**The University of Texas at Dallas**

**CS 6322.001**

**Information Retrieval**

**Fall 2016**

**Class Project Proposal**

*Project TITLE: Music Search Engine*

Students:

*Sharan Babu, SXP141731@utdallas.edu*

*Aditya Mahajan, AXM156630@utdallas.edu*

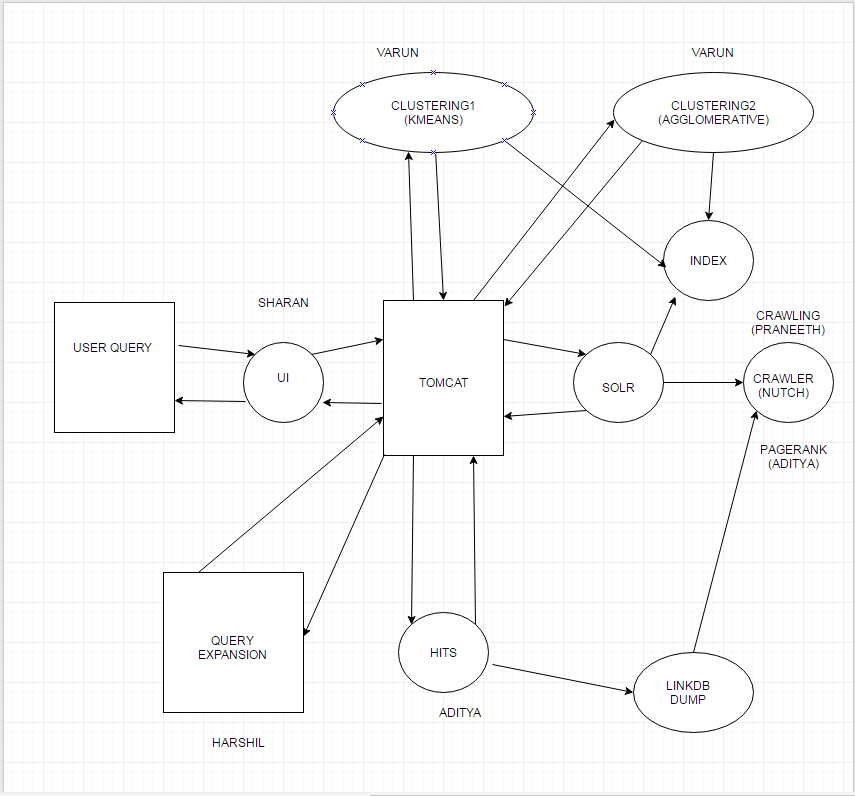
*Sairam Praneeth Vegesana, PSV150030@utdallas.edu*

*Harshil Shah, HXS155030@utdallas.edu*

*Varun Muthanna, VKM150030@utdallas.edu*

**PROJECT IMPLEMENTATION:** A Search Engine for Music

The Project Architecture is as shown below:



The Project Group consists of the following members with their respective responsibilities:

Sharan - **UI**

Aditya Mahajan – **Indexing and Clustering**

Sairam Praneeth Vegesana - **Crawling**

Harshil Shah – **Query Expansion**

Varun Muthanna – **Clustering**

**1. Crawling – Sairam Praneeth Vegesana**

For crawling web pages associated with music, The Apache Nutch Framework was utilized for crawling as well as feeding the Fetched Content from crawling to the Solr Framework hosted on localhost for indexing the fetched web pages as well as creating web graphs for implementing Page Rank and the HITS algorithms.

The number of pages crawled were 1,286,377

The number of web pages crawled & fetched were 166,180.

The Apache Nutch Framework was made to crawl Wikipedia.com, musicbrainz.org, billboard.com, rollingstone.com, youtube.com, etc.

The Nutch Framework was fed 127 URLs as seed URL to start crawling. These URLs mainly constituted Famous Musicians, Genres of Music, All Time Hits, Famous Video Channels on online video streaming websites, Famous Music Bands and Famous online music Distributors.

