

CHAT BOT

Chatbot Integration with ML & Dialogflow

Name: Eluru Naga Chandrika

Date: 02/10/24



Abstract:

The project offers the development of a conversational chatbot powered by **machine learning** and **integrated with Dialogflow** for the automation of customer service interactions. The chatbot **identifies user intents** from natural language inputs and **furnishes the relevant responses** accordingly. A **logistic regression** model was prepared on a dataset of customer service dialogues with techniques like TF-IDF vectorization and label encoding. This model, integrated with a **Flask-based webhook** and **Dialogflow**, enables the chatbot to effectively manage queries from order cancellations to customer support.

Introduction:

The market for **smart, automated customer service solutions** is greater today than ever before. Tipped by **machine learning (ML)** and **natural language processing (NLP)**, Chatbots have surfaced as an effective weapon for businesses to streamline their customer interactions even better. The project is about **developing a conversational chatbot** which uses **machine learning** for intent recognition and **Dialog flow** for natural language understanding and integration.

The chatbot is built to **recognize what your user's type (utterances)** and categorize these into different customer intents and **provide appropriate responses** by easing some kinds of customer service tasks that order cancellations, requests for information or feedback. The input dataset which is used to train the machine learning model is customer service dialogues where utterances are associated with intents. This was the dataset which would be used to train the model.

Our goal here is to **predict the right intent** for this input from the user.

The result consumer-facing interface for interacting with the chat bot leverages **Google Dialog flow** in order to integrate the machine-learning model that was trained earlier. Towards this, a **web-interface is developed with flask** which makes use of chatbot and communicates as per the chats. Using this API along with ML makes the chatbot capable of dynamically responding to an enormous range of user input accurately, creating a more streamlined experience for the user.

In this document, I will show you what it takes to build a chatbot step by step:

- **data preprocessing**
 - **model training**
 - **Dialog flow integration**
 - **web deployment.**
-

Problem Statement:

Since it is an era of automation, prompt response to the questions asked by customers can prove very beneficial in enhancing customer satisfaction. The original approaches to customer service rely on human agents, which is likely to become less and less sustainable as demand increases. By using NLP and ML, a chatbot can be created to solve this issue by automatically responding to customers and answering their questions 24/7.

An example of the project is the following one implemented with a machine learning model that classifies into orders, cancellations, invoices and other call intents. In response to the input of the user, the chatbot should accurately identify this intent and execute relevant instructions.

Dataset Description

The dataset used for this project is the Bitext Sample Customer Service Training Dataset. This dataset includes multiple customer service interactions with the following key features:

- **Utterance:** The customer query or statement.
 - **Category:** The broad category of the service issue.
 - **Intent:** The specific action the customer wants to take, such as placing an order or checking a refund.
-

Data Preprocessing and ML

Before feeding the data into the machine learning model, we applied several preprocessing steps to ensure the text data was clean and structured.

Steps:

1. **Handling Missing Values:** Checked for any null values in the dataset and cleaned them if necessary.
2. **Text Cleaning:** Applied natural language processing (NLP) techniques such as:
 - Removing special characters.
 - Lowercasing the text.
 - Removing extra spaces.
3. **Label Encoding:** Converted the categorical labels (intents) into numerical format using Label Encoding.
4. **Feature Extraction:** Applied **TF-IDF Vectorization** to convert the text (user utterances) into numerical vectors that the machine learning model can interpret.

First **Load the dataset**

- View the first few rows
- Check for null values

Code:

```
import pandas as pd

df = pd.read_csv('Bitext_Sample_Customer_Service_Training_Dataset.csv')

df.head()

df.isnull().sum()
```

- Plot of distribution of 'intent'

-Plot of distribution of 'category'

Code:

```
import seaborn as sns

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10,6))
sns.countplot(data=df, x='intent')
plt.title('Distribution of Intents')
plt.xticks(rotation=45)
plt.show()
```

```
plt.figure(figsize=(10,6))
sns.countplot(data=df, x='category')
plt.title('Distribution of Categories')
plt.xticks(rotation=45)
plt.show()
```

-apply Natural language processing (nlp) to clean text in ‘utterance’

Code:

```
import re

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

def clean_text(text):
    text = re.sub(r'\W', ' ', text) # Removes special characters
    text = re.sub(r'\s+', ' ', text) # Removes extra spaces
    return text.lower()

df['utterance'] = df['utterance'].apply(clean_text)
```

-Model training

I chose **Logistic Regression** as the classification model for predicting the user's intent based on the user utterance.

Steps:

1. **Splitting Data:** The data was split into training (80%) and testing (20%) sets.
2. **Vectorization:** The utterances were vectorized using **TF-IDF** to transform text into numerical features.
3. **Training:** The Logistic Regression model was trained using the processed data.
4. **Evaluation:** The model's accuracy was evaluated on the test data.

