SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 3rd semester

Cognitive Analytics Tools and Techniques (PGI20G05J)- Lab Manual

Lab 1: Customer Segmentation using Machine Learning in Python

Title

Customer Segmentation using Machine Learning in Python

Aim

To implement customer segmentation using various machine learning algorithms in Python to identify distinct customer groups based on their characteristics and behavior.

Procedure

[Detailed steps for performing the experiment, including data loading, preprocessing, model selection, training, evaluation, and visualization. This section should be filled in by the user based on the specific implementation details.]

Source Code

```
# [Python code for customer segmentation]
# Example:
# import pandas as pd
# from sklearn.cluster import KMeans
# import matplotlib.pyplot as plt
# import seaborn as sns

# # Load data
# # df = pd.read_csv('customer_data.csv')

# # Preprocessing (e.g., scaling, handling missing values)
# # ...

# # Apply KMeans
# # kmeans = KMeans(n_clusters=3, random_state=42)
# # df['cluster'] = kmeans.fit_predict(df[['feature1', 'feature2']])

# # Visualization
# # sns.scatterplot(x='feature1', y='feature2', hue='cluster', data=df)
# # plt.title('Customer Segments')
# # plt.show()
```

Input

[Description of the input data required for the program, e.g., "A CSV file named `customer_data.csv` containing customer features such as age, income, spending score, etc."]

Expected Output

[Description of the expected results, e.g., "Visualizations (scatter plots, bar charts) showing distinct customer clusters, along with metrics like silhouette score or inertia to evaluate the clustering performance. A segmented dataset with a new 'cluster' column indicating the assigned segment for each customer."]

Lab 2: Implementation of Simple Machine AI real time problem

Title

Implementation of Simple Machine AI for a Real-time Problem

Aim

To implement a simple machine learning model to solve a real-time problem, demonstrating the basic principles of AI application in a practical scenario.

Procedure

[Detailed steps for defining the real-time problem, collecting/simulating data, choosing a suitable simple ML algorithm (e.g., linear regression, logistic regression), training the model, and deploying it for real-time predictions.]

Source Code

```
# [Python code for simple machine AI real-time problem]
# Example:
# import numpy as np
# from sklearn.linear_model import LinearRegression

# # Simulate real-time data
# # X_train = np.array([[1], [2], [3], [4], [5]])
# # y_train = np.array([2, 4, 5, 4, 5])

# # Train a simple model
# # model = LinearRegression()
# # model.fit(X_train, y_train)

# # Real-time prediction (example)
# # new_data = np.array([[6]])
# # prediction = model.predict(new_data)
# # print(f"Prediction for new data: {prediction[0]}")
```

Input

[Description of the input data for the real-time problem, e.g., "Streaming sensor data, user input, or simulated real-time events."]

Expected Output

[Description of the expected results, e.g., "Real-time predictions or classifications based on the incoming data, displayed on a console or a simple UI."]

Lab 3: Build a model for Information Processing using Cognitive science

Title

Building a Model for Information Processing using Cognitive Science Principles

Aim

To construct a computational model that simulates aspects of human information processing, drawing insights from cognitive science theories (e.g., memory models, attention mechanisms).

Procedure

[Detailed steps for selecting a cognitive theory, designing the model architecture, implementing the processing steps (e.g., encoding, storage, retrieval), and evaluating its behavior against theoretical predictions or human data.]

Source Code

```
# [Python code for information processing model]
# Example:
# class MemoryModel:
#     def __init__(self):
#         self.short_term_memory = []
#         self.long_term_memory = {}

#     def encode(self, info):
#         # Simulate encoding process
#         self.short_term_memory.append(info)
#         print(f"Encoded: {info}")

#     def store(self, info, category):
#         # Simulate storage to long-term memory
#         if category not in self.long_term_memory:
#             self.long_term_memory[category] = []
#         self.long_term_memory[category].append(info)
#         print(f"Stored '{info}' in '{category}'")

#     def retrieve(self, query):
#         # Simulate retrieval process
#         for category, items in self.long_term_memory.items():
#             if query in items:
#                 return f"Found '{query}' in '{category}'"
#         return f"'{query}' not found."

# # model = MemoryModel()
# # model.encode("apple")
# # model.store("apple", "fruit")
# # print(model.retrieve("apple"))
```

Input

[Description of the input for the cognitive model, e.g., "A sequence of stimuli or information items to be processed, or queries for memory retrieval."]

