[Extra Credit] CSCI 572: Assignment II Report
Building an Apache-Solr / ElasticSearch based Search Engine, Ranking Algorithms and NER for Weapons Datasets

Team number: 33
Team members: Gaurav Shenoy, Mahesh Kumar Lunawat, Pramod Nagarajarao, Presha Thakkar, Karthik Kini

Task 1: 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:

The main concept is that there will be a set of topics which may be the result of a query. These topics may be related to each other or dissimilar but they must be found in the existing documents. LDA technique makes use of these topics and ranks the pages on the set of topics that pages might appear to be in.

We defined the set of topics keeping in mind that each topics is probabilistic distribution over the set of words appearing in all the documents (vocabulary). After this depending on the contents of a document, the words which appear are tokenized and then each word is assigned to a topic.

There might be probability that a topic can belong to more than one topic which can lead to ambiguity. Simple method to overcome this may be implementing a weighted priority which involves finding which topic cover most of words present in the document and then assigning the document to that topic.

Next what percentage of each document corresponds to each topic is computed. To determine the percentage for each topic we measure the frequency of each word belonging to the topic and occurring in the document. Now, we divide the calculated frequency by the total no of words in document and multiply by 100.

The documents returned after running the Lucene LDA technique belong to a much narrow domain than documents returned after running the content and linked based algorithm cover a much wider domain.

In LDA, all the documents that belong to topics of interest are returned first instead of documents that contain the query words most no of times which explains why the documents that are semantically more relevant to the queries are given precedence over those that merely contain the search keywords a lot of times. However the LDA algorithm complexity is much greater than the content based and link based algorithm.

Task 2: Figure out how to integrate your relevancy algorithms into Nutch.

Follow the steps below to Integrate the relevancy algorithms into Nutch:
1. In Eclipse, under the File menu, click on Import.
2. In the pop-up, select Projects from Git under the Git category. Click on Next.
3. In the next pop-up, click on Clone URI and then click on Next.
4. Now, in the text box for ‘URI’, enter the following - https://github.com/apache/nutch.git , which is the Git repository for Nutch.
5. Upon clicking Next in last step, In the Branch Selection pop-up, let all the files be selected (default). Click on Next.
6. In the Local Destination pop-up, enter the directory (in our case, /Users/JK/git/nutch) where you want the Nutch repository to be stored and click on Next.
7. Wait for the projects to download and select the Import as general project option. Click on Next.
8. Type the project name and click on Finish.

9. Now, in the Package Explorer, you should see a project named nutch.
10.Go to nutch -> src -> java -> org -> apache -> nutch-> scoring. In this folder, you should see a file named ScoringFilters.java where you can add your link-based algorithm's code.

Apache Nutch has a scope for adding Page Ranking as a scoring mechanism for retrieving relevant pages. Below are the methods needed to implement in order to incorporate a custom scoring algorithm into Nutch are:

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

Task 3: 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.

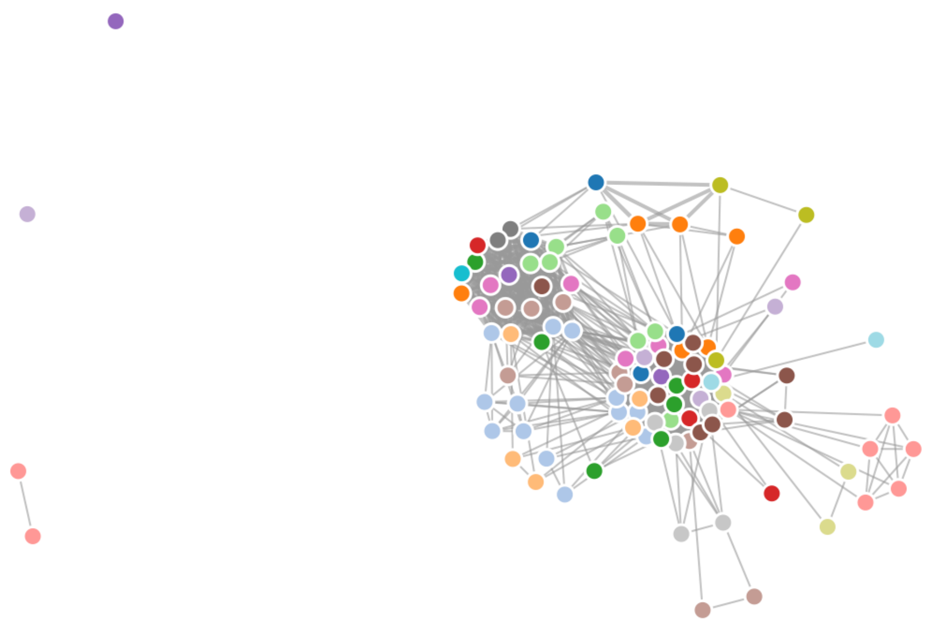

Fig 1: D3 Visualization of linked-based relevancy algorithm

We have used the force-layout D3 visualization technique for link-based relevancy. The technique expects two arrays. An array of objects that are the nodes, which can have different attributes and an array of links, which are objects that need to have a .source and .target attribute that point to the array position of the nodes that they link. Whatever we export should be an array of JSON objects for the nodes and use hash to translate the id values of source and target to the array position of those objects in that array.

The D3 visualization is established using index.html and input.json. In this task we first started with more than 500 number of documents. The resultant JSON and the D3 visualization was way too cluttered and became very hard to distinguish between various nodes and the links. Later we started with 100 documents and above visualization was obtained.

We created a json file which had two keys arrays, nodes and links. The value to each key is an array of objects. The nodes has the “name” and “group” as the object properties. The name field represents the “name” of the documents and the “group” property represents the geo location. Each geolocation group is represented by a different color in the D3 visualization. All documents with the same geo location group are represented with the same color.

The other key, links, has the following object properties - source , target and value. The source and the target define the source document which is linked by an edge to a target document. The value field shows the amount of similarity between two documents. The more the value the thicker is the edge between two nodes. Thicker the edges, more is the similarity. We have “Location”, “Gun type”, “manufacturer” and “keywords” as similarity measures.

Please open index.html in Mozilla Firefox. Make sure the input.json is in the same folder as index.html. All the necessary documents associated are available in “D3 Visualization” Folder.