Dr. D. Y. Patil Pratishthan's

DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT & RESEARCH

Approved by A.I.C.T.E, New Delhi , Maharashtra State Government, Affiliated to Savitribai Phule Pune University
Sector No. 29, PCNTDA , Nigidi Pradhikaran, Akurdi, Pune 411044. Phone: 020–27654470, Fax: 020-27656566
Website :www.dypiemr.ac.in Email : principal.dypiemr@gmail.com

Department of Artificial Intelligence and Data Science

LAB MANUAL
Computer Laboratory-II
(BE)
Semester I

Prepared by:
Ms. Arti Singh
Mrs. Sneha Kanawade
Ms. Ashwini Dhumal

Computer Laboratory -II

Course Code	Course Name	Teaching Scheme (Hrs./ Week)	Credits
417526	Computer Laboratory-II: Quantum AI	4	2
417526	Computer Laboratory-II: Information Retrieval	4	2

Course Objectives:

- To develop real-world problem-solving ability
- To enable the student to apply AI techniques in applications that involve perception, reasoning, and planning
- To work in a team to build industry-compliant Quantum AI applications

Course Outcomes:

On completion of the course, learner will be able to–

- CO1: Evaluate and apply core knowledge of Quantum AI to various real-world problems.
- CO2: Illustrate and demonstrate Quantum AI tools for different dynamic applications.

Operating System recommended: Practical can be performed on suitable development platform

Course Objectives:

- Understand the concepts of information retrieval and web mining
- Understand information retrieval process using standards available tools

Course Outcomes:

On completion of the course, learner will be able to–

- CO1: Apply various tools and techniques for information retrieval and web mining
- CO2: Evaluate and analyze retrieved information

Operating System recommended: 64-bit Open source Linux or its derivative

Table of Contents

Lab Assignment No.	1
Title	Implementations of 16 Qubit Random Number Generator
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II :Quantum AI
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 01

Title: Implementations of 16 Qubit Random Number Generator

Problem Statement: Implementations of 16 Qubit Random Number Generator

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to build 16 Qubit Random Number Generator

Outcomes:

After completion of this assignment students are able to understand how to build 16 Qubit Random Number Generator.

Theory: A 16-qubit random number generator is a device or system that utilizes a quantum computer's 16 qubits (quantum bits) to generate random numbers. Unlike classical computers that rely on deterministic algorithms to generate pseudo-random numbers, quantum computers leverage the inherent uncertainty and superposition properties of quantum states to create genuinely unpredictable outcomes.

In a 16-qubit random number generator, the qubits are prepared in specific quantum states that undergo controlled operations, leading to an entangled quantum state. The final measurement of these qubits in their entangled state produces a sequence of random bits that are the result of quantum effects. Due to the probabilistic nature of quantum measurements, the outcome cannot be predicted with certainty, ensuring the generated numbers are truly random.

These quantum random number generators have applications in various fields such as cryptography, secure communications, simulations, and scientific research, where high-quality random numbers are crucial for ensuring security and enhancing the performance of certain algorithms.

A 16-qubit random number generator uses quantum bits to create truly unpredictable numbers. Its applications include:

Super Secure Codes: Generates keys for ultra-safe encryption.

Unhackable Messages: Makes sure messages stay private in communication.

Better Guesses: Improves computer predictions and simulations.

Fair Selection: Helps pick nodes fairly in blockchain networks.

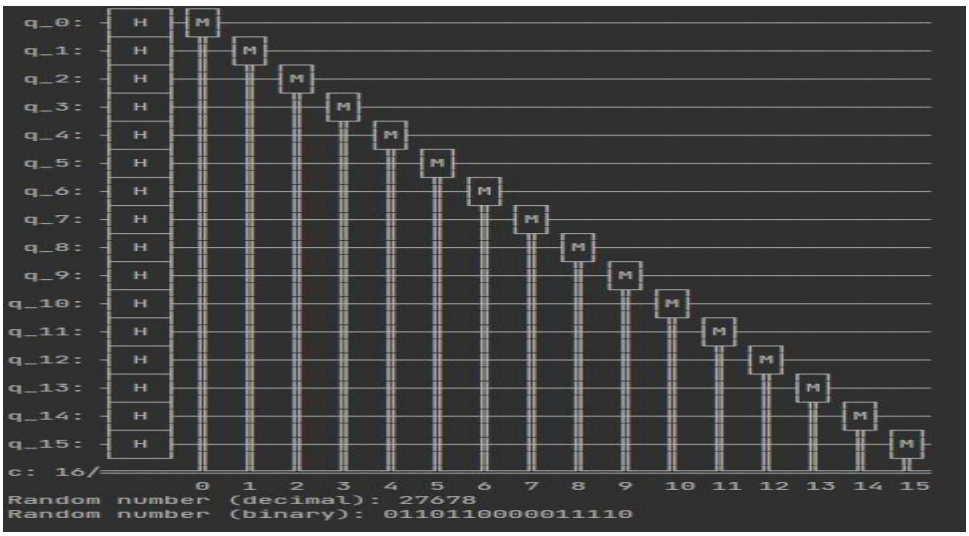
Smarter AI Learning: Adds randomness to help AI learn better.

Cool Creative Stuff: Creates unique art and surprises for entertainment.

Steps:

- Step 1: Initialize the quantum and classical registers
- Step 2: Create the circuit
- Step 3: Apply a Hadamard gate to all qubits
- Step 4: Measure the qubits

Output:



Conclusion: I have understood how to generate 16 Qubit Random Number Generator.

Lab Assignment No.	02
Title	Tackle Noise with Error Correction
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II: Quantum AI
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 02

Title: Tackle Noise with Error Correction

Problem Statement: Tackle Noise with Error Correction

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Tackle Noise with Error Correction

Outcomes:

After completion of this assignment students are able to understand how to Tackle Noise with Error Correction

Theory: Tackling noise with error correction is a crucial concept in quantum computing to ensure the reliability and accuracy of quantum computations. Here's a brief description:

In a quantum computer, quantum bits (qubits) are susceptible to errors due to external factors like temperature fluctuations or electromagnetic interference. These errors can disrupt the delicate quantum states required for computation. Error correction involves using additional qubits and quantum operations to detect and rectify errors, maintaining the integrity of quantum information.

Quantum error correction codes encode the logical qubits across multiple physical qubits in a way that allows errors to be detected and corrected without directly measuring the quantum state. This is achieved through carefully designed quantum gates that manipulate the qubits and enable error detection. If an error is detected, the information can be recovered through quantum operations, ensuring the correct outcome of the computation.

Error correction techniques, such as the surface code or the stabilizer codes, help extend the lifespan of quantum information and enable quantum computers to perform complex computations with high accuracy. However, error correction comes at the cost of requiring additional qubits and more intricate quantum operations, which poses a challenge in terms of hardware and computational resources.

A brief description of how error correction helps tackle noise in quantum computing, presented in bullet points:

Error Vulnerability: Quantum computers use delicate quantum states (qubits) that are sensitive to external factors, leading to errors in computations.

Quantum Error Correction: Error correction is a technique to mitigate errors by encoding qubits across multiple physical qubits in a way that allows errors to be detected and corrected.