Expected Output

[Description of the expected results, e.g., "Simulation outputs demonstrating the model's processing of information, such as successful encoding, storage, and retrieval, or patterns of errors consistent with human cognitive biases."]

Lab 4: Implementation of Decision tree and K-Mean algorithm-A Low Level cognitive approach

Title

Implementation of Decision Tree and K-Means Algorithm: A Low-Level Cognitive Approach

Aim

To implement and compare the Decision Tree and K-Means algorithms, exploring their underlying mechanisms from a low-level cognitive perspective, focusing on how they mimic basic categorization and decision-making processes.

Procedure

[Detailed steps for preparing a dataset, implementing both Decision Tree and K-Means from scratch or using libraries, applying them to the dataset, and analyzing their performance and how their logic relates to cognitive processes.]

Source Code

```
# [Python code for Decision Tree and K-Means]
# Example:
# from sklearn.tree import DecisionTreeClassifier
# from sklearn.cluster import KMeans
# from sklearn.datasets import load_iris
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import accuracy_score

# # Load dataset
# # iris = load_iris()
# # X, y = iris.data, iris.target

# # Decision Tree
# # X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X, y,
test_size=0.3, random_state=42)
# # dt_model = DecisionTreeClassifier()
# # dt_model.fit(X_train_dt, y_train_dt)
# # dt_predictions = dt_model.predict(X_test_dt)
# # print(f"Decision Tree Accuracy: {accuracy_score(y_test_dt,
dt_predictions)}")

# # K-Means
# # kmeans_model = KMeans(n_clusters=3, random_state=42)
# # kmeans_labels = kmeans_model.fit_predict(X)
# # print(f"K-Means cluster labels: {kmeans_labels}")
```

Input

[Description of the input dataset, e.g., "A dataset suitable for classification (for Decision Tree) and clustering (for K-Means), such as the Iris dataset or a custom dataset."]

Expected Output

[Description of the expected results, e.g., "Trained Decision Tree model with its accuracy, and K-Means cluster assignments for the data points. Analysis of how the algorithms partition data and make decisions, relating it to cognitive categorization."]

Lab 5: Build a Bayesian Model for Anomaly Detection

Title

Building a Bayesian Model for Anomaly Detection

Aim

To develop a Bayesian model capable of identifying anomalies or outliers within a dataset, leveraging probabilistic reasoning to assess the likelihood of data points.

Procedure

[Detailed steps for understanding Bayesian principles, selecting appropriate probabilistic distributions, constructing the Bayesian network or model, training it on normal data, and using it to detect deviations indicative of anomalies.]

Source Code

```
# [Python code for Bayesian Anomaly Detection]
# Example:
# import numpy as np
# from scipy.stats import norm

# # Simulate normal data
# # data = np.random.normal(loc=0, scale=1, size=100)

# # Simulate an anomaly
# # anomaly = 5.0
# # data = np.append(data, anomaly)

# # Fit a Gaussian distribution to the normal data
# # mu, std = norm.fit(data[:-1]) # Fit on normal data

# # Calculate probability density for each point
# # p_data = norm.pdf(data, loc=mu, scale=std)

# # Identify anomalies (e.g., points with very low probability)
# # threshold = 0.01 # Example threshold
# # anomalies_detected = data[p_data < threshold]
# # print(f"Detected anomalies: {anomalies_detected}")
```

Input

[Description of the input dataset, e.g., "A dataset containing numerical observations, where some observations might be anomalous."]

