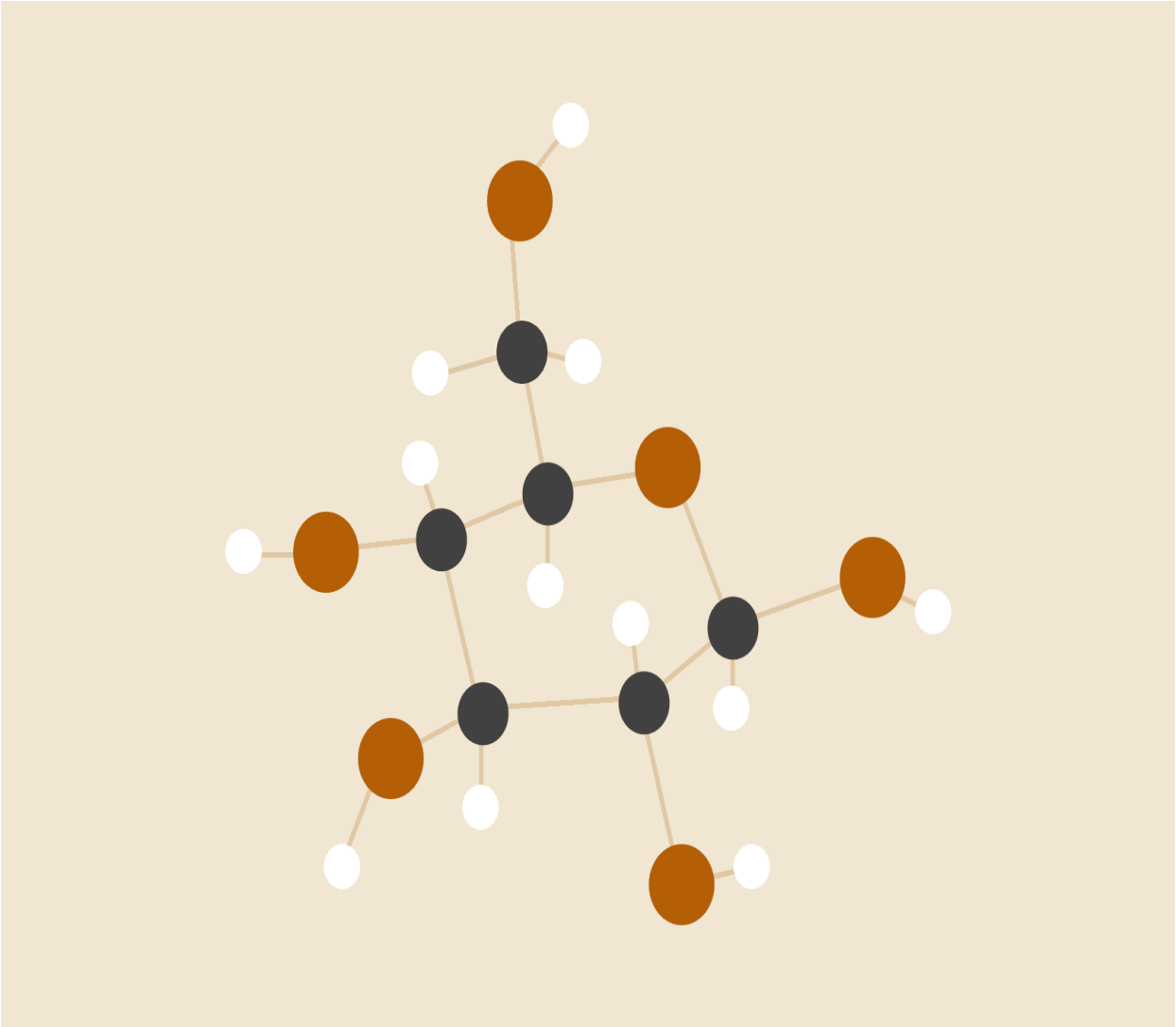
IR ASSIGNMENT REPORT



Mayank

09.02.2024
MT23045
M.tech CSE

Data Preprocessing

INTRODUCTION

Data preprocessing is a crucial step in the Information Retrieval (IR) pipeline. In this section, we discuss the steps taken to preprocess text data from a dataset of text files. The preprocessing tasks include lowercase conversion, tokenization, removal of stopwords and punctuation, and elimination of blank space tokens.

METHODOLOGY/APPROACH

1. Lowercasing the Text

Lowercasing the text involves converting all characters to lowercase. This is done to ensure uniformity and consistency in text representation.

2. Tokenization

Tokenization is the process of breaking down text into individual units, usually words or tokens. NLTK's `word_tokenize` function is employed for this task.

3. Removing Stopwords

Stopwords are common words that do not contribute much to the overall meaning of a document. The NLTK library provides a set of predefined stopwords for English, and these are removed from the text.

4. Removing Punctuation

Punctuation marks are often unnecessary for information retrieval tasks, so they are removed from the text.

5. Removing Blank Space Tokens

After preprocessing, some tokens may result in empty strings or spaces. These are removed to ensure a clean representation of the text.

