Proximity Searching

Information Retrieval

Project Report

Zawar Ahmed Tahir
DHA Suffa University
CS-151084

Maryium Awan
DHA Suffa University
CS-151068

Sadaf Khan
DHA Suffa University
CS-151065

Hafiz Junaid Khan
DHA Suffa University
CS-151094

I. Abstract:

Search Engines uses algorithms which produces too many matches for proposed query. In proximity search we only target those documents which fulfils the criteria of the searched query.

Proximity search engines uses different operators for different search requirements. Such as in [1] Proximity Search Engines uses (/p) operator to search in paragraph, next to word(/w) to search keywords which helps in producing more relevant searches to the users need.

According to recent study proximity search give more benefit if the raw data is extracted from sources where Meta data is available like XML files, JSON format files and etc.

In this project we are performing Proximity within sentence /s to search across XML documents for Keywords of query imposed by the users need. This method shall be used for search the documents and the result is ranked through cosine similarity and chain rule. The end result of the project is proximity within sentence. Ranking methodology used is cosine ranking. Top 10 documents will be displayed to the user.

Keywords:

Commercial Search Engines, Proximity, Query, Relevant, Operator, Rank, Custom search XML.

II. Table of Content:

III. Definition:

Before we move any further in describing our working, following are some key terminologies that might be useful to know:

Proximity Search:

Proximity searching is a way to search for two or more words that occur within a certain number of words from each other. The proximity operators are composed of a letter (N or W) and a number (to specify the number of words), Proximity search also allows generic operators which helps searching within paragraphs or sentences.

Ranking:

Ranking criteria are phrased in terms of relevance of documents with respect to an information need expressed in the query.

XML:

A metalanguage which allows users to define their own customized markup languages, especially in order to display documents on the Internet. XML Meta data characteristics helps is data retrieval within the document.

Cosine Similarity:

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter.

Dictionary:

Data structure we have used to create posting list and positional index list is Dictionary. It's a very helpful data structure to get data from the long lists by only giving key value without iterating all the way to the end of list.

IV. Introduction:

Search Engines like Google, Bing and Yahoo are the most essential elements for the users to search queries of their interests but when they see huge number of documents related to their query they get easily confuse. To overcome this problem we proposed an idea which is to facilitate the user with such documents which are most relevant to the user's query by using proximity in documents.

Proximity search is successfully used in information retrieval (IR) systems to locate documents that have Query words occurring within sentences /s, paragraph /p and words /w [2] Proximity Search looks for documents where two or more separately matching term occurrences are within a specified distance, where distance is the number of intermediate words or characters.

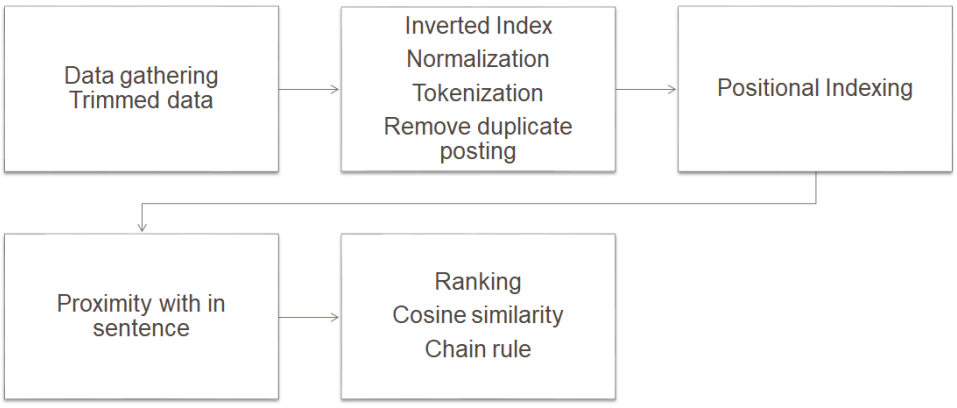
In our project we have achieved proximity search within sentence. We have declared our own operator (/s) which user can use within query to search query within document corpus. Initially we removed stop words from the postings and positional list but after realizing the need to fill missing

position of tokens in positional indexes we decided to also use stop words to complete the positional indexes sequence. These additional indexes shall be used in the future development of the project.

