RAG System using Scikit-Learn Documentation

Sri Harshan Ganugula, Mahesh Babu Kommalapati, Srilakshmi Kanagala

Abstract

The advancement of information retrieval systems has been pivotal in enhancing user interaction with extensive databases. This project proposes the development of a Retrieval-Augmented Generation (RAG) system designed specifically for enhancing the usability of scikit-learn's documentation. Addressing the challenge of navigating extensive technical data, the system allows users to pose natural language queries and promptly receive relevant, concise information. This functionality not only mitigates the need for manual exploration of voluminous documentation but also streamlines user interaction with the scikit-learn framework. The system's core operations involve retrieving relevant documents, generating potential responses, and delivering the most accurate answers accompanied by source citations. 400 question-and-answer pairs were crafted using GPT-4 to validate the models used. The results demonstrate significant improvements in information retrieval and answer quality, confirming the efficacy of integrating contextual data into the response generation process.

Introduction

In the rapidly evolving field of machine learning, the ability to quickly and effectively access technical documentation is critical. Scikit-learn, a leading Python library for machine learning, offers robust tools for data analysis and modeling but is accompanied by comprehensive and complex documentation. This wealth of information, while invaluable, can often be daunting for both newcomers and experienced practitioners alike.

To address this challenge, our project introduces a Retrieval-Augmented Generation (RAG) system designed specifically to transform how users interact with scikit-learn documentation. By integrating advanced natural language processing and information retrieval technologies using LLM's, our system enables users to engage with the documentation through intuitive natural language queries. This approach simplifies the search process and enhances the learning and implementation experience by providing precise, contextually relevant answers through a chat interface.

Scraped official scikit-learn documentation to create a comprehensive dataset, from which a synthetic question-and-answer dataset is generated using GPT-4, providing a robust platform for training and evaluation. Our findings suggest that this approach significantly enhances the accessibility and usability of scikit-learn, paving the way for more efficient learning and application of machine learning techniques. Our goal is to make scikit-learn's resources more accessible and user-friendly, thereby empowering more developers to leverage this powerful tool in their machine-learning projects.

Background

1. Retrieval-Augmented Generation (RAG) Framework:

- **Concept Overview:** RAG combines the capabilities of information retrieval (IR) systems with advanced state-of-the-art generation models to answer queries more accurately. It uses an IR component to fetch relevant documents or data snippets and a generative model to synthesize responses from the retrieved information.

- **Previous Work:** Introduced by Lewis et al., in the paper "Retrieval-Augmented Generation for Knowledge-Intensive Tasks", RAG has been primarily used to enhance performance on NLP tasks such as question answering and fact verification by leveraging external knowledge sources.

2. Scikit-learn Documentation:

- An extensive framework that includes user guides, API references, and tutorials, making it a perfect candidate for applying RAG technology to improve the usability of such dense informational resources.

3. Information Retrieval Techniques:

- **Vector Space Models:** Techniques such as TF-IDF and semantic embeddings allow systems to retrieve information based on the contextual similarity rather than mere keyword matching. These models represent documents and queries as vectors in a high-dimensional space, facilitating the retrieval of relevant documents even if the exact terms do not match.

- **Database Technology:** Vector databases like FAISS, ChromaDB, Pinecone are used to store and efficiently retrieve high-dimensional vector embeddings of documents, which are essential for the quick fetching of relevant information in response to user queries. The content is converted into numerical embeddings and are stored in the vector database to enable quick retrieval.

4. Natural Language Processing and Machine Learning:

- **Hugging Face Transformers:** The Hugging Face Transformers library is a comprehensive, open-source library designed especially for natural language processing (NLP) tasks. The library offers access to thousands of pre-trained models that are optimized for a wide range of NLP tasks. The generation models and embedding models used in this project are chosen from Multitask Text Embedding Benchmark (MTEB) dashboard[10]. The MTEB dashboard on Hugging Face is a valuable resource for evaluating and selecting the appropriate embeddings model for specific tasks. The chosen models are imported and integrated from Huggingface using transformers library.

