Price Negotiating Chatbot for E-commerce Website

Abstract

In today’s competitive e-commerce landscape, providing personalized shopping experiences is paramount for retaining customers and driving sales. This project introduces a price negotiating chatbot for e-commerce platforms, leveraging machine learning techniques to determine optimal prices through real-time negotiation. The chatbot facilitates user engagement by allowing customers to negotiate prices seamlessly via text or voice interactions. Key functionalities include user authentication, product browsing with detailed information display, order placement, review posting with sentiment analysis, and order viewing. The system architecture follows a client-server model, with Flask serving as the backend framework and MySQL as the database. Future enhancements include implementing advanced machine learning models for price prediction, enhancing voice interaction capabilities, incorporating dynamic pricing strategies, and adding multi-language support. The project demonstrates the potential of chatbots to enhance user engagement and satisfaction in the e-commerce domain while paving the way for future innovations in personalized shopping experiences.

Introduction

In today's highly competitive e-commerce landscape, the importance of delivering an exceptional customer experience cannot be overstated. With the rapid growth of online shopping, customers have become increasingly savvy, seeking not only the best deals but also personalized and engaging shopping experiences. Traditional static pricing models often fail to meet these dynamic expectations, leading to missed opportunities for both customer satisfaction and sales growth. This is where innovative solutions, such as a price negotiating chatbot, come into play.

A price negotiating chatbot represents a significant leap forward in the realm of e-commerce. Unlike conventional chatbots that offer standard customer support, this chatbot is designed to engage users in real-time price negotiations, providing a more interactive and tailored shopping experience. By leveraging machine learning techniques, the chatbot can assess various factors—such as customer purchase history, browsing behavior, and market conditions—to offer personalized pricing that aligns with both customer preferences and business goals.

The concept of negotiation in e-commerce is not new; however, its implementation has largely been limited to traditional in-person settings or through manual online interactions. Automating this process with a chatbot not only streamlines negotiations but also makes them accessible to a broader audience, enhancing convenience and user engagement. Customers can now interact with the e-commerce platform in a conversational manner, negotiating prices just as they would with a human sales representative.

The primary goal of this project is to design and implement a sophisticated price negotiating chatbot that can seamlessly integrate with existing e-commerce platforms. This chatbot will use advanced machine learning algorithms to determine the best possible price a customer can receive through negotiation, balancing the need for competitive pricing with the business’s profit margins. The system aims to enhance user engagement, boost conversion rates, and foster customer loyalty by providing a unique, interactive shopping experience.

Importance of the Project

Incorporating a price negotiating chatbot into an e-commerce platform offers numerous benefits. For customers, it provides a sense of empowerment and satisfaction as they actively participate in setting the price they pay. This interactive process can lead to increased customer retention and repeat purchases, as customers are more likely to return to a platform where they feel they receive personalized attention and better deals.

From a business perspective, the data gathered from these interactions can be invaluable. Insights into customer behavior, preferences, and negotiation patterns can inform broader pricing strategies and marketing efforts. Additionally, the ability to dynamically adjust prices based on real-time data and market trends ensures that the business remains competitive and responsive to changes in the marketplace.

Technological Advancements

The development of such a chatbot is made possible by advancements in several key areas of technology. Natural Language Processing (NLP) enables the chatbot to understand and interpret customer inputs effectively, facilitating smooth and natural conversations. Machine learning algorithms, particularly in the fields of predictive analytics and dynamic pricing, allow the system to make informed decisions about optimal pricing strategies. Integration with e-commerce platforms and payment gateways ensures that negotiated prices are seamlessly applied during the checkout process.

Potential Challenges

While the benefits are clear, the implementation of a price negotiating chatbot also presents several challenges. Ensuring the chatbot's responses are both realistic and satisfactory requires sophisticated NLP and machine learning models. Balancing customer satisfaction with profitability necessitates careful calibration of the negotiation algorithms. Moreover, seamless integration with existing e-commerce infrastructure is crucial to avoid any disruptions in the user experience.

Literature Survey

The integration of chatbots into e-commerce has been a subject of extensive research, focusing on various functionalities such as customer service, product recommendations, and personalized marketing. While the use of chatbots for price negotiation is a relatively novel approach, the existing literature provides valuable insights into the potential benefits and challenges of this technology. This survey reviews key studies and projects in three main areas: chatbot technology in e-commerce, machine learning for price prediction, and sentiment analysis in e-commerce.

Chatbot Technology in E-commerce

1. Lu, W., Zhang, P., & Lin, Z. (2018) explored the impact of chatbots on customer service in e-commerce. Their study found that chatbots significantly enhance customer satisfaction by providing immediate and personalized responses, leading to increased customer retention and loyalty.
2. Arenas-Gaitán, J., Ramírez-Correa, P. E., & Rondán-Cataluña, F. J. (2019) investigated how chatbots influence sales conversion rates. They discovered that customers are more likely to complete purchases when assisted by chatbots, which offer timely information and resolve queries efficiently.

3. Shawar, B. A., & Atwell, E. (2007) provided a comprehensive overview of the development and application of conversational agents in e-commerce. They highlighted the technological advancements that have enabled chatbots to perform complex tasks, such as handling multiple customer inquiries simultaneously and providing tailored product recommendations.

