Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

☑ TEXT PREPROCESSING & TOKENIZATION USING NLTK & SPACY

📌 CHAPTER 1: INTRODUCTION TO TEXT PREPROCESSING & TOKENIZATION

1.1 What is Text Preprocessing?

Text preprocessing is the process of cleaning and transforming raw text data into a format suitable for **Natural Language Processing (NLP)** tasks. It helps remove noise and standardizes text before feeding it into machine learning models.

1.2 Why is Text Preprocessing Important?

✔ **Improves model accuracy** by removing irrelevant information.
✔ **Reduces dimensionality** for better computational efficiency.
✔ **Ensures consistency** in data representation (e.g., lowercase conversion).
✔ **Prepares text for tokenization**, vectorization, and feature extraction.

1.3 Common Text Preprocessing Steps

1. **Lowercasing** – Converts all text to lowercase.
2. **Removing Punctuation & Special Characters** – Eliminates unnecessary symbols.
3. **Removing Stopwords** – Removes common words (e.g., "the", "is") that do not add meaning.
4. **Tokenization** – Splits text into words or sentences.
5. **Lemmatization & Stemming** – Converts words to their root form.
6. **Removing Numbers** – Filters out numerical values.
7. **Removing Extra Whitespaces** – Ensures clean text formatting.

This study material covers these steps using **NLTK** and **SpaCy**, two powerful NLP libraries.

📌 Chapter 2: Text Preprocessing Using NLTK

NLTK (Natural Language Toolkit) is a Python library that provides text processing utilities for NLP.

2.1 Installing NLTK

```
!pip install nltk
```

2.2 Importing Required Libraries

```
import nltk

import re
```

```
import string
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('punkt')  # Tokenization
nltk.download('stopwords')  # Stopwords
nltk.download('wordnet')  # Lemmatization
```

2.3 Preprocessing Steps Using NLTK

🔹 1️⃣ Convert Text to Lowercase

```
text = "Natural Language Processing (NLP) is Amazing!"
text_lower = text.lower()
print(text_lower)
```

Output: "natural language processing (nlp) is amazing!"

🔹 2️⃣ Remove Punctuation & Special Characters

```
text_clean = re.sub(r'[^\w\s]', '', text_lower)  # Remove punctuation
print(text_clean)
```

Output: "natural language processing nlp is amazing"

🔹 3️⃣ Tokenization (Words & Sentences)

```
text = "Hello! How are you? NLP is exciting."

# Word Tokenization
word_tokens = word_tokenize(text)
print("Word Tokens:", word_tokens)

# Sentence Tokenization
sent_tokens = sent_tokenize(text)
print("Sentence Tokens:", sent_tokens)
```

Output:

```
Word Tokens: ['Hello', '!', 'How', 'are', 'you', '?', 'NLP', 'is', 'exciting', '.']
Sentence Tokens: ['Hello!', 'How are you?', 'NLP is exciting.']
```

🔹 4️⃣ Remove Stopwords

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

filtered_text = [word for word in word_tokens if word.lower() not in stop_words]

print("Filtered Text:", filtered_text)

Output: ['Hello', '!', 'NLP', 'exciting', '.']
(*"How", "are", "you", "is"* are removed)

🔹 5 Stemming (Reducing Words to Base Form)

stemmer = PorterStemmer()

stemmed_words = [stemmer.stem(word) for word in filtered_text]

print("Stemmed Words:", stemmed_words)

Output: ['hello', '!', 'nlp', 'excit', '.']
(*"exciting"* → *"excit"*)

🔹 6 Lemmatization (Converting to Root Word)

lemmatizer = WordNetLemmatizer()

lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_text]

print("Lemmatized Words:", lemmatized_words)

Output: ['Hello', '!', 'NLP', 'exciting', '.']
(*"exciting" → remains unchanged, unlike stemming*)

📌 Chapter 3: Text Preprocessing Using SpaCy

SpaCy is a more advanced NLP library optimized for efficiency.

3.1 Installing SpaCy

```
!pip install spacy

!python -m spacy download en_core_web_sm
```

3.2 Importing Required Libraries

```
import spacy


# Load English NLP Model

nlp = spacy.load("en_core_web_sm")
```

3.3 Preprocessing Steps Using SpaCy

🔹 1️⃣ Tokenization

```
text = "Hello! SpaCy is powerful for NLP."


# Process text with SpaCy

doc = nlp(text)
```

```
# Tokenize words
tokens = [token.text for token in doc]
print("Tokens:", tokens)
```

Output: ['Hello', '!', 'SpaCy', 'is', 'powerful', 'for', 'NLP', '.']

🔹 2 Removing Stopwords

```
tokens_no_stopwords = [token.text for token in doc if not token.is_stop]
print("Filtered Text:", tokens_no_stopwords)
```

Output: ['Hello', '!', 'SpaCy', 'powerful', 'NLP', '.']
(*"is", "for" removed*)

🔹 3 Lemmatization

```
lemmatized_text = [token.lemma_ for token in doc]
print("Lemmatized Text:", lemmatized_text)
```

Output: ['Hello', '!', 'SpaCy', 'be', 'powerful', 'for', 'NLP', '.']
(*"is" → "be"*)

🔹 4 Named Entity Recognition (NER)

```
text = "Apple is looking at buying OpenAI for $1 billion."
```

```
doc = nlp(text)


for ent in doc.ents:
    print(f"Entity: {ent.text}, Label: {ent.label_}")
```

Output:

Entity: Apple, Label: ORG

Entity: OpenAI, Label: ORG

Entity: $1 billion, Label: MONEY

(*Recognizes company names and monetary values*)

📌 Chapter 4: NLTK vs SpaCy – Which One to Use?