Code Overview:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
import joblib

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y_encoded, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)

accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

- Save the Trained Model (pickle files)

```
joblib.dump(model, 'chatbot_model/model.pkl')
joblib.dump(tfidf, 'chatbot_model/tfidf_vectorizer.pkl')
joblib.dump(label_encoder, 'chatbot_model/label_encoder.pkl')
```

Dialog flow Integration with Flask-based Webhook

The trained machine learning model was integrated with Dialogflow to allow users to interact with the chatbot. Dialogflow communicates with the Flask server via a webhook, sending user queries and receiving responses from the model.

Steps for Integration:

- 1. Create a Dialogflow Agent:**

- Set up an agent in Dialogflow for customer service interactions.

- 2. Create Intents:**

- Create multiple intents that the chatbot will respond to, such as `cancel_order`, `place_order`, etc.

- 3. Set up a Webhook:**


- In Dialogflow, navigate to Fulfillment and enable the webhook. Enter the ngrok URL to forward requests to your Flask server.

- 4. Test the Chatbot:**

- Use the Dialogflow console to test the chatbot. The webhook sends the user's query to the Flask app, which responds based on the intent classified by the model.

Prototype using dialogflow:

Try it now




Agent

USER SAYS

COPY CURL

report payment issue

 DEFAULT RESPONSE

▼

Your 'payment_issue' request is accepted. Go to my_account click on payments click payment_issue , write your issue then click on 'send' button.

CONTEXTS

RESET CONTEXTS

__system_counters__

INTENT

payment_issue

ACTION

Not available

DIAGNOSTIC INFO

Creating html application for chatbot interface

chatbot.html

Code overview:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Chatbot</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      background-color: #f4f4f4;

      margin: 0;

      padding: 20px;

    }

    #chat-container {

      max-width: 600px;

      margin: auto;

      border: 1px solid #ccc;

      border-radius: 10px;

      background-color: #fff;

      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    }

    #chat-area {

      padding: 10px;

      height: 400px;
```

```
    overflow-y: auto;

    border-bottom: 1px solid #ccc;

    display: flex;

    flex-direction: column;
}

.message {

    margin: 5px 0;

    padding: 10px;

    border-radius: 5px;

    max-width: 75%;

    word-wrap: break-word;
}

.user-message {

    background-color: #d1e7dd;

    align-self: flex-end;
}

.bot-message {

    background-color: #f8d7da;

    align-self: flex-start;
}

#user-input-container {

    display: flex;

    padding: 10px;
}

#user-input {

    flex: 1;

    padding: 10px;

    border: 1px solid #ccc;

    border-radius: 5px;
}
```

```

button {
  padding: 10px 15px;
  margin-left: 10px;
  border: none;
  border-radius: 5px;
  background-color: #007bff;
  color: white;
  cursor: pointer;
}
button:hover {
  background-color: #0056b3;
}
</style>
</head>
<body>
  <div id="chat-container">
    <div id="chat-area"></div>
    <div id="user-input-container">
      <input type="text" id="user-input" placeholder="Type a message..." onkeydown="if(event.key
=== 'Enter'){sendMessage();}">
      <button onclick="sendMessage()">Send</button>
    </div>
  </div>

<script>
function sendMessage() {
  const userMessage = document.getElementById('user-input').value;

  const userDiv = document.createElement('div');
  userDiv.className = 'message user-message';

```

```

userDiv.textContent = userMessage;

document.getElementById('chat-area').appendChild(userDiv);


document.getElementById('user-input').value = "";


fetch('https://5d8f-117-198-141-197.ngrok-free.app/webhook', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    queryResult: {
      queryText: userMessage
    }
  })
})

.then(response => response.json())
.then(data => {
  const botReply = data.fulfillmentText;

  const botDiv = document.createElement('div');
  botDiv.className = 'message bot-message';
  botDiv.textContent = botReply;
  document.getElementById('chat-area').appendChild(botDiv);

  document.getElementById('chat-area').scrollTop = document.getElementById('chat-
area').scrollHeight;

})

