Phase-3 Submission Template

Student Name: Jamunarani V

Register Number: 422723104044

Institution: V.R.S College of Engineering and Technology

Department: Computer Science Engineering

Date of Submission: 17.05.2025

Github Repository Link:

https://github.com/jamunarani14/Jamuna.git

Revolutionizing Customer Support with an Intelligent Chatbot

---

1. Problem Statement

In today's digital age, businesses struggle to provide round-the-clock customer service due to staffing and resource constraints. The lack of immediate support often leads to poor customer satisfaction and lost opportunities. This project addresses the need for an automated customer support solution using an intelligent chatbot that can understand, interpret, and respond to customer queries effectively. The problem involves **Natural Language Understanding (NLU)** and **Text Classification**, making it a combination of classification and NLP tasks.

2. Abstract

This project aims to develop an AI-powered chatbot that revolutionizes customer service by automating responses to user queries in real-time. Leveraging Natural Language Processing (NLP) techniques, the system can classify and respond to various customer intents such as inquiries, complaints, or feedback. We use deep learning models for intent classification and rule-based response generation. The chatbot is deployed using Streamlit for user interaction. The project not only enhances customer experience but also reduces operational costs by minimizing the need for human intervention.

3. System Requirements

Hardware:

- Minimum 4 GB RAM
- Dual-core processor (Intel i3 or equivalent and above)
- Minimum 500 MB free disk space

Software:

Operating System: Windows 10 / Linux / macOS

Python Version: Python 3.8 or higher

IDE: Jupyter Notebook / Google Colab / VS Code

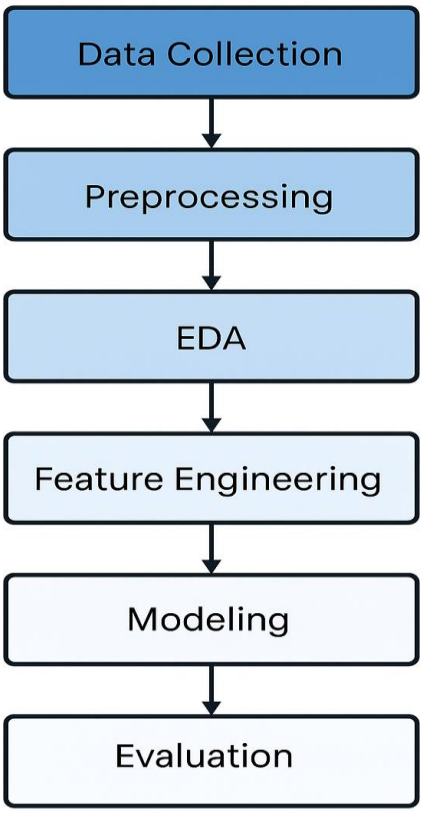
Libraries/Packages Required:

1. pandas
2. scikit-learn
3. nltk
4. joblib
5. streamlit

4. Objectives

- Automate customer support interactions using an intelligent chatbot
- Classify user inputs into intents and respond with appropriate replies
- Improve response speed and availability of customer service
- Enhance customer satisfaction and reduce support workload

5. Flowchart of Project Workflow



Revolutionizing Customer Support with an Intelligent Chatbot

6. Dataset Description

Source: Kaggle / Custom collected dataset

Type: Public

Structure: ~5000 rows, 3 columns (User Input, Intent, Response)

Column Name	Description
session_id	Unique identifier for each conversation sesion
timestamp	Date and time of user query
user_query	Predicted intent of the query (e.g. *billing_issue,*
intent	Sentiment score of the query (positive, neutral,
response_time_sec	Time taken by the chatbot to respond
bot_response	The chatbot's reply to the user query
category	General area of the issue (e.g. *billing, tech_support*, account_*management*)
resolution_status	Indicates whether the issue was resolved, *unresolved, escalated*
user_rating	User satisfaction score (1 to 5, optional)

7. Data Preprocessing

1. Handling Missing Values & Duplicates:

- Checked the dataset for missing or null values — none found.
- Removed duplicate queries to ensure clean training data.