Machine Learning for Price Prediction

4. Li, H., Xie, K., & Wang, Q. (2019) examined the use of Support Vector Machines (SVM) for price prediction in the real estate market. Their research demonstrated the high accuracy and reliability of SVM models in predicting property prices based on historical data and market trends.
5. Zhang, X., & Pan, G. (2017) applied K-Nearest Neighbors (KNN) algorithms to forecast commodity prices. Their findings indicated that KNN models are effective in capturing the non-linear relationships between variables, resulting in precise price predictions.
6. Sun, Y., & Luo, Y. (2019) explored the application of neural networks in dynamic pricing strategies for e-commerce platforms. They showed that neural networks could adjust prices in real-time based on demand fluctuations and competitor pricing, optimizing revenue and competitiveness.

Sentiment Analysis in E-commerce

7. Hutto, C. J., & Gilbert, E. (2014) developed VADER (Valence Aware Dictionary and sEntiment Reasoner), a sentiment analysis tool that has been widely used to analyze customer reviews and feedback in e-commerce. Their work demonstrated VADER's ability to accurately gauge customer sentiment, providing valuable insights that can influence pricing and marketing strategies.
8. Pang, B., & Lee, L. (2008) reviewed various sentiment analysis methodologies, emphasizing their importance in understanding consumer attitudes and behaviors. Their research highlighted how sentiment analysis can inform product development, customer service improvements, and pricing decisions.

Implications for Price Negotiation Chatbots

While these studies provide a solid foundation for the development of price negotiation chatbots, they also underscore the complexities involved. Integrating machine learning

Requirements Specifications

The development of a price negotiating chatbot for an e-commerce platform requires detailed functional and non-functional requirements to ensure a comprehensive, secure, and user-friendly experience. The specifications are divided into functional requirements, which outline the essential features and operations of the system, and non-functional requirements, which define the quality attributes and system constraints.

Functional Requirements

1. User Authentication:
    - Signup: New users must be able to create an account by providing necessary details such as name, email, and password.
    - Login: Existing users must be able to log in using their email and password.
    - Password Recovery: Users should have an option to recover their password via email verification.
    - Profile Management: Users should be able to update their personal information and password.
2. Product Browsing:
    - Product Display: The system must display a list of products with details including price, description, and image.
    - Search Functionality: Users should be able to search for products by name, category, or keywords.
    - Filter and Sort: Users should be able to filter products by categories, price range, and other attributes, as well as sort them by price, popularity, and rating.
3. Price Negotiation:
    - Negotiation Interface: Users must be able to negotiate prices through a user-friendly interface that supports text and voice interactions.
    - AI-Driven Negotiation: The chatbot must use machine learning algorithms to determine counteroffers and negotiate with the user in real-time.
    - Negotiation History: Users should be able to view their negotiation history with each product.
4. Order Placement:
    - Add to Cart: Users must be able to add products to a shopping cart.
    - Checkout Process: The system must guide users through a secure and straightforward checkout process, including payment gateway integration.
    - Order Confirmation: Users should receive an order confirmation with details via email and on the platform.
5. Review Posting:

- Review Submission: Users should be able to post reviews for products they have purchased.
- Sentiment Analysis: The system must analyze reviews using sentiment analysis tools like VADER and provide feedback to users about their review sentiment.
- Review Display: Reviews should be displayed on product pages, with sentiment scores highlighted.

6. Order Viewing:
    - Order History: Users must be able to view their past orders, including order details, status, and history.
    - Reorder Option: Users should have an option to reorder previously purchased items directly from their order history.

Non-Functional Requirements

1. Security:
    - Data Protection: Ensure the secure storage and transmission of user data through encryption and secure protocols (e.g., HTTPS).
    - Authentication and Authorization: Implement robust authentication mechanisms and role-based access control to protect user accounts and data.
    - Fraud Detection: Incorporate systems to detect and prevent fraudulent activities and transactions.
2. Usability:
    - User Interface Design: The platform must have an intuitive and user-friendly interface, with clear navigation and consistent design.
    - Accessibility: Ensure the platform is accessible to users with disabilities, complying with standards such as WCAG (Web Content Accessibility Guidelines).
    - Customer Support: Provide accessible customer support options, including FAQs, live chat, and contact forms.
3. Performance:
    - Response Time: The system must handle user requests efficiently, with minimal latency, especially during negotiations and transactions.
    - Load Handling: Ensure the platform can handle multiple concurrent users without performance degradation.

    - Optimization: Regularly optimize the backend and frontend components for faster load times and smoother user experience.
4. Scalability:
    - Horizontal and Vertical Scaling: The system architecture should support both horizontal (adding more servers) and vertical (enhancing server capabilities) scaling to accommodate growing numbers of users and products.
    - Database Scalability: Implement scalable database solutions to manage increasing data volumes, ensuring quick data retrieval and storage.
    - Cloud Integration: Utilize cloud services for scalable infrastructure, enabling dynamic resource allocation based on demand.

Hardware and Software Specifications

To successfully develop and deploy the price negotiating chatbot for an e-commerce platform, specific hardware and software requirements must be met. These specifications ensure that the system operates efficiently, securely, and reliably, providing an optimal user experience.

