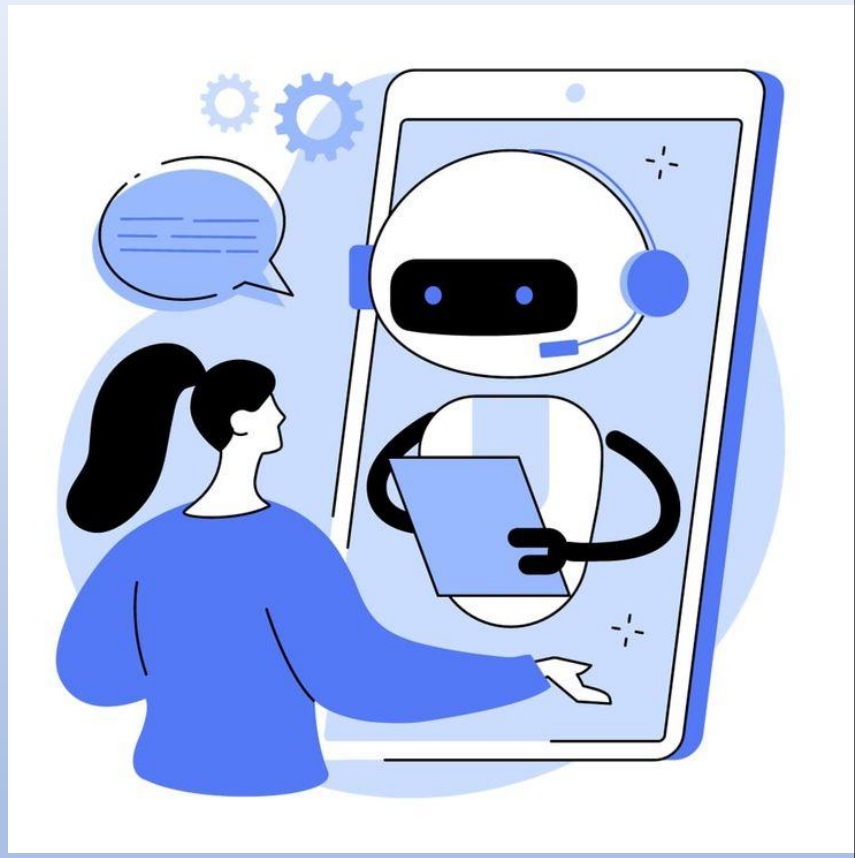
Revolutionizing Customer Support with AI-Driven Chatbot Solutions

Table of Contents

Introduction

- *In today's fast-paced, digital-first world, customer service plays a critical role in shaping customer experiences and business success. To meet the growing demand for immediate, reliable, and personalized support, I have developed an AI-powered chatbot.*
- *This intelligent assistant is specifically designed to address common customer issues such as account creation, password recovery, and payment troubleshooting.*
- *By automating these routine tasks, the chatbot significantly reduces response times, enhances customer satisfaction, and allows human agents to focus on more complex inquiries.*
- *My solution not only streamlines the customer journey but also provides round-the-clock assistance, ensuring that customers can resolve their issues anytime, anywhere.*



Scope of the Project

The AI-powered chatbot is designed to handle a wide range of customer support tasks to streamline the user experience and reduce reliance on human agents. The chatbot's scope includes addressing common customer service inquiries such as:

1. Order Management: Assisting customers in placing, changing, or canceling orders, and tracking order status.

2. Account Management: Helping users create, edit, delete, or switch accounts, recover passwords, and resolve registration problems.

3. Shipping and Delivery Support: Providing information on delivery options, updating shipping addresses, tracking deliveries, and estimating delivery periods.

4. Payment and Refund Assistance: Addressing payment issues, offering information on payment methods, checking invoices, processing refunds, and tracking refund status.

5. Customer Support and Escalation: Managing customer complaints, facilitating contact with customer service or human agents, and providing responses to general inquiries.

6. Policy and Subscription Information: Informing customers about the refund policy, cancellation fees, and assisting with newsletter subscriptions.

This comprehensive scope ensures that customers have instant access to the information they need, enabling faster resolutions and an enhanced service experience.