Expected Output

[Description of the expected results, e.g., "Identification of anomalous data points with their corresponding low probabilities, demonstrating the model's ability to distinguish between normal and unusual observations."]

Lab 6: Implement Knowledge representation using predicate logics

Title

Implementing Knowledge Representation using Predicate Logics

Aim

To implement a system that represents knowledge using predicate logic, allowing for logical inference and querying of facts and relationships.

Procedure

[Detailed steps for defining predicates, constants, variables, and logical connectives. Implementing a simple inference engine (e.g., forward or backward chaining) to answer queries based on the represented knowledge base.]

Source Code

```
# [Python code for Knowledge Representation using Predicate Logics]
# Example (using a simplified approach for demonstration):
# class KnowledgeBase:
#     def __init__(self):
#         self.facts = set() # Stores facts like "is_animal(dog)"

#     def add_fact(self, fact):
#         self.facts.add(fact)
#         print(f"Added fact: {fact}")

#     def query(self, query_fact):
#         return query_fact in self.facts

# # kb = KnowledgeBase()
# # kb.add_fact("is_animal(dog)")
# # kb.add_fact("has_legs(dog, 4)")
# # print(f"Is dog an animal? {kb.query('is_animal(dog)')}")
# # print(f"Does cat have 4 legs? {kb.query('has_legs(cat, 4)')}")
```

Input

[Description of the input, e.g., "A set of facts and rules expressed in predicate logic, and queries to be evaluated against the knowledge base."]

Expected Output

[Description of the expected results, e.g., "Truth values (True/False) for queries, demonstrating the system's ability to infer new knowledge or confirm existing facts based on the predicate logic representation."]

Lab 7: Implement model using speech analytics techniques

Title

Implementing a Model using Speech Analytics Techniques

Aim

To develop a model that processes and analyzes speech data to extract meaningful insights, such as sentiment, speaker identification, or topic detection.

Procedure

[Detailed steps for acquiring speech data, performing speech-to-text conversion, applying natural language processing (NLP) techniques to the transcribed text, and building a model for the desired speech analytics task.]

Source Code

```
# [Python code for Speech Analytics]
# Example (conceptual, requires external libraries like SpeechRecognition,
NLTK):
# import speech_recognition as sr
# from nltk.sentiment import SentimentIntensityAnalyzer

# # Initialize recognizer
# # r = sr.Recognizer()

# # Load audio file (replace with actual audio file)
# # with sr.AudioFile("audio.wav") as source:
# #     audio_data = r.record(source)
# #     try:
# #         text = r.recognize_google(audio_data)
# #         print(f"Transcribed Text: {text}")

# #         # Perform sentiment analysis
# #         sid = SentimentIntensityAnalyzer()
# #         sentiment_scores = sid.polarity_scores(text)
# #         print(f"Sentiment Scores: {sentiment_scores}")

# #     except sr.UnknownValueError:
# #         print("Google Speech Recognition could not understand audio")
# #     except sr.RequestError as e:
# #         print(f"Could not request results from Google Speech Recognition
service; {e}")
```

Input

[Description of the input, e.g., "Audio files (e.g., WAV, MP3) containing speech, or live audio input from a microphone."]

Expected Output

[Description of the expected results, e.g., "Transcribed text from speech, sentiment scores (positive, negative, neutral), identified topics, or speaker characteristics, depending on the specific analytics task."]

Lab 8: Implement Data Visualization using your own dataset

Title

Implementing Data Visualization using a Custom Dataset

Aim

To create insightful and effective data visualizations using a self-selected dataset, demonstrating the ability to communicate patterns, trends, and relationships graphically.

Procedure

[Detailed steps for choosing a dataset, exploring its characteristics, selecting appropriate visualization types (e.g., bar charts, scatter plots, line graphs, heatmaps), and implementing the visualizations using Python libraries like Matplotlib or Seaborn.]