Hardware Specifications

Server-Side Hardware

1. Web Servers:
    - Processor: Multi-core processors (e.g., Intel Xeon or AMD EPYC) with a minimum of 8 cores.
    - Memory: At least 32 GB of RAM, scalable based on load.
    - Storage: SSD storage with a minimum capacity of 1 TB, with RAID configurations for redundancy.
    - Network: High-speed network interfaces (1 Gbps or higher) to handle large volumes of traffic.
    - Backup: Automated backup systems with off-site storage capabilities.
2. Database Servers:
    - Processor: High-performance processors optimized for database operations.
    - Memory: Minimum 64 GB of RAM, with options to scale up as data grows.

- Storage: SSD storage with high IOPS performance, configured for high availability and redundancy.
- Network: High-speed network interfaces for fast data access and replication.
- Backup and Recovery: Robust backup solutions and disaster recovery plans.

3. AI/ML Servers:
    - Processor: GPUs (e.g., NVIDIA Tesla/Quadro) or TPUs for machine learning model training and inference.
    - Memory: At least 64 GB of RAM, with scalability options.
    - Storage: High-speed SSD storage for handling large datasets and model files.
    - Network: High-speed network interfaces to support data transfer and model deployment.

Client-Side Hardware

- User Devices: Compatible with modern web browsers and mobile devices (e.g., desktops, laptops, tablets, smartphones).
- Network: Reliable internet connection with sufficient bandwidth for smooth interaction with the platform.

Software Specifications

Server-Side Software

1. Operating Systems:
    - Linux-based OS (e.g., Ubuntu Server, CentOS) for web, database, and AI/ML servers.
2. Web Server Software:
    - Web Servers: Apache HTTP Server or Nginx for serving web content.
    - Application Servers: Node.js, Django, or Flask for backend application logic.
    - Load Balancers: HAProxy or Nginx for distributing incoming traffic across multiple servers.
3. Database Management Systems:
    - Relational Databases: MySQL, PostgreSQL for structured data storage.

    - NoSQL Databases: MongoDB, Cassandra for handling unstructured data.
4. AI/ML Frameworks:
    - Machine Learning Libraries: TensorFlow, PyTorch, Scikit-learn for developing and training machine learning models.
    - NLP Tools: NLTK, SpaCy for natural language processing tasks.
    - Sentiment Analysis Tools: VADER, TextBlob for analyzing customer reviews.
5. Security Software:
    - SSL/TLS: OpenSSL for secure communications.
    - Firewalls: UFW, iptables for network security.
    - Intrusion Detection Systems: Snort, OSSEC for monitoring and protecting against threats.

Client-Side Software

1. Web Technologies:
    - Frontend Frameworks: React, Angular, or Vue.js for building dynamic and responsive user interfaces.
    - CSS Frameworks: Bootstrap, Tailwind CSS for styling and layout.
    - JavaScript Libraries: jQuery, D3.js for enhanced interactivity and data visualization.
2. Mobile Technologies:
    - Mobile Frameworks: React Native, Flutter for developing cross-platform mobile applications.

Development and Deployment Tools

1. Version Control:
    - Repository Management: Git, GitHub, GitLab for version control and collaboration.
2. Continuous Integration/Continuous Deployment (CI/CD):
    - CI/CD Tools: Jenkins, Travis CI for automated testing and deployment.
3. Containerization:
    - Container Platforms: Docker for containerizing applications.
    - Container Orchestration: Kubernetes for managing containerized applications at scale.
4. Monitoring and Logging:

- Monitoring Tools: Prometheus, Grafana for system and application monitoring.
- Logging Tools: ELK Stack (Elasticsearch, Logstash, Kibana) for centralized logging and analysis.

By meeting these requirements, the price negotiating chatbot for e-commerce platforms will provide a secure, efficient, and engaging shopping experience, catering to the evolving needs of modern consumers and ensuring business growth and customer satisfaction.

System Design

The system for the price negotiating chatbot in an e-commerce platform is designed to provide a seamless and interactive user experience. It incorporates a client-server architecture with robust backend processing, efficient data management, and advanced machine learning capabilities.

Architecture

Client-Server Architecture:

- Frontend: The frontend is built using HTML, CSS, and JavaScript, providing a dynamic and responsive user interface. It includes features for product browsing, price negotiation, order placement, review posting, and order viewing. The frontend communicates with the backend through API calls.
- Backend: The backend is implemented using Flask, a lightweight and flexible Python web framework. Flask handles user authentication, product data management, negotiation logic, order processing, and review sentiment analysis.

Machine Learning Integration:

- Price Prediction Model: The chatbot employs machine learning models to predict optimal prices during negotiations. These models are trained on historical sales data, customer behavior, and market trends.
- Sentiment Analysis: For user reviews, the system uses sentiment analysis tools such as VADER to analyze the sentiment of the reviews, providing insights that can influence pricing strategies and customer feedback.

System Components:

1. User Interface (UI): Built with HTML, CSS, and JavaScript frameworks (e.g., React or Vue.js) for an interactive user experience.
2. Web Server: Flask serves as the web server, managing HTTP requests and routing them to appropriate backend services.
3. Application Logic: Implemented in Flask, including user authentication, product management, negotiation algorithms, and order processing.
4. Database: MySQL is used for structured data storage, ensuring reliable and efficient data management.
5. Machine Learning Services: Separate services for price prediction and sentiment analysis, using frameworks like TensorFlow or PyTorch.
6. APIs: RESTful APIs for communication between the frontend and backend, ensuring modularity and scalability.