V. Literature Review:

We have studied many articles and research paper but the one which suits the most is **"Keyword Proximity search in XML Trees,"** in order to identify the most suitable algorithms which will help us with our project. The recent work which will have done on this research paper is keyword proximity search in querying XML documents in addition to text document. In this paper we have studied about XML trees and how XML keyword proximity queries to return the set of minimum connecting trees (MCTs) of the matches to the individual keywords in the query. The efficient algorithms to compute MCTs works in two cases: (i) when the XML data has been pre-processed and relevant indices have been constructed, and (ii) when the XML data has not been pre-processed, i.e., the XML data can only be processed sequentially. The algorithm is efficient because it only give MCTs which explains how keyword are connected and identifying only MCTs whose root is not an ancestor of the root of another MCT.

VI. Work Flow :



VII. Discussion:

Data acquiring:

To start with we acquired data from ACL dataset. We downloaded zip folders using python script. Zip files includes research paper XML files which have tags within them which make data extraction easy from the file. We generated a .txt file which has all the files path which our program will use to extract XML data from folders.

Data parsing:

We have downloaded approximately 22000 files using the above mentioned script. We have used body tag data from XML file. Each body tag data within document is appended to make a single document file. For our project each research paper is single document.

Normalization:

After extracting body tag data we have tokenize the text and cleaned it using data normalization techniques. Initially we removes stop words from the lists but in later stages of the project we realized the importance of stop words and we also indexed them. We also tried nltk normalization but it was too slow to be implemented in semester project. Another problem was that is normalized data which we don't want to get normalized.

Posting list creation:

After normalization we have used dictionary data structure of python to create list. Key value pair feature helped us in later stages of project in retrieving data from list. Each token is used as key for the dictionary and all the documents in which the token is present are appended in a list and are assigned to the token key.

```
attend ['A/A00/A00-4000-parscit.130908.xml', 'A/A88/A88-1001-parscit.130908.xml']
volunteer ['A/A00/A00-4000-parscit.130908.xml']
everyone ['A/A00/A00-4000-parscit.130908.xml', 'A/A92/A92-1023-parscit.130908.xml']
```

Here attend is a key and the document list appended next to the key.

After creating posting list we have stored then as object using pickle library in python. This retains the original data structure of the list on the retrieval of data.

Positional Indexing:

Positional indexing is also done after normalization of data. It is also based on dictionary data structure. The main difference between posting list and positional index list is that is consist of two dimensional dictionary. In it each token has multiple document list and each document has position of token within the document.

```
natural
    A/A00/A00-1001-parscit.130908.xml: [4, 63, 89, 97, 157, 177, 410, 424, 503, 547, 921, 1202, 1400, 1420, 1647,
    A/A00/A00-1002-parscit.130908.xml: [943]
    A/A00/A00-1003-parscit.130908.xml: [297]
    A/A00/A00-1005-parscit.130908.xml: [23, 562, 588, 610, 639, 652, 736, 927, 1097, 1374]
```

Here natural has multiple document as child in which have all the positions of token present in the document.

Saving indexes:

After creating indexes we have stored them as object using pickle library in python. This retains the original data structure of the list on the retrieval of data. These files cannot be opened on normal ides or notepads as they have different format to open with.

pickle.dump(positional_list, open("**postingPickles/positionListObjectV3.pickle**", "**wb**"))

We also generated sentence list for the documents to facilitate proximity search within sentences. After storing all the indexes we then started to process queries. These queries shall contain /s operator to identify tokens in query within same sentence.

Query processing:

User shall enter query. Language /s NLP. First the program will split the query string by "/s". After getting each token in the list tokens are matched in the sentence if they are present in list. If they are present than append name to the dictionary. After getting list for each token find the intersection of each document list set of the tokens.

After getting intersected document we are going to check if the tokens lie within sentence or not. This is done using positions of full stop in positional index. Checks if position of term of query lies within the indexes of full stop or not. If yes then take intersection of the document set of all terms in query.

Similarly proximity search using distance operator can also be done using "~d~" operator where 'd' can be any integer value. For distance approximation positional indexes are used to get distance within terms. If the condition is fulfilled then intersection of the document set of all terms in query is taken.