- **Generative Models:** Open source and OpenAI's generative models, particularly Mistral 7B, Mixtral 7x8B, Llama-3-8b-chat-hf, and GPT-4-Turbo, are pivotal in generating coherent and contextually appropriate responses based on the inputs received from the retrieval system.

- **Embeddings from Transformers:** These models (gte-large, gte-base, and text-embeddings-3-large) provide deep contextualized word embedding that capture complex characteristics of word use and how these uses vary across linguistic contexts, enhancing the understanding and generation capabilities of the system.

5. Evaluation Metrics:

- **Dual-scoring Framework:** Involves assessing both the relevance of the information retrieved (Retrieval Score) and the quality of the responses generated (Quality Score), ensuring the system's effectiveness in real-world applications.

- By grounding the project in these established theories and practices, the work on the RAG system for scikit-learn documentation not only builds on proven methods but also pushes the boundaries of what's possible with AI-enhanced educational tools. This background provides a thorough understanding of the project's theoretical and practical dimensions, setting the stage for appreciating the innovative approaches and solutions implemented in the RAG system.

Related Work

Developing information retrieval systems that integrate with language models represents a significant trend in improving user interactions with data-rich environments. This section reviews several key advancements and existing systems that provide context and foundation for the current project.

Foundational Research and Developments in RAG Systems:

1. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks:**
   This paper introduces the RAG architecture that combines a neural retriever and a neural generator to enhance the performance of language models on knowledge-intensive tasks. Our RAG system for scikit-learn documentation adopts a similar architecture to fetch relevant documentation segments before generating responses, ensuring that the generated content is both accurate and informative.

2. **Retrieval-Augmented Generation for Large Language Models:**
   Explores the scalability of RAG techniques to larger language models and diverse datasets, demonstrating improvements in information retrieval and response generation quality. Relevance to Project: The scalability aspects discussed in this study guided the design of our system, ensuring it remains effective as the volume of scikit-learn documentation grows.

3. **REALM: Retrieval-Augmented Language Model Pre-Training:**
   Describes a method for integrating retrieval into the pre-training process of language models, which helps models utilize external knowledge effectively. Relevance to Project: Inspired by REALM, our project also incorporates retrieval mechanisms during the pre-training of embeddings specifically tuned to the scikit-learn documentation, enhancing the model's understanding of machine learning concepts.

4. **Dense Passage Retrieval for Open-Domain Question Answering:**
   Introduces a novel retrieval method that uses dense vector representations to retrieve relevant passages across large text corpora efficiently. Relevance to Project: This method influenced our approach to embedding scikit-learn documentation, allowing for precise retrieval of information based on semantic similarity rather than lexical matches.

Alternative Methods and Comparison:

- **Sparse Vector Space Models:**
  Traditional information retrieval systems often use sparse vector representations like TF-IDF or BM25. While effective for keyword-matched queries, they lack the semantic understanding necessary for complex query answering, which is critical for technical documentation like scikit-learn.
- **Foundation Models:**
  Complete reliance on generative models without a retrieval step can lead to responses that are plausible but factually incorrect or hallucinated. Our choice to integrate a retrieval step mitigates this by grounding responses in the actual documentation.

How They Relate to Our Method:

The alternate methods - Sparse Vector Space Models and Foundation Models, and the RAG bot emphasize the importance of integrating retrieval mechanisms to enhance the accuracy and reliability of the generated responses in documentation and educational tools. They share a common theme in addressing the limitations of purely generative or traditional sparse vector retrieval methods in handling complex, technical content.

Why we did not use them:

Our decision to use a Retrieval-Augmented Generation model over other methods was influenced by the need for high accuracy and reliability in accessing and synthesizing technical content. RAG systems provide a balance of dynamic knowledge retrieval and advanced text generation, making them ideal for educational and documentation tools where up-to-date and contextually accurate information is crucial.

