Sage Plagarism Report

Powered by schoolworkspro.com

Hari Sharan Shrestha

150212

12 Results Found

17.26 % Match Percentage

Submitted to: Softwarica	**10.96%**
Submitted to: Softwarica	**1.98%**
Submitted to: Softwarica	**0.82%**
Submitted to: Softwarica	**0.62%**
Source: Static analysis Link: https://en.wikipedia.org/wiki/Seismic_analysis	**0.58%**
Submitted to: Softwarica	**0.58%**
Source: Confusion Matrix for Machine Learning Link: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/	**0.41%**
Submitted to: Softwarica	**0.37%**
Submitted to: Softwarica	**0.37%**
Submitted to: Softwarica	**0.29%**
Submitted to: Softwarica	**0.16%**
Submitted to: Softwarica	**0.12%**

REFLECTIVE REPORT
Search Engine for text retrieval
STW7071CEM – Information
Name: Hari sharan shrestha College ID: 150212

GitHub Repository Link: https://github.com/softwarica-github/Hari_sharan_shrestha_150212_stw7071cem/tree/master
YouTube Link: https://youtu.be/7dz8p7SwNOY

Contents Part 1 - Search engine.....3

Part 1 - Search engine

A crawler, also referred to as a web spider or bot in the context of search engines, is a computer program designed to automatically search for and index the content found on websites and other online sources.

Its primary function is to traverse the web by visiting the provided URLs, analyzing, and categorizing web pages.

During this process, the crawler extracts various types of information from the visited pages, including the page contents, links, metadata, and other relevant data.

To ensure compliance with website owners' preferences and guidelines, crawlers typically examine the robots.txt file present on web pages, which specify the rules and restrictions for bots accessing the site.

The indexed information collected by the crawler is subsequently made searchable within the search engine's index.

Crawlers frequently follow specific standards and rules to ensure effective and respectful crawling.

The analysis of the robots.txt file available on websites is an important feature.

This file specifies crawler rights and restrictions, specifying which portions of the site can be viewed and which should be avoided.

Crawlers who follow these standards preserve a pleasant relationship with website owners and reduce the risk of overloading servers with excessive queries.

Crawlers are essential to the operation of search engines.

Crawler data is handed on to the indexing system, where it is processed and organized.

This indexed data serves as the foundation for search engine functionality, allowing users to conduct searches and receive appropriate results depending on their queries.

Crawlers are also often intended to continuously scan and return previously crawled pages to identify updates, ensuring that the search engine's index is kept up to date and reflects the most recent content available on the web.

The purpose of the crawler script was to navigate and browse the web pages of Coventry University profiles.

It was provided with a base URL containing links to the profiles, and the crawler effectively crawled all 62 profiles.

During the crawling process, the script performed scraping operations to extract specific content from the obtained dataset.

The targeted information included the Publication Name, date, URL, and author for each article.

Overall, the crawler successfully collected 544 publications from the web pages.

The gathered data was then stored in a file named "scrapper_output_data.json," which can be seen in the accompanying screenshots.
In addition to the functionality, the crawler implemented a random few seconds pause between accessing each URL page.
This deliberate delay was incorporated to ensure a considerate approach towards Google's servers.
As a result, the entire crawling process, including the time delay, could be completed within a few minutes.
It's important to note that the crawler was operated manually, implying that human involvement was required to initiate the crawling of newly published articles.
This manual operation allowed for careful and selective crawling of specific content, ensuring that only the desired articles were included in the dataset.
Explanation of the crawler code: Step 1: Import required libraries
Step 2:
Step 3:
Step 4:
After the crawling and scraping process, the gathered data undergoes multiple transformations to create a consolidated list.
This list includes crucial details such as the article title, publication date, and author affiliated with Coventry University.
Subsequently, the data is formatted, resulting in a document that is ready for further processing.
To facilitate efficient retrieval of information, the transformed data is indexed using an inverted index approach.
This comprehensive indexing procedure ensures that the documents are organized and readily accessible.
In essence, the primary objective of the crawler is to establish a structured index for the collected data, streamlining its storage and retrieval processes.
Once the data has been converted to a list format, it can be further processed and indexed using an inverted index.
An inverted index is a data structure that is often used in search engines to store and retrieve information effectively.
It associates terms or keywords with the texts that contain them.
material becomes searchable by constructing an inverted index, allowing for speedy identification and access to relevant documents or information based on user queries.
It is critical to create an organized and searchable index for efficient information retrieval.
It allows users to search for certain articles or details within the collected dataset.
The indexed data can also be used for data analysis, trend spotting, and creating insights from the obtained information.
Some steps are implemented in this indexer part.
The outcome that has been extracted is contained within the output generated by the scraper, and in order to enhance the effectiveness and relevance of the search, the text is segmented into tokens, which are individual words.
These tokens are then subjected to further processing to return them to their base or root form.
To save time and computational resources, common stop-words such as "the," "and," "or," and "a" have been eliminated from the document.

By removing the stop-words from the query, unnecessary utilization of computer power is avoided, ensuring that the obtained results are more likely to align with the user's intended outcome.
Parsing and normalization
After the tokenization process is completed then the tokens are further processed by eliminating the punctuation, changing their case to lowercase, and performing additional normalization procedures to ensure uniform representation.
It is usual to encounter vast amounts of text data while scraping information from multiple sources, which must be processed and analyzed correctly.
To accomplish this, the scraped text is separated into tokens, which often represent individual words.
This tokenization stage facilitates text manipulation and analysis.
Following tokenization, another critical step is to further process the tokens in order to return them to their base forms.
This is referred to as stemming or lemmatization.
Variations of the same word (e.g., "running" and "runs") can be considered the same word by reducing them to their base or root forms, allowing for easier matching and retrieval of important information.
Index creation
One common step before indexing data is to tokenize the text by using the `split ()` function in Python.
Tokenization involves dividing the text into smaller units, such as words or meaningful entities, called tokens.
This process facilitates effective analysis and indexing of the text.
After tokenization, an inverted index is created or updated.
The inverted index is a data structure that maps terms or tokens to the documents where they appear.
It maintains references or pointers to the documents containing each token.
Additionally, the inverted index can contain extra information about each token within a document.
This information aids in ranking search results based on relevance.
For instance, the frequency of a term in a document indicates its importance or relevance to that document.
The index can also store the location of each term within a document, such as its position in a sentence or paragraph.
Including this supplementary information in the index enables search engines to determine the relevance and ranking of documents when responding to user queries.
Consequently, search engines can provide more precise and meaningful search results by assigning higher relevance scores to documents that contain the searched terms more frequently or prominently.
Explanation of the indexer code: Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:

Step 8:
Step 9:
Query processor
The query processor plays a crucial role in search engines, serving as essential software responsible for handling user queries and generating a comprehensive and highly relevant list of search results.
Its main objective is to comprehend the user's intention and locate the appropriate documents within the search engine's index.
Within this specific vertical search engine, the query processor goes beyond basic query capabilities by supporting Boolean queries, enabling users to employ operators like AND and OR to refine their searches.
This functionality empowers users to create more precise and specific queries, aiding in the retrieval of documents that meet their desired criteria.
To determine the ranking of indexed documents, the query processor employs an advanced ranking algorithm.
This algorithm takes various factors into consideration that contribute to a document's relevance to the user's query.
These factors include the frequency of occurrence of query terms within the document's content, the proximity of these terms to one another in the text, and the document's overall relevance when compared to other highly ranked documents obtained from related searches.
By analyzing these diverse factors, the query processor assigns a ranking to each document, reflecting its relevance and suitability for the user's search query.
This ranking system ensures that the most pertinent and valuable documents receive prominence in the search results, thereby enhancing the overall search experience and providing users with highly accurate and meaningful information.
In summary, the query processor is an integral component of search engines, responsible for processing user queries and generating relevant search results.
It understands user intent, supports Boolean queries, and uses a sophisticated ranking algorithm to determine the relevance of documents.
This ensures that the most appropriate and valuable documents are ranked highly and delivered to users, resulting in an enhanced search experience.
Examples of queries results are shown below:
The following example includes a critical spelling error to demonstrate that if a query is unrelated to a document, the application will not crash.
This example demonstrates the way query processors will execute if there is no matching term or if there is a matching phrase.
The below example explains what occurs when random capitalization is used in the query.
The authors and publication date are included in the papers in the inverted index; both terms can be used to find the desired result.
There are two parts to the query processor listed below:
The query processing that was done.
The user is given a search bar in which they can find the results of their inquiry.
The split () function tokenizes the query, then each phrase in the query is transformed to lowercase and all punctuation is deleted.
The same processing method occurs in the document to ensure the best possible possibilities of matching to the document searched.

The nltk library is used to remove the stopwords from the query that avoids the search engine looking for useless words to find the relevant words.
The ranking of search results based on the premise that the searched terms could be found in any document.
Cosine similarity is used to rank items in this search engine.
In text analysis, cosine similarity is a measure of similarity that is widely used to determine document similarity.
The formula for computing cosine similarity is.
Cosine similarity = (A.B) / (|A|.
Where A and B are vectors,
- A.B is the dot product of A and B.
It is calculated as the sum of element wise products of A and B.
- |A|.|B| is the L2 norm of A and B.
It is calculated as the square root of the sum of squares of elements.
Cosine similarity is determined in the Python code by importing cosine_similarity from the sklearn.metrics.pairwise module.
Explanation of query processor:
Step 1:
Step 2:
Step 3:
Step 4:
Part 2 - Subject or Text classification
The training data was manually gathered from the internet and saved in a csv file.
Naive Bayes, a simple machine learning algorithm, was used.
The python library scikit-learn was used to do text classification.
The Bayes theorem underpins the Naive Bayes classification approach.
It is a fast, precise, and dependable method that performs well on large datasets.
The Naive Bayes algorithm is a machine learning technique widely utilized for text classification tasks due to its simplicity and effectiveness.
It is rooted in the fundamental concept of Bayes theorem, which plays a key role in probability theory.
Naive Bayes assumes the conditional independence of features (words or tokens in the case of text classification), leading to its "naive" designation.
This assumption simplifies probability calculations and enhances the algorithm's computational efficiency.
One notable advantage of Naive Bayes is its swift and efficient performance, particularly when dealing with extensive datasets.
Despite its straightforward approach, Naive Bayes demonstrates strong performance across various text classification scenarios.
It finds extensive applications in tasks like spam filtering, sentiment analysis, and document categorization.
The confusion matrix is used to assess the performance of this algorithm.
The model was tested using recall, precision, accuracy, and the F measure, i.e., F1 score.
Accuracy is a statistic that provides a proportion of correctly recognized classes based on all identifications made.
The recall metric yields a percentage that represents the number of true positives that were correctly detected.

The precision metric returns the percentage of correct positive identifications.
The F- measure is determined as follows: F1 Score = 2 * (Precision * Recall) / (Precision + Recall) Because the train and test files are in csv format, they are read with the panda read_csv function.
First, the stopwords in the train and test data with the header ABSTRACT are eliminated, followed by the special characters.
As illustrated in the code screenshot below, the data is divided into four categories: computer science, physics, mathematics, and statistics.
Using the sklearn.pipeline library, define the pipeline parameters as shown below.
To predict the test data from the trained data, compute the accuracy score and the f1 score, i.e., the f measure.
F1 score is a machine learning evaluation metric that measures a model's accuracy.
It combines the precision and recall scores of a model.
The accuracy metric computes how many times a model made a correct prediction across the entire dataset.
Explanation of text classification code: Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:
Step 9:
Step 10:
Step 11:
The crawler effectively crawled all 62 profiles.
During the crawling process, the script performed scraping operations to extract specific content from the obtained dataset.
The targeted information included the Publication Name, date, URL, and author for each article.
Overall, the crawler successfully collected 544 publications from the web pages.
In indexer step, the transformed data is indexed using an inverted index approach.
To test search engine, some queries were entered in search box to display result as shown in above screen shots.
For text classification, text data and train data downloaded from internet source to test and train data in text classification step.
Anon., 2023.
Naive Bayes Classification Tutorial using Scikit-learn.
[Online] Available at: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn [Accessed 2 6 2023].
sGril, A., 2023. web crawler.
[Online] Available at: https://www.techtarget.com/whatis/definition/crawler [Accessed 1 6 2023].