The following are a sample list of URLs used:

https://en.wikipedia.org/wiki/Pop\_music

https://en.wikipedia.org/wiki/Michael\_Jackson

https://en.wikipedia.org/wiki/List\_of\_online\_music\_databases

http://musicbrainz.org

http://www.billboard.com/charts

http://www.slantmagazine.com/features/article/best-singles-of-the-1990s

https://www.discogs.com/

http://www.last.fm/music

http://equipboard.com/

http://www.pandora.com/

http://www.jango.com/music/

http://www.soundtrack.net/

https://musicmp3.ru/

http://www.filmtracks.com/composers/

http://www.listal.com/musicexplore

http://www.iheart.com/artist/

http://www.axs.com/browse/music

http://rateyourmusic.com/list\_classifier/Music/

http://www.popvortex.com/music/charts/

http://www.soundike.com/charts/usa-top-100-songs.html

http://www.soundike.com/

http://www.ellatha.com/itunes/

http://www.top100songscharts.com/

http://www.rollingstone.com/music/lists/the-500-greatest-songs-of-all-time-20110407

http://www.rollingstone.com/music

http://www.allmusic.com/

https://www.yahoo.com/music

https://play.google.com/store/music

https://soundcloud.com/

http://tunein.com/radio/music/

http://www.ebizmba.com/articles/music-websites

https://pro.keepvid.com/music-resource/listen-to-music-free-online-without-downloading.html

https://emp3world.unblocked.uno/

http://beemp3s.org/

http://mp3skulls.info/free-mp3-download.html

https://www.sonymusic.com/

https://www.youtube.com/user/VEVO

http://mp3raid0.com/

http://mp3raid0.com/archive/archive/h/

http://rateyourmusic.com/charts/top/

The Command for used for facilitating the Crawling procedure in Apache Nutch is:

**bin/crawl URLS/ Crawling/ http://localhost:8983/solr 3**

where the parameters imply the following:

URLS: The directory containing the “seeds.txt” file which contains the seed URLs for crawling.

Crawling: The directory which stores the resultant directories generated by the Crawl procedure.

http://localhost:8983/solr: the URL link for the Solr host to which the crawled web pages and their contents are fed for indexing by Solr.

3: The number of iterations for which the Crawl script is executed. In every iteration, the Crawl script keeps crawling and fetching contents from unvisited URLs in the CrawlDB.

Apache Nutch generates 3 folder during the crawling operation:

1. **CRAWLDB:** it maintains the information about URLs such as the fetch status, fetching schedule, metadata, etc.
2. **LINKDB**: For each URL, the LINKDB maintains the incoming and outgoing URLs for that URL which are further used to facilitate PAGE RANKING algorithm and the HITS algorithm.
3. **SEGMENTS:** contains multiple subdirectories within it. During Crawling, the crawl script creates multiple directory to store information for Crawl Fetching, Crawl Content, Crawl Parsing, Parsed Data and Parsed Text.

The Crawling Method can be described by the following methods:

1. First Seed URLS are injected into the Crawl Database of Nutch. It maintains the list of URLs which are parsed by Nutch or are still in pending to be parsed.
2. Nutch next generates Segments for the injected Seed URLs. During the Crawl procedure, more than one segment may be generated. This helps Nutch determine which URLs have been crawled and parsed and maintains consistency with the REGEX filters.
3. Nutch begins Crawling and parsing the URL links in the Segments. It also stores information about the incoming and outgoing links in the URL.
4. When Nutch has finished crawling a set of URLs, these URLs are then added to the Crawl Database of Nutch. Also LINKDB is updated with the incoming and outgoing links of each URL.

To generate indexing, Page Rank and HITS algorithm, I collaborated with the student responsible for Indexing, PageRank and Hits. Next for implementing Page Rank and HITS algorithm, we shall require a Dump of the LINKDB inoder to utilize the incoming and outgoing links of every URL for implementing these algorithms. The command used for creating the Dump is:

**bin/nutch readlinkdb Crawling/LinkDB -dump LinkDBDUMP**

This generates a Dump containing the incoming and outgoing links for each URL.

