QA System Technical Documentation

Table of Contents

Introduction

The Question-Answering (QA) system presented here is designed to retrieve, rank, and present textual passages based on user queries. Using Elasticsearch, the system is adept at processing vast amounts of data, providing relevance-based scoring, and ensuring quick response times.

System Architecture

Frontend: A Command-Line Interface (CLI) ensures a straightforward approach to input queries and view results.
Backend: An Elasticsearch instance, handling the indexing, searching, and scoring of passages in the dataset.
Data Store: CSV files serve as the primary storage medium for data, evaluation parameters, and resultant metrics.

System Requirements

Environment: Development and execution require Python 3.7 or higher.
Database: Elasticsearch 7.x or compatible versions.
Python Libraries: `requests` and `csv`.

Data Preparation & Management

Dataset: `legal_data.csv`
Data Cleaning: Remove duplicates, handle missing values, and normalize text.

Retrieval System: Backend Implementation

Elasticsearch Setup

Installation, configuration, and initialization.

Retrievals Class

Core of the retrieval system within `retrieval.py`.

Data Indexing

Indexing data into Elasticsearch for query processing.

Configuration Management

Different configurations for various environments (development, production, testing).

Performance Evaluation & Metrics

Data Retrieval for Evaluation

Use `user_queries.txt` for queries.

Manual Relevance Rating

Human evaluation for relevance.

Accuracy Calculation

Metrics saved in `performance.csv`.

User Interface for Retrieval

CLI usage: `python script_name.py "User's Query"`

Design Choices

Elasticsearch as Backend

Choice: Using Elasticsearch.
Rationale: Elasticsearch offers full-text search capabilities, distributed nature, and scalability. Its built-in scoring mechanisms save time and effort.

CLI as Frontend

Choice: Building a Command-Line Interface.
Rationale: A CLI is lightweight, platform-independent, and provides a quick way to test the backend without heavy UI development.

CSV for Data Storage

Choice: Employing CSV files.
Rationale: CSV is a universally accepted format, easy to read/write, and requires no special database setup. It's ideal for tabular data, as in our use case.

Configuration Management

Choice: Multiple configurations for different environments.
Rationale: Different environments (development, production, testing) have distinct requirements. By setting configurations tailored to each, we ensure optimal performance and easier debugging.

Manual Relevance Rating

Choice: Using human evaluators for relevance rating.
Rationale: While machine algorithms can predict relevance, the final arbiter of a passage's relevance to a human query should be a human. This step ensures the system's results are genuinely meaningful to users.

Conclusions & Future Recommendations

The QA system is robust and efficient. Future recommendations include advanced NLP techniques, GUI deployment, and feedback integration.