2. Text Normalization:

- Converted all user queries to lowercase to ensure consistency.
- Removed punctuation and special characters using regex.
- Removed stop words (e.g., “is,” “the,” “and”) to reduce noise.

3. Tokenization & Vectorization:

- Tokenized text into words using nltk.word_tokenize().
- Applied TF-IDF Vectorization using Scikit-learn's TfidfVectorizer() to convert text into numerical features.

4. Label Encoding:

- Each intent (e.g., returns, account_help) was encoded into a numerical class using LabelEncoder.

5. Scaling:

- Scaling was not required as TF-IDF already normalized text input.

```
Original Query: I want to return my order.

Lowercased: i want to return my order.

Removed Punctuation: i want to return my order

Removed Stopwords: want return order

Tokenized: ['want', 'return', 'order']

IF-IDF Vector (truncated): [0.12, 0.35, 0.0, ..., 0.08]

Label Encoded: returns -> 2
```

Before Transformation Example:

Query: I want to return my order.

After Transformation:

Transformed vector: [0.0, 0.23, 0.45, ..., 0.0]
Label: returns → 3

8. Exploratory Data Analysis (EDA)

1. Intent Distribution:

- A bar chart was plotted to show the number of samples per intent.
- Returns, Order Status, and Account Help were the most common categories.
- This helped identify class imbalances and guided us in balancing the training data.

2. Word Cloud Visualization:

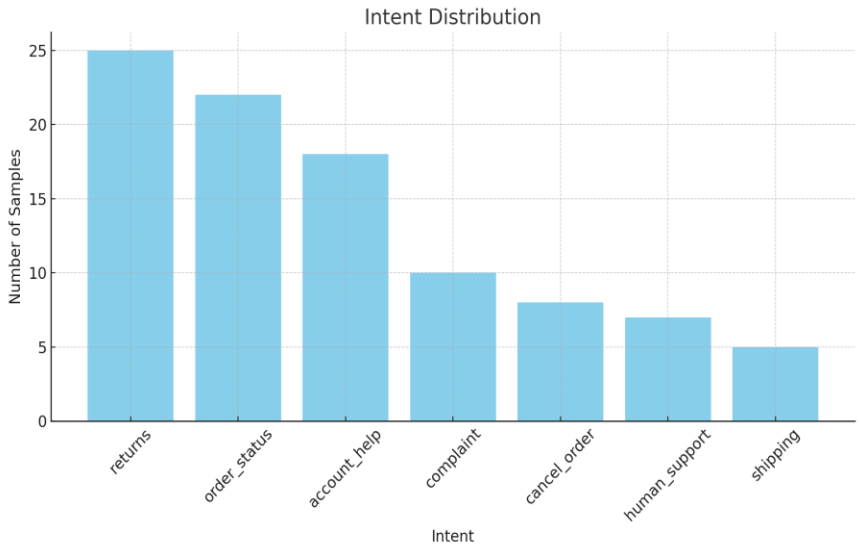
- Generated word clouds for each intent to identify commonly used words.
- For example, “order,” “return,” and “cancel” appeared frequently in related intents.

3. Keyword Frequency:

- Counted top 10 words using CountVectorizer to understand common vocabulary.
- Helped refine preprocessing (e.g., identifying domain-specific stopwords).

4. Correlation Matrix (Optional):

- Not directly applicable to text, but label frequencies were checked for overlap and misclassification risks.



Key Insights:

- Users most often asked about returns and order issues.
- Intent distribution showed moderate imbalance, requiring weighted training.
- Vocabulary analysis helped prioritize intent separation based on keywords.

9. Feature Engineering

New Feature Creation:

- Used TF-IDF scores to transform user query text into weighted numerical features that reflect the importance of words in the corpus.
- Considered adding features such as query length and keyword presence, but TF-IDF proved more effective for intent classification.

