INDEX

Ex No	Date	Name of the Experiment	Page no	Staff sign
1		Demonstrating Mathematical functions using Webppl		
2		Implementation of Rule based Reasoning using Java script		
3		Creating and importing csv file using pandas library		
4		Developing an Application system using Linear Regression Model		
5		Developing an Application system using Logistic Regression Model		
6		Developing an Application system using Generative AI model		
7		Developing an Application system using Random Forest Classifier Algorithm		
8		Developing an Application system using Gaussian Mixture Model (Unsupervised Learning Technique)		
9		Developing an Application system using Conditional Learning Inference Model(Naive Bayes Algorithm)		
10		Developing an Application system using Hierarchical Clustering Model (Unsupervised Learning Technique)		

Ex No: 01	DEMONSTRATING MATHEMATICAL
Date:	FUNCTIONS USING WEBPPL

AIM:

To develop a program that implements the mathematical functions using webppl.

PROCEDURE:

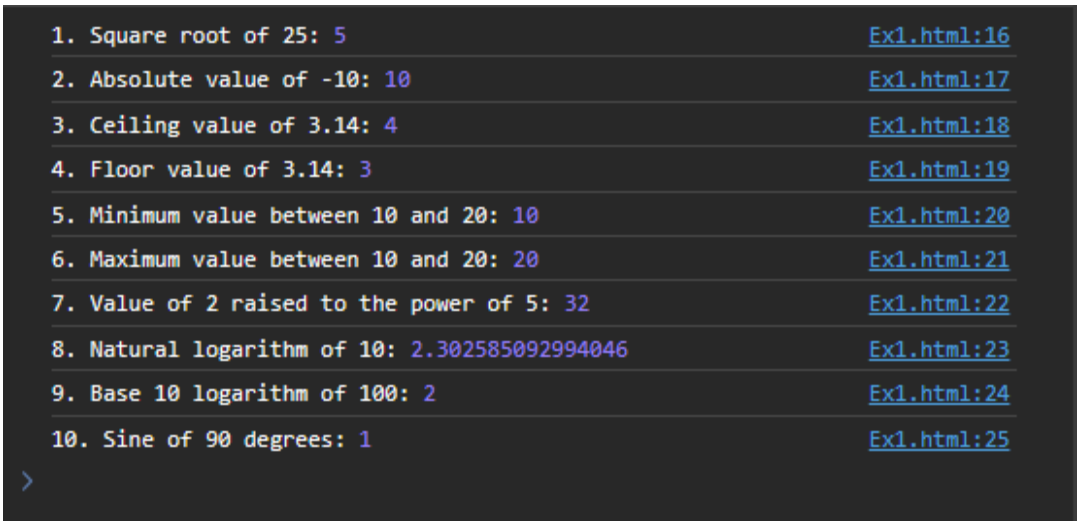
1) Start the program.
2) Start HTML document with doctype declaration.
3) Set document language to English and define UTF-8 encoding.
4) Head section: Set viewport, title to "Math Functions".
5) Body: Add h1 with "Mathematical Functions" and a paragraph.
6) Script: Output results of mathematical functions to console.
7) End HTML document.
8) Stop the program.

PROGRAM:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Math Functions</title>

