Laboratory Journal for

Data Science Lab (ITL701)

BE(IT) / VII / ITL701:
Data Science Lab

Journal Submitted by:
Name: Abhay Singh
Roll No: 54

Department of Information Technology

The Bombay Salesian Society's
Don Bosco Institute of Technology, Mumbai 400070
(Affiliated to the University of Mumbai)

Academic Year 2024-2025

Sr. No	Module	Detailed Content	Date	Page No.
I	Uncertainty in AI	1) Implement Inferencing with Bayesian Network in Python		3
II	Cognitive Computing	2) Building a Cognitive Healthcare application 3) Smarter cities: Cognitive Computing in Government 4) Cognitive computing in Insurance 5) Cognitive computing in Customer Service		7
III	Fuzzy Logic & Its Applications	6) Implementation of Fuzzy Membership Functions. 7) Implementation of fuzzy set Properties. 8.Design of a Fuzzy control system using Fuzzy tool		13
IV	Introduction to Deep Learning	8) Implementing Deep Learning Applications like : a. Image Classification System b. Handwritten Digit Recognition System (like MNIST Dataset) c. Traffic Signs Recognition System. d. Image Caption Generator		17
V	Advanced ML Classification Techniques	9) Implementation of supervised learning algorithm like a. Ada-Boosting b. Random forests 10) Evaluation of Classification Algorithms.		25
VI	Mini-project on trends and applications in Data Science	11) Build text/ image/ video/ audio based DS Applications such as a. Chatbot b. Document Classification c. Sentiment Analysis d. Bounding Box Detection e. Music/Video Genre Classification		29

Experiment 01:

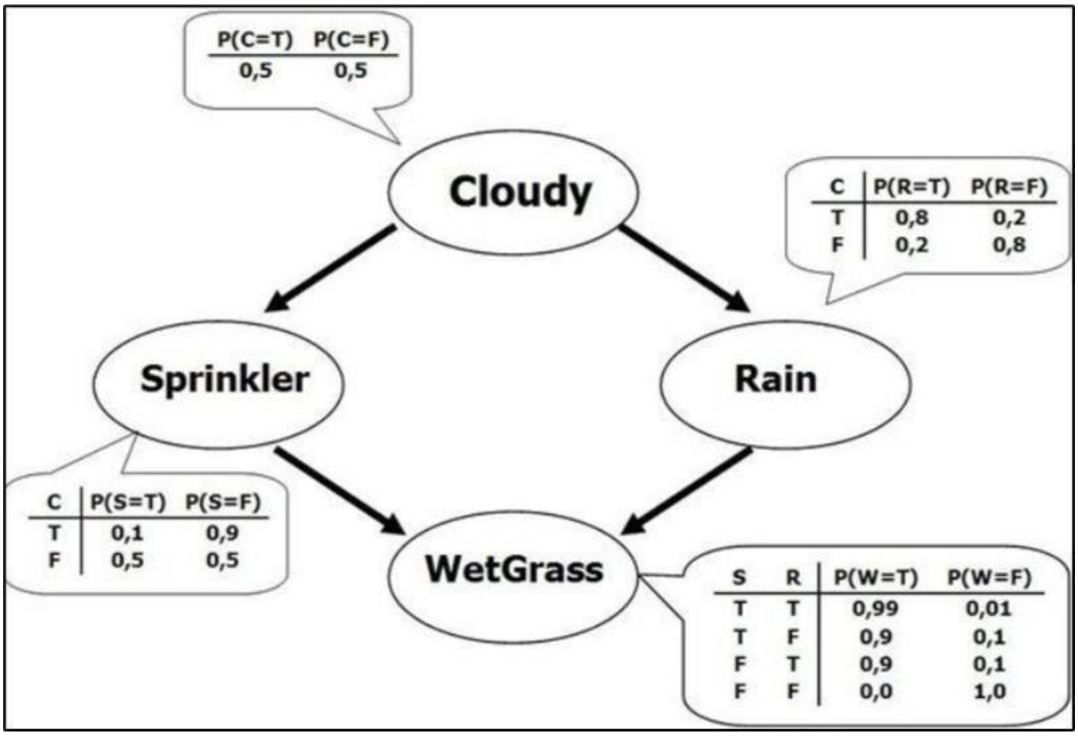
Uncertainty in AI

Aim: Implement Inferencing with Bayesian Network in Python

Theory:

A Bayesian network, often referred to as a Bayes network or belief network, is a powerful probabilistic graphical model used in data science and artificial intelligence. It consists of two main components: a directed acyclic graph (DAG) and conditional probability tables. The DAG visually represents a set of variables as nodes and their relationships as directed edges. These edges signify conditional dependencies, meaning the probability of a variable depends on its parents in the graph. Each node is associated with a conditional probability table that quantifies how a variable's value is influenced by the values of its parent nodes. This structure allows Bayesian networks to model complex relationships and make probabilistic inferences..

Bayesian networks find applications in various fields, including medical diagnosis, natural language processing, and decision-making under uncertainty. They enable reasoning about uncertainty and provide a systematic way to update beliefs as new information becomes available. By representing dependencies between variables, Bayesian networks help in making informed predictions, classifications, and decisions, making them a valuable tool for data analysis and machine learning tasks.



Conditional Probability:

Conditional probability is a measure of the likelihood of an event occurring provided that another event has already occurred (through assumption, supposition, statement, or evidence). If A is the event of interest and B is known or considered to have occurred, the conditional probability of A given B is generally stated as P(A|B) or, less frequently, PB(A) if A is the event of interest and B is known or thought to have occurred. This can also be expressed as a percentage of the likelihood of B crossing with A:

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

Joint Probability:

The chance of two (or more) events together is known as the joint probability. The sum of the probabilities of two or more random variables is the joint probability distribution.

For example, the joint probability of events A and B is expressed formally as:

- The letter P is the first letter of the alphabet (A and B).
- The upside-down capital "U" operator or, in some situations, a comma "," represents the "and" or conjunction.
- P(A ^ B)
- P(A, B)

By multiplying the chance of event A by the likelihood of event B, the combined probability for occurrences A and B is calculated.

Posterior Probability:

In Bayesian statistics, the conditional probability of a random occurrence or an ambiguous assertion is the conditional probability given the relevant data or background. "After taking into account the relevant evidence pertinent to the specific subject under consideration," "posterior" means in this case.

The probability distribution of an unknown quantity interpreted as a random variable based on data from an experiment or survey is known as the posterior probability distribution.

```
pip install pgmpy

Requirement already satisfied: pgmpy in /usr/local/lib/python3.10/dist-packages (0.1.26)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.2.2)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.4)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.4.1+cu121)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.5)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.4.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.4.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.1)
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.7.2)
Requirement already satisfied: google-ai-generativelanguage==0.6.6 in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (0.6.6)
```

```python
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

```python
# Define the Bayesian Network structure
model = BayesianNetwork([('Cloudy', 'Sprinkler'),
                         ('Cloudy', 'Rain'),
                         ('Sprinkler', 'WetGrass'),
                         ('Rain', 'WetGrass')])
```

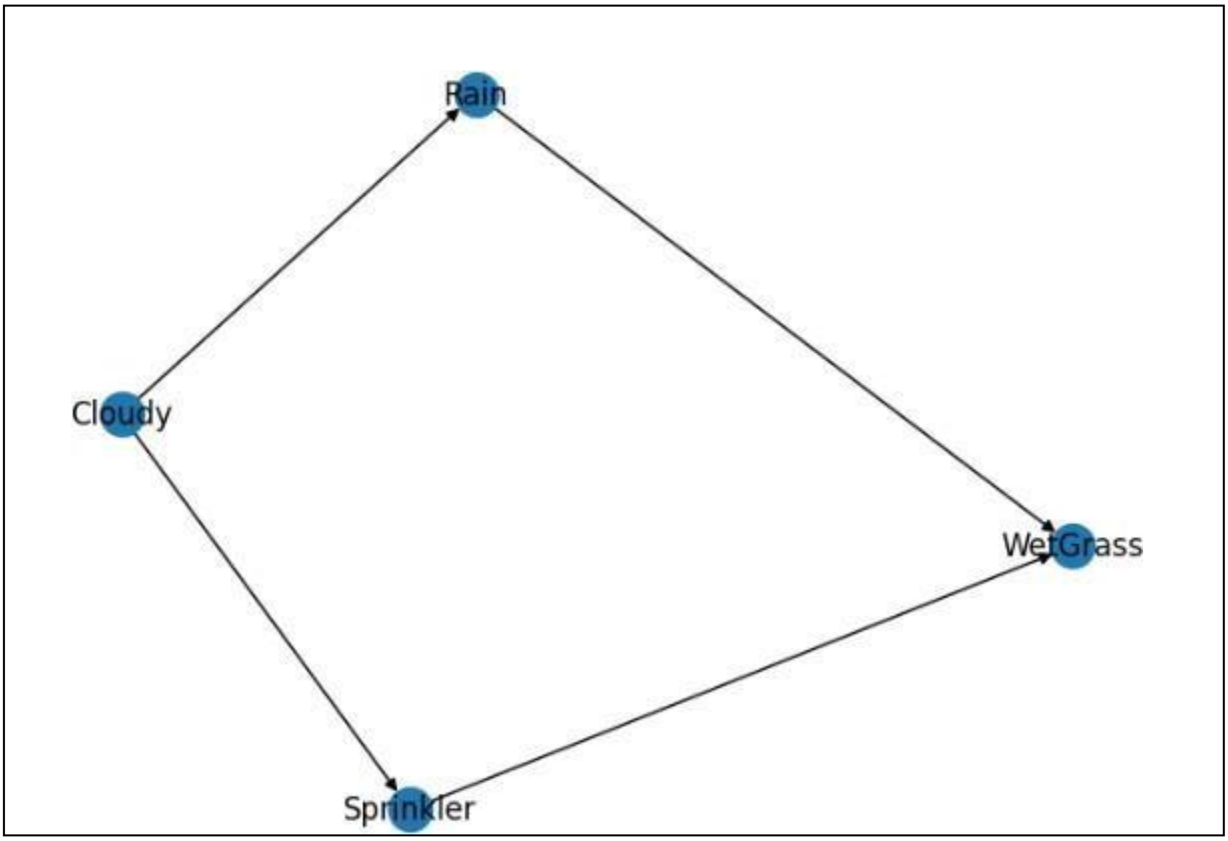
```python
# Define Conditional Probability Distributions (CPDs)
# CPD for Cloudy
cpd_cloudy = TabularCPD(variable='Cloudy', variable_card=2, values=[[0.5], [0.5]])

