**Phase-2 Submission Template**

**Student Name:** Ram prasath S

**Register Number:** 712523104048

**Institution:**PPG INSTITUTE OF TECHNOLOGY

**Department:**COMPUTER SCIENCE ENGINEERING

**Date of Submission:** 16.05.2025

**Github Repository Link:** [**https://github.com/Ram-Prasath-12/ram-prasath**](https://github.com/Ram-Prasath-12/ram-prasath)

### **Problem Statement**

In today’s fast-paced digital world, businesses are overwhelmed with a high volume of customer queries across multiple channels. Traditional customer support systems, which rely heavily on human agents, are often inefficient, slow, and unable to provide 24/7 assistance. This leads to delayed responses, poor user experiences, and increased operational costs.

The problem lies in automating customer support using an intelligent chatbot that can understand and respond to user queries effectively. Based on our enhanced understanding of the dataset—comprising customer queries, intents, and responses—we aim to design a system that classifies user inputs into predefined categories (intents) and delivers appropriate automated responses. Therefore, this is a multi-class classification problem, where the chatbot must identify the correct intent behind a user query.

### **2. Project Objectives**

* Intent Classification: Build a machine learning model (e.g., using NLP and deep learning) to accurately classify user queries into predefined intents.
* Response Generation: Integrate a response system that provides relevant, context-aware replies based on the predicted intent.
* Text Preprocessing & Embedding: Implement robust text preprocessing (tokenization, lemmatization, stop-word removal) and use word embeddings (e.g., Word2Vec, BERT) to convert queries into meaningful vector representations.
* Model Evaluation & Optimization: Evaluate the model using metrics such as accuracy, precision, recall, and F1-score, and fine-tune it for optimal performance.
* Deployment & Integration: Deploy the chatbot as a real-time web application or API, and ensure it can be integrated with customer-facing platforms.

### **3. Flowchart of the Project Workflow**

* Data Collection

Source: Predefined dataset of customer queries, intents, and responses

* Data Preprocessing

Tokenization & lemmatization

Label encoding (for intents)

* Feature Extraction

Convert text to numerical vectors using methods like TF-IDF, Word2Vec, or BERT

* Model Selection & Training

Choose algorithms (e.g., Logistic Regression, LSTM, or Transformer models)

Train on labeled data (input: query, output: intent)

* Model Evaluation

Metrics: Accuracy, Precision, Recall, F1-score

Confusion Matrix to evaluate performance

* Response Mapping

Link predicted intent to predefined responses

Add fallback responses for unknown intents

### 

### **4. Data Description**

Dataset Name: Chatbot Intents Dataset (or a similar name, based on the actual file)

Source: Kaggle (or specify another if different)

Type of Data: Unstructured text data (user queries) with structured labels (intents)

Number of Records: Approximately 4,000 to 10,000 user queries

Number of Features:

patterns: user input texts

tags: intent labels (target variable)

responses: predefined chatbot responses

Target Variable: tags (intent classification label)

Data Format: Usually stored in JSON or CSV

### **5. Data Preprocessing**

1. *Load Dataset*

*Import pandas as pd*

*Import json*

*# For JSON format*

*With open(‘intents.json’) as f:*

*Data = json.load(f)*

*# Flatten the JSON into a DataFrame*

*Intents = []*

*For intent in data[‘intents’]:*

*For pattern in intent[‘patterns’]:*

*Intents.append({‘pattern’: pattern, ‘tag’: intent[‘tag’]})*

*Df = pd.DataFrame(intents)*

1. *Handle Missing Values*

*# Check for missing values*

*Print(df.isnull().sum())*

*# Drop rows with missing values if any*

*Df.dropna(inplace=True)*

*No imputation needed as all rows are expected to have text and labels.*

1. *Remove Duplicate Records*

*# Check for duplicates*

*Print(“Duplicates:”, df.duplicated().sum())*

*# Remove duplicates*

*Df.drop\_duplicates(inplace=True)*

1. *Outlier Detection (Text Length Based)*

*Df[‘text\_length’] = df[‘pattern’].apply(len)*

*Df = df[(df[‘text\_length’] > 2) & (df[‘text\_length’] < 300)] # basic bounds*

*Df.drop(‘text\_length’, axis=1, inplace=True)*

1. *Text Normalization and Cleaning*

*Import re*

*Import nltk*

*From nltk.corpus import stopwords*

*From nltk.stem import WordNetLemmatizer*

*Nltk.download(‘stopwords’)*

*Nltk.download(‘punkt’)*

*Nltk.download(‘wordnet’)*

*Stop\_words = set(stopwords.words(‘english’))*

*Lemmatizer = WordNetLemmatizer()*

*Def clean\_text(text):*

*Text = text.lower()*

*Text = re.sub(r’[^a-zA-Z\s]’, ‘’, text)*

*Tokens = nltk.word\_tokenize(text)*

*Tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words]*

*Return ‘ ‘.join(tokens)*

*Df[‘cleaned\_pattern’] = df[‘pattern’].apply(clean\_text)*

### 

### **6. Exploratory Data Analysis (EDA)**

**1. Distribution of Query Lengths**

import matplotlib.pyplot as plt

import seaborn as sns

df['text\_length'] = df['cleaned\_pattern'].apply(len)