Feature	NLTK	SpaCy
Ease of Use	More manual work	Easier, built-in functions
Performance	Slower	Faster (optimized for efficiency)
Tokenization	Needs separate functions	Automatic & efficient
Stopword Removal	Requires manual list	Built-in

Stemming & Lemmatization	Uses separate modules	Integrated & faster
Named Entity Recognition (NER)	Not built-in	Built-in

👉 **Use NLTK for research & flexibility, and SpaCy for efficiency & production-ready applications.**

📌 Chapter 5: Case Study – Text Preprocessing for Sentiment Analysis

We use text preprocessing techniques before feeding data into **machine learning models**.

text = "I really love this movie! The acting was fantastic, and the plot was engaging."

Preprocess using SpaCy

doc = nlp(text)

cleaned_text = " ".join([token.lemma_ for token in doc if not token.is_stop and not token.is_punct])

print("Processed Text:", cleaned_text)

Output: "love movie acting fantastic plot engaging"
(*Removes stopwords, punctuation & applies lemmatization*)

📌 Chapter 6: Summary

✅ **Text preprocessing** is crucial for NLP tasks.
✅ **NLTK** is flexible but requires manual work.
✅ **SpaCy** is optimized for efficiency & real-world applications.
✅ **Tokenization, stopword removal, and lemmatization** are key preprocessing steps.

☑ Word Embeddings: Word2Vec, GloVe, FastText

📌 Chapter 1: Introduction to Word Embeddings

1.1 What are Word Embeddings?

Word embeddings are numerical vector representations of words that **capture their meanings, relationships, and context** in a high-dimensional space. Unlike traditional one-hot encoding, word embeddings:

- Represent words in a **continuous vector space**.
- Preserve **semantic and syntactic relationships**.
- Reduce **dimensionality** and **sparsity**.

1.2 Why are Word Embeddings Important?

✅ Improve performance in **Natural Language Processing (NLP) tasks** like sentiment analysis, translation, and search.
✅ Capture **contextual meaning**, enabling word similarity detection.
✅ Allow **transfer learning**, reusing pre-trained models for new tasks.

1.3 Types of Word Embeddings

There are three main techniques:

1. **Word2Vec** - Predicts words based on their context.
2. **GloVe** - Uses word co-occurrence matrices.

3. **FastText** - Uses subword information to handle rare words.

📌 Chapter 2: Word2Vec

2.1 What is Word2Vec?

Word2Vec is a deep learning-based word embedding technique developed by **Google** that represents words as dense vectors based on their **context in a large corpus**.

2.2 How Word2Vec Works?

Word2Vec is trained using two architectures:

1. **CBOW (Continuous Bag of Words)** - Predicts the **target word** from surrounding words.
2. **Skip-gram** - Predicts **context words** given a target word.

Method	Training Goal
CBOW	Predicts a word given its surrounding words.
Skip-gram	Predicts surrounding words given a single word.

2.3 When to Use Word2Vec?

✅ When dealing with **large corpora** (e.g., Wikipedia, news articles).
✅ When **word similarity and analogy** tasks are needed.
✅ When **speed is important**, as Word2Vec is fast and scalable.

2.4 Advantages of Word2Vec

✓ Captures **semantic relationships** (e.g., "king - man + woman = queen").

✓ Efficient to train on **large datasets**.
✓ Pre-trained models available (e.g., Google's Word2Vec).

2.5 Disadvantages of Word2Vec

✗ Does not handle **out-of-vocabulary (OOV) words**.
✗ Ignores **subword information** (e.g., prefixes/suffixes).

2.6 Implementing Word2Vec in Python

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize


# Sample corpus
sentences = [
   "Natural language processing is fascinating",
   "Word embeddings help understand text",
   "Deep learning models power NLP applications"
]


# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for
sentence in sentences]


# Train Word2Vec model
```

```
word2vec_model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1, workers=4)


# Get word vector

print(word2vec_model.wv['language'])
```

📌 Chapter 3: GloVe (Global Vectors for Word Representation)

3.1 What is GloVe?

GloVe (developed by Stanford) is a word embedding technique based on **word co-occurrence statistics**. Unlike Word2Vec, which uses local context, GloVe **captures both local and global relationships**.

3.2 How GloVe Works?

1. **Creates a word co-occurrence matrix** from a large text corpus.
2. **Factorizes the matrix** to learn word embeddings.
3. **Generates dense vectors** where similar words have similar representations.

3.3 When to Use GloVe?

✅ When working with **semantic similarity tasks** (e.g., word analogies).
✅ When using **pre-trained embeddings** (GloVe offers pre-trained vectors).
✅ When a **balanced global+local context** is needed.

3.4 Advantages of GloVe

✓ Captures both **global co-occurrence and local context**.
✓ Pre-trained models available (trained on Wikipedia, Twitter, etc.).
✓ Works well for **word similarity and analogy tasks**.

3.5 Disadvantages of GloVe

✗ Requires **more memory** to store word co-occurrence matrices.
✗ **Slower training** compared to Word2Vec.

3.6 Implementing GloVe in Python

```
import gensim.downloader as api


# Load pre-trained GloVe embeddings

glove_model = api.load("glove-wiki-gigaword-50")


# Get word vector

print(glove_model['language'])


