**Ecommerce Product Reviews - RAG Pipeline with Groq API**

**Purpose**

This document provides a comprehensive overview of a Retriever-Augmented Generation (RAG) pipeline designed to handle structured and unstructured datasets.

**Objectives**

1. Create an efficient, scalable data pipeline for ingesting and processing structured/unstructured data.
2. Implement vector-based similarity retrieval for high-accuracy querying.
3. Integrate Groq API for natural language generation to enhance user experience.
4. Ensure robust monitoring, logging, and documentation for enterprise-grade reliability.

**Architecture Overview:**

**Technology Stack**

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Reasoning** |
| **Data Storage** | MongoDB | Efficient handling of semi-structured data. |
| **Vectorization** | SentenceTransformer (all-MiniLM-L6-v2) | State-of-the-art embeddings for high-quality similarity. |
| **Text Generation** | Groq API (llama3-8b-8192) | Advanced model for context-based response generation. |
| **Backend** | Flask | Lightweight API framework for integration. |
| **Logging/Monitoring** | Python Logging, MongoDB Ping | Ensures pipeline health and issue detection. |

**Detailed Implementation**

**1. Data Ingestion**

The pipeline ingests product review data from CSV files, such as amazon-review.csv and amazon.csv. Schema validation ensures that data is consistent with the required format. MongoDB stores the ingested data in a product\_reviews4 collection, optimized for querying with indexed fields like product\_id, review\_content, and rating.

**2. Data Preprocessing**

* **Handling Missing Values**: Text fields are filled with “Unknown”, numeric fields are replaced with column means, and date fields default to 1970-01-01.
* **Text Normalization**: Text data is converted to lowercase, extraneous punctuation and emojis are removed, and leading/trailing whitespaces are trimmed.
* **Duplicate Removal**: Ensures uniqueness of entries for better storage and query performance.

**3. Vectorization**

Text data is vectorized using the **SentenceTransformer (all-MiniLM-L6-v2)** model. This model is pre-trained and fine-tuned for natural language tasks, making it ideal for embedding text and supporting similarity-based queries. The resulting embeddings are stored in MongoDB in a JSON-serializable format for efficient retrieval.

**4. Query and Retrieval**

To enable high-accuracy querying:

* **Query Encoding**: User queries are encoded into embeddings using the same SentenceTransformer model.
* **Similarity Calculation**: Cosine similarity is used to compare the query embedding with stored document embeddings, retrieving the top 5 most similar results.
* **Context Creation**: The most relevant reviews are aggregated to form a context string used for response generation.

**5. Response Generation with Groq API**

The pipeline integrates the **Groq API (llama3-8b-8192)** to generate context-aware responses:

* **Input**: User query text and the context string generated from retrieved reviews.
* **Output**: A generated response that addresses the user's query with accurate and relevant information.
* **Authorization**: API requests are securely authenticated using a Bearer token and communicate over HTTPS to ensure data privacy and security.

**Key Design Considerations**

* **Dynamic Query Handling**: The pipeline handles a variety of query types, from detailed product information to abstract feedback, ensuring versatility in user interactions.
* **Performance**: The pipeline is designed to ensure fast query responses, with vector similarity calculations optimized for sub-second latency.

**6.Monitoring and Scalability**

* **Logging**: Tracks data ingestion, preprocessing, and API call statuses and detailed error logs identify schema mismatches, embedding issues, or query failures.
* **Parallel Processing**: Batch processing for large datasets ensures scalability.
* **Indexing**: Indexed key fields (product\_id, review\_content) to optimize query performance.

**Security Considerations**

**API Security**: API key is stored securely and accessed through environment variables, API requests use HTTPS to encrypt communication.

**Future Enhancements**:

1. Implement fine-tuning for domain-specific embeddings.
2. Introduce caching mechanisms to optimize query response times.
3. Enable support for multi-language datasets to extend usability across regions.

**Code Repository**: [RAG\_ProductReview\_Project](https://github.com/preshma284/RAG_ProductReview)