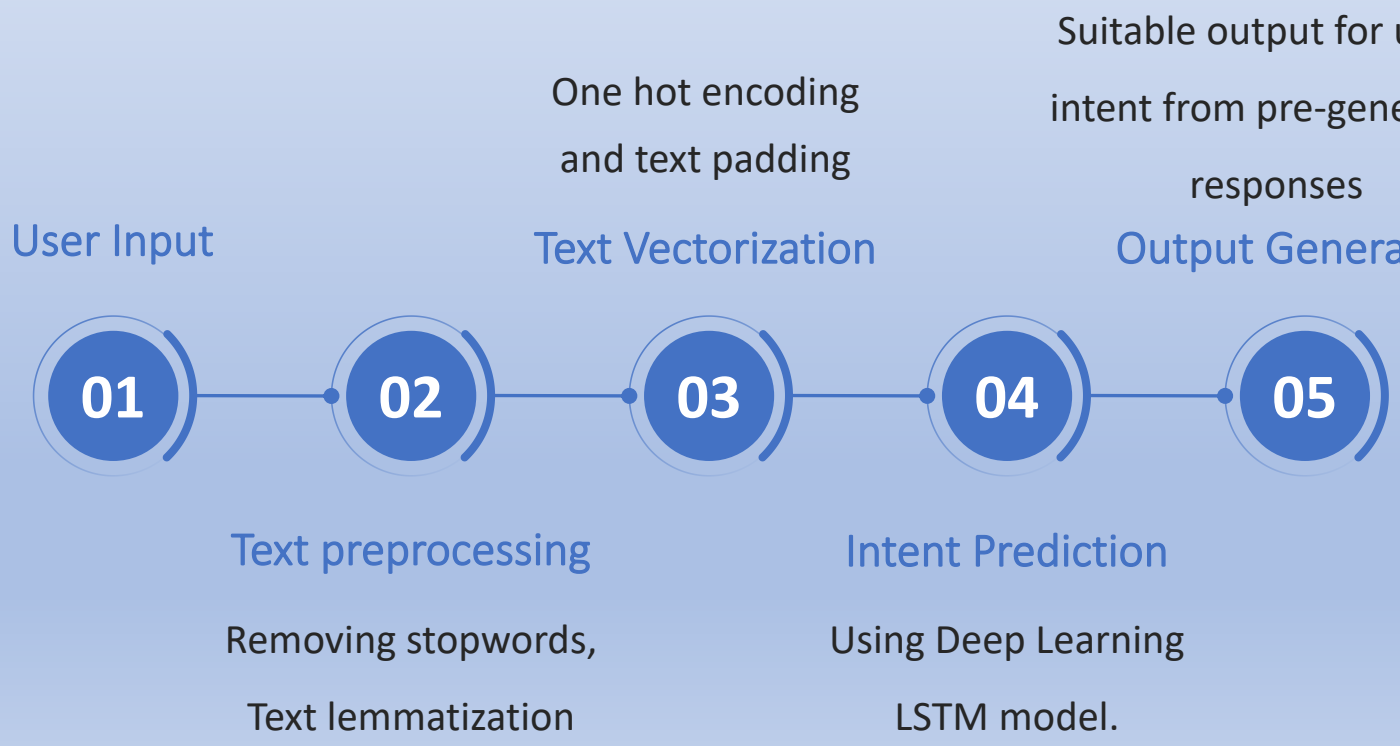
System Architecture & Workflow:

The chatbot operates on a rule-based system, first identifying the user's intent from their prompt and then responding with an appropriate solution from a pre-defined set of responses tailored to that intent.

Let's break down the process step by step to understand how the chatbot functions and generates responses (each step will be explained in detail in the upcoming slides):

- *The user's input prompt is captured and stored.*
- *Standard text preprocessing is applied, such as lemmatization and stopword removal.*
- *The processed input is then transformed into a vector representation for model prediction.*
- *A deep learning model is employed to classify the user's intent based on the input data.*