# CPD for Sprinkler given Cloudy
cpd_sprinkler = TabularCPD(variable='Sprinkler', variable_card=2,
                           values=[[0.5, 0.9], [0.5, 0.1]],
                           evidence=['Cloudy'], evidence_card=[2])

# CPD for Rain given Cloudy
cpd_rain = TabularCPD(variable='Rain', variable_card=2,
                      values=[[0.8, 0.2], [0.2, 0.8]],
                      evidence=['Cloudy'], evidence_card=[2])

# CPD for WetGrass given Sprinkler and Rain
cpd_wet_grass = TabularCPD(variable='WetGrass', variable_card=2,
                           values=[[1.0, 0.1, 0.1, 0.01], [0.0, 0.9, 0.9, 0.99]],
                           evidence=['Sprinkler', 'Rain'], evidence_card=[2, 2])

# Add CPDs to the model
model.add_cpds(cpd_cloudy, cpd_sprinkler, cpd_rain, cpd_wet_grass)

# Check if the model is valid
print("Is model valid? ", model.check_model())
```

```
Is model valid?  True
```

```python
# Perform inference using Variable Elimination
inference = VariableElimination(model)

# Query the probability of WetGrass being wet, given that it is cloudy
result = inference.query(variables=['WetGrass'], evidence={'Cloudy': 1})
print(result)
```

```
+-------------+-----------------+
| WetGrass    |   phi(WetGrass) |
+=============+=================+
| WetGrass(0) |          0.2548 |
+-------------+-----------------+
| WetGrass(1) |          0.7452 |
+-------------+-----------------+
```

```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a networkx graph
G = nx.DiGraph()

# Add nodes and edges
G.add_nodes_from(['Cloudy', 'Sprinkler', 'Rain', 'WetGrass'])
G.add_edges_from([('Cloudy', 'Sprinkler'), ('Cloudy', 'Rain'), ('Sprinkler', 'WetGrass'), ('Rain', 'WetGrass')

# Draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos=pos, with_labels=True)
plt.show()
```



References:
https://analyticsindiamag.com/a-guide-to-inferencing-with-bayesian-network-in-python/
https://gist.github.com/PurpleVen/42fa5bbbfff1186c8a9971f702ce8fbf
https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fanalyticsindiamag.com%2Fa-guide-to-inferencing-with-bayesian-network-in-python%2F

EXPERIMENT NO.: 2

Cognitive Computing

Aim: Building a Cognitive Healthcare application, Smarter cities: Cognitive Computing in Government, Cognitive computing in Insurance and Cognitive computing in Customer Service

Cognitive computing involves mimicking human thought processes using a computerized model. It leverages AI technologies like machine learning, natural language processing, and deep learning to enable computers to "think" and make decisions similarly to humans. These systems continuously learn from data and use insights to solve complex challenges, making them invaluable across numerous industries. This experiment explores how cognitive computing is reshaping sectors like healthcare, government, insurance, and customer service.

1. Developing a Cognitive Healthcare Application

The healthcare industry is undergoing a transformation due to cognitive computing, thanks to the vast amount of data it generates from patient records, medical histories, and real-time monitoring. Key areas where cognitive healthcare applications make a significant impact include:

- **Patient Data Analysis**
  AI processes large volumes of patient data faster than traditional methods, examining patient records, medical histories, diagnostic reports, and even genetic information to identify patterns and anomalies often missed by humans. This aids doctors in making better, more informed decisions.
  For example, AI analyzes medical images like MRIs or X-rays to detect early-stage diseases such as cancer. Additionally, analyzing electronic health records (EHRs) can help identify trends and speed up diagnoses.
- **Predictive Analytics**
  Cognitive computing supports predictive analytics, which forecasts health risks and outcomes based on historical and current data. By identifying patterns, AI can predict if a patient is at risk for conditions like diabetes, heart disease, or cancer. AI models can also identify patients likely to be readmitted and help healthcare providers take preventive action.
  Predictive analytics is also crucial for public health, as it forecasts disease outbreaks, helping authorities manage resources and plan effectively.
- **Personalized Treatment**
  Tailoring treatments for individual patients enhances outcomes. Cognitive computing considers various aspects of a patient's profile, including genetic data, lifestyle, and medical history, to suggest personalized treatment plans. This individualized approach ensures more effective and efficient care.
  Pharmaceutical companies also utilize cognitive systems to analyze vast datasets,

determining how different drugs affect specific patient groups and enabling the development of targeted and personalized medications.

- **Virtual Health Assistants**
  AI-powered virtual health assistants are increasingly used for managing patient inquiries, appointment scheduling, and health-related advice. These assistants leverage natural language processing (NLP) to understand and respond to patient questions in real-time, improving accessibility and reducing the workload on medical staff.
  For instance, patients can interact with virtual assistants to schedule appointments, ask about medications, or monitor recovery progress after surgery. These assistants can also send reminders for follow-ups and medication schedules.
- **Remote Monitoring**
  Wearable devices and mobile health apps collect real-time data on patient metrics like heart rate, activity levels, and glucose levels. Cognitive computing analyzes this data, alerting healthcare providers to any anomalies. Remote monitoring is especially valuable for managing chronic conditions like diabetes and hypertension or for elderly patients needing continuous supervision.
  AI algorithms track patient data from these devices, notifying doctors or caregivers when urgent action is necessary, enabling early intervention and improving patient outcomes.

2. Smarter Cities: Cognitive Computing in Government

Governments are implementing cognitive computing to create smarter cities, improving urban management, public safety, and citizens' quality of life.

- **Traffic Management**
  Cognitive computing systems analyze real-time traffic data from cameras, sensors, and GPS to optimize traffic flow and reduce congestion. Predictive algorithms anticipate peak traffic times, suggest alternate routes, or adjust traffic signal timings to ease congestion.
  For example, cities like Singapore and Los Angeles use AI-powered traffic management systems to mitigate traffic issues by adjusting traffic lights and providing real-time updates to commuters through mobile apps.
- **Public Safety**
  Governments use cognitive computing to analyze data from surveillance cameras, social media, and public records to detect crime patterns and predict where to deploy resources. Predictive analytics highlight high-risk areas for criminal activities, enabling proactive law enforcement.
  For instance, the Chicago Police Department uses AI models to predict crime hotspots and adjust patrol schedules, reducing crime rates.
- **Waste Management**
  AI monitors fill levels in garbage bins using IoT sensors and optimizes waste collection routes, reducing fuel consumption, labor costs, and environmental impact. Cognitive systems predict where and when garbage collection is needed, enhancing city operations' efficiency.
- **Citizen Engagement**
  Governments employ chatbots to provide round-the-clock support for public service

inquiries, such as tax filings, permit applications, or emergency reports. These systems handle a wide range of questions, enhancing efficiency and accessibility.