</head>

<body>

<h1>Mathematical Functions</h1>

<p>Open the console to see the results of the mathematical functions.</p>

<script>

  // Builtin mathematical functions
```

```
  console.log("1. Square root of 25:", Math.sqrt(25));
  console.log("2. Absolute value of -10:", Math.abs(-10));
  console.log("3. Ceiling value of 3.14:", Math.ceil(3.14));
  console.log("4. Floor value of 3.14:", Math.floor(3.14));
  console.log("5. Minimum value between 10 and 20:", Math.min(10, 20));
  console.log("6. Maximum value between 10 and 20:", Math.max(10, 20));
  console.log("7. Value of 2 raised to the power of 5:", Math.pow(2, 5));
  console.log("8. Natural logarithm of 10:", Math.log(10));
  console.log("9. Base 10 logarithm of 100:", Math.log10(100));
  console.log("10. Sine of 90 degrees:", Math.sin(Math.PI / 2));
</script>
</body>
</html>
```

OUTPUT:



RESULT:

Thus the program that implements the mathematical function is developed successfully and the output is displayed successfully.

Ex No: 02	IMPLEMENTATION OF RULE BASED REASONING
Date:	USING JAVA SCRIPT

AIM:

To write a program that implements the rule based reasoning using javascript.

PROCEDURE:

1) Open the Notepad and type the code.
2) Start HTML document with doctype declaration.
3) Set document language to English and define UTF-8 encoding.
4) Add Event Listener for HTML element in script.
5) Display the function and compare with choice.
6) Display the required output as per the choice selection.
7) Minimum 4 choices need to be added to the code.
8) The output can be displayed in the document of paragraph tag.
9) Save the coding in the filename index.html.
10) Open the web browser to view the output.

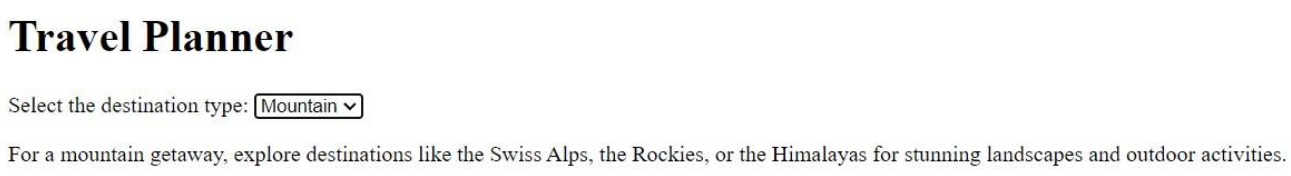
PROGRAM:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Travel Planner</title>
</head>
<body>
<h1>Travel Planner</h1>
<label for="destinationTypes">Select the destination type:</label>
   <select id="destinationTypes">
```

```
        <option value="beach">Beach</option>
        <option value="mountain">Mountain</option>
        <option value="city">City</option>
        <option value="historic">Historic</option>
    </select>
 <p id="travelRecommendation">Travel recommendation will be displayed here.</p>
<script>
        // Get the HTML elements
        const destinationTypesDropdown = document.getElementById('destinationTypes');
        const travelRecommendationParagraph =
document.getElementById('travelRecommendation');
 // Add event listener to the dropdown
 destinationTypesDropdown.addEventListener('change', function() {
 // Call the rule-based reasoning function with the selected destination type
 travelRecommendation(destinationTypesDropdown.value);
 });
 // Rule-based reasoning function for travel recommendation
 function travelRecommendation(selectedDestinationType) {
  // Perform logic based on the selected destination type
 switch (selectedDestinationType) {
 case 'beach':
                travelRecommendationParagraph.textContent = 'For a beach destination,
consider places like Bali, Maldives, or Miami for a relaxing and sunny vacation.';
                break;
            case 'mountain':
```

```
            travelRecommendationParagraph.textContent = 'For a mountain getaway, explore destinations like the Swiss Alps, the Rockies, or the Himalayas for stunning landscapes and outdoor activities.';
            break;
        case 'city':
            travelRecommendationParagraph.textContent = 'For a city escape, visit vibrant cities such as Tokyo, New York, or Paris for cultural experiences, shopping, and nightlife.';
            break;
        case 'historic':
            travelRecommendationParagraph.textContent = 'If you enjoy history, consider visiting historic places like Rome, Athens, or Cairo to explore ancient ruins and learn about rich cultural heritage.';
            break;
        default:
            travelRecommendationParagraph.textContent = 'Invalid destination type';
    }
  }
 </script>
</body>
</html>
```

OUTPUT:



RESULT:

Thus the program that creates and importing the csv files is developed and the output is verified successfully.

Ex No: 03	CREATING AND IMPORTING CSV FILE
Date:	USING PANDAS LIBRARY

AIM:

To write a program that implements Creating and importing csv file using pandas library.

PROCEDURE:

1. Start.
2. Import the pandas library for data manipulation and the csv module for working with CSV files.
3. Define the data to be written to the CSV file as a list of lists.
4. Specify the file name for the CSV file.
5. Open the CSV file in write mode using a context manager to ensure proper file handling.
6. Create a CSV writer object using the csv module.
7. Write the data to the CSV file using the writerows() method.
8. Print a success message indicating that the CSV file has been created.
9. Read the contents of the created CSV file using pandas's read_csv() function.
10. Print the contents of the CSV file.
11. End.

PROGRAM:

```
import pandas

import csv

# Data to be written to the CSV file

data = [

   ["Name", "Age", "City"],

   ["John Doe", 25, "New York"],

   ["Jane Smith", 30, "London"],

   ["Bob Johnson", 22, "Tokyo"]

]

# Specify the file name

file_name = "example1.csv"

# Writing data to the CSV file

with open(file_name, mode='w', newline='') as file:
```

```
    writer = csv.writer(file)
    writer.writerows(data)
print(f"CSV file '{file_name}' created successfully.")
csv_file=pandas.read_csv('example1.csv')
print(csv_file)
```

OUTPUT:

NAME	AGE	CITY
John Doe	25	New York
Jane Smith	30	London
Bob Johnson	22	Tokyo

RESULT:

Thus the program that creates and importing the csv files is developed and the output is verified successfully.

Ex No: 04	DEVELOPING AN APPLICATION SYSTEM USING
Date:	LINEAR REGRESSION MODEL

AIM:

To create a program that implements the linear regression model using the python.