Database Design

The MySQL database schema consists of several interconnected tables to manage user data, products, reviews, and purchase orders effectively.

Tables:

1. Users:
   - `UserID` (Primary Key): Unique identifier for each user.
   - `Username`: User's chosen username.
   - `PasswordHash`: Encrypted password for secure authentication.
   - `Email`: User's email address.
   - `ProfileInfo`: Additional profile information (e.g., name, address).
2. Products:
   - `ProductID` (Primary Key): Unique identifier for each product.
   - `ProductName`: Name of the product.
   - `Description`: Detailed description of the product.
   - `Price`: Base price of the product.
   - `ImageURL`: URL to the product image.
   - `Stock`: Quantity available in inventory.
3. Reviews:
   - `ReviewID` (Primary Key): Unique identifier for each review.
   - `UserID` (Foreign Key): ID of the user who posted the review.

    - `ProductID` (Foreign Key): ID of the reviewed product.
    - `ReviewText`: Content of the review.
    - `SentimentScore`: Sentiment analysis score of the review (e.g., positive, neutral, negative).
4. PurchaseOrder:
    - `OrderID` (Primary Key): Unique identifier for each order.
    - `UserID` (Foreign Key): ID of the user who placed the order.
    - `ProductID` (Foreign Key): ID of the purchased product.
    - `NegotiatedPrice`: Final price agreed upon after negotiation.
    - `OrderDate`: Date and time when the order was placed.
    - `OrderStatus`: Current status of the order (e.g., pending, shipped, delivered).

Data Flow

The data flow within the system follows a structured sequence to ensure efficient handling of user requests and seamless interaction between components.

1. User Interaction:
    - Users interact with the frontend through their web browsers or mobile devices.
    - User actions, such as browsing products, initiating price negotiations, and placing orders, generate requests sent to the server.
2. Request Handling:
    - The frontend sends HTTP requests to the Flask server, which processes these requests through appropriate routes.
    - For instance, a login request is routed to the authentication service, while a product search request is handled by the product management service.
3. Backend Processing:
    - The Flask server interacts with the MySQL database to retrieve or update data as needed. For example, it fetches product details for display or updates user profiles.
    - When a price negotiation is initiated, the server uses the machine learning service to generate a counteroffer based on the negotiation model.
4. Machine Learning Integration:
    - For price negotiations, the server sends relevant data (e.g., product details, user profile, historical prices) to the machine learning model, which predicts an optimal price.

  - For review posting, the sentiment analysis tool processes the review text and returns a sentiment score, which is stored in the database along with the review.
5. Response Generation:
  - After processing the request, the Flask server generates a response containing the required data or confirmation.
  - This response is sent back to the frontend, which updates the user interface accordingly. For example, it displays the negotiated price or confirms the order placement.
6. User Interface Update:
  - The frontend dynamically updates based on the server responses, providing real-time feedback to the user.
  - Users can see updated product listings, negotiation outcomes, order confirmations, and review sentiments without reloading the page.

By following this architecture and design, the system ensures a smooth and interactive user experience while maintaining robust backend processing, secure data management, and advanced machine learning capabilities. This comprehensive approach enables the price negotiating chatbot to meet the dynamic needs of modern e-commerce platforms.