Source Code

```
# [Python code for Data Visualization]
# Example:
# import pandas as pd
# import matplotlib.pyplot as plt
# import seaborn as sns

# # Load your dataset (replace with your actual dataset)
# # df = pd.read_csv('my_dataset.csv')

# # Example visualizations:
# # plt.figure(figsize=(10, 6))
# # sns.histplot(df['numerical_column'], kde=True)
# # plt.title('Distribution of Numerical Column')
# # plt.xlabel('Value')
# # plt.ylabel('Frequency')
# # plt.show()

# # plt.figure(figsize=(10, 6))
# # sns.scatterplot(x='feature_X', y='feature_Y', hue='category_column',
data=df)
# # plt.title('Scatter Plot of X vs Y by Category')
# # plt.xlabel('Feature X')
# # plt.ylabel('Feature Y')
# # plt.show()
```

Input

[Description of the input, e.g., "A CSV file or other structured data file containing your chosen dataset."]

Expected Output

[Description of the expected results, e.g., "A series of well-designed and labeled plots and charts that effectively convey insights from the dataset, such as distributions, correlations, and comparisons."]

Lab 9: Explore the roles that metadata play in decision making, memory retrievals, and learning

Title

Exploring the Roles of Metadata in Decision Making, Memory Retrievals, and Learning

Aim

To investigate and demonstrate, through conceptual modeling or simulation, how metadata influences cognitive processes such as decision-making, memory organization and retrieval, and various forms of learning.

Procedure

[Detailed steps for defining what constitutes "metadata" in a cognitive context, designing scenarios or simplified models that show how metadata can guide attention, facilitate memory search, or contribute to the formation of new knowledge structures. This lab might involve more theoretical discussion and conceptual modeling than direct coding.]

Source Code

```
# [Python code for conceptual modeling of metadata's role (if applicable)]
# This lab might be more conceptual and involve less direct coding,
# focusing on theoretical explanations and examples.
# If coding, it would likely involve simulations of information retrieval
# or decision processes guided by associated metadata.

# Example (conceptual):
# class MemorySystem:
#     def __init__(self):
#         self.memories = {} # {item_id: {'content': '...', 'metadata':
{'tag': '...', 'context': '...'}}}

#     def store_memory(self, item_id, content, metadata):
#         self.memories[item_id] = {'content': content, 'metadata': metadata}
#         print(f"Stored memory '{item_id}' with metadata: {metadata}")

#     def retrieve_by_metadata(self, query_metadata):
#         found_memories = []
#         for item_id, mem_data in self.memories.items():
#             if all(mem_data['metadata'].get(key) == value for key, value in
query_metadata.items()):
#                 found_memories.append(mem_data['content'])
#         return found_memories

# # ms = MemorySystem()
# # ms.store_memory("event1", "Had coffee", {"location": "cafe", "mood":
"happy"})
# # ms.store_memory("event2", "Read book", {"location": "home", "mood":
"relaxed"})
# # print(f"Memories from cafe: {ms.retrieve_by_metadata({'location':
'cafe'})}")
```

Input

[Description of the input, e.g., "Conceptual scenarios, simulated memory items with associated metadata, or examples of decision tasks where metadata plays a role."]

Expected Output

[Description of the expected results, e.g., "Demonstrations or explanations of how metadata can speed up memory retrieval, bias decision-making towards certain options, or facilitate the generalization of learned concepts."]

Lab 10: Text Detection and Extraction using OpenCV and OCR

Title

Text Detection and Extraction using OpenCV and OCR

Aim

To implement a system that can detect text regions within an image and then extract the text content using Optical Character Recognition (OCR) techniques, primarily leveraging OpenCV.

Procedure

[Detailed steps for loading an image, applying image processing techniques (e.g., grayscale conversion, thresholding, contour detection) to identify text areas, and then using an OCR engine (like Tesseract) to extract the text from the detected regions.]

