$n_{out}$  the number of output units.

The input from unit  $i$  into unit  $j$  is denoted  $x_{ji}$ , and the weight from unit  $i$  to unit  $j$  is denoted  $w_{ji}$

**Steps in Backpropagation algorithm**

1. Create a feed-forward network with  $n_i$  inputs,  $n_{hidden}$  hidden units, and  $n_{out}$  output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do

For each  $(x, t)$ , in training examples, Do

Propagate the input forward through the network:

1. Input the instance  $x$ , to the network and compute the output  $o_u$  of every unit  $u$  in the network.

Propagate the errors backward through the network

2. For each network unit  $k$ , calculate its error term  $\delta_k$

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

3. For each network unit  $h$ , calculate its error term  $\delta_h$

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

4. Update each network weight  $w_{ji}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

### **Backpropogation program in python programming:**

```
import numpy as np
X = np.array([[2, 9], [1, 5], [3, 6]]) # Hours Studied, Hours Slept
y = np.array([92, [86], [89]]) # Test Score
y = y/100 # max test score is 100

#Sigmoid Function
def sigmoid(x): #this function maps any value between 0 and 1
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
```

```
epoch=10000 #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 of output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bias_hidden=np.random.uniform(size=(1,hiddenlayer_neurons)) #bias matrix to the hidden layer
weight_hidden=np.random.uniform(size=(hiddenlayer_neurons,output_neurons)) #weight matrix
to the output layer
bias_output=np.random.uniform(size=(1,output_neurons)) # matrix to the output layer

for i in range(epoch):
    #Forward Propagation
    hinp1=np.dot(X,wh)
    hinp= hinp1 + bias_hidden #bias_hidden GRADIENT DISCENT
    hlayer_activation = sigmoid(hinp)

    outinp1=np.dot(hlayer_activation,weight_hidden)
    outinp= outinp1 + bias_output
    output = sigmoid(outinp)

    #Backpropagation
    EO = y-output #Compare prediction with actual output and calculate the gradient of error
    (Actual – Predicted)

    outgrad = derivatives_sigmoid(output) #Compute the slope/ gradient of hidden and output
    layer neurons
    d_output = EO * outgrad #Compute change factor(delta) at output layer, dependent on the
    gradient of error multiplied by the slope of output layer activation
```

EH = d\_output.dot(weight\_hidden.T) #At this step, the error will propagate back into the network which means error at hidden layer. we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (weight\_hidden.T).

hiddengrad = derivatives\_sigmoid(hlayer\_activation) #how much hidden layer weight contributed to error

d\_hiddenlayer = EH \* hiddengrad

#update the weights

weight\_hidden += hlayer\_activation.T.dot(d\_output) \*lr# dot product of nextlayererror and currentlayerop

bias\_hidden += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

bias\_output += np.sum(d\_output, axis=0,keepdims=True) \*lr

print("Input: \n" + str(X))

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

print("Predicted Output: \n",output)

### **Output:**

Input:

[[2 9]

[1 5]

[3 6]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89685473]

[0.873499 ]

[0.89828536]]

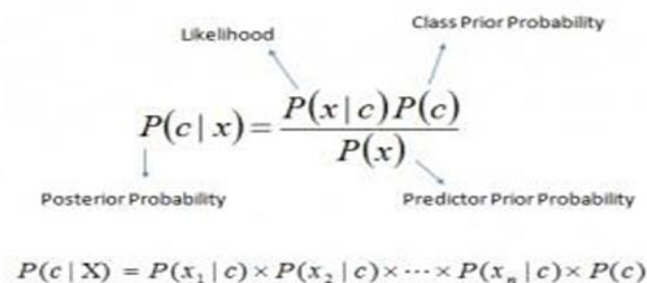
## Experiment No:5

**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 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

Bayes theorem provides a way of calculating posterior probability  $P(c|x)$  from  $P(c)$ ,  $P(x)$  and  $P(x|c)$ .


$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Above,

- $P(c|x)$  is the posterior probability of class (c, target) given predictor (x, attributes).
- $P(c)$  is the prior probability of class.
- $P(x|c)$  is the likelihood which is the probability of predictor given class.
- $P(x)$  is the prior probability of predictor.

