CSCI–572 Assignment 2

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

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

Team #32

Sanjay Jagadeesh

Manohar Thagadur Nataraju

Soumya Gowda Basvana

Pramod P. Setlur

CSCI–572 Assignment 2

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

a. We installed Apache Solr from Luecene’s 4.10 branch.
b. N/A
c. We built Geotopic parser, OCR and CTAKES as per the instructions provided to us. We updated lucene/ivy-versions.properties to include the version of the apache tika version as 1.11-SNAPSHOT.
d. We first executed ‘bin/nutch dump’ to create the dump file. We filtered HTML and image files from this dump file The metadata extracted from these URLs were indexed to Solr using *dump_index.py*.
e. N/A

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

a. We upgraded Tika to 1.11-SNAPSHOT and also included the content parser support.
b. The metadata extracted from Nutch using Tika included: 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 the Nutch dump images using Tika-python were indexed to Solr using Solrpy. This included image height and width, file size, compression type, description and content type.

   Overall, the metadata generated by Solrpy was more detailed and descriptive, helpful 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

a. 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 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, we compare 2 vectors, 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.
In addition, certain index fields were boosted at query time that increased the relevancy of content based algorithm. This ensured that the boosted fields were given priority over the other fields while querying.

Algorithm

1. Given a query q. We represent it as a vector with each dimension of the vector corresponding to the product of the normalized term frequency of each word in the query and its inverse document frequency (idf) over the entire set of indexed documents.
   Note: The idf of a word is a constant for a set of documents that we have indexed.
2. For each indexed document d, we create a vector as we did for the query
3. Let the query vector be q and document i 's vector be d_i . We then compute the cosine similarity between q and d_i as:
   $$\text{Cosine Similarity}(q, d_i) = \frac{q.\ d_i}{|q|\ |d_i|}$$
4. We calculate the cosine similarity between a given query and all documents that we have indexed. We then return the documents in decreasing order of cosine similarity.

   Input: Query String and a file containing the contents of all the indexed documents separated by new lines.
   Output: Pages in decreasing order of relevancy score computed.
   Test Proof: We created a query and a text file for a subset of the documents we indexed and the final score
   calculated by the algorithm was as expected.

b. 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 created a graph using the NetworkX library of python 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 page rank is calculated for each node and is added as an additional index field for the corresponding document.

The algorithm consists of various phases:

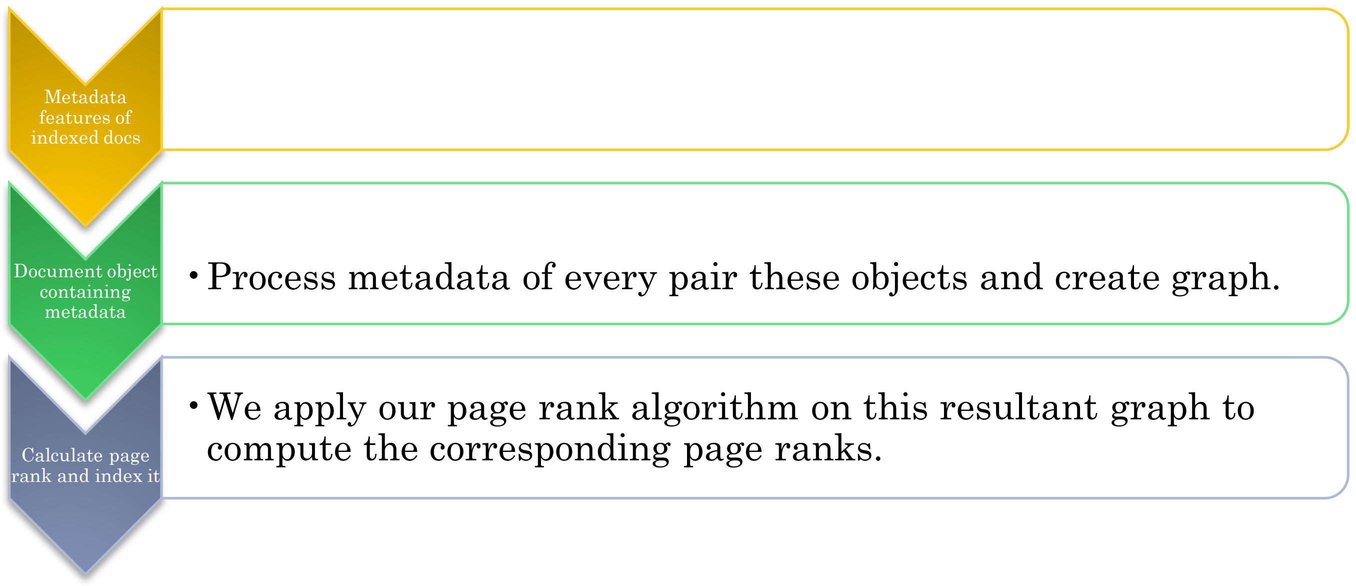
i. Features used for constructing links of the graph
   The metadata that we used were
   1. The latitudes and longitudes that are associate with the regions
   2. Timestamp
   3. Gun type
ii. Creating the graph
   1. Divide the map of U.S. into 6 regions. NE, SE, N, S, NW, SW.

2. Two documents are compared with respect to their latitude and longitude. If they fall in the same region a link is drawn and a weight of 5 is assigned.
3. The two files are again compared with the timestamp. If these two again match, the link's weight is increased to 10.
4. Further, if the gun type also match, then, the link's weight is increased to 15.
5. The steps 2 through 5 are performed with all the pairs of files.

iii. Page Rank Calculation

1. 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.
2. The NetworkX library uses the above formula to compute the page rank.

WORK FLOW OF THE ALGORITHM


Metadata features of indexed docs
Document object containing metadata
• Process metadata of every pair these objects and create graph.
Calculate page rank and index it
• We apply our page rank algorithm on this resultant graph to compute the corresponding page ranks.

Input: Metadata of all indexed documents in JSON format.
Output: Pages in decreasing order of relevancy score computed.
Test Proof: Testing of the graph creation and the page rank calculation parts of the algorithm were carried out independently. We provided a JSON file containing metadata of a subset of indexed files whose graph structure we manually computed, the algorithm obtained same results. We then fed the algorithm a graph, the page rank of whose documents we already computed.

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

QUESTIONS	QUERIES	EXPLANATION
What time-based trends exist in Gun ads? Can you correlate temporal and spatial properties with buyers? For example can you identify based on ad time-window and/or based on geospatial area places where people try and purchase guns on behalf of someone unauthorized to purchase them?	http://localhost:8983/solr/collection2/select?q=(description:ad OR url:ad OR title:ad) AND (geo_name:Texas)&facet=true&facet.date=tstamp&facet.date.start=NOW/month-12MONTHS&facet.date.end=NOW/MONTH&facet.date.gap=%2B1MONTH	The faceted query gives the time based trends for gun ads at the location Texas for every month, starting from 12 months prior to current date. Here, Solr aggregates the result count against the corresponding month. This way we get time based trends in a monthly manner. Similarly, the time based trends can be obtained for any given location with the required time period. The temporal (ad-time window) and spatial (geo spatial areas) properties can be correlated with the buyers wherein we could identify the increase or decrease in the sales of weapons for a particular region at the particular time.
Can you identify similar firearms image types (e.g., shotguns) that are sold in the same region and time? Does this indicate influx related to stolen goods?	localhost:8983/solr/collection2/select?defType=edismax&qf=description^10&qf=itemOffered^10&q=(itemOffered:Shotguns AND geo_name:Idaho)&fq={!field f="seller startdate" op=Contains}[2010-10]&rows=150&wt=json&indent=true	In the query, we are searching for a particular firearm type 'Shotguns' in a particular region 'Idaho' in a particular month of the year. Similarly the query can be extended to search for other firearm types such as Rifles, Pistols,etc for a particular region and time.
When a shipment of bulk firearms is stolen, the rate of ads and images may indicate an increase in sales of that particular make/model – can you identify these?	localhost:8983/solr/collection2/select?defType=edismax&qf=description^10&qf=title^10&q=(itemOffered:Shotguns) AND (title:Remington OR description:Remington)&rows=1500&wt=json&indent=true	In the query, we found multiple ads for one of the make or model 'Remington'. This indicated the increase in sales of 'Remington' model. Similarly we can find the increase in the rate of ads or sales in a particular make or model by including the required make or model in the query.
Can you identify ads and/or weapons images that are posted by persons, whom are underage or in which the weapons are de-identified (by type and/or serial number, etc.)		Identifying by person's age: We were unable to find any metadata related to the age of the person posting the weapons' images or ads. Hence we could not identify such ads or images sold by people who are underage. Identifying by serial number: We tried using Tesseract to extract the serial number of the weapon.

Can you identify ads and/or images that relate to the unlawful transfer, sale, and possession of explosives, WMD devices, and precursors ?	http://localhost:8983/solr/collection2/select?defType=edismax&qf=description^10&qf=title^10&q=description:nuclear OR description:chemical OR description:fusion OR description:fission&rows=1500&wt=json&indent=true	We tried to find Weapons of Mass Destruction Devices by querying the search terms such as, Nuclear, Chemical, Biological, Fusion and Fission bomb, etc.

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, query_run.py.

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.

We implemented Lucene- Latent Dirichlet Allocation (LDA) technique the following way:

Intuition: The topics of query 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 is a high probability that a topic can belong to more than one document, leading to uncertainty. Weighted priority solved this problem. The topic that covers most of the words present on the document is assigned to the document.

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 is a subset of topics. LDA can be concluded to better and flexible than Content-based and Link-based, since they drive the classification. 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.

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

There's a scope for incorporating Page Ranking as a scoring mechanism for retrieving the relevant documents in NUTCH. The following three java classes need to be updated.

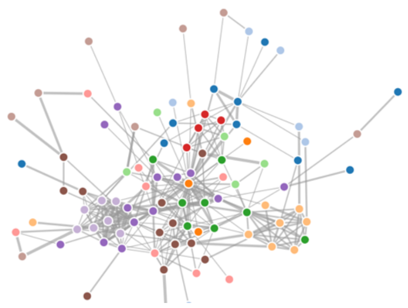
- ScoringFilter.java
- ScoringFilters.java
- ScoringFilterException.java

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 method is used to set the sorting to descending order to retrieve the most relevant documents first.
- initialScore :This method is used to initialize the scoring for the initial set of pages. injectedScore():This method re-runs the scores initialization when a new document is added.
- updateDbScore(): This method calls the page ranking algorithm.
- indexerScore():This method calls the page ranking algorithm and then it returns the hash map values. These are the relevancy parameters used for ranking.

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 below figure shows the way in which the link-based relevancy is computed. This is done using the "Force-Directed Graph" from D3. The links are generated based on the link-based results by using location as the feature. The data points that are of same color are grouped into common groups. This was generated based on threshold the link weights and computing the documents that are close. The graph shows the nature of the documents that are present.



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

Input.json has the nodes, which are 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 is 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 shown is done using just 100 odd documents.

A number of nodes were present as outliers during visualization. This was mainly 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?

Content Based algorithm were seen to be summarization algorithms. These algorithms returned documents that had words corresponding to a query as unique keywords. However, the presence of a word in the query as a unique keyword did not always reflect the relevancy of the document with respect to the query. Link based algorithm however yielded better results purely in terms of how relevant a document was to our queries irrespective of its pure textual content. The citation based ranking method proved that a lot of documents citing a document with respect to a query adequately demonstrated its relevance.

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

Explained in table present in task #4. Link based algorithm was superior with regard to region based questions. Link based answered queries with regions and routes effectively as this algorithm takes advantage of the location information present in the metadata.

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.

This has been answered section 3a and 3b.

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

We used Solrpy instead of SolrCell to avoid tweaking of ExtractRequestHandler. Indexing using Solrpy was easier because of the flexibility it provides. Nutch/Tika + SolrIndexing cannot add fields during POSTing of data. The fields that are mentioned in the Schema.xml are taken by default and whatever Tika generates with respect to those fields are indexed automatically. The indexing process is very rigid. Solrpy allows adding content in a easy manner with dynamic field generation. This approach enabled us to answer the queries of Task #4 efficiently as the indexed data contained additional fields relative to Nutch/Tika + Solrindexing.

The objective of the assignment was to develop a notion of relevancy related to the weapons datasets that were crawled in the first assignment (Web crawling of weapon images). The relevancy among weapon datasets was measured using two generations of algorithms:

Text-based summarization: TF-IDF technique.
Graph-based summarization: Link based technique.

We faced multiple challenges throughout the project. The following are the major challenges:

Identifying the important fields to build the index. Based on the data model of weapons data, we had to figure out what were the important fields for relevancy measurement. The timestamp is an important field which enabled us to answer queries relating temporal properties of the weapon dataset. There were other fields such as color component quantization which weren't of much use and had to be ignored when compared to other important fields such as timestamp, image width, description, etc.
Developing query suites for the task #4 was challenging as it involved the usage of various Solr query parameters such as facet query, functional query and combining multiple query parameters to achieve the desired result.
Upgrading SolrCell to include 1.11-SNAPSHOT tika trunk. Though this looked straight forward, it required to dig deeper into the JIRA issues of tika trunk upgradation for SolrCell. Detailed analysis of JIRA issues helped us to identify the right ivy-verisons-properties wherein the required version of tika had to be updated.
Working with the open source technologies such as GeoTopic Parser, OCR and Annotating the Solr documents with the developed link based relevancy algorithms.