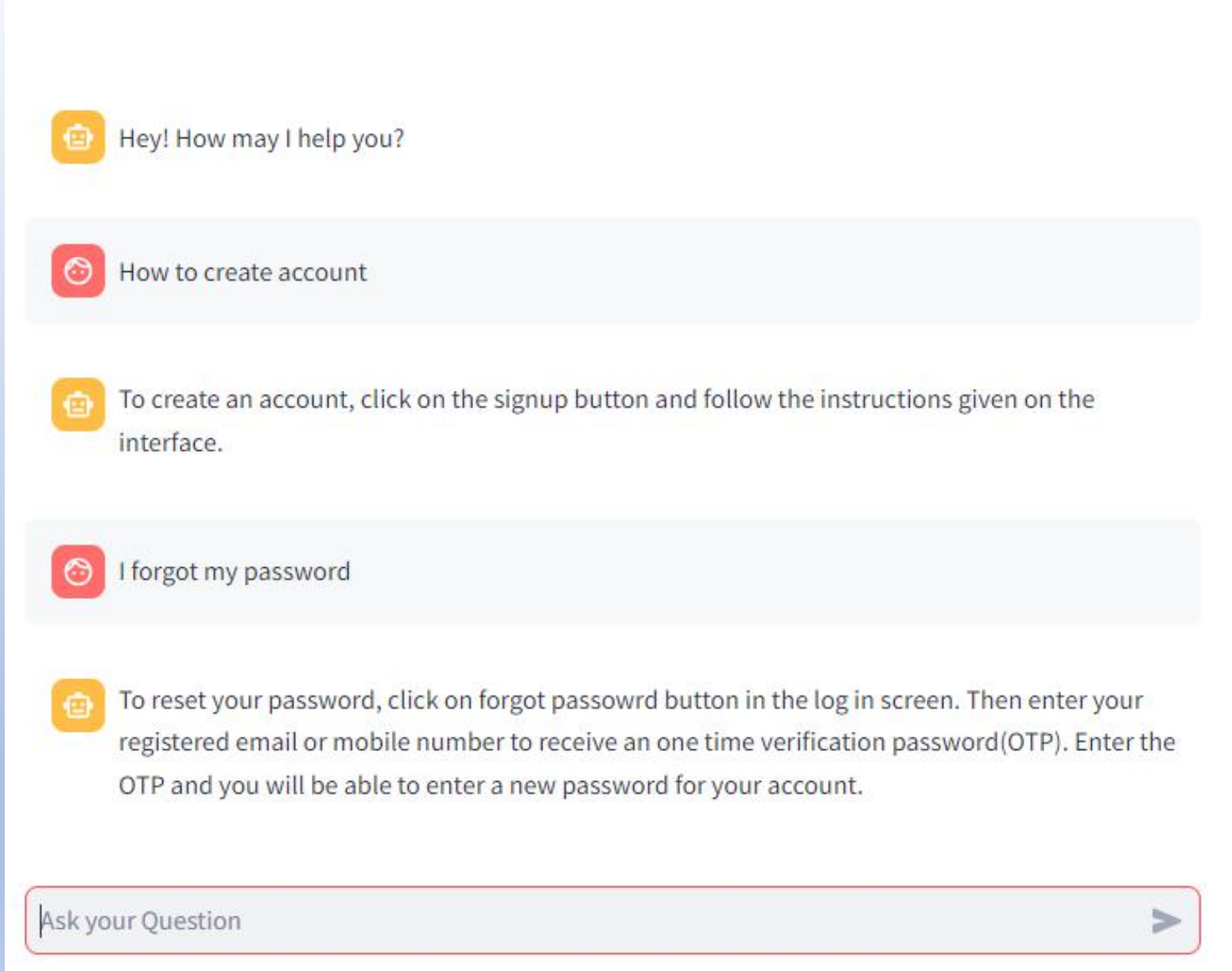
- *The model outputs the probability of the prompt belonging to each possible intent class.*
- *The intent with the highest probability is selected as the final class.*
- *A response is then randomly chosen from the pre-generated set of responses for that intent and displayed to the user.*



Model Overview:

The model receives user issue prompts as input and responds with the most suitable solution from a pre-generated set of responses, addressing common and frequently occurring problems efficiently.