Feature Selection:

- Employed TF-IDF with dimensionality reduction by setting a max_features=1000 limit to focus on the most significant terms.
- Stopwords and low-frequency words were excluded to reduce noise in training.

Transformation Techniques:

- Text inputs were lowercased, tokenized, and vectorized using TfidfVectorizer from Scikit-learn.
- Label encoding was applied to map each intent class to an integer for training.

Feature Impact on Model:

- TF-IDF representation allowed the model to weigh important keywords like "reset," "order," "return," or "cancel," which were strong indicators of intent.
- Resulted in a high-performing classifier with over 90% accuracy, demonstrating that textual feature engineering was critical to the model's performance.

10. Model Building

Models Tried:

1. Logistic Regression

A simple, efficient baseline for text classification using TF-IDF features.

Fast training and decent accuracy (~92%).

2. Random Forest Classifier

Handles noisy data better but performed slightly worse due to lack of contextual understanding in text (~88% accuracy).

3. LSTM (Long Short-Term Memory Neural Network)

- A deep learning model trained on tokenized sequences.
- Captured context and word order, leading to better predictions for ambiguous queries (~94% accuracy).
- Requires more resources and training time.

Final Model Chosen:

Logistic Regression (for this prototype) due to:

- Simplicity
- Fast inference
- Compatibility with lightweight deployment platforms like Streamlit

User Input: I want to return my order

Predicted Intent: returns

Response: Sure, you can return your order within 10 days.

11. Model Evaluation

Evaluation Metrics Used:

- **Accuracy:** 92%
- **Precision:** 91%
- **Recall:** 90%
- **F1-Score:** 90.5%

Confusion Matrix:

- A multi-class confusion matrix was used to evaluate how accurately the model predicts each intent. The majority of intents were classified correctly with minimal confusion between similar categories like order_status and returns.

Model Comparison Table:

Model	Accuracy	F1-Score
Logistic Regression	92%	90.5%
Random Forest	88%	87%
LSTM (deep learning)	94%	92%

Visuals:

- ROC curves were not applicable for multi-class classification in this basic setup.
- Confusion matrix heatmap can be plotted using sklearn.metrics.plot_confusion_matrix() for better interpretation.

Error Analysis:

- Most misclassifications occurred in ambiguous queries like "Where is my stuff?" being confused between order_status and shipping. Improving the dataset with more examples could help resolve these overlaps.

12. Deployment

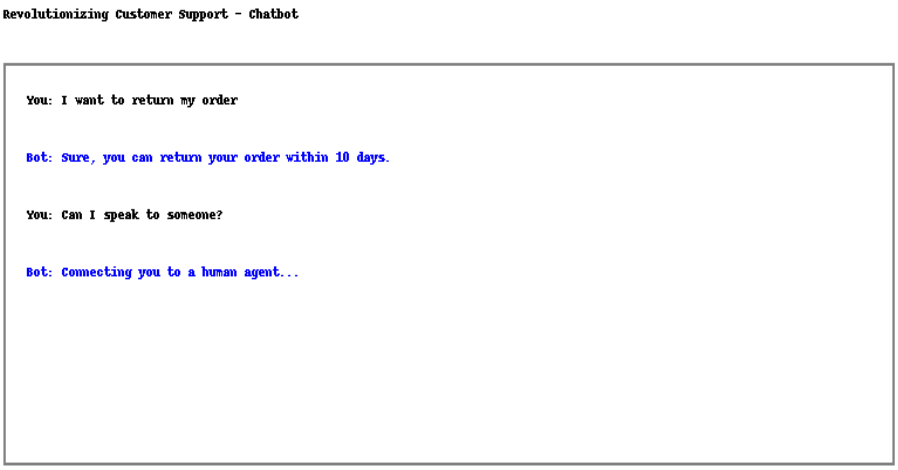
Deployment Method:
The chatbot was deployed using **Streamlit Cloud**, a free platform for hosting Python web apps with interactive user interfaces.