Source Code

```
# [Python code for Text Detection and Extraction]
# Example (requires OpenCV and Tesseract OCR installed):
# import cv2
# import pytesseract # pip install pytesseract

# # Path to your Tesseract executable (change if necessary)
# # pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'

# # Load the image
# # image = cv2.imread('image_with_text.png')
# # if image is None:
# #     print("Error: Could not load image.")
# # else:
# #     # Convert to grayscale
# #     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# #     # Apply OCR
# #     text = pytesseract.image_to_string(gray)
# #     print(f"Extracted Text:\n{text}")

# #     # (Optional) Visualize text detection (more complex with OpenCV)
# #     # boxes = pytesseract.image_to_boxes(gray)
# #     # h, w, _ = image.shape
# #     # for b in boxes.splitlines():
# #     #     b = b.split(' ')
# #     #     img = cv2.rectangle(image, (int(b[1]), h - int(b[2])),
(int(b[3]), h - int(b[4])), (0, 255, 0), 2)
# #     # cv2.imshow('Detected Text', img)
# #     # cv2.waitKey(0)
# #     # cv2.destroyAllWindows()
```

Input

[Description of the input, e.g., "An image file (e.g., JPG, PNG) containing text."]

Expected Output

[Description of the expected results, e.g., "The extracted text content from the input image printed to the console, and optionally, the image displayed with detected text regions highlighted."]

Lab 11: Age predictor and Gender classifier project using OpenCV.

Title

Age Predictor and Gender Classifier Project using OpenCV

Aim

To develop a system that can predict the age and classify the gender of individuals from facial images using computer vision techniques with OpenCV.

Procedure

[Detailed steps for acquiring a dataset of faces with age and gender labels, pre-processing images (e.g., face detection, alignment), training a machine learning model (e.g., deep learning model) for age and gender prediction, and integrating it with OpenCV for real-time or image-based inference.]

Source Code

```
# [Python code for Age Predictor and Gender Classifier]
# Example (conceptual, requires pre-trained models or training data):
# import cv2
# import numpy as np

# # Load pre-trained face detector (e.g., Haarcascades or DNN model)
# # face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# # Load pre-trained age and gender models (e.g., from Caffe or TensorFlow)
# # age_net = cv2.dnn.readNetFromCaffe('deploy_age.prototxt',
'age_net.caffemodel')
# # gender_net = cv2.dnn.readNetFromCaffe('deploy_gender.prototxt',
'gender_net.caffemodel')

# # AGE_LIST = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)',
'(48-53)', '(60-100)']
# # GENDER_LIST = ['Male', 'Female']

# # image = cv2.imread('person.jpg')
# # if image is None:
# #     print("Error: Could not load image.")
# # else:
# #     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# #     faces = face_cascade.detectMultiScale(gray, 1.3, 5)

# #     for (x, y, w, h) in faces:
# #         face_img = image[y:y+h, x:x+w].copy()
# #         blob = cv2.dnn.blobFromImage(face_img, 1.0, (227, 227),
(78.4263377603, 87.7689143744, 114.895847746), swapRB=False)

# #         # Predict gender
# #         # gender_net.setInput(blob)
# #         # gender_preds = gender_net.forward()
# #         # gender = GENDER_LIST[gender_preds[0].argmax()]

# #         # Predict age
# #         # age_net.setInput(blob)
# #         # age_preds = age_net.forward()
# #         # age = AGE_LIST[age_preds[0].argmax()]
```

```
# #         # cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
# #         # cv2.putText(image, f"{gender}, {age}", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 255), 2, cv2.LINE_AA)

# #     # cv2.imshow('Age and Gender Prediction', image)
# #     # cv2.waitKey(0)
# #     # cv2.destroyAllWindows()
```

Input

[Description of the input, e.g., "Image files (e.g., JPG, PNG) containing faces, or live video stream from a webcam."]

Expected Output

[Description of the expected results, e.g., "The predicted age range and classified gender (Male/Female) displayed on the image or console for each detected face."]