ASSUMPTIONS

- The preprocessing steps are applied consistently to each document in the dataset.
- The dataset primarily consists of English text.
- Apostrophes are removed from words during the preprocessing to handle cases like "it's" becoming "its."

IMPLEMENTATIONS

The implementation was carried out in a Python environment, utilizing libraries such as NLTK for text processing. The steps are encapsulated in the `preprocess_text` function, which is applied to each file in the dataset.

The `preprocess_all_files` function is used to iterate through each file, perform preprocessing, and save the preprocessed content in a new file.

RESULTS

The results of the preprocessing are presented by showcasing the contents of five sample files before and after each operation. The content before preprocessing is displayed, followed by the content after preprocessing.

CONCLUSION

Data preprocessing is a foundational step in information retrieval. The implemented preprocessing steps contribute to the creation of a more efficient and accurate inverted index for subsequent retrieval tasks.

REFERENCES

- NLTK documentation
- Google Colab

Unigram Inverted Index and Boolean Queries

INTRODUCTION

In this section, we discuss the creation of a unigram inverted index from preprocessed text files and the implementation of boolean query operations. The tasks involve creating an inverted index, saving and loading it using Python's `pickle` module, and supporting boolean operations such as AND, OR, AND NOT, and OR NOT. Additionally, the system accommodates generalized queries.

METHODOLOGY/APPROACH

1. Unigram Inverted Index Creation

The unigram inverted index is created by processing each preprocessed text file. For each term in the document, a set of document identifiers is maintained in the inverted index. This is achieved through the `create_inverted_index` function.

2. Saving and Loading the Inverted Index

Python's `pickle` module is utilized to save and load the inverted index. This allows for efficient storage and retrieval of the index, promoting ease of use for subsequent tasks.

3. Boolean Query Execution

Boolean query operations (AND, OR, AND NOT, OR NOT) are implemented to support queries. The `execute_query` function processes the input query, extracts terms and operators, and performs the specified operations on the inverted index.

4. Generalized Queries

The system supports generalized queries in the form of a sequence of terms with corresponding logical operators. This flexibility allows users to construct complex queries, enhancing the system's versatility.

ASSUMPTIONS

- The inverted index is based on unigrams, considering each individual term as a separate unit.
- The boolean operations are case-insensitive for simplicity.
- Generalized queries follow a sequential pattern of terms and operators.

RESULTS

Unigram Inverted Index

The creation and loading of the unigram inverted index were successful. The initial entries of the loaded index were printed for verification, confirming the successful creation and storage of the inverted index.

Boolean Queries

The system successfully executed boolean queries according to the provided input. The results include the query text, the number of documents retrieved, and the names of the documents retrieved.

CONCLUSION

The implementation of a unigram inverted index and boolean query operations is a fundamental step in information retrieval systems. The created system demonstrates efficiency in handling various queries, providing users with the ability to retrieve relevant documents based on their requirements.

REFERENCES

- The use of `pickle` for efficient storage and retrieval.
- Google Colab for providing a convenient development environment.
- The dataset provider for making text files available for experimentation.

Positional Index and Phrase Queries

INTRODUCTION

In this section, we discuss the creation of a positional index from preprocessed text files and the implementation of phrase queries. The tasks include creating a positional index, saving and loading it using Python's `pickle` module, and supporting phrase queries. Additionally, the system performs preprocessing steps on input sequences and assumes the length of the input sequence to be less than or equal to 5.

METHODOLOGY/APPROACH

1. Positional Index Creation

The positional index is created by processing each preprocessed text file. For each term in the document, the index stores the document identifier along with the positions of the term in that document. This is achieved through the `create_positional_index` function.

2. Saving and Loading the Positional Index

Python's `pickle` module is utilized to save and load the positional index. This ensures efficient storage and retrieval of the index, allowing for seamless integration into subsequent tasks.

3. Phrase Query Execution

The system supports phrase queries by verifying the positional relationship between terms in documents. The `execute_positional_query` function processes the input query, extracts terms, and checks for the correct positional relationship between consecutive terms.

4. Preprocessing Steps

Input sequences undergo preprocessing steps similar to those in Q1, including lowercasing, tokenization, stopword removal, punctuation removal, and removal of blank

space tokens. This ensures consistency between the input queries and the preprocessed text files.

ASSUMPTIONS

- The length of the input sequence is assumed to be less than or equal to 5 for simplicity.
- Phrase queries are constructed with a sequential pattern of terms.

RESULTS

Positional Index

The creation and loading of the positional index were successful. The initial entries of the loaded index were printed for verification, confirming the successful creation and storage of the positional index.

Phrase Queries

The system successfully executed phrase queries according to the provided input. The results include the number of documents retrieved and the names of the documents retrieved for each phrase query.

CONCLUSION

The implementation of a positional index and support for phrase queries enhance the system's capability to handle more complex retrieval scenarios. The created system serves as a foundation for applications requiring precise term positioning in document retrieval.

REFERENCES

- The use of `pickle` for efficient storage and retrieval.
- Google Colab for providing a convenient development environment.
- The dataset provider for making text files available for experimentation.