# README

- It consists of implementation of *Inverted Index* and *Positional Query.*
- Used for searching through large document to find a word or a query that are at certain distance apart from each other.

***How to Run the Code?***

1. Extract the files of 20_newsgroup dataset.
2. Must have NLTK.
3. Open Jupyter Notebook and make sure that files are available in the same folder as jupyter notebook and then start running the code from the beginning.

***Assumptions:***

1. *Query words already present in files.*
2. *In Inverted Index, query word has no repetition.*
3. *All spellings are correct in the files.*
4. *The document id i.e doc_id of the word is same as it's file name.*
5. *All the words are normalized according to english dictionary, not according to mobile dictionary.*
6. *Since **not** is a unary operator, so **X not Y** is not a valid query instead **X and not Y** and **X or not Y** is valid.*
7. *In this program, first **not** query words is computed then **and** query words and at the end **or** will be computed.*

***Preprocessing Steps:***

1. Download stop_words and punkt using:
   *nltk.download('stopwords')*
   *nltk.download('puntk')*
2. *Remove header* from a file.
3. *Normalization:*
   - converting all text to the same case (upper or lower).
   - removing punctuation
   - removing digits
   - removing blank spaces.
   - removing single character

4. *Stop Words :*
   We may omit very common words such as **the, a, to, of, be** from unigram inverted index not from positional index.

5. *Lemmatization :*
   We may wish different forms of a root to match such **car, cars** turned as car only.
6. *Tokenization :*
   Cut character sequence into word tokens.

**Methodology for Inverted Index:**

- Inverted index is created using dictionary in python.
- The key of the dictionary is the word and and the value corresponding to that word contains the list of intergers which corresponds to the document id's.
- **Commands:**
  1. X **or** Y: In this scenario, there are two lists from the inverted index. Merge procedure is performed on the query which returns the list of relevant documents id's.
     Number of comparisons: O(m+n) where m is the length of X and n is the length of Y.
  2. X **and** Y: Similar to previous operation, here also merge procedure is used. But the difference is that it only return those document id's which are present in both X and Y respectively.
     Number of comparisons: O(m+n).
  3. X **and not** Y: While implementing these kind of queries, similar procedure is used but only those document containing X will be included where Y is not present.
     Number of comparisons: O(m+n)
  4. X **or not**Y: In this case, the documents containing Y query word is taken out and those document id's are removed from the list. Then from that filtered list OR is computed.
     Number of comparisons: list of the documents retrieved.
- **Optimization:**
  *The query with more than one operator will be computed according to operator precedence. Thus to reduce the number of comparisons, the documents are retrieved amongst same operator followed sorted order that is the word with minimum length will be processed first.*

**Methodology for Positional Indexing:**

- Positional index is created using nested dictionary in python i.e inner dictionary and outer dictionary.
- The key of outer dictionary is the token and value corresponding to that token is again a dictionary referred as inner dictionary having key as document id and value corresponding to that document id is the list of total occurrence of that token in a respective document.
- **Phrase query** are those queries where a string of words are given and the document containing those words are retrieved.
- For implementation of phrase query , common documents id's are retrieved for every pair of words.
- Then, for every common document id's, consecutive occurrence of those pair of words are taken into account and all such document id's are appended to seperatelist.
- Same procedure is followed until all words are exhausted in a query.
- Hence, the resulting list is as follow:
  [[list1],[list2],.....[listn]] where listk = document id's containing consecutive occurence of word(k-1) and word(k) in the query.
- Intersection of all such documents are found out and resulted as answer to the phrase query.