Estonia's government, for example, offers an AI chatbot that helps citizens with tax filings, healthcare inquiries, and legal services.

- **Energy Management**
  AI-driven energy management systems monitor electricity use in public buildings, optimizing heating, cooling, and lighting to reduce energy consumption and costs, contributing to environmental sustainability.
  Smart cities use AI to manage energy in public buildings, adjusting usage dynamically to save power during off-peak times.

3. Cognitive Computing in Insurance

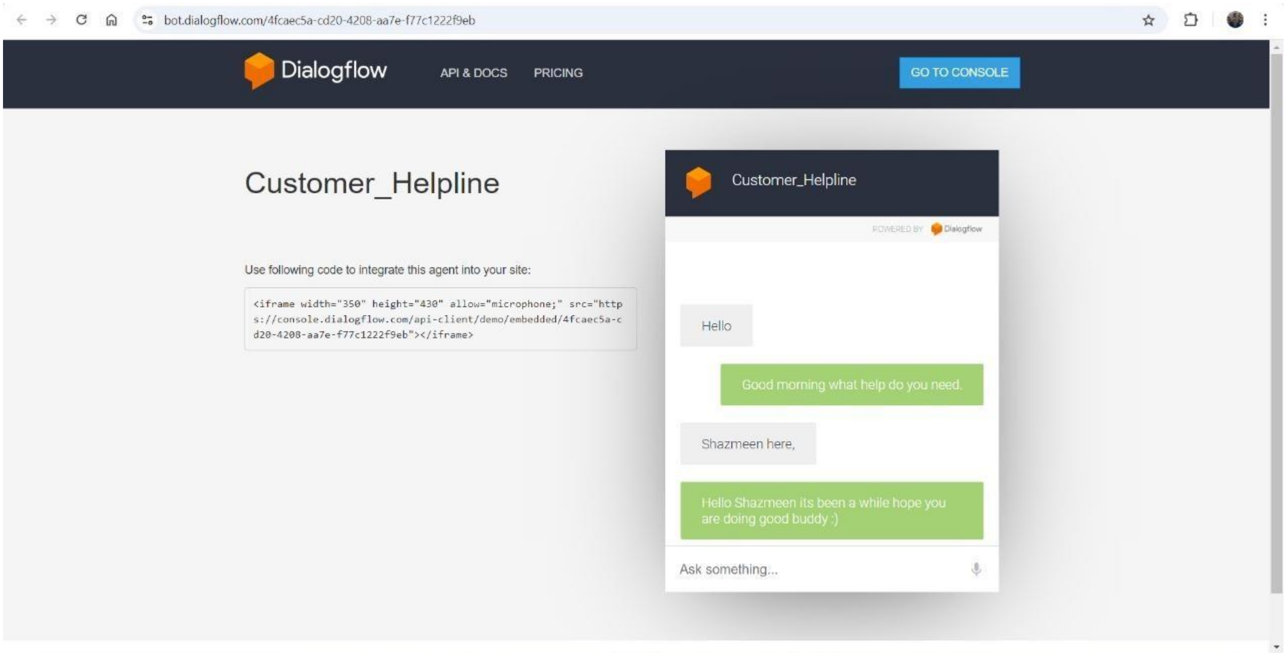
Insurance companies are leveraging cognitive computing to streamline operations, personalize services, and improve fraud detection.

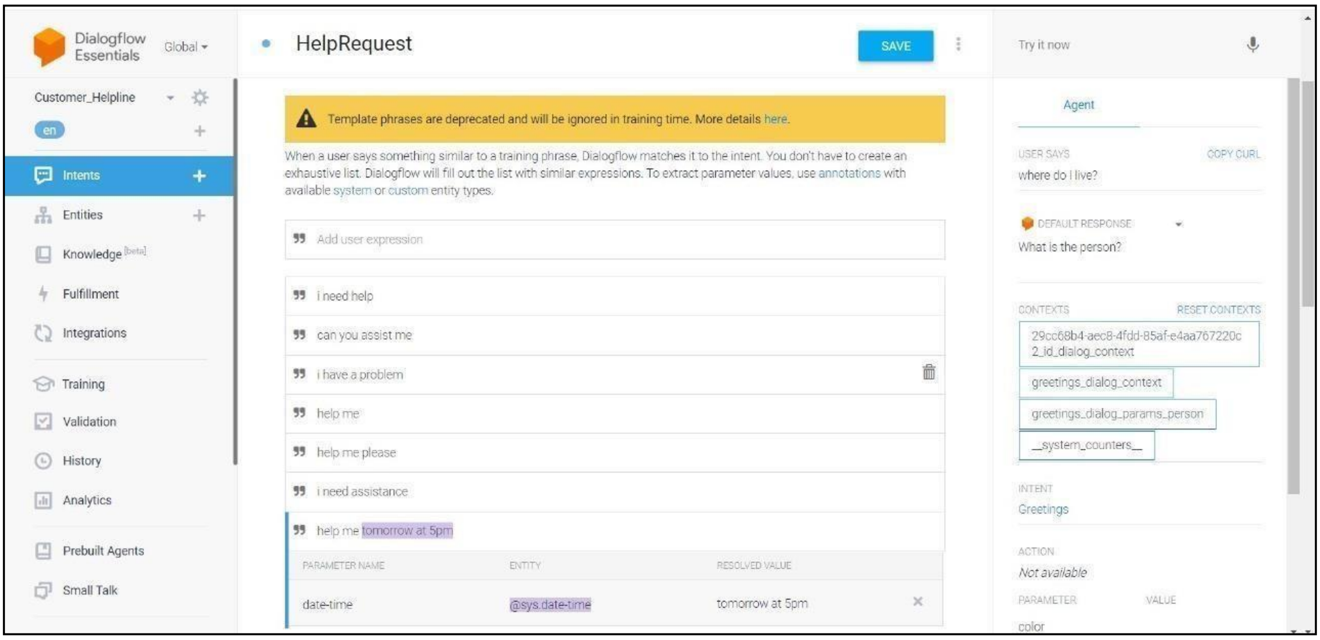
- **Fraud Detection**
  AI algorithms analyze claims data to spot patterns indicative of fraud. By cross-referencing claims data with historical trends and external data (like social media or transaction records), cognitive systems flag suspicious claims for further review.
  For example, AI can detect patterns such as multiple claims for similar accidents in a short period or discrepancies between reported injuries and vehicle damage.
- **Risk Assessment**
  Cognitive computing enhances risk assessment by utilizing big data and predictive models. Insurers evaluate risks more accurately by incorporating data such as driving behavior, health records, and environmental factors. Predictive analytics help set premium rates and estimate potential losses more precisely.
- **Customer Insights**
  Cognitive systems analyze customer data, including interactions and behavior, to create personalized insurance products and offers. This allows insurers to tailor policies to customers' needs, enhancing satisfaction and retention.
- **Claims Processing**
  AI automates claims processing, leading to faster and more efficient outcomes. Cognitive systems automatically assess claims, reducing human intervention, minimizing errors, and speeding up the process.
- **Chatbots**
  AI chatbots provide 24/7 support for insurance customers, answering questions, guiding them through the claims process, and handling policy inquiries, ensuring a quicker and more consistent customer experience.

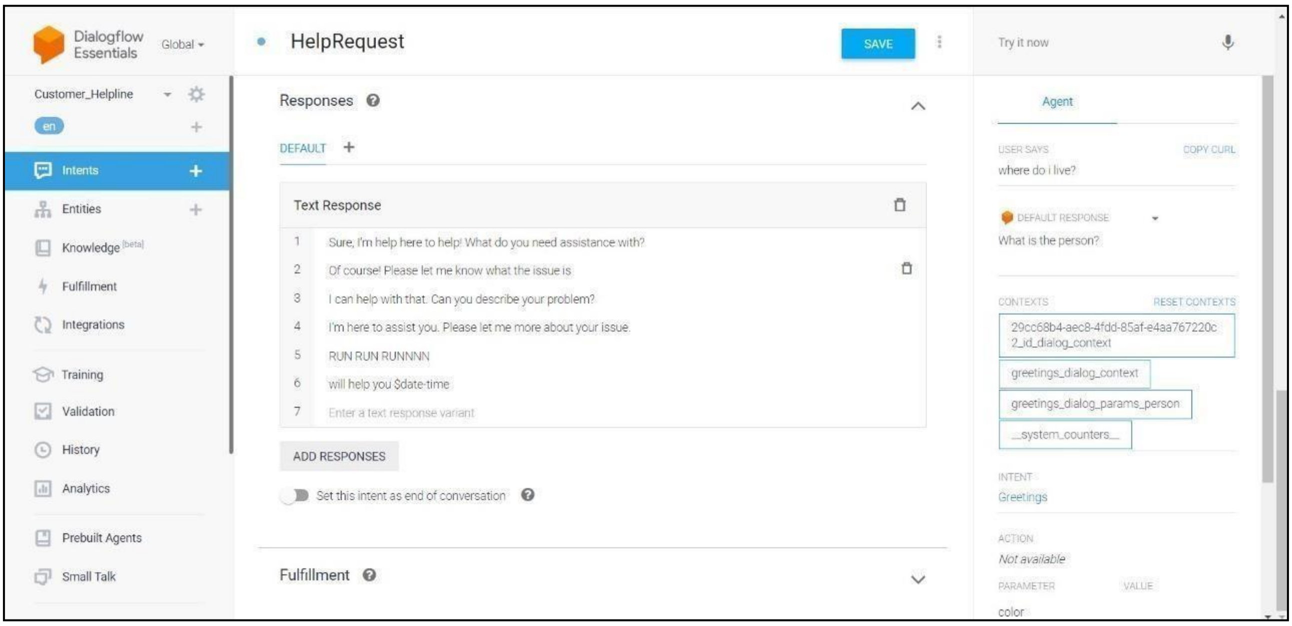
4. Cognitive Computing in Customer Service

Cognitive computing is transforming customer service by making interactions smarter, quicker, and more personalized.

- **Automated Support**
  AI chatbots handle routine customer inquiries, such as order status, billing issues, and product information, reducing the workload on human agents and ensuring faster responses. Companies like Amazon and H&M use AI-powered customer support systems that manage common inquiries around the clock, enhancing customer satisfaction and reducing waiting times.
- **Sentiment Analysis**
  AI systems analyze customer feedback from text, voice, or social media to gauge sentiment and identify areas for product or service improvement. Real-time analysis enables companies to respond quickly to customer concerns.
- **Personalization**
  AI customizes customer interactions based on previous behaviors, preferences, and purchase history, enhancing loyalty and improving the overall customer experience through targeted product recommendations or tailored solutions.
- **Predictive Support**
  Cognitive computing anticipates customer needs, offering proactive assistance. For instance, an AI system can notify customers of potential issues with their orders or suggest product replacements based on historical data.
- **Data Analytics**
  AI analyzes customer data to optimize service strategies, improve operations, and support data-driven decision-making. Cognitive systems identify trends in customer interactions, complaints, or product performance, enabling smarter business decisions.







Conclusion

Cognitive computing is revolutionizing various sectors by using AI to analyze vast data volumes, generate actionable insights, and improve decision-making. In healthcare, smart cities, insurance, and customer service, the ability to process and learn from real-time data allows organizations to enhance efficiency, increase customer satisfaction, and solve complex issues proactively. As these technologies evolve, their impact will only grow, driving innovation and transforming traditional processes.

References:

Cognitive Computing and its Applications - Great Learning (mygreatlearning.com)
https://magazine.wharton.upenn.edu/digital/cognitive-computing-in-health-care/
https://enterrasolutions.com/smart-cities-iot-cognitive-computing/
https://ieeexplore.ieee.org/document/7536499
https://www.spiceworks.com/tech/artificial-intelligence/articles/cognitive-computing-vs-ai/

Experiment 03: Fuzzy Logic & Its Applications

Aim: Implementation of Fuzzy Membership Functions, Implementation of fuzzy set Properties and Design of a Fuzzy control system using Fuzzy tool.

Theory:

Fuzzy logic is a mathematical logic that attempts to solve problems with an open, imprecise solution. Fuzzy logic is used in many applications, including control systems, expert systems, and artificial intelligence. The implementation of fuzzy membership functions and fuzzy set properties are the building blocks of fuzzy logic. A fuzzy control system is designed using fuzzy tools to control a physical system by adjusting the output of the system based on the input. The fuzzy control system is composed of a fuzzifier, a fuzzy rule base, a fuzzy knowledge base, an inference engine, and a defuzzifier.

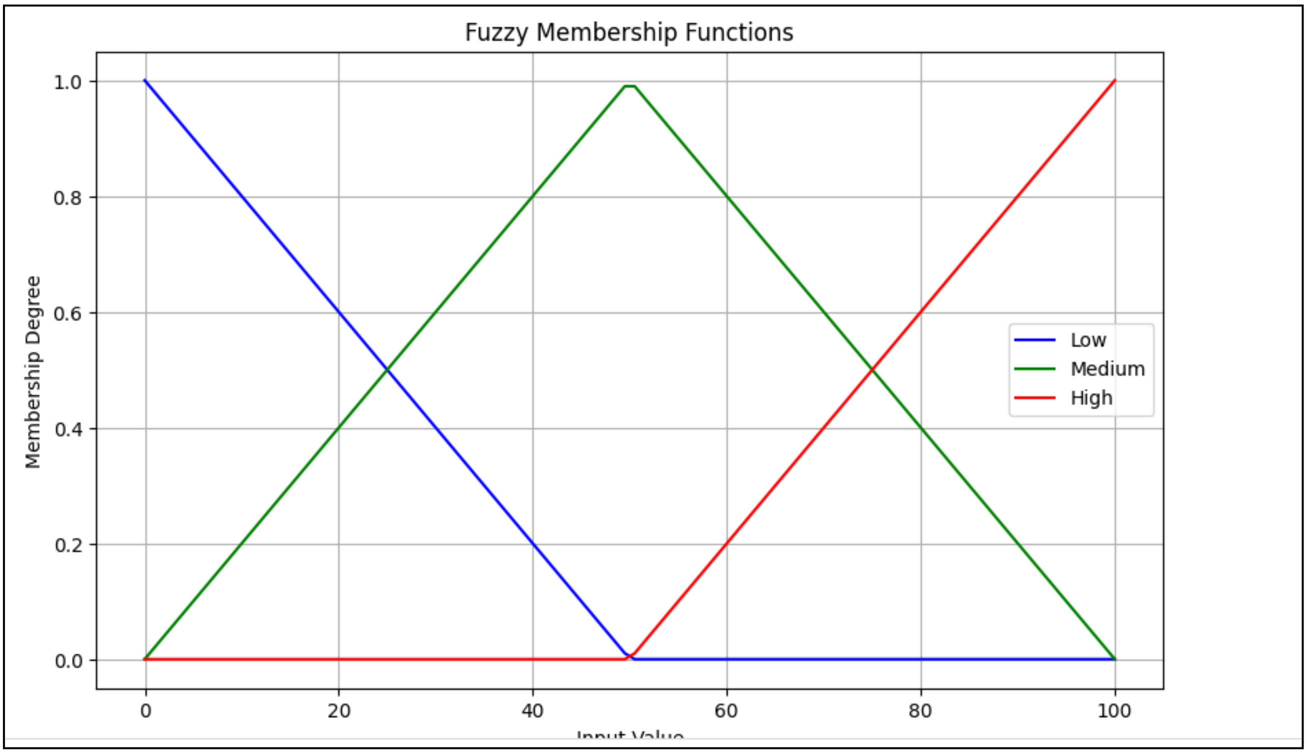
Code & Output:

1) Implementation of Fuzzy Membership Functions.

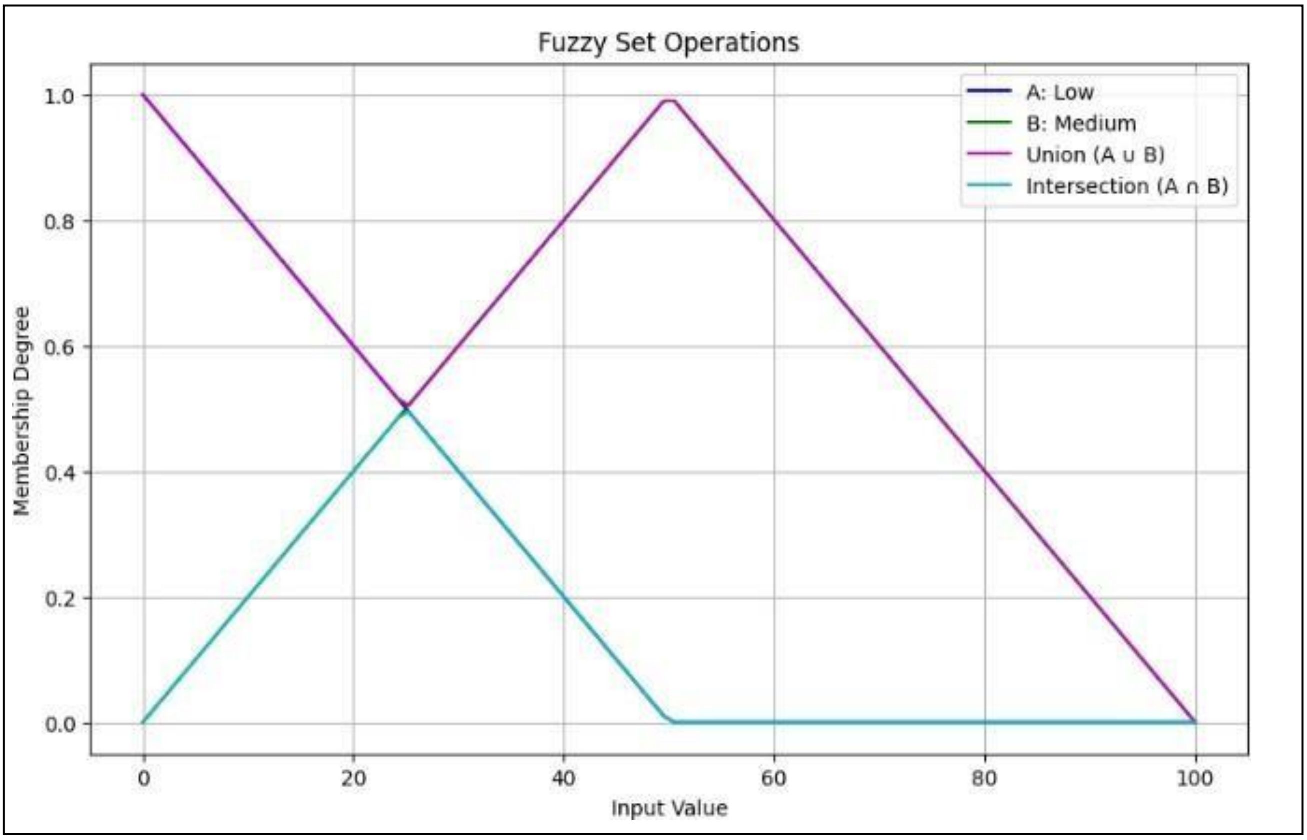
```
!pip install scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl.metadata (2.6 kB)
Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl (920 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 920.8/920.8 kB 7.8 MB/s eta 0:00:00
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.5.0
```

