Assignment 2

Relevancy related to the Weapons Dataset

Task 1: Indexing System using Apache Solr and its ExtractingRequestHandler(SolrCell)

a. We installed Solr using

svn co http://svn.apache.org/repos/asf/lucene/dev/branches/lucene_solr_4_10/ Solr

b. We didn't use ElasticSearch for this assignment.

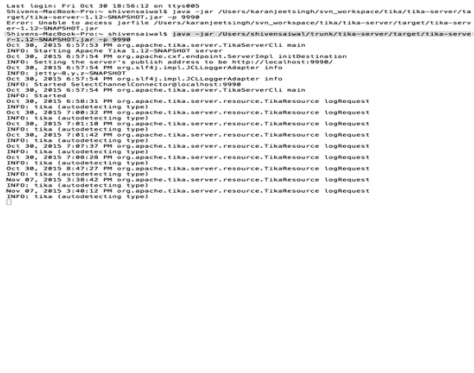
c. We upgraded SolrCell similarity as follow:

- We built Tika trunk (1.12-SNAPSHOT) in a trunk folder using this command –
  svn checkout http://svn.apache.org/repos/asf/tika/trunk tika-site
  mvn install
- We now built the **GeoTopic Parser** over Tika trunk –
  For running the Geotopic Parser Server on 9992:
  java -classpath /Users/shivensaiwal/trunk/tika-serr/target/tika-server-1.12-SNAPSHOT.jar:/Users/shivensaiwal/location-ner-model:/Users/shivensaiwal/geotopic-mime org.apache.tika.server.TikaServerCli -p 9992

  To get the data from the Geotopic Parser Server:
  curl -T /Users/shivensaiwal/Desktop/Image.geot -H "Content-Disposition: attachment; filename=image.geot" http://localhost:9992/rmeta

- We now built the **Tika OCR** over Tika trunk –
  For running the Tika OCR Server on 9990:
  java -jar /Users/shivensaiwal/trunk/tika-server/target/tika-server-1.12-SNAPSHOT.jar -p 9990

  To get the data from the Tika OCR Server:
  curl -T /Users/shivensaiwal/Desktop/572Assignment/Data/bullet.jpg http://localhost:9990/tika --header "X-Tika-OCRLanguage: eng"



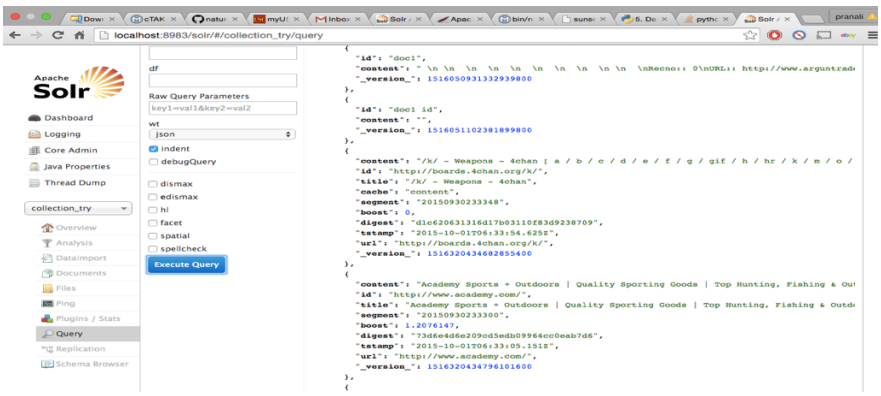
- We now built the **CTakes** over Tika trunk –
  For running the CTakes Server using Tika-App:

