DEPARTMENT OF COMPUTER SCIENCE

Information Retrieval - CS 429

Project Report:

TEXT SIMILARITY SEARCH SYSTEM USING NATURAL LANGUAGE PROCESSING

By:

SAI KARTHEEK GOLI (A20546631)

Under the Guidance

of

Mr. Jawahar Panchal

Professor CS 429

Department of Computer Science.

ABSTRACT

The project presents a comprehensive solution for efficiently identifying similar text documents based on user queries. Using advanced natural language processing (NLP) techniques, the system aims to provide accurate and relevant results to users seeking information from large text corpora. The project employs a combination of TF-IDF (Term Frequency-Inverse Document Frequency) vectorization and cosine similarity to compute the similarity between documents. TF-IDF, a fundamental technique in information retrieval, assigns weights to terms based on their frequency in a document and across the entire corpus. Cosine similarity measures the cosine of the angle between two vectors, providing a measure of similarity between documents. The architecture of the system consists of three main components: a Flask-based REST API server, an indexer module responsible for building and querying the TF-IDF index, and an optional web crawler for collecting text documents from web pages. The Flask server serves as the interface for users to submit queries and receive relevant document matches. Operationally, users interact with the system by sending POST requests to the /query endpoint of the Flask server, providing their query string and optionally specifying the number of top results desired. The system then processes the query, retrieves the most similar documents from the index, and returns them to the user.

OVERVIEW

The text similarity search system using natural language processing (NLP) offers a solution to the increasingly complex task of document retrieval and similarity analysis. In the era of information overload, where vast amounts of textual data are generated every second, the need for efficient methods to search, retrieve, and organize this information has become paramount. Traditional keyword-based search engines often fall short in providing relevant results, especially when dealing with queries or large text corpora. This project addresses these challenges by use of NLP techniques, specifically TF-IDF vectorization and cosine similarity, to deliver accurate and contextually relevant document matches.

Solution Outline:

The proposed solution revolves around the construction of a text similarity search engine that using TF-IDF vectorization and cosine similarity to measure the similarity between textual documents. Cosine similarity then calculates the cosine of the angle between two TF-IDF vectors, providing a measure of similarity between documents.[1] By combining these techniques, the system can accurately identify documents that are semantically similar to a given query, even in the absence of exact keyword matches.

$$cosine\_similarity(A, B) = \frac{A \cdot B}{\parallel A \parallel \times \parallel B \parallel}$$

Where:

- A and B are the TF-IDF vectors representing the two documents.
- $A \cdot B$ represents the dot product of the TF-IDF vectors A and B.
- $\|A\|$ and $\|B\|$ represent the Euclidean norms (or magnitudes) of the TF-IDF vectors A and B respectively.

[1] Goyal, Kanav, and Megha Sharma. "Comparative Analysis of Different Vectorizing Techniques for Document Similarity using Cosine Similarity." In *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*, pp. 1-5. IEEE, 2022.

LITERATURE

The quest for effective document similarity analysis and retrieval has been a cornerstone of information retrieval research for decades. The exponential growth of digital content and the proliferation of online information sources have underscored the importance of extensive techniques for extracting meaningful insights from textual data.[2] The seminal works and recent advancements in the fields of natural language processing (NLP), information retrieval (IR), and text mining, focusing on methodologies for document similarity assessment and their implications for real-world applications.

Vector Space Models

Vector space models (VSMs) represent documents and queries as vectors in a high-dimensional space, enabling efficient computation of similarity metrics. Salton and McGill's pioneering work in the 1970s introduced the term frequency-inverse document frequency (TF-IDF) weighting scheme, which remains a cornerstone of VSM-based retrieval systems[3]. TF-IDF assigns weights to terms based on their frequency in a document and across the entire corpus, effectively capturing the importance of terms in distinguishing documents[4]. While TF-IDF has demonstrated effectiveness in capturing document semantics and mitigating the effects of term frequency, it is limited in its ability to capture semantic relationships between terms and documents.

Probabilistic Models

Probabilistic models, such as the Okapi BM25 algorithm, offer an alternative approach to document retrieval by modeling the probabilistic relevance of documents to a given query.[5] BM25 incorporates term frequency, document length, and term saturation into its scoring function, allowing for more relevance assessments. Unlike TF-IDF, which assumes independence between terms, BM25 accounts for term dependencies and document length normalization, leading to improved retrieval performance in many cases. However, probabilistic models may suffer from computational complexity and parameter sensitivity, requiring careful tuning for optimal performance.