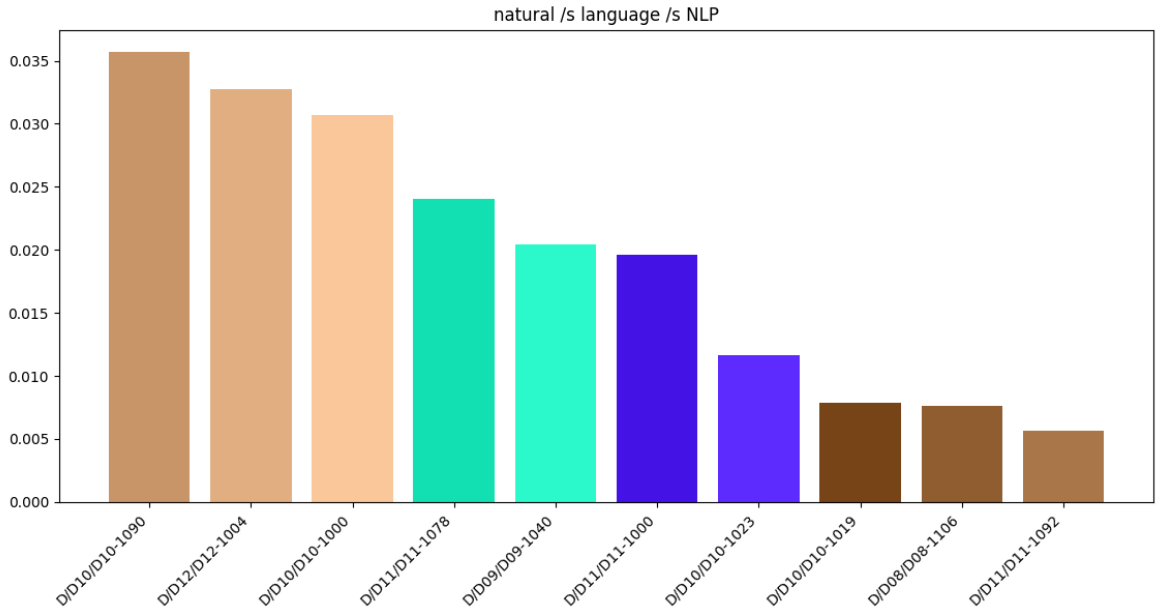
After finding docs we used cosine similarity.

Ranking:

We have ranked our output using cosine similarity and checked each document with others. Document having highest score are shown on top and the rest are bellow it.

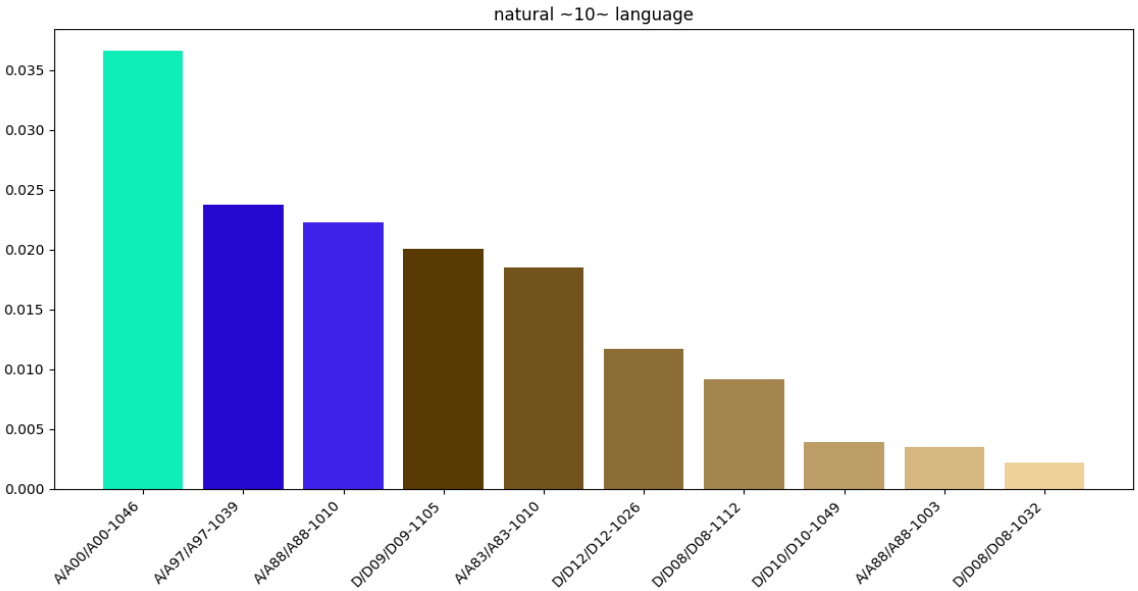
Proximity search using within sentence operator:

```
Enter Query: information /s retrieval /s system
Searched Query: 'information /s retrieval /s system' found!

0. D/D10/D10-1090-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0356841709637
1. D/D12/D12-1004-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0327193232472
2. D/D10/D10-1000-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0306602279583
3. D/D11/D11-1078-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0240735516494
4. D/D09/D09-1040-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0203909826658
5. D/D11/D11-1000-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.019649550276
6. D/D10/D10-1023-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.0116700653228
7. D/D10/D10-1019-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.00788168486302
8. D/D08/D08-1106-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.00758607863782
9. D/D11/D11-1092-parscit.130908.xml, Total hits in document: 1 ,  Cosine Similarity:  0.00563316256421
```