java -classpath /Users/pranalijhaveri/Desktop/src/ctakes-config:/usr/local/Cellar/tika/1.10/libexec/tika-app-1.10.jar:/Users/pranalijhaveri/Desktop/src/apache-ctakes-3.2.2/desc:/Users/pranalijhaveri/Desktop/src/apache-ctakes-3.2.2/resources:/Users/pranalijhaveri/Desktop/src/apache-ctakes-3.2.2/lib/* org.apache.tika.cli.TikaCLI --config=/Users/pranalijhaveri/Desktop/src/ctakes-config/tika-config.xml -m /Users/pranalijhaveri/Desktop/CS572/htmlindex/g5.html

```
Content-Encoding: UTF-8
Content-Length: 36598
Content-Location: .
Content-Type: application/xhtml+xml; charset=UTF-8
Content-Type-Hint: text/html; charset=UTF-8
Description: Marlin Model 39 Century LTD 22 S-L-LR lever act...
Keywords: gun classifieds, free gun classifieds, guns for sale, used shotguns, used handguns, used pistols, buy used guns,
 sell guns, buy guns online, sell guns online
X-Parsed-By: org.apache.tika.parser.CompositeParser
X-Parsed-By: org.apache.tika.parser.ctakes.CTAKESParser
X-Parsed-By: org.apache.tika.parser.html.HtmlParser
author: Luka Cvrk (www.solucija.com)
ctakes:DiseaseDisorderMention: Condition:1361:1370:C0012634
ctakes:MedicationMention: blade:1228:1233:C2948008
ctakes:MedicationMention: blade:1249:1254:C2948008
ctakes:RomanNumeralAnnotation: L:1055:1056:
ctakes:SignSymptomMention: sight:1234:1239:C0042789,C0042789
ctakes:SignSymptomMention: sight:1255:1260:C0042789,C0042789
ctakes:SignSymptomMention: ads:1654:1657:C0332133
ctakes:schema: coveredText:start:end:ontologyConceptArr
dc:title: FREE Gun Classifieds - Marlin Model 39 Century
resourceName: g5.html
title: FREE Gun Classifieds - Marlin Model 39 Century
Pranalis-MacBook-Air:Desktop pranalijhaveri$
```

d. We use:
bin/nutch readseg –dump /Users/pranalijhaveri/Desktop/nutch/pranali/crawldata/crawl_1/segments/ /Users/pranalijhaveri/Desktop/nutch/pranali/crawldata/crawl_1/dump
and then we followed Task #2 to index the data.

Task 2: Indexing Process of Nutch/Tika+SolrIndexing and SolrCell:
Using the command ***bin/nutch solrindex***, we were able to index the crawl data that was done by nutch in solr. All the data was indexed using SolrIndexer, that was initially parsed by Tika, Tesseract, Selenium creating crawldb, linkdb and segments folders. This process includes customization such as mapping fields as specified in the schema. URL filtering and normalization is also provided. Solr added the following fields: content, id, version, boost, segment, title, timestamp, url and digest.



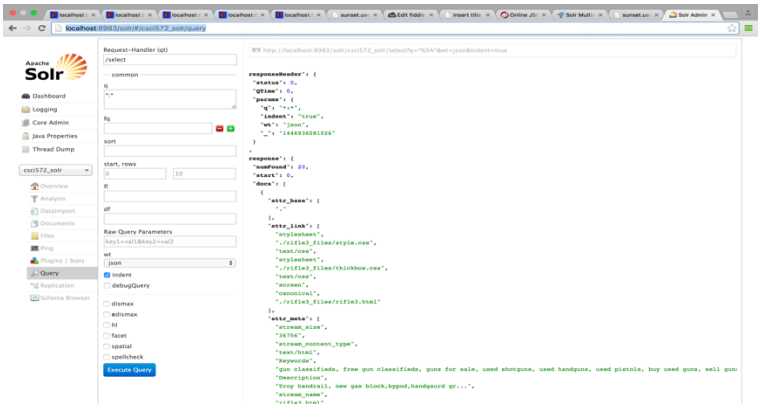
Before using SolrCell functionality, we dumped the crawl data (ads and images).
We did indexing using SolrCell by using post.jar and mentioning parameters using *command*. Hence, SolrCell adds following metadata:

- Stream_Name: name of Content Stream
- Stream_Size: size of stream
- Stream_content_type: Content type of stream if it is available eg. text/html, images/*
- Stream_source_info: source info about the stream

Solr Cell provides a lot of customization. We used attri prefix to add prefix to some metatypes. Solr Cell also allowed us to boost the documents while indexing so as to have more efficient results while searching. We added this field

<dynamicField name="attr_*" type="text_general" indexed="true" stored="true" multiValued="true"/>

to our schema.xml to enable the metadata content that we are fetching from ads and images.