**Gaussian Naive Bayes** :A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution.

**Representation for Gaussian Naive Bayes** :We calculate the probabilities for input values for each class using a frequency. With realvalued inputs, we can calculate the mean and standard

deviation of input values (x) for each class to summarize the distribution. This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

**Gaussian Naive Bayes Model from Data** :The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\begin{aligned}\mu &= \frac{1}{n} \sum_{i=1}^n x_i && \text{Mean} \\ \sigma &= \left[ \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 \right]^{0.5} && \text{Standard deviation} \\ f(x) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} && \text{Normal distribution}\end{aligned}$$

Examples:

- The data set used in this program is the Pima Indians Diabetes problem.
- This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome .
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

### **Navies Bayseian Classifier in python programming:**

```
import csv
```

```
import random
```

```
import math
```

```
# 1.Data Handling
```

```
# 1.1 Loading the Data from csv file of Pima indians diabetes dataset.
```

```
def loadcsv(filename):
```

```
    lines = csv.reader(open(filename, "r"))
```

```
    dataset = list(lines)
```

```
    for i in range(len(dataset)):
```

```
        # converting the attributes from string to floating point numbers
```

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

```
    return dataset
```

```
#1.2 Splitting the Data set into Training Set
```

```
def splitDataset(dataset, splitRatio):
```

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

```
    trainSet = []
```

```
    copy = list(dataset)
```

```
    while len(trainSet) < trainSize:
```



```
index = random.randrange(len(copy)) # random index
trainSet.append(copy.pop(index))
return [trainSet, copy]
```

## #2.Summarize Data

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

### #2.1: Separate Data By Class

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

```
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
```

```
#The mean is the central middle or central tendency of the data,
# and we will use it as the middle of our gaussian distribution
# when calculating probabilities
```

### #2.2 : Calculate Mean

```
def mean(numbers):
    return sum(numbers)/float(len(numbers))
```

#The standard deviation describes the variation of spread of the data,  
#and we will use it to characterize the expected spread of each attribute  
#in our Gaussian distribution when calculating probabilities.

#### #2.3 : Calculate Standard Deviation

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

#### #2.4 : Summarize Dataset

#Summarize Data Set for a list of instances (for a class value)  
#The zip function groups the values for each attribute across our data instances  
#into their own lists so that we can compute the mean and standard deviation values  
#for the attribute.

```
def summarize(dataset):  
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]  
    del summaries[-1]  
    return summaries
```

#### #2.5 : Summarize Attributes By Class

#We can pull it all together by first separating our training dataset into  
#instances grouped by class. Then calculate the summaries for each attribute.

```
def summarizeByClass(dataset):  
    separated = separateByClass(dataset)  
    summaries = { }  
    for classValue, instances in separated.items():  
        summaries[classValue] = summarize(instances)  
    return summaries
```

### #3. Make Prediction

#### #3.1 Calculate 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
```

#### #3.2 Calculate Class Probabilities

```
def calculateClassProbabilities(summaries, inputVector):  
    probabilities = { }  
    for classValue, classSummaries in summaries.items():  
        probabilities[classValue] = 1  
        for i in range(len(classSummaries)):  
            mean, stdev = classSummaries[i]  
            x = inputVector[i]  
            probabilities[classValue] *= calculateProbability(x, mean, stdev)  
    return probabilities
```

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

```
def predict(summaries, inputVector):  
    probabilities = calculateClassProbabilities(summaries, inputVector)  
    bestLabel, bestProb = None, -1  
    for classValue, probability in probabilities.items():  
        if bestLabel is None or probability > bestProb:  
            bestProb = probability  
            bestLabel = classValue  
    return bestLabel
```

### #4. Make Predictions

# Function which return predictions for list of predictions

# For each instance

```
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions
```

#### #5. Computing Accuracy

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

#### #Main Function

```
def main():
    filename = 'D:\\20-21\\17CSL76-ML-LAB\\pima_indian.csv'
    splitRatio = 0.67
    dataset = loadcsv(filename)