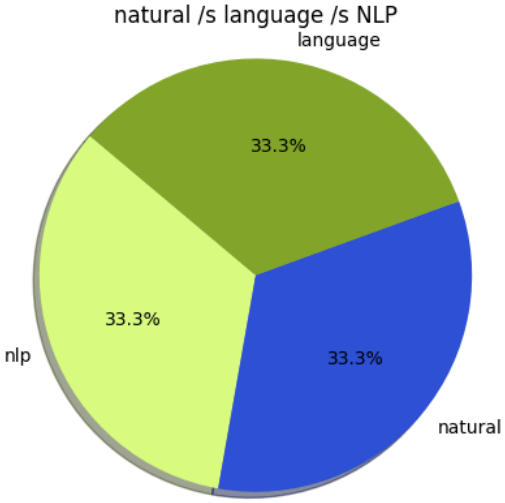
Proximity using distance operator.

```
Enter Query: natural ~10~ language
Searched Query: 'natural ~10~ language' found!

0. A/A00/A00-1046-parscit.130908.xml, Total hits in document:  --  0.0366125255004
1. A/A97/A97-1039-parscit.130908.xml, Total hits in document:  --  0.0237678114247
2. A/A88/A88-1010-parscit.130908.xml, Total hits in document:  --  0.0222284313415
3. D/D09/D09-1105-parscit.130908.xml, Total hits in document:  --  0.0200829463194
4. A/A83/A83-1010-parscit.130908.xml, Total hits in document:  --  0.0185080463499
5. D/D12/D12-1026-parscit.130908.xml, Total hits in document:  --  0.011712946562
6. D/D08/D08-1112-parscit.130908.xml, Total hits in document:  --  0.00915944642628
7. D/D10/D10-1049-parscit.130908.xml, Total hits in document:  --  0.00392477064096
8. A/A88/A88-1003-parscit.130908.xml, Total hits in document:  --  0.00349818198635
9. D/D08/D08-1032-parscit.130908.xml, Total hits in document:  --  0.00213853390596
```



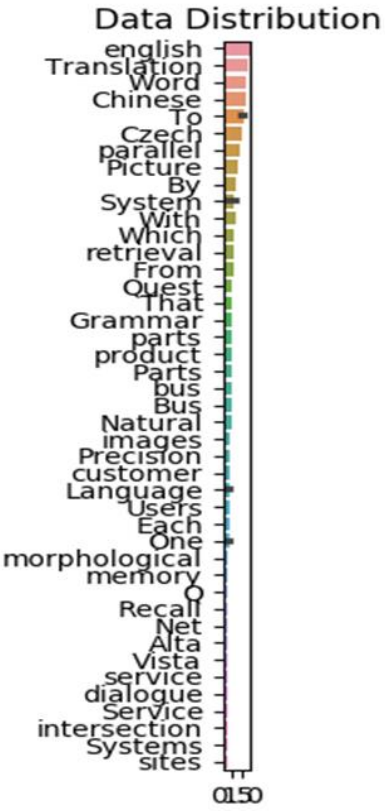
Chain Rule:

We applied chain rule on the query to figure out the importance of term in query within itself.