Deep Learning Techniques

Recent advancements in deep learning have revolutionized document similarity analysis, offering unprecedented capabilities in modeling complex semantic relationships. Neural network architectures, such as Siamese networks and transformer-based models like BERT (Bidirectional Encoder Representations from Transformers), have shown remarkable

[2] Padmanaban, Harish. "Revolutionizing Regulatory Reporting through AI/ML: Approaches for Enhanced Compliance and Efficiency." *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023* 2, no. 1 (2024): 57-69.

[3] Kim, Yoon, and Owen Zhang. "Credibility adjusted term frequency: A supervised term weighting scheme for sentiment analysis and text classification." *arXiv preprint arXiv:1405.3518* (2014).

[4] Kim and Zhang. Credibility adjusted term frequency. 2024

[5] Robertson, Stephen, and Hugo Zaragoza. "The probabilistic relevance framework: BM25 and beyond." *Foundations and Trends® in Information Retrieval* 3, no. 4 (2009): 333-389.

success in capturing contextual information and semantic similarities between documents.[6] By learning distributed representations of words and documents in an unsupervised manner, these models can generalize across domains and languages, offering a more extensive and scalable solution to document similarity analysis.
Real-World Applications

Document similarity analysis has diverse applications across various domains, including information retrieval, document clustering, recommendation systems, and plagiarism detection.[7] In information retrieval, similarity-based search engines enable users to find relevant documents based on query semantics rather than exact keyword matches, enhancing the user experience and facilitating knowledge discovery. Document clustering algorithms uses similarity measures to group related documents into clusters, enabling exploratory analysis and topic modeling[8]. Recommendation systems utilize document similarities to provide personalized content recommendations to users based on their preferences and browsing history. Finally, plagiarism detection systems rely on similarity analysis to identify instances of content reuse and intellectual property violations,[9] safeguarding academic integrity and copyright compliance.

Proposed System Overview

The proposed system is a document similarity analysis tool composed of three main components: a web crawler, an indexer, and a query processor. These components work together to crawl web documents, index them using TF-IDF representation, and allow users to query the indexed documents to find similar ones.

[6] Deepa, Ms D. "Bidirectional encoder representations from transformers (BERT) language model for sentiment analysis task." *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12, no. 7 (2021): 1708-1721.
[7] Han, Mengting, Xuan Zhang, Xin Yuan, Jiahao Jiang, Wei Yun, and Chen Gao. "A survey on the techniques, applications, and performance of short text semantic similarity." *Concurrency and Computation: Practice and Experience* 33, no. 5 (2021): e5971.
[8] Han *et al*. e5971
[9] Arabi, Hamed, and Mehdi Akbari. "Improving plagiarism detection in text document using hybrid weighted similarity." *Expert Systems with Applications* 207 (2022): 118034.

DESIGN - SYSTEM CAPABILITIES, INTERACTIONS, INTEGRATION

The design of the document similarity analysis system encompasses a range of capabilities, interactions, and integrations aimed at providing extensive and scalable solutions for real-world applications. In this section, outline the key components and design considerations that underpin the functionality and performance of the system.

System capabilities

1. Document Indexing Engine

The document indexing engine is responsible for processing and indexing large volumes of textual documents efficiently. Using techniques such as term frequency-inverse document frequency (TF-IDF) and cosine similarity, the engine generates vector representations of documents, enabling fast and accurate retrieval based on semantic similarity.[10] Additionally, the engine supports incremental indexing, allowing for real-time updates to the index as new documents are added or existing documents are modified.

2. Query Processing Module

The query processing module facilitates user interactions with the system by interpreting user queries and retrieving relevant documents from the index. Upon receiving a query, the module applies the same vectorization techniques used during indexing to transform the query into a vector representation compatible with the document space. By computing the cosine similarity between the query vector and indexed document vectors, the module identifies the most relevant documents matching the user's query.[11]

3. User Interface and Interaction

The Graphical user interface (UI) serves as the primary interaction point between users and the system, providing intuitive tools for querying, exploring, and display document similarities (UI). The UI features a search interface where users can enter queries and receive ranked lists of relevant documents based on similarity scores (http://127.0.0.1:5000/query confirms connection). Additionally, the UI supports interactive visualization of document clusters and similarity networks, enabling users to gain insights into the underlying structure of the document corpus (port http://127.0.0.1:5000/ to confirm app.py successful execution).

[10] Lahitani, Alfirna Rizqi, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. "Cosine similarity to determine similarity measure: Study case in online essay assessment." In *2016 4th International conference on cyber and IT service management*, pp. 1-6. IEEE, 2016.

[11] Bayatmakou, Farnoush, Azadeh Mohebi, and Abbas Ahmadi. "An interactive query-based approach for summarizing scientific documents." *Information Discovery and Delivery* 50, no. 2 (2022): 176-191.