This review of related work not only highlights the foundational theories and practices that our project builds upon but also illustrates the advancements and unique contributions our RAG system brings to enhancing the accessibility and usability of scikit-learn documentation. By comparing alternative methods, we further validate our choice of a RAG system, emphasizing its suitability for complex, knowledge-intensive tasks in technical domains.

Project Description

Overview

This project aims to design and implement a Retrieval-Augmented Generation (RAG) system specifically for scikit-learn documentation, addressing the challenge of efficiently navigating and utilizing extensive technical documentation. By leveraging advanced natural language processing (NLP) techniques and retrieval mechanisms, the system will enable users to interact with scikit-learn documentation through intuitive natural language queries, thereby enhancing their learning and development experience. The architecture of this system integrates several components: user query input, document retrieval, context processing, and response generation. Below, we describe the workflow and key components in detail:

System Components

1. Data Collection and Preparation:

a) Source Data: The system utilizes scikit-learn's official documentation, including user guides, API references, and tutorials, sourced directly from the scikit-learn website. This comprehensive dataset forms the foundation of the knowledge base for the RAG system. The data is scraped from the official scikit-learn documentation using Beautiful soup and 'html2text'. Utilizing web scraping tools, we meticulously collected HTML content from the documentation pages. Post-collection, the data underwent a cleaning process to remove unnecessary HTML tags, normalize the text, and segment the content logically. Each segment was enhanced with metadata to facilitate efficient retrieval and processing in subsequent stages.

Algorithm 1: Extract Sections from HTML Document

```
function ExtractSections(record)
    html ← LoadHTML(record["path"])
    RemoveExamplesUsingSection(html)
    soup ← ParseHTML(html)
    sections ← soup.FindAll("section")
    sectionList ← new list
    if Empty(sections) then
        markdown ← ConvertToMarkdown(soup)
        uri ← ConstructURI(record["path"])
        if markdown then
            Append(sectionList, {"source": uri, "text": markdown})
        end if
    else
        for section in sections do
            id ← GetAttribute(section, "id")
            markdown ← ConvertToMarkdown(section)
            uri ← ConstructURI(record["path"], id)
            if markdown then
                Append(sectionList, {"source": uri, "text": markdown})
            end if
        end for
    end if
    return sectionList
end function
```

b) Evaluation Dataset: To evaluate the system, a synthetic dataset of approximately 400 question-answer pairs is generated using GPT-4 by providing specific instructions, appropriate source and relevant content. This dataset is designed to mimic real-world queries that users might pose, covering a broad range of topics within scikit-learn's documentation.

The following prompt is used to generate the synthetic data:

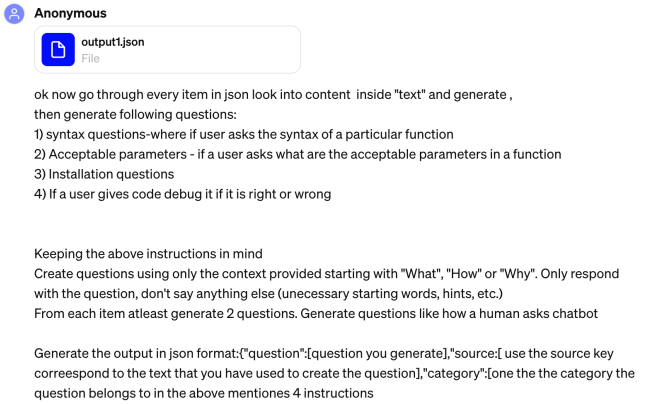

Figure 1: Prompt used to generate Synthetic Data

The following is the format of the synthetic data:

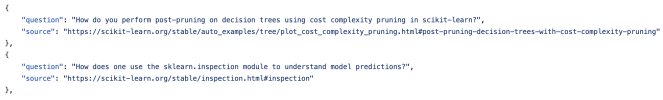

Figure 2: Format of the Synthetic Data

2. Information Retrieval System:

a) Chunking Documentation: To facilitate effective information retrieval, the scikit-learn documentation is first segmented into smaller, manageable chunks. This segmentation allows the system to focus on discrete pieces of information, which are more directly comparable to user queries.