# Find similar words

print(glove_model.most_similar('king'))
```

📌 CHAPTER 4: FASTTEXT

4.1 What is FastText?

FastText (developed by Facebook) is an improved version of Word2Vec that represents words as a **sum of subword vectors**. This makes it effective for handling **rare words and misspellings**.

4.2 How FastText Works?

- Unlike Word2Vec, FastText **breaks words into character n-grams** (e.g., "playing" → "pla", "lay", "ayi").
- Each word is represented by the **sum of its subword vectors**.
- This allows **better handling of out-of-vocabulary (OOV) words**.

4.3 When to Use FastText?

✅ When dealing with **morphologically rich languages** (e.g., German, Finnish).
✅ When handling **rare or misspelled words**.
✅ When **flexibility in word representation** is needed.

4.4 Advantages of FastText

✓ Handles **rare words and out-of-vocabulary (OOV) words**.
✓ Captures **morphological variations** (e.g., "running" vs. "runner").
✓ Works better for **low-resource languages**.

4.5 Disadvantages of FastText

✕ **Slower training** compared to Word2Vec.
✕ Larger model size due to **subword embeddings**.

4.6 Implementing FastText in Python

```
from gensim.models import FastText
```

```
# Train FastText model
fasttext_model = FastText(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1, workers=4)


# Get word vector
print(fasttext_model.wv['language'])


# Find similar words
print(fasttext_model.wv.most_similar('playing'))
```

📌 Chapter 5: Comparison of Word Embedding Techniques

Feature	Word2Vec	GloVe	FastText
Based On	Contextual prediction	Word co-occurrence matrix	Subword information
Handles OOV Words	❌ No	❌ No	✅ Yes
Best For	General NLP tasks	Word similarity tasks	Morphologically rich languages
Training Speed	⚡ Fast	🐢 Slow	🐌 Slower

📌 Chapter 6: Real-World Applications of Word Embeddings

Application	Embedding Used
Chatbots	FastText
Search Engines	Word2Vec
Machine Translation	GloVe
Sentiment Analysis	Word2Vec
Named Entity Recognition	FastText

📌 Chapter 7: Summary & Conclusion

7.1 Key Takeaways

✅ **Word2Vec** is great for **context-based** word representations.
✅ **GloVe** is best for **word similarity and analogy** tasks.
✅ **FastText** is powerful for **handling rare words and spelling variations**.

7.2 Next Steps

- **Try pre-trained embeddings** (Word2Vec, GloVe, FastText) on your dataset.
- **Use embeddings for sentiment analysis, chatbots, and text classification**.
- **Explore contextual embeddings** like **BERT and GPT**.

☑ Sentiment Analysis using ML & Deep Learning

📌 Chapter 1: Introduction to Sentiment Analysis

1.1 What is Sentiment Analysis?

Sentiment Analysis is a **Natural Language Processing (NLP) technique** that determines the **emotional tone** behind a text. It classifies opinions into categories such as:

- **Positive** 😊 (e.g., "I love this product!")
- **Negative** 😡 (e.g., "This service is terrible!")
- **Neutral** 😐 (e.g., "The movie was okay.")

1.2 Why is Sentiment Analysis Important?

- **Customer Feedback Analysis**: Understand customer opinions from reviews.
- **Brand Monitoring**: Track public sentiment about a company.
- **Market Research**: Analyze social media trends and customer preferences.
- **Political Analysis**: Study public reactions to political events.

📌 Chapter 2: Machine Learning Approach for Sentiment Analysis

2.1 Steps in Machine Learning-Based Sentiment Analysis

1. **Data Collection**: Gather labeled text data (e.g., movie reviews, tweets).
2. **Text Preprocessing**: Clean the text (removing stopwords, punctuation, etc.).
3. **Feature Extraction**: Convert text into numerical features (TF-IDF, Bag of Words).
4. **Model Training**: Train a Machine Learning model (e.g., Naïve Bayes, SVM, Logistic Regression).
5. **Evaluation**: Measure accuracy and optimize the model.

2.2 Text Preprocessing

Before feeding data into a model, we **clean and normalize** the text.

Steps in Preprocessing

✓ Convert text to **lowercase**
✓ Remove **punctuation & special characters**
✓ Remove **stopwords** (words like "the", "is", "and")
✓ Perform **stemming/lemmatization** (reduce words to root form)

Example of Preprocessing

```
import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer


nltk.download('stopwords')
```

```
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = text.split()
    words = [word for word in words if word not in stopwords.words('english')]  # Remove stopwords
    stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in words]  # Stemming
    return " ".join(words)

sample_text = "I really enjoyed the new movie! It was fantastic. 😊"
print(preprocess_text(sample_text))
```

2.3 Feature Extraction

Machine learning models cannot work directly with text, so we convert words into **numerical features**.

Methods of Feature Extraction

1. **Bag of Words (BoW)**: Counts word occurrences.

2. **TF-IDF (Term Frequency - Inverse Document Frequency)**: Weights words based on importance.
3. **Word Embeddings** (For Deep Learning): Converts words into vector representations.

Example: TF-IDF Feature Extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer


# Sample text corpus

documents = ["I love this movie", "This movie was terrible", "An amazing film"]


# Convert to TF-IDF features

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(documents)


# Display feature names

print(vectorizer.get_feature_names_out())

print(X.toarray())  # TF-IDF Matrix
```

2.4 Machine Learning Models for Sentiment Analysis

Common ML models used:

- **Naïve Bayes** (Fast & effective for text classification)

- **Logistic Regression** (Interpretable & efficient)
- **Support Vector Machines (SVMs)** (Good for small datasets)
- **Random Forest** (Handles imbalanced data well)

Example: Training a Sentiment Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Sample dataset
X_train, X_test, y_train, y_test = train_test_split(X, [1, 0, 1], test_size=0.2, random_state=42)

# Train Naïve Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

📌 Chapter 3: Deep Learning Approach for Sentiment Analysis

While traditional ML models rely on **hand-crafted features (TF-IDF, BoW)**, Deep Learning uses **Neural Networks** to learn features automatically from text.

