**Team #32**

Sanjay Jagadeesh

Manohar Thagadur Nataraju

Soumya Gowda Basvana

Pramod P. Setlur

CSCI–572 Assignment 2

*An Apache-Solr based Search Engine, Ranking Algorithms and NER for Weapons Datasets*

We indexed all the data of our first assignment into Apache-Solr. We implemented *Content-Based and Link-Based* for retrieval of results from the index.

CSCI–572 Assignment 2

## Question 1: Develop an indexing system using Apache Solr and its ExtractingRequestHandler (“SolrCell”)

1. We installed Apache Solr from Lecene’s 4.10 branch.
2. N/A
3. We built Geotopic parser, OCR and ctakes as per the instructions provided to us.
4. We first executed bin/nutch dump to create the dump files. We then use this dump file in *dump\_index.py* to index the data to Solr.
5. N/A

## Question 2: Leverage the Nutch indexing system to build up an Apache Solr Index

1. We upgraded Tika to 1.11-SNAPSHOT and also included the content parser support.
2. The metadata extracted from NUTCH using TIKA included the following: id (unique id), title (name of document), segment (which segment it was uploaded from), boost (value to determine relevancy), digest, tstamp, type (metadata about file), date, contentLength, url and version.

The metadata extracted from SOLRCELL using TIKA included additional metadata along with the metadata that have been discussed above. This included stream source info, stream content type, stream size, content encoding, stream name and content type. Also additional HTML tags, Images attributes were also generated.

Overall, the metadata generated by SOLRCELL was more detailed and descriptive with regard to using this for the page ranking algorithms. Also there was more flexibility for user defined indexes which was not present using NUTCH Solrindex.

## Question 3: Design and implement two ranking algorithms for your Weapons data documents

1. The *content based* algorithm that we implemented was one similar to the algorithm employed by google. The algorithm uses a combination of 2 metrics: “tf.idf” (term frequency and inverse document frequency) and cosine similarity.

Term frequency of a word given by tf(word) in a document is simply the number of times the word appears in the document.

Inverse document frequency of a word given by IDF (word) is given by :

IDF (word) = Total Number of Documents / Number of documents containing the given word

We use the tf.idf metric to represent each document in our index and the query itself as vectors. We use the cosine similarity metric to determine how similar 2 vectors are to each other.

In this case, the 2 vectors that we compare are the query vector and a document vector. The document whose corresponding document vector resulted in the highest cosine similarity when computed with respect to the query vector is considered most relevant.

1. The metadata present in the documents were structured enough for us to link documents based on certain patterns that we observed for each metadata feature. We could hence create a graph where the nodes are the documents and the edges were the ‘links’ that we created between them. The graph is bidirectional, we do not give specific importance to the direction of reference. (Document A referring to B is the same as document B referring to A)

The link based relevancy ranking algorithm that we developed is then applied on this graph of documents. The algorithm is an implementation of the page rank algorithm and assigns a page rank to each document in the graph based on the number of links between the various nodes.

The algorithm consists of various phases:

1. Creating the Graph

The metadata that we used were

1. The latitudes and longitudes that are associate with the regions
2. Timestamp
3. Gun type
4. Creating and weighing the links
5. Divide the country of U.S. into 6 regions. NE, SE, N, S, NW, SW.
6. Two files are compared with respect to their latitude and longitude. If they match a link is drawn and a weight of 5 is assigned.
7. The two files are again compared with the timestamp. If these two again match, the link’s weight is increased to 10.
8. Further, if the gun type also match, then, the link’s weight is increased to 15.
9. The steps 2 through 5 are performed with all the pairs of files.
10. Page Rank Calculation
11. If page A is linked T1, T2, … ,Tn then page rank, PR(A) is:

PR(A) = PR(T1) / C(T1) + PR(T2) / C(T2) + … + PR(Tn) / C(Tn)

Where C(Ti) is number of links emanating from Ti.

## Question 4: Develop a suite of queries that demonstrate answers to the relevant weapons related questions below.

1. Test
2. Test
3. Test
4. Test
5. Test

Question 5: Develop a program in Python that runs your queries against your Solr index and outputs the results in an easy to read list of results demonstrating your relevancy algorithms and answers to your challenge questions from Task #4.

We implemented a python program for the above.

EXTRA CREDITS

Question 6: Develop a Lucene-latent Dirichlet allocation (LDA) technique for topic modeling on your index and use it to rank and return documents.

Lucene- Latent Dirichlet Allocation (LDA) technique was implemented using the following:

Intuition: The key concept involves a set of topics which can be the result of a query. The topics may be related or can be dissimilar but they all are found to be existing in the documents. The LDA technique makes use of all these topics and ranks the pages on the set of topics that the pages might appear to be in.

Each topic is considered to be a probabilistic distribution over the set of words appearing in all the documents (vocabulary). Now based on the contents of a document, the words which appear are tokenized and then each word is assigned to a topic.

There are high chances that a topic can belong to more than one topic and thus this can lead to ambiguity. A simple method to overcome this, was to implement a weighted priority which comprises of which topic covers most of the words present in the document and then assigning the document to that topic.

This topic based classification resulted in the percentage of relevancy of every document to every topic chosen. Thus the most likely pair corresponds to the topic for which the most number of words match. An updated version of the same can include bi-grams and tri-grams for topic selection.

The documents returned from the Lucene-LDA technique belong to a set of topics. These topics drive the classification which can be user chosen, which is much more flexible than both content-based and link-based. The LDA algorithm is not completely document dependent, as compared to content and link-based, both of which is driven by the data that is present in the documents. LDA is driven by the topics, so the ranking order is determined by the topic the queries belong to. This incorporates the meaning of the document rather than just taking all the words and determining their TF.IDF values. The semantics of the document are taken into account. However, the complexity of the LDA algorithm is really complex and the runtime of the algorithm is much greater than the content-based as well as link-based algorithms. In some cases, topics that are known might not be discovered as the algorithm tends to fit the document to a set of predetermined topics, which might not always be the case.