PROCEDURE:

1. Start the program.
2. Import necessary libraries.
3. Define age and weight data.
4. Prepare data for regression analysis.
5. Fit a linear regression model.
6. Obtain model coefficients and intercept.
7. Predict weight using the model.
8. Calculate RMSE and R^2.
9. Plot age versus weight and regression line.
10. Show the plot.
11. End the program.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
age=[67,20,12,11,8,45]
weight=[80,52,34,31,23,67]
x=np.array([age]).T
y=np.array(weight)
rmodel=LinearRegression()
rmodel=rmodel.fit(x,y)
rmodel_slope=rmodel.coef_
rmodel_intercept=rmodel.intercept_
print("Model Slope",rmodel_slope)
print("Model Intercept",rmodel_intercept)
```

Ex No: 05	DEVELOPING AN APPLICATION SYSTEM USING LOGISTIC REGRESSION MODEL
Date:	

AIM:

To write a program that implements the logistic regression model using the python.

PROCEDURE:

1. Start
2. Import the pandas library as pd and necessary modules from sklearn.
3. Load the dataset "Book.csv" into a DataFrame named movie_info.
4. Display the entire dataset.
5. Split the dataset into input and output dataset.
6. Display the input and output dataset.
7. Create a Logistic Regression model with a maximum iteration of 1000.
8. Train the model using the input and output datasets.
9. Define new data points for prediction.
10. Predict the target variable for the new data points using the trained model.
11. Print the predicted values.
12. End.

PROGRAM:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
movie_info = pd.read_csv(r"E:\Cognitive science\Book.csv")
print(movie_info.head())
input_dataset = movie_info.drop(columns=["Marks"])
output_dataset = movie_info["Marks"]
print(input_dataset.head())
print(output_dataset.haed())
movie_model = LogisticRegression(max_iter=1000)
movie_model.fit(input_dataset, output_dataset)
new_data_points = [[14, 1, 95], [15, 0, 99]]
true_values = [69, 69]
```

```
movie_interest = movie_model.predict(new_data_points)
print(movie_interest)
print("Accuracy is", metrics.accuracy_score(true_values, movie_interest))
```

OUTPUT:

```
Studentid  Age  Gender  Marks
0          1   14      1     95
2          3   15      1     81
3          4   15      1     75
4          5   14      1     89
   Studentid  Age  Gender
0          1   14      1
1          2   15      1
2          3   15      1
3          4   15      1
4          5   14      1
0    95
1    99
2    81
3    75
4    89
[69 69]
Accuracy is 1.0
```

RESULT:

Thus the program that implements the logistic regression is developed and the output is verified successfully.

Ex No: 06	DEVELOPING AN APPLICATION SYSTEM USING
Date:	GENERATIVE AI MODEL

AIM:

Generative AI, also referred to as GenAI, allows users to input a variety of prompts to generate new content, such as text, images, videos, sounds, code, 3D designs, and other media. It "learns" and is trained on documents and artifacts that already exist online. Creating a simple text-generating AI can be a fun and insightful project. This generator will use a simple approach called a Markov chain, which models language as a series of probabilistic transitions between states.

Requirements

- Python installed on your computer.
- Basic understanding of Python programming.
- An IDE or a simple text editor.

Step-by-Step Guide

Step 1: Install Python Libraries

First, ensure that you have the necessary Python libraries installed. For this project, you might use numpy for handling arrays. You can install it via pip if it's not already installed:

bashCopy code

pip install numpy

Step 2: Prepare the Text Data

You need some text data to train your Markov chain model. This could be any large text file, like a book from Project Gutenberg. For this example, let's say you have a plain text file named sample.txt.

Step 3: Read and Process the Text

Create a Python script to read and process this text. The processing will involve cleaning the text and splitting it into a list of words.

Python code

```
#Import required library
import numpy as np
# Function to load and clean text
def load_text(filename):
    with open(filename, encoding='utf-8') as f:
```

```
        text = f.read()
    # Convert text to lowercase
    text = text.lower()
    # Replace any characters that are not alphabets or spaces with nothing
    text = ''.join(char for char in text if char.isalpha() or char.isspace())
    words = text.split()
    return words
# Load words from a file
words = load_text('sample.txt')
```

Step 4: Build the Markov Chain

A Markov chain needs to understand the probability of moving from one word to the next. We can achieve this by building a dictionary where each key is a word, and its value is another dictionary that keeps track of the words that follow and their frequency.

```
def build_markov_chain(words):
    markov_chain = {}
    for current_word, next_word in zip(words[:-1], words[1:]):
        if current_word not in markov_chain:
            markov_chain[current_word] = {}
        if next_word not in markov_chain[current_word]:
            markov_chain[current_word][next_word] = 1
        else:
            markov_chain[current_word][next_word] += 1
    return markov_chain
markov_chain = build_markov_chain(words)
```

Step 5: Generate Text from the Markov Chain

To generate text, we start from a random word and then use the chain to find probable next words. We continue this process to generate as much text as we desire.

```
def generate_sentence(chain, length=20):
    # Select a random word to start with
    word = np.random.choice(list(chain.keys()))
    sentence = [word]
    for i in range(length - 1):
```

```
        next_words = list(chain[word].keys())
        word_weights = list(chain[word].values())
        word = np.random.choice(next_words, p=np.array(word_weights) /
np.array(word_weights).sum())
        sentence.append(word)
    return ' '.join(sentence)
print(generate_sentence(markov_chain))
```

Step 6: Final Steps and Running Your Code

Save this script as a Python file (e.g., text_generator.py) and run it using Python. It will read your text file, build a Markov chain model, and then generate a sentence based on that model.

python text_generator.py

This is a very basic text generator. More advanced text generators, such as those based on neural networks (e.g., using TensorFlow or PyTorch), can generate more coherent and contextually appropriate text. For those, you would typically explore libraries like Hugging Face's Transformers, which provide easy access to models like GPT (Generative Pre-trained Transformer).