b) Embedding Generation: Each chunk is then transformed into a numerical format known as embeddings. We use state-of-the-art embedding models to convert text into high-dimensional vectors that capture the semantic essence of the text. This process involves models like:

- **thenlper/gte-base:** A general text embedding model suitable for a broad range of text types.
- **thenlper/gte-large:** An advanced version of gte-base, offering more detailed embeddings suitable for complex datasets.
- **text-embedding-3-large:** Specifically designed for larger text blocks, providing deep semantic analysis.

c) Indexing in Pinecone: The generated embeddings are then indexed in a Pinecone vector database. Pinecone is chosen for its efficiency in handling high-dimensional data and its capability to perform fast semantic similarity searches. This setup ensures that when a user query is received, the system can quickly identify and retrieve the most semantically similar document chunks from the database.

The Information Retrieval System segment of the RAG system plays a crucial role in ensuring that users' queries are matched with the most relevant and semantically similar documentation segments. By employing advanced embedding models and leveraging Pinecone's efficient vector database, the system enhances the user's ability to quickly and effectively access the information they need from the scikit-learn documentation. This infrastructure not only supports the rapid retrieval of information but also ensures that the data remains structured and accessible in a way that maximizes the relevance of search results.

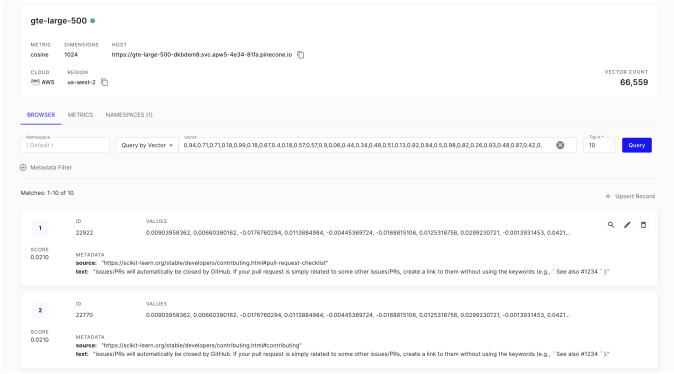

Figure 3: Embeddings stored in Pinecone

3. Prompt and Query Initiation:

The system begins with user input, where users articulate their queries in natural language. This input is then combined with a predefined system prompt that helps in forming a more structured query for internal processing.

a) Query Processing for Relevant Information: Each query is processed using a vector database (Pinecone), where embeddings of the scikit-learn documentation are stored. This process involves searching the vector space for the closest embeddings that match the query's intent, ensuring that the retrieval is contextually aligned with the user's request.

b) Information Retrieval for Context Enhancement: The most relevant documents are retrieved based on their cosine similarity to the query vector. This step is crucial as it determines the quality and relevance of the information that will be used in generating the response.

4. Generative Model Integration:

a) Language Models: Several advanced, pre-trained language models, namely gpt-4-turbo, lLama-2-70b-chat-hf, mistral-7b, and mixtral-8x7b were used since they are capable of understanding and generating natural language responses, thereby significantly enhancing the system's ability to interact intuitively with users.

b) Augmentation via RAG: Combining the retrieved document snippets with the query, the language model generates a coherent and contextually relevant response. This step synthesizes the information from the documentation with the generative capabilities of the language model, producing precise and actionable answers.

5. System Architecture:

a) The architecture includes a user-friendly interface where users can input their queries in natural language.