Model Webapp - Click Here



Technical Deep Dive: Behind the Chatbot's functionality

ETL: Quick overview of the Data

Code:

```python
# importing pandas for data manipulation

import pandas as pd
```

Python

```python
data = pd.read_csv("./Bitext_Sample_Customer_Service_Training_Dataset.csv")   # Loads the data into a dataframe
data.head()     # Shows 5 rows from the top of the data
```

Python

Output:

	flags	utterance	category	intent
0	BM	I have problems with canceling an order	ORDER	cancel_order
1	BIM	how can I find information about canceling ord...	ORDER	cancel_order
2	B	I need help with canceling the last order	ORDER	cancel_order
3	BIP	could you help me cancelling the last order I ...	ORDER	cancel_order
4	B	problem with cancelling an order I made	ORDER	cancel_order

ETL: Quick overview of the Data

The Dataset contains four different columns:

1. Flags: Annotations for linguistic phenomena, which can be used to adapt bot training to different user language profiles. Some flags are:
   - B - Basic syntactic structure
   - S - Syntactic structure
   - L - Lexical variation (synonyms), etc.

2. Utterance: The user input prompt to the bot.
3. Category: The category of the query that is prompted by the user.
4. Intent: The intention of the user behind the query.

Even though for the labelled dataset we have four columns, for the new prompts we know that the input will just be the user's query. That's why we only used the data in the utterance column to classify the intent of the user behind the query. So we used utterance as the only feature and intent as the label to train our model.

ETL: Quick overview of the Data

Code:

```
data["intent"].unique()          # list all the unique intents in the data
```

Output:

```
array(['cancel_order', 'change_order', 'change_shipping_address',
       'check_cancellation_fee', 'check_invoice', 'check_payment_methods',
       'check_refund_policy', 'complaint', 'contact_customer_service',
       'contact_human_agent', 'create_account', 'delete_account',
       'delivery_options', 'delivery_period', 'edit_account',
       'get_invoice', 'get_refund', 'newsletter_subscription',
       'payment_issue', 'place_order', 'recover_password',
       'registration_problems', 'review', 'set_up_shipping_address',
       'switch_account', 'track_order', 'track_refund'], dtype=object)
```

We have customer's prompt for 27 different types of queries which we used to train our chatbot, to assist our users with further queries.

ETL: Quick overview of the Data

Code:

```
print(data.shape)       # prints shape of the data i.e number of rows and columns
data[["intent","category"]].value_counts()      # total records for different intents
```

The shape of our data, i.e the number of rows and column is (8175,4) means 8175 rows and 4 columns.
The second line of code shows the number of records we have for every type of query intent. We can observe that we have almost equal number of records for every type of query, so we can easily use this data for our model training without the model getting biased by a particular intent.

Output:

```
(8175, 4)

intent                    category
get_invoice               INVOICE              324
check_invoice             INVOICE              324
payment_issue             PAYMENT              323
review                    FEEDBACK             315
track_refund              REFUND               308
set_up_shipping_address   SHIPPING_ADDRESS     307
place_order               ORDER                306
track_order               ORDER                305
cancel_order              ORDER                305
change_order              ORDER                304
delivery_options          DELIVERY             302
check_refund_policy       REFUND               302
delivery_period           DELIVERY             301
contact_customer_service  CONTACT              299
create_account            ACCOUNT              298
check_cancellation_fee    CANCELLATION_FEE     298
recover_password          ACCOUNT              298
complaint                 FEEDBACK             298
delete_account            ACCOUNT              298
check_payment_methods     PAYMENT              297
change_shipping_address   SHIPPING_ADDRESS     297
contact_human_agent       CONTACT              297
registration_problems     ACCOUNT              296
newsletter_subscription   NEWSLETTER           295
get_refund                REFUND               294
edit_account              ACCOUNT              294
switch_account            ACCOUNT              290
```

Text Preprocessing: Data preparation

Code:

```
# Importing libraries for Natural Language Processing
import nltk       # NLTK library for lemmatizing and removing stopwords

import re       # for text preprocessing
# nltk.download("wordnet")      #downloads wordnet for lemmatization
from nltk.corpus import stopwords



from nltk.stem.wordnet import WordNetLemmatizer         # Lemmatizer from nltk library

lemmatizer = WordNetLemmatizer()      # Initializing lemmatizer object
corpus = []     # Empty corpus to load prompts after preprocessing
messages = x.copy()     # Loading the messages

# Loop iterates through every prompts in the messages list
for i in range(0,len(messages)):
    print(i)
    review = re.sub('[^a-zA-Z]',' ', messages[i])       # Removes everything except from capital A-Z and small a-z
    review = review.lower()         # Lowers all the character
    review = review.split()         # Split every word in the sentence

    review = [lemmatizer.lemmatize(word) for word in review if word not in stopwords.words("english")]      # Removes the stopwor
    review = ' '.join(review)       # Join all the words after preprocessing to form the sentence again
    corpus.append(review)       # Append prompts after preprocessing to the corpus list
```


This process removes stopwords from the prompts and lemmatizes each word, reducing it to its root form. This ensures consistency across different versions of the same word, preventing the model from treating similar words as distinct and assigning them different weights.