    #print("\n The Data Set :\n",dataset)
    print("\n The length of the Data Set : ",len(dataset))

    print("\n The Data Set Splitting into Training and Testing \n")
    trainingSet, testSet = splitDataset(dataset, splitRatio)

    print('\n Number of Rows in Training Set:{0} rows'.format(len(trainingSet)))
    print('\n Number of Rows in Testing Set:{0} rows'.format(len(testSet)))
```

```
print("\n First Five Rows of Training Set:\n")
for i in range(0,5):
    print(trainingSet[i],"\n")

print("\n First Five Rows of Testing Set:\n")
for i in range(0,5):
    print(testSet[i],"\n")

# prepare model
summaries = summarizeByClass(trainingSet)
print("\n Model Summaries:\n",summaries)

# test model
predictions = getPredictions(summaries, testSet)
print("\nPredictions:\n",predictions)

accuracy = getAccuracy(testSet, predictions)
print("\n Accuracy: {0}%'.format(accuracy))

main()
```

**Output:**

The length of the Data Set : 768

The Data Set Splitting into Training and Testing

Number of Rows in Training Set:514 rows

Number of Rows in Testing Set:254 rows

First Five Rows of Training Set:

[4.0, 158.0, 78.0, 0.0, 0.0, 32.9, 0.803, 31.0, 1.0]

[3.0, 102.0, 74.0, 0.0, 0.0, 29.5, 0.121, 32.0, 0.0]

[7.0, 97.0, 76.0, 32.0, 91.0, 40.9, 0.871, 32.0, 1.0]

[5.0, 109.0, 75.0, 26.0, 0.0, 36.0, 0.546, 60.0, 0.0]



## Experiment No: 7

**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.**

### Introduction to 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.

**Distance functions**

Euclidean	$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$
Manhattan	$\sum_{i=1}^k  x_i - y_i $
Minkowski	$\left( \sum_{i=1}^k ( x_i - y_i )^q \right)^{1/q}$

### EM algorithm in python programming:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
```

```
import sklearn.metrics as sm
import pandas as pd
import numpy as np
#import matplotlib inline

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

#colormap = np.array(['red', 'lime', 'black'])

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

# View the results

# Set the size of the plot
plt.figure(figsize=(14,7))

# Create a colormap
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
```



```
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')

# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))
# Create a colormap
#print("The accuracy score: ",sm.accuracy_score(y, model.labels_))
#sm.confusion_matrix(y, model.labels_)

predY = np.choose(model.labels_, [0, 1, 2]).astype(np.int64)
print (predY)

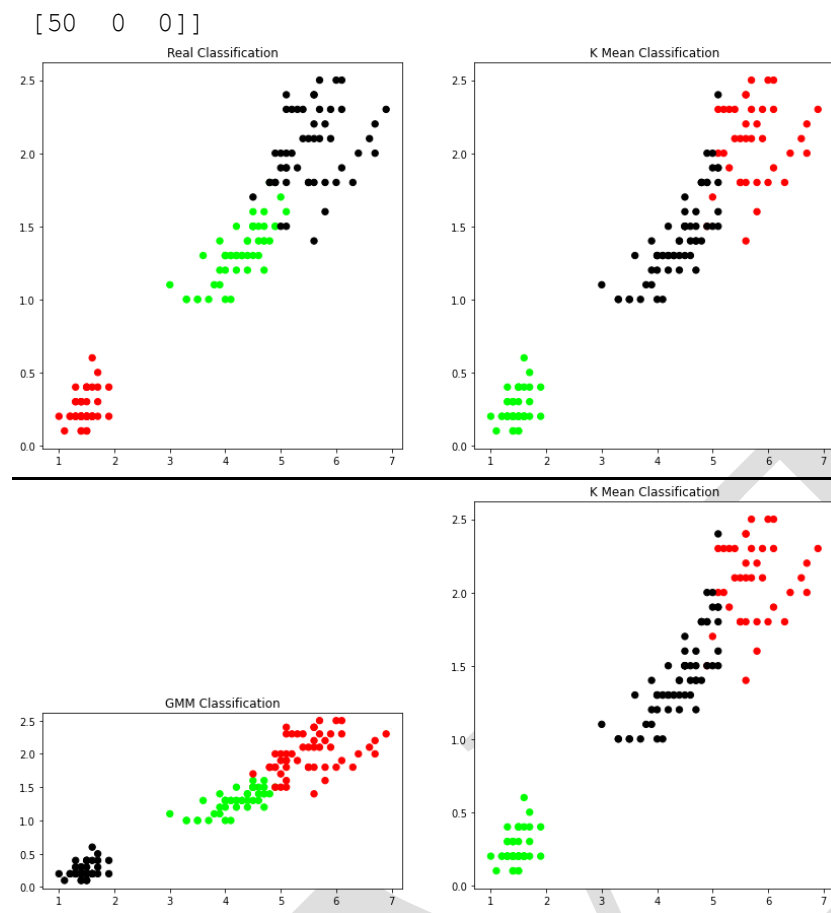
#colormap = np.array(['red', 'lime', 'black'])
# Plot Original
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
# Plot Predicted with corrected values
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[predY], s=40)
plt.title('K Mean Classification')

print("The accuracy score of K-Mean: ",sm.accuracy_score(y, model.labels_))
print("The Confusion matrixof K-Mean: ",sm.confusion_matrix(y, model.labels_))
```