3.1 Word Embeddings (Feature Representation)

Instead of simple word counts, **word embeddings** capture the meaning of words based on their context.

- **Word2Vec** (Google)
- **GloVe** (Stanford)
- **FastText** (Facebook)

Example: Converting Text to Word Embeddings

```
from gensim.models import Word2Vec


# Sample text corpus

sentences = [["I", "love", "this", "movie"], ["This", "movie", "is", "bad"]]


# Train Word2Vec model

model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
```

```
# Get vector representation of a word

print(model.wv['movie'])
```

3.2 Building a Deep Learning Model (LSTM)

LSTMs (Long Short-Term Memory networks) are powerful for NLP tasks as they **remember past words** to understand context.

Steps to Build an LSTM Sentiment Classifier

1. **Convert text into word embeddings** (Word2Vec, GloVe).
2. **Build an LSTM model** to process sequences.
3. **Train & evaluate the model**.

Example: Implementing Sentiment Analysis with LSTM

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences


# Sample dataset

texts = ["I love this movie", "This movie was terrible", "An amazing film"]

labels = [1, 0, 1]  # 1 = Positive, 0 = Negative
```

```
# Tokenization & padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
X = tokenizer.texts_to_sequences(texts)
X = pad_sequences(X, maxlen=5)

# Define LSTM model
model = Sequential([
  Embedding(input_dim=1000, output_dim=16, input_length=5),
  LSTM(16),
  Dense(1, activation='sigmoid')
])

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train model
model.fit(X, labels, epochs=10, verbose=1)
```

```
# Predict sentiment
print("Sentiment Prediction:", model.predict(X))
```

📌 Chapter 4: Comparing ML & Deep Learning Approaches

Feature	Machine Learning	Deep Learning
Feature Extraction	TF-IDF, BoW	Word Embeddings (Word2Vec, GloVe)
Algorithms	Naïve Bayes, SVM, Logistic Regression	LSTMs, CNNs, Transformers (BERT)
Computational Cost	Low	High
Accuracy	Good for small datasets	Better for large datasets
Interpretability	High	Low

📌 Chapter 5: Real-World Applications

- **E-Commerce**: Analyze product reviews (Amazon, Flipkart).
- **Social Media**: Track sentiment on Twitter & Facebook.
- **Finance**: Predict stock trends based on news sentiment.
- **Healthcare**: Analyze patient feedback.

📌 Chapter 6: Summary & Next Steps

✅ **Machine Learning** models (Naïve Bayes, SVM) work well for small datasets.
✅ **Deep Learning** (LSTMs, Transformers) captures contextual meaning better.
✅ Use **Word Embeddings** (Word2Vec, GloVe) instead of raw text features.
✅ Sentiment Analysis is widely used in **business, finance, and social media**.

Next Steps

- Implement **BERT (Transformer-based)** models for advanced NLP.
- Apply sentiment analysis to **real-world datasets** (Twitter, IMDB Reviews).
- Build a **Web App** using Flask to deploy the model.

☑ Building AI-Powered Chatbots with Dialogflow & GPT-4

📌 Chapter 1: Introduction to AI-Powered Chatbots

1.1 What is an AI Chatbot?

An **AI-powered chatbot** is a software application that uses **artificial intelligence (AI) and natural language processing (NLP)** to simulate human-like conversations with users.

1.2 Why Build AI-Powered Chatbots?

✅ Automates **customer support** and reduces workload.
✅ Available **24/7**, improving customer engagement.
✅ Provides **personalized recommendations**.
✅ Integrates with various platforms like **websites, WhatsApp, and Facebook Messenger**.

1.3 Key Technologies Behind AI Chatbots

1. **Natural Language Processing (NLP)** – Helps understand human language.
2. **Machine Learning (ML)** – Learns from past interactions to improve responses.
3. **Dialogflow** – A tool for building rule-based and AI-driven chatbots.
4. **GPT-4** – An advanced AI model for generating human-like responses.

📌 Chapter 2: Introduction to Dialogflow

2.1 What is Dialogflow?

Dialogflow (by Google) is a **cloud-based chatbot development platform** that helps create **conversational agents** for different applications.

2.2 Features of Dialogflow

- **Natural Language Understanding (NLU)** – Detects user intent and extracts relevant data.
- **Multi-Channel Support** – Works with WhatsApp, Facebook, Telegram, and websites.
- **Context Management** – Handles user conversations effectively.
- **Prebuilt Agents** – Ready-to-use chatbots for common tasks.

📌 Chapter 3: Creating a Chatbot with Dialogflow

3.1 Setting Up Dialogflow

Step 1: Create a Dialogflow Account

1. Go to Dialogflow Console.
2. Sign in with a **Google account**.
3. Click on **"Create Agent"** and give it a name.

Step 2: Define an Intent

1. Click **"Create Intent"**.
2. Add **training phrases** (e.g., "What are your services?", "Tell me about pricing").

3. Add **responses** (e.g., "We offer AI chatbot development services.").

Step 3: Test the Chatbot

1. Use the **built-in test console** in Dialogflow.
2. Type a message and check the bot's response.

📌 Chapter 4: Enhancing the Chatbot with GPT-4

4.1 What is GPT-4?

GPT-4 is an **advanced AI language model** developed by OpenAI. It can:

- **Understand context** better than previous versions.
- **Generate human-like responses**.
- **Handle multiple languages**.

4.2 Integrating GPT-4 with Dialogflow

To enhance your chatbot's intelligence, you can **connect GPT-4 to Dialogflow**.

Step 1: Get OpenAI API Key

1. Sign up at OpenAI.
2. Get your **API key** from the developer dashboard.

Step 2: Use Webhooks to Connect GPT-4

1. Create a **Webhook** in Dialogflow.

2. Write a Python function to send **Dialogflow user messages** to **GPT-4** and receive a response.

Example Code for GPT-4 Integration