**2. Indexing,PageRank and HITS-Aditya Mahajan**

**2.1 Indexing**

For indexing we used apache solr. After running the crawler we get our output in three folders which are linkdb, crawldb and segments. Apache solr makes use of these 3 folders and generates the index of the documents

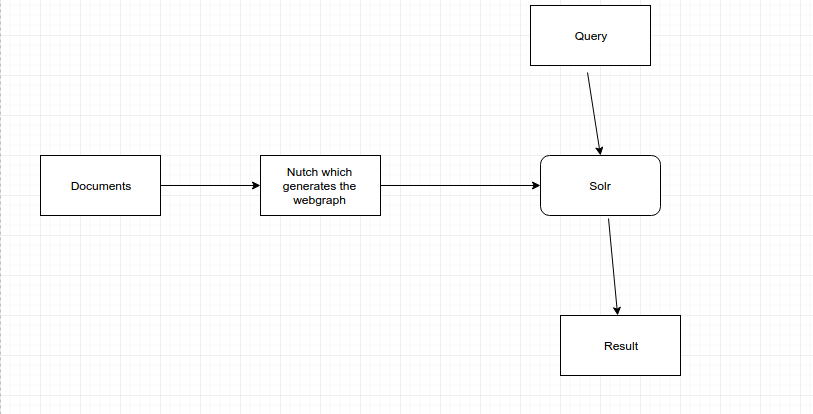
The commands used to feed the three folders(linkdb,crawldb and segments) is:

bin/nutch index -Dsolr.server.url=<url to the solr server> <path to crawldb> -linkdb <path to linkdb> -dir <path to the segmenst folder>

**2.2 PageRank**

To generate pagerank we used a webgraph which contains the number of inlinks , outlinks and the score of a given node using the pagerank algorithm. This webgraph is generated by nutch and is then fed into solr, which takes into account the TF-IDF score and the pagerank score in the webgraph to display the relevant results.(The damping factor here is 0.85)

The below figure shows the how recommendation takes place



The following steps were involved in recommending results:

1. Generate the webgraph using nutch. The comsmand used is : db bin/nutch org.apache.nutch.scoring.webgraph.LinkRank -webgraphdb crawl/<name of the webgraph folder>

2. Update the pagerank score of the crawldb, using the command:

bin/nutch org.apache.nutch.scoring.webgraph.ScoreUpdater -crawldb <path to crawl db> -webgraphdb <path to webgraphdb>

3. Index the documents using Apache solr using the command:

bin/nutch index -Dsolr.server.url=<url to the solr server> <path to crawldb> -linkdb <path to linkdb> -dir <path to the segmenst folder>

Total Number of links = 560172

Total Number of nodes = 150800

The largest number of ingoing links =203459

The largest number of outgoing links = 30

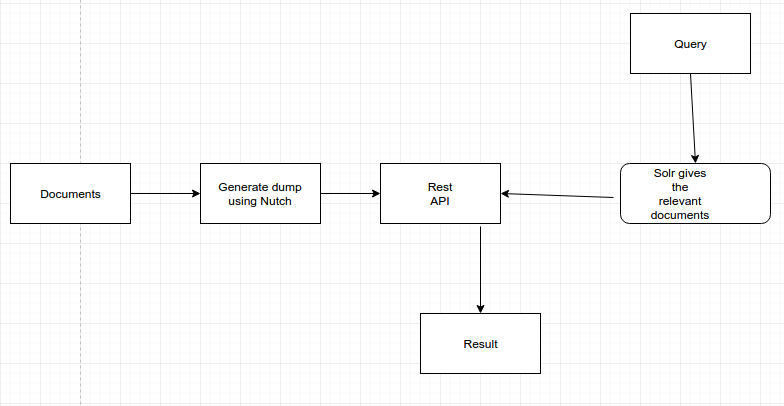
**2.3 HITS Score**

For the HITS score we made our own rest API and wrote our own code in python to generate the authority and the hub score in python.

In python we used the **networkx library** to create the graph and ran the hits algorithm on the graph to generate the hub and the authority scores.To generate the graph we needed the inlinks and outlinks to a given document. To get these inlinks and outlinks we used nutch and generated a dump of the all the documents crawled. To do this we used the command:

bin/Nutch readlinkdb <linkdb> -dump <dump\_directory>

The below image shows how hits score is generated and how it is used to show the results



The highest hubscore was assigned to:

https://en.wikipedia.org/wiki/Rock\_music (0.3318)

The highest authority score was assigned to:

<https://www.youtube.com/music> (0.2975)

**Collaboration –**

1. Clustering- Wrote the java code to generate retrieve the data from solr and generate a file which contains the url and the corresponding content/
2. Crawling- I got the segment, crawldb and linkdb from the person who was doing crawler.
3. UI- The pages generated from solr were already pagerranked and to obtain the hits score, I wrote a rest API in python which accept the urls and gives the authority and hub score for each of the urls.