Question 7: Figure out how to integrate your relevancy algorithms into Nutch

The NUTCH platform has a scope for incorporating Page Ranking as a scoring mechanism for retrieving relevant documents. The three java classes that need to be updated are:

* ScoringFilter.java
* ScoringFilters.java
* ScoringFilterException.java

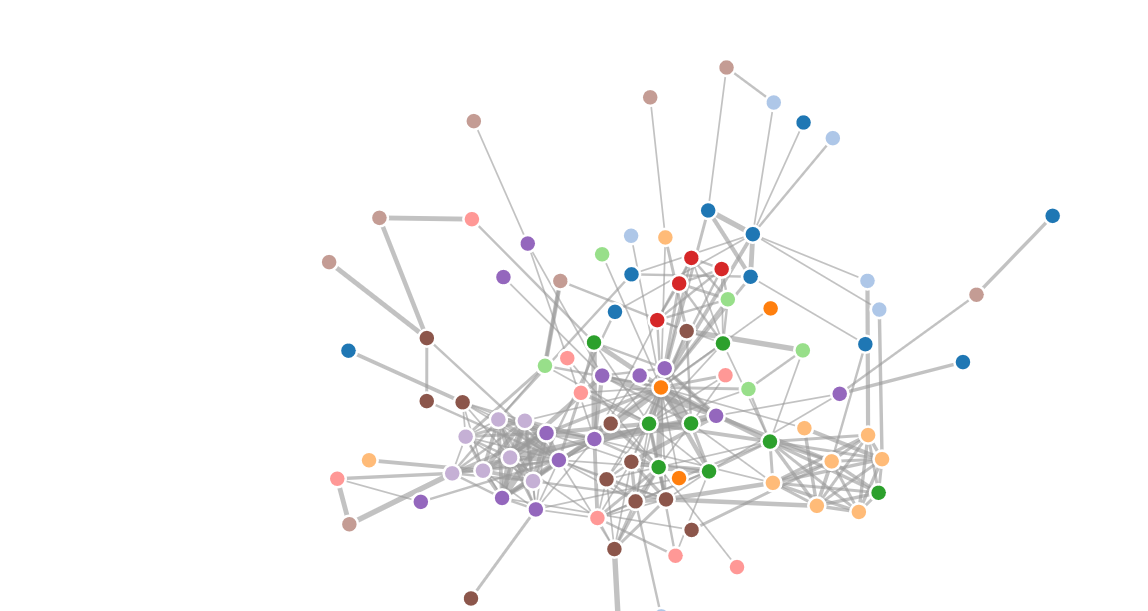
ScoringFilter.java is the interface which has the methods that need to be implemented in order to incorporate a custom scoring algorithm into NUTCH. The methods which were implemented are:

* generatorSortValue() : This is used to set the sorting to descending order so as to retrieve the most relevant documents first.
* initialScore(): which is used to initialize the scoring for the initial set of pages.
* injectedScore(): which is used to re run the scores initializations when a new document is added.
* updateDbScore(): calls the page ranking algorithm.
* indexerScore(): call the page ranking algorithm and returns the hash maps values which are the relevancy parameter used for ranking.

The three files are included along with the project in the “Extra Credit” folder and successfully complied along with NUTCH.

Question 8: Create a D3-based visualization of your link-based relevancy. Provide a capability to generate D3 relevancy visualizations as a Nutch REST service using Apache CXF. Integrate the service into nutch-python.

The following diagram shows the way in which the link-based relevancy is computed. This is done leveraging the “Force-Directed Graph” from D3. The links are generated based on the link-based results incorporating location as the feature. The data points represented in the same color are grouped into a common groups. This was generated based on thresholding the link weights and computing the documents that are close. The graph depicts the nature of the documents present.



The D3 visualization was done using index.html and input.json.

Input.json has the nodes, which is the names of the documents. The links between them are based on the groups which is represented as “group” in the json. The group is determined the using the link-based algorithm. The links in the json are determined by the distances of each document form one another, computed based on the latitude-longitude occurrence of each document. The centroid of each of the document is obtained and the distance are calculated from the centroids of all the documents. Assuming the documents are ‘n’, there will be n! such values, which makes this problem computationally hard. Hence the data shows is done using just 100 odd documents. (NOTE: 1000 documents were given as input, but Mozilla was not able to compute the massive number of links. Hence the data set was reduced to 100, so that the visualization is also effective).

Index.html and input.json have been included in the “Extra Credit” folder

During visualization, a number of nodes were present as outliers. This was due to the fact that the link-based algorithm did not find any relevancy of such documents to the central mass of the documents that were present or maybe the distance was too large to be depicted on the D3 visualization. By hovering over each of the nodes, the name of the file can be obtained. Also a Force Directed graph was chosen as this depicts that any node of the peripheral outline of the visualization can be moved without affecting the overall mass. However, the documents present towards the center of the mass are important, and moving these nodes in the D3 can shift the entire mass. The reason maybe the number of links for these nodes is high.

NOTE: Please open index.html in Mozilla Firefox. Make sure the input.json is in the same folder as index.html

CONCLUSION

How effective the link-based algorithm was compared to the content-based ranking algorithm in light of these weapons challenge questions?

What questions were more appropriate for the link based algorithm compared to the content one?

Describe in detail and formally both of your ranking algorithms. You should describe the input, what your algorithms do to compute a rank, how to test them

Describe the indexing process – what was easier – Nutch/Tika + SolrIndexing; or SolrCell or ElasticSearch?