The output of the script using the Markov chain model for text generation will depend on the text data that you provided (sample.txt) and the random processes within the script. Since the model is built on a basic statistical method where transitions from one word to the next are probabilistic but only based on the frequency of occurrence, the generated text can vary widely in coherence and relevance.

Here is an example of how this might work. Let's assume your text file contains a simple body of text from a classic novel or a collection of sentences about a particular subject. The output will be a randomly generated sentence of 20 words (as specified by the length=20 in the function call). Here's an example output scenario:

Sample Text Input

If the text in sample.txt was something straightforward and repetitive like:

the cat in the hat came back. the cat in the hat loves to play. the cat in the hat is happy.

Building the Markov Chain

The resulting Markov chain dictionary might look something like this, simplified for illustration:

Python code

{ 'the': {'cat': 3}, 'cat': {'in': 3}, 'in': {'the': 3}, 'hat': {'came': 1, 'loves': 1, 'is': 1}, 'came': {'back': 1}, 'back': {'the': 1}, 'loves': {'to': 1}, 'to': {'play': 1}, 'play': {'the': 1}, 'is': {'happy': 1}, 'happy': {'the': 1} }

Example Output

the cat in the hat came back the cat in the hat loves to play the cat in

Explanation

This example demonstrates a simple 20-word sentence generated based on the probabilities in the Markov chain. Since our example input text was quite simple and repetitive, the generated text tends to stick closely to the patterns in the input. However, if your source text (sample.txt) contains a richer vocabulary and more complex sentence structures, the output will be more varied but potentially less coherent, as the Markov model does not understand context beyond the immediate next word.

Coherence and Utility

The coherence of the text generated by such a simple model is limited. For more sophisticated text generation, especially for applications needing higher quality and contextually aware output, more advanced models based on deep learning, such as LSTM networks or Transformer models like GPT-2 or GPT-3, are recommended. These models understand longer dependencies and subtleties in text data, producing more coherent and contextually appropriate outputs.

The sample.txt file mentioned in the Python example for creating a text generator is a hypothetical file containing text data that you provide. This file is meant to be a plain text file consisting of any textual content that you want to use to train the Markov chain model for generating new text.

Creating a sample.txt File

To proceed with the example provided, you need to create a sample.txt file. Here's how you can create this file and add some sample content:

1. Choose Your Text: The text can be anything from a book excerpt, a collection of quotes, speeches, articles, or any large body of text. For educational or testing purposes, it's common to use public domain texts such as those from Project Gutenberg.
2. Create the File:
    - On Windows, you can use Notepad or any other text editor. Simply open the editor, paste your chosen text, and save the file as sample.txt.
    - On macOS, you can use TextEdit or another text editor, ensuring you save the file in plain text format.
    - On Linux, you can use Gedit, Nano, or Vim, and follow similar steps to create and save your file.

Example Content

If you're looking for a quick way to create a sample.txt, you can use the following excerpt from "Alice's Adventures in Wonderland" by Lewis Carroll, which is a public domain text:

Plain text Copy code

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?' So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her. There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on, Alice started to her feet, for she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

Usage

Once you've created your sample.txt with suitable content, the Python script you write (as described in previous messages) will read from this file to build the Markov chain model. The text you choose will directly influence the style, vocabulary, and patterns of the generated text, so selecting a text that represents the kind of output you desire is essential.

To predict the potential output of a Markov chain text generator using the provided sample text from "Alice's Adventures in Wonderland" by Lewis Carroll, let's consider how a basic Markov chain model processes this data. Since this model uses a very simplistic approach, it builds a series of transitions based solely on the occurrence of words and the immediate following words without understanding context or grammar.

How the Markov Chain Works:

1. Reading Input: The input text you provided is read and split into a sequence of words.

2. Building the Chain: A dictionary is created where each key is a word from the text, and each value is another dictionary. The inner dictionary tracks the words that follow the key word in the text and how often each successor word appears.

3. Generating Text: To generate text, the model starts with a random word (or a specified start word) and then randomly selects the next word based on the probabilities derived from the frequency of successor words in the training data. This process is repeated to form sentences of a desired length.

Example of Chain Construction:

The Markov chain might include entries like:

pythonCopy code

{ "Alice": {"was": 1}, "was": {"beginning": 1}, "beginning": {"to": 1}, "to": {"get": 1}, "get": {"very": 1}, ... }

Generating Output:

When generating text, if the starting word is "Alice," the model looks up "Alice" in the dictionary, sees "was" as the next word (since it's the only option provided by this sample), then looks up "was," finds "beginning," and so on. The randomness primarily comes into play when there are multiple options for the next word.

