Highrise FAQ Chatbot

This project is a prototype of an FAQ chatbot designed to assist users with questions about the Highrise app. The chatbot can answer user queries by providing relevant responses from the Highrise FAQ section found at https://support.highrise.game/en/.

Author: Keenan Kalra
Date: December 2024

Table of Contents

Project Structure

HighriseAssignment/ ├── README.md ├── data/ | ├── faq_data.json | ├── processed_faq_data.json | └── rag_processed_faq_data.json ├── logs/ | ├── interactions.log | └── unanswered_queries.log ├── requirements.txt ├── scraping.ipynb └── src/ ├── tfidf_chatbot/ | ├── chatbot.py | ├── main.py | ├── preprocess.py | ├── synonyms.py | └── utils.py ├── rag_chatbot/ | ├── **init**.py | ├── chatbot.py | ├── compute_embeddings.py | ├── main.py | ├── preprocess.py | ├── utils.py | └── vector_store.py └── **init**.py

- **README.md**: This file.
- **data/**: Contains the scraped FAQ data and processed versions.
- **logs/**: Contains logs of user interactions and unanswered queries.
- **requirements.txt**: Lists all Python dependencies.
- **scraping.ipynb**: Jupyter notebook used for scraping the FAQ data.
- **src/**: Contains source code for both the TF-IDF and RAG chatbots.
    - **tfidf_chatbot/**: Contains the TF-IDF-based chatbot implementation.
    - **rag_chatbot/**: Contains the Retrieval-Augmented Generation (RAG) chatbot implementation.

Setup Instructions

Prerequisites

- Python 3.7 or higher
- An OpenAI API key (for the RAG chatbot)
- Required Python packages (listed in `requirements.txt`)

Installation Steps

1. **Clone the Repository** git clone https://github.com/yourusername/HighriseAssignment.git cd HighriseAssignment

2. **Install Dependencies**

   Install all required Python packages using pip:

   pip install -r requirements.txt

3. **Set Up Environment Variables**

   For the RAG chatbot, set your OpenAI API key as an environment variable:

   export OPENAI_API_KEY='your-api-key-here'

   Replace 'your-api-key-here' with your actual OpenAI API key.

4. **Download NLTK Data**

   For text processing, some NLTK data packages are required. Run the following commands in a Python shell:

   import nltk nltk.download('punkt') nltk.download('stopwords') nltk.download('wordnet')

How to Run the Chatbot

TF-IDF Chatbot

The TF-IDF chatbot is a simple implementation that uses TF-IDF vectorization and cosine similarity to match user queries to the FAQ data.

Steps to Run:

1. **Preprocess the Data**

   python src/tfidf_chatbot/preprocess.py

   This script preprocesses the FAQ data for the TF-IDF model.

2. **Run the Chatbot**

   python src/tfidf_chatbot/main.py

RAG Chatbot

The RAG chatbot uses OpenAI's GPT models to generate responses, augmenting retrieval from the FAQ data.

Steps to Run:

1. **Preprocess the Data**

   python src/rag_chatbot/preprocess.py

   This script processes the FAQ data, splitting articles into individual question-answer pairs and cleaning the text.

2. **Compute Embeddings**

   python src/rag_chatbot/compute_embeddings.py

   This script computes embeddings for the processed data using OpenAI's embedding models.

3. **Build the Vector Store**

   python src/rag_chatbot/vector_store.py

   This script builds a FAISS index for efficient similarity search.

4. **Run the Chatbot**

   python src/rag_chatbot/main.py

Optional Features Implemented

- **Basic NLP Techniques**: Implemented lemmatization and synonym expansion in the TF-IDF chatbot to improve text matching accuracy.
- **Retrieval-Augmented Generation (RAG)**: Integrated OpenAI's GPT models to enhance response generation in the RAG chatbot.
- **Logging**: Both chatbots implement logging of user interactions and track unanswered queries for future improvements.

Logs

- **interactions.log**: Logs all user queries and chatbot responses with timestamps.
- **unanswered_queries.log**: Records user queries that the chatbot couldn't answer satisfactorily.

These logs are located in the `logs/` directory.

Next Steps

- **Deployment**: Deploy the chatbot on a platform like Vercel or Heroku to allow direct interaction.
- **User Interface**: Create a simple web or command-line interface for better user experience.
- **Improved Error Handling**: Enhance exception handling for network errors, API rate limits, and other potential issues.
- **User Feedback Mechanism**: Implement a feature to allow users to rate responses as helpful or unhelpful, storing this data for future improvements.
- **Learning Mode**: Enable the chatbot to learn from unanswered or misclassified questions by incorporating them into the dataset after review.

Acknowledgments

- **Highrise**: For providing the FAQ data used in this project.
- **OpenAI**: For the GPT models used in the RAG chatbot.
- **NLTK**: For the natural language processing tools used in preprocessing.