plt.figure(figsize=(8, 4))

sns.histplot(df['text\_length'], bins=30, kde=True)

plt.title("Distribution of Query Lengths")

plt.xlabel("Length of Query (characters)")

plt.ylabel("Frequency")

plt.show()

Insight: Most customer queries are short (10–100 characters), indicating they are direct and to-the-point.

**2. Intent Frequency Count**

plt.figure(figsize=(10, 6))

sns.countplot(y='tag', data=df, order=df['tag'].value\_counts().index)

plt.title("Frequency of Each Intent")

plt.xlabel("Count")

plt.ylabel("Intent")

plt.show()

Insight: Some intents (e.g., greetings, goodbye, thanks) are much more common, while others are less represented. This may require class balancing for modelling.

#### **Bivariate / Multivariate Analysis**

**3. Query Length vs. Intent**

plt.figure(figsize=(12, 6))

sns.boxplot(x='tag', y='text\_length', data=df)

plt.xticks(rotation=90)

plt.title("Query Length Distribution by Intent")

plt.show()

Insight: Some intents have longer average query lengths, such as problem-related intents, while simple intents like greetings are shorter.

**4. Word Cloud by Intent (Optional Visualization)**

from wordcloud import WordCloud

intent\_text = df[df['tag'] == 'booking']['cleaned\_pattern'].str.cat(sep=' ')

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(intent\_text)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.title("Common Words in 'booking' Intent")

plt.show()

Insight: Helps visualize common keywords within specific intents.

**5. Intent Correlation (if using TF-IDF or Bag of Words)**

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

cv = CountVectorizer()

X\_counts = cv.fit\_transform(df['cleaned\_pattern'])

X\_df = pd.DataFrame(X\_counts.toarray(), columns=cv.get\_feature\_names\_out())

X\_df['intent'] = df['tag']

# Compute mean presence of words by intent

intent\_word\_means = X\_df.groupby('intent').mean()

plt.figure(figsize=(12, 8))

sns.heatmap(intent\_word\_means.T.iloc[:25], cmap='Blues')

plt.title("Top 25 Word Frequencies Across Intents")

plt.xlabel("Intent")

plt.ylabel("Words")

plt.show()

### 

### **7. Feature Engineering**

1. **Query Length Features**

Created features that capture the complexity and verbosity of the query.

# Character count

Df[‘char\_count’] = df[‘cleaned\_pattern’].apply(len)

# Word count

Df[‘word\_count’] = df[‘cleaned\_pattern’].apply(lambda x: len(x.split()))

Justification: Longer queries are more likely to represent complex intents (e.g., complaints or booking), while short queries are often greetings or farewells. Including these as features helps the model distinguish them better.

1. **Presence of Keywords (Optional Enhancement)**

**Df[‘has\_cancel’] = df[‘cleaned\_pattern’].apply(lambda x: 1 if ‘cancel’ in x else 0)**

**Df[‘has\_booking’] = df[‘cleaned\_pattern’].apply(lambda x: 1 if ‘book’ in x else 0)**

**Justification: Keywords that strongly signal specific intents (e.g., “cancel” for cancel\_order) can boost accuracy, especially in smaller datasets.**

1. **Embedding Text Features (TF-IDF / Word2Vec / BERT)**

**Used TF-IDF for converting text into numerical vectors. (Word2Vec/BERT can be used in advanced models.)**

**From sklearn.feature\_extraction.text import TfidfVectorizer**

**Vectorizer = TfidfVectorizer(max\_features=500) # Limit for efficiency**

**X\_tfidf = vectorizer.fit\_transform(df[‘cleaned\_pattern’])**

**Justification: TF-IDF captures both frequency and uniqueness of words, offering better representation than raw counts.**

1. **Feature Concatenation**

**Combine TF-IDF vectors with engineered numeric features.**

**Import scipy.sparse**

**X\_numeric = df[[‘char\_count’, ‘word\_count’, ‘has\_cancel’, ‘has\_booking’]].values**

**From sklearn.preprocessing import StandardScaler**

**X\_numeric = StandardScaler().fit\_transform(X\_numeric)**

**# Combine sparse TF-IDF with dense features**

**X\_final = scipy.sparse.hstack((X\_tfidf, X\_numeric))**

**Y\_final = df[‘**

### **8. Model Bu****ilding**

### **Logistic Regression (Multinomial)**

* + Suitable for baseline performance in text classification.
  + Efficient, interpretable, and works well with TF-IDF features.