```
import requests

import json


OPENAI_API_KEY = "your-api-key"


def chat_with_gpt(user_message):
    url = "https://api.openai.com/v1/chat/completions"
    headers = {"Authorization": f"Bearer {OPENAI_API_KEY}",
"Content-Type": "application/json"}


    payload = {
        "model": "gpt-4",
        "messages": [{"role": "user", "content": user_message}],
        "temperature": 0.7
    }


    response = requests.post(url, headers=headers,
data=json.dumps(payload))
    return response.json()["choices"][0]["message"]["content"]
```

```
# Test GPT-4 response
print(chat_with_gpt("What is AI?"))
```

Step 3: Deploy the Webhook in Dialogflow

1. Go to **Dialogflow Console** → **Fulfillment**.
2. Enable **Webhook** and add your **Python API URL**.
3. Modify **intent settings** to call the webhook for specific queries.

📌 Chapter 5: Deploying the Chatbot

5.1 Deploy on a Website

1. Go to **Dialogflow** → **Integrations**.
2. Click **"Web Demo"** and copy the provided **iframe code**.
3. Paste it into your website's **HTML file**.

5.2 Deploy on WhatsApp

To deploy on **WhatsApp**, use **Twilio for Dialogflow integration**:

```
from twilio.twiml.messaging_response import MessagingResponse

def whatsapp_reply(request):
    user_message = request.values.get("Body")
    bot_reply = chat_with_gpt(user_message)
```

```
response = MessagingResponse()

response.message(bot_reply)


return str(response)
```

- Get a **Twilio WhatsApp number**.
- Connect it to **Dialogflow via Webhook**.

📌 Chapter 6: Summary

Feature	Dialogflow	GPT-4
Purpose	Handles structured conversations	Generates natural responses
Training	Requires intent-based training	Learns dynamically
Best for	Rule-based chatbots	AI-powered conversations
Deployment	Web, WhatsApp, Telegram	API integration

📌 Chapter 7: Conclusion & Next Steps

7.1 Key Takeaways

✅ **Dialogflow** helps build structured chatbots.
✅ **GPT-4** adds human-like intelligence to responses.

✅ **Integration with webhooks** allows powerful AI capabilities.
✅ **Deployment is easy** across multiple platforms.

7.2 Next Steps

🚀 **Try building a customer support chatbot** with GPT-4.
🚀 **Deploy on WhatsApp, Telegram, or Messenger**.
🚀 **Train the bot on custom datasets** for better responses.

☑ Deploying AI Chatbots for Customer Support

📌 Chapter 1: Introduction to AI Chatbots

1.1 What is an AI Chatbot?

An **AI chatbot** is a software program that uses **artificial intelligence (AI)** to simulate human-like conversations with users. Chatbots can be rule-based or powered by **Natural Language Processing (NLP)** and **Machine Learning (ML)** to provide intelligent responses.

1.2 Why Use AI Chatbots for Customer Support?

AI chatbots help businesses **automate customer service,** reducing human effort while providing **instant responses** to customers.

✅ **24/7 Availability** – Handles customer queries anytime.
✅ **Faster Response Times** – Reduces wait time for customers.
✅ **Cost-Effective** – Saves costs on hiring support agents.
✅ **Scalable** – Can handle thousands of queries simultaneously.
✅ **Personalized Experience** – Uses past interactions to improve responses.

1.3 Types of Chatbots

Type	Description	Example
Rule-Based Chatbot	Uses predefined rules and scripts for responses	Basic FAQ bots

AI-Powered Chatbot	Uses NLP & ML to understand intent and learn from interactions	ChatGPT, Google Bard
Hybrid Chatbot	Combines rule-based logic with AI-driven responses	Many enterprise chatbots

---

📌 Chapter 2: Key Components of AI Chatbots

2.1 Natural Language Processing (NLP)

NLP allows chatbots to understand human language and intent. **Key NLP tasks include:**

- **Tokenization** – Breaking sentences into words.
- **Named Entity Recognition (NER)** – Identifies entities like names, dates, or locations.
- **Sentiment Analysis** – Detects emotions (positive, neutral, negative).

2.2 Machine Learning & AI

AI chatbots use ML algorithms to **learn from past conversations** and **improve their accuracy** over time.

2.3 Integration with Databases & APIs

Chatbots can **connect to databases, CRM systems, and third-party APIs** to retrieve real-time information.

2.4 User Interface (UI)

A chatbot can be deployed via:

- **Web chat** (Live chat widget on websites)
- **Messaging apps** (WhatsApp, Messenger, Telegram)
- **Voice assistants** (Alexa, Google Assistant)

📌 Chapter 3: Designing an AI Chatbot

3.1 Defining the Chatbot's Purpose

- What problems should the chatbot solve?
- Should it provide **basic FAQs, support tickets,** or **real-time solutions**?

3.2 Creating a Conversation Flow

A chatbot should have a structured flow:

1. **Greeting** – Welcome message.
2. **Understanding User Intent** – Identifying customer queries.
3. **Providing Responses** – Answering questions or escalating to human agents.
4. **Closing the Conversation** – Providing a resolution or feedback option.

3.3 Choosing the Right NLP Platform

Platform	Features
Dialogflow (Google)	NLP-powered chatbot with Google Cloud integration
IBM Watson Assistant	AI chatbot with enterprise-grade security

Microsoft Bot Framework	Build chatbots with Azure integration
Rasa (Open Source)	Customizable chatbot with Python support

📌 Chapter 4: Implementing an AI Chatbot

4.1 Install Dependencies

We will build a chatbot using **Python, TensorFlow, and NLTK**.

