SOMM: A WINE RECOMMENDATION ENGINE

EXECUTIVE SUMMARY

Palak Arora Pavel Kovalev Vishal Kumar Veronica Miatto Jade Nguyen

Overview

We designed a wine recommendation app that helps users find a wine based on its main flavor notes and the taste the users are looking for. Users can find it difficult to choose a wine among hundreds of thousands of wine varieties, whose flavor varies across wineries, geographical regions, and years of production. Moreover, wine distribution and categorization are very inconsistent, with only a small and ever-changing part of all existing wines available at a specific physical or online store. A wine recommendation engine based on flavor might, therefore, bridge that gap between inconsistent wine availability and taste. This will benefit both the average wine buyer, whose knowledge of wines is limited and who will be able to find wine to their liking more easily, and wine distributors and resellers, who might see an increase in their sales once they start offering this service or use the engine as a procurement tool. Besides recommending specific wines, our engine also suggests styles (e.g., Chardonnay from France) based on users' tastes, providing a useful feature if the suggested wines are not in stock. Current wine recommendation engines, such as Vivino and Delectable, are inadequate in recommending a wine based on flavor. In fact, the first only allows searching for wines based on categorical factors like name, grape type, price, while the second only allows access to information on the wine flavor by scanning a wine label.

Dataset & EDA

We used a dataset containing information and reviews on 130k wines (from Kaggle, based on reviews from WineEnthusiast). The information provided about the wines includes the name of the wine, country, region, winery, price, and a description that illustrates its flavor in a manner reminiscent of a sommelier's description. The dataset was cleaned by dropping rows with NaN entries in essential columns, removing duplicate rows, and eliminating unnecessary columns, such as taster and tater's Twitter handle (information on who wrote the review). The exploratory analysis conducted found that most wines are from the US (40%), France (15.7%), and Italy (15%), while the most common varieties of wine are Pinot Noir (12.4%), Chardonnay (9.9%), Red blend (9.3%), Cabernet Sauvignon (7.3%), and Riesling (5.5%).

Solutions

We explored various iterations of two main approaches: classification models and a Retrieval Augmented Generation (RAG) pipeline. Both approaches begin with text embeddings of wine reviews generated by the OpenAI embedding model ada-002.

The idea of the classification approach is to classify wines by styles based on text embeddings of reviews. We fed the user's query to the fitted model and predicted the most likely styles (e.g., Red Blend from Italy). Among the predicted styles, we filtered for top wines that best fit the user's tastes. To do this, we first defined 86 styles, which combine Region (or Country or State, based on wine provenance), and Grape Variety. We dropped styles with fewer than 200 reviews, resulting in 40% of the data being dropped in this approach. We implemented various classification algorithms and found that XGBoost gives the most robust style recommendations, with a top-3 accuracy of 80%. Although classification models performed quite well in recommending a style of wine, they did not handle negations well when recommending a specific wine.

The implemented RAG pipeline for a wine recommendation system employs Pinecone as the vector database, Cohere Rerank as the reranker, and OpenAI GPT-4 as the large language model (LLM), all integrated through LangChain. The pipeline uses a pre-created Pinecone vector index containing information about wines from the dataset. When presented with a user's query and previous chat history (if any), the LLM generates a new query (or uses the original query if the chat history is empty) and queries the Pinecone index. The top N most similar vectors, based on cosine similarity, are then returned. Subsequently, the documents corresponding to these vectors undergo reranking by Cohere Rerank, and the top n (out of N) most relevant vectors are retained, where $n < N$. The original query, chat history, hard-coded prompt, and the n documents are provided to the LLM, which produces the final result.

The benefits of employing RAG in our system are manifold. Firstly, it caters to user preferences by offering concise summaries instead of lengthy reviews, improving readability. Additionally, using the knowledge embedded in the LLM allows us to enhance the output by instructing the LLM to include informative columns, such as food pairings for each wine. RAG also streamlines the information output by preventing the inclusion of redundant or irrelevant details, such as the common occurrence of "Other" in our dataset if such an instruction is included in the prompt. Furthermore, the system has memory, enabling users to ask follow-up questions. This can be used either to refine or change the original results or to modify the output format, adjusting it to the user's preference.

We evaluated Classification and RAG with 100 test queries. For wine recommendation, we found that RAG scored 11% higher than Classification in terms of relevance between the user's query and the model's suggestions. For style recommendation, we measured the similarity between the user's query and all wines underlying the suggested styles; we found that all models performed equally well. We chose RAG for both deliverables to ensure consistency between them.

Finally, we implemented a user-friendly web app that returns a table recommending the five wines that best fit the user's query, containing the title, description, variety, country, region, winery, province, and recommended food pairing of the wines. The default format of the output can be controlled by the prompt, and furthermore, the user can force a different format of displaying the results by mentioning their preferences in the query.

Pitfalls and Future Work

The most persistent problem is the poor ability of semantic search to handle negations. We tried several text embedding models, but all of them failed to account for negations. For example, when searching for "fruity but not oaky wines", the result of the similarity search would consistently return oaky wines. Since the first step of RAG is also a similarity search, this problem also persists in our RAG approach. However, GPT-4 manages to fix this problem with subsequent user queries, i.e., when the conversation history is nonempty.

Another potential point of improvement is filtering by the metadata. Currently, the first stage of the RAG pipeline only uses similarity search. Adding an automatic filtering component may further improve search results when the user is looking for wines from a specific country or winery. While LangChain's SelfQueryRetriever can be used for this purpose, we found that it does not work well with queries containing negations and requires further improvement.

Lastly, the dataset may need to be expanded to include wines from a more diverse geographic distribution and more than one sommelier review per wine. This expansion will make our models stronger and enable it to more accurately pinpoint the flavor profile of a wine.