LAB MANUAL

Lab Name:	Machine Learning Lab
Lab Code:	**6**IT4-22
Branch:	Information Technology
Year:	3rd Year

Jaipur Engineering College and Research Center, Jaipur

Department of Information Technology

(Rajasthan Technical University, KOTA)

VISSION AND MISSION OF INSTITUTE

Vision:

To become a renowned centre of outcome based learning, and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission:

- Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
- Identify areas of focus and provide platform to gain knowledge and solutions based on informed perception of Indian, regional and global needs.
- Offer opportunities for interaction between academia and industry.
- Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

VISION AND MISSION OF THE DEPARTMENT

Vision:

To establish outcome based excellence in teaching, learning and commitment to support IT Industry.

MISSION OF THE DEPARTMENT

Mission:

M1: To provide outcome based education.
M2: To provide fundamental & Intellectual knowledge with essential skills to meet current and future need of IT Industry across the globe.
M3: To inculcate the philosophy of continues learning, ethical values & Social Responsibility.

PROGRAM OUTCOMES

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects in multidisciplinary environments.
12. **Life –long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes needed.

PROGRAM EDUCATIONAL OBJECTIVES

1. To strengthen students with fundamental knowledge, effective computing, problem solving and communication skills enable them to have successful career in Information Technology.

2. To enable students in acquiring Information Technology's latest tools, technologies and management principles to give them an ability to solve multidisciplinary engineering problems.

3. To impart students with ethical values and commitment towards sustainable development in collaborative mode.

4. To reinforce students with research aptitude and innovative approaches which help them to identify, analyse, formulate and solve real life problems and motivates them for lifelong learning.

5. To empower students with leadership quality and team building skills that prepare them for employment, entrepreneurship and to become competent professionals to serve societies and global needs

PROGRAM SPECIFIC OUTCOME

PSO1: Graduates of the program would be able to develop mobile and web based IT solutions for real time problems.

PSO2: Graduates of the program would be able to apply the concepts of artificial intelligence, machine learning and deep learning.

Rajasthan Technical University Syllabus

Year/Semester: III Year- VI Semester: B.Tech. (Information Technology)

Subject Code	6IT4-22	IA Marks	45
Subject	Machine Learning Lab	ETE	30
Number of Lecture Hours/Week	0L+0T+3P	End Term Exam Hours	2 hours

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

Subject: Machine Learning Lab **Code:** 6IT4-22

Course Outcomes	Description	Experiment Number
CO1	Analyze and Interpret data.	1,2, 10
CO2	Classify data set.	3,5,6,9
CO3	Build clusters.	8
CO4	Build machine learning models.	4, 7

Mapping of Course Outcomes with Program Outcomes:

High (H): 3, Medium (M):2, Low (L):1

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	H	H	M	H	H	M	M	M	M	M	M	M
CO2	H	H	M	M	H	M	M	M	M	M	M	M
CO3	H	H	M	M	H	M	M	M	M	M	M	M
CO4	H	H	M	M	H	M	M	M	M	M	M	M

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	2	3	3	2	2	2	2	2	2	2
CO2	3	3	2	2	3	2	2	2	2	2	2	2
CO3	3	3	2	2	3	2	2	2	2	2	2	2
CO4	3	3	2	2	3	2	2	2	2	2	2	2

INTRODUCTION TO MACHINE LEARNING

Machine Learning (ML) is basically that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do. In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The key focus of ML is to allow computer systems to learn from experience without being explicitly programmed or human intervention.

Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven't surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, "to make decisions, based on data, with efficiency and scale".

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can't do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Why & When to Make Machines Learn?

We have already discussed the need for machine learning, but another question arises that in what scenarios we must make the machine learn? There can be several circumstances where we need machines to take data-driven decisions with efficiency and at a huge scale. The followings are some of such circumstances where making machines learn would be more effective.

Lack of human expertise

The very first scenario, in which we want a machine to learn and take data-driven decisions, can be the domain where there is a lack of human expertise. The examples can be navigations in unknown territories or spatial planets.

Dynamic scenarios

There are some scenarios which are dynamic in nature i.e. they keep changing over time. In case of these scenarios and behaviors, we want a machine to learn and take data-driven decisions. Some of the examples can be network connectivity and availability of infrastructure in an organization.

Difficulty in translating expertise into computational tasks

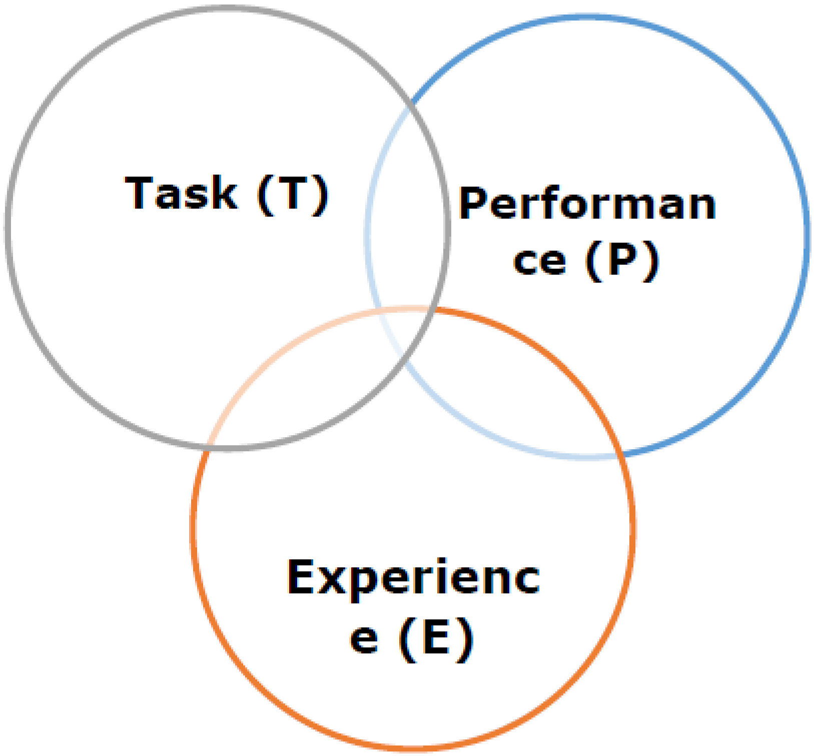
There can be various domains in which humans have their expertise; however, they are unable to translate this expertise into computational tasks. In such circumstances we want machine learning. The examples can be the domains of speech recognition, cognitive tasks etc.

Machine Learning Model

ML is a field of AI consisting of learning algorithms that −

- Improve their performance (P)
- At executing some task (T)
- Over time with experience (E)

Based on the above, the following diagram represents a Machine Learning Model −



Task (T)

From the perspective of problem, we may define the task T as the real-world problem to be solved. The problem can be anything like finding best house price in a specific location or to find best marketing strategy etc. On the other hand, if we talk about machine learning, the definition of task is different because it is difficult to solve ML based tasks by conventional programming approach.

A task T is said to be a ML based task when it is based on the process and the system must follow for operating on data points. The examples of **ML based tasks are Classification, Regression, Structured annotation, Clustering, Transcription etc.**

Experience (E)

As name suggests, it is the knowledge gained from data points provided to the algorithm or model. Once provided with the dataset, the model will run iteratively and will learn some inherent pattern. The learning thus acquired is called experience (E). Making an analogy with human learning, we can think of this situation as in which a human being is learning or gaining some experience from various attributes like situation, relationships etc. **Supervised, unsupervised and reinforcement learning are some ways to learn or gain experience**. The experience gained by out ML model or algorithm will be used to solve the task T.

Performance (P)

An ML algorithm is supposed to perform task and gain experience with the passage of time. The measure which tells whether ML algorithm is performing as per expectation or not is its performance (P). P is basically a quantitative metric that tells how a model is performing the task, T, using its experience, E. There are many metrics that help to understand the ML **performance, such as accuracy score, F1 score, confusion matrix, precision, recall, sensitivity etc.**

Challenges in Machines Learning

While Machine Learning is rapidly evolving, making significant strides with cyber security and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.

Curse of dimensionality − another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

Applications of Machines Learning

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML −

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping

An Introduction to Python

Python is a popular object-oriented programing language having the capabilities of high-level programming language. It's easy to learn syntax and portability capability makes it popular these days. The followings facts gives us the introduction to Python −

- Python was developed by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands.
- It was written as the successor of programming language named 'ABC'.
- Its first version was released in 1991.
- The name Python was picked by Guido van Rossum from a TV show named Monty Python's Flying Circus.
- It is an open source programming language which means that we can freely download it and use it to develop programs. It can be downloaded from www.python.org..
- Python programming language is having the features of Java and C both. It is having the elegant 'C' code and on the other hand, it is having classes and objects like Java for object-oriented programming.
- It is an interpreted language, which means the source code of Python program would be first converted into bytecode and then executed by Python virtual machine.

Strengths and Weaknesses of Python

Every programming language has some strengths as well as weaknesses, so does Python too.

Strengths

According to studies and surveys, Python is the fifth most important language as well as the most popular language for machine learning and data science. It is because of the following strengths that Python has −

Easy to learn and understand − the syntax of Python is simpler; hence it is relatively easy, even for beginners also, to learn and understand the language.

Multi-purpose language − Python is a multi-purpose programming language because it supports structured programming, object-oriented programming as well as functional programming.

Huge number of modules − Python has huge number of modules for covering every aspect of programming. These modules are easily available for use hence making Python an extensible language.

Support of open source community − as being open source programming language, Python is supported by a very large developer community. Due to this, the bugs are easily fixed by the Python community. This characteristic makes Python very robust and adaptive.

Scalability − Python is a scalable programming language because it provides an improved structure for supporting large programs than shell-scripts.

Weakness

Although Python is a popular and powerful programming language, it has its own weakness of slow execution speed.

The execution speed of Python is slow as compared to compiled languages because Python is an interpreted language. This can be the major area of improvement for Python community.

Installing Python

For working in Python, we must first have to install it. You can perform the installation of Python in any of the following two ways −

- Installing Python individually
- Using Pre-packaged Python distribution: **Anaconda**

Installing Python Individually

If you want to install Python on your computer, then then you need to download only the binary code applicable for your platform. Python distribution is available for Windows, Linux and Mac platforms.

The following is a quick overview of installing Python on the above-mentioned platforms −

On UNIX and Linux platform

With the help of following steps, we can install Python on Unix and Linux platform −

- First, go to www.python.org/downloads/.

- Next, click on the link to download zipped source code available for Unix/Linux.
- Now, Download and extract files.
- Next, we can edit the *Modules/Setup* file if we want to customize some options.
    - Next, write the command **run ./configure script**
    - make
    - make install

On Windows platform

With the help of following steps, we can install Python on Windows platform −

- First, go to https://www.python.org/downloads/.
- Next, click on the link for Windows installer python-XYZ.msi file. Here XYZ is the version we wish to install.
- Now, we must run the file that is downloaded. It will take us to the Python install wizard, which is easy to use. Now, accept the default settings and wait until the install is finished.

Why Python for Data Science?

Python is the fifth most important language as well as most popular language for Machine learning and data science. The following are the features of Python that makes it the preferred choice of language for data science −

Extensive set of packages

Python has an extensive and powerful set of packages which are ready to be used in various domains. It also has packages like **numpy, scipy, pandas, scikit-learn** etc. which are required for machine learning and data science.

Easy prototyping

Another important feature of Python that makes it the choice of language for data science is the easy and fast prototyping. This feature is useful for developing new algorithm.

Collaboration feature

The field of data science basically needs good collaboration and Python provides many useful tools that make this extremely.

One language for many domains

A typical data science project includes various domains like data extraction, data manipulation, data analysis, feature extraction, modelling, evaluation, deployment and updating the solution. As Python is a multi-purpose language, it allows the data scientist to address all these domains from a common platform.

Components of Python ML Ecosystem

- **Jupyter Notebook** − Jupyter notebooks basically provides an interactive computational environment for developing Python based Data Science applications.