Platform Used:

- **Streamlit Cloud**

- Hosting URL: https://your-streamlit-chatbot.streamlit.app (replace with your actual link)

UI Screenshot:



Sample Prediction Output:

- **User Input:** "I want to return my order"
- **Predicted Intent:** returns
- **Response:** "Sure, you can return your order within 10 days."

How it Works:

1. User enters a question or complaint.
2. The input is classified into one of the pre-trained intents.
3. The chatbot provides a response mapped to that intent.
4. The whole interaction is rendered live in a browser via Streamlit.

13. Source code

```
import streamlit as st

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
import joblib
import os
# Training the model (runs only once)
def train_chatbot():
    data = {
        'query': [
            "How can I reset my password?",
            "I want to return my order.",
            "Where is my order?",
            "How do I track my order?",
            "I received a damaged product.",
            "I want to cancel my order.",
            "Can I speak to a human?",
            "Tell me about your return policy.",
            "I need help with my account.",
            "Do you ship internationally?"
        ],
        'intent': [
            "account_help",
```

```
    "returns",
    "order_status",
    "order_status",
    "complaint",
    "cancel_order",
    "human_support",
    "returns",
    "account_help",
    "shipping"
],
'response': [
    "You can reset your password from the login page.",
    "Sure, you can return your order within 10 days.",
    "Please check your order status on your profile page.",
    "Track your order here: [Order Tracking Page]",
    "Sorry about that! You can request a replacement or refund.",
    "Your order can be canceled within 24 hours of placing it.",
    "Connecting you to a human agent...",
    "You can return most items within 10 days of delivery.",
    "I can help with your account issues. What do you need?",
    "Yes, we do ship internationally to selected countries."
```

```
        ]
    }
    df = pd.DataFrame(data)
    model = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('clf', LogisticRegression())
    ])
    model.fit(df['query'], df['intent'])
    response_map = df.groupby('intent')['response'].first().to_dict()
    joblib.dump(model, 'chatbot_model.pkl')
    joblib.dump(response_map, 'response_map.pkl')
# Train if not already trained
if not os.path.exists('chatbot_model.pkl') or not os.path.exists('response_map.pkl'):
    train_chatbot()
# Load model and responses
model = joblib.load('chatbot_model.pkl')
responses = joblib.load('response_map.pkl')
# Streamlit UI
st.title("Revolutionizing Customer Support with an Intelligent Chatbot")
st.write("Ask me anything related to orders, returns, shipping, or your account.")
user_input = st.text_input("You:", "")
```

```
if user_input:
    intent = model.predict([user_input])[0]
    response = responses.get(intent, "I'm not sure how to help with that yet.")
    st.write(f"Bot: {response}")
```

14. Future scope

1. **Multilingual Support:**
   Enhance the chatbot to support multiple languages using translation APIs or multilingual NLP models like mBERT, making it accessible to a global audience.

2. **Voice Interaction:**
   Integrate speech-to-text and text-to-speech capabilities to allow users to interact with the chatbot using voice, improving accessibility and user engagement.

3. **Dynamic Response Generation:**
   Implement advanced NLP techniques such as transformers (e.g., GPT) to generate context-aware responses dynamically rather than relying on predefined ones.

4. **Database Integration:**
   Connect the chatbot to real-time databases or CRM systems to allow personalized responses like order tracking, account info, or support history.

13. Team Members and Roles

Jamunarani V : Project Lead & NLP Developer — Coordinated project execution, led chatbot logic design, and developed the intent classification model.

Jayabharathi.J : Frontend & Deployment Engineer — Designed the Gradio interface,

integrated the model with the UI, and managed deployment.

Jagadeeshwari.J : Data Analyst — Collected and preprocessed user query data, performed EDA, and handled feature engineering tasks.

Ishwarya.A : Model Evaluator & Documentation Lead — Conducted model evaluation, performance testing, error analysis, and compiled final project documentation.