Error Detection: Quantum error correction codes involve measuring specific properties of the encoded qubits without directly measuring the fragile quantum state.

Tackling noise with error correction in quantum computing offers several key advantages that are pivotal for the reliable and practical implementation of quantum technologies. **Here are its advantages:**

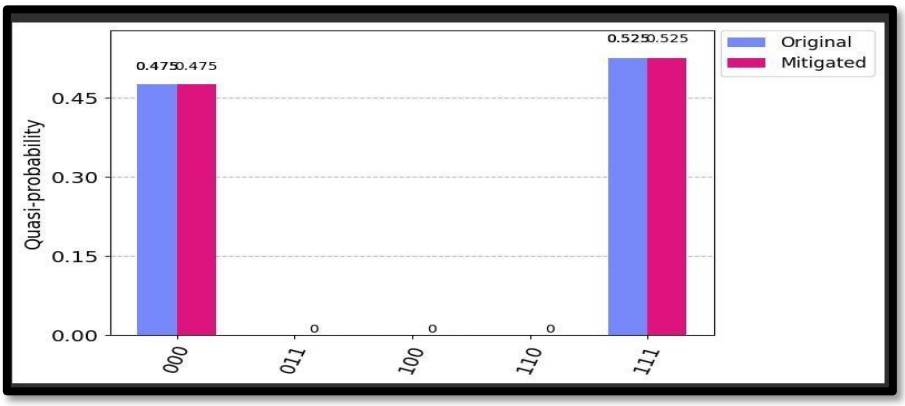
Enhanced Reliability: Error correction enables quantum computations to remain accurate and reliable even in the presence of noisy and error-prone quantum hardware.

Extended Qubit Lifespan: By actively identifying and correcting errors, error correction helps maintain the coherence and stability of quantum states, prolonging the effective lifespan of qubits.

Higher-Quality Results: The use of error correction ensures that the outcomes of quantum computations are closer to the desired results, minimizing the impact of errors on the f inal output.

Steps-

1. Identify the noisy channel.
2. Choose an error correction technique.
3. Implement error detection and correction.
4. Integrate into your system.
5. Test and optimize.
6. Monitor and maintain.
7. **OUTPUT:**



Conclusion: I have understood how to Tackle Noise with Error Correction

Lab Assignment No.	03
Title	Implement Quantum Teleportation algorithm in Python.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II: : Quantum AI
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 03

Title: Implement Quantum Teleportation algorithm in Python.

Problem Statement: Implement Quantum Teleportation algorithm in Python.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Implement Quantum Teleportation algorithm in Python.

Outcomes:

After completion of this assignment students are able to understand Implement Quantum Teleportation algorithm in Python.

Theory: Quantum teleportation is a fundamental concept in quantum mechanics that allows the transfer of quantum information from one location to another without the physical transfer of particles. It's important to note that this process doesn't involve "teleporting" matter in the way it's often depicted in science fiction; rather, it's a transfer of quantum states between particles. Here's a detailed explanation of quantum teleportation:

Entanglement and Quantum States:

Entanglement is a phenomenon in quantum mechanics where two or more particles become correlated in such a way that the state of one particle is dependent on the state of another, even when they are separated by large distances.

Quantum states, such as the spin or polarization of particles, can be in superposition, meaning they exist in a combination of multiple states simultaneously.

Principle of Quantum Teleportation:

Quantum teleportation involves transferring the complete quantum state of one particle (the "sender" or "Alice's" qubit) to another distant particle (the "receiver" or "Bob's" qubit) through entanglement and classical communication.

The sender and receiver particles are entangled beforehand, usually using a process like the Bell state measurement.

Teleportation Process:

Assume Alice has a qubit in an unknown state she wants to teleport to Bob.

Alice and Bob share an entangled pair of qubits. This shared entanglement serves as the "quantum channel" for teleportation.

Alice performs a joint measurement (Bell measurement) on her qubit and the qubit she wants to teleport. This measurement collapses both qubits into one of four Bell states.

Classical Communication:

Alice sends the result of her Bell measurement to Bob using classical communication. This result consis ts of two classical bits of information.

Conditional Operations by Bob:

Based on the information received from Alice, Bob applies specific quantum gates to his qubit to transform it into the desired state.

Bob's qubit now holds the quantum state that was initiallyon Alice's qubit. The state has effectively "teleported" from Alice to Bob.

Properties and Implications:

Quantum teleportation ensures the transfer of the exact quantum state, including its superposition and entanglement properties.

It's important to note that the process involves destroying the original state on Alice's qubit. The no-cloning theorem of quantum mechanics prevents exact copying of an arbitrary quantum state.

Applications:

Quantum teleportation is a crucial building block for various quantum communication and quantum networking protocols.

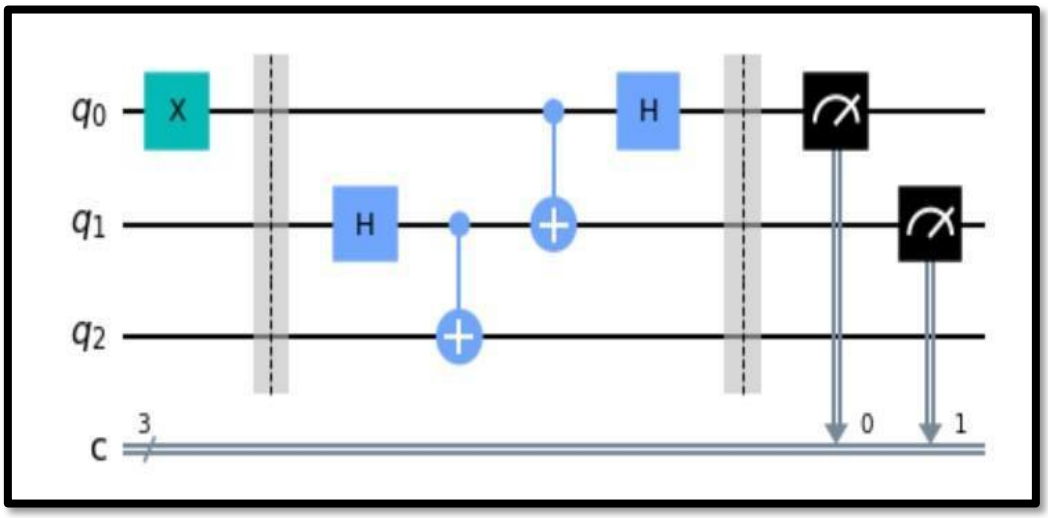
It's used in quantum cryptography for secure key distribution, quantum teleportation-based quantum repeaters for long-distance quantum communication, and potentially in future quantum computing archi tectures.

Quantum teleportation showcases the non-intuitive and unique aspects of quantum mechanics, demonstrating the entanglement and superposition properties that distinguish quantum systems from classical ones.

Steps-

1. Import the necessary libraries (Quantum Circuit, Aer, execute, plot_bloch_multivector, and plot_histogram).
2. Create a quantum circuit with three qubits and three classical bits.
3. Prepare the initial state by applying gates to create entanglement.
4. Perform the teleportation protocol by applying gates and measurements.