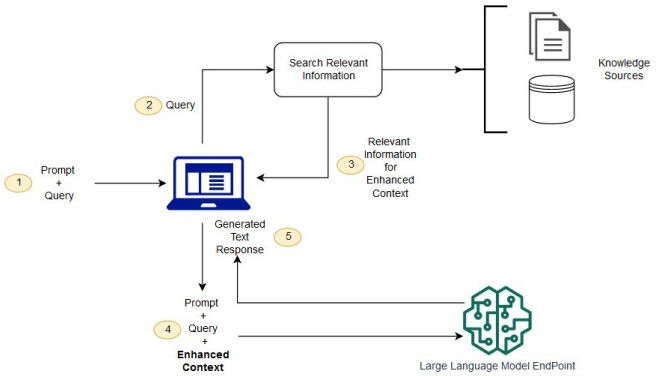

Figure 4: Architecture of RAG

b) The backend consists of the retrieval module and the generative module, which work in tandem to process queries and generate responses. The integration ensures that the context from the retrieved documents is effectively utilized to inform the response generation.

6. Evaluation and Metrics

The effectiveness of the RAG system is measured using a dual-scoring framework:

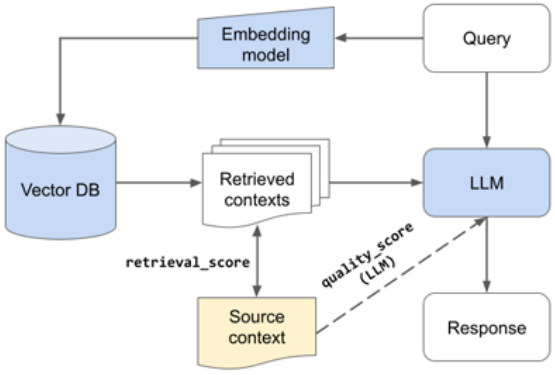

Figure 5: Components evaluation

Retrieval Score: Assesses how effectively the system identifies and retrieves relevant information in response to a user's query. A higher score indicates better accuracy in fetching pertinent chunks.

Algorithm 2: Calculate Retrieval Score

function CALCULATERETRIEVALSCORE($testData$, $retrievalSystem$)
  $successfulRetrievals \leftarrow 0$
  $totalQueries \leftarrow$ LENGTH($testData$)
  for each $query$ in $testData$ **do**
    $expectedResults \leftarrow query[\text{"}correctDocumentIds\text{"}]$
    $retrievedResults \leftarrow$ RETRIEVETOPDOCUMENTS($retrievalSystem$, $query[\text{"}input\text{"}]$)
    if INTERSECT($retrievedResults$, $expectedResults$) $\neq \emptyset$ **then**
      $successfulRetrievals \leftarrow successfulRetrievals + 1$
    end if
  end for
  $retrievalScore \leftarrow successfulRetrievals/totalQueries$
  return $retrievalScore$
end function

Quality Score: Evaluates the relevance, coherence, and informativeness of the responses generated by the system. This metric ensures that the answers not only match the user's informational needs but are also clear and well-structured.

```
# Evaluate responses
evaluation_system_content = """
    Your job is to rate the quality of our generated answer {generated_answer}
    given a query {query} and a reference answer {reference_answer}.
    Your score has to be between 1 and 5.
    You must return your response in a line with only the score.
    Do not return answers in any other format.
    On a separate line provide your reasoning for the score as well.
    """
evaluate_responses(
    experiment_name=experiment_name,
    evaluator=evaluator,
    temperature=0.0,
    max_context_length=MAX_CONTEXT_LENGTHS[evaluator],
    system_content=evaluation_system_content,
    assistant_content="",
    experiments_dir=experiments_dir,
    references_fp=references_fp,
    responses_fp=str(Path(experiments_dir, "responses", f"{experiment_name}.json")),
    num_samples = num_samples)
```

Figure 6: Assessing Quality score

Empirical Results

1. Evaluation Metrics and Scoring

The system's performance was evaluated using two main metrics:

Retrieval Score: Measured on a scale from 0 (poor) to 1 (ideal), this score assesses how effectively the system can find and retrieve relevant information in response to a user's query.

Quality Score: Rated from 1 (poor) to 5 (ideal), this score evaluates the relevance, coherence, and contextual appropriateness of the responses generated by the RAG system.