Lab 12: Case Study on Ethical, Fairness and Privacy considerations in Cognitive Science

Title

Case Study on Ethical, Fairness, and Privacy Considerations in Cognitive Science

Aim

To analyze and discuss a specific case study related to cognitive science applications, focusing on the ethical implications, fairness biases, and privacy concerns that may arise.

Procedure

[Detailed steps for selecting a relevant case study (e.g., AI in hiring, personalized education, neuro-marketing), researching its technical aspects and societal impact, identifying potential ethical dilemmas, fairness issues (e.g., algorithmic bias), and privacy risks, and proposing mitigation strategies.]

Source Code

```
# [This lab is primarily a theoretical case study and does not involve
coding.]
# This section will contain a detailed written analysis of the chosen case
study.
# It might involve:
# - Description of the cognitive science application.
# - Identification of ethical principles violated or challenged.
# - Analysis of potential biases and their impact on fairness.
# - Discussion of data privacy issues and potential breaches.
# - Recommendations for ethical design and deployment.
```

Input

[Description of the input, e.g., "A specific real-world or hypothetical scenario involving cognitive science technology or research that presents ethical, fairness, or privacy challenges."]

Expected Output

[Description of the expected results, e.g., "A comprehensive written report or presentation outlining the chosen case study, a critical analysis of its ethical, fairness, and privacy dimensions, and well-reasoned recommendations for addressing these concerns."]

Lab 13: Build a cognitive assistant for Visually Impaired

Title

Building a Cognitive Assistant for the Visually Impaired

Aim

To develop a cognitive assistant that leverages AI and cognitive computing techniques to provide support and enhance the independence of visually impaired individuals.

Procedure

[Detailed steps for identifying specific needs of visually impaired users, selecting relevant AI technologies (e.g., object recognition, text-to-speech, navigation assistance), designing the assistant's functionalities, implementing the core components, and testing its usability and effectiveness.]

Source Code

```
# [Python code for Cognitive Assistant for Visually Impaired]
# Example (conceptual, integrating multiple AI components):
# import cv2
# import pyttsx3 # Text-to-speech
# from ultralytics import YOLO # Object detection (requires model)
# from PIL import Image

# # Initialize text-to-speech engine
# # engine = pyttsx3.init()
# # engine.setProperty('rate', 150) # Speed of speech

# # Load object detection model (e.g., YOLOv8)
# # model = YOLO('yolov8n.pt') # Load a pre-trained YOLOv8n model

# # Function to describe objects in an image
# # def describe_image(image_path):
# #     img = Image.open(image_path)
# #     results = model(img)
# #     detections = []
# #     for r in results:
# #         for box in r.boxes:
# #             class_id = int(box.cls[0])
# #             label = model.names[class_id]
# #             detections.append(label)
# #     if detections:
# #         description = f"I see: {', '.join(set(detections))}."
# #     else:
# #         description = "I don't see any recognizable objects."
# #     engine.say(description)
# #     engine.runAndWait()

# # Example usage:
# # describe_image("room_scene.jpg")
```

Input

[Description of the input, e.g., "Images from a camera, audio commands from the user, or environmental sensor data."]

Expected Output

[Description of the expected results, e.g., "Auditory descriptions of objects in the environment, navigation instructions, reading aloud of text, or responses to user queries, providing practical assistance to the visually impaired user."]

Lab 14: Build and train a self learning Chatbot

Title

Building and Training a Self-Learning Chatbot

Aim

To develop a chatbot that can engage in conversational interactions and improve its responses over time through self-learning mechanisms, such as reinforcement learning or continuous fine-tuning.

Procedure

[Detailed steps for designing the chatbot's architecture (e.g., rule-based, retrieval-based, generative), acquiring conversational data, implementing self-learning components (e.g., feedback loops, user corrections), training the model, and evaluating its conversational fluency and learning progress.]