Implementation

To comprehensively implement and understand the price negotiating chatbot for an e-commerce platform, we'll expand and split the given Flask application code into different sections. Each section will be explained in detail, covering its purpose and functionality within the overall system.

```
from flask import Flask, render_template, request, redirect, url_for,
session, make_response

import pymysql

import datetime

import pandas as pd
import numpy as np

from sklearn.svm import SVR

from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsRegressor

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

import os

import subprocess

import speech_recognition as sr



app = Flask(__name__)



app.secret_key = 'welcome'

global uname

global original_price, predicted_price, final_price, product_name,
product_id



sid = SentimentIntensityAnalyzer()

recognizer = sr.Recognizer()



@app.route('/ViewReview', methods=['GET', 'POST'])

def ViewReview():
    if request.method == 'GET':

        global uname

        font = '<font size="3" color="black">'

        output = '<table border="1" width="100%">'

        output += '<tr><th><font size="3"
color="black">Username</font></th>'

        output += '<th><font size="3" color="black">Review</font></th>'

        output += '<th><font size="3"
color="black">Sentiment</font></th></tr>'

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root',
password = '9391091987@India', database = 'negotiate',charset='utf8')

        index = 0

        with con:

            cur = con.cursor()

            cur.execute("select * FROM reviews")

            rows = cur.fetchall()

            for row in rows:

                output += "<tr><td>"+font+str(row[0])+"</td>"

                output += "<td>"+font+str(row[1])+"</td>"

                output += "<td>"+font+str(row[2])+"</td>"

        return render_template('ViewReview.html', msg=output)
```

```
@app.route('/ViewOrders', methods=['GET', 'POST'])

def ViewOrders():

    if request.method == 'GET':

        global uname

        font = '<font size="3" color="black">'

        output = '<table border="1" width="100%">'

        output += '<tr><th><font size="3" color="black">Purchaser
Name</font></th>'

        output += '<th><font size="3" color="black">Product
ID</font></th>'

        output += '<th><font size="3" color="black">Product
Name</font></th>'

        output += '<th><font size="3" color="black">Amount</font></th>'

        output += '<th><font size="3" color="black">Purchase
Date</font></th></tr>'

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root',
password = '9391091987@India', database = 'negotiate',charset='utf8')

        index = 0

        with con:

            cur = con.cursor()

            cur.execute("select * FROM purchaseorder where
username='"+uname+"'")

            rows = cur.fetchall()

            for row in rows:
                output += "<tr><td>"+font+str(row[0])+"</td>"

                output += "<td>"+font+str(row[1])+"</td>"

                output += "<td>"+font+str(row[2])+"</td>"

                output += "<td>"+font+str(row[3])+"</td>"

                output += "<td>"+font+str(row[4])+"</td>"

        return render_template('ViewOrders.html', msg=output)



@app.route('/CompleteOrder', methods=['GET', 'POST'])

def CompleteOrder():

    global uname

    global original_price, predicted_price, final_price, product_name,
product_id

    if request.method == 'POST':

        if predicted_price != 0:

            now = datetime.datetime.now()

            current_time = now.strftime("%Y-%m-%d %H:%M:%S")

            status = "Error in cinfirming order"

            db_connection = pymysql.connect(host='127.0.0.1',port =
3306,user = 'root', password = '9391091987@India', database =
'negotiate',charset='utf8')

            db_cursor = db_connection.cursor()

            student_sql_query = "INSERT INTO
purchaseorder(username,product_id,product_name,amount,transaction_date)
```

```
VALUES('"+uname+"','"+product_id+"','"+product_name+"','"+str(predicted_pr
ice)+"','"+str(current_time)+"')"

            db_cursor.execute(student_sql_query)

            db_connection.commit()

            if db_cursor.rowcount == 1:

                status = 'Your Order completed'

        else:

            status = "First negotiate price from chatbot then confirm
order"

        return render_template('UserScreen.html', msg=status)




@app.route('/PostReviewAction', methods=['GET', 'POST'])

def PostReviewAction():

    if request.method == 'POST':

        global uname

        review = request.form['t1']

        sentiment_dict = sid.polarity_scores(review)

        compound = sentiment_dict['compound']

        result = ''

        if compound >= 0.05 :

            result = 'Positive'
        elif compound <= - 0.05 :

            result = 'Negative'

        else :

            result = 'Neutral'

        db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user
= 'root', password = '9391091987@India', database =
'negotiate',charset='utf8')

        db_cursor = db_connection.cursor()

        student_sql_query = "INSERT INTO
reviews(username,review,sentiment)
VALUES('"+uname+"','"+review+"','"+result+"')"

        db_cursor.execute(student_sql_query)

        db_connection.commit()

        status = "Error in taking review"

        if db_cursor.rowcount == 1:

            status = 'Your review accepted & sentiment predicted :
'+result

        return render_template('PostReview.html', msg=status)






@app.route('/PostReview', methods=['GET', 'POST'])

def PostReview():
    return render_template('PostReview.html', msg='')



@app.route('/UserScreen', methods=['GET', 'POST'])

def UserScreen():

    global uname

    return render_template('UserScreen.html', msg="Welcome "+uname)




@app.route('/index', methods=['GET', 'POST'])

def index():

    return render_template('index.html', msg='')




@app.route('/Login', methods=['GET', 'POST'])

def Login():

   return render_template('Login.html', msg='')



@app.route('/Signup', methods=['GET', 'POST'])

def Signup():

    return render_template('Signup.html', msg='')
```

```
@app.route('/ChatData', methods=['GET', 'POST'])

def ChatData():

    if request.method == 'GET':

        global predicted_price

        query = request.args.get('mytext')

        query = query.strip("\n").strip()

        output = "Sorry! i am not trained for given question"

        if 'price' in query.lower():

            output = "You can get product at $:"+str(predicted_price)

        if "final" in query.lower() or "discount" in query.lower() or "my"
in query.lower():

            discount = (predicted_price / 100) * 5

            predicted_price = predicted_price - discount

            output = "The final price you can get this product is
$:"+str(predicted_price)

        response = make_response(output, 200)

        response.mimetype = "text/plain"

        return response



@app.route('/record', methods=['GET', 'POST'])

def record():
    if request.method == 'POST':

        global predicted_price

        data = request.files['data'].read()

        if os.path.exists('static/audio/audio.wav'):

            os.remove('static/audio/audio.wav')

        if os.path.exists('static/audio/audio1.wav'):

            os.remove('static/audio/audio1.wav')

        with open("static/audio/audio.wav", "wb") as fh:

            fh.write(data)

        fh.close()

        path = os.path.abspath(os.getcwd())+'/static/audio/'

        print("===================="+path)

        res = subprocess.check_output(path+'ffmpeg.exe -i
'+path+'audio.wav '+path+'audio1.wav', shell=True)

        with sr.WavFile(path+'audio1.wav') as source:

            audio = recognizer.record(source)

        try:

            text = recognizer.recognize_google(audio, language="en-IN")

        except Exception as ex:

            text = "unable to recognize"

        print(text)
        query = text.strip("\n").strip()

        output = "Sorry! i am not trained for given question"

        if 'price' in query.lower():

            output = "You can get product at $:"+str(predicted_price)

        if "final" in query.lower() or "discount" in query.lower() or "my"
in query.lower():

            discount = (predicted_price / 100) * 5

            predicted_price = predicted_price - discount

            output = "The final price you can get this product is
$:"+str(predicted_price)

        response = make_response("Your Query : "+query+"\nChatbot:
"+output, 200)

        response.mimetype = "text/plain"

        return response



@app.route('/Chatbot', methods=['GET', 'POST'])

def Chatbot():

    if request.method == 'GET':

        global original_price, predicted_price, final_price, product_name,
product_id

        product_id = request.args.get('t1') #user will select product for
which he want negotiate

        types = request.args.get('t2')
        dataset = pd.read_csv("Dataset/model.csv") #read dataset

        dataset.fillna(0, inplace = True) #replace missing values in
dataset with 0

        products = dataset.loc[dataset['index'] == product_id] #read all
rows from dataset which is matches with user selected product

        products = products.values #convert dataframe to array

        print(products)

        original_price = products[0,5] #get original price from dataset

        product_name = products[0,2] #get product name from dataset

        X = products[:,5:6] #get original prices as X training data

        Y = products[:,6:7] #get negotiating prices as Y data

        sc = MinMaxScaler(feature_range = (0, 1)) #can be used to
normalize dataset

        X = sc.fit_transform(X) #normalize the X values

        Y = sc.fit_transform(Y) #normalize the Y values

        svr_regression = SVR(C=1.0, epsilon=0.2) #create SVM object

        #training SVR with X and Y data

        svr_regression.fit(X, Y.ravel()) #trained SVM with X and Y data

        #performing prediction on test data

        predict = svr_regression.predict(X) #perform prediction to get
best price

        predict = predict.reshape(predict.shape[0],1)

        predict = sc.inverse_transform(predict)
        predict = predict.ravel()

        labels = sc.inverse_transform(Y)

        labels = labels.ravel()



        knn = KNeighborsRegressor(n_neighbors=2) #here we are training
with KNN

        #training KNN with X and Y data

        knn.fit(X, Y.ravel())

        #performing prediction on test data

        predict = knn.predict(X)

        predict = predict.reshape(predict.shape[0],1)

        predict = sc.inverse_transform(predict)

        predict = predict.ravel()

        labels = sc.inverse_transform(Y) #back to original values from
normalization

        labels = labels.ravel()

        predicted_price = predict[0] #get best predicted price

        output = "Hi! this is Nego.<br/>Your selected Product :
"+product_name+".<br/>Its Current Price : "+str(original_price)+".<br/>"

        page = 'Chatbot.html'

        if types == 'voice':

            page = 'VoiceBot.html'
        return render_template(page, msg=output)







@app.route('/BrowseProducts', methods=['GET', 'POST'])

def BrowseProducts():

    if request.method == 'GET':

        font = '<font size="3" color="black">'

        output = '<table border="1" width="100%">'

        output += '<tr><th><font size="3" color="black">Product
Type</font></th>'

        output += '<th><font size="3" color="black">Product
Name</font></th>'

        output += '<th><font size="3"
color="black">Description</font></th>'

        output += '<th><font size="3" color="black">Product
Image</font></th>'

        output += '<th><font size="3" color="black">Price</font></th>'

        output += '<th><font size="3" color="black"><font size="3"
color="black">Text Negotiate with Chatbot</font></th>'

        output += '<th><font size="3" color="black"><font size="3"
color="black">Voice Negotiate with Chatbot</font></th></tr>'

        dataset = pd.read_csv("Dataset/ecommerce.csv")

        dataset.fillna(0, inplace = True)
        dataset = dataset.values

        for i in range(len(dataset)):

            index = str(dataset[i,0])

            types = str(dataset[i,1])

            name = str(dataset[i,2])

            desc = str(dataset[i,3])

            price = str(dataset[i,5])

            output+="<tr><td>"+font+types+"</font></td>"

            output+="<td>"+font+name+"</font></td>"

            output+="<td>"+font+desc+"</font></td>"

            output+='<td><img src="static/img/'+index+'.png" width="150"
height="150"></img></td>'

            output+="<td>"+font+price+"</font></td>"

            output+='<td><a href="Chatbot?t1='+index+'&t2=text">Text Based
Chatbot to Negotiate</a></td>'

            output+='<td><a href="Chatbot?t1='+index+'&t2=voice">Voice
Based Chatbot to Negotiate</a></td></tr>'

        return render_template('BrowseProducts.html', msg=output)




@app.route('/LoginAction', methods=['GET', 'POST'])

def LoginAction():
    global uname

    if request.method == 'POST':

        user = request.form['t1']

        password = request.form['t2']

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root',
password = '9391091987@India', database = 'negotiate',charset='utf8')

        index = 0

        with con:

            cur = con.cursor()

            cur.execute("select * FROM users")

            rows = cur.fetchall()

            for row in rows:

                if row[0] == user and password == row[1]:

                    uname = user

                    index = 1

                    break

        if index == 0:

            return render_template('Login.html', msg="Invalid login
details")

        else:

            return render_template('UserScreen.html', msg="Welcome
"+uname)
```

```
@app.route('/SignupAction', methods=['GET', 'POST'])

def SignupAction():

    if request.method == 'POST':

        user = request.form['t1']

        password = request.form['t2']

        phone = request.form['t3']

        email = request.form['t4']

        address = request.form['t5']

        gender = request.form['t6']

        status = "none"

        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root',
password = '9391091987@India', database = 'negotiate',charset='utf8')

        with con:

            cur = con.cursor()

            cur.execute("select * FROM users")

            rows = cur.fetchall()

            for row in rows:

                if row[0] == user:

                    status = user+" Username already exists"
```

```python
                    break

        if status == 'none':

            db_connection = pymysql.connect(host='127.0.0.1',port =
3306,user = 'root', password = '9391091987@India', database =
'negotiate',charset='utf8')

            db_cursor = db_connection.cursor()

            student_sql_query = "INSERT INTO
users(username,password,contact_no,emailid,address,gender)
VALUES('"+user+"','"+password+"','"+phone+"','"+email+"','"+address+"','"+
gender+"')"

            db_cursor.execute(student_sql_query)

            db_connection.commit()

            if db_cursor.rowcount == 1:

                status = 'Signup process completed'

        return render_template('Signup.html', msg=status)




@app.route('/Logout')

def Logout():

    return render_template('index.html', msg='')

if __name__ == '__main__':

    app.run()
```