Text Preprocessing: Data preparation

This is how our prompts looks like after the preprocessing. It is evident that the unnecessary words used to form sentences, like: I, he, she, was, etc. are now removed from the corpus.

This helps avoid the model to give these words weightage in the label prediction, as we know that these words doesn't have any significant importance in predicting the intent of the query.

Also we have reducted the words to its root form to maintain consistency across the use of same word in different verb forms.

```
corpus
```

```
['problem canceling order',
 'find information canceling order',
 'need help canceling last order',
 'could help cancelling last order made',
 'problem cancelling order made',
 'help canceling last order',
 'know cancel last order made',
 'problem canceling order',
 'problem cancelling last order made',
 'could give information order cancellation',
 'need help canceling last order made',
 'need help cancelling order made',
 'want order',
 'could find information cancelling order',
 'would give information order cancellation',
 'problem cancelling order',
 'cancel order',
 'problem cancelling order made',
 'assistance cancelling order made',
 'give information canceling order',
 'problem canceling order',
 'know cancel last order made',
 'would like cancel order made',
 'need help canceling order',
 'could get information cancelling order',
```

Text Preprocessing: Data transformation

Code:

```
from keras._tf_keras.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=2000)       # Tokenizer object for vector representation

tokenizer.fit_on_texts(corpus)      # Fit the words present in the corpus to make their vector representation

one_hot_repr = tokenizer.texts_to_sequences(corpus)     # Converts words into vector form
one_hot_repr[:5]
```

Output:

```
[[65, 88, 7],
 [41, 19, 88, 7],
 [3, 1, 88, 20, 7],
 [8, 1, 90, 20, 7, 42],
 [65, 90, 7, 42]]
```

This process transforms text inputs into vector form, a crucial step in Natural Language Processing (NLP), as machines cannot interpret text in its natural language. Each number in the vector corresponds to a different word.

Text Preprocessing: Data transformation

Code:

```
sent_length = 15        # Defines maximum length

padded_doc = pad_sequences(one_hot_repr, padding='pre', maxlen=sent_length)
padded_doc[:5]
```

Output:

```
array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 65, 88,  7],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 41, 19, 88,  7],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,  1, 88, 20,  7],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  8,  1, 90, 20,  7, 42],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 65, 90,  7, 42]])
```

This process adds zeros in the start for every prompt to unsure that the input shape for every prompt is the same. This step is very necessary as the model can't be trained on irregular input shape. The input and output shape must be same for every instance.

Model Architecture: Making a Deep Learning Neural Network

Code:

```
embedding_features = 40
num_classes = 27        # Total number of different intents
voc_size = 2000
model = Sequential()        # Initializing sequential model

model.add(Embedding(voc_size,embedding_features, input_shape = (sent_length,)))     # Embedding layer to place similar words close to eac
model.add(Bidirectional(LSTM(100)))         # Bidirectional LSTM layer

model.add(Dense(num_classes, activation="softmax"))         # Using softmax activation function which gives the output as probability of

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])      # Compiles the model with the followi

model.summary()
```

This code creates a deep learning LSTM model structure which we trained using our data to predict the intents when feeded with new unseen data.

Our deep learning model consists of the follwoing three layers:

1. Input layer: The embedding layer is our input layer.
2. Hidden layer: The LSTM layer is our hidden layer.
3. Output Layer: The last Dense layer is our output layer which uses softmax activation function to predict the labels

Model Architecture: Model Summary

This is our model summary. Our model has three layers.

1. Embedding: The Embedding layer converts each token (word) in the padded input sequence into a dense vector of a fixed size (i.e., the embedding size). These vectors are learned during training, meaning the model learns a semantic representation for each word.