Example Output:

Using the full provided text, an example output of 20 words might look like this:

vbnetCopy code

Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to

This output directly follows one possible path through the actual text. However, if there were more variety in the successors of each word (i.e., more text or more complex text structures), the output could start to diverge more significantly from the input text, producing new and unique sentences.

Conclusion:

Thus the Generative AI encompasses a range of technologies that can generate new content, whether it is text, images, music, or even code, based on the data they have been trained on. These technologies have shown significant progress and promise, with applications spanning creative, commercial, and educational field.

Ex No: 07	RANDOM FOREST CLASSIFIER
Date:	

AIM:

To write a program that implements the Random Forest Classifier model using the python.

PROCEDURE:

1. Start
2. Import necessary libraries.
3. Read the dataset into a Data Frame and display information and the first few rows.
4. Split the dataset into training and testing sets.
5. Create and train a Random Forest Classifier model.
6. Predict outcomes and calculate accuracy.
7. Create a Random Forest Classifier model with 200 estimators and fit it to the training data.
8. Visualize feature importance using a barplot.
9. Set labels and title for the plot.
10. Show the plot.
11. End

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
df=pd.read_csv(r"E:\Cognitive science\diabetes.csv")
df.info()
df.head()
x=df.iloc[:,df.columns!='Outcome']
y=df.iloc[:,df.columns=='Outcome']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
model.fit(x_train,y_train)
predict_output=model.predict(x_test)
print(predict_output)
from sklearn.metrics import accuracy_score
acc=accuracy_score(predict_output,y_test)
print(acc)
classifier=RandomForestClassifier(n_estimators=200)
classifier.fit(x_train,y_train)
feature_importances_df=pd.DataFrame({"feature":list(x.columns),"importance":classifier.f
eature_importances_}).sort_values("importance",ascending=False)
sns.barplot(x=feature_importances_df.feature,y=feature_importances_df.importance)
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Visualizing Important Features")
plt.xticks(rotation=45, horizontalalignment="right",fontweight="light",fontsize="x-large")
plt.show()
```

OUTPUT:

```
RangeIndex: 191 entries, 0 to 190
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Glusose                     191 non-null    int64
 1   BloodPressure               191 non-null    int64
 2   Skin Thickness              191 non-null    int64
 3   Insulin                     191 non-null    int64
 4   BMI                         191 non-null    float64
 5   Diabetes Pedigree Function  191 non-null    float64
 6   Age                         191 non-null    int64
```

Ex No: 08	DEVELOPING AN APPLICATION SYSTEM USING GAUSSIAN MIXTURE MODEL
Date:	

AIM:

To write a program that implements the gaussian mixture model using the python.

Introduction to Clustering :

Clustering refers to grouping similar data points together, based on their attributes or features.

For example, if we have the income and expenditure for a set of people, we can divide them into

the following groups:

- First – Earn high, spend high
- Second – Earn high, spend low
- Third – Earn low, spend low
- Fourth – Earn low, spend high

Each of these groups would hold a population with similar features and can be useful in pitching the relevant scheme/product to the group. Think of credit cards, car/property loans, and so on. In simple words:

The idea behind clustering is grouping data points together, such that each individual cluster holds the most similar points.

There are various clustering algorithms out there. One of the most popular clustering algorithms is k-means. Let us understand how the k-means algorithm works and what are the possible scenarios where this algorithm might come up short of expectations.

Introduction to k-means Clustering:

k-means clustering is a distance-based algorithm. This means that it tries to group the closest points to form a cluster.

So, we first define the number of groups that we want to divide the population into – that's the value of k. Based on the number of clusters or groups we want, we then randomly initialize k centroids.

The data points are then assigned to the closest centroid and a cluster is formed. The centroids are then updated and the data points are reassigned. This process goes on iteratively until the location of centroids no longer changes.

Drawbacks of k-means Clustering

Let's take the same income-expenditure example we saw above. The k-means algorithm seems to be working pretty well, right? Hold on – if you look closely, you will notice that all the clusters created have a circular shape. This is because the centroids of the clusters are updated iteratively using the mean value.

Introduction to Gaussian Mixture Models (GMMs):

The Gaussian Mixture Model (GMM) is a probabilistic model used for clustering and density estimation. It assumes that the data is generated from a mixture of several Gaussian distributions, each representing a distinct cluster. GMM assigns probabilities to data points, allowing them to belong to multiple clusters simultaneously. The model is widely used in machine learning and pattern recognition applications.

Gaussian Mixture Models (GMMs) assume that there are a certain number of Gaussian distributions, and each of these distributions represent a cluster. Hence, a Gaussian Mixture Model tends to group the data points belonging to a single distribution together.

Let's say we have three Gaussian distributions (more on that in the next section) – GD1, GD2, and GD3. These have a certain mean ($\mu1$, $\mu2$, $\mu3$) and variance ($\sigma1$, $\sigma2$, $\sigma3$) value respectively. For a given set of data points, our GMM would identify the probability of each data point belonging to each of these distributions.Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters.

Here, we have three clusters that are denoted by three colors – Blue, Green, and Cyan. Let's take the data point highlighted in red. The probability of this point being a part

of the blue cluster is 1, while the probability of it being a part of the green or cyan clusters is 0.

Characteristics of the Normal or Gaussian Distribution

Characteristics of the normal or Gaussian distribution:

• It's bell-shaped with most values around the average.

• It has only one peak or mode.

• It stretches out forever in both directions.

• Its mean, median, and mode are the same.

• Its spread is measured by its standard deviation.

• The total area under its curve equals 1.

What is Expectation-Maximization?

Expectation-Maximization (EM) is a statistical algorithm for finding the right model parameters. We typically use EM when the data has missing values, or in other words, when the data is incomplete.

These missing variables are called latent variables. We consider the target (or cluster number) to be unknown when we're working on an unsupervised learning problem.

It's difficult to determine the right model parameters due to these missing variables. Think of it this way – if you knew which data point belongs to which cluster, you would easily be able to determine the mean vector and covariance matrix.

Since we do not have the values for the latent variables, Expectation-Maximization tries to use the existing data to determine the optimum values for these variables and then finds the model parameters. Based on these model parameters, we go back and update the values for the latent variable, and so on.

Broadly, the Expectation-Maximization algorithm has two steps:

• E-step: In this step, the available data is used to estimate (guess) the values of the missing variables

• M-step: Based on the estimated values generated in the E-step, the complete data is used

to update the parameters

Expectation-Maximization is the base of many algorithms, including Gaussian Mixture Models.

Expectation-Maximization in Gaussian Mixture Models:

Let's say we need to assign k number of clusters. This means that there are k Gaussian distributions, with the mean and covariance values to be μ1, μ2, .. μk and Σ1, Σ2, .. Σk . Additionally, there is another parameter for the distribution that defines the number of points for the distribution. Or in other words, the density of the distribution is represented with Πi.

PROGRAM:

```
import pandas as pd

import matplotlib.pyplot as plt

data=pd.read_csv(r"E:\Cognitive science\height.csv")

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

plt.scatter(data["Weight"],data["Height"])

plt.xlabel('Weight')

plt.ylabel('Height')

plt.title('Data Distribution')

plt.show()

from sklearn.cluster import KMeans

kmeans=KMeans(n_clusters=4)

kmeans.fit(data)

pred=kmeans.predict(data)

frame=pd.DataFrame(data)

frame['cluster']=pred

frame.columns=['Weight','Height','cluster']
```

```
color=['blue','green','cyan','black']
for k in range(0,4):
    data=frame[frame["cluster"]==k]
    plt.scatter(data["Weight"],data["Height"],c=color[k])
plt.show()
import pandas as pd
data=pd.read_csv(r"E:\Cognitive science\height.csv")
from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=4)
gmm.fit(data)
labels=gmm.predict(data)
frame=pd.DataFrame(data)
frame['cluster']=labels
frame.colors=["Weight","Height","cluster"]
color=["blue","green","cyan","black"]
for k in range(0,4):
```

Ex No: 09	DEVELOPING AN APPLICATION SYSTEM USING
Date:	CONDITIONAL LEARNING INFERENCE MODEL

AIM:

To write a program that implements the conditional learning inference model using the python.

INTRODUCTION:

Naive Bayes is one of the simplest classification machine learning algorithm. As the name suggests it's based on the Bayes theorem. Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

One of the most simple and effective classification algorithms, the Naïve Bayes classifier aids in the rapid development of machine learning models with rapid prediction capabilities.

Naïve Bayes algorithm is used for classification problems. It is highly used in text classification. In text classification tasks, data contains high dimension (as each word represent one feature in the data). It is used in spam filtering, sentiment detection, rating classification etc. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

- **Feature independence:** The features of the data are conditionally independent of each other, given the class label.
- **Continuous features are normally distributed:** If a feature is continuous, then it is assumed to be normally distributed within each class.

- **Discrete features have multinomial distributions:** If a feature is discrete, then it is assumed to have a multinomial distribution within each class.

- **Features are equally important:** All features are assumed to contribute equally to the prediction of the class label.

- **No missing data:** The data should not contain any missing values.

BAYES THEOREM:

The Bayes theorem can be understood as the description of the probability of any event which is obtained by prior knowledge about the event. We can say that Bayes' theorem can be used to describe the conditional probability of any event where we have data about the event and also we have prior information about the event with the prior knowledge about the conditions related to the event. For example, when we talk about statistical models such as **logistic regression** we can consider them as the **probability distribution** for the given data, and for this model, we can use the Bayes theorem to estimate the parameters of the model. Since the probabilities in the Bayesian statistics can be treated as a degree of belief, we can directly assign the probability distribution that can generate the belief to the parameter (or set of parameters) by quantifying them using the Bayes theorem. Mathematically the Bayes theorem can be expressed as

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Where A and B are two events

In this equation, using Bayes theorem, we can find the probability of A, given that B occurred. A is the hypothesis, and B is the evidence.

P(B|A) is the probability of B given that A is True.

P(A) and P(B) are the independent probabilities of A and B.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.

- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).

- P(B) is Marginal Probability: Probability of Evidence.

- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

- P(B|A) is Likelihood probability i.e the likelihood that a hypothesis will come true based on the evidence.

PRIOR PROBABILITY:

In the formula of the Bayes theorem, P(A) is the prior probability of A. We can define the prior probability or prior as the probability distribution of any uncertain quantity which is an expression of one's belief about uncertain quantity without or before takingany evidence into account.

LIKELIHOOD FUNCTION:

In the formula of Bayes theorem, P(B|A) is the likelihood function which can be simply called the likelihood which can be defined as the parameter functions of any statistical model which helps in describing the joint probability of the observed data. It can be more simplified as the probability of B when it is known that A is true. Numerically it is the support value where the support to proposition A is given by evidence B.

POSTERIOR PROBABILITY

In the formula of the Bayes theorem, P(B|A) is a posterior probability that can be defined as the conditional probability of any random event or uncertain proposition when there is knowledge about the relevant evidence that is present and taken into account. According to the Bayes theorem probability formula, this value represents the probability of A when evidence B has been taken into account.

From the above intuitions and definitions, we can say that the Bayes theorem can be written as the posterior probability will be equal to the product of likelihood probability, prior probability, and the inverse of the probability of evidence(B) being true.

Applications of Naive Bayes Classifier

- **Spam Email Filtering**: Classifies emails as spam or non-spam based on features.
- **Text Classification**: Used in sentiment analysis, document categorization, and topic classification.
- **Medical Diagnosis:** Helps in predicting the likelihood of a disease based on symptoms.
- **Credit Scoring:** Evaluates creditworthiness of individuals for loan approval.
- **Weather Prediction**: Classifies weather conditions based on various factors.

PROGRAM:

```
import pandas as pd

data=pd.read_csv(r'E:\Cognitive science\Naive.csv')

prior=data.groupby('Buys_Computer').size().div(len(data))

print(prior)

likelihood={}

likelihood['Credit_Rating']=data.groupby(['Buys_Computer','Credit_Rating']).size().div(len(data)).div(prior)

likelihood['Age']=data.groupby(['Buys_Computer','Age']).size().div(len(data)).div(prior)

likelihood['Income']=data.groupby(['Buys_Computer','Income']).size().div(len(data)).div(prior)

likelihood['Student']=data.groupby(['Buys_Computer','Student']).size().div(len(data)).div(prior)

print(likelihood)

p_yes=likelihood['Age']['yes']['<=30'] * likelihood['Income']['yes']['medium'] * \

    likelihood['Student']['yes']['yes'] * likelihood['Credit_Rating']['yes']['fair']\

    *prior['yes']

p_no=likelihood['Age']['no']['<=30'] * likelihood['Income']['no']['medium'] * \

    likelihood['Student']['no']['yes'] * likelihood['Credit_Rating']['no']['fair']\

        * prior['no']

print('Yes:',p_yes)

print('No:',p_no)

from sklearn.preprocessing import LabelEncoder

encoded_data=data.apply(LabelEncoder().fit_transform)

from sklearn.naive_bayes import MultinomialNB

import numpy as np

clf=MultinomialNB()
```

```
clf.fit(encoded_data.drop(['Buys_Computer'],axis=1),encoded_data['Buys_Computer'])
X = np.array([1, 2, 1, 1]).reshape(1, -1)
print("Highest value is chosen and it corresponds to buying a computer")
prediction = clf.predict(X)
print("Prediction of:", prediction)
```

OUTPUT:

