README

1. Unigram Inverted Index

Preprocessing steps

- Removed escape sequences from the text and replaced with space
- Converted to lower case: converted the text to lower case using the python lower() function
- Removed punctuations: remove the punctuations by removing the characters that are in python string.punctuation and replaced with space except apostrophe which was replaced by empty string
- Tokenization: tokenize the text using word_tokenize() from nltk
- Removed words with special characters
- Removed stopwords: remove the stopwords in nltk's stopword corpus for english
- Removed purely numeric tokens: remove the tokens that are numeric

Methodology

- Preprocessing as mentioned above
- Creating doc to doc-id mapping: assigning a document id to every document
- Creating the unigram inverted index: creating the postings lists for every term by storing a list of the documents that the term occurs in and the frequency of the number of such documents
- The input query is preprocessed following the same steps as used for the text data.
- From left to right the operations are processed in sequence except when all the operations are either OR or AND in which case we take the smallest two postings lists each time to merge.
- Query Processing:
    - OR: We simultaneously iterate over the two postings lists to be merged, comparing the doc ids. If both postings lists have the same doc id at their respective pointers, we add that doc id to the merged list and increment both pointers. If the pointers are at unequal doc ids, we add the lower doc id and increment the respective pointer. After iterating over the lists if any list has doc ids remaining in it, we add all the remaining doc ids to the merged list.
    - AND: We simultaneously iterate over the two postings lists to be merged, comparing the doc ids. If both postings lists have the same doc id at their respective pointers, we add that doc id to the merged list and increment both pointers. If the pointers are at unequal doc ids, we increment the respective pointer of the one that is at the lower doc id value.
    - OR NOT: We simultaneously iterate over the two postings lists to be merged, comparing the doc ids. We add all doc ids that are present in the first postings list and all the doc ids missing in the second postings list by iterating over the gaps in the doc id values of the second postings list. Then, we add all doc ids from the

    complete set of docs, with values beyond the doc ids in the two postings lists being merged.
  - AND NOT: We simultaneously iterate over the two postings lists to be merged, comparing the doc ids. If both postings lists have the same doc id at their respective pointers, we increment both the pointers. If the first pointer is at a lower doc id, we add its doc id to the merged list and increment the first pointer, if the second pointer is at a lower doc id, we just increment the second pointer.

Assumptions
- Numbers removed from text and query
- Stemming not done, can uncomment to add stemming
- Escape sequences removed from text
- Punctuations replaced with space except apostrophe replaced with empty string before tokenization
- Words with special characters removed from the text
- The input operation sequence is a comma+space separated string
- Operations are applied left to right except when all operations are OR or AND in which case the smallest lists are merged first to improve efficiency

2. Positional Index

Preprocessing steps
- Removed special characters
- Converted to lower case: convert the text to lower case using the python lower() function
- Removed punctuations: remove the punctuations by removing the characters that are in python string.punctuation
- Tokenization: tokenized the text using word_tokenize() from nltk
- Removed stopwords: remove the stopwords in nltk's stopword corpus for english
- Removed purely numeric tokens: remove the tokens that are numeric

Methodology
- Preprocessing as mentioned above
- Creating doc to doc-id mapping: assigning a document id to every document
- Creating the positional index: creating the nested postings lists for every term by storing index of each occurrence of a term in a document. Thus the postings lists are a dictionary which contains document ids as key. Each key (document_id) has a list of indexes which refer to the positional occurrence of the word in the corresponding document
- The input query is preprocessed following the same steps as used for the text data.
- Query Processing:
  - We iterate over each occurrence of the first term of the query in every document that it appears.

  - For each occurrence, we check if the following term of the query is present in same document at index current_occurance + 1. If we find a case that all the tokens in the query occur at consecutive indices in a document, we update the corresponding document key in our result list.
  - We print the length of our result list which represents the number of documents contains the phrase
  - By reverse mapping document ids to document names, we print the file names as well.

Assumptions

- Numbers not removed from text and query as not mentioned
- Stemming not done as not mentioned