**3. User Interface – Sharan Babu Paramasivam Murugesan**

Technologies Used: HTML5, JavaScript, jQuery, CSS, Bootstrap, Java Servlets

**3.1 Design:**

The system is designed as a multi-layer architecture. The following are the components of the architecture.

**3.1.1 Presentation Layer (UI Layer):**

The layout of the front end was designed using HTML5 and JavaScript(jQuery and Boostrap). The UI has the following components embedded:

1. Logo of the search engine

2. Search text box with a search button

3. Multiple tabs one each for showing results from various models.

Here are the following tabs:

3.1 *Google* – Results from Google search API

3.2 *Bing* – Results from Microsoft Bing Search API

3.3 *Vector Space + Pagerank*: Results from Solr server after indexing using vector space model and pageranking

3.4 *Vector Space + Pagerank + HITS*: Results from HITS server after applying HITS over pagerank results from Solr

3.5 *Vector Space + Pagerank + Flat Clustering*: Results from Clustering server after applying K-means clustering over Pagerank results

* 1. *Vector Space + Pagerank + Single Link Clustering*: Results from Clustering server after

applying agglomerative clustering over Pagerank results

3.7 *Vector Space + Pagerank + Complete Link Clustering*: Results from Clustering server after applying agglomerative clustering over Pagerank results

3.8 *Vector Space + Pagerank + Average Link Clustering*: Results from Clustering server after applying agglomerative clustering over Pagerank results

3.9 *Vector Space + Pagerank + Query Expansion*: Results from Query Expansion server after expanding the query using relevance feedback

All search requests are sent to the server via AJAX requests and the responses are

handled inside JavaScript.

**3.1.2 Application Layer (Controller layer):**

The search requests from the user are handled by the application layer which accepts GET and POST requests via AJAX calls from UI. There are six controller programs written in Java which performs the handling of AJAX calls.

*ServletHandler (Controller 1)*: Handles AJAX requests related to Google, Bing, Solr and HITS

*ClusterServletHandler (Controller 2)*: Handles AJAX requests related to K-means clustering

*ClusterServletHandler2 (Controller 3)*: Handles AJAX requests related to agglomerative clustering using single link

*ClusterServletHandler3 (Controller 4)*: Handles AJAX requests related to agglomerative clustering using complete link

*ClusterServletHandler4 (Controller 5)*: Handles AJAX requests related to agglomerative clustering using average link

*QueryExpansionServletHandler (Controller 6)*: Handles AJAX request related to Query Expansion

**3.1.4 DataAccessLayer:**

The requests handled via controllers are routed to appropriate servers and the response received are routed back to the Controller layer.

**3.1.5 Business layer:**

There are 6 servers running locally that contains the business logic for various relevance models.

Server 1: Solr server for handling indexing and Pagerank requests

Server 2: Python server for handling HITS/query expansion requests

Server 3: Python server for handling Flat Clustering requests

Server 4: Python server for handling the Agglomerative Clustering – single link

Server 5: Python server for handling the Agglomerative Clustering – complete link

Server 6: Python server for handling the Agglomerative Clustering – average link

All requests from the Data Access layer are routed to these six servers through the APIs exposed by these servers. JSON is chosen as the data exchange format for ease of implementation and support by various programming languages.

**3.2 Collaborations**

**3.2.1 Collaboration with student responsible for Indexing/Pagerank/HITS:**

Indexing and pageranking are done using Solr module. The server running the Solr module exposes an API for accessing the indexed results. Using the information provided by the student about the Solr API and its requirements, the API was accessed from the DataAccess Layer via an exclusive call to the Solr server. This API exposes various control parameters such as number of pages to return and the format of the response expected. Owing to the scope of the project to study the effects of various relevance models are their comparison with Google and Bing, the number of results fetched was limited to 15 per server.

Result format (Common results format used through out the project): The JSON results from the Solr contain various information about the pages being retrieved. The following fields are selected for display in the UI:

1. URL: the URL address of the page

2. Title: the Title of the page

3. Content: the HTML content of the page (with HTML tags removed) – truncated to 200 characters for display in the UI as a short description about the webpage

The HITS algorithm is implemented in python and hosted in a separate server by this student. So the information about this python API was collected from this student to make the API call. Since the HITS algorithm is dependent on the indexing results from Solr, results from Solr API was sent to this API via POST and the response was collected back.