Differences:

SolrCell does all the indexing, parsing and extracting on the server side whereas Nutch/tika+solr indexing does all the work on client side. There might be an issue when you don't know the content type of the document on the client side. In that case server-side is helpful. It was possible in SolrCell to add an extra field for date modified which was not possible in Nutch/Tika + SolrIndexer. SolrCell was also helpful in adding dynamic fields such as uprefix_ and copyfields for extracting body content. Thus, SolrCell indexing was more helpful and easier as it allowed us to customize the indexing so that it could be more helpful while querying the index.

Task 3: Content Based Relevancy Algorithm

Input: The query URL will be the input. A request is sent to the server with this URL which gives the extracted JSON as the output. The JSON is parsed and the Content is extracted.

Working:

1. Split the query into words, removing all stop words
2. N-grams of the words in the query are found

For ex. if query is "ads weapons images", the n-grams(queryPhrases) will be "ads weapons images, ads weapons, weapons images, ads images, ads, weapons and images"

3. Find the synonyms and similar words from the goldstandard.txt
4. for each URL:
   a. Find n-grams of the words of the content
   b. for each current word of the query:
      Find the TF-IDF of the word/phrase
   c. for each current word of the content:
      Find the TF-IDF of the word/phrase
      Display the top three TF-IDF values

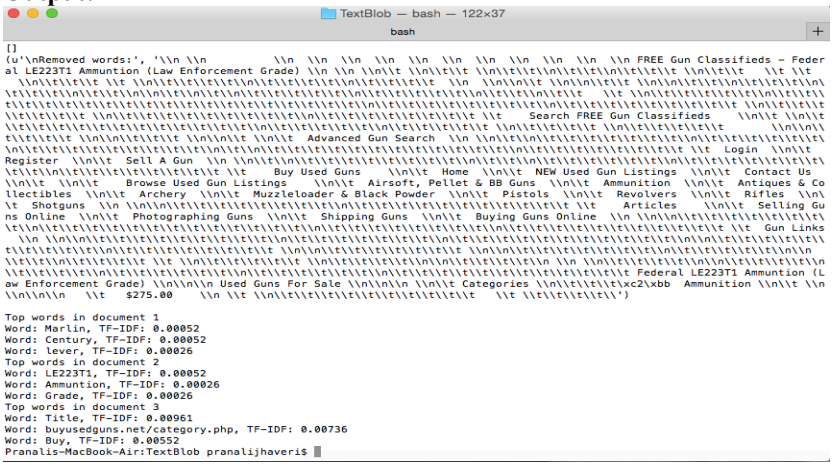
d. Boosting of the TF-IDF values using:

- If the TF-IDF values of the document match that of the query, then we boost the document score by a factor of 10
- The **phrases** are boosted by a factor of the number of words in the phrase

External Resources Used:

Text Blob Python Package
Math Python Package
Decimal Python Package

Output:



Link Based Relevancy Algorithm:

Input:
A request is sent to the Solr server and we get the extracted JSON data as the resultant output. The JSON data is parsed and from that, we extracted several fields along with the metadata features such as doc id, html body, attr_links, attr_keywords, attr_geoname and attr_time which are used in defining relevancy. We have created a link based algorithm on the basis of the metadata features (attr_geoname and attr_time).