Output:



Conclusion: I have understood the implementation of Quantum Teleportation algorithm in Python.

Lab Assignment No.	04
Title	The Randomized Benchmarking Protocol
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II: : Quantum AI
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 04

Title: The Randomized Benchmarking Protocol.

Problem Statement: The Randomized Benchmarking Protocol

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Implement the Randomized Benchmarking Protocol

Outcomes:

After completion of this assignment students are able to understand the Randomized Benchmarking Protocol

Theory: Randomized Benchmarking Protocol (RB) is a technique used in quantum information science to assess the quality of quantum gates and operations in a quantum computing system. Quantum gates are the fundamental building blocks of quantum circuits, and their accurate implementation is crucial for the reliable execution of quantum algorithms. RB provides a standardized way to quantify the error rates of these gates and to characterize the overall performance of a quantum processor.

Basic Idea: RB involves applying a sequence of random gate operations (also known as "cliffords") to a quantum state, followed by an inverse sequence of gates to return the state to its original form. The error accumulation during this process provides insight into the overall gate quality.

Randomized Operations (Clifford Gates):

RB employs sequences of random Clifford gates, which are a well-defined set of quantum gates with known mathematical properties. Clifford gates are chosen because they are easilyimplementable and form a universal set, meaning they can be combined to approximate any quantum operation.

Procedure:

The RB procedure can be broken down into the following steps:

Choose a set of Clifford gates to use in the protocol.

Create a random sequence of Clifford gates, also known as a "randomized sequence."

Apply the randomized sequence to a specific initial quantum state, often the logical zero state (|0⟩).

Apply the inverse of the randomized sequence to the resulting quantum state to attempt to return it to the initial state.

Advantages:

RB has several advantages, including:

It is relatively robust to certain types of errors, making it suitable for characterizing gate quality in noisy quantum systems.

It provides a standardized metric for comparing the performance of gates across different quantum computing platforms.

It is a practical tool for identifying areas of improvement in gate operations and guiding error correction strategies.

12. Limitations:

RB has some limitations:

It assumes that errors are largely independent and do not exhibit strong correlations over the gate sequence.

It only provides information about the average gate fidelity and doesn't capture the entire error spectrum.

13. Applications:

RB is widely used by researchers, engineers, and practitioners in the quantum computing field for:

Benchmarking and validating quantum hardware by quantifying gate performance.

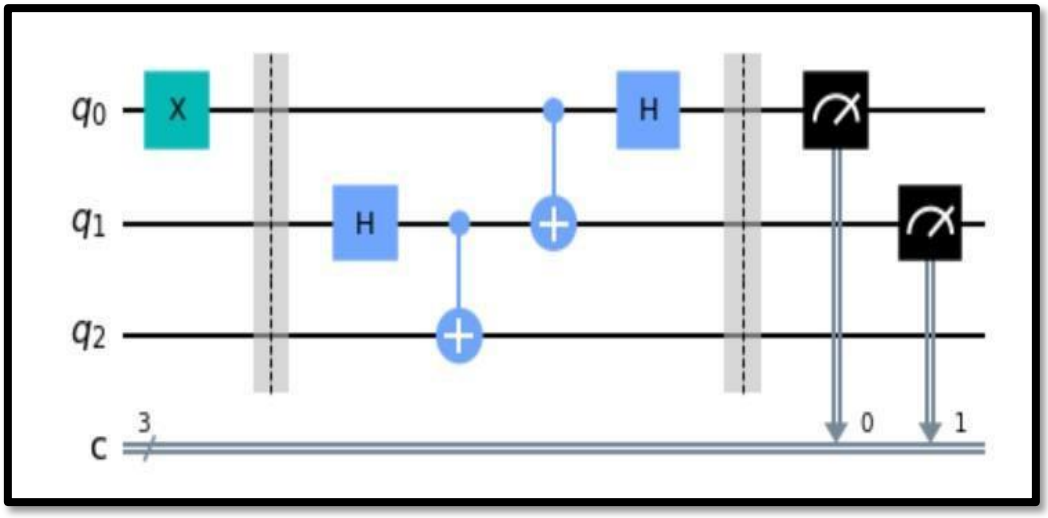
Identifying areas for gate operation enhancement and optimization.

Guiding error mitigation and error correction efforts.

In summary, the Randomized Benchmarking Protocol is a powerful technique for assessing the quality of quantum gates by subjecting them to randomized gate sequences and analyzing the decay in fidelity. It offers insights into the average error rates of quantum operations and plays a crucial role in characterizing and improving the performance of quantum computing systems.

- Steps-
- Import the necessary libraries (qiskit, numpy, matplotlib, etc.).
- Define a benchmarking sequence of random Clifford gates.
- Create benchmarking circuits by applying the sequence to qubits.
- Execute the circuits on a quantum simulator or device.
- Analyze the measurement data to calculate fidelity and error rates.
- Visualize the results by plotting the fidelity decay curve.

Output:



Conclusion: I have understood working of the Randomized Benchmarking Protocol.

Lab Assignment No.	05
Title	Implementing a 5 qubit Quantum Fourier Transform
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II: : Quantum AI
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 05

Title: Implementing a 5 qubit Quantum Fourier Transform

Problem Statement: Implementing a 5 qubit Quantum Fourier Transform

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Implement 5 qubit Quantum Fourier Transform

Outcomes:

After completion of this assignment students are able to understand the Randomized Benchmarking Protocol

Theory: The Quantum Fourier Transform (QFT) is a fundamental operation in quantum computing that plays a crucial role in many quantum algorithms, including Shor's algorithm for integer factorization and quantum phase estimation. The QFT essentially performs a discrete Fourier transform on the amplitudes of a quantum state, leading to a change in the basis of the state.

Steps of performing a 5-qubit Quantum Fourier Transform:

1. Initial State:

Start with a 5-qubit quantum register initialized in the computational basis state $|0\rangle$. This means that all qubits are in the state $|0\rangle$.

$|00000\rangle$

2. Apply Hadamard Gates:

Apply a Hadamard gate to each qubit. The Hadamard gate creates a superposition of the basis states.

$(H \otimes H \otimes H \otimes H \otimes H) |00000\rangle = (1/\sqrt{32}) \sum x=0^{31} |x\rangle$

Here, $\sum x=0^{31}$ indicates a sum over all possible 5-bit binary numbers.

3. Apply Controlled Rotations:

Apply controlled-phase rotations to create the quantum interference necessary for the Fourier transformation. This involves applying controlled Rk gates, where k is determined by the qubit index and rotation angle. For a 5-qubit QFT, the rotations are defined as follows:

C-R1: Rotate qubit 1 by $\pi/2$

C-R2: Rotate qubit 2 by $\pi/4$

C-R3: Rotate qubit 3 by $\pi/8$

C-R4: Rotate qubit 4 by $\pi/16$

4. Swap Qubits:

Perform a series of controlled-swap operations (also known as swap gates) to rearrange the qubits. Swapping qubits is necessary to achieve the desired QFT ordering of amplitudes.