Testing and Deployment

Testing

Unit Testing:

- Purpose: Validate individual components of the application to ensure each part functions correctly in isolation.
- Components to Test:
    - User Authentication: Verify signup and login functionalities, ensuring users can securely create accounts and log in.
    - Product Browsing: Test the product display feature to ensure accurate and complete product information is shown, including price, description, and image.
    - Price Negotiation: Check the negotiation process to confirm the chatbot correctly interprets and processes user input, predicting reasonable prices.
    - Order Placement: Validate the order completion process, ensuring users can finalize purchases.
    - Review Posting: Test the review submission functionality, including sentiment analysis accuracy.
    - Order Viewing: Ensure users can view their past orders accurately.

Integration Testing:

- Purpose: Confirm that individual components work together seamlessly.
- Components to Test:
    - End-to-End Workflow: Test the entire user journey from signup, product browsing, price negotiation, order placement, review posting, to order viewing.

    - Database Interactions: Ensure that all CRUD (Create, Read, Update, Delete) operations between the application and the MySQL database work correctly.
    - API Interactions: Validate interactions between the frontend and backend, ensuring data is correctly passed and received.

User Acceptance Testing (UAT):

- Purpose: Gather feedback from real users to enhance the chatbot's functionality and overall user experience.
- Process:
    - User Feedback Sessions: Conduct sessions where actual users interact with the chatbot and provide feedback on usability, response accuracy, and overall satisfaction.
    - Iteration: Use the feedback to make necessary improvements, focusing on the chatbot's negotiation skills and user interface.

Testing Tools:

- Unit Testing Framework: Use Python's `unittest` **or** `pytest` **for unit tests.**
- **Integration Testing Tools: Utilize tools like Postman for API testing and Selenium for end-to-end testing.**
- **Sentiment Analysis Validation: Compare the chatbot's sentiment analysis results with manual sentiment assessments to ensure accuracy.**

Deployment

Cloud Service Providers:

- Amazon Web Services (AWS):
    - EC2 Instances: Deploy the Flask application on EC2 instances for scalable computing capacity.
    - RDS (Relational Database Service): Use RDS for managing MySQL databases.
    - S3 (Simple Storage Service): Store static assets like images and CSS files.
- Google Cloud Platform (GCP):
    - Google App Engine: Deploy the application on App Engine for automatic scaling.
    - Cloud SQL: Use Cloud SQL for managed MySQL databases.
    - Cloud Storage: Store static assets in Cloud Storage.
- Heroku:
    - Dynos: Deploy the Flask application using Heroku Dynos, which provide scalable execution environments.
    - Heroku Postgres: Utilize Heroku's managed PostgreSQL service as an alternative to MySQL.
    - Heroku Add-ons: Leverage add-ons for logging, monitoring, and performance optimization.

Deployment Steps:

1. Preparation:
    - Environment Configuration: Set up environment variables for database credentials, secret keys, and other sensitive information.
    - Dependency Management: Ensure all dependencies are listed in a `requirements.txt` **file for easy installation.**
2. **Deployment:**
    - **Version Control: Push the latest code to a version control system like GitHub.**

- CI/CD Pipeline: Set up a CI/CD pipeline using tools like GitHub Actions, CircleCI, or Jenkins to automate testing and deployment.
- Deploy Application:
  - AWS: Use Elastic Beanstalk or EC2 instances with auto-scaling groups.
  - GCP: Deploy using Google Cloud Run or App Engine for serverless execution.
  - Heroku: Use `git push heroku main` to deploy directly from the GitHub repository.

3. Post-Deployment:
   - Monitoring and Logging: Implement monitoring using tools like AWS CloudWatch, Google Cloud Monitoring, or Heroku's built-in logging.
   - Scaling: Set up auto-scaling rules to handle increased traffic.
   - Performance Tuning: Optimize database queries and application code for better performance.