```python
#  Implementation of Fuzzy Membership Functions
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# Generate a range of values
x = np.linspace(0, 100, 100)

# Define fuzzy membership functions
low = fuzz.trimf(x, [0, 0, 50])
medium = fuzz.trimf(x, [0, 50, 100])
high = fuzz.trimf(x, [50, 100, 100])

# Plot the membership functions
plt.figure(figsize=(10, 6))
plt.plot(x, low, 'b', label='Low')
plt.plot(x, medium, 'g', label='Medium')
plt.plot(x, high, 'r', label='High')
plt.title('Fuzzy Membership Functions')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid()
plt.show()
```



```python
# Implementation of Fuzzy Set Properties
# Define fuzzy sets
A = low
B = medium

# Union and Intersection
union = np.maximum(A, B)
intersection = np.minimum(A, B)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(x, A, 'b', label='A: Low')
plt.plot(x, B, 'g', label='B: Medium')
plt.plot(x, union, 'm', label='Union (A ∪ B)')
plt.plot(x, intersection, 'c', label='Intersection (A ∩ B)')
plt.title('Fuzzy Set Operations')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid()
plt.show()
```



```python
import numpy as np
import skfuzzy as fuzz

# Define the fuzzy variables and their membership functions
temperature = np.linspace(0, 100, 100)
fan_speed = np.linspace(0, 100, 100)

# Membership functions for temperature
cold = fuzz.trimf(temperature, [0, 0, 50])
comfortable = fuzz.trimf(temperature, [0, 50, 100])
hot = fuzz.trimf(temperature, [50, 100, 100])

# Membership functions for fan speed
low_speed = fuzz.trimf(fan_speed, [0, 0, 50])
medium_speed = fuzz.trimf(fan_speed, [0, 50, 100])
high_speed = fuzz.trimf(fan_speed, [50, 100, 100])

# Fuzzification (example input)
input_temp = 75
cold_level = fuzz.interp_membership(temperature, cold, input_temp)
comfortable_level = fuzz.interp_membership(temperature, comfortable, input_temp)
hot_level = fuzz.interp_membership(temperature, hot, input_temp)

# Initialize output membership degrees
fan_speed_low = cold_level  # Low speed if the temperature is cold
fan_speed_medium = comfortable_level  # Medium speed if the temperature is comfortable
fan_speed_high = hot_level  # High speed if the temperature is hot

# Step 5: Aggregate the output membership functions
aggregated_fan_speed = np.fmax(np.fmax(low_speed * fan_speed_low, medium_speed * fan_speed_medium), high_speed * fan_speed_high)

# Defuzzification (Crisp Output)
defuzzified_output = fuzz.defuzz(fan_speed, aggregated_fan_speed, 'centroid')

print(f'Fuzzified Fan Speed Output: {defuzzified_output:.2f}%')
```

```
Fuzzified Fan Speed Output: 58.34%
```

Conclusion:

In conclusion, the implementation of fuzzy membership functions, fuzzy set properties, and the design of a fuzzy control system using fuzzy tools can help control physical systems by adjusting the output of the system based on the input.

References:

[Fuzzy Logic Control System - GeeksforGeeks]()
[Fuzzy Logic Toolbox - MATLAB (mathworks.com)]()
https://gist.github.com/PurpleVen/43120a6cc158bd0ab63b8cf51914df06
https://gist.github.com/PurpleVen/fba72a70128fda58679a600a9911fe0a
https://gist.github.com/PurpleVen/e8ca39476b59f2a927d2eb16a33b2e43

Experiment 04: Introduction to Deep Learning

Aim: Implementing Deep Learning Applications like
a. Image Classification System
b. Handwritten Digit Recognition System (like MNIST Dataset)
c. Traffic Signs Recognition System.
d. Image Caption Generator

Theory:

Deep learning is a branch of machine learning that is based on artificial neural network architecture. It uses artificial neural networks to learn from lots of data without needing explicit programming. Deep learning can be used for things like recognizing images, understanding speech, and processing language. To implement deep learning, one can follow these steps: learn machine learning basics, start learning Python, choose a deep learning framework, learn neural network basics, practice with toy datasets, and work on real-world projects. To create an image classification system, handwritten digit recognition system, traffic signs recognition system, or image caption generator, one can use deep learning techniques such as convolutional neural networks (CNNs) and long short-term memory (LSTM) units.

Image recognition refers to the task of inputting an image into a neural network and having it output some kind of label for that image. The label that the network outputs will correspond to a predefined class. There can be multiple classes that the image can be labeled as, or just one. If there is a single class, the term "recognition" is often applied, whereas a multi-class recognition task is often called "classification".
A subset of image classification is object detection, where specific instances of objects are identified as belonging to a certain class like animals, cars, or people.

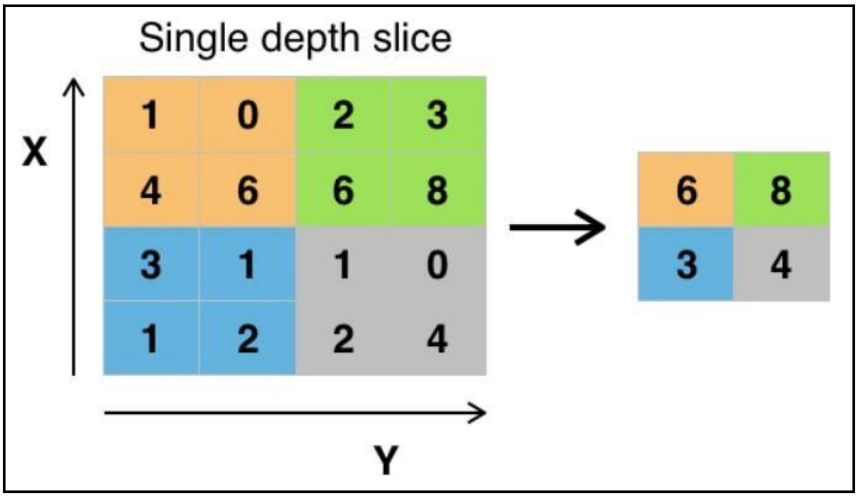
Code & Output:
a. Image Classification System

Theory:
In order to carry out image recognition/classification, the neural network must carry out feature extraction. Features are the elements of the data that you care about which will be fed through the network. In the specific case of image recognition, the features are the groups of pixels, like edges and points, of an object that the network will analyze for patterns.
Feature recognition (or feature extraction) is the process of pulling the relevant features out from an input image so that these features can be analyzed. Many images contain annotations or metadata about the image that helps the network find the relevant features.

Activation Functions: After the feature map of the image has been created, the values that represent the image are passed through an activation function or activation layer. The activation function takes values that represent the image, which are in a linear form (i.e. just a list of numbers) thanks to the convolutional layer, and increases their non-linearity since images themselves are non-linear.

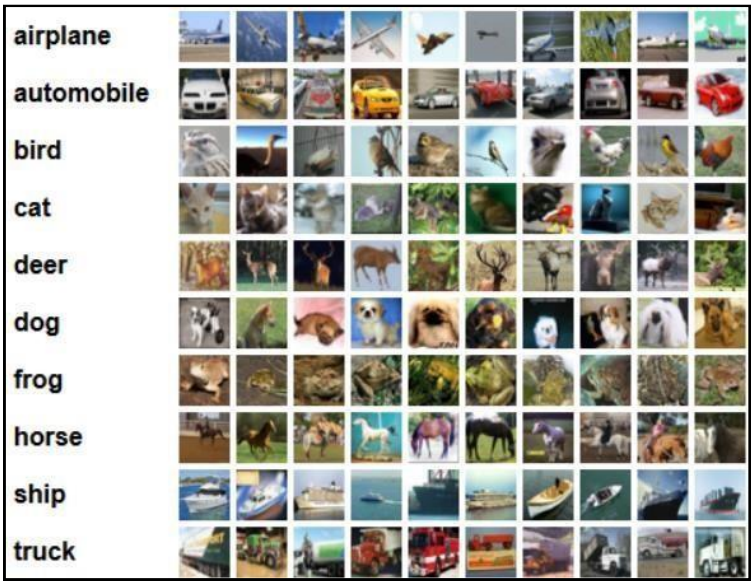
Pooling Layers : After the data is activated, it is sent through a pooling layer. Pooling "down-samples" an image, meaning that it takes the information which represents the image and compresses it, making it smaller. The pooling process makes the network more flexible and more adept at recognizing objects/images based on the relevant features. When we look at an image, we typically aren't concerned with all the information in the background of the image, only the features we care about, such as people or animals.

Similarly, a pooling layer in CNN will abstract away the unnecessary parts of the image, keeping only the parts of the image it thinks are relevant, as controlled by the specified size of the pooling layer. Because it has to make decisions about the most relevant parts of the image, the hope is that the network will learn only the parts of the image that truly represent the object in question. This helps prevent overfitting, where the network learns aspects of the training case too well and fails to generalize to new data.



Flattening : The final layers of our CNN, the densely connected layers, require that the data is in the form of a vector to be processed. For this reason, the data must be "flattened". The values are compressed into a long vector or a column of sequentially ordered numbers.

To begin with, we'll need a dataset to train on. In this example, we will be using the famous CIFAR-10 dataset. CIFAR-10 is a large image dataset containing over 60,000 images representing 10 different classes of objects like cats, planes, and cars. The images are full-color RGB, but they are fairly small, only 32 x 32. One great thing about the CIFAR-10 dataset is that it comes prepackaged with Keras, so it is very easy to load up the dataset and the images need very little preprocessing.



b. Handwritten Digit Recognition System (like MNIST Dataset)

Theory:

Dataset Description: The MNIST dataset, created by Yann LeCun, Corinna Cortes, and Christopher Burges, is a benchmark dataset for evaluating machine learning models on handwritten digit classification. It is derived from scanned document datasets available from the National Institute of Standards and Technology (NIST), leading to the name Modified NIST or MNIST dataset.
Demonstration Steps:

- Loading the MNIST Dataset in Keras: We'll start by showing how to load the MNIST dataset using the Keras library, which provides an easy and convenient way to access the dataset.
- Developing and Evaluating a Baseline Neural Network Model: We'll walk through the process of building a basic neural network model for the MNIST problem. This serves as a foundational model to establish a performance baseline.
- Implementing and Evaluating a Simple Convolutional Neural Network (CNN): We'll delve into the application of Convolutional Neural Networks, a powerful architecture for image classification, to the MNIST dataset. We'll design a simple CNN and evaluate its performance.
- Implementing a Close to State-of-the-Art Deep Learning Model: To push the performance boundaries, we'll demonstrate the implementation of a more advanced deep learning model for MNIST, aiming to achieve state-of-the-art results. This model will leverage advanced techniques and architectural innovations to maximize accuracy.

c. Traffic Signs Recognition System.

Theory:

A Traffic Sign Recognition System (TSRS) is a technology using computer vision and deep learning to detect and recognize traffic signs in images or videos. Its main goal is to assist drivers in obeying traffic regulations by providing real-time information. TSRS is essential in advanced driver assistance systems (ADAS) and autonomous vehicles. The process involves:

- Traffic Sign Detection: Identify traffic sign positions in images or video frames using object detection or region proposal techniques.
- Traffic Sign Classification: Classify detected signs into specific categories like speed limits or stop signs using image classification techniques.
- Decision-Making and Alerts: Based on classification results, the TSRS triggers actions or alerts the driver or vehicle control system, such as warning about speed limits or suggesting lane changes.

Step 1: Dataset Collection (GTSRB - German Traffic Sign Recognition Benchmark)

Obtain the "GTSRB - German Traffic Sign Recognition Benchmark" dataset from Kaggle, which includes labeled images of German traffic signs with diverse conditions.

Step 2: Data Preprocessing

Preprocess images by resizing them to a consistent size, normalizing pixel values, and applying data augmentation to ensure the model can handle real-world variations.

Step 3: Model Architecture

Design a Convolutional Neural Network (CNN) using Keras, typically with convolutional and fully connected layers. The final layer should have units equal to unique traffic sign classes, using softmax activation for classification.

Step 4: Model Training

Split the dataset into training and validation sets. Train the CNN using categorical cross-entropy loss and an optimization algorithm like Adam. Monitor performance on the validation set and apply techniques like early stopping to prevent overfitting.

Step 5: Model Evaluation

Evaluate the model's performance on a separate test dataset that it hasn't seen during training to provide an unbiased assessment of its ability to recognize traffic signs.

d. Image Caption Generator

Theory :

Image captioning is a very classical and challenging problem coming to the Deep Learning domain, in which we generate the textual description of an image using its property, but we will not use Deep learning here. In this article, we will simply learn how to simply caption the images using PIL. Preprocessing on images is a great utility provided by the Python PIL library. Not only can we change size, mode, orientation but we can draw on images, write text over it as well, Install the required libraries.