5. Measurement:

Perform measurements on each qubit to collapse the quantum state into classical bit strings. The measurement outcomes provide the results of the Quantum Fourier Transform in terms of the probabilities of different bit strings.

The final measurement outcomes will represent the amplitudes in the new basis, which corresponds to the Fourier transformed values of the input state. The probability distribution of the measurement outcomes should ideally match the QFT of the original state.

It's important to note that implementing the Quantum Fourier Transform on real quantum hardware can be challenging due to the sensitivity of quantum states to errors and noise. As a result, practical implementations may require error correction techniques and careful calibration of quantum gates.

The 5-qubit Quantum Fourier Transform serves as a building block for more complex quantum algorithms and demonstrates the power of quantum computing in efficiently performing certain mathematical operations that are classically hard to compute.

- **Steps**
- Import the necessary libraries: Import the required libraries for quantum circuit creation and execution
- Create a quantum circuit with 5 qubits.
- Apply Hadamard gates to each qubit.
- Apply controlled phase shift gates to implement the QFT.
- Measure the qubits to obtain the final result.

Key Properties:

1. The Quantum Fourier Transform takes advantage of quantum parallelism to perform computations on multiple states simultaneously.
2. The QFT is a critical component of Shor's algorithm for integer factorization, where it enables efficient period finding.
3. The QFT is used in quantum phase estimation to estimate eigenvalues of unitary operators, which is crucial for various quantum algorithms.
4. The QFT plays a role in quantum algorithms for solving problems in number theory, cryptography, and optimization.

Conclusion: I have understood the implementation of a 5 qubit Quantum Fourier Transform.

Lab Assignment No.	06
Title	Write a program for pre-processing of a text document such as stop word removal, stemming.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 06

Title: Write a program for pre-processing of a text document such as stop word removal, stemming.

Problem Statement: Write a program for pre-processing of a text document such as stop word removal, stemming.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to remove stop words and stemming

Outcomes:

After completion of this assignment students are able to understand how to remove stop words and stemming

Theory:

The process of converting data to something a computer can understand is referred to as **pre-processing.** One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

What are Stop words?

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory. home/pratima/nltk_data/corpora/stopwords are the directory

address.(Do not forget to change your home directory name)

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

To check the list of stopwords you can type the following commands in the python shell.

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
print(stopwords.words('english'))
```

Output:

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

Note: You can even modify the list by adding words of your choice in the English .txt. file in the stopwords directory.

Removing stop words with NLTK

The following program removes stop words from a piece of text:

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize
```

```
example_sent = """This is a sample sentence,
                    showing off the stop words filtration."""


stop_words = set(stopwords.words('english'))


word_tokens = word_tokenize(example_sent)
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
#with no lower case conversion
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)


print(word_tokens)
print(filtered_sentence)
```

Output:

['This', 'is', 'a', 'sample', 'sentence', ',', 'showing',

'off', 'the', 'stop', 'words', 'filtration', '.']

['This', 'sample', 'sentence', ',', 'showing', 'stop',

'words', 'filtration', '.']

Performing the Stopwords operations in a file

In the code below, text.txt is the original input file in which stopwords are to be removed. filteredtext.txt is the output file. It can be done using following code:

import io

from nltk.corpus import stopwords

```
from nltk.tokenize import word_tokenize

# word_tokenize accepts

# a string as an input, not a file.

stop_words = set(stopwords.words('english'))

file1 = open("text.txt")

# Use this to read file content as a stream:

line = file1.read()

words = line.split()

for r in words:

   if not r in stop_words:

      appendFile = open('filteredtext.txt','a')

      appendFile.write(" "+r)

      appendFile.close()
```

This is how we are making our processed content more efficient by removing words that do not contribute to any future operations.

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words. How do we get these tokenized words? Well, tokenization involves breaking down the document into different words.

Stemming is a natural language processing technique that is used to reduce words to their base form, also known as the root form. The process of stemming is used to normalize text and make it easier to process. It is an important step in text pre-processing, and it is commonly used in information retrieval and text mining applications.

There are several different algorithms for stemming, including the Porter stemmer, Snowball stemmer, and the Lancaster stemmer. The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words. The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English. The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer.

Stemming can be useful for several natural language processing tasks such as text classification, information retrieval, and text summarization. However, stemming can also have some negative effects such as reducing the readability of the text, and it may not always produce the correct root form of a word.

Errors in Stemming:

There are mainly two errors in stemming –

over-stemming

under-stemming

Over-stemming occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false-positives. Over-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer produces a root form that is not a valid word or is not the correct root form of a word. This can happen when the stemmer is too aggressive in removing suffixes or when it does not consider the context of the word.

Over-stemming can lead to a loss of meaning and make the text less readable. For example, the word "arguing" may be stemmed to "argu," which is not a valid word and does not convey the same meaning as the original word. Similarly, the word "running" may be stemmed to "run," which is the base form of the word but it does not convey the meaning of the original word.

To avoid over-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing valid root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.

Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Under-stemming occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false-negatives. Under-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer does not produce the correct root form of a word or does not reduce a word to its base form. This can happen when the stemmer is not aggressive enough in removing suffixes or when it is not designed for the specific task or language.

Under-stemming can lead to a loss of information and make it more difficult to analyze text. For example, the word "arguing" and "argument" may be stemmed to "argu," which does not convey the meaning of the original words. Similarly, the word "running" and "runner" may be stemmed to "run," which is the base form of the word but it does not convey the meaning of the original words.

To avoid under-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing

the correct root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.

Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Applications of stemming :

Stemming is used in information retrieval systems like search engines.

It is used to determine domain vocabularies in domain analysis.

To display search results by indexing while documents are evolving into numbers and to map documents to common subjects by stemming.

Sentiment Analysis, which examines reviews and comments made by different users about anything, is frequently used for product analysis, such as for online retail stores. Before it is interpreted, stemming is accepted in the form of the text-preparation mean.

A method of group analysis used on textual materials is called document clustering (also known as text clustering). Important uses of it include subject extraction, automatic document structuring, and quick information retrieval.

Fun Fact: Google search adopted a word stemming in 2003. Previously a search for "fish" would not have returned "fishing" or "fishes".

Some Stemming algorithms are:

Porter's Stemmer algorithm
It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.
Example: EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

Advantage: It produces the best output as compared to other stemmers and it has less error rate.

Limitation: Morphological variants produced are not always real words.

Lovins Stemmer
It is proposed by Lovins in 1968, that removes the longest suffix from a word then the word is recorded to convert this stem into valid words.
Example: sitting -> sitt -> sit

Advantage: It is fast and handles irregular plurals like 'teeth' and 'tooth' etc.

Limitation: It is time consuming and frequently fails to form words from stem.

Dawson Stemmer
It is an extension of Lovins stemmer in which suffixes are stored in the reversed order indexed by their length and last letter.

Advantage: It is fast in execution and covers more suffices.

Limitation: It is very complex to implement.

Krovetz Stemmer
It was proposed in 1993 by Robert Krovetz. Following are the steps:
1) Convert the plural form of a word to its singular form.
2) Convert the past tense of a word to its present tense and remove the suffix 'ing'.
Example: 'children' -> 'child'

Advantage: It is light in nature and can be used as pre-stemmer for other stemmers.