**Output:**

---

Dept. of ISE, SVIT, Bengaluru



## Experiment No:8

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

**Task: The task of this program is to classify the IRIS data set examples by using the k-Nearest Neighbour algorithm. The new instance has to be classified based on its k nearest neighbors.**

KNN falls in the supervised learning family of algorithms. Informally, this means that we are given a labeled dataset consisting of training observations  $(x, y)$  and would like to capture the relationship between  $x$  and  $y$ . More formally, our goal is to learn a function  $h: X \rightarrow Y$  so that given an unseen observation  $x$ ,  $h(x)$  can confidently predict the corresponding output  $y$ .

The KNN classifier is also a non parametric and instance-based learning algorithm.

- Non-parametric means it makes no explicit assumptions about the functional form of  $h$ , avoiding the dangers of mismodeling the underlying distribution of the data. For example, suppose our data is highly non-Gaussian but the learning model we choose assumes a Gaussian form. In that case, our algorithm would make extremely poor predictions.
- Instance-based learning means that our algorithm doesn't explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as "knowledge" for the prediction phase. Concretely, this means that only when a query to our database is made (i.e. when we ask it to predict a label given an input), will the algorithm use the training instances to spit out an answer.

In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance given by

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \dots + (x_n - x'_n)^2}$$

but other measures can be more suitable for a given setting and include the Manhattan, Chebyshev and Hamming distance. More formally, given a positive integer  $K$ , an unseen observation  $x$  and a similarity metric  $d$ , KNN classifier performs the following two steps:

- It runs through the whole dataset computing  $d$  between  $x$  and each training observation. The

K points in the training data that are closest to  $x$  are called the set  $A$ . Note that  $K$  is usually odd to prevent tie situations.

- It then estimates the conditional probability for each class, that is, the fraction of points in  $A$  with that given class label. (Note  $I(x)$  is the indicator function which evaluates to 1 when the argument  $x$  is true and 0 otherwise)

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in A} I(y^{(i)} = j)$$

- Finally, the input  $x$  gets assigned to the class with the largest probability.

### **k-Nearest Neighbour program in python programming**

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris=datasets.load_iris()
x = iris.data
y = iris.target
print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
#To make predictions on our test data
y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