1. **Random Forest Classifier**
   * Captures nonlinear relationships and interactions.
   * Handles categorical target variables well and is less prone to overfitting than single decision trees.

### **Data Splitting**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_final, y\_final, test\_size=0.2, stratify=y\_final, random\_state=42

)

Stratification ensures balanced class distribution in training and test sets.

### **Model 1: Logistic Regression**

from sklearn. linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

lr\_model = LogisticRegression(max\_iter=1000)

lr\_model.fit(X\_train, y\_train)

y\_pred\_lr = lr\_model.predict(X\_test)

print("Logistic Regression Performance:\n")

print(classification\_report(y\_test, y\_pred\_lr))

### **Model 2: Random Forest Classifier**

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(n\_estimators=100)

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

print("Random Forest Performance:\n")

print(classification\_report(y\_test, y\_pred\_rf))

### **Evaluation Metrics Used**

* **Accuracy**: Overall correctness.
* **Precision**: Correctness of positive predictions.

### **9. Visualization of Results & Model Insights***.*

1. *Confusion Matrix*

*From sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay*

*Cm = confusion\_matrix(y\_test, y\_pred\_rf)*

*Disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=label\_encoder.classes\_)*

*Disp.plot(xticks\_rotation=’vertical’, cmap=’Blues’)*

*Plt.title(“Confusion Matrix – Random Forest”)*

*Plt.show()*

1. *Classification Report Summary (Bar Chart)*

*From sklearn.metrics import precision\_recall\_fscore\_support*

*Import numpy as np*

*Precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred\_rf, average=None)*

*Intents = label\_encoder.classes\_*

*Plt.figure(figsize=(12, 5))*

*X = np.arange(len(intents))*

*Plt.bar(x – 0.2, precision, width=0.2, label=’Precision’)*

*Plt.bar(x, recall, width=0.2, label=’Recall’)*

*Plt.bar(x + 0.2, f1, width=0.2, label=’F1-score’)*

*Plt.xticks(x, intents, rotation=90)*

*Plt.title(“Per-Class Performance Metrics – Random Forest”)*

*Plt.legend()*

*Plt.tight\_layout()*

*Plt.show()*

1. *Feature Importance (Random Forest)*

*Importances = rf\_model.feature\_importances\_*

*Feature\_names = vectorizer.get\_feature\_names\_out().tolist() + [‘char\_count’, ‘word\_count’, ‘has\_cancel’, ‘has\_booking’]*

*# Combine and sort top 20 features*

*Feat\_df = pd.DataFrame({‘feature’: feature\_names, ‘importance’: importances})*

*Top\_feats = feat\_df.sort\_values(by=’importance’, ascending=False).head(20)*

*Plt.figure(figsize=(10, 6))*

*Sns.barplot(x=’importance’, y=’feature’, data=top\_feats)*

*Plt.title(“Top 20 Important Features – Random Forest”)*

*Plt.show()*

### 

### **10. Tools and Technologies Used**

*Python: Chosen for its simplicity, rich ecosystem, and strong support for data science and NLP.*

*IDE / Notebook Environment*

*Google Colab: Used for model development, EDA, and visualization. Provides free GPU support and seamless collaboration.cal execution.)*

*Machine Learning Models Used:*

*Logistic Regression: Baseline linear model.*

*Random Forest Classifier: For non-linear intent classification.*

*(Optional: Extendable to XGBoost, LSTM, or BERT for future enhancements.)*

*Visualization Tools:*

*Seaborn & Matplotlib: Used for EDA, performance metrics, and feature importance visualizations.*

*(Optional Advanced Tools: Plotly for interactive plots or Tableau/Power BI for dashboards.)*

### 

### **11. Team Members and Contributions**

|  |  |  |
| --- | --- | --- |
| **NAME** | **ROLE** | **REG\_NO** |
| **DHARUN D** | **LEADER** | **712523104015** |
| **GOKUL JB** | **MEMBER** | **712524104015** |
| **AGILESHWARAN S** | **MEMBER** | **712523104001** |
| **RAM PRASATH S** | **MEMBER** | **712523104048** |
| **UDHAYAKUMAR R** | **MEMBER** | **712523104064** |