Security Considerations:

- Data Encryption: Ensure all data is encrypted in transit using HTTPS and at rest using database encryption features.
- Authentication and Authorization: Implement robust authentication mechanisms, using OAuth if necessary, and ensure proper authorization checks.
- Regular Updates: Keep all software dependencies updated to protect against vulnerabilities.

By following these detailed testing and deployment strategies, the e-commerce price negotiating chatbot can be robustly validated and efficiently deployed, providing a seamless and secure user experience.

Future Enhancements

Enhanced Machine Learning Models:

- Implementation of Advanced Algorithms: Incorporate more sophisticated machine learning models such as Gradient Boosting Machines (GBMs), Random Forests, and Neural Networks to improve the accuracy of price predictions.
- Real-Time Learning: Develop a system that continuously learns from new data, allowing the model to adapt to changing market conditions and user behaviors.
- Feature Engineering: Identify and integrate additional features such as user demographics, purchasing history, and seasonal trends to enhance the predictive power of the models.

Voice Interaction:

- Improved Speech Recognition: Enhance the accuracy of speech recognition by integrating advanced natural language processing (NLP) libraries and APIs like Google Speech-to-Text or Microsoft Azure Speech Service.
- Voice Response Capabilities: Implement text-to-speech (TTS) functionalities to provide verbal responses, creating a more conversational and interactive experience.
- Noise Handling: Incorporate noise reduction algorithms to improve speech recognition accuracy in noisy environments.

Dynamic Pricing Strategies:

- Market Trend Analysis: Utilize real-time data analytics to monitor market trends and adjust prices dynamically based on demand and supply fluctuations.
- Competitor Price Monitoring: Implement web scraping and API integrations to track competitor prices and adjust the chatbot's pricing strategies accordingly.

- Personalized Offers: Leverage user data to provide personalized discounts and offers, increasing conversion rates and customer loyalty.

Multi-Language Support:

- Language Detection: Implement language detection mechanisms to automatically identify the user's preferred language.
- Translation Services: Use translation APIs like Google Translate or Microsoft Translator to support multiple languages, ensuring accurate and context-aware translations.
- Cultural Adaptation: Adapt the chatbot's responses to align with cultural nuances and preferences, enhancing the user experience for a global audience.

Conclusion

The price negotiating chatbot for e-commerce websites exemplifies the potential of leveraging advanced technologies to enhance the online shopping experience. By integrating machine learning and sentiment analysis, the chatbot not only provides personalized and dynamic pricing but also creates a more engaging and interactive environment for users.

Enhanced User Engagement:

- **Real-Time Interactions: The chatbot's ability to negotiate prices in real-time makes the shopping process more interactive and engaging for customers. This feature can significantly enhance user satisfaction by providing a sense of participation and control over their purchases.**
- **Immediate Responses: Unlike traditional customer service methods, the chatbot can respond instantly to user queries and negotiation attempts, reducing wait times and improving overall user experience.**

Improved Customer Satisfaction:

- **Personalized Pricing: By utilizing machine learning algorithms, the chatbot can offer prices tailored to individual customers based on their behavior, preferences, and purchase history. This personalized approach can lead to higher customer satisfaction and loyalty.**
- **Sentiment Analysis: The inclusion of sentiment analysis allows the chatbot to gauge user emotions and respond appropriately. Understanding customer sentiments helps in refining pricing strategies and improving customer relations.**

Business Benefits:

- **Increased Sales Conversion: Personalized and dynamic pricing can attract more customers and increase sales conversion rates. By offering competitive prices and special deals, the chatbot can help in converting more visitors into buyers.**
- **Market Competitiveness: The ability to adjust prices dynamically based on market trends and competitor prices ensures that the business remains competitive. This adaptability can be crucial in staying ahead in a highly competitive e-commerce landscape.**

Scalability and Efficiency:

- **Handling Multiple Users: The chatbot is designed to handle multiple user requests simultaneously, ensuring that the platform can scale efficiently with growing user numbers.**
- **Reduced Operational Costs: By automating price negotiations and customer interactions, businesses can reduce their reliance on human customer service agents, leading to cost savings.**

Future Potential:

- Advanced Machine Learning Models: As the project evolves, implementing more sophisticated machine learning algorithms will enhance the accuracy and reliability of price predictions, making the chatbot even more effective.
- Voice Interaction Improvements: Enhancing speech recognition capabilities will facilitate smoother voice-based interactions, making the chatbot more accessible and user-friendly.
- Dynamic Pricing Strategies: Incorporating real-time market data and competitor analysis will enable the chatbot to implement dynamic pricing strategies, further improving its effectiveness.
- Multi-Language Support: Adding support for multiple languages will allow the chatbot to cater to a global audience, expanding its reach and usability.

In conclusion, the price negotiating chatbot represents a significant advancement in the e-commerce sector, blending the power of machine learning and sentiment analysis to offer a unique and improved shopping experience. As the chatbot continues to evolve with advanced features and capabilities, it promises to revolutionize the way customers interact with e-commerce platforms, driving higher engagement, satisfaction, and business growth. This project not only highlights the current capabilities of chatbots but also sets the stage for future innovations in automated customer service and personalized shopping experiences.