2. Results Overview

The empirical testing focused on various configurations and their impact on the Retrieval and Quality Scores. The findings include:

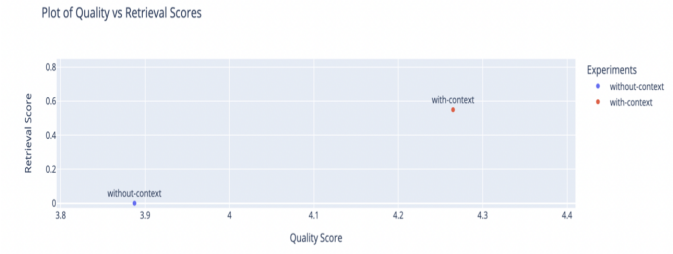

Figure 7: Retrieval and Quality scores With and Without Context

	Without Context	With Context
Retrieval Score	0.0	0.54
Quality Score	3.88	**4.26**

Table 1: Comparison of Retrieval and Quality Scores Without and With Context

Figure 8: Retrieval and Quality scores by varying chunk sizes

Chunk-Size	Retrieval Score	Quality Score
300	**0.59**	4.13
500	0.58	**4.31**
750	0.58	4.21
1000	0.54	4.25

Table 2: Comparison of Retrieval and Quality Scores by varying Chunk-Sizes

Figure 9: Retrieval and Quality scores by varying number of chunks

No. of chunks	Retrieval Score	Quality Score
1	0.14	4.12
3	0.36	4.20
5	0.58	4.33
7	0.64	**4.38**
10	**0.70**	4.34

Table 3: Comparison of Retrieval and Quality Scores by varying number of chunks

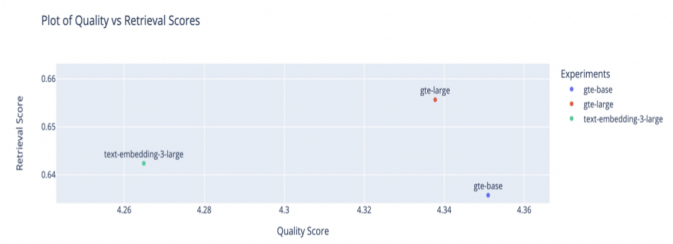

Figure 10: Retrieval and Quality scores using different Embedding Models

Embedding Models	Embedding Dimensions	Retrieval Score	Quality Score
thenlper/gte-base	768	0.63	**4.35**
thenlper/gte-large	1024	**0.65**	4.33
text-embedding-3-large	3072	0.64	4.26

Table 4: Comparison of Retrieval and Quality Scores using different Embedding Models

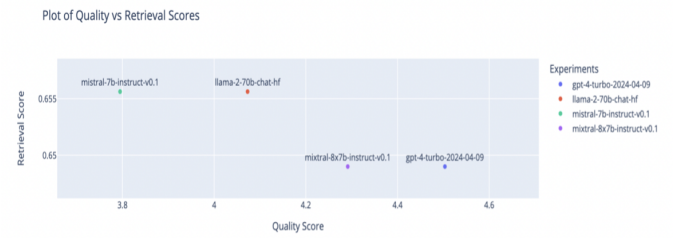

Figure 11: Retrieval and Quality scores using various generation Models

Generation Models	Retrieval Score	Quality Score
gpt-4-turbo-2024-04-09	0.64	**4.50**
llama-2-70b-chat-hf	0.65	4.07
mistral-7b-instruct-v0.1	**0.66**	3.79
mixtral-8x7b-instruct-v0.1	0.64	4.29

Table 5: Comparison of Retrieval and Quality Scores using various generation models

Contextual Relevance: Incorporating contextual information into the model significantly improved performance, raising the Quality Score from 3.88 to 4.26 and proving that this RAG system is indeed worth the effort.

Impact of Chunk Size: Testing different chunk sizes (300, 500, 750, 1000) showed that a medium chunk size (500) optimized both Retrieval and Quality Scores, achieving scores of 0.58 and 4.31, respectively.

Varying Number of Chunks: Increasing the number of chunks from 1 to 10 demonstrated a positive trend in both scores, with the Retrieval Score peaking at 0.70 and the Quality Score at 4.38 with 7 chunks.