```
!pip install nltk tensorflow flask
```

4.2 Building a Simple AI Chatbot

```
import nltk

from nltk.chat.util import Chat, reflections


# Define chatbot conversation patterns

pairs = [

  ["hi|hello", ["Hello! How can I assist you today?"]],

  ["what is your name?", ["I'm an AI chatbot created for customer support."]],

  ["how can I track my order?", ["You can track your order using the tracking link sent to your email."]],

  ["bye|goodbye", ["Goodbye! Have a great day."]]

]
```

```
# Initialize chatbot
chatbot = Chat(pairs, reflections)
chatbot.converse()
```

4.3 Deploying a Web-Based Chatbot using Flask

```
from flask import Flask, render_template, request
from nltk.chat.util import Chat, reflections

app = Flask(__name__)

# Define chatbot conversation pairs
pairs = [
    ["hi|hello", ["Hello! How can I assist you today?"]],
    ["how to reset my password?", ["Visit the 'Forgot Password' section and follow the instructions."]],
    ["bye|goodbye", ["Goodbye! Have a great day."]]
]

chatbot = Chat(pairs, reflections)

@app.route("/")
```

```
def home():
    return render_template("chat.html")


@app.route("/get")
def get_bot_response():
    user_text = request.args.get("msg")
    return str(chatbot.respond(user_text))


if __name__ == "__main__":
    app.run(debug=True)
```

4.4 Connecting to a Customer Support System

To enhance chatbot functionality, integrate it with **CRM (e.g., HubSpot, Zendesk)** using APIs.

Example API call:

```
import requests

# Fetch customer support ticket data
response = requests.get("https://api.zendesk.com/tickets")
print(response.json())
```

📌 Chapter 5: Deploying an AI Chatbot

5.1 Deployment Options

Method	Platform
Web Application	Flask, Django, FastAPI
Cloud-Based Deployment	AWS, Google Cloud, Azure
Messaging Platforms	WhatsApp API, Telegram Bot API
Enterprise Integration	Slack, Microsoft Teams

5.2 Deploying on AWS

```
# Install AWS CLI

pip install awscli


# Deploy chatbot

aws lambda create-function --function-name ChatbotLambda --runtime python3.8 --handler chatbot.handler
```

📌 Chapter 6: Monitoring and Improving Chatbot Performance

6.1 Measuring Performance

✅ **Accuracy** – How often does the chatbot give correct responses?
✅ **Response Time** – How fast does the chatbot reply?
✅ **Customer Satisfaction (CSAT)** – Collected through feedback surveys.

6.2 Continuous Improvement

- **Retrain the chatbot** with more conversation data.
- **Use AI analytics tools** (e.g., Google Analytics, Power BI) to track chatbot usage.
- **Implement fallback responses** for unknown questions.

📌 Chapter 7: Real-World Use Cases

7.1 AI Chatbots in Different Industries

Industry	Use Case
E-commerce	Order tracking, product recommendations
Healthcare	Appointment scheduling, symptom checking
Banking	Transaction inquiries, fraud alerts
Travel & Hospitality	Flight booking, hotel reservations
Education	Student FAQs, online course support

7.2 Case Study: AI Chatbot for E-Commerce

Company: Amazon
Problem: High customer inquiries about order tracking.
Solution: AI chatbot that provides **real-time order status** using APIs.
Outcome: 30% reduction in support tickets and **faster response times**.

📌 Chapter 8: Summary

8.1 Key Takeaways

✅ AI chatbots improve **customer support efficiency** and **reduce costs**.
✅ **NLP and ML** allow chatbots to understand human language.
✅ Chatbots can be deployed on **websites, messaging apps, and cloud platforms**.
✅ Monitoring chatbot performance ensures **continuous improvement**.

📌 Chapter 9: Next Steps

🚀 **Experiment with ChatGPT API** for an advanced chatbot.
🚀 **Integrate a chatbot with WhatsApp** using Twilio API.
🚀 **Deploy a chatbot with voice support** using Google Assistant SDK.

📌 ✨ ASSIGNMENT 1:

🎯 BUILD A SENTIMENT ANALYSIS MODEL TO CLASSIFY CUSTOMER REVIEWS.

📌 ✨ Assignment Solution 1: Build a Sentiment Analysis Model to Classify Customer Reviews

---

🎯 Objective

The goal of this assignment is to build a **Sentiment Analysis Model** to classify customer reviews as **positive or negative** using **Natural Language Processing (NLP) and Machine Learning**.

We will: ✅ Load and preprocess a dataset of customer reviews
✅ Convert text into numerical data using **TF-IDF**
✅ Train a **Machine Learning model** to classify sentiment
✅ Evaluate the model's accuracy
✅ Test the model with new reviews

🛠 Step 1: Install and Import Required Libraries

We will use **pandas, sklearn, and nltk** for data preprocessing and model training.

1.1 Install Dependencies (if not installed)