**Output:**

```
sepal-length sepal-width petal-length petal-width
[[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 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 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]





## Experiment No: 9

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

**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 that creates a smooth line through a timeplot or scatter plot to help you to see relationship between variables and foresee trends.

Locally weighted regression is a very powerful non-parametric model used in statistical learning.

### **Introduction :**

Scatter-diagram smoothing (e.g. using the **lowess()** or **loess()** functions) involves drawing a smooth curve on a scatter diagram to summarize a relationship, in a fashion that makes few assumptions initially about the form or strength of the relationship. It is related to (and is a special case of) *nonparametric regression*, in which the objective is to represent the relationship between a response variable and one or more predictor variables, again in way that makes few assumptions about the form of the relationship. In other words, in contrast to “standard” linear regression analysis, no assumption is made that the relationship is represented by a straight line (although one could certainly think of a straight line as a special case of nonparametric regression).

If the basic decomposition-of-the-data model is:

*data = predictable component + noise,*

then for the standard bivariate or multiple (linear) regression, the model is

*data = straight-line, polynomial or linearizable function + noise,*

while for nonparametric regression, the model is

$data = smooth\ function\ determined\ by\ data + noise.$

Another way of looking at scatter diagram smoothing is as a way of depicting the “local” relationship between a response variable and a predictor variable over parts of their ranges, which may differ from a “global” relationship determined using the whole data set. Nonparametric regression can be thought of as generalizing the scatter plot smoothing idea to the multiple-regression context

Locally Weighted Learning is a class of function approximation techniques, where a prediction is done by using an approximated local model around the current point of interest.

The **goal** of function approximation and regression is to find the underlying relationship between input and output. In a supervised learning problem training data, where each input is associated to one output, is used to create a model that predicts values which come close to the true function. All of these models use complete training data to derive global function.

### 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 a function

This is **an instance-based learning method**

**The idea: whenever you want to classify a sample:**

- Build a local model of the function (using a linear function, quadratic, neural network, etc.)
- Use the model to predict the output value
- Throw the model away.
- Our final method combines advantages of parametric methods with non-parametric. The idea is to fit a regression model locally, weighting examples by the kernel  $K$ .
- Locally Weighted Regression Algorithm

Our final method combines advantages of parametric methods with non-parametric. The idea is to fit a regression model locally, weighting examples by the kernel  $K$ .

Locally Weighted Logistic Regression Algorithm

1. Given training data  $D = \{x_i, y_i\}$ , Kernel function  $K(\cdot)$  and input  $x$
2. Fit weighted logistic regression  $w = \arg\min_w \sum_{i=1}^n K(x, x_i) \log(1 + \exp\{-y_i w \cdot x_i\})$

3. Return logistic regression prediction  $\text{sign}(\mathbf{w}(\mathbf{x}) \mathbf{x})$ .

**Locally weighted regression in python programming:**

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

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

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

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

    # predict value
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernel Bias Function

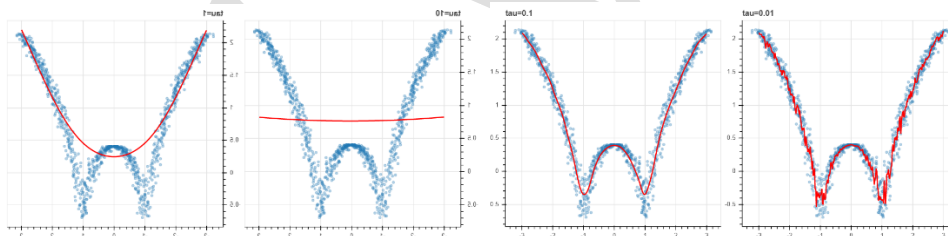
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
```

```
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
    # prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output:



## VIVA QUESTIONS

1. What is machine learning?
2. Define supervised learning
3. Define unsupervised learning
4. Define semi supervised learning
5. Define reinforcement learning
6. What do you mean by hypotheses
7. What is classification
8. What is clustering
9. Define precision, accuracy and recall
10. Define entropy
11. Define regression
12. How Knn is different from k-means clustering
13. What is concept learning
14. Define specific boundary and general boundary
15. Define target function
16. Define decision tree
17. What is ANN
18. Explain gradient descent approximation
19. State Bayes theorem
20. Define Bayesian belief networks
21. Differentiate hard and soft clustering
22. Define variance
23. What is inductive machine learning
24. Why K nearest neighbour algorithm is lazy learning algorithm
25. Why naïve Bayes is naïve
26. Mention classification algorithms
27. Define pruning
28. Differentiate Clustering and classification
29. Mention clustering algorithms
30. Define Bias

SVIT