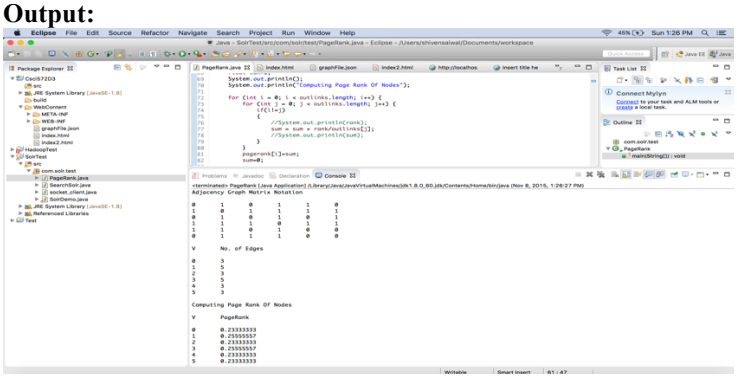
Working:

1. All the documents are extracted on executing the query on the Solr server
2. Created two subsets of the documents
    a. Subset 1- Based on the attr_geoname
    b. Subset 2- Based on the attr_time
3. Each of these documents that are present in both the above subsets are represented by a node in the graph
4. Created an edge between any two nodes based on metadata features like location and year
5. If these metadata features are present in both the subsets then, we link the documents (nodes)
6. The document that has the maximum number of edges becomes the most relevant document
7. The page rank for each of the relevant documents is computed
8. At the time of indexing, the relevant documents that have the same location is boosted by a factor of 2

External Resources Used:

solr-solrj-4.0.0.jar
org-apache-commons-codec.jar
apache-commons-logging.jar
slf4j-1.7.12/integration/lib/junit-3.8.1.jar
slf4j-1.7.12/integration/lib/slf4j-api-1.6.99.jar
slf4j-1.7.12/integration/lib/slf4j-simple-1.6.99.jar
httpmime-4.3.jar
httpclient-4.2.3.jar
httpcore-4.2.1.jar

Output:



Task 4: Queries

1. 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?

    Query:
    http://localhost:8983/solr/csci572_solr/select?q=attr_input%3Aunauthorized&attr_geoname%3ATexas&%0A&rows=20&fl=id%2Cattr_keywords%2Cbody%2Cattr_geoname%2Cscore%2Cattr_pagerank%2Cattr_input%2Cattr_time&wt=json&indent=true

    Result:
    We are searching on the basis of the fields attr_geoname and attr_input. The search parameter for attr_geoname is “Texas”. The search parameter for attr_input is “unauthorized”. The result we are getting shows that most of the documents are there in the years 2014 or 2015 and is closely related to “Texas”. The result is shown in #Query1.png

2. 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?

    Query:
    http://localhost:8983/solr/csci572_solr/select?q=attr_geoname%3AFlorida&attr_keywords%3Ashotguns&body%3Ashotguns%5E2.0+attr_keywords%3Ashotguns&fl=id%2Cattr_keywords%2Cattr_img%2Cscore%2Cattr_pagerank%2Cattr_geoname%2Cbody&wt=json&indent=true

Result:
We are searching on the basis of the fields attr_geoname and attr_keywords. The search parameter for attr_geoname is "Florida". The search parameter for attr_keywords is "shotguns". On the basis of our search, we found that the documents that have a keyword called "stolen" in their content, which shows that there an influx related to stolen goods. The result is shown in #Query2.png

3. 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?

Query:
http://localhost:8983/solr/csci572_solr/select?q=body%3Astolen&rows=20&fl=id%2Cattr_links%2Cattr_geoname%2Cbody%2Cattr_pagerank%2Cscore%2Cattr_keywords&wt=json&indent=true

Result:
We are searching on the basis of the field body and the search parameters are "stolen". The result we are getting are specific links of handguns and rifles, which goes to show that there is an increase in the sales of handguns and rifles if the bulk of the shipment is stolen. The result is shown in #Query3.png

4. 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.)

Query:
http://localhost:8983/solr/csci572_solr/select?q=attr_links%3Aanyone%5E3&body%3Arifles&wt=json&indent=true

Result:
We are searching on the basis of the field attr_input and the search parameter is "anyone". We have found a string "Guns are available to anyone" in our result, which is shown in #Query4.png

5. Can you identify ads and/or images that relate to the unlawful transfer, sale, and possession of explosives, WMD devices, and precursors?