```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

```python
# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

# Normalize the pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Reshape the images to add a channel dimension (28x28x1)
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```python
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes for digits 0-9
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: U
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 62s 32ms/step - accuracy: 0.8950 - loss: 0.3320 - val_accuracy: 0.9860 - val_loss: 0.0453
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9845 - loss: 0.0500 - val_accuracy: 0.9845 - val_loss: 0.0453
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 27ms/step - accuracy: 0.9898 - loss: 0.0313 - val_accuracy: 0.9886 - val_loss: 0.0352
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 28ms/step - accuracy: 0.9927 - loss: 0.0233 - val_accuracy: 0.9895 - val_loss: 0.0312
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 53s 28ms/step - accuracy: 0.9945 - loss: 0.0163 - val_accuracy: 0.9898 - val_loss: 0.0315
<keras.src.callbacks.history.History at 0x79fd1f2a8fa0>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

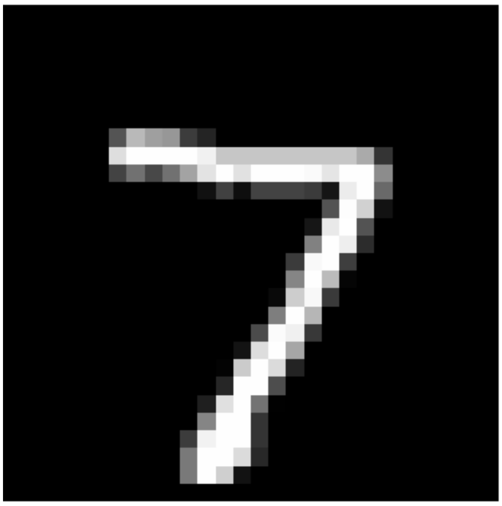
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9868 - loss: 0.0395
Test accuracy: 0.989799976348877
```

```
predictions = model.predict(test_images)

# Example: Print the prediction for the first image
print(f'Predicted label: {predictions[0].argmax()}')
print(f'Actual label: {test_labels[0]}')
```

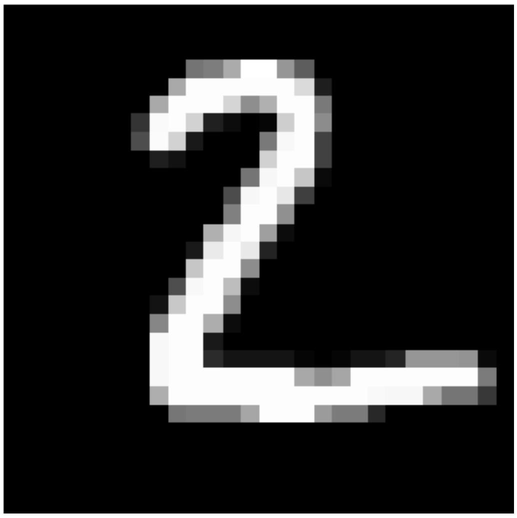
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
Predicted label: 7
Actual label: 7
```