```
!pip install pandas numpy scikit-learn nltk matplotlib seaborn
```

1.2 Import Required Libraries

```
import pandas as pd

import numpy as np

import re

import nltk
```

```
import matplotlib.pyplot as plt

import seaborn as sns


from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Download required NLTK data

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')
```

📥 Step 2: Load and Explore the Dataset

For this assignment, we will use a **customer review dataset** that contains text reviews and their sentiment labels (0 = Negative, 1 = Positive).

2.1 Load the Dataset

```
# Load dataset (Example dataset with 'review' and 'sentiment' columns)
```

```
df = pd.read_csv("https://raw.githubusercontent.com/laxmimerit/IMDB-Movie-Reviews-LSTM/master/IMDB%20Dataset.csv")


# Display first few rows

print(df.head())
```

2.2 Check Dataset Information

```
# Check for missing values and dataset info

print(df.info())


# Check class distribution

print(df['sentiment'].value_counts())
```

Expected Output:

- The dataset contains **50,000 reviews** labeled as "positive" or "negative".
- Balanced class distribution.

🧹 Step 3: Preprocess the Text Data

Text data needs **cleaning and tokenization** before training the model.

3.1 Define a Function to Clean Text

```
def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'<.*?>', '', text)  # Remove HTML tags
```

```
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    text = ' '.join(word for word in text.split() if word not in stopwords.words('english'))  # Remove stopwords
    return text
```

3.2 Apply Cleaning to Dataset

```
df['cleaned_review'] = df['review'].apply(clean_text)
print(df.head())
```

3.3 Convert Sentiment Labels to Binary (0 and 1)

```
df['sentiment'] = df['sentiment'].map({'positive': 1, 'negative': 0})
```

🔢 Step 4: Convert Text into Numerical Features

Since machine learning models cannot understand raw text, we use **TF-IDF Vectorization**.

```
vectorizer = TfidfVectorizer(max_features=5000)

# Transform text data into numerical format
X = vectorizer.fit_transform(df['cleaned_review'])

# Target variable (Sentiment: 0 = Negative, 1 = Positive)
y = np.array(df['sentiment'])
```

📊 Step 5: Split Data into Training and Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


print("Training Data Shape:", X_train.shape)

print("Testing Data Shape:", X_test.shape)
```

🤖 Step 6: Train the Sentiment Analysis Model

We will use **Multinomial Naïve Bayes**, which is effective for text classification.

```
# Train Naïve Bayes Model

model = MultinomialNB()

model.fit(X_train, y_train)


print("Model Training Complete ✅ ")
```

📌 Step 7: Evaluate the Model

We check the **accuracy, classification report, and confusion matrix**.

7.1 Accuracy Score

```
# Predict on test set

y_pred = model.predict(X_test)


# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Model Accuracy: {accuracy:.2f}")
```

7.2 Classification Report

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

7.3 Confusion Matrix

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```

📝 Step 8: Test the Model on New Customer Reviews

We test the model on custom **customer reviews**.

```
def predict_sentiment(text):
    text = clean_text(text)  # Clean text
    text_tfidf = vectorizer.transform([text])  # Convert to TF-IDF
    prediction = model.predict(text_tfidf)[0]  # Make prediction
    return "Positive" if prediction == 1 else "Negative"


# Example reviews

review1 = "This product is amazing! It exceeded my expectations."

review2 = "I had a terrible experience. The quality was very poor."
```

```
print("Review 1 Sentiment:", predict_sentiment(review1))
print("Review 2 Sentiment:", predict_sentiment(review2))
```

✅ Final Summary

Step	Description
Step 1	Install and import necessary libraries
Step 2	Load and explore the dataset
Step 3	Preprocess text (cleaning, tokenization)
Step 4	Convert text into numerical features
Step 5	Split data into training and testing sets
Step 6	Train a **Naïve Bayes model**
Step 7	Evaluate model performance
Step 8	Test on new customer reviews

🌟 Conclusion

- **Sentiment analysis** is useful for understanding customer feedback.
- The **Naïve Bayes model** efficiently classifies sentiment.
- **TF-IDF** converts text into a numerical format suitable for ML models.
- **Preprocessing (cleaning & tokenization) is crucial** for accuracy.

📌 ✨ ASSIGNMENT 2:

🎯 DEVELOP AN AI-POWERED CHATBOT FOR A RETAIL BUSINESS.

📌 ✨ Assignment Solution 2: Develop an AI-Powered Chatbot for a Retail Business

---

🎯 Objective

The goal of this assignment is to **develop an AI-powered chatbot** using **Natural Language Processing (NLP) and Machine Learning** to assist customers in a retail business. The chatbot will:

✅ Answer frequently asked questions (FAQs).
✅ Assist customers in **product recommendations**.
✅ Handle **order tracking** queries.

We will implement the chatbot using **Python, TensorFlow/Keras, and the Rasa or ChatterBot library**.

🛠️ Step 1: Install and Import Necessary Libraries

We will use:

- **NLTK (Natural Language Toolkit)** for text preprocessing.
- **ChatterBot** for chatbot logic.
- **Flask** for deploying the chatbot.