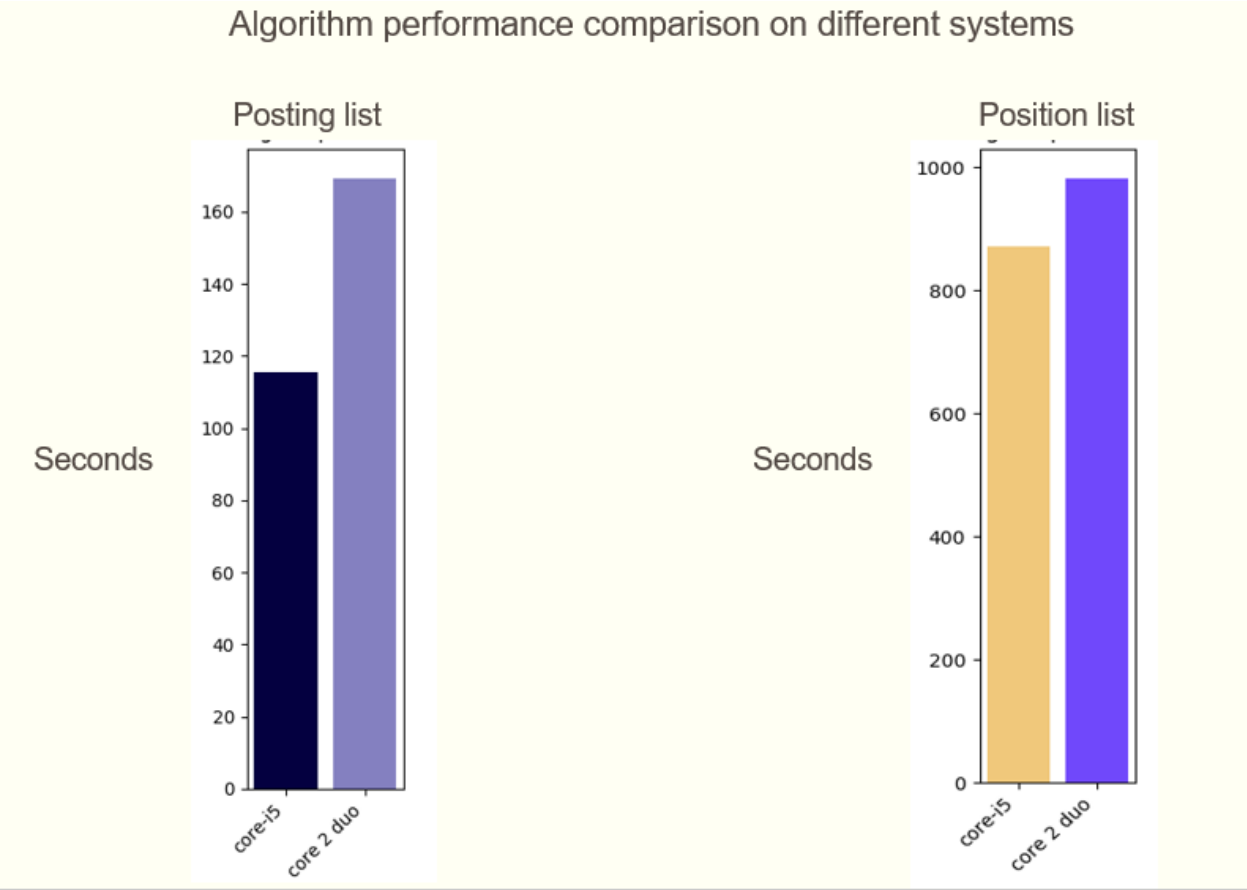
TFIDF:

Initially we calculated of tfidf to find out presence score of the tokens in the documents. Tfidf are used within cosine similarity ranking.



Testing:

We ran our algorithm on two different systems, one was hp envy series core-i5 third gen laptop with 8GB RAM. On the other we hat hp Core 2 Duo with 4 GB of RAM.



Here we can see that systems with up to date or new specs has great impact on the indexing of the tokens.

VIII. Conclusion:

By applying these techniques we were able to perform proximity search with in a sentence. We further refined our algorithm by doing proximity search on the bases of distance within words of a sentence. Our later work shall include making our algorithm more robust and make it more efficient by reducing same and time requirements.

IX. Reference:

[1] Liu, Zemin, Vincent W. Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. "Distance-aware dag embedding for proximity search on heterogeneous graphs." AAAI, 2018.

[2] Hristidis, Vagelis, Nick Koudas, Yannis Papakonstantinou, and Divesh Srivastava. "Keyword proximity search in XML trees." *IEEE Transactions on Knowledge and Data Engineering* 18, no. 4 (2006): 525-539.

[3] Fagan, Joel L. "Automatic P hrase Indexing for Document Retrieval: An Examination of Syntactic and Non-Syntactic Methods." In *ACM SIGIR Forum*, vol. 51, no. 2, pp. 51-61. ACM, 2017.

[4] Goldman, Roy, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. "Proximity search in databases." (1998).