3. Model Comparisons

Embedding Models: Different embedding models were tested including thenlper/gte-base and thenlper/gte-large. The gte-large model with 1024 dimensions performed the best, achieving a Retrieval Score of 0.65 and a Quality Score of 4.33.

Generation Models: Among the generation models tested, the proprietary model **"GPT-4-turbo-2024-04-09"** outperformed the others with a Quality Score of 4.50. On the other hand, the open source alternative model **"mixtral-8x7b-instruct-v0.1"** achieved a quality score of 4.29. This highlights the effectiveness of the latest language models in generating contextually relevant and coherent responses.

4. Summary of Findings

The results indicate that the effectiveness of the RAG system is highly dependent on the configuration of chunk sizes and the number of chunks processed. Additionally, the choice of embedding and generation models plays a critical role in maximizing the accuracy and relevance of the retrieved information and generated responses.

Broader implications

The Retrieval-Augmented Generation (RAG) system, developed for scikit-learn documentation, marks a leap forward in educational tools for machine learning. It makes complicated concepts simpler to increase learning interactions and shorten the learning curve. It also improves information access and provides contextually relevant responses. This enhances skill with machine learning technologies and facilitates efficient self-learning. Furthermore, by lowering entrance hurdles, improving accessibility of technical documentation, and facilitating the advancement of a diverse spectrum of individuals in machine learning and data science, the RAG system fosters inclusion in the tech industry.

By simplifying access to necessary documentation, the solution also increases productivity for data science specialists, freeing them up to concentrate more on innovative and useful applications. This results in better problem-solving, faster project turnaround times, and higher output in work

environments. There is a large market for similar technology, as seen by the RAG system's potential applications beyond scikit-learn, including software development, medical research, and legal services.

This initiative establishes a standard for upcoming AI-enhanced instructional materials in addition to advancing the domains of information retrieval and natural language processing. It opens the door for more interactive, responsive, and customized educational technologies by encouraging educators and developers to think creatively about how to present and interact with educational content. The created RAG system serves as an example of how incorporating cutting-edge AI can change professional and educational environments, promoting a more diverse and effective technological environment.

The scikit-learn documentation RAG system has the power to revolutionize user interaction in a variety of domains where retrieving information is essential. This development improves the usefulness and accessibility of technical knowledge by integrating AI technology in a substantial way.

Conclusion

The development of the RAG system significantly enhances the accessibility and usability of scikit-learn documentation by leveraging advanced information retrieval and natural language processing technologies. This integration, as evidenced by the findings presented in Table 6, demonstrates that contextual integration substantially improves both retrieval accuracy and quality scores, particularly when optimized for specific chunk sizes and numbers.

Moreover, our system effectively addresses the knowledge cut-off and reduces the risk of hallucination, as shown in Figures 12, 13, 14, and 15 below, where it outpaces existing models like GPT-4 in recognizing current features and support in scikit-learn. These advancements underscore the necessity for continued refinement and application of context-aware systems in technical domains, suggesting promising directions for future research to further improve the accuracy and reliability of automated documentation retrieval systems.

Future enhancements:

1. **Lexical Search Capabilities:** Incorporating lexical search capabilities along with semantic search might improve the system's comprehension and processing of user requests for certain type of queries.
2. **Re-ranking Mechanisms:** Re-ranking techniques could be used to improve the retrieval process and make sure that the most pertinent papers are shown first.
3. **Fine-Tuning Embeddings:** The system might be kept up to date by regularly updating and improving the embeddings to take into account the constantly changing vocabulary and ideas of machine learning.
4. **Periodic Updates:** Creating ETL pipelines for routine updates would guarantee that the system is always up to date with the newest versions of scikit-learn and documentation modifications.

Advice for future DS5983 students:

Students engaging in similar projects should focus on understanding the nuances of both the technology and the domain-specific requirements. Experimenting with different configurations and parameters, like chunk sizes and embedding models, can significantly influence the outcomes. Additionally, maintaining a clear focus on the end-user experience and how effectively the system meets their needs should guide the project's development and enhancements.