```
# Plot the first 5 test images and their predicted labels
for i in range(5):
    plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
    plt.title(f'Predicted: {predictions[i].argmax()}, Actual: {test_labels[i]}')
    plt.axis('off')
    plt.show()
```

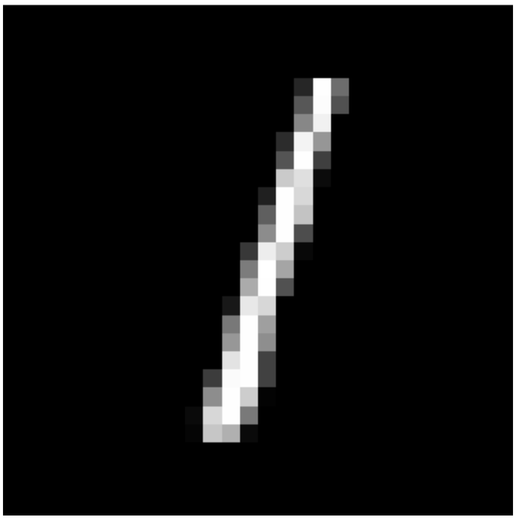
Predicted: 7, Actual: 7



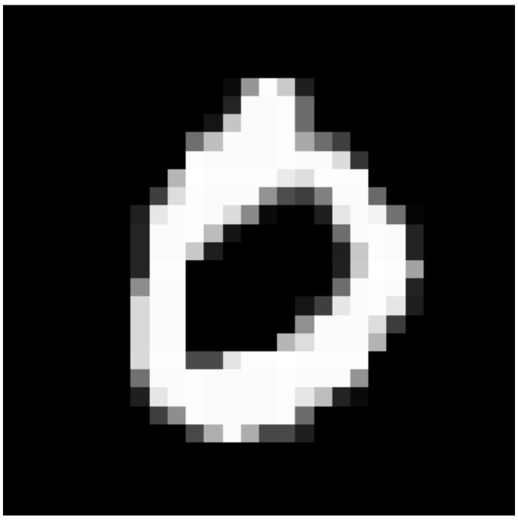
Predicted: 2, Actual: 2



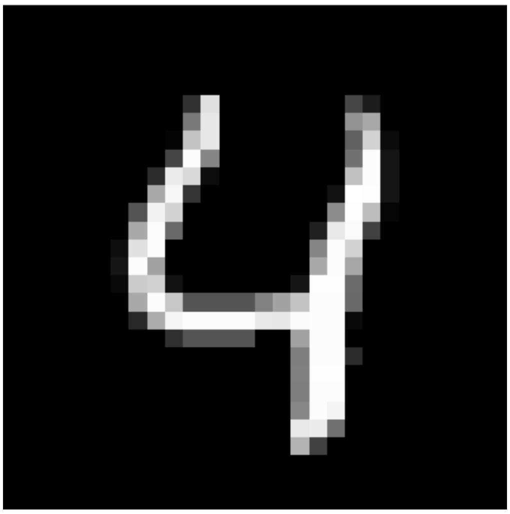
Predicted: 1, Actual: 1



Predicted: 0, Actual: 0



Predicted: 4, Actual: 4



Conclusion:

1) Deep learning in image classification was implemented successfully
2) Handwritten Digit Recognition System using MNIST Dataset was created successfully

3) Deep learning was implemented in Traffic Signs Recognition System

4) Image Caption Generator created using python libraries like PIL , urllib

References:

[Image recognition with Machine Learning on Python, Convolutional Neural Network | by Jonathan Leban | Towards Data Science]

[Image Recognition and Classification in Python with TensorFlow and Keras (stackabuse.com)]

[Traffic Signs Recognition using CNN and Keras in Python (analyticsvidhya.com)]

[Image Captioning using Python - GeeksforGeeks]

https://gist.github.com/PurpleVen/b1c42a86687dc3fcc7cca2170ee2f77f

https://gist.github.com/PurpleVen/c6accf26055596c9fa4d34b64f1a087d

https://gist.github.com/PurpleVen/a4649885c069e1ba14c42d251f55aee1

Experiment 05: Advanced ML Classification Techniques

Aim: Implementation of supervised learning algorithms like
a. Ada-Boosting
b. Random forests

Evaluation of Classification Algorithms.

Theory:
Supervised learning is a fundamental concept in machine learning where algorithms are trained on labeled data to make predictions or classifications on unseen data. In this theoretical framework, we will explore the implementation of two popular supervised learning algorithms, AdaBoost and Random Forests, and discuss the evaluation of classification algorithms, which is crucial for assessing their performance.

1. AdaBoost (Adaptive Boosting): AdaBoost is an ensemble learning algorithm that combines the predictions of weak classifiers to create a strong classifier. It focuses on the instances that were previously misclassified by adjusting their weights in subsequent iterations. The algorithm can be described as follows:

a. Initialization: Assign equal weights to all training instances.
b. Iteration: Repeatedly train weak classifiers on the data while giving more weight to misclassified instances.
c. Combine: Combine the weak classifiers' predictions with weighted voting to create a strong classifier.
Advantages of AdaBoost:

- It is effective in handling both binary and multiclass classification problems.
- It adapts well to complex datasets.
- It is less prone to overfitting compared to some other algorithms.

2. Random Forests: Random Forests is another ensemble learning technique, but it constructs multiple decision trees and aggregates their predictions. It works as follows:

a. Bootstrap Sampling: Randomly select subsets of the training data with replacement.
b. Feature Selection: For each tree, choose a random subset of features to split on.
c. Decision Trees: Grow decision trees using the selected data and features.
d. Voting: Combine the predictions of individual trees through majority voting (classification) or averaging (regression).
Advantages of Random Forests:

- It is robust against overfitting.
- It provides feature importance scores, aiding in feature selection.
- It is effective on both classification and regression tasks.

3. Evaluation of Classification Algorithms: Evaluating classification algorithms is crucial to understand their performance and make informed decisions about their usage. Common evaluation metrics include:

a. Accuracy: The ratio of correctly classified instances to the total number of instances.
b. Precision: The ratio of true positives to the sum of true positives and false positives.
c. Recall (Sensitivity): The ratio of true positives to the sum of true positives and false negatives. d. F1 Score: The harmonic mean of precision and recall, balancing precision and recall.
e. Confusion Matrix: A table that visualizes the performance of a classification algorithm.

To evaluate these algorithms, it is essential to perform cross-validation, holdout testing, or other techniques to assess their generalization performance and mitigate issues like overfitting.

Code & Output:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the Iris dataset
data = load_iris()
X = data.data  # Features
y = data.target  # Labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

a. Ada-Boosting

```
[ ] # Initialize AdaBoost Classifier
    adaboost_model = AdaBoostClassifier(n_estimators=100, random_state=42)

    # Train the model
    adaboost_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred_adaboost = adaboost_model.predict(X_test)

    # Evaluate the model
    print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred_adaboost))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The
  warnings.warn(
AdaBoost Accuracy: 1.0
```

b. Random forests

```
# Initialize Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 1.0
```

```
# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_model, X, y, cv=5)
print("Random Forest Cross-Validation Scores:", rf_cv_scores)
print("Average CV Score (Random Forest):", np.mean(rf_cv_scores))

# Cross-validation for AdaBoost
adaboost_cv_scores = cross_val_score(adaboost_model, X, y, cv=5)
print("AdaBoost Cross-Validation Scores:", adaboost_cv_scores)
print("Average CV Score (AdaBoost):", np.mean(adaboost_cv_scores))
```

```
Random Forest Cross-Validation Scores: [0.96666667 0.96666667 0.93333333 0.966666
Average CV Score (Random Forest): 0.9666666666666668
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
AdaBoost Cross-Validation Scores: [0.96666667 0.93333333 0.93333333 0.9        1.
Average CV Score (AdaBoost): 0.9466666666666665
```

Conclusion :

In conclusion, the implementation and evaluation of supervised learning algorithms like AdaBoost and Random Forests are critical steps in machine learning that enable the effective solving of classification problems with enhanced predictive performance.

References:

https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/
https://www.geeksforgeeks.org/random-forest-regression-in-python/
https://www.geeksforgeeks.org/implementing-the-adaboost-algorithm-from-scratch/
https://www.javatpoint.com/machine-learning-random-forest-algorithm
https://github.com/Rieesteves/EXP-5---DS/blob/main/EXP_5_ada.ipynb
https://github.com/Rieesteves/EXP-5---DS/blob/main/EXP_5_Random_Forest.ipynb

Mini Project

Title – Sentiment Analysis and Prediction

❖ **Abstract:**

Sentiment analysis in the context of movie reviews involves extracting and interpreting the emotions and opinions expressed in textual data, providing valuable insights into audience perceptions and feedback. This study focuses on utilizing natural language processing (NLP) techniques to analyze movie reviews and classify them based on sentiment polarity, specifically positive and negative sentiments. The VADER (Valence Aware Dictionary and sEntiment Reasoner) tool is employed to compute sentiment scores for each review, leveraging its lexicon-based approach to effectively capture nuances in emotions and language intensity. By analyzing these scores, the system identifies patterns, aggregates insights, and delivers a comprehensive understanding of audience sentiment, enabling filmmakers and marketers to enhance their offerings.

Furthermore, the sentiment analysis process includes an exploratory analysis of sentiment distribution, allowing for the identification of trends and correlations within the dataset. This study operates without machine learning models, relying solely on a rule-based approach to assign sentiment labels based on VADER polarity scores. This methodology ensures efficient analysis and facilitates real-time sentiment tracking and prediction. By integrating these insights into decision-making processes, stakeholders can proactively address audience needs, optimize marketing strategies, and improve overall film quality, resulting in higher viewer satisfaction and engagement.

❖ **Methodology: Sentiment Analysis and Prediction**

1. **Data Collection**:
   - The dataset consists of 50,000 movie reviews, structured with two key columns: 'review' (the text of the review) and 'sentiment' (the associated sentiment label, either positive or negative).
2. **Data Preprocessing**:
   - **Text Cleaning**: Reviews are cleaned to remove punctuation, special characters, numbers, and stopwords.
   - **Tokenization**: The cleaned text is split into tokens (individual words).
   - **Lowercasing**: Text is converted to lowercase for uniformity.

- **Lemmatization/Stemming**: Words are reduced to their root form for consistency in analysis.

3. **Polarity Scoring**:
   - **Sentiment Scoring**: The VADER sentiment analysis tool is applied to each review, calculating a polarity score ranging from -1 (negative) to 1 (positive). The score indicates the overall sentiment expressed in the review.
   - **Categorization**: Based on the polarity score, reviews are categorized as positive (polarity > 0), negative (polarity < 0), or neutral (polarity = 0).
4. **Exploratory Data Analysis (EDA)**:
   - EDA is performed to examine the distribution of sentiment scores across various segments of the dataset.
5. **Analysis**:
   - The sentiment scores are aggregated to assess the overall sentiment distribution across the dataset, helping to identify the proportion of positive and negative sentiments in the reviews.
   - Correlation analysis is conducted to explore any relationships between sentiment scores and other potential features within the dataset, though in this case, the main focus is on the review text and sentiment label.
6. **Visualization**:
   - Visualization techniques, including confusion matrices and word clouds, are employed to present the distribution of sentiments, highlight common words or phrases in positive and negative reviews, and provide insights into customer feedback trends.

This methodology leverages VADER sentiment scoring on movie reviews to categorize sentiments accurately, using both traditional machine learning models and visualization techniques. The findings aim to offer insights into audience perceptions of films, helping filmmakers and marketers make informed decisions based on the analyzed sentiment data.

❖ **Dataset Table:**

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

- **Implementation:**

```
[112] import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      sns.set_style('darkgrid')
```

```
[113] from google.colab import drive
      drive.mount('/content/drive')
```

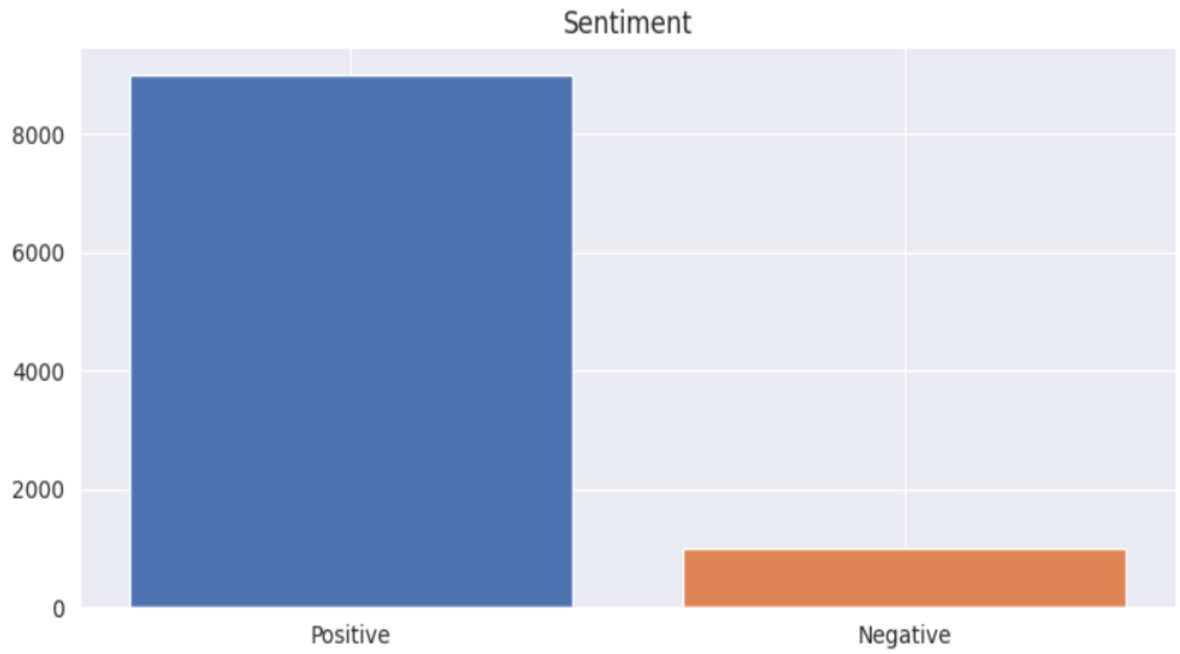
```
Mounted at /content/drive
```

```
[114] df_review = pd.read_csv("/content/sample_data/IMDB Dataset.csv")
      df_review.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
[115] df_positive = df_review[df_review['sentiment']=='positive'][:9000]
      df_negative = df_review[df_review['sentiment']=='negative'][:1000]

      df_review_imb = pd.concat([df_positive,df_negative ])
```

```
[116] colors = sns.color_palette('deep')

      plt.figure(figsize=(8,4), tight_layout=True)
      plt.bar(x=['Positive', 'Negative'],
              height=df_review_imb.value_counts(['sentiment']),
              color=colors[:2])
      plt.title('Sentiment')
      plt.savefig('sentiment.png')
      plt.show()
```




```
[117] from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state= 0)
df_review_bal,df_review_bal['sentiment']=rus.fit_resample(df_review_imb[['review']],df_review_imb['sentiment'])


df_review_bal
```

	review	sentiment
3	Basically there's a family where a little boy ...	negative
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
10	Phil the Alien is one of those quirky films wh...	negative
11	I saw this movie when I was about 12 when it c...	negative
...	...	...
10833	Knute Rockne led an extraordinary life and his...	positive
1039	At the height of the 'Celebrity Big Brother' r...	positive
5255	This is another of Robert Altman's underrated ...	positive
12237	This movie won a special award at Cannes for i...	positive
15014	You'd be forgiven to think a Finnish director ...	positive

2000 rows × 2 columns

```
[118] print(df_review_imb.value_counts('sentiment'))
      print(df_review_bal.value_counts('sentiment'))
```

```
sentiment
positive    9000
negative    1000
Name: count, dtype: int64
sentiment
negative    1000
positive    1000
Name: count, dtype: int64
```

```
[119] from sklearn.model_selection import train_test_split

      train,test = train_test_split(df_review_bal,test_size =0.33,random_state=42)
```

```
[120] train_x, train_y = train['review'], train['sentiment']
      test_x, test_y = test['review'], test['sentiment']
```

```
[121] train_y.value_counts()
```

	count
sentiment	
negative	675
positive	665

dtype: int64

```
[122] from sklearn.feature_extraction.text import TfidfVectorizer

      tfidf = TfidfVectorizer(stop_words='english')
      train_x_vector = tfidf.fit_transform(train_x)
      # also fit the test_x_vector
      test_x_vector = tfidf.transform(test_x)
```

```
[123] from sklearn.svm import SVC
      svc = SVC(kernel='linear')
      svc.fit(train_x_vector, train_y)
```

```
SVC
SVC(kernel='linear')
```

```
[124] from sklearn.tree import DecisionTreeClassifier

      dec_tree = DecisionTreeClassifier()
      dec_tree.fit(train_x_vector, train_y)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
[125] from sklearn.ensemble import RandomForestClassifier

      # Initialize the Random Forest Classifier
      rf_model = RandomForestClassifier(random_state=42)

      # Train the model
      rf_model.fit(train_x_vector, train_y)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
[126] print(svc.score(test_x_vector, test_y))
      print(dec_tree.score(test_x_vector, test_y))
      print(rf_model.score(test_x_vector, test_y))
```

```
0.8409090909090909
0.6636363636363637
0.8015151515151515
```

```
[127] from sklearn.metrics import f1_score

      f1_score(test_y,svc.predict(test_x_vector),
               labels = ['positive','negative'],average=None)
```

```
array([0.84671533, 0.83464567])
```

```
[128] from sklearn.metrics import classification_report

      print(classification_report(test_y,
                                  svc.predict(test_x_vector),
                                  labels = ['positive','negative']))
```

```
              precision    recall  f1-score   support

    positive       0.83      0.87      0.85       335
    negative       0.85      0.82      0.83       325

    accuracy                           0.84       660
   macro avg       0.84      0.84      0.84       660
weighted avg       0.84      0.84      0.84       660
```

```
[126] print(svc.score(test_x_vector, test_y))
      print(dec_tree.score(test_x_vector, test_y))
      print(rf_model.score(test_x_vector, test_y))
```

```
0.8409090909090909
0.6636363636363637
0.8015151515151515
```

```
[127] from sklearn.metrics import f1_score

      f1_score(test_y,svc.predict(test_x_vector),
                labels = ['positive','negative'],average=None)
```

```
array([0.84671533, 0.83464567])
```

```
[128] from sklearn.metrics import classification_report

      print(classification_report(test_y,
                                  svc.predict(test_x_vector),
                                  labels = ['positive','negative']))
```

```
              precision    recall  f1-score   support

    positive       0.83      0.87      0.85       335
    negative       0.85      0.82      0.83       325

    accuracy                           0.84       660
   macro avg       0.84      0.84      0.84       660
weighted avg       0.84      0.84      0.84       660
```

```
[130] from wordcloud import WordCloud
      import matplotlib.pyplot as plt

      # Separate positive reviews
      positive_reviews = train_x[train_y == 'positive']

      # Combine all positive reviews into a single string
      positive_text = " ".join(positive_reviews)

      # Generate the word cloud
      plt.figure(figsize=(10, 10))
      WC = WordCloud(width=1000, height=500, max_words=500, min_font_size=5, background_color='white')
      positive_words = WC.generate(positive_text)

      # Display the word cloud
      plt.imshow(positive_words, interpolation='bilinear')
      plt.axis('off')  # Remove axis for better visualization
      plt.show()
```

[131]

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Separate negative reviews
negative_reviews = train_x[train_y == 'negative']

# Combine all negative reviews into a single string
negative_text = " ".join(negative_reviews)

# Generate the word cloud
plt.figure(figsize=(10, 10))
WC = WordCloud(width=1000, height=500, max_words=500, min_font_size=5, background_color='white')
negative_words = WC.generate(negative_text)

# Display the word cloud
plt.imshow(negative_words, interpolation='bilinear')
plt.axis('off')  # Remove axis for better visualization
plt.show()
```


```
[132] print(svc.predict(tfidf.transform(['A good movie with a fantastic storyline!'])))
      print(svc.predict(tfidf.transform(['An excellent movie that kept me on the edge of my seat!'])))
      print(svc.predict(tfidf.transform(['I did not like this movie at all.'])))
      print(svc.predict(tfidf.transform(['I absolutely loved every minute of it!'])))
      print(svc.predict(tfidf.transform(['The film was boring and too long.'])))
      print(svc.predict(tfidf.transform(['This is one of the best movies I have ever seen!'])))
      print(svc.predict(tfidf.transform(['Very disappointing, I won't watch it again.'])))
      print(svc.predict(tfidf.transform(['Highly recommend it to everyone!'])))
      print(svc.predict(tfidf.transform(['This movie exceeded my expectations!'])))
      print(svc.predict(tfidf.transform(['A complete disappointment from start to finish.'])))
```

```
['positive']
['positive']
['negative']
['positive']
['negative']
['positive']
['negative']
['positive']
['positive']
['negative']
```

Conclusion:

The sentiment analysis of movie reviews demonstrates the effectiveness of using natural language processing techniques to gauge audience perceptions. By employing a combination of text preprocessing, sentiment scoring with VADER, and machine learning models such as Decision Trees and Random Forests, the project successfully categorized reviews into positive and negative sentiments. The results not only highlighted prevailing trends in audience feedback but also provided valuable insights for filmmakers and marketers in understanding audience preferences. Overall, this methodology lays the groundwork for further exploration of sentiment analysis in various domains, enhancing decision-making based on feedback.