Limitation: It is inefficient in case of large documents.

Xerox Stemmer
Example:

'children' -> 'child'

'understood' -> 'understand'

'whom' -> 'who'

'best' -> 'good'

N-Gram Stemmer
An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in common.
Example: 'INTRODUCTIONS' for n=2 becomes : *I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*

Advantage: It is based on string comparisons and it is language dependent.

Limitation: It requires space to create and index the n-grams and it is not time efficient.

<u>Snowball Stemmer:</u>

When compared to the Porter Stemmer, the Snowball Stemmer can map non-English words too. Since it supports other languages the Snowball Stemmers can be called a multi-lingual stemmer. The Snowball stemmers are also imported from the nltk package. This stemmer is based on a programming language called 'Snowball' that processes small strings and is the most widely used stemmer. The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to

as Porter2 Stemmer. Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed.

Lancaster Stemmer:

The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. But they are not as efficient as Snowball Stemmers. The Lancaster stemmers save the rules externally and basically uses an iterative algorithm.

Some more example of stemming for root word "like" include:

-> "likes"

-> "liked"

-> "likely"

-> "liking"

Errors in Stemming: There are mainly two errors in stemming
– ***Overstemming*** and ***Understemming***. Overstemming occurs when two words are stemmed from the same root that are of different stems. Under-stemming occurs when two words are stemmed from the same root that is not of different stems.

Applications of stemming are:

Stemming is used in information retrieval systems like search engines.

It is used to determine domain vocabularies in domain analysis.

Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.

Below is the implementation of stemming words using NLTK:

```
from nltk.stem import PorterStemmer

from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed

words = ["program", "programs", "programmer", "programming", "programmers"]


for w in words:

   print(w, " : ", ps.stem(w))
```

Output:

```
program : program
programs : program
programmer : program
programming : program
programmers : program
```

Code #2: Stemming words from sentences

```
# importing modules
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
ps = PorterStemmer()
sentence = "Programmers program with programming languages"
words = word_tokenize(sentence)
for w in words:
    print(w, " : ", ps.stem(w))
```

Output :

```
Programmers : program
program : program
with : with
programming : program
languages : language
```

Conclusion: The pre-processing of text data not only reduces the dataset size but also helps us to focus on only useful and relevant data so that the future model would have a large percentage of efficiency. With the help of pre-processing techniques like tokenization, stemming, lemmatization, removing stop-words, and part of speech tag we can remove all the irrelevant text from our dataset and make our dataset ready for further processing or model building.

Lab Assignment No.	07
Title	Implement a program for retrieval of documents using inverted files.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 07

Title: Implement a program for retrieval of documents using inverted files.

Problem Statement: Implement a program for retrieval of documents using inverted files.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to retrieve documents using inverted files

Outcomes:

After completion of this assignment students are able to understand how to retrieve documents using inverted files

Theory:

An **Inverted Index** is a data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms. In an inverted index, the index is organized by terms (words), and each term points to a list of documents or web pages that contain that term.

Inverted indexes are widely used in search engines, database systems, and other applications where efficient text search is required. They are especially useful for large collections of documents, where searching through all the documents would be prohibitively slow.

An inverted index is an index data structure storing a mapping from content, such as words or number s, to its locations in a document or a set of documents. In simple words, it is a hashmap- like data structure that directs you from a word to a document or a web page.

Example: Consider the following documents.

Document 1: The quick brown fox jumped over the lazy dog.

Document 2: The lazy dog slept in the sun.

To create an **inverted index** for these documents, we first tokenize the documents into terms, as follows.

Document 1: The, quick, brown, fox, jumped, over, the lazy, dog.

Document 2: The, lazy, dog, slept, in, the, sun.

Next, we create an index of the terms, where each term points to a list of documents that contain that term, as follows.

The -> Document 1, Document 2
Quick -> Document 1
Brown -> Document 1
Fox -> Document 1
Jumped -> Document 1
Over -> Document 1
Lazy -> Document 1, Document 2
Dog -> Document 1, Document 2
Slept -> Document 2
In -> Document 2
Sun -> Document 2

To search for documents containing a particular term or set of terms, the search engine queries the inverted index for those terms and retrieves the list of documents associated with each term. The search engine can then use this information to rank the documents based on relevance to the query and present them to the user in order of importance.
There are two types of inverted indexes:

- **Record-Level Inverted Index:** Record Level Inverted Index contains a list of references to documents for each word.
- **Word-Level Inverted Index:** Word Level Inverted Index additionally contains the positions of each word within a document. The latter form offers more functionality but needs more processing power and space to be created.

Suppose we want to search the texts "hello everyone, " "this article is based on an inverted index, " and "which is **hashmap-like data structure**". If we index by (text, word within the text), the index with a location in the text is:

hello	(1, 1)
everyone	(1, 2)
this	(2, 1)
article	(2, 2)
is	(2, 3); (3, 2)
based	(2, 4)
on	(2, 5)
inverted	(2, 6)
index	(2, 7)
which	(3, 1)
hashmap	(3, 3)
like	(3, 4)
data	(3, 5)
structure	(3, 6)

The word "he **l**o" is in document 1 ("hello everyone") starting at word 1, so has an entry (1, 1), and the word "is" is in documents 2 and 3 at '3rd' and '2nd' positions respectively (here position is based on the word).
The index may have weights, frequencies, or other indicators.

Steps to Build an Inverted Index

- **Fetch the Document:** Removing of Stop Words: Stop words are the most occurring and useless words in documents like "I", "the", "we", "is", and "an".

- **Stemming of Root Word:** Whenever I want to search for "cat", I want to see a document that has information about it. But the word present in the document is called "cats" or "catty" instead of "cat". To relate both words, I'**l** chop some part of every word I read so that I could get the "root word". There are standard tools for performing this like "Porter's Stemmer".
- **Record Document IDs:** If the word is already present add a reference of the document to index else creates a new entry. Add additional information like the frequency of the word, location of the word, etc.

Example:

Words	Document
ant	doc1
demo	doc2
world	doc1, doc2

Implementing Inverted Index

```
# Define the documents
document1 = "The quick brown fox jumped over the lazy dog."
document2 = "The lazy dog slept in the sun."

# Step 1: Tokenize the documents
# Convert each document to lowercase and split it into words
tokens1 = document1.lower().split()
tokens2 = document2.lower().split()

# Combine the tokens into a list of unique terms
terms = list(set(tokens1 + tokens2))

# Step 2: Build the inverted index
# Create an empty dictionary to store the inverted index
inverted_index = {}

# For each term, find the documents that contain it
for term in terms:
    documents = []
    if term in tokens1:
        documents.append("Document 1")
    if term in tokens2:
        documents.append("Document 2")
    inverted_index[term] = documents

# Step 3: Print the inverted index
for term, documents in inverted_index.items():
    print(term, "->", ", ".join(documents))
```

Advantages of Inverted Index

- The inverted index is to allow fast full-text searches, at a cost of increased processing when a document is added to the database.
- It is easy to develop.
- It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

Disadvantages of Inverted Index

- Large storage overhead and high maintenance costs on updating, deleting, and inserting.
- Instead of retrieving the data in decreasing order of expected usefulness, the records are retrieved in the order in which they occur in the inverted lists.