Parameter	Value
Chunk Size	500
No. of Chunks	7
Embedding Model	thenlper/gte-large
Generation Model	gpt-4-turbo-2024-04-09

Table 6: Optimal configuration as measured on the evaluation dataset

You
How to use root_mean_squared_error in scikit-learn?

ChatGPT
In scikit-learn, while there is no direct function named `root_mean_squared_error`, you can easily compute the Root Mean Squared Error (RMSE) using the `mean_squared_error` function from the `sklearn.metrics` module and taking the square root of the result. The `mean_squared_error` function allows you to calculate the Mean Squared Error (MSE) and by setting the `squared` parameter to `False`, it returns the square root of MSE, which is the RMSE.

Here is how you can calculate the RMSE in scikit-learn:

python Copy code

```python
from sklearn.metrics import mean_squared_error
import numpy as np

# Example actual and predicted values
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

# Calculate RMSE
rmse = mean_squared_error(y_true, y_pred, squared=False)
print("Root Mean Squared Error:", rmse)
```

Figure 12: Knowledge Cut-Off (GPT-4)

You
Does any of the class offers GPU support in scikit-learn?

ChatGPT
No, scikit-learn does not offer GPU support for its classes and functions. Scikit-learn primarily focuses on CPU-based computing and is designed to be simple and easy to use, with an emphasis on traditional machine learning algorithms which generally do not benefit much from GPU acceleration.

For tasks where GPU support could significantly improve performance, such as deep learning, other libraries like TensorFlow, PyTorch, and cuML are more suitable. These libraries are designed to leverage GPUs to speed up computation, especially for large-scale data processing and complex model training scenarios.

Figure 13: Hallucination(GPT-4)

Figure 14: Addressing Knowledge-Cut-Off

Figure 15: Addressing Hallucination

References

[1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Retrieved from http://arxiv.org/abs/2005.11401

[2] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... Wang, H. (2023). Retrieval-Augmented Generation for large language models: A survey. Retrieved from http://arxiv.org/abs/2312.10997

[3] Guu, K., Lee, K., Tung, Z., Pasupat, P., Chang, M.-W. (2020). REALM: Retrieval-Augmented Language Model pre-training. Retrieved from http://arxiv.org/abs/2002.08909

[4] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... Yih, W.-T. (2020). Dense passage retrieval for open-domain question answering. Retrieved from http://arxiv.org/abs/2004.04906

[5] Shahmirzadi, O., Lugowski, A., Younge, K. (2018). Text similarity in vector space models: A comparative study. Retrieved from http://arxiv.org/abs/1810.00664

[6] https://scikit-learn.org/stable/

[7] Magdy, W., Jones, G. J. F. (2010, July 19). PRES. Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval.doi:10.1145/1835449.1835551

[8] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. de las, ... Sayed, W. E. (2023). Mistral 7B. Retrieved from http://arxiv.org/abs/2310.06825

[9] Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... Sayed, W. E. (2024). Mixtral of Experts. Retrieved from http://arxiv.org/abs/2401.04088

[10] Muennighoff, N., Tazi, N., Magne, L., Reimers, N. (2023). MTEB: Massive Text Embedding Benchmark. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics. Presented at the Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, Dubrovnik, Croatia. doi:10.18653/v1/2023.eacl-main.148

[11] Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., Zhang, M. (2023). Towards general text embeddings with multi-stage contrastive learning. Retrieved from http://arxiv.org/abs/2308.03281

[12] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., ... Zoph, B. (2023). GPT-4 Technical Report. Retrieved from http://arxiv.org/abs/2303.08774

[13] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. Retrieved from http://arxiv.org/abs/2307.09288

[14] Xie, S., Vosoughi, S., Hassanpour, S. (2022). Interpretation Quality Score for Measuring the Quality of interpretability methods. Retrieved from http://arxiv.org/abs/2205.12254