Ticket Categorisation

Introduction

This document is written to just give a brief introduction of a solution to classify tickets and potentially generate responses to customer queries. I believe that just using the openai api to classify the tickets will lead to greater divergence in accuracy and lead to higher costs and lower efficiency.

So here I have discussed another potential method we could use to classify tickets and generate responses to customer queries.

Approach

I believe we need to take a multifaceted approach to make sure that we are getting much more accurate results in the long term, given that we have solid underlying business logic but more nuances in understanding customer queries and give custom solutions to customers in some cases.

My code architecture revolves around 2 primary components:

1) An embedding model for semantic understanding of queries
   I used ‘OpenAiEmbeddings’ to transform customer queries into high dimensional vectors. Using this we can understand the semantic essence of each query rather than just focusing on the meaning of the text.
2) A similarity search engine for efficient categorization. I used similarity search with the FAISS (Facebook ai similarity search) library which is an open source library. This stores the dataset in memory for fast retrieval and search results.This allows to accurately retrieve the most similar historical queries making the categorization efficient.

What this allows us to do is that , when we have better data we can just swap out the data file and it will continually allow us to scale and adapt and make the accuracy higher.

Libraries used

Here, I have mainly used Langchain combination libraries, the libraries which are not mentioned here are standard libraries.

langchain_community.vectorstores.FAISS: An extension of the langchain library, integrating FAISS for efficient similarity search in large sets of high-dimensional vectors. It's used to index and search for similar historical queries in your application.

langchain_openai.OpenAIEmbeddings: A component of the langchain library that interfaces with OpenAI's models. This library is used for generating embeddings of text data, leveraging OpenAI's advanced NLP capabilities.

langchain.prompts.PromptTemplate: Another part of the langchain suite, this tool is used to create structured prompts for AI models. It allows for the customization of input and output formats when interacting with language models.

langchain_openai.ChatOpenAI: Integrated within the langchain framework, this library facilitates interaction with OpenAI's GPT models, particularly for generating language-based responses.

langchain.chains.LLMChain: This library chains together various components (like prompt creation and AI response generation) to streamline the processing of text data through language models.

API Documentation

I have also included an api written in python which has the same functionality, you can host this azure web service for ease of integration as everything else in our company is running on azure, you won't be able to use serverless components as we use in memory for Faiss.

The api needs a POST request in JSON

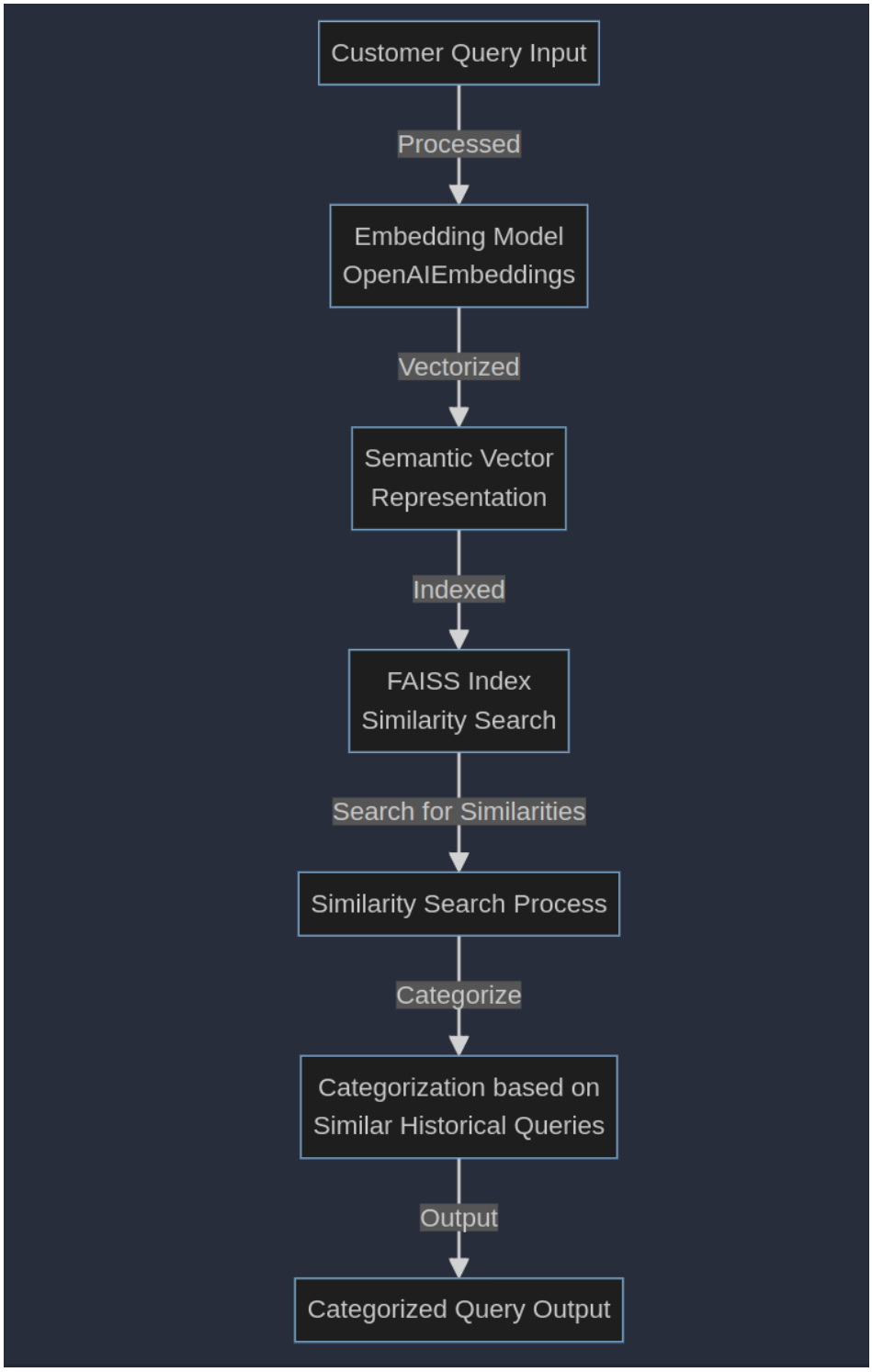
Request Body:

```
{
  "message": "I have a question about my energy bill this month."
}
```

Response:

```
{
  "response": "Category: Billing Inquiry"
}
```

Code architecture



Future Improvements:

We can further improve the code, by using the fine tuned model gpt 3, which can be trained with a couple of thousand examples. Which can improve the efficiency and accuracy and reduce costs at the same time.

Future Use Cases:

This code can be easily modified to generate agent responses along with the category. GIven that we have a clean and sorted dataset for agent responses.