**3.2.2 Collaboration with student responsible for Clustering:**

The clustering algorithms are implemented and the cluster information was stored in separate files. These files are given to me by the student responsible for clustering and they are read to find the most relevant cluster for the user query by analyzing the top 100 results from Solr. The top 15 results were shown in the UI based on the chosen cluster. So the results being displayed would be ordered based on their relevance to each other.

The response format used is similar to the format used for Solr API.

**3.2.3 Collaboration with student responsible for Query Expansion:**

The query expansion algorithm is implemented and hosted in the same server as HITS. The input to this Query Expansion API is the initial query string made by the user and the initial Solr output. Final output from the API is the expanded query string. This expanded string is then used to retrieve a new set of results from the Solr API.

**3.3 Comparison with Google and Bing:**

In order to compare the results with Google and Bing results, Google web search API and Bing Search API were used to retrieve their results. Following observations were made during the evaluation:

a. For certain queries which contained Artist or Song or Album names, our results from Pagerank/HITS/Clustering were closely in accordance with results from Google and Bing. We believe this is due to the selection of very good seed URLs to cover most of the albums and their artists. Also, from the type of pages that has been crawled, it is observed that the information about artists or their albums is static information and has good keywords. For example, popular pages about the band ‘The beatles’ contained keywords such as ‘beatles’ and their artists names which helped us to retrieve them and rank them higher.

b. For certain queries requesting albums or songs from a particular year, our search engine did not retrieve relevant pages in comparison to Google and Bing. Upon analyzing the results from Google and Bing with our document collection, it is observed that the websites that are popular for such information like billboard.com, youtube.com denied access to our crawler. Hence those pages were missing from our index.

c. Using seed URLs from Google and Bing proved useful in retrieving the top pages during the search. Especially, Wikipedia pages were immensely useful in improving our search results and were retrieved by our search engine despite the limited number of inlinks and outlinks in comparison to the entire web used by Google and Bing. This proved the effect of Vector Space model combined with Pagerank.

**3.4 Testing strategy:**

Testing was done throughout the development of the project to provide immediate feedback to students implementing the relevance models. Approximately 200 queries were used to test the results of various models. About 50 queries about artists, album names and song names were used in collaboration with the students building the relevance models and the rest of queries were generated based on various topics such ‘genre’, ‘lyrics’, ‘artist names’, ‘artist names + song names’, ‘bands’ etc.

When a particular page from the top results of Google/bing does not come up with our relevance model, the page was first checked for existence and if present, the page rank score of the page was compared with the top scoring page from our relevance model. In few cases, the score was too low due to insufficient inlinks and outlinks and in many cases, the page in question was not present in our collection.

In HITS and Query expansion testing, certain times, the algorithms did not converge within the given number of iterations and were communicated to the student in charge and algorithm parameters were appropriately improved.

**3.5 Results**

The queries for demonstration were chosen from the testing we have done and gave good results. Each query chosen was targeting a different area such as Genres, Lyrics, artists, song names, band names etc.

Query 1: *Beethoven songs*

Top 5 results from our model (after clustering) :

1. http://www.classicfm.com/composers/beethoven/
2. http://www.gramophone.co.uk/review/beethoven-complete-piano-sonatas-5
3. http://mp3raid0.com/download/beethoven.html
4. https://en.wikipedia.org/wiki/Beethoven
5. https://en.wikipedia.org/wiki/Ludwig\_van\_Beethoven

Top 5 results from Google:

1. https://www.youtube.com/watch?v=k\_UOuSklNL4

2. https://en.wikipedia.org/wiki/List\_of\_compositions\_by\_Ludwig\_van\_Beethoven

3. http://www.songfacts.com/artist-ludwig\_van\_beethoven.php

4. https://www.quora.com/What-are-Beethovens-most-famous-pieces

5. http://imslp.org/wiki/26\_Welsh\_Songs,\_WoO\_155\_(Beethoven,\_Ludwig\_van)

Top 5 results from Bing:

1. https://en.wikipedia.org/wiki/List\_of\_compositions\_by\_Ludwig\_van.