There are various ML algorithms, techniques and methods that can be used to build models for solving real-life problems by using data. In this chapter, we are going to discuss such different kinds of methods.

Different Types of Methods

The following are various ML methods based on some broad categories −

- **Based on human supervision**
- **Unsupervised Learning**
- **Semi-supervised Learning**
- **Reinforcement Learning**

Based on Generalization Approach

In the learning process, followings are some methods that are based on generalization approaches

Instance based Learning

- Instance based learning method is one of the useful methods that build the ML models by doing generalization based on the input data. It is opposite to the previously studied learning methods in the way that this kind of learning involves ML systems as well as methods that uses the raw data points themselves to draw the outcomes for newer data samples without building an explicit model on training data.
- In simple words, instance-based learning basically starts working by looking at the input data points and then using a similarity metric, it will generalize and predict the new data points.

Model based Learning

- In Model based learning methods, an iterative process takes place on the ML models that are built based on various model parameters, called hyperparameters and in which input data is used to extract the features. In this learning, hyperparameters are optimized based on various model validation techniques. That is why we can say that Model based learning methods uses more traditional ML approach towards generalization.

In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules.

Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries. **Python libraries** that used in Machine Learning are:

- Numpy
- Scipy
- Scikit-learn
- Theano
- TensorFlow
- Keras
- PyTorch
- Pandas
- Matplotlib

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

Skikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

Theano: We all know that Machine Learning is basically mathematics and statistics. Theano is a popular python library that is used to define, evaluate and optimize mathematical expressions involving multi-dimensional arrays in an efficient manner. It is achieved by optimizing the utilization of CPU and GPU. It is extensively used for unit-testing and self-verification to detect and diagnose different types of errors. Theano is a very powerful library that has been used in large-scale computationally intensive scientific projects for a long time but is simple and approachable enough to be used by individuals for their own projects.

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library which is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that supports on Computer Vision, Natural Language Processing (NLP) and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

Matpoltlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc.

EXPERIMENT. No. 1

How to Implement Find-S Algorithm in Machine Learning?

In **Machine Learning,** concept learning can be termed as "*a problem of searching through a predefined space of potential hypothesis for the hypothesis that best fits the training examples"*

What is Find-S Algorithm in Machine Learning?

In order to understand Find-S algorithm, you need to have a basic idea of the following concepts as well:

1. Concept Learning
2. General Hypothesis
3. Specific Hypothesis

1. Concept Learning

Let's try to understand concept learning with a real-life example. Most of human learning is based on past instances or experiences. For example, we are able to identify any type of vehicle based on a certain set of features like make, model, etc., that are defined over a large set of features.

These special features differentiate the set of cars, trucks, etc. from the larger set of vehicles. These features that define the set of cars, trucks, etc. are known as concepts.

Similar to this, machines can also learn from concepts to identify whether an object belongs to a specific category or not. Any algorithm that supports concept learning requires the following:

- Training Data
- Target Concept
- Actual Data Objects

2. General Hypothesis

Hypothesis, in general, is an explanation for something. The general hypothesis basically states the general relationship between the major variables. For example, a general hypothesis for ordering food would be *I want a burger.*

G = { '?', '?', '?', …..'?'}

3. Specific Hypothesis

The specific hypothesis fills in all the important details about the variables given in the general hypothesis. The more specific details into the example given above would be *I want a cheeseburger with a chicken pepperoni filling with a lot of lettuce.*

S = {'Φ','Φ','Φ', ……,'Φ'}

The Find-S algorithm follows the steps written below:

1. Initialize ‘h’ to the most specific hypothesis.
2. The Find-S algorithm only considers the positive examples and eliminates negative examples. For each positive example, the algorithm checks for each attribute in the example. If the attribute value is the same as the hypothesis value, the algorithm moves on without any changes. But if the attribute value is different than the hypothesis value, the algorithm changes it to ‘?’.

How Find-S algorithm works.

1. The process starts with initializing ‘h’ with the most specific hypothesis, generally, it is the first positive example in the data set.
2. We check for each positive example. If the example is negative, we will move on to the next example but if it is a positive example we will consider it for the next step.
3. We will check if each attribute in the example is equal to the hypothesis value.
4. If the value matches, then no changes are made.
5. If the value does not match, the value is changed to ‘?’.
6. We do this until we reach the last positive example in the data set.

Limitations of Find-S Algorithm

There are a few limitations of the Find-S algorithm listed down below:

1. There is no way to determine if the hypothesis is consistent throughout the data.
2. Inconsistent training sets can actually mislead the Find-S algorithm, since it ignores the negative examples.
3. Find-S algorithm does not provide a backtracking technique to determine the best possible changes that could be done to improve the resulting hypothesis.

Now that we are aware of the limitations of the Find-S algorithm, let us take a look at a practical implementation of the Find-S Algorithm.

Implementation of Find-S Algorithm in Python

The concept of this particular problem will be on what days does a person likes to go on walk.

Step 1: Create csv file using given data set

Sample Data (exp1.csv)

Time	**Weather**	**Temperature**	**Company**	**Humidity**	**Wind**	**Goes**
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Looking at the data set, we have six attributes and a final attribute that defines the positive or negative example. In this case, yes is a positive example, which means the person will go for a walk.

So now, the general hypothesis is:

h_0 = {‘Morning’, ‘Sunny’, ‘Warm’, ‘Yes’, ‘Mild’, ‘Strong’}

This is our general hypothesis, and now we will consider each example one by one, but only the positive examples.

h_1= {‘Morning’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

h_2 = {‘?’, ‘Sunny’, ‘?’, ‘Yes’, ‘?’, ‘?’}

We replaced all the different values in the general hypothesis to get a resultant hypothesis. Now that we know how the Find-S algorithm works.

Step 2: Import Libraries

```
1. import pandas as pd

2. import numpy as np
```

Step 3: Read csv File

```
3. d1 = pd.read_csv("exp1.csv")
```

Step 4: Making an array of all the attributes

```
4. d2 = np.array(d1)[:,:-1]
```

Step 5: Segregating the target that has positive and negative examples

```
5. t1 = np.array(d1)[:,-1]
```

Step 6: Function to implement find-s algorithm

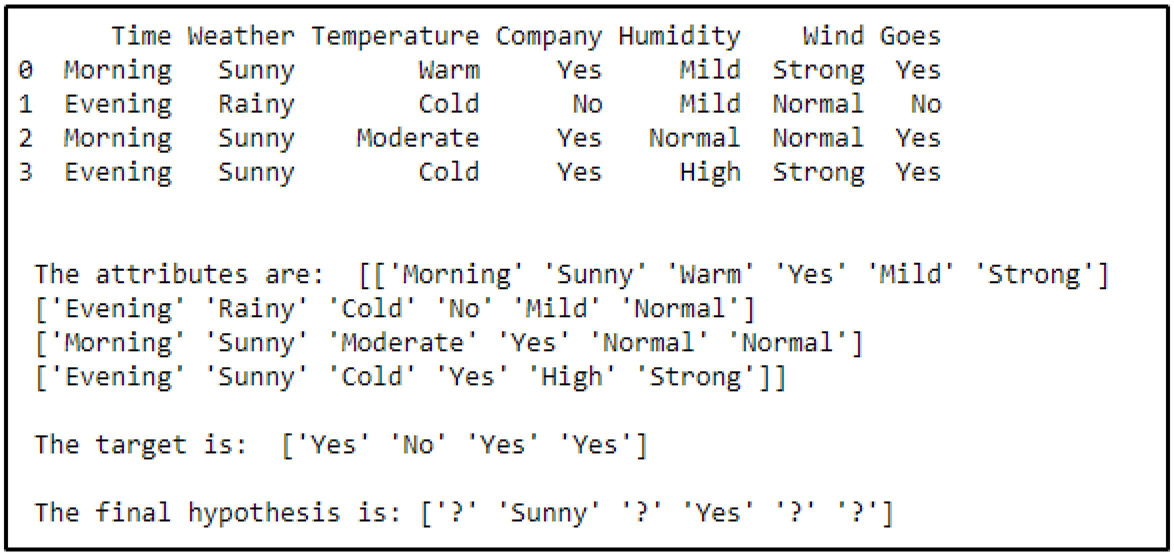
```
6. def finds(c,t):

    7.for i, val in enumerate(t):

        8. if val == "Yes":

           9.spec_hypo = c[i].copy()

           10.break

11.for i, val in enumerate(c):

    12.if t[i] == "Yes":

        13.for x in range(len(spec_hypo)):
```

```
        14. if val[x] != spec_hypo[x]:
      15.spec_hypo[x] = '?'
   16. else:
       17.pass
18.return spec_hypo
```

Step 7: Obtaining the final hypothesis

```
19.print(" The final hypothesis is:",finds(d2,t1))
```

Output:



```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes


 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']

 The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

EXPERIMENT. No. 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.

Candidate Elimination Algorithm:

Candidate-elimination searches an incomplete set of hypotheses (i.e. only a subset of the potentially teachable concepts is included in the hypothesis space). **Candidate-elimination** finds every hypothesis that is consistent with the training data, meaning it searches the hypothesis space completely.

Candidate-Elimination Algorithm performs a bidirectional search in the hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data.

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

```
1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
    1. If d is a positive example
        1. Remove from G any hypothesis inconsistent with d
        2. For each hypothesis s in S that is not consistent with d
            1. Remove s from S
            2. Add to S all minimal generalizations h of s such that
                1. h is consistent with d, and some member of G is more
                   general than h
            3. Remove from S any hypothesis that is more general than another
               hypothesis in S
    2. If d is a negative example
        1. Remove from S any hypothesis inconsistent with d
        2. For each hypothesis g in G that is not consistent with d
            1. Remove g from G
            2. Add to G all minimal specializations h of g such that
                1.h is consistent with d, and some member of S is more specific
                  than h
```

3. Remove from G any hypothesis that is less general than another hypothesis in G

Implementation of Candidate-Elimination Algorithm in Python

Step 1: Create csv file using given data set

Sample Data (exp2.csv)

Row-ID	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

Step 2: Import Libraries

1. import pandas as pd

2. import numpy as np

Step 3: Read csv File

3. d1 = pd.read_csv("exp2.csv")

Step 4: Making an array of all the attributes

4. d2 = np.array(d1)[:,:-1]

Step 5: Segregating the target that has positive and negative examples

5. t1 = np.array(d1)[:,-1]

Step 6: Function to implement candidate-elimination algorithm

6. def candel(c,t):

7. spec_h = c[0].copy()

8. print("initialization of specific_h and general_h")

9. print(spec_h)

10. genl_h = [["?" for i in range(len(spec_h))] for i in range(len(spec_h))]

11. print(genl_h)

```
for i, h in enumerate(c):
    if t[i] == "yes":
        print("If instance is Positive ")
        for x in range(len(spec_h)):
            if h[x]!= spec_h[x]:
                spec_h[x] ='?'
                genl_h[x][x] ='?'
    if t[i] == "no":
.       print("If instance is Negative ")
        for x in range(len(spec_h)):
            if h[x]!= spec_h[x]:
                genl_h[x][x] = spec_h[x]
            else:
                genl_h[x][x] = '?'
    print(spec_h)
    print(genl_h)
indices = [i for i, val in enumerate (genl_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    genl_h.remove (['?', '?', '?', '?', '?', '?'])
return spec_h, genl_h
```

Step 7: Obtaining the final hypothesis

```
s_final, g_final = candel(d2,t1)
print("Final Spec_h:", s_final, sep="\n")
print("Final Genl_h:", g_final, sep="\n")
```

Output:

```
Final Spec_h: ['sunny' 'warm' '?' 'strong' '?' '?']

Final Genl_h: [['sunny', '?', '?', '?', '?', '?'],

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

EXPERIMENT. No. 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.

ID3 is a machine learning **algorithm** for building classification trees developed by Ross Quinlan in/around 1986. The **algorithm** is a greedy, recursive **algorithm** that partitions a data set on the attribute that maximizes information gain.

Decision trees are supervised learning algorithms used for both, classification and regression.

Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning. We can use decision trees for issues where we have continuous but also categorical input and target features.

The main idea of decision trees is to find those descriptive features which contain the most **"information"** regarding the target feature and then split the dataset along the values of these features such that the target feature values for the resulting sub datasets are as pure as possible.

The descriptive feature which leaves the target feature most purely is said to be the most informative one. This process of finding the "most informative" feature is done until we accomplish a stopping criteria where we then finally end up in so called **leaf nodes**. The leaf nodes contain the predictions we will make for new query instances presented to our trained model.

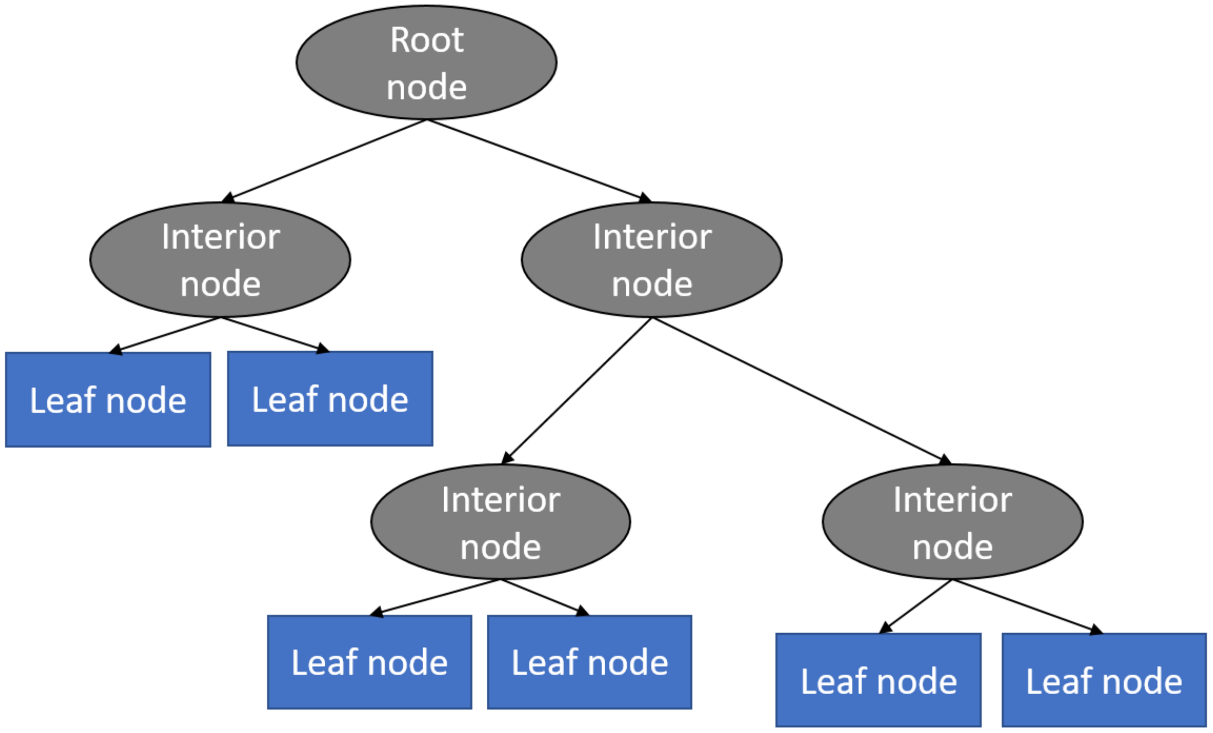
A decision tree mainly contains of a **root node**, **interior nodes**, and **leaf nodes** which are then connected by **branches**.

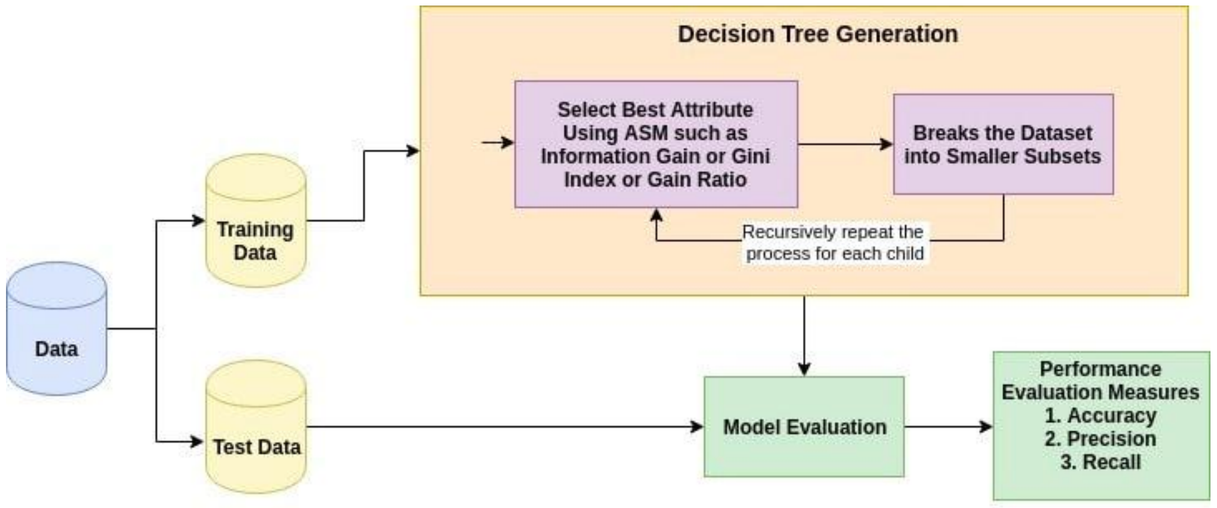
Entropy: It is a fundamental theorem which commonly used in information theory to measure important of information relative to its *size*. Let x is our training set contains positive and negative examples, then the entropy of x relative to this classification is:

$$H(x) = -\ p_+\ log_2\ p_+ - p_-\ log_2\ p_-$$

Information Gain: For training set x and its attribute y, the formula of Information Gain is:

$$G(x, y) = H(x) - \sum_{i \in value(y)} \frac{|\Delta y_i|}{|\Delta y|}\ H(y_i)$$


Root node
Interior node
Interior node
Leaf node
Leaf node
Interior node
Interior node
Leaf node
Leaf node
Leaf node
Leaf node


Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Training Data
Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall

The most notable types of decision tree algorithms are:-

1. Iterative Dichotomiser 3 (ID3): These algorithms uses Information Gain to decide which attribute are to be used classify the current subset of the data. For each level of the tree, information gain is calculated for the remaining data recursively.

2. C4.5: This algorithm is the successor of the ID3 algorithm. This algorithm uses either Information gain or Gain ratio to decide upon the classifying attribute. It is a direct improvement from the ID3 algorithm as it can handle both continuous and missing attribute values.

3. **Classification and Regression Tree (CART):** It is a dynamic learning algorithm which can produce a regression tree as well as a classification tree depending upon the dependent variable.

How does the Decision Tree algorithm work?

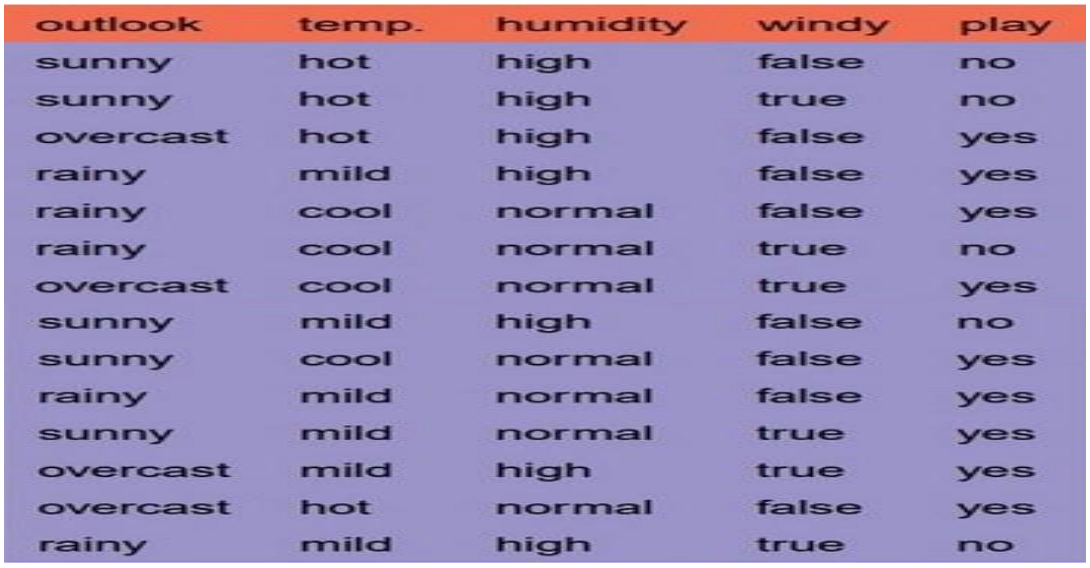
The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures (ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Starts tree building by repeating this process recursively for each child until one of the condition will match:

- All the tuples belong to the same attribute value.
- There are no more remaining attributes.
- There are no more instances.

Decision Tree Algorithm: Example

Data Set



outlook	temp.	humidity	windy	play
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

Entropy of given data set:

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c)$$

C ={yes,no}

Out of 14 instances, 9 are classified as yes, and 5 as no

pyes = —(9/14)*log2(9/14) = 0.41

pno = —(5/14)*log2(5/14) = 0.53

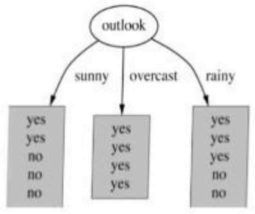
H (S) = pyes + pno = 0.94

Information Gain

For training set x and its attribute y, the formula of Information Gain is:

$$G(x,y) = H(x) - \sum_{i \in value(y)} \frac{|\Delta y_i|}{|\Delta y|} H(y_i)$$

Information Gain for given data set:



E (Outlook=sunny) = $-\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right) = 0.971$

E (Outlook=overcast) = $-1\log(1) - 0\log(0) = 0$

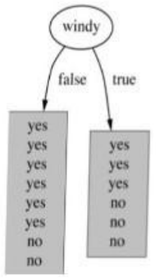
E (Outlook=rainy) = $-\frac{3}{5}\log\left(\frac{3}{5}\right) - \frac{2}{5}\log\left(\frac{2}{5}\right) = 0.971$

} H(S,Outlook)

Average Entropy information for Outlook

I (Outlook) = $\frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.693$ } $\sum_{t \in T} p(t)H(t)$

Gain (Outlook) = E(S) – I (outlook) = 0.94-.693 = 0.247 ⇨ $IG(A,S) = H(S) - \sum_{t \in T} p(t)H(t)$



E (Windy=false) =$-\frac{6}{8}\log\left(\frac{6}{8}\right) - \frac{2}{8}\log\left(\frac{2}{8}\right) = 0.811$

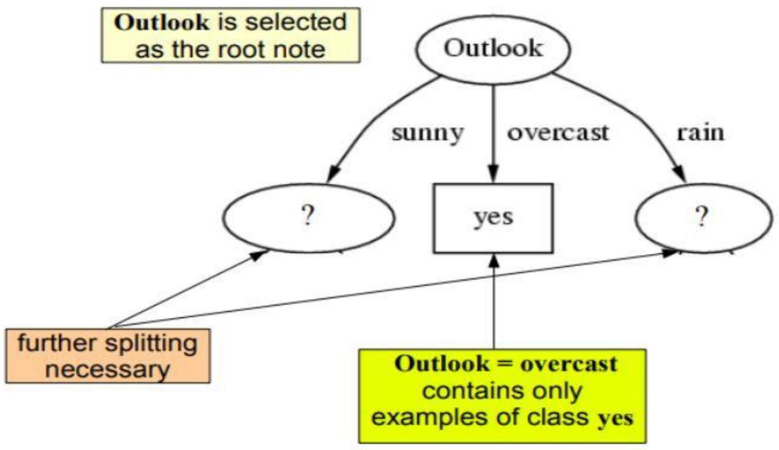
E (Windy=true) = $-\frac{3}{6}\log\left(\frac{3}{6}\right) - \frac{3}{6}\log\left(\frac{3}{6}\right) = 1$

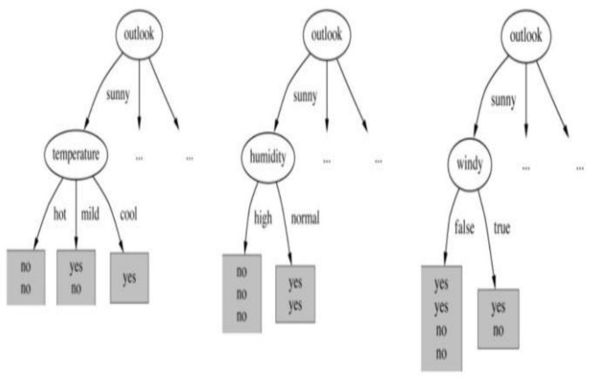
Average entropy information for Windy

I (Windy) = $\frac{8}{14} * 0.811 + \frac{6}{14} * 1 = 0.892$

Gain (Windy) = E(S) – *I* (Windy) = 0.94-0.892=0.048

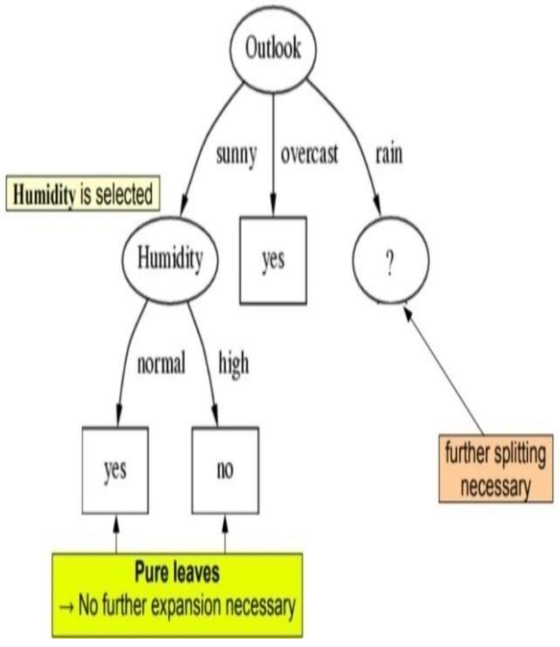
Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	$\boxed{0.247}$	Gain: 0.940-0.911	0.029
Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048


Outlook is selected as the root note
Outlook
sunny
overcast
rain
?
yes
?
further splitting necessary
Outlook = overcast contains only examples of class **yes**



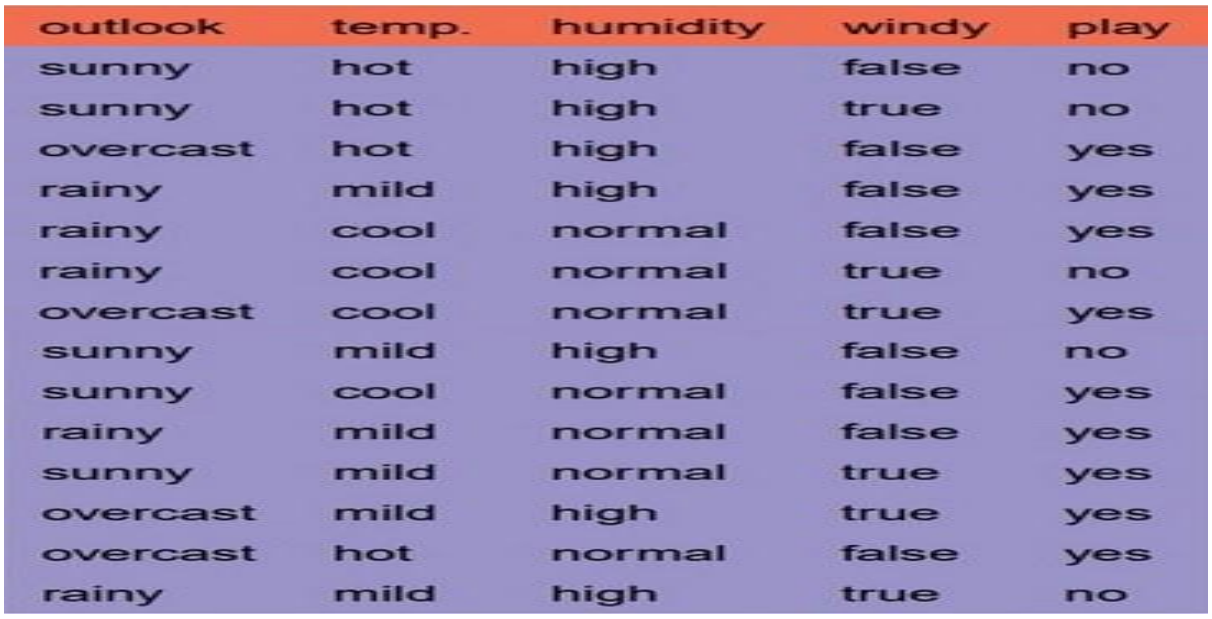
$$\left.\begin{array}{ll} \text{Gain}(\textit{Temperature}) & = 0.571 \text{ bits} \\ \text{Gain}(\textit{Humidity}) & = 0.971 \text{ bits} \\ \text{Gain}(\textit{Windy}) & = 0.020 \text{ bits} \end{array}\right\}$$

Humidity is selected



Implementation of Decision Tree Algorithm (ID3) in Python

Step 1: Create csv file using given data set (exp3.csv)



outlook	temp.	humidity	windy	play
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

Step 2: Import Libraries

```
1. import numpy as np
2. import pandas as pd
3. from sklearn.model_selection import train_test_split
4. from sklearn import tree
5. from sklearn.metrics import accuracy_score
```

Step 3: Read csv file

```
6.d1 = pd.read_csv("exp3.csv")
```

Step 4: Encode attributes in numerical values.

```
7. from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
8. label_encoder = preprocessing.LabelEncoder()

# Encode labels
9. d1['outlook']= label_encoder.fit_transform(d1['outlook'])
10. out_enc=d1['outlook'].unique()
11. print(out_enc)
```

```
12. d1['temp']= label_encoder.fit_transform(d1['temp'])
13. d1['temp'].unique()
14. d1['humidity']= label_encoder.fit_transform(d1['humidity'])
15. d1['humidity'].unique()
16. d1['windy']= label_encoder.fit_transform(d1['windy'])
 17. d1['windy'].unique()
18. d1['play']= label_encoder.fit_transform(d1['play'])
  19. d1['play'].unique()
```

Step 5: Separate dataset into data and target.

```
20. d2 = np.array(d1)[:,:-1]  #data
21. t1 = np.array(d1)[:,-1]     #target
22. print(d2)
23. print(t1)
```

Step 6: Split dataset into training and testing dataset

```
24. X_train, X_test, y_train, y_test = train_test_split(d2,t1, test_size=0.3, random_state=1)
```

Step 7: Build Decision Tree

```
25. model = tree.DecisionTreeClassifier(criterion='entropy', random_state=100 )
26. model.fit(X_train,y_train)
27. y_predict = model.predict(X_test)
28. print(y_predict)
```

Output: 0 0 0 1 0

Step 8: Test accuracy of model (Decision Tree using ID3)

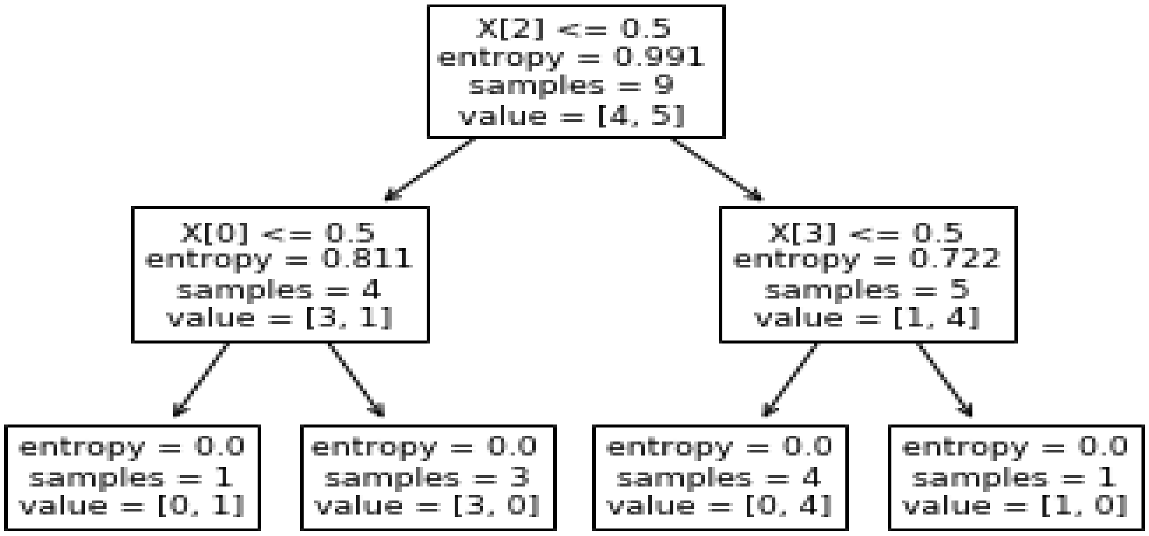
```
29, from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
30. print("Accuracy:",metrics.accuracy_score(y_test, y_predict))
```

Output: 0.4 [40%]

Step 9: Visualize Decision Tree

```
31. import graphviz
32. import pydotplus
33. tree.plot_tree(model)
```

Output:


X[2] <= 0.5
entropy = 0.991
samples = 9
value = [4, 5]
X[0] <= 0.5
entropy = 0.811
samples = 4
value = [3, 1]
X[3] <= 0.5
entropy = 0.722
samples = 5
value = [1, 4]
entropy = 0.0
samples = 1
value = [0, 1]
entropy = 0.0
samples = 3
value = [3, 0]
entropy = 0.0
samples = 4
value = [0, 4]
entropy = 0.0
samples = 1
value = [1, 0]

EXPERIMENT. NO. 4

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

What is Artificial Neural Networks?

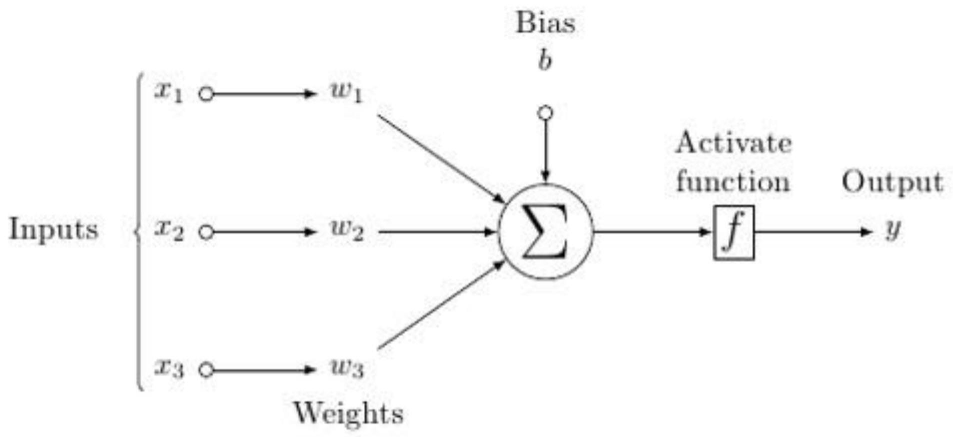
A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.

What is Backpropagation?

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

How Backpropagation Works: Simple Algorithm



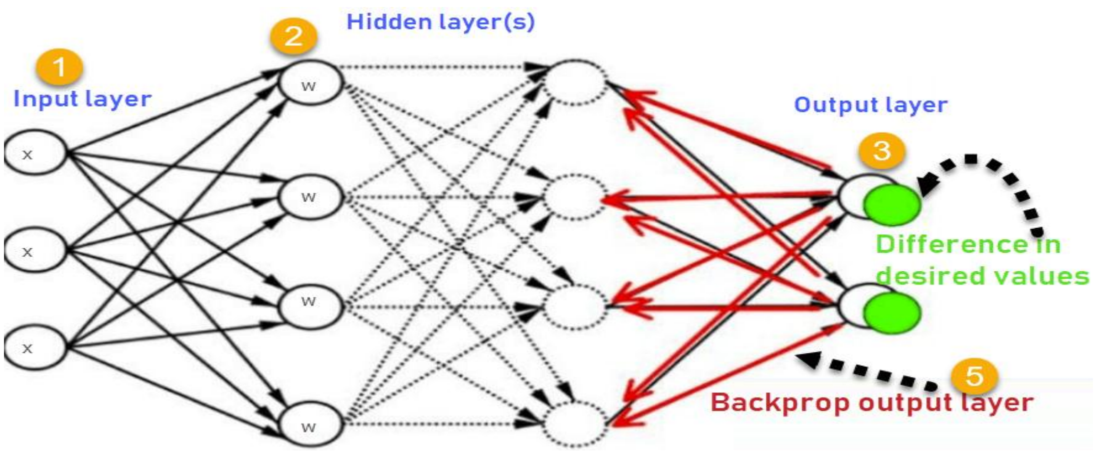
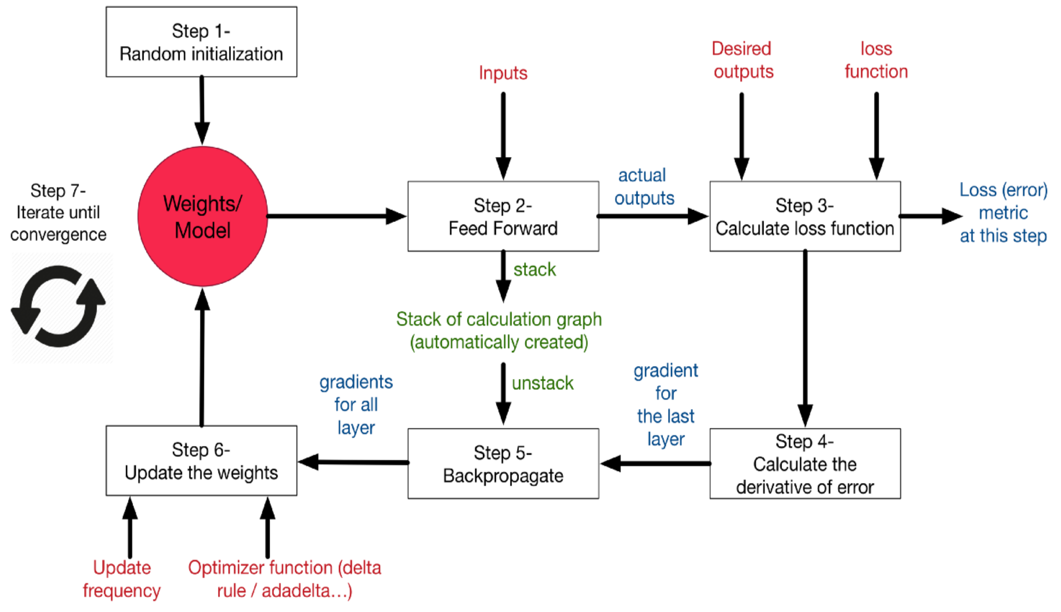


Figure 1: Neural Network

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

   ErrorB= Actual Output – Desired Output

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.



Step 1- Random initialization
Inputs
Desired outputs
loss function
Step 7- Iterate until convergence
Weights/ Model
Step 2- Feed Forward
actual outputs
Step 3- Calculate loss function
Loss (error) metric at this step
stack
Stack of calculation graph (automatically created)
unstack
gradients for all layer
gradient for the last layer
Step 6- Update the weights
Step 5- Backpropagate
Step 4- Calculate the derivative of error
Update frequency
Optimizer function (delta rule / adadelta…)

Figure 2: Learning Process of Neural Network

Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.
Input:
D, a data set consisting of the training tuples and their associated target values;
l, the learning rate;
network, a multilayer feed-forward network.
Output: A trained neural network.
Method:

```
(1) Initialize all weights and biases in network;
(2) while terminating condition is not satisfied
(3)  {
(4) for each training tuple X in D
(5) {
(6) // Propagate the inputs forward:
(7) for each input layer unit j
(8) {
(9) $Oj = Ij$ ; // output of an input unit is its actual input value
(10) for each hidden or output layer unit j
(11) {
(12) $Ij = \sum wijOi + \Theta j$ ; //compute the net input of unit j with respect to the previous layer, i
(13) $Oj = 1/(1+e^{(-Ij)})$ ; } // compute the output of each unit j
(14) // Backpropagate the errors:
(15) for each unit j in the output layer
(16) $Errj = Oj.(1-Oj)(Tj - Oj)$; // compute the error
(17) for each unit j in the hidden layers, from the last to the first hidden layer
(18) $Errj = Oj(1 - Oj)\sum Err_k w_{jk}$ ; // compute the error with respect to the next higher layer, k
(19) for each weight $w_{ij}$ in network
 (20) {
(21) $\Delta w_{ij} = (l)Err_j O_i$ ; // weight increment
(22) $wij = wij + \Delta wij$ ; } // weight update
(23) for each bias $\Theta j$ in network
(24) {
(25) $\Delta \Theta j = (l) Errj$ ; // bias increment
(26) $\Theta j = \Theta j + \Delta \Theta j$ ; } // bias update
(27) }}
```

Implementation of Backpropgation Algorithm in Python

Step 1: Import libraries

1. import numpy as np

Step 2: Initialize Input and Output

2. inp = np.array(([2, 5], [4, 9], [3, 8]), dtype=float) # Inputs

3. aoutp = np.array(([76], [84], [98]), dtype=float) # Actual Output

4. inp = inp/np.amax(inp,axis=0) # Normalize Input Values

5. aoutp = aoutp/100 # Normalize Output values

Step 3: Sigmoid Function

6. def sigmoid (x):

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

Step 4: Derivative of Sigmoid Function

8. def D_sigmoid(x):

   9. return x * (1 - x)

Step 5: Variable Initialization

10. epoch = 2000   #Setting training iterations
11.lr = 0.7   #Setting learning rate
12.iln = 2   #number of features in data set(input layer neurons)
13.hln = 3 #number of hidden layers neurons
14.on = 1 #number of neurons at output layer

#weight and bias initialization

15. wh=np.random.uniform(size=(iln,hln))
16.bh = np.random.uniform(size=(1,hln))
17. wout = np.random.uniform(size=(hln,on))
18. bout = np.random.uniform(size=(1,on))

Step 6: Forward and Backward propagation

19.for i in range(epoch):

#Forward Propogation

   20. hinp1= np.dot(inp,wh)
   21.hinp=hinp1 + bh
   22.hlayer_act = sigmoid(hinp)
   23.outinp1=np.dot(hlayer_act,wout)
   24.outinp= outinp1+ bout
   25.output = sigmoid(outinp)

#Backpropagation

   26.EO = aoutp-output
   27.outgrad = D_sigmoid(output)

28.d_output = EO* outgrad

29.EH = d_output.dot(wout.T)

30.hiddengrad = D_sigmoid(hlayer_act) **#how much hidden layer weights contributed to error**

31. d_hiddenlayer = EH * hiddengrad

32.wout += hlayer_act.T.dot(d_output) *lr **# dotproduct of nextlayererror and currentlayerop**

33.wh += inp.T.dot(d_hiddenlayer) *lr

Step 7: Print Output

34. print("Input: \n" + str(inp))

35.print("Actual Output: \n" + str(aoutp))

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

1. Output: when learning rate =0.7 and number of epochs=2000

```
Input:
[[0.5        0.55555556]
 [1.         1.        ]
 [0.75       0.88888889]]
Actual Output:
[[0.76]
 [0.84]
 [0.98]]
Predicted Output:
 [[0.70648114]
 [0.72133686]
 [0.71535547]]
```

2. Output: when learning rate =0.1 and number of epochs=8000

```
Input:
[[0.5        0.55555556]
 [1.         1.        ]
 [0.75       0.88888889]]
Actual Output:
[[0.76]
 [0.84]
 [0.98]]
Predicted Output:
 [[0.83882493]
 [0.85196553]
 [0.84599087]]
```

3. Output: when learning rate =0.9 and number of epochs=7000

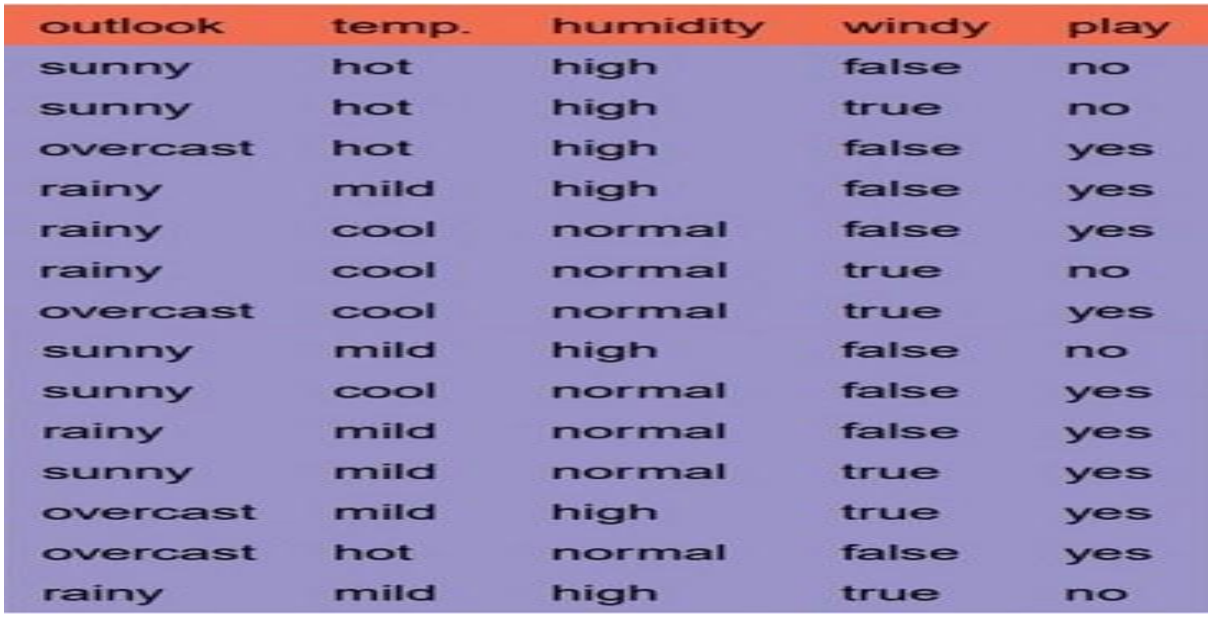
```
Input:
[[0.5        0.55555556]
 [1.         1.        ]
 [0.75       0.88888889]]
Actual Output:
[[0.76]
 [0.84]
 [0.98]]
Predicted Output:
 [[0.68216336]
 [0.69565799]
 [0.69171821]]
```

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.

Implementation of naïve **Bayesian classifier in Python**

Step 1: Create csv file using given data set (exp3.csv)



outlook	temp.	humidity	windy	play
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

Step 2: Import Libraries

```
1. import numpy as np
2. import pandas as pd
3. from sklearn.model_selection import train_test_split
4. from sklearn import tree
5. from sklearn.metrics import accuracy_score
```

Step 3: Read csv file

```
6.d1 = pd.read_csv("exp3.csv")
```

Step 4: Encode attributes in numerical values.

```
7. from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
```

8. label_encoder = preprocessing.LabelEncoder()

# Encode labels
9. d1['outlook']= label_encoder.fit_transform(d1['outlook'])
10. out_enc=d1['outlook'].unique()
11. print(out_enc)
12. d1['temp']= label_encoder.fit_transform(d1['temp'])
13. d1['temp'].unique()
14. d1['humidity']= label_encoder.fit_transform(d1['humidity'])
15. d1['humidity'].unique()
16. d1['windy']= label_encoder.fit_transform(d1['windy'])
17. d1['windy'].unique()
18. d1['play']= label_encoder.fit_transform(d1['play'])
19. d1['play'].unique()

Step 5: Separate dataset into data and target.
20. d2 = np.array(d1)[:,:-1] #data
21. t1 = np.array(d1)[:,-1]   #target
22. print(d2)
23. print(t1)

Step 6: Split dataset into training and testing dataset
24. X_train, X_test, y_train, y_test = train_test_split(d2,t1, test_size=0.3, random_state=1)

Step 7: Naïve Bayes Classifier
25. from sklearn.naive_bayes import GaussianNB
26. gnb = GaussianNB()
27. gnb.fit(X_train, y_train)
28..y_pred = gnb.predict(X_test)
Output: 0 0 1 0 0

Step 8: Test accuracy of model (Naïve Bayes Classifier)
29. from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
30. print("Accuracy:",metrics.accuracy_score(y_test, y_predict))
Output: 0.4 [40%]

EXPERIMENT. NO. 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.

Implementation of naïve **Bayesian Classifier in Python**

Step 1: Create csv file using given data set (exp6.csv):

Sentence	**Label**
I love this sandwich	**Positive**
This is amazing place	**Positive**
I feel very good about these beers	**Positive**
This is my best work	**Positive**
What an awesome view	**Positive**
I do not like this restaurant	**Negative**
I am tired of this stuff	**Negative**
I can't deal with this,	**Negative**
He is my sworn enemy	**Negative**
My boss is horrible	**Negative**
This is an awesome place	**Positive**
I do not like the taste of this juice	**Negative**
I love to dance	**Positive**
I am sick and tired of this place	**Negative**

What a great holiday	**Positive**
That is a bad locality to stay	**Negative**
We will have good fun tomorrow	**Positive**
I went to my enemy's house today	**negative**

Step 2: Import Libraries

1. import numpy as np
2. import pandas as pd
3. from sklearn.model_selection import train_test_split

Step 3: Read csv file

4.d1 = pd.read_csv("exp6.csv")

Step 4: Encode attributes in numerical values.

5. from sklearn import preprocessing
6. label_encoder = preprocessing.LabelEncoder()

# Encode label

7. d1['Label'']= label_encoder.fit_transform(d1['Label'])
8. L_enc=d1['Label'].unique()
9. print(L_enc)

Step 5: Separate dataset into data and target.

10. d2 = np.array(d1)[:,:-1] #data
11. t1 = np.array(d1)[:,-1]   #target
12. print(d2)
13. print(t1)

Step 6: Split dataset into training and testing dataset

14. X_train, X_test, y_train, y_test = train_test_split(d2,t1)

Step 7: O**utput of count vectoriser is a sparse matrix**

15.from sklearn.feature_extraction.text import CountVectorizer

```
16.count_vect = CountVectorizer()

17.X_train_trans = count_vect.fit_transform(X_train)

18.X_test_trans=count_vect.transform(X_test)

19.print(count_vect.get_feature_names())

 20. df=pd.DataFrame(X)train_trans.toarray(),columns=count_vect.get_feature_names())

21. print(df)     #tabular representation

22. print(X_train_trans)   #sparse matrix representation
```

Step 8: Training Naive Bayes (NB) classifier on training data.

```
23. from sklearn.naive_bayes import MultinomialNB

24. clf = MultinomialNB().fit(X_train_trans,y_train)

25. predicted = clf.predict(X_test_trans)
```

Step 8: Printing Accuracy Metrics

```
26. from sklearn import metrics

27. print('Accuracy metrics')

28. print('Accuracy of the classifer is',metrics.accuracy_score(y_test,predicted))

29.print('Confusion matrix')

30. print(metrics.confusion_matrix(y_test,predicted))

31. print('Recall and Precison ')

32. print(metrics.recall_score(y_test,predicted))

33. print(metrics.precision_score(y_test,predicted))
```

Output:

Output of line number 19.

```
['am', 'amazing', 'an', 'and', 'awesome', 'bad', 'best', 'can', 'dance', 'deal', 'do', 'enemy', 'great', 'he', 'holiday',
'house', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sick', 'stay', 'sworn', 'taste',
'that', 'the', 'this', 'tired', 'to', 'today', 'view', 'went', 'what', 'with', 'work']
```

Output of step 8:

```
Accuracy metrics
Accuracy of the classifer is 0.4
Confusion matrix
[[1 1]
 [2 1]]
Recall and Precison
0.3333333333333333
0.5
```

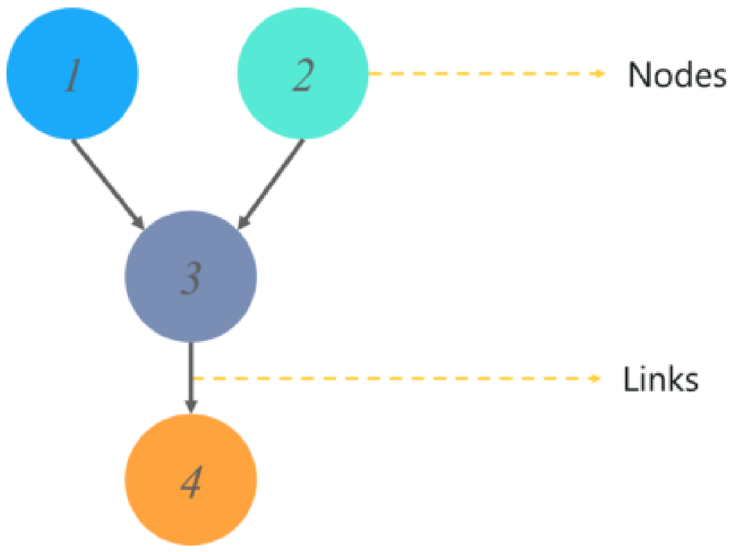
EXPERIMENT. NO. 7

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.

A **Bayesian Network** falls under the category of Probabilistic Graphical Modelling (PGM) technique that is used to compute uncertainties by using the concept of probability. Popularly known as Belief **Networks**, **Bayesian Networks** are used to model uncertainties by using Directed Acyclic Graphs (DAG).

What Is A Directed Acyclic Graph?

A Directed Acyclic Graph is used to represent a Bayesian Network and like any other statistical graph, a DAG contains a set of nodes and links, where the links denote the relationship between the nodes.



The nodes here represent random variables and the edges define the relationship between these variables.

A DAG models the uncertainty of an event occurring based on the *Conditional Probability Distribution* (CDP) of each random variable. A *Conditional Probability Table* (CPT) is used to represent the CPD of each variable in the network.

Joint Probability?

Joint Probability is a statistical measure of two or more events happening at the same time, i.e., P(A, B, C), The probability of event A, B and C occurring. It can be represented as the probability of the intersection two or more events occurring.

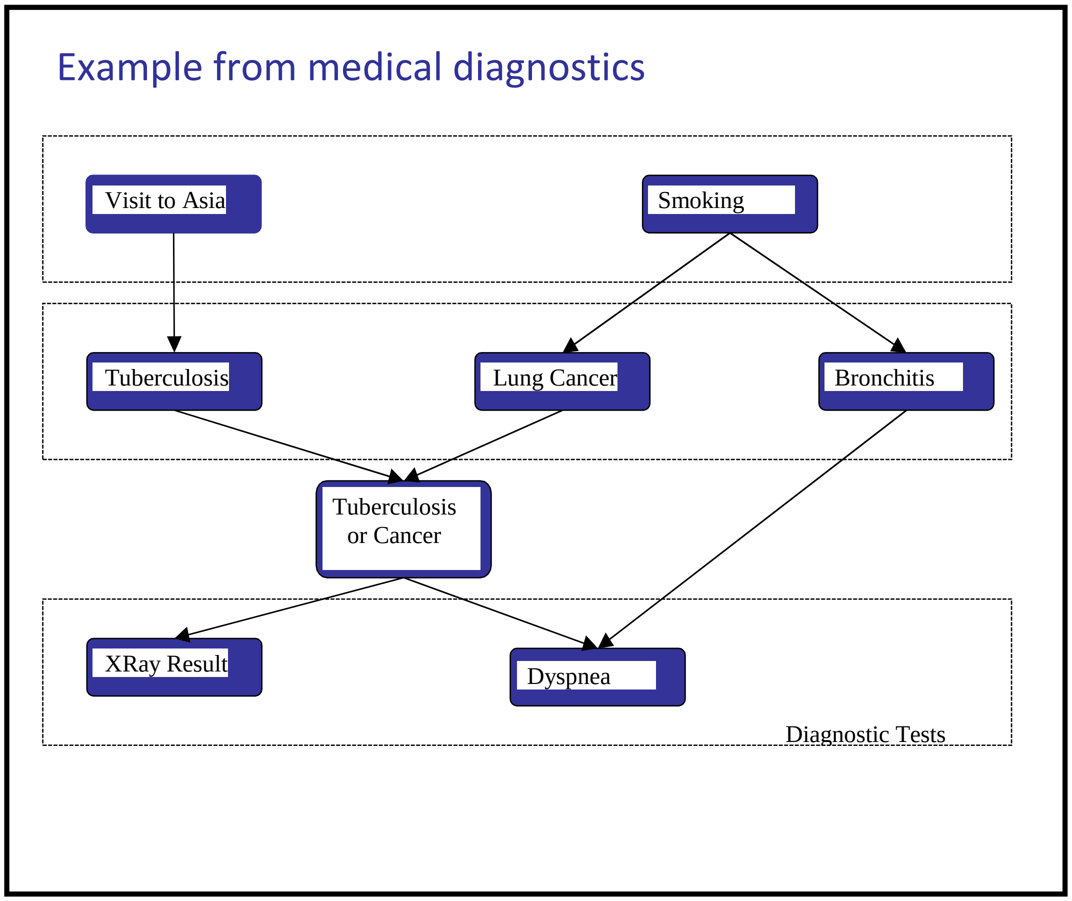
Conditional Probability?

Conditional Probability of an event X is the probability that the event will occur given that an event Y has already occurred.

P(X| Y) is the probability of event X occurring, given that event, Y occurs.

- If X and Y are dependent events then the expression for conditional probability is given by:
  P (X| Y) = P (X and Y) / P (Y)
- If A and B are independent events then the expression for conditional probability is given by:
  P(X| Y) = P (X)

Example Problem



Program to construct a Bayesian network considering medical data.

Dataset: Heart Disease Data set used to implement this program. This Dataset consist 14 attributes as follows

6. Age: in years
7. Sex: (1 = male; 0 = female)
8. Cp :(Chest pain type)
9. Trestbps: (resting blood pressure (in mm Hg on admission to the hospital)
10. Chol :(serum cholestoral in mg/dl)
11. Fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
12. Restecg: (resting electrocardiographic results)
13. Thalach: (maximum heart rate achieved)
14. Exang:(exercise induced angina (1 = yes; 0 = no)
15. Oldpeak: (ST depression induced by exercise relative to rest)
16. Slope: (The slope of the peak exercise ST segment)
17. Ca : (number of major vessels (0-3) colored by flourosopy)
18. Thal : (3 = normal; 6 = fixed defect; 7 = reversable defect)
19. Target : 1 or 0

EXPERIMENT. NO. 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.

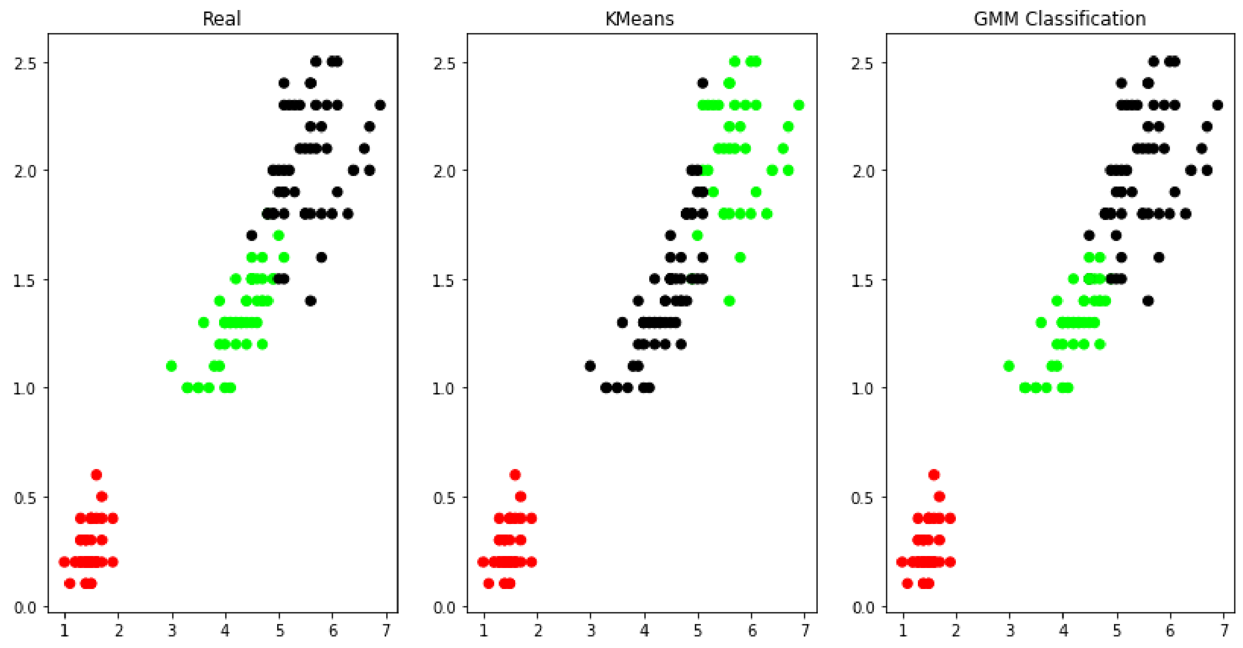
Program:

```
from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load_iris

import sklearn.metrics as sm

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset=load_iris()

# print(dataset)

X=pd.DataFrame(dataset.data)

X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y=pd.DataFrame(dataset.target)

y.columns=['Targets']

# print(X)

plt.figure(figsize=(14,7))

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

# REAL PLOT

plt.subplot(1,3,1)

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)

plt.title('Real')
```

```
# K-PLOT

plt.subplot(1,3,2)

model=KMeans(n_clusters=3)

model.fit(X)

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

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)

plt.title('KMeans')

# GMM PLOT

scaler=preprocessing.StandardScaler()

scaler.fit(X)

xsa=scaler.transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

gmm=GaussianMixture(n_components=3)

gmm.fit(xs)

y_cluster_gmm=gmm.predict(xs)

plt.subplot(1,3,3)

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)

plt.title('GMM Classification')
```

Output:

```
Text(0.5, 1.0, 'GMM Classification')
```



EXPERIMENT. NO. 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.

Introduction

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps −

Step 1: For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2: Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3: For each point in the test data do the following −

3.1 − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

3.2 − Now, based on the distance value, sort them in ascending order.

3.3 − Next, it will choose the top K rows from the sorted array.

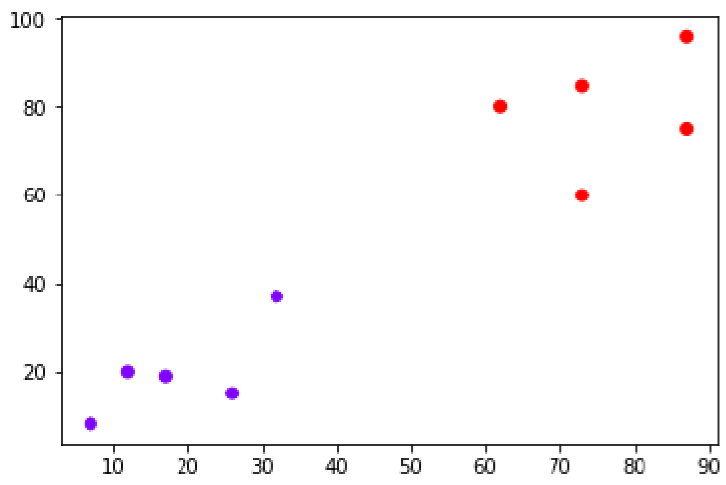
3.4 − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 − End

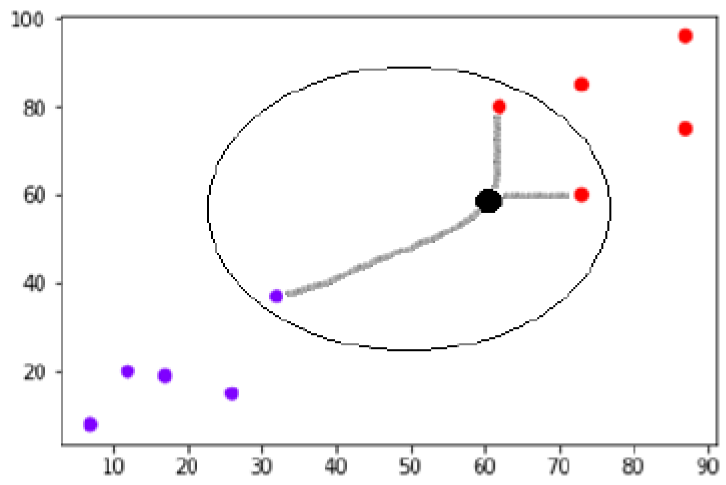
Example

The following is an example to understand the concept of K and working of KNN algorithm

Suppose we have a dataset which can be plotted as follows −



Now, we need to classify new data point with black dot (at point 60, 60) into blue or red class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the next diagram –



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the black dot will also be assigned in red class.

Implementation of K-nearest neighbor (KNN) classifier in Python

Step 1: Import necessary python packages

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import pandas as pd

Step 2: Download the iris dataset from its web link as follows

4. path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Step 3: Assign column names to the dataset as follows

5. headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

Step 5: Read dataset to pandas dataframe as follows

6. dataset = pd.read_csv(path, names = headernames)
7. dataset.head()

Step 6: Data Preprocessing

8. X = dataset.iloc[:, :-1].values
9. y = dataset.iloc[:, 4].values

Step 7: Divide the data into train and test split. Following code will split the dataset into 60% training data and 40% of testing data

10.from sklearn.model_selection import train_test_split
11.X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)

Step 8: Data Scaling

12.from sklearn.preprocessing import StandardScaler
13.scaler = StandardScaler()
14.scaler.fit(X_train)
15.X_train = scaler.transform(X_train)
16.X_test = scaler.transform(X_test)

Step 9:Train the model with the help of KNeighborsClassifier class of sklearn as follows

17.from sklearn.neighbors import KNeighborsClassifier
18.classifier = KNeighborsClassifier(n_neighbors = 8)
19.classifier.fit(X_train, y_train)

Step 10: Make Prediction.

20. y_pred = classifier.predict(X_test)

Step 11: Print Results

21.from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
22.result = confusion_matrix(y_test, y_pred)
23.print("Confusion Matrix:")
24.print(result)
25.result1 = classification_report(y_test, y_pred)
26.print("Classification Report:",)
27.print (result1)
28.result2 = accuracy_score(y_test,y_pred)
29.print("Accuracy:",result2)

Output

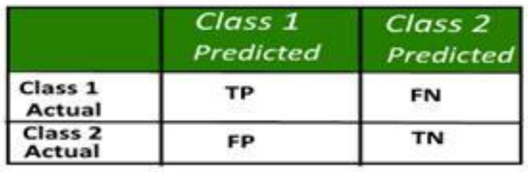
Confusion Matrix:
```
[[21 0 0]
[ 0 16 0]
[ 0 7 16]]
```

Classification Report:

```
                  precision   recall   f1-score   support
    Iris-setosa      1.00       1.00      1.00       21
Iris-versicolor      0.70       1.00      0.82       16
 Iris-virginica      1.00       0.70      0.82       23
      micro avg      0.88       0.88      0.88       60
      macro avg      0.90       0.90      0.88       60
   weighted avg      0.92       0.88      0.88       60
```

Accuracy: 0.88 (88%)

Confusion matrix:



	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

20. **True Positive (TP):** Observation is positive, and is predicted to be positive.
21. **False Negative (FN):** Observation is positive, but is predicted negative. (Also known as a "Type II error.")
22. **True Negative (TN):** Observation is negative, and is predicted to be negative.
23. **False Positive (FP):** Observation is negative, but is predicted positive. (Also known as a "Type I error.")

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textit{Recall} = \frac{TP}{TP + FN}$$

$$\textit{Precision} = \frac{TP}{TP + FP}$$

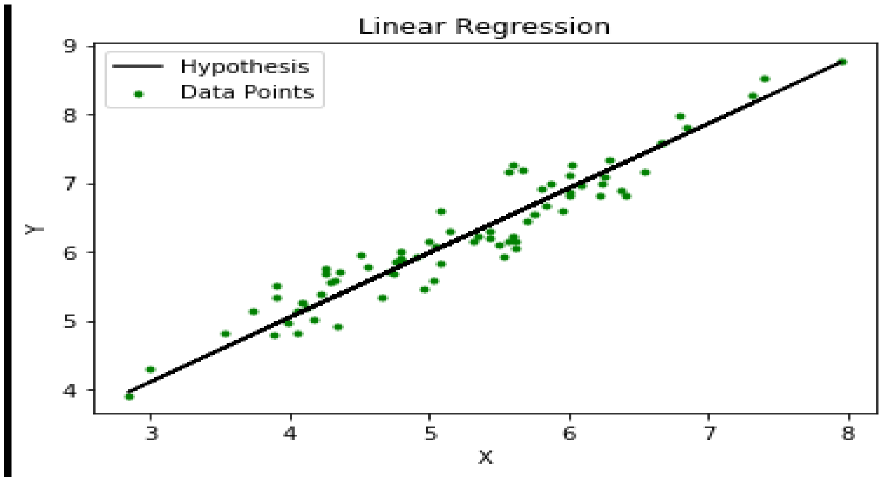
$$\textit{F - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

EXPERIMENT. NO. 10

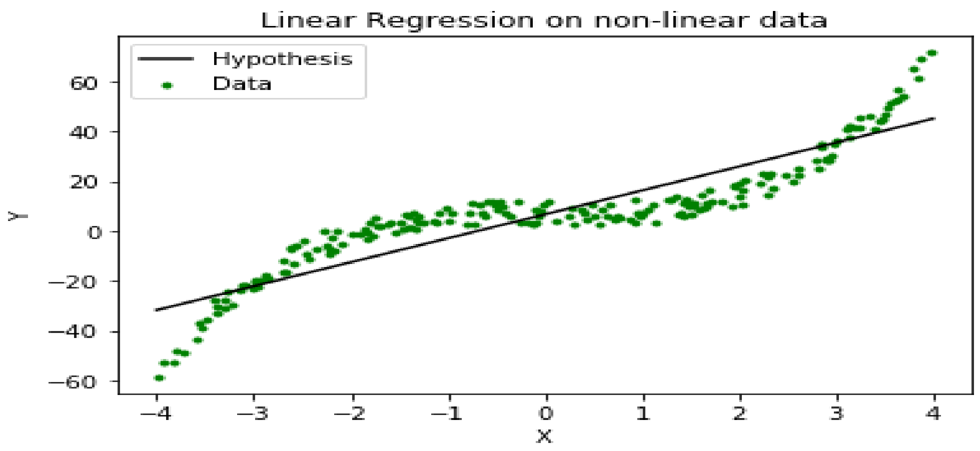
Implement the non-parametric **Locally Weighted Regression** algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Introduction

Linear regression is a supervised learning algorithm used for computing linear relationships between input (X) and output (Y).



As evident from the image below, this algorithm cannot be used for making predictions when there exists a non-linear relationship between X and Y. In such cases, locally weighted linear regression is used.



Locally weighted linear regression is a non-parametric algorithm, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression.

Rather parameters theta (ϴ) is computed individually for each query point $\varkappa$. While computing theta (ϴ), a higher "preference" is given to the points in the training set lying in the vicinity of $\varkappa$ than the points lying far away from $\varkappa$

.The modified cost function is

$$J(\theta) = \sum_{i=1}^{m} w^{\varkappa(i)} \left(\theta^T w^{\varkappa(i)} - y^i\right)^2$$

Where $w^{\varkappa(i)}$ is a non-negative "weight" associated with training point $\varkappa^{(i)}$

For $\varkappa^{(i)}$ s lying closer to the query point $\varkappa$, the value of $w^{\varkappa(i)}$ is large, while for $\varkappa^{(i)}$ s lying far away from $\varkappa$ the value of $w^{\varkappa(i)}$ is small.

The typical choice of $w^{\varkappa(i)}$ is:

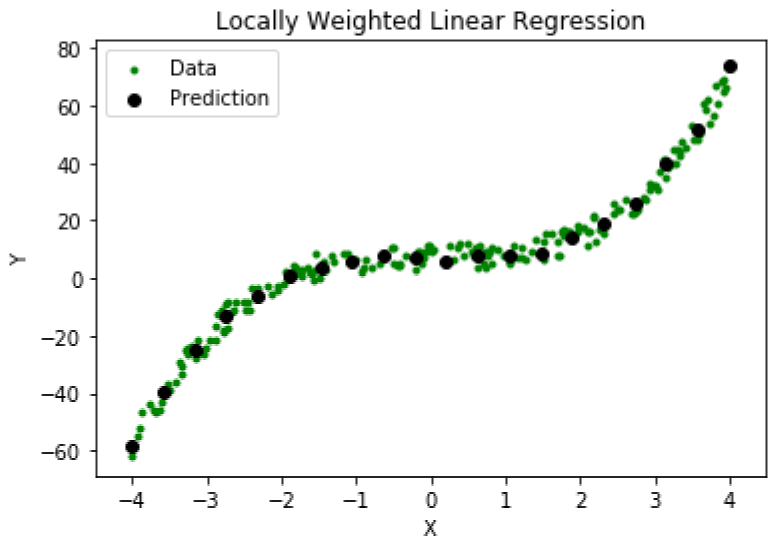
$$w^{\varkappa(i)} = \exp\left(\frac{-\left(\varkappa^{(i)} - \varkappa\right)^2}{2\Gamma^2}\right)$$

Where, **tau** is called the bandwidth parameter and controls the rate at which $w^{\varkappa(i)}$ falls with distance from $\varkappa$

Clearly if | $\varkappa^{(i)} - \varkappa$ | is small $w^{\varkappa(i)}$ is close to 1 and if | $\varkappa^{(i)} - \varkappa$ | is large $w^{\varkappa(i)}$ is close to 0.

Thus, the training-set-points lying closer to the query point $\varkappa$ contribute more to the cost

J (ϴ) than the points lying far away from $\varkappa$.



Steps involved in locally weighted linear regression are:

1. Compute ϴ to minimize the cost:

$$J(\theta) = \sum_{i=1}^{m} w^{\varkappa(i)} \left(\theta^T w^{\varkappa(i)} - y^i\right)^2$$

2. Predict output for given query point $\varkappa$.
3. Return: $\Theta^T \varkappa$

Locally Weighted Regression

Model-based methods, such as neural networks and the mixture of Gaussians, use the data to build a parameterized model. After training, the model is used for predictions and the data are generally discarded. In contrast, ``memory-based" methods are non-parametric approaches that explicitly retain the training data, and use it each time a prediction needs to be made. Locally weighted regression (LWR) is a memory-based method that performs a regression around a point of interest using only training data that are ``local" to that point.

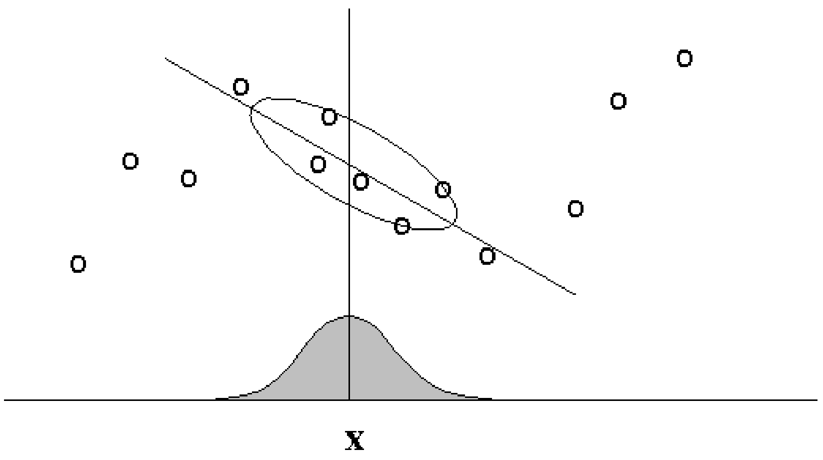


Figure 2: In locally weighted regression, points are weighted by proximity to the current **x** in question using a kernel. A regression is then computed using the weighted points.

Implementation of Locally Weighted Regression algorithm in Python

Step 1: Import necessary python packages

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
```

Step 2: Write Kernel Function

```
def KER(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights
```

Step 3: Write Local Weight Function

```
def LW(point,xmat,ymat,k):
    weight = KER(point,xmat,k)
    W=(X.T*(weight*X)).I*(X.T*(weight*ymat.T))
    return W
```

Step 4: Write Local Weight Regression function to implement LWR algorithm

```
def LWR(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*LW(xmat[i],xmat,ymat,k)
    return ypred
```

Step 5: Download the Tips dataset from its web link as follows

```
path = "https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/tips.csv"
```

Step 6: Assign column names to the dataset as follows

```
headernames = ['total_bill','tip','sex','smoker','day','time','size']
```

Step 7: Read dataset to pandas dataframe as follows

```
dataset = pd.read_csv(path, names = headernames)
```

Step 8: Data Preprocessing

```
bill = np1.array(dataset['total_bill'])
tip = np1.array(dataset['tip'])
bill1=bill[1:255]
tip1=tip[1:255]
bill2 = bill1.astype(np1.float) # covert str to float
tip2 = tip1.astype(np1.float) # convert str to float
```

Step 9: Main Function

```
#preparing and add 1 in bill
mbill = np1.mat(bill2)
mtip = np1.mat(tip2)
m= np1.shape(mbill)[1]
```

35. one = np1.mat(np1.ones(m))

36. X= np1.hstack((one.T,mbill.T))

#set k here

37. ypred = LWR(X,mtip,2)

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

39. xsort = X[SortIndex][:,0]

Step 10: Plot Graph

40. fig = plt.figure()

41. ax = fig.add_subplot(1,1,1)

42. ax.scatter(bill2,tip2, color='green')

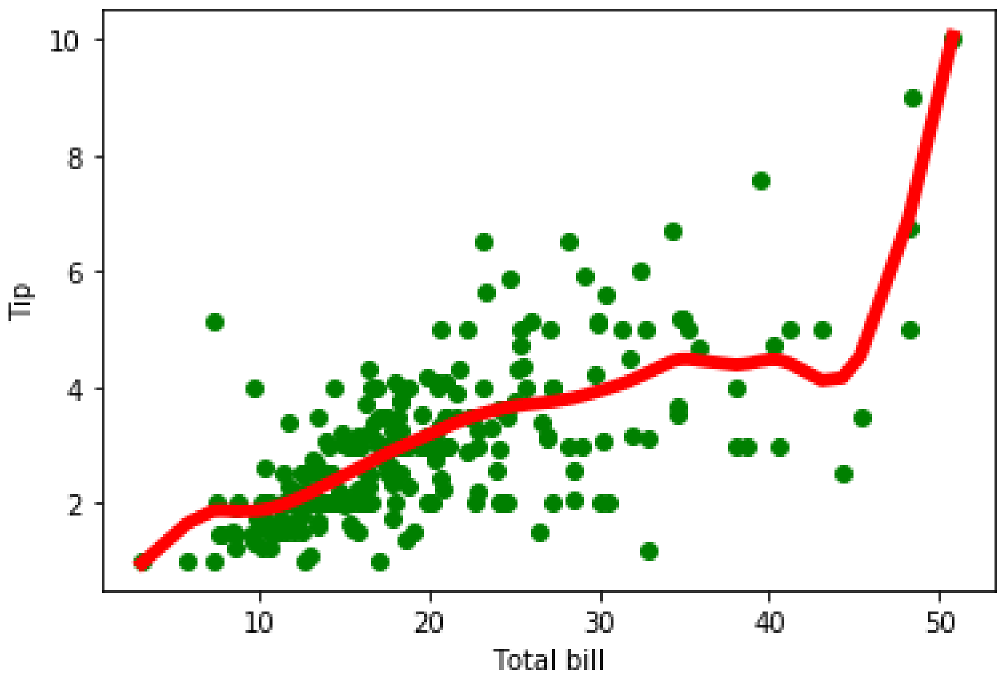
43.ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

44.plt.xlabel('Total bill')

45.plt.ylabel('Tip')

46. plt.show();

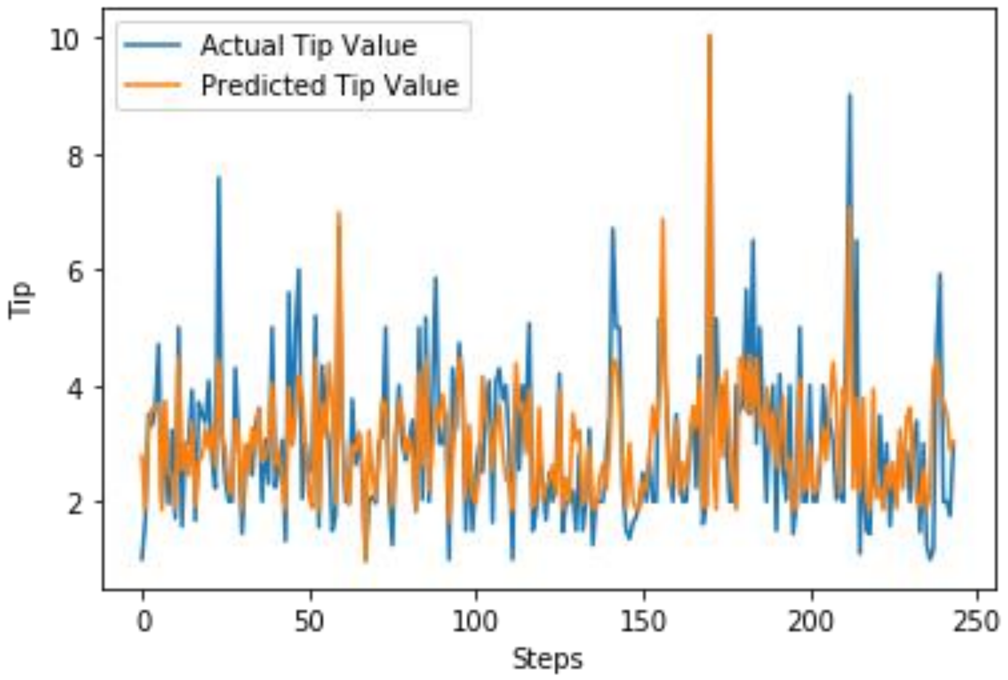
Output:



Step 11: Plot Graph between actual tip value and predicted tip value

```
47. import matplotlib.pyplot as plt
48. z1=np1.arange(244)
49. ax.scatter(bill2,tip2, color='green')
50. plt.plot(z1,tip2,label='Actual Tip Value')
51.plt.plot(z1,ypred,label='Predicted Tip Value')
52. plt.xlabel('Steps')
53. plt.ylabel('Tip')
54. plt.legend()
55. plt.show();
```

Output:


Actual Tip Value
Predicted Tip Value
Tip
Steps