Features of Inverted Indexes

- **Efficient search:** Inverted indexes allow for efficient searching of large volumes of text-based data. By indexing every term in every document, the index can quickly identify all documents that contain a given search term or phrase, significantly reducing search time.
- **Fast updates:** Inverted indexes can be updated quickly and efficiently as new content is added to the system. This allows for near-real-time indexing and searching for new content.
- **Flexibility:** Inverted indexes can be customized to suit the needs of different types of information retrieval systems. For example, they can be configured to handle different types of queries, such as Boolean queries or proximity queries.
- **Compression:** Inverted indexes can be compressed to reduce storage requirements. Various techniques such as delta encoding, gamma encoding, variable byte encoding, etc. can be used to compress the posting list efficiently.
- **Support for stemming and synonym expansion:** Inverted indexes can be configured to support stemming and synonym expansion, which can improve the accuracy and relevance of search results. Stemming is the process of reducing words to their base or root form, while synonym expansion involves mapping different words that have similar meanings to a common term.
- **Support for multiple languages:** Inverted indexes can support multiple languages, allowing users to search for content in different languages using the same system.

Conclusion: - This way, Implemented a program for inverted files.

Lab Assignment No.	8
Title	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set (You can use Java/Python ML library classes/API.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 08

Title: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set (You can use Java/Python ML library classes/API.

Problem Statement: To construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set (You can use Java/Python ML library classes/API.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to construct a Bayesian network considering medical data.

Outcomes:

After completion of this assignment students are able to understand how to construct a Bayesian network considering medical data.

Theory:

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer,but the computer does not start (observation/evidence). We would like to know

which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.
The corresponding directed acyclic graph is depicted in below figure.

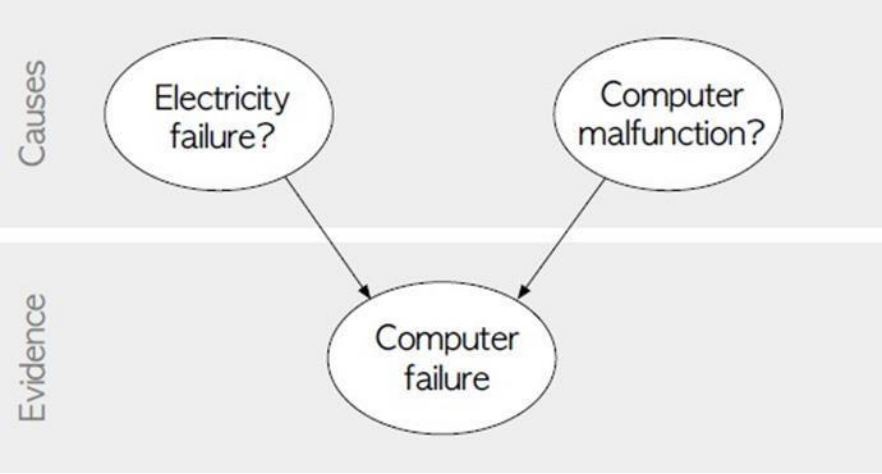


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Data Set:

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart diseasein the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years

2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes'criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. ca = number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. Heartdisease: It is integer valued from 0 (no presence) to 4. Diagnosis of heart disease(angiographic disease status)

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
import numpy as np import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.inference import
VariableElimination

#read Cleveland Heart Disease data   heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)


#display the data          print('Few examples from the dataset are given below')
print(heartDisease.head())


#Model Bayesian Network      Model=BayesianModel([('age','trestbps'),('age','fbs'),
('sex','trestbps'),('exang','trestbps'),('trestbps','heartdise
ase'),('fbs','heartdisease'),('heartdisease','restecg'),
('heartdisease','thalach'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators          print('\n Learning CPD using Maximum
likelihood estimators') model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network  print('\n Inferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

#computing the Probability of HeartDisease given Age         print('\n 1. Probability of HeartDisease
given Age=30') q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'age':28})
print(q['heartdisease'])

#computing the Probability of HeartDisease given cholesterol print('\n 2. Probability of HeartDisease
given cholesterol=100') q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'chol':100})
print(q['heartdisease'])
```

Output:

```
Few examples from the dataset are given below
```

```
     age  sex cp  trestbps     ...slope  ca  thal
heartdisease
0    63    1   1      145       ...  3   0     6              0
1    67    1   4      160       ...  2   3     3              2
2    67    1   4      120       ...  2   2     7              1
3    37    1   3      130       ...  3   0     3              0
4    41    0   2      130       ...  1   0     3              0

[5 rows x 14 columns]

Learning CPD using Maximum likelihood

estimatorsInferencing with Bayesian Network:


1. Probability of HeartDisease given Age=28
```

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

2. Probability of HeartDisease given cholesterol=100

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

Conclusion: This way constructed a Bayesian network considering medical data.

Lab Assignment No.	09
Title	Implement Agglomerative hierarchical clustering algorithm using appropriate dataset.
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 09

Title: Implement Agglomerative hierarchical clustering algorithm using appropriate dataset.

Problem Statement: Implement Agglomerative hierarchical clustering algorithm using appropriate dataset

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Implement Agglomerative hierarchical clustering algorithm using appropriate dataset

Outcomes:

After completion of this assignment students are able to understand how to Implement Agglomerative hierarchical clustering algorithm using appropriate dataset

Theory:

In data mining and statistics, hierarchical clustering analysis is a method of clustering analysis that seeks to build a hierarchy of clusters i.e. tree-type structure based on the hierarchy.

In machine learning, clustering is the unsupervised learning technique that groups the data based on similarity between the set of data. There are different-different types of clustering algorithms in machine learning. **Connectivity-based clustering:** This type of clustering algorithm builds the cluster based on the connectivity between the data points. Example: Hierarchical clustering

- **Centroid-based clustering:** This type of clustering algorithm forms around the centroids of the data points. Example: K-Means clustering, K-Mode clustering
- **Distribution-based clustering:** This type of clustering algorithm is modeled using statistical distributions. It assumes that the data points in a cluster are generated from a particular probability distribution, and the algorithm aims to estimate the parameters of the distribution to group similar data points into clusters Example: Gaussian Mixture Models (GMM)
- **Density-based clustering:** This type of clustering algorithm groups together data points that are in high-density concentrations and separates points in low-concentrations regions. The basic idea

is that it identifies regions in the data space that have a high density of data points and groups those points together into clusters. Example: DBSCAN(Density-Based Spatial Clustering of Applications with Noise)

Hierarchical clustering

Hierarchical clustering is a connectivity-based clustering model that groups the data points together that are close to each other based on the measure of similarity or distance. The assumption is that data points that are close to each other are more similar or related than data points that are farther apart.

A dendrogram, a tree-like figure produced by hierarchical clustering, depicts the hierarchical relationships between groups. Individual data points are located at the bottom of the dendrogram, while the largest clusters, which include all the data points, are located at the top. In order to generate different numbers of clusters, the dendrogram can be sliced at various heights.

The dendrogram is created by iteratively merging or splitting clusters based on a measure of similarity or distance between data points. Clusters are divided or merged repeatedly until all data points are contained within a single cluster, or until the predetermined number of clusters is attained.

We can look at the dendrogram and measure the height at which the branches of the dendrogram form distinct clusters to calculate the ideal number of clusters. The dendrogram can be sliced at this height to determine the number of clusters.

Types of Hierarchical Clustering

Basically, there are two types of hierarchical Clustering:

1. Agglomerative Clustering
2. Divisive clustering

Hierarchical Agglomerative Clustering

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.

Agglomerative Clustering is one of the most common hierarchical clustering techniques. Dataset – Credit Card Dataset.

Assumption: The clustering technique assumes that each data point is similar enough to the other data points that the data at the starting can be assumed to be clustered in 1 cluster. Step 1: Importing the required libraries