2. www.last.fm/music/Ludwig+van+Beethoven

3. www.youtube.com/playlist?list=PLEF4FCEA775F67968

4. www.thetoptens.com/best-beethoven-songs

5. www.lvbeethoven.com/Oeuvres/ListOpus.html

Query 2: *beat it michael jackson*

Top 5 results from our model (After Pagerank):

1. http://www.mtv.com/artists/michael-jackson/

2.http://www.metrolyrics.com/beat-it-lyrics-michael-jackson.html3\_download.html

3. http://www.lyricsfreak.com/m/michael+jackson/beat+it\_20092702.html

4. https://musicbrainz.org/artist/f27ec8db-af05-4f36-916e-3d57f91ecf5e

5. http://www.last.fm/tag/michael+jackson

Top 5 results from Google:

1. https://www.youtube.com/watch?v=Ym0hZG-zNOk

2. https://en.wikipedia.org/wiki/Beat\_It

3. https://www.youtube.com/watch?v=T2PAkPp0\_bY

4. http://www.azlyrics.com/lyrics/michaeljackson/beatit.html

5. https://www.youtube.com/watch?v=-PXnCiMinoc

Top 5 results from Bing:

1. www.youtube.com/watch?v=oRdxUFDoQe0

2. www.last.fm/music/Michael+Jackson/\_/Beat+It

3. https://en.wikipedia.org/wiki/Beat\_It

4. www.vevo.com/watch/michael-jackson/Beat-It/USSM20301089

5. www.metrolyrics.com/beat-it-lyrics-michael-jackson

Query 3: *rock bands*

Top 5 results from our model (after HITS):

1. https://en.wikipedia.org/wiki/Category:Lists\_of\_rock\_musicians\_by\_subgenre

2. https://en.wikipedia.org/wiki/Category:Lists\_of\_bands

3. http://www.ranker.com/crowdranked-list/the-best-rock-bands-of-all-time

4. http://www.ranker.com/list/best-hard-rock-bands-and-artists/reference

5. https://en.wikipedia.org/wiki/Category:Lists\_of\_heavy\_metal\_bands

Top 5 results from Google:

1. http://www.spin.com/2015/10/the-50-best-rock-bands-right-now/

2. http://ultimateclassicrock.com/artists/

3. http://www.thebestschools.org/magazine/100-best-rock-bands/

4. http://www.leatherrockbands.com/

5. http://www.thebestschools.org/magazine/100-best-rock-bands-part-4/

Top 5 results from Bing:

1. https://en.wikipedia.org/wiki/List\_of\_alternative\_rock\_artists

2. https://en.wikipedia.org/wiki/Rock\_music

3. www.leatherrockbands.com

4. www.spin.com/2015/10/the-50-best-rock-bands-right-now

5. www.ranker.com/crowdranked-list/the-best-rock-bands-of-all-time

**4. CLUSTERING – Varun Muthanna**

**Input:**

Input is a file with two columns [URL, Content]. The URL reperesent the id and the content is the content of the webpage.

**Output:**

Output of this module is a file which contains the URL and the respective cluster it belongs to.

**Collaboration:**

**With the student who Implemented Indexing:**

We together developed a code to get the data from the solr was fetched and it was parsed such that to generate a file with just the URL and the content of the webpage, each represented in a single line.

**With the student who Implemented the UI:**

The clustering helps in finding similar pages based on the contents of the webpage. So we decided up on the using this property by first retrieving the query results from the pageranking. The top 100 of the pageranking results are checked with the clustered results by comparing how many of these webpages belonged to same clusters and we consider the cluster which contains the maximum webpages and display 15 of the webpages on to the UI.

**Models Used:**

I have used both Flat Clustering and Agglomerative Clustering.

**Flat Clustering:**

Kmeans algorithm was used to implement the flat clustering.

Tool used: Python Sklearn library was used to generate the clusters.

Parameter Values: Number of cluster = 500 max\_iterations = 100

Taking the time complexity and the accuracy we were obtaining we decided 500 clusters as optimal.

**Agglomerative Clustering:**

Three Agglomerative clustering have been implemented.

Single-link, Complete-link and Average methods are shown in the UI

Tools used: Python SciPy library was used to generate the clusters.

Python Scipy library generates a file which contains the URL ids which are connected in the tree format. A Java program was written to convert this format to put each URL into certain clusters. For this I have started from the leaves which are the URLs and once it reaches certain height in the tree it is put into a cluster.

Running agglomