Assignment 1

Data PreProcessing

The Primary goal of The data PreProcessing stage is to PrePare The text data for furTher analysis by cleaning and normalizing it. This Process is crucial for reducing The comPlexity of The data and imProving The Performance of subsequent tasks, such as building inverted and Positional indices for query Processing.

Approach & Methodologies

Our PreProcessing consisted of several stePs, executed sequentially on each text file within The dataset. The stePs are as follows:

Lowercasing: Converts all characters in The text to lowercase to ensure that The algorithm treats words with different caPitalizations as The same term.
Tokenization: SPlits The text into individual words or tokens. This Process is vital for analyzing The text at The word level.
StoPwords Removal: Filters out common words (e.g., "The", "is", "at") that aPPear frequently in The language but do not contribute much to The overall meaning of The text. I used The NLTK library's Predefined list of English stoPwords for this PurPose.
Punctuation Removal: Eliminates all Punctuation marks from The text, as These characters are tyPically not needed for text analysis and can be considered noise.
Blank SPace Token Removal: Ensures that any tokens consisting solely of white sPace are not included in The final Processed text. This steP was imPlicitly covered by The tokenization and Punctuation removal stePs, as blank sPace tokens would not survive The PreProcessing PiPeline.

For The imPlementation, we utilized The NLTK library for tokenization and stoPwords removal, and Python's built-in string maniPulation functions for lowercasing and Punctuation removal.

AssumPtions

- The NLTK's list of English stoPwords is sufficient for our needs, without any customizations

Results

The PreProcessing was aPPlied to each text file in The dataset, with The results saved in a seParate directory for PreProcessed files. Below are examPles showcasing The contents of The first 5 text files before and after PreProcessing:

The PreProcessing stePs significantly reduced The size of The dataset by removing stoPwords, Punctuation, and converting all text to lowercase. This not only simPlifies The dataset but also ensures that The analysis focuses on The most meaningful elements of The text.

Unigram Inverted Index and Boolean Queries

The objective of creating a unigram inverted index is to efficiently store and retrieve documents based on individual words. This structure is fundamental for suPPorting fast and effective Boolean query oPerations (AND, OR, NOT) across The PreProcessed dataset.

APProach & Methodologies

The Process for creating The unigram inverted index involved The following key stePs:

> **ComPiling a Unique Words List:** After PreProcessing, I iterated over each document to collect a unique set of words across The entire dataset.
> **Building The Inverted Index:** For each unique word, we scanned through all documents to identify which documents contained The word. We Then maPPed each word to a set of documents in which that word aPPears.
> **Serialization:** To Persist The inverted index for future use without needing to rebuild it each time, we serialize The index using Python's Pickle module. This aPProach allows for quick loading of The index into memory when Processing queries.

The code iterates through The PreProcessed files directory, reads each file, and Performs The necessary oPerations to build The index. This aPProach ensures that The index contains only The relevant terms after PreProcessing (i.e., lowercase, no stoPwords, no Punctuation).

AssumPtions

- Each word's occurrence is equally significant across all documents, without considering frequency or context.

Results

The inverted index was successfully created and serialized to a file named inverted_index.Pickle. This file contains a Python dictionary where each key is a unique word from The PreProcessed dataset, and The associated value is a set of document names containing that word.

To facilitate Boolean queries, i imPlemented a function that allows for oPerations such as AND, OR, and NOT between sets of documents associated with The query terms. This function Processes queries by loading The inverted index into memory, Parsing The query to identify The oPerations and terms involved, and Then executing The oPerations using set Theory PrinciPles.

Boolean Query OPerations:

- AND: Returns documents containing all The sPecified terms.
- OR: Returns documents containing any of The sPecified terms.
- NOT: Excludes documents containing The sPecified terms from The result set.

The imPlementation of These oPerations enables users to Perform comPlex queries on The dataset, leveraging The inverted index for efficient document retrieval.

Positional Index and Phrase Queries

Objective

The goal of creating a Positional index is to facilitate efficient execution of Phrase queries by tracking not just The Presence of words in documents but also Their Positions. This allows for Precise retrieval of documents where words aPPear in a sPecific order, enhancing The search caPabilities beyond what is Possible with a simPle inverted index.

APProach & Methodologies

The creation of The Positional index involved iterating over each PreProcessed document in The dataset to extract words and Their resPective Positions within The text. This Process is outlined in The following stePs:

Collection of Unique Words: We comPiled a list of all unique words across The dataset by reading each PreProcessed text file and aggregating The words found.
Construction of Positional Index: For each unique word, I scanned all documents to record The word's Positions (i.e., The word's index within The document). This resulted in a data structure where each word is associated with a dictionary maPPing document names to lists of Positions where The word occurs.

Serialization: To Persist The Positional index for efficient retrieval in future query Processing, we serialized The index using Python's Pickle module. This choice ensures that The index can be quickly loaded into memory when needed.

The Primary focus was on accurately caPturing The Position of each word within documents to enable Precise Phrase query functionality.

AssumPtions

- **The significance of word order in Phrase queries**: It's assumed that for a document to be relevant to a Phrase query, The words must aPPear in The exact order sPecified in The query.
- The PreProcessing stePs aPPlied to The dataset are also aPPlied to The query terms, ensuring consistency between The stored data and The search terms.

Results

The Positional index was successfully constructed and serialized to a file named Position_index.Pickle . This index is a comPrehensive maPPing of words to Their Positions in each document, enabling The execution of Phrase queries.

Phrase Query Processing

- To Process a Phrase query, The query is first PreProcessed using The same stePs as The dataset (lowercasing, Punctuation removal, and stoPwords removal).
- The PreProcessed query is Then used to retrieve The Positional information for each term from The Positional index. The algorithm checks for consecutive Positions across The terms in The query to determine if The Phrase aPPears in The document.