```
import pandas as pd
```

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.cluster import AgglomerativeClustering

from sklearn.preprocessing import StandardScaler, normalize

from sklearn.metrics import silhouette_score

import scipy.cluster.hierarchy as shc
```

Step 2: Loading and Cleaning the data

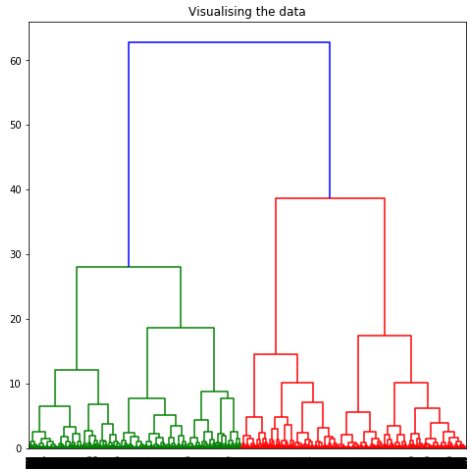
```
# Changing the working location to the location of the file

cd C:\Users\Dev\Desktop\Kaggle\Credit_Card


X = pd.read_csv('CC_GENERAL.csv')


# Dropping the CUST_ID column from the data

X = X.drop('CUST_ID', axis = 1)


# Handling the missing values

X.fillna(method ='ffill', inplace = True)
```

Step 3: Preprocessing the data

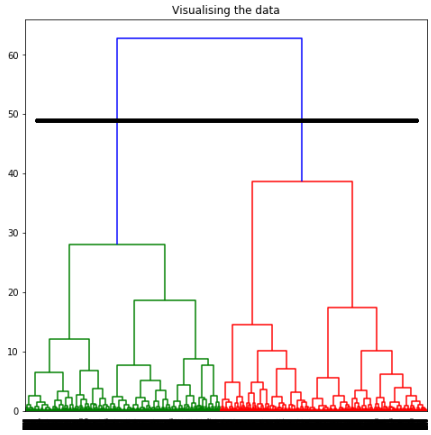
```
# Scaling the data so that all the features become comparable
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalizing the data so that the data approximately
# follows a Gaussian distribution
X_normalized = normalize(X_scaled)

# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

Step 4: Reducing the dimensionality of the Data
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

Dendrograms are used to divide a given cluster into many different clusters. Step 5: Visualizing the working of the Dendrograms
plt.figure(figsize =(8, 8))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='ward')))



To determine the optimal number of clusters by visualizing the data, imagine all the horizontal lines as being completely horizontal and then after calculating the maximum distance between any two horizontal lines, draw a horizontal line in the maximum distance calculated.

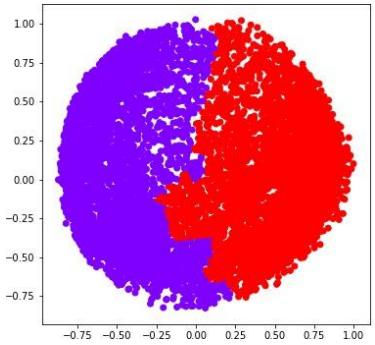


The above image shows that the optimal number of clusters should be 2 for the given data. Step 6: Building and Visualizing the different clustering models for different values of k a) k = 2

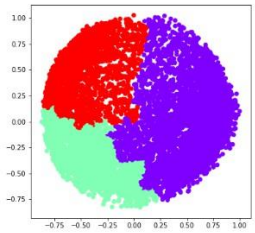
ac2 = AgglomerativeClustering(n_clusters = 2)

Visualizing the clustering

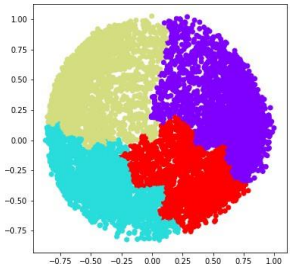
```
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
        c = ac2.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



```
ac3 = AgglomerativeClustering(n_clusters = 3)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
        c = ac3.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```

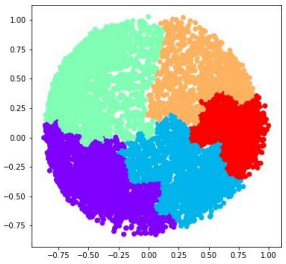


```
ac4 = AgglomerativeClustering(n_clusters = 4)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
         c = ac4.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```

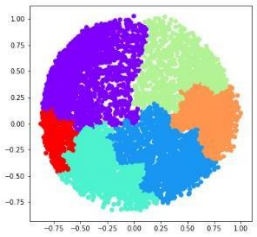


```
ac5 = AgglomerativeClustering(n_clusters = 5)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
```

```
        c = ac5.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```

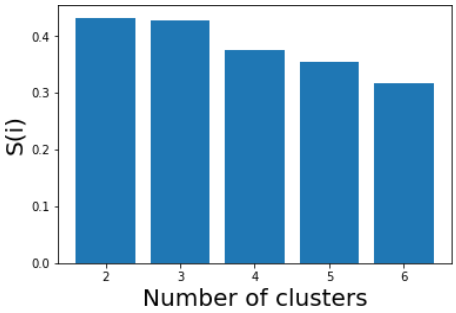


```
ac6 = AgglomerativeClustering(n_clusters = 6)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
        c = ac6.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



We now determine the optimal number of clusters using a mathematical technique. Here, We will use the Silhouette Scores for the purpose. Step 7: Evaluating the different models and Visualizing the results.

```
k = [2, 3, 4, 5, 6]

# Appending the silhouette scores of the different models to the list
silhouette_scores = []
silhouette_scores.append(
      silhouette_score(X_principal, ac2.fit_predict(X_principal)))
silhouette_scores.append(
      silhouette_score(X_principal, ac3.fit_predict(X_principal)))
silhouette_scores.append(
      silhouette_score(X_principal, ac4.fit_predict(X_principal)))
silhouette_scores.append(
      silhouette_score(X_principal, ac5.fit_predict(X_principal)))
silhouette_scores.append(
      silhouette_score(X_principal,  ac6.fit_predict(X_principal)))

# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```



with the help of the silhouette scores, it is concluded that the optimal number of clusters for the given data and clustering technique is 2.

Conclusion :- This way Implemented Agglomerative hierarchical clustering algorithm using appropriate dataset

Lab Assignment No.	10
Title	Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 10

Title: Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Problem Statement: Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Outcomes:

After completion of this assignment students are able to understand how to Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Theory:

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measurin g the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

It is not the only algorithm used by Google to order search engine results, but it is the first algorit hm that was used by the company, and it is the best-known.
The above centrality measure is not implemented for multi-graphs.

Algorithm

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collectio ns of documents of any size. It is assumed in several research papers that the distributio n is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, ca lled “iterations”, through the collection to adjust approximat e PageRank values to more closely reflect the theoretical true value.

Simplified algorithm

Assume a small universe of four web pages: A, B, C, and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25.

The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.

If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value , 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one-third of its existing value, or approximately 0.083, to A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.

In other words, the PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links L( ).

In the general case, the PageRank value for any page u can be expressed as:

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v. The algorithm involves a damping factor for the calculation of the PageRank. It is like the income tax which the govt extracts from one despite paying him itself.

Following is the code for the calculation of the Page rank.