```
Buys_Computer
no     0.357143
yes    0.642857
dtype: float64
{'Credit_Rating': Buys_Computer  Credit_Rating
no            excellent        0.600000
              fair             0.400000
yes           excellent        0.333333
              fair             0.666667
dtype: float64, 'Age': Buys_Computer  Age
no            <=30     0.600000
              >40      0.400000
yes           31-40    0.444444
              <=30     0.222222
              >40      0.333333
dtype: float64, 'Income': Buys_Computer  Income
no            high      0.400000
              low       0.200000
              medium    0.400000
yes           high      0.222222
```

low 0.333333

medium 0.444444

dtype: float64, 'Student': Buys_Computer Student

no no 0.800000

yes 0.200000

yes no 0.333333

yes 0.666667

dtype: float64}

Yes: 0.028218694885361544

No: 0.006857142857142855

Highest value is chosen and it corresponds to buying a computer

Prediction of: [1]

RESULT:

Thus the model for conditional inference learning using Naive Bayes classifier algorithm as a supervised learning technique is successfully implemented.

Ex No: 10	DEVELOPING AN APPLICATION SYSTEM USING
Date:	HIERARCHICAL CLUSTERING MODEL

AIM:

To write a program that implements the hierarchical clustering model using the python.

INTRODUCTION:

Hierarchical clustering is a powerful technique in the realm of data analysis and pattern recognition, offering a nuanced understanding of the relationships within datasets. This comprehensive guide delves into the intricacies of hierarchical clustering, specifically tailored for implementation in Python.

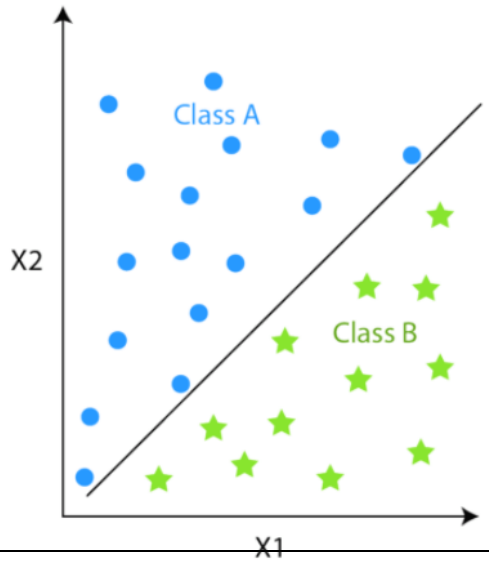
What is hierarchical clustering?

Hierarchical clustering is a technique in unsupervised machine learning that involves the organization of data into a hierarchy of nested clusters. Unlike other clustering methods, hierarchical clustering creates a tree-like structure of clusters (dendrogram), which visually represents the relationships between data points.

Difference between clustering and classification

Both classification and clustering try to group the data points into one or moreclasses based on the similarity of various features. The difference lies in the way both works.Classification is a supervised algorithm, where there are predefined labels (yi) assigned to each input data point (Xi).

Whereas, clustering is an unsupervised algorithm where labels are missing meaning the dataset contains only input data points (Xi).



The other major difference is since classification techniques have labels, there is a need for training and test datasets to verify the model. In clustering, there are no labels so there is no need for training and test datasets.

Popular examples of classification algorithms are:

1. Logistic Regression
2. Support Vector Classifier
3. Naive Bayes
4. Decision Trees
5. Random Forest
6. Neural Networks

Examples of clustering algorithms are:

1. Hierarchical clustering
2. K-Means Clustering
3. Mean Shift Clustering
4. Spectral Clustering

HIERARCHICAL CLUSTERING:

Group data into hierarchical clusters, forming a tree-like structure (dendrogram). For instance, hierarchical clustering could help create a hierarchy of stocks based on their correlation, indicating how closely related they are.

DENDROGRAM:

A dendrogram is a tree-like diagram that shows the hierarchical relationship between the observations. It contains the memory of hierarchical clustering algorithms.

Types of hierarchical clustering

There are two types of hierarchical clustering:

- Agglomerative hierarchical clustering
- Divisive hierarchical clustering

PROGRAM:

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()                    # Import iris data
iris_data = pd.DataFrame(iris.data)
iris_data.columns = iris.feature_names
iris_data['flower_type'] = iris.target
print(iris_data.head())
iris_X = iris_data.iloc[:, [0, 1, 2,3]].values
iris_Y = iris_data.iloc[:,4].values
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 7))
plt.scatter(iris_X[iris_Y == 0, 0], iris_X[iris_Y == 0, 1], s=100, c='blue', label='Type 1')
plt.scatter(iris_X[iris_Y == 1, 0], iris_X[iris_Y == 1, 1], s=100, c='yellow', label='Type 2')
plt.scatter(iris_X[iris_Y == 2, 0], iris_X[iris_Y == 2, 1], s=100, c='green', label='Type 3')
plt.legend()
plt.xlabel('Sample Index')
plt.ylabel('Euclidean Distance')
plt.show()
import scipy.cluster.hierarchy as sc
```

```
plt.figure(figsize=(20, 7))              # Plot dendrogram
plt.title("Dendrograms")
sc.dendrogram(sc.linkage(iris_X, method='ward'))          # Create dendrogram
plt.title('Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Euclidean distance')
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=3, linkage='ward')
cluster.fit(iris_X)
labels = cluster.labels_
print(labels)
plt.figure(figsize=(10, 7))
plt.scatter(iris_X[labels == 0, 0], iris_X[labels == 0, 1], s = 100, c = 'blue', label = 'Type
1')
plt.scatter(iris_X[labels == 1, 0], iris_X[labels == 1, 1], s = 100, c = 'yellow', label = 'Type
2')
plt.scatter(iris_X[labels == 2, 0], iris_X[labels == 2, 1], s = 100, c = 'green', label = 'Type
3')
plt.legend()
plt.xlabel('Sample Index')
plt.ylabel('Euclidean Distance')
plt.show()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower_type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

RESULT:

Thus the program that implements the hierarchical clustering model is developed and the output is verified successfully.