Source Code

```
# [Python code for Self-Learning Chatbot]
# Example (conceptual, using a simple NLTK-based chatbot with learning):
# import random
# import json

# class SimpleChatbot:
#     def __init__(self, knowledge_file="chatbot_knowledge.json"):
#         self.knowledge_file = knowledge_file
#         self.knowledge = self._load_knowledge()

#     def _load_knowledge(self):
#         try:
#             with open(self.knowledge_file, 'r') as f:
#                 return json.load(f)
#         except FileNotFoundError:
#             return {"greetings": ["Hello!", "Hi there!"], "fallback": ["I'm
not sure how to respond to that.", "Can you rephrase?"]}

#     def _save_knowledge(self):
#         with open(self.knowledge_file, 'w') as f:
#             json.dump(self.knowledge, f, indent=4)

#     def get_response(self, user_input):
#         user_input_lower = user_input.lower()
#         for intent, responses in self.knowledge.items():
#             if intent in user_input_lower: # Simple keyword matching
#                 return random.choice(responses)
#         return random.choice(self.knowledge["fallback"])

#     def learn_from_user(self, user_input, correct_response,
intent_tag="new_learning"):
#         if intent_tag not in self.knowledge:
#             self.knowledge[intent_tag] = []
#         self.knowledge[intent_tag].append(correct_response)
#         self._save_knowledge()
#         print(f"Learned: '{user_input}' -> '{correct_response}' under
'{intent_tag}'")

# # chatbot = SimpleChatbot()
# # print(chatbot.get_response("hi"))
```

```
# # print(chatbot.get_response("what is your name"))
# # chatbot.learn_from_user("what is your name", "I am a chatbot created to
assist you.", "identity")
# # print(chatbot.get_response("what is your name"))
```

Input

[Description of the input, e.g., "Text-based user queries or conversational turns."]

Expected Output

[Description of the expected results, e.g., "Contextually relevant and coherent text responses from the chatbot, demonstrating its ability to understand user input and improve its conversational capabilities over time through learning."]

Lab 15: Time Series Analysis in health care domain

Title

Time Series Analysis in the Healthcare Domain

Aim

To apply time series analysis techniques to healthcare data to identify trends, forecast future health outcomes, detect anomalies, or understand temporal patterns in patient data.

Procedure

[Detailed steps for acquiring a healthcare time series dataset (e.g., patient vital signs, disease prevalence over time), performing data cleaning and preprocessing, selecting appropriate time series models (e.g., ARIMA, Prophet, LSTM), training the model, and interpreting the results for healthcare insights.]

Source Code

```
# [Python code for Time Series Analysis in Healthcare]
# Example (conceptual, using Pandas and Statsmodels/Prophet):
# import pandas as pd
# import matplotlib.pyplot as plt
# from statsmodels.tsa.arima.model import ARIMA
# # from prophet import Prophet # if using Facebook Prophet

# # Load healthcare time series data (replace with your actual data)
# # data = pd.read_csv('healthcare_data.csv', index_col='Date',
parse_dates=True)
# # data = data['Patient_Admissions'] # Example: focus on one metric

# # Plot the time series
# # plt.figure(figsize=(12, 6))
# # plt.plot(data)
# # plt.title('Patient Admissions Over Time')
# # plt.xlabel('Date')
# # plt.ylabel('Admissions')
# # plt.show()

# # Fit an ARIMA model (example)
# # model = ARIMA(data, order=(5,1,0)) # p, d, q
# # model_fit = model.fit()
# # print(model_fit.summary())

# # Forecast
# # forecast = model_fit.predict(start=len(data), end=len(data)+10)
# # print(f"Forecasted admissions:\n{forecast}")
```

Input

[Description of the input, e.g., "A time series dataset from the healthcare domain, such as daily patient admissions, weekly disease incidence, or hourly vital signs readings."]

Expected Output

[Description of the expected results, e.g., "Visualizations of time series trends, forecasted future values, identified seasonal patterns or anomalies, and statistical summaries of the time series model's performance and insights."]