```
def pagerank(G, alpha=0.85, personalization=None,
             max_iter=100, tol=1.0e-6, nstart=None, weight='weight',
             dangling=None):
    """Return the PageRank of the nodes in the graph.

    PageRank computes a ranking of the nodes in the graph G based on
    the structure of the incoming links. It was originally designed as
    an algorithm to rank web pages.

    Parameters
    ----------
    G : graph
      A NetworkX graph.  Undirected graphs will be converted to a directed
      graph with two directed edges for each undirected edge.

    alpha : float, optional
      Damping parameter for PageRank, default=0.85.

    personalization: dict, optional
      The "personalization vector" consisting of a dictionary with a
      key for every graph node and nonzero personalization value for each node.
```

```
  By default, a uniform distribution is used.

max_iter : integer, optional
  Maximum number of iterations in power method eigenvalue solver.

tol : float, optional
  Error tolerance used to check convergence in power method solver.

nstart : dictionary, optional
  Starting value of PageRank iteration for each node.

weight : key, optional
  Edge data key to use as weight.  If None weights are set to 1.

dangling: dict, optional
  The outedges to be assigned to any "dangling" nodes, i.e., nodes without
  any outedges. The dict key is the node the outedge points to and the dict
  value is the weight of that outedge. By default, dangling nodes are given
  outedges according to the personalization vector (uniform if not
  specified). This must be selected to result in an irreducible transition
  matrix (see notes under google_matrix). It may be common to have the
  dangling dict to be the same as the personalization dict.

Returns
-------
pagerank : dictionary
   Dictionary of nodes with PageRank as value

Notes
-----
The eigenvector calculation is done by the power iteration method
and has no guarantee of convergence.  The iteration will stop
after max_iter iterations or an error tolerance of
number_of_nodes(G)*tol has been reached.

The PageRank algorithm was designed for directed graphs but this
algorithm does not check if the input graph is directed and will
execute on undirected graphs by converting each edge in the
directed graph to two edges.


"""
if len(G) == 0:
    return {}

if not G.is_directed():
    D = G.to_directed()
else:
    D = G

# Create a copy in (right) stochastic form
W = nx.stochastic_graph(D, weight=weight)
N = W.number_of_nodes()

# Choose fixed starting vector if not given
```

```
    if nstart is None:
        x = dict.fromkeys(W, 1.0 / N)
    else:
        # Normalized nstart vector
        s = float(sum(nstart.values()))
        x = dict((k, v / s) for k, v in nstart.items())

    if personalization is None:

        # Assign uniform personalization vector if not given
        p = dict.fromkeys(W, 1.0 / N)
    else:
        missing = set(G) - set(personalization)
        if missing:
            raise NetworkXError('Personalization dictionary '
                                'must have a value for every node. '
                                'Missing nodes %s' % missing)
        s = float(sum(personalization.values()))
        p = dict((k, v / s) for k, v in personalization.items())

    if dangling is None:

        # Use personalization vector if dangling vector not specified
        dangling_weights = p
    else:
        missing = set(G) - set(dangling)
        if missing:
            raise NetworkXError('Dangling node dictionary '
                                'must have a value for every node. '
                                'Missing nodes %s' % missing)
        s = float(sum(dangling.values()))
        dangling_weights = dict((k, v/s) for k, v in dangling.items())
    dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

    # power iteration: make up to max_iter iterations
    for _ in range(max_iter):
        xlast = x
        x = dict.fromkeys(xlast.keys(), 0)
        danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
        for n in x:

            # this matrix multiply looks odd because it is
            # doing a left multiply x^T=xlast^T*W
            for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
            x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]

        # check convergence, l1 norm
        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N*tol:
            return x
    raise NetworkXError('pagerank: power iteration failed to converge '
                        'in %d iterations.' % max_iter)
```

The above code is the function that has been implemented in the networkx library.

To implement the above in networkx, you will have to do the following:

```
>>> import networkx as nx
>>>  G=nx.barabasi_albert_graph(60,41)
>>> pr=nx.pagerank(G,0.4)
>>> pr
```

Below is the output, you would obtain on the IDLE after required installations.

```
{0: 0.012774147598875784, 1: 0.013359655345577266, 2: 0.013157355731377924,
3: 0.012142198569313045, 4: 0.013160014506830858, 5: 0.012973342862730735,
   6: 0.012166706783753325, 7: 0.011985935451513014, 8: 0.012973502696061718,
  9: 0.013374146193499381, 10: 0.01296354505412387, 11: 0.013163220326063332,
 12: 0.013368514624403237, 13: 0.013169335617283102, 14: 0.012752071800520563,
15: 0.012951601882210992, 16: 0.013776032065400283, 17: 0.012356820581336275,
18: 0.013151652554311779, 19: 0.012551059531065245, 20: 0.012583415756427995,
 21: 0.013574117265891684, 22: 0.013167552803671937, 23: 0.013165528583400423,
 24: 0.012584981049854336, 25: 0.013372989228254582, 26: 0.012569416076848989,
 27: 0.013165322299539031, 28: 0.012954300960607157, 29: 0.012776091973397076,
 30: 0.012771016515779594, 31: 0.012953404860268598, 32: 0.013364947854005844,
33: 0.012370004022947507, 34: 0.012977539153099526, 35: 0.013170376268827118,
 36: 0.012959579020039328, 37: 0.013155319659777197, 38: 0.013567147133137161,
 39: 0.012171548109779459, 40: 0.01296692767996657, 41: 0.028089802328702826,
 42: 0.027646981396639115, 43: 0.027300188191869485, 44: 0.02689771667021551,
 45: 0.02650459107960327, 46: 0.025971186884778535, 47: 0.02585262571331937,
48: 0.02565482923824489, 49: 0.024939722913691394, 50: 0.02458271197701402,
 51: 0.024263128557312528, 52: 0.023505217517258568, 53: 0.023724311872578157,
 54: 0.02312908947188023, 55: 0.02298716954828392, 56: 0.02270220663300396,
 57: 0.022060403216132875, 58: 0.021932442105075004, 59: 0.021643288632623502}
```

Conclusion:- Thus, this way the centrality measure of Page Rank is calculated for the given graph.