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 15, 40)	80,000
bidirectional (Bidirectional)	(None, 200)	112,800
dense (Dense)	(None, 27)	5,427

2. Bi-directional LSTM: In a Bidirectional LSTM, two separate LSTM layers are used. One processes the input sequence in the forward direction and the other processes the input in the backward direction.The outputs of both the forward and backward LSTM layers are then concatenated at each time step. This combination allows the model to access both past (preceding words) and future (subsequent words) information simultaneously.

3. The final Dense layer uses the 'softmax' activation function and predicts the possability for the prompt to belong in each class. The output is the probability of each class and thus the output shape is 1 row and 27 columns. (Since we have 27 different intents as label.)

Model Training: Training Data Preparation

We did the following steps to prepare our data for model training.

1. Load the padded data into a new variable, so our features is ready to fit in the model.
2. Encode the labels into an integer representation for model training.
3. Split the data into two sets for training and testing purposes.
    - We splitted data such that 80% of the data is used for training and 20% is used for testing.
    - We used the testing data for model validation data.

1.
```
import numpy as np
x = np.array(padded_doc)        # Final features after the preprocessing
```

2.
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()     # Initializes the encoder

y = le.fit_transform(y) # Encodes the labels into integers
```

3.
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=25)
```

Model Training: Fitting the data

We now trained our model using the training data and validated it with the test data and obtained an accuracy of 99.75% on the training data and 99.45% on the validation data in the last epoch.

```python
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10, batch_size=64)        # Train the model on training data and using
```

Python

```
Epoch 1/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 10s 36ms/step - accuracy: 0.2330 - loss: 3.0419 - val_accuracy: 0.8783 - val_loss: 0.6565
Epoch 2/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - accuracy: 0.9156 - loss: 0.4441 - val_accuracy: 0.9694 - val_loss: 0.2034
Epoch 3/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9699 - loss: 0.1581 - val_accuracy: 0.9780 - val_loss: 0.0978
Epoch 4/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9862 - loss: 0.0727 - val_accuracy: 0.9865 - val_loss: 0.0736
Epoch 5/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9930 - loss: 0.0398 - val_accuracy: 0.9853 - val_loss: 0.0611
Epoch 6/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.9917 - loss: 0.0371 - val_accuracy: 0.9859 - val_loss: 0.0638
Epoch 7/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.9930 - loss: 0.0271 - val_accuracy: 0.9914 - val_loss: 0.0424
Epoch 8/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.9975 - loss: 0.0138 - val_accuracy: 0.9939 - val_loss: 0.0340
Epoch 9/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 36ms/step - accuracy: 0.9985 - loss: 0.0112 - val_accuracy: 0.9902 - val_loss: 0.0442
Epoch 10/10
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 0.9975 - loss: 0.0104 - val_accuracy: 0.9945 - val_loss: 0.0334
```

Model Evaluation: Checking prediction accuracy

To check our model performance we did the following steps:

1. We used our model to predict on the training data and store its predictions to check for accuracy.
2. Converted the output of probabilities into a single class output by using the argmax function of numpy to get the class with the highest probability.
3. Used the accuracy score by scik-it learn module to check how well the model performed.

1.

```
predictions = model.predict(x_test)
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step
```

Output:

```
predictions
```

```
array([[9.1756149e-08, 1.2803212e-06, 1.1964839e-05, ..., 1.7044285e-05,
        1.6950687e-05, 3.2979392e-06],
       [4.1226161e-07, 6.3772641e-06, 3.4473521e-07, ..., 4.4370640e-06,
        7.6029130e-05, 2.1607733e-05],
       [5.2268366e-08, 9.9685600e-08, 3.4346584e-05, ..., 5.9801419e-06,
        5.2458790e-06, 2.4894762e-05],
       ...,
       [1.1964481e-07, 1.3587581e-06, 1.1754718e-05, ..., 1.3369713e-05,
        1.6623810e-05, 3.2607816e-06],
       [1.3779851e-06, 4.9049886e-06, 5.6596711e-08, ..., 4.5796103e-08,
        2.7966218e-05, 6.0457573e-04],
       [5.3038871e-08, 3.4368401e-08, 1.1195218e-05, ..., 1.9649929e-06,
        3.6348385e-06, 1.5961779e-05]], dtype=float32)
```

Model Evaluation: Checking prediction accuracy

2.

```
predicted_class = np.argmax(predictions, axis=1)
```

3.