.catch(error => {
  console.error('Error:', error);

```

```
});
```

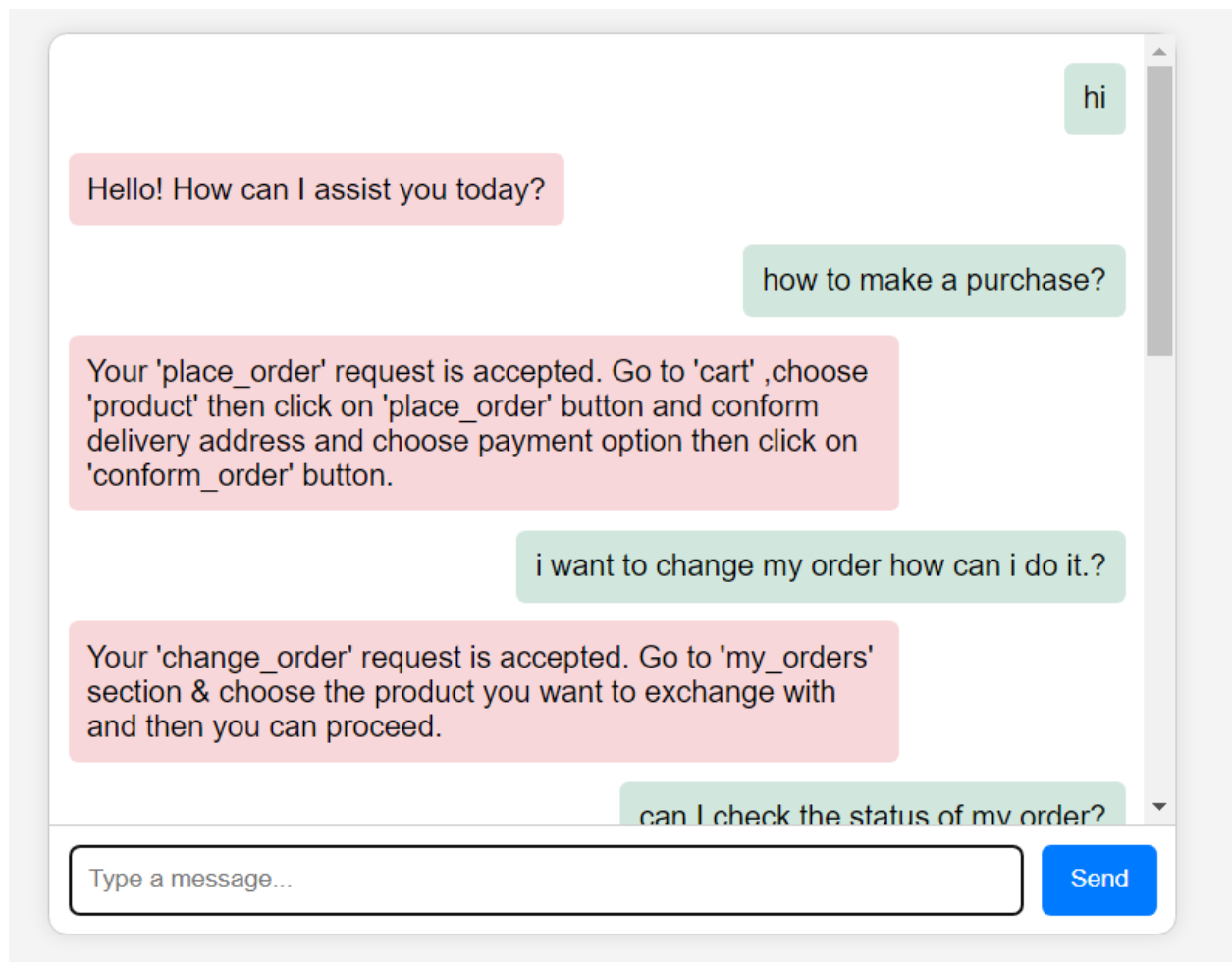
```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Prototype



Open browser and run: (<http://localhost:8000/chatbot.html>) to see chatbot interface

Tools and Technologies Used

1. Python

- Python is the core programming language used for developing the chatbot. Its libraries make it an ideal choice for natural language processing, machine learning, and web frameworks.
- Version: Python 3.x

2. Flask

- Flask is a lightweight web framework used to build the webhook that interacts with Dialogflow. It handles HTTP requests, processes user queries, and returns appropriate responses based on the machine learning model's predictions.
- Use: Building a webhook to handle queries from Dialogflow.

3. Ngrok

- Ngrok is used to create a secure tunnel to your local server, allowing Dialogflow to communicate with your local Flask application through a publicly accessible URL.
- Use: Exposing the local Flask server to the internet for Dialogflow integration.

4. Dialog flow

- Dialog flow is a natural language understanding platform that allows you to design and integrate conversational agents (chatbots). It processes user inputs, detects intents, and passes them to the webhook for further processing.
- Use: Managing intents and handling user interactions.

5. Pandas

- Pandas is a data manipulation library used for loading, processing, and analyzing the dataset in this project.
- Use: Data preprocessing and exploratory data analysis.

6. Seaborn & Matplotlib

- These are used for data visualization, particularly for understanding the distribution of categories and intents in the dataset.
- Use: Plotting visualizations for data distribution.

7. Scikit-learn

- Scikit-learn is used for building the machine learning model, including preprocessing, feature extraction, and classification. Specific modules used include:

- TfidfVectorizer: For converting text into numerical features.
- LabelEncoder: For encoding target labels into numerical form.
- LogisticRegression: For training the classification model.
- Train-test split: For splitting the dataset into training and testing sets.
- Use: Preprocessing, feature extraction, model training, and evaluation.

8. Joblib

- Joblib is used for saving and loading the trained machine learning model, TF-IDF vectorizer, and other preprocessing objects.

- Use: Persisting the model and preprocessing objects for later use.

Conclusion:

This chatbot project properly applied a machine learning model to manage customer support queries. The chatbot can act as an interactive conversational agent as it integrates into Dialogflow, providing automated responses based on what users input. Using this kind of chatbot, various common customer service tasks may be streamlined through accurate intent classification and responding with a Flask-based webhook, thus improving overall user experience.