ARCHITECTURE

System Architecture:

The architecture of the document similarity analysis system is designed to be modular, scalable, and flexible, accommodating various software components, interfaces, and implementation strategies. This section provides an overview of the system architecture, highlighting its key components, interfaces, and implementation considerations.

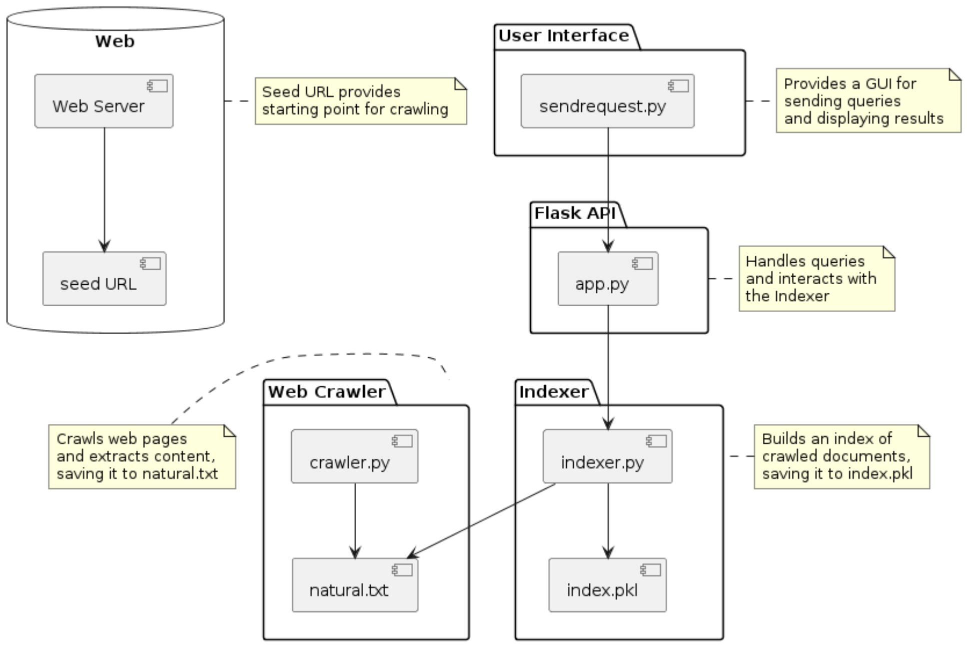

Web
Web Server
seed URL
Seed URL provides starting point for crawling
User Interface
sendrequest.py
Provides a GUI for sending queries and displaying results
Flask API
app.py
Handles queries and interacts with the Indexer
Web Crawler
crawler.py
natural.txt
Crawls web pages and extracts content, saving it to natural.txt
Indexer
indexer.py
index.pkl
Builds an index of crawled documents, saving it to index.pkl

Figure 1 Overall system architecture

Software Components

1. Document Indexing Engine: At the core of the architecture is the document indexing engine, responsible for processing and indexing textual documents. This component utilizes advanced natural language processing (NLP) techniques to transform raw text data into structured representations suitable for similarity analysis.
2. Query Processing Module: The query processing module interprets user queries and retrieves relevant documents from the index. It applies vectorization techniques to convert user queries into vector representations compatible with the document space, enabling efficient similarity computation.
3. User Interface (UI): The UI component serves as the primary interaction point between users and the system. It provides intuitive tools for querying, exploring, and showing document similarities, enhancing user experience and usability.
4. External Data Integration: This component facilitates integration with external data sources and systems, allowing data exchange and interoperability.

Interfaces

1. API Endpoints: The system exposes RESTful API endpoints for programmatic access to its functionalities. These endpoints enable developers to interact with the system programmatically, performing tasks such as querying for document similarities and retrieving indexed documents.
2. User Interface: The UI component provides a graphical interface for users to interact with the system. It features search interfaces, and interactive components for querying, exploring, and analyzing document similarities.
3. Data Ingestion Interfaces: The system supports various data ingestion interfaces for importing textual data from external sources. web scraping utilities, allowing users to ingest data from sources.

Implementation

1. Technology Stack: The system is implemented using a combination of programming languages, frameworks, and libraries. Python serves as the primary programming language, using libraries such as scikit-learn for NLP tasks and Flask for building RESTful APIs.
2. Scalability and Performance: Implementation considerations include scalability and performance optimizations to handle large volumes of textual data efficiently. Techniques such as distributed computing, parallel processing, and caching are employed to ensure optimal system performance under varying workloads.

OPERATION - SOFTWARE COMMANDS, INPUTS, INSTALLATION

Component	Operation Summary	Software Commands	Inputs	Ports	Installation
app.py	Flask application for handling free text queries	python app.py	None	5000	Flask, scikit-learn (pip install flask scikit-learn)
sendrequest.py	UI for sending requests to the Flask API	python sendrequest.py	Query text	N/A	requests (pip install requests)
crawler.py	Scrapy-based web crawler for downloading web documents	python crawler.py	Seed URL, Max Pages, Max Depth	N/A	Scrapy (pip install scrapy)
indexer.py	Indexer for constructing an inverted index	python indexer.py	Text documents (from crawler)	N/A	scikit-learn (pip install scikit-learn)

Operation:

1. Running Scripts: Execute the respective Python scripts (app.py, sendrequest.py, crawler.py, indexer.py) using Python interpreter. App.py must run before sendrequest.py.
2. Usage: Follow the command-line prompts for crawler.py. For app.py and sendrequest.py, interact via UI or send requests programmatically.
3. Output: Results will be displayed in the terminal for crawler.py, and in the UI window for sendrequest.py.

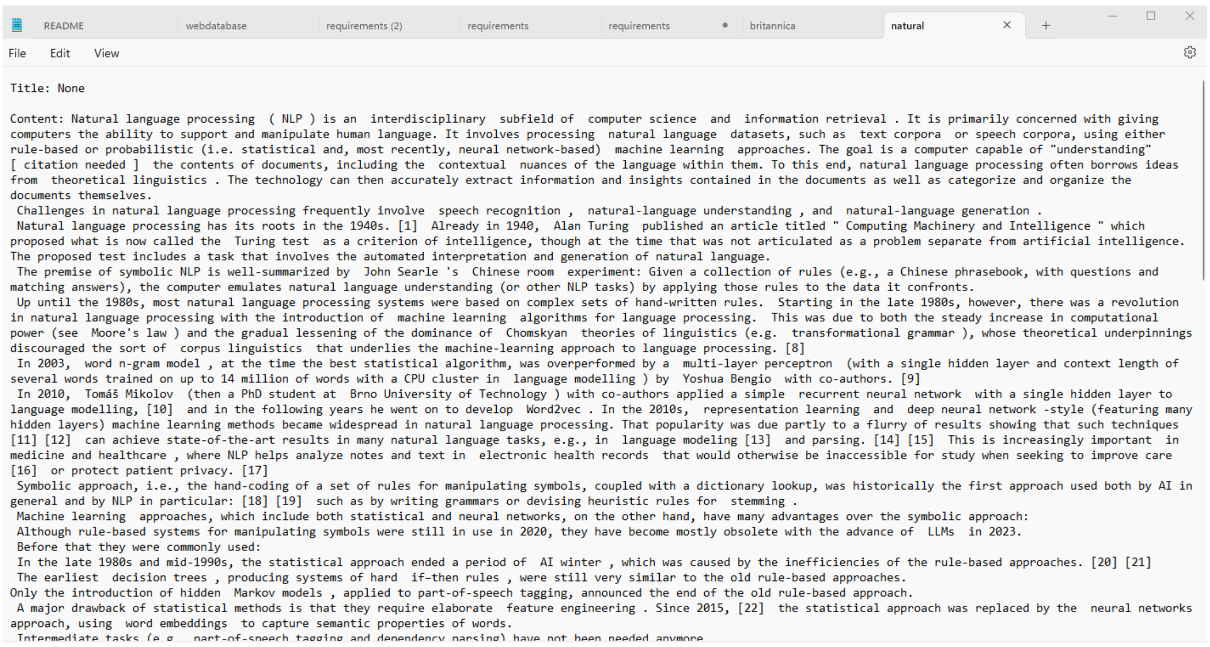
CONCLUSION

The document similarity analysis system demonstrates success in providing a extensive solution for users to perform efficient content-based searches. Through the integration of web crawling, indexing, and query processing components, the system enables users to input queries and retrieve relevant documents accurately. The user interface facilitates interaction with the system, allowing for intuitive query input and visualization of search results. Outputs from the system include top-K ranked documents based on similarity scores, presented in a clear and understandable format. However, caution should be exercised regarding the scalability of the system, particularly with large text corpora, as it may impact performance and resource utilization. Additionally, proper error handling and input validation should be implemented to ensure system stability and security, particularly when handling user-generated queries.

DATA SOURCES - LINKS, DOWNLOADS, ACCESS INFORMATION

Links: https://en.wikipedia.org/wiki/Natural_language_processing'

Downloads:

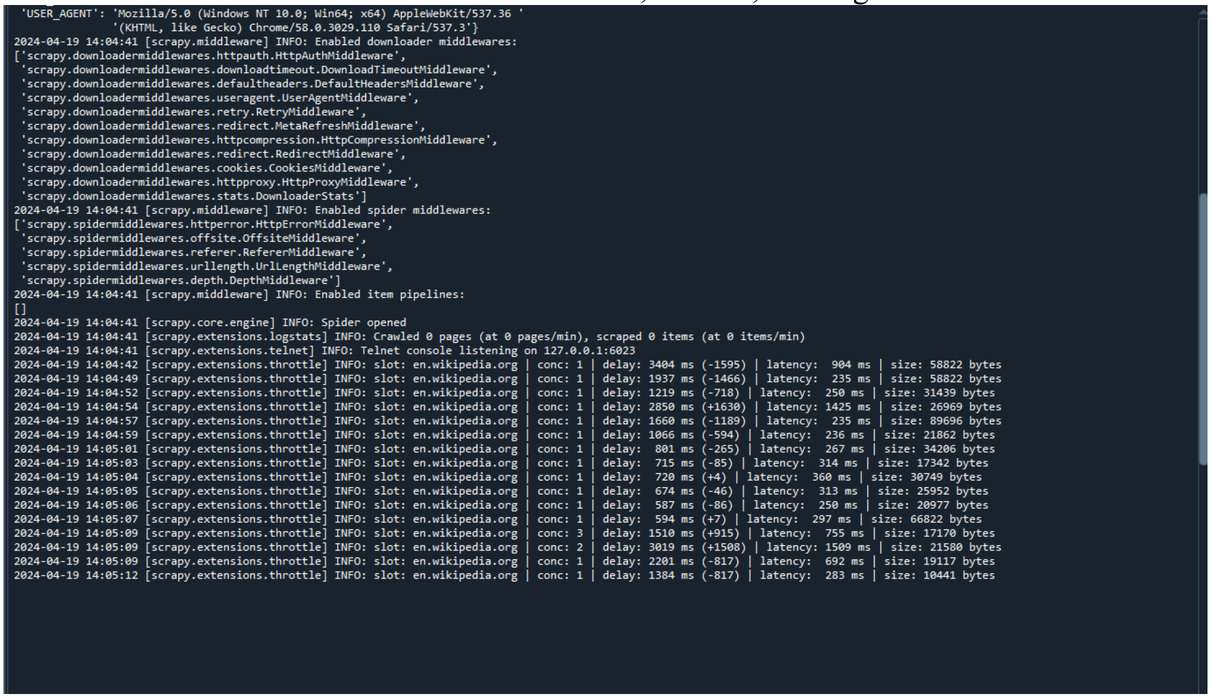


```
Title: None

Content: Natural language processing  ( NLP ) is an  interdisciplinary  subfield of  computer science  and  information retrieval . It is primarily concerned with giving
computers the ability to support and manipulate human language. It involves processing  natural language  datasets, such as  text corpora  or speech corpora, using either
rule-based or probabilistic (i.e. statistical and, most recently, neural network-based)  machine learning  approaches. The goal is a computer capable of "understanding"
[ citation needed ]  the contents of documents, including the  contextual  nuances of the language within them. To this end, natural language processing often borrows ideas
from  theoretical linguistics . The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the
documents themselves.
 Challenges in natural language processing frequently involve  speech recognition ,  natural-language understanding , and  natural-language generation .
 Natural language processing has its roots in the 1940s. [1]  Already in 1940,  Alan Turing  published an article titled " Computing Machinery and Intelligence " which
proposed what is now called the  Turing test  as a criterion of intelligence, though at the time that was not articulated as a problem separate from artificial intelligence.
The proposed test includes a task that involves the automated interpretation and generation of natural language.
 The premise of symbolic NLP is well-summarized by  John Searle 's  Chinese room  experiment: Given a collection of rules (e.g., a Chinese phrasebook, with questions and
matching answers), the computer emulates natural language understanding (or other NLP tasks) by applying those rules to the data it confronts.
 Up until the 1980s, most natural language processing systems were based on complex sets of hand-written rules.  Starting in the late 1980s, however, there was a revolution
in natural language processing with the introduction of  machine learning  algorithms for language processing.  This was due to both the steady increase in computational
power (see  Moore's law ) and the gradual lessening of the dominance of  Chomskyan  theories of linguistics (e.g.  transformational grammar ), whose theoretical underpinnings
discouraged the sort of  corpus linguistics  that underlies the machine-learning approach to language processing. [8]
 In 2003,  word n-gram model , at the time the best statistical algorithm, was overperformed by a  multi-layer perceptron  (with a single hidden layer and context length of
several words trained on up to 14 million of words with a CPU cluster in  language modelling ) by  Yoshua Bengio  with co-authors. [9]
 In 2010,  Tomáš Mikolov  (then a PhD student at  Brno University of Technology ) with co-authors applied a simple  recurrent neural network  with a single hidden layer to
language modelling, [10]  and in the following years he went on to develop  Word2vec . In the 2010s,  representation learning  and  deep neural network -style (featuring many
hidden layers) machine learning methods became widespread in natural language processing. That popularity was due partly to a flurry of results showing that such techniques
[11] [12]  can achieve state-of-the-art results in many natural language tasks, e.g., in  language modeling [13]  and parsing. [14] [15]  This is increasingly important  in
medicine and healthcare , where NLP helps analyze notes and text in  electronic health records  that would otherwise be inaccessible for study when seeking to improve care
[16]  or protect patient privacy. [17]
 Symbolic approach, i.e., the hand-coding of a set of rules for manipulating symbols, coupled with a dictionary lookup, was historically the first approach used both by AI in
general and by NLP in particular: [18] [19]  such as by writing grammars or devising heuristic rules for  stemming .
 Machine learning  approaches, which include both statistical and neural networks, on the other hand, have many advantages over the symbolic approach:
 Although rule-based systems for manipulating symbols were still in use in 2020, they have become mostly obsolete with the advance of  LLMs  in 2023.
 Before that they were commonly used:
 In the late 1980s and mid-1990s, the statistical approach ended a period of  AI winter , which was caused by the inefficiencies of the rule-based approaches. [20] [21]
 The earliest  decision trees , producing systems of hard  if-then rules , were still very similar to the old rule-based approaches.
Only the introduction of hidden  Markov models , applied to part-of-speech tagging, announced the end of the old rule-based approach.
 A major drawback of statistical methods is that they require elaborate  feature engineering . Since 2015, [22]  the statistical approach was replaced by the  neural networks
approach, using  word embeddings  to capture semantic properties of words.
 Intermediate tasks (e.g.  part-of-speech tagging and dependency parsing) have not been needed anymore
```

Figure 2 Sample Natural language processing data crawled

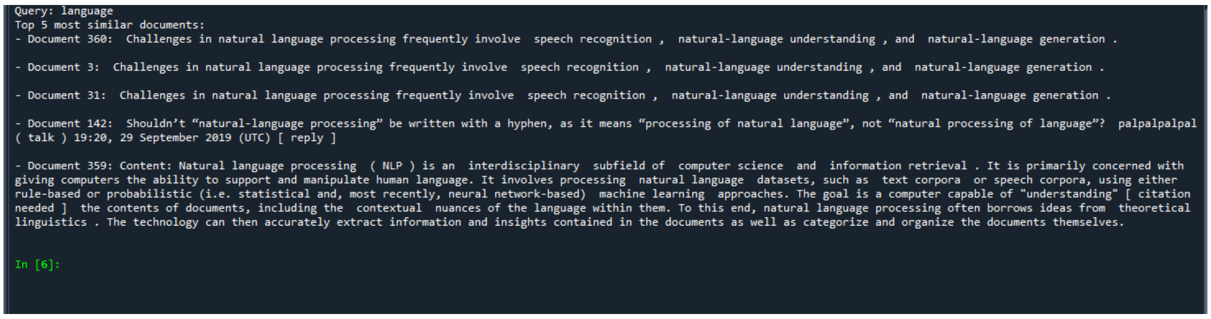
Access to the data sources is subject to the terms and conditions of the respective websites or data providers. Therefore, Users should ensure compliance with copyright and usage policies when accessing and utilizing web content for analysis.

Test Cases - Framework, harness, coverage



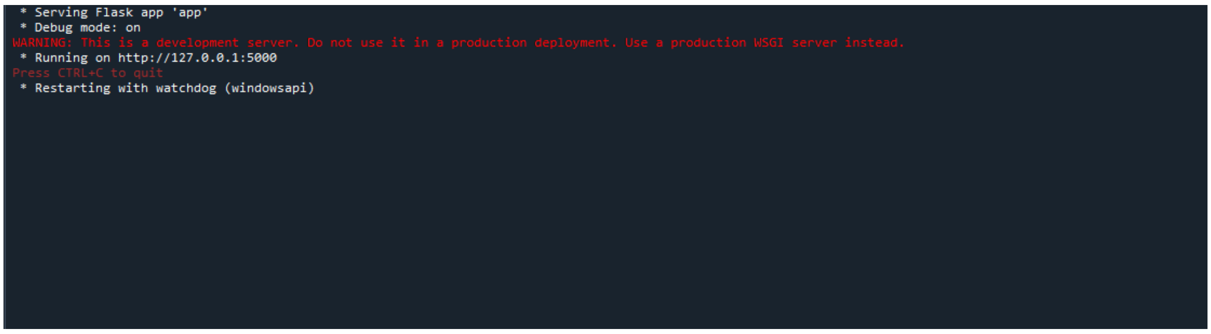
```
 'USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 '
               '(KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'}
2024-04-19 14:04:41 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2024-04-19 14:04:41 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2024-04-19 14:04:41 [scrapy.middleware] INFO: Enabled item pipelines:
[]
2024-04-19 14:04:41 [scrapy.core.engine] INFO: Spider opened
2024-04-19 14:04:41 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2024-04-19 14:04:41 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6023
2024-04-19 14:04:42 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 3404 ms (-1595) | latency:  904 ms | size: 58822 bytes
2024-04-19 14:04:49 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 1937 ms (-1466) | latency:  235 ms | size: 58822 bytes
2024-04-19 14:04:52 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 1219 ms (-718) | latency:  250 ms | size: 31439 bytes
2024-04-19 14:04:54 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 2850 ms (+1630) | latency: 1425 ms | size: 26969 bytes
2024-04-19 14:04:57 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 1660 ms (-1189) | latency:  235 ms | size: 89696 bytes
2024-04-19 14:04:59 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 1066 ms (-594) | latency:  236 ms | size: 21862 bytes
2024-04-19 14:05:01 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  801 ms (-265) | latency:  267 ms | size: 34206 bytes
2024-04-19 14:05:03 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  715 ms (-85) | latency:  314 ms | size: 17342 bytes
2024-04-19 14:05:04 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  720 ms (+4) | latency:  360 ms | size: 30749 bytes
2024-04-19 14:05:05 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  674 ms (-46) | latency:  313 ms | size: 25952 bytes
2024-04-19 14:05:06 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  587 ms (-86) | latency:  250 ms | size: 20977 bytes
2024-04-19 14:05:07 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay:  594 ms (+7) | latency:  297 ms | size: 66822 bytes
2024-04-19 14:05:09 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 3 | delay: 1510 ms (+915) | latency:  755 ms | size: 17170 bytes
2024-04-19 14:05:09 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 2 | delay: 3019 ms (+1508) | latency: 1509 ms | size: 21580 bytes
2024-04-19 14:05:09 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 2201 ms (-817) | latency:  692 ms | size: 19117 bytes
2024-04-19 14:05:12 [scrapy.extensions.throttle] INFO: slot: en.wikipedia.org | conc: 1 | delay: 1384 ms (-817) | latency:  283 ms | size: 10441 bytes
```

Figure 3 crawler.py in action



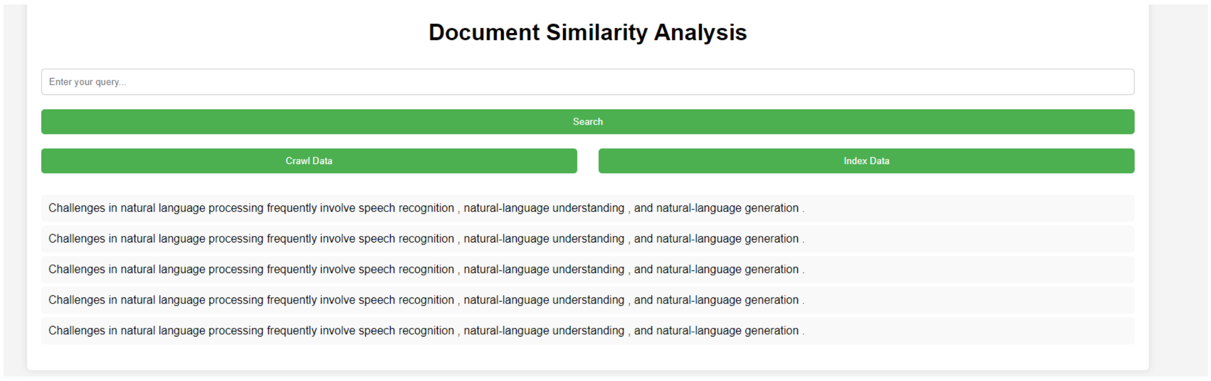
```
Query: language
Top 5 most similar documents:
- Document 360:  Challenges in natural language processing frequently involve  speech recognition ,  natural-language understanding , and  natural-language generation .

- Document 3:  Challenges in natural language processing frequently involve  speech recognition ,  natural-language understanding , and  natural-language generation .

- Document 31:  Challenges in natural language processing frequently involve  speech recognition ,  natural-language understanding , and  natural-language generation .

- Document 142:  Shouldn't "natural-language processing" be written with a hyphen, as it means "processing of natural language", not "natural processing of language"?  palpalpalpal
( talk ) 19:20, 29 September 2019 (UTC) [ reply ]

- Document 359: Content: Natural language processing  ( NLP ) is an  interdisciplinary  subfield of  computer science  and  information retrieval . It is primarily concerned with
giving computers the ability to support and manipulate human language. It involves processing  natural language  datasets, such as  text corpora  or speech corpora, using either
rule-based or probabilistic (i.e. statistical and, most recently, neural network-based)  machine learning  approaches. The goal is a computer capable of "understanding" [ citation
needed ]  the contents of documents, including the  contextual  nuances of the language within them. To this end, natural language processing often borrows ideas from  theoretical
linguistics . The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.


In [6]:
```

Figure 4 indexer.py in action



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
```

Figure 5 Flask app.py in action running on port http://127.0.0.1:5000



Document Similarity Analysis

Challenges in natural language processing frequently involve speech recognition , natural-language understanding , and natural-language generation .

Challenges in natural language processing frequently involve speech recognition , natural-language understanding , and natural-language generation .

Challenges in natural language processing frequently involve speech recognition , natural-language understanding , and natural-language generation .

Challenges in natural language processing frequently involve speech recognition , natural-language understanding , and natural-language generation .

Challenges in natural language processing frequently involve speech recognition , natural-language understanding , and natural-language generation .

Figure 6 send request.py in action retrieving the information, the searched information is in bracket, I.e language

Source Code - Listings, documentation, dependencies (open-source).

LINK TO GITHUB

Documentation

The four main modules:

1. Crawler Module: This module (crawler.py) is responsible for crawling web documents in HTML format. It utilizes Scrapy, a Python framework for web scraping, to extract content from web pages. The crawler is configurable with parameters such as the seed URL, maximum number of pages to crawl, and maximum depth of crawling.
2. Indexer Module: The indexer.py module implements a TF-IDF based indexing system using the scikit-learn library. It constructs an inverted index to represent documents in a vector space model. The indexer also provides functionality for saving and loading the index to/from a pickle file.
3. Flask Application: The Flask application (app.py) serves as the backend for handling free-text queries. It exposes a POST endpoint /query to receive queries in JSON format. Upon receiving a query, it utilizes the indexer to perform a similarity search and returns the top-K ranked results.
4. Send Request Module: This module (sendrequest.py) provides a UI interface for users to interact with the Flask application. It allows users to enter a query and sends a POST request to the Flask server. Upon receiving the response, it highlights the search word in the output content and displays the results.

Dependencies

The project relies on the following dependencies:

- Python 3.10+
- scikit-learn 1.2+
- Scrapy 2.11+
- Flask 2.2+
- Requests

These dependencies can be installed via pip using the following commands:

- pip install scikit-learn scrapy Flask requests or
- conda install scikit-learn scrapy Flask requests

Bibliography

Arabi, Hamed, and Mehdi Akbari. "Improving plagiarism detection in text document using hybrid weighted similarity." *Expert Systems with Applications* 207 (2022): 118034. https://doi.org/10.1016/j.eswa.2022.118034

Bayatmakou, Farnoush, Azadeh Mohebi, and Abbas Ahmadi. "An interactive query-based approach for summarizing scientific documents." *Information Discovery and Delivery* 50, no. 2 (2022): 176-191. https://doi.org/10.1108/IDD-10-2020-0124

Deepa, Ms D. "Bidirectional encoder representations from transformers (BERT) language model for sentiment analysis task." *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12, no. 7 (2021): 1708-1721. https://www.turcomat.org/index.php/turkbilmat/article/view/3055

Goyal, Kanav, and Megha Sharma. "Comparative Analysis of Different Vectorizing Techniques for Document Similarity using Cosine Similarity." In *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*, pp. 1-5. IEEE, 2022. https://doi.org/10.1109/ICATIECE56365.2022.10046766

Han, Mengting, Xuan Zhang, Xin Yuan, Jiahao Jiang, Wei Yun, and Chen Gao. "A survey on the techniques, applications, and performance of short text semantic similarity." *Concurrency and Computation: Practice and Experience* 33, no. 5 (2021): e5971. https://doi.org/10.1002/cpe.5971

Kim, Yoon, and Owen Zhang. "Credibility adjusted term frequency: A supervised term Weighting scheme for sentiment analysis and text classification." *arXiv preprint arXiv:1405.3518* (2014).

Lahitani, Alfirna Rizqi, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. "Cosine similarity to determine similarity measure: Study case in online essay assessment." In *2016 4th International conference on cyber and IT service management*, pp. 1-6. IEEE, 2016. https://doi.org/10.1109/CITSM.2016.7577578

Padmanaban, Harish. "Revolutionizing Regulatory Reporting through AI/ML: Approaches for Enhanced Compliance and Efficiency." *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023* 2, no. 1 (2024): 57-69. https://doi.org/10.60087/jaigs.v2i1.p69

Robertson, Stephen, and Hugo Zaragoza. "The probabilistic relevance framework: BM25 and beyond." *Foundations and Trends® in Information Retrieval* 3, no. 4 (2009): 333-389. http://dx.doi.org/10.1561/1500000019