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test,predicted_class)      # Shows the model's prediction accuracy on testing data
```

```
0.9944954128440368
```

We obtained a great accuracy of **99.45%** and now our model is ready to give predictions on the real world problems.

Model Serialization: Saving the models

After training the models, for prediction, tokenization and encoding labels, we saved the model so that it can be used across different instances, systems, or sessions.

Code snippets:

```
# Saving the tokenizer to have same vector representation for words accross different instances

with open('tokenizer.pkl', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
# Loading models into pickle files

with open('label_encoder.pkl', 'wb') as file:
    pickle.dump(le, file)

with open('model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

Chatbot Architecture: Loading the pretrained models

To create a working chatbot, we created a separate python file and used the streamlit library to make the user interface. The first step was to load our model, tokenizer & encoder. Loading the pretrained tokenizer is as important as loading the model because we want to ensure that our new inputs gets encoded in the same way as our training inputs.

Code:

```
## deep learning model
with open("./pickles/model.pkl","rb") as file:
    model = pickle.load(file)

## tokenizer for one hot encoding
with open('./pickles/tokenizer.pkl', 'rb') as handle:
    tokenizer = pickle.load(handle)

## label encoder object for class classification
with open("./pickles/label_encoder.pkl","rb") as file:
    encoder = pickle.load(file)
```

Chatbot Architecture: Generating response set

Generated a response for every intent so that the chatbot can respond to the user when a query of that particular intent arises.

Responses:

```
# Generating a set of responses for every intent
responses = {'cancel_order':["To cancel your order, go to your profile, select 'Your Orders', then choose the product you w
             'change_order': ["To remove or add items in your order, you can use the update order option. Go to your profile
             'change_shipping_address': ["To change shipping address, go to your profile, select 'Saved Address', then click
             'check_cancellation_fee': ["You will not be charged any cancellation fee for the orders that are cancelled befo
             'check_invoice': ["You can download the invoice from your purchase in the 'Your Orders' section. Go to your pro
             'check_payment_methods': ["Currently we accept payments through Credit card, Debit card, Net banking and UPI. F
             'check_refund_policy': ["To check our policy regarding refunds, you can read this blogpost on www.somearticle.c
             'complaint': ["We're truly sorry to hear that you have concerns about our company. We would appreciate it if yo
             'contact_customer_service': ["To talk to our customer service agent, please write to us on support@example.com.
             'contact_human_agent': ["To talk to our customer service agent, please write to us on support@example.com. Our
             'create_account': ["To create an account, click on the signup button and follow the instructions given on the i
             'delete_account': ["To delete your account, click on the settings button, then go to Account & Security and cli
             'delivery_options': ["Rest assured, we will deliver the product right at your house (or any provided location c
             'delivery_period': ["To check your shipment, go to your profile, select 'Your Orders', then choose the product
             'edit_account': ["To edit your account information, click on the settings button, then go to Account & Security
             'get_invoice': ["You can download any invoice from your purchase in the 'Your Orders' section. Go to your profi
             'get_refund': ["To get a refund on any of your purchases you can raise a ticket in the 'Refund' section in the
             'newsletter_subscription': ["To subscribe to our newsletter, click on the link here: randomnewsletterlink.com"]
             'payment_issue': ["For any payment related issue, write to us on payment@example.com and our payments team will
             'place_order': ["You can place order by adding your desired item in the card. Then you have to click on the 'Ca
             'recover_password': ["To reset your password, click on forgot passowrd button in the log in screen. Then enter
             'registration_problems': ["If you are facing an error in the sign up process, or not receiving an OTP try resta
             'review': ["To give us some feedbacks you can rate us on the app store and write your feedback. Our team values
             'set_up_shipping_address': ["To add a shipping address, go to your profile, select 'Saved Address', then click
             'switch_account': ["To switch account, you can go to settings and click 'Log Out'. After logging out, you can t
             'track_order': ["To track your shipment, go to your profile, select 'Your Orders', then choose the product you
             'track_refund': ["To track status of your refund, go to the 'Refund' section in the settings. Then click on 'R
```