🔹 Install Dependencies

```
!pip install nltk chatterbot chatterbot_corpus flask
```

🔹 Import Required Libraries

```
import nltk
```

```
from nltk.chat.util import Chat, reflections

from chatterbot import ChatBot

from chatterbot.trainers import ChatterBotCorpusTrainer

from flask import Flask, request, jsonify
```

📥 Step 2: Define Chatbot Responses Using Rule-Based Approach

Before training a chatbot with machine learning, let's implement a **simple rule-based chatbot**.

```
# Define chatbot responses using pattern matching

pairs = [

  (r"(hi|hello|hey)", ["Hello! How can I assist you today?"]),

  (r"what are your store hours?", ["Our store is open from 9 AM to 9 PM."]),

  (r"do you have (.*)", ["Yes, we have %1 in stock. Would you like to place an order?"]),

  (r"how can I track my order?", ["You can track your order using the tracking link sent to your email."]),

  (r"bye", ["Goodbye! Have a great day!"])

]


# Create chatbot

chatbot = Chat(pairs, reflections)


# Start conversation
```

```
while True:
    user_input = input("You: ")
    if user_input.lower() in ["bye", "exit"]:
        print("Chatbot: Goodbye!")
        break
    print("Chatbot:", chatbot.respond(user_input))
```

📊 Step 3: Train a Machine Learning-Based Chatbot

Instead of manually defining rules, we train a chatbot using **ChatterBot**.

🔹 Create and Train the Chatbot

```
# Create chatbot
chatbot = ChatBot("RetailBot")

# Train chatbot with English corpus
trainer = ChatterBotCorpusTrainer(chatbot)
trainer.train("chatterbot.corpus.english")

# Train chatbot with custom retail-related questions
trainer.train([
    "What products do you sell?",
    "We sell clothing, electronics, and home essentials.",
    "How can I return an item?",
```

```
    "You can return items within 30 days with a receipt.",
    "Do you offer discounts?",
    "Yes, we have seasonal discounts and loyalty programs."
])
```

- **Test the Chatbot**

```
response = chatbot.get_response("Do you offer discounts?")
print("Chatbot:", response)
```

📌 Step 4: Deploy the Chatbot Using Flask

Now, let's integrate the chatbot into a **web application**.

- **Create a Flask API for the Chatbot**

```
app = Flask(__name__)

@app.route("/chat", methods=["POST"])
def chat():
    user_input = request.json["message"]
    response = chatbot.get_response(user_input)
    return jsonify({"response": str(response)})

if __name__ == "__main__":
    app.run(port=5000)
```

- **Test the API**

Send a **POST request** to http://127.0.0.1:5000/chat with:

{

"message": "Do you sell electronics?"

}

Expected response:

{

"response": "Yes, we sell clothing, electronics, and home essentials."

}

📌 Step 5: Integrate the Chatbot With WhatsApp or Telegram

You can integrate this chatbot with **WhatsApp (Twilio API)** or **Telegram Bot API**.

- **WhatsApp Integration Using Twilio**

1. **Sign up** on Twilio.
2. **Get API credentials**.
3. **Send messages to Twilio from your Flask chatbot**.

from twilio.rest import Client

Twilio credentials

ACCOUNT_SID = "your_account_sid"

AUTH_TOKEN = "your_auth_token"

WHATSAPP_NUMBER = "whatsapp:+14155238886" # Twilio sandbox number

```
# Initialize Twilio client
client = Client(ACCOUNT_SID, AUTH_TOKEN)

def send_whatsapp_message(message, recipient):
    client.messages.create(
        body=message,
        from_=WHATSAPP_NUMBER,
        to=f"whatsapp:{recipient}"
    )

# Example usage
send_whatsapp_message("Hello! How can I assist you?", "+123456789")
```

📌 Step 6: Advanced Features (Optional Enhancements)

🔹 Add Product Recommendations Using Machine Learning

Use NLP and a recommendation engine to suggest products.

```
import random

# Sample products
products = {
    "clothing": ["T-Shirt", "Jeans", "Jacket"],
```

```
    "electronics": ["Laptop", "Smartphone", "Headphones"],
    "home essentials": ["Blender", "Microwave", "Vacuum Cleaner"]
}

def recommend_product(category):
    return random.choice(products.get(category.lower(), ["No recommendations available."]))

# Test recommendation system
print(recommend_product("electronics"))
```

- **Sentiment Analysis for Customer Queries**

Integrate **sentiment analysis** to handle angry customers better.

```
from textblob import TextBlob

def analyze_sentiment(user_message):
    sentiment_score = TextBlob(user_message).sentiment.polarity
    if sentiment_score < -0.2:
        return "I'm sorry to hear that. How can I assist you?"
    return "Thank you for reaching out! How can I help?"

# Test sentiment analysis
print(analyze_sentiment("I'm really unhappy with my order!"))
```

✅ Final Summary

Step	Description
Step 1	Install and import necessary libraries
Step 2	Implement a rule-based chatbot
Step 3	Train a machine learning chatbot using ChatterBot
Step 4	Deploy chatbot using Flask API
Step 5	Integrate chatbot with WhatsApp or Telegram
Step 6	Add product recommendations & sentiment analysis

🌟 Conclusion

- **Chatbots improve customer support and engagement** in retail.
- **Machine learning enhances chatbot intelligence** over time.
- **Deployment using Flask allows easy integration** with various platforms.
- **Additional features like recommendations and sentiment analysis** improve user experience.

📌 Next Steps

- **Enhance the chatbot using deep learning (Transformer models like GPT)**.
- **Deploy on cloud services** (AWS, GCP, or Azure).
- **Integrate chatbot into a website or mobile app**.