Query:
http://localhost:8983/solr/csci572_solr/select?q=body%3Atannerite&body%3Aexplosivesrows=20&fl=attr_keywords%2Cattr_pagerank%2Cid%2Cscore%2Cbody%2Cattr_geoname&wt=json&indent=true

Result:
We are searching on the basis of the field body and the search parameters are "tannerite and explosives". The result is those documents that have "tannerite" and "explosives" in the body, which is shown in #Query5.png

Task 5:
We have developed a program in Python that fires a query to our Solr Server and fetches the corresponding JSON Data.

Final Observations as per the Relevancy Algorithms and Search Queries:

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?

Our observation based on the query results showed that Link based algorithm is more relevant as compared to the Content based algorithm. Content based algorithm produced a lot of false information. In Content based algorithm, we found that there is a chance that not all the stop words are not removed. This makes the scoring values inaccurate. Link based algorithm used external features such as location and date to rank the documents, while the content based algorithm was just based on the words and phrases in the query which may contain a lot of irrelevant words which are present in almost all documents but still are not relevant. The questions where we wanted information on the particular make/model of the weapons in a specific time-window(year) were more appropriate for the link based algorithm since in Link Based we used the location and time metadata features to create links which then made our search easy and quicker.

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

We have already answered this question in **Task 2**, where we explained the indexing process and the differences between Nutch/Tika + SolrIndexing and Tike SolrCell.

Extra Credit:

Task 6:

Latent Dirichlet allocation (LDA) is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. To achieve topic modeling, we ran LDA using Mallet to generate 3 files from it . The achieve it is as follows:
1. Compressed text file containing the words in the corpus with their topic assignments.
2. Topic composition of documents
3. This file contains a "key" consisting of the top k words for each topic (where k is defined by the --num-top-words option). This can be useful for checking that the model is working as well as displaying results of the model. In addition, this file reports the Dirichlet parameter of each topic.

We use the these topics to refine our document results generated on running a query.The steps followed are :

1.Extract words from topic and find similar words and synonyms for them using word net dataset.
2.Calculate TFIDF for all documents returned by query.
3.Adjust TFIDF values using topics generated from LDA by increasing the value if we get a term match with a topic key.
4.Generate new TFIDF using adjusted values.

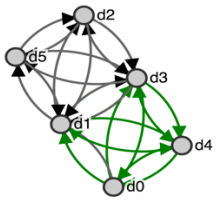
Using these TDIDF values, we generate a much better relevant sorted list of documents. This happens because we boost TFIDF values for topics generated using Mallet's LDA . Please find LDA.java included in the project which contains the algorithm. To run compile and pass the query file as a command line argument.

Task 7:

We created a plugin "SSPGTeam6Filter" using the algorithm we used for extra credit 1 —
1.We use in links count and out links count to provide a score during fetching and parsing stage.
2.When calculating the indexer score(Lucene document boost) we used the list of topics generated by Mallet's LDA and checked each documents for those topic keywords . If topic keywords exist , we increase a variable count , which is initially set to 0 in case none of the keywords match . We then multiply this keyword to a multiplier which can be configured in nutch_site.xml using the property id "sspgteam6.indexer.score.multiplier" which should be ideally less than 1 since we may find many keywords.
3.To use the plugin in nutch :
a. Copy paste the plugin to nutch_trunk/src/plugin.
b) Run "ant runtime" I'm terminal from notch root directory.
c) add "SSPGTeam6Filter" to "plugin.includes" property in nutch_site.xml.
d) Configure "sspgteam6.indexer.score.multiplier" property in nutch_site.xml . Default the value is 0.02f if not specified.

Task 8: D3-based visualization

We create a REST service for D3 Visualizations using Apache CXF's HTTP Bindings. The HTTP binding allows us to take any operation and map it to arbitrary URLs and HTTP verbs (i.e. GET, PUT, POST, DELETE). D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation. We created a simple CRUD based Java class for taking the query JSON file as an argument which we extract post running the SOLR query. We pass the JSON file to D3 Javascript APIs to create visuals on port 9005. The home URL is https://localhost:9005/ and the way to call the service is https://localhost:9005/visual/{filepath} , where file path is path of the JSON file generated from the query.



The code is mentioned in d3.js file.