Chatbot Architecture: Function for chatbot responses

The function performs the following steps:

1. Takes the user input and store it.
2. Performs text preprocessing like removing stopwords, lemmatization.
3. Generate a vector representation of words using the pretrained tokenizer object
4. Do zero padding of the vector representation
5. Predict intent class using the pretrained model
6. Returns the response for the particular predicted intent class

```
def chatbot_response(user_input):
    # creating a list object for the user input prompt
    text = [user_input]

    # text preprossesing

    ## using lemmatizer by nltk library to mintain uniformity among different word forms for same words
    lemmatizer = WordNetLemmatizer()

    ## creating an empty corpus to contain the text after preprocessing
    corpus = []


    review = text[0].lower()  # converts the prompt into lower case
    review = review.split()  # split the sentence into word tokens

    review = [lemmatizer.lemmatize(word) for word in review if word not in stopwords.words("english")] # lemmatizes all the words in the prompt and removes

    review = ' '.join(review) #joins all the different words in the sentence with a " " (space) between them
    corpus.append(review) # appends the sentence in the empty corpus

    # one hot encoding and padding
    one_hot_repr = tokenizer.texts_to_sequences(text)  # encodes the input prompt according to the same formatting as used in the model training

    # pre padding
    sent_length = 15    # max length of the array to ensure regularity
    padded_doc = pad_sequences(one_hot_repr, padding='pre', maxlen=sent_length)     # adds zeros in the start of maintain fixed input dimensions

    # model prediction
    predictions = model.predict(padded_doc)     # using already trained LSTM model for predictions

    predicted_class = np.argmax(predictions, axis=1)    # returns the class with the highest probability
    a = predicted_class[0]      # assigns the encoded class to a variable
    intent = labels[a]      # uses the label dictionary to get exact class (i.e class before the encoding)
    bot = responses[intent]       # store the response according to the intent

    return bot[0]       # returns the bot's response
```

Chatbot Architecture: User Interface

The following piece of code performs these actions:

1. Creates a title for the homepage. ("Ask Panda Bot")
2. If there are no messages in the session yet, the bot response will be to greet.
3. When there are new messages in the session (either by bot or the user) it is stored and displayed on the chatbox.
4. Whenever a prompt is entered in the textbox, the bot response function is called with the prompt as the input for the function.
5. The response returned by the function is displayed as the bot's response in the chatbox.

```
# importing streamlit to make user interface
import streamlit as st

# Create a Streamlit interface
st.title("Ask Panda Bot")
st.write("Type your queries and the AI will assist you.")

# using conditions to generate a welcome message
if "messages" not in st.session_state:
    st.session_state.messages = [{"role":"assistant","content":"Hey! I am Panda bot. How may I help you?"}]


# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])



# Accept user input
if prompt := st.chat_input("Ask your Question"):
    # Display user message in chat message container
    with st.chat_message("user"):
        st.markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})

    # Generate the chat response by calling chatbot_response function using user prompt as input
    bot_response = chatbot_response(prompt)
    # Display chatbot message in chat message container
    with st.chat_message("assistant"):
        st.markdown(bot_response)
    # Add chatbot message to chat history
    st.session_state.messages.append({"role": "assistant", "content": bot_response})
```

Conclusive Summary

The implementation of our chatbot demonstrates a significant advancement in providing timely, efficient, and scalable customer support. By leveraging natural language processing and rule-based interactions, the chatbot is able to handle routine inquiries, troubleshoot common issues, and provide 24/7 assistance, ensuring an enhanced customer experience.

Key benefits of the chatbot include:

- *Improved customer satisfaction: Instant responses to queries reduce wait times and enhance service quality.*
- *Cost-effective support: Automating routine tasks allows human agents to focus on more complex issues, reducing operational costs.*
- *Scalability: The chatbot can handle a large volume of queries without compromising performance, making it a flexible solution as the company grows.*
- *Future enhancements may include integrating AI-driven learning for more personalized interactions, expanding multi-language support, and seamless handover to human agents for complex issues. Overall, this chatbot positions for success in maintaining strong customer relationships in an increasingly digital marketplace.*

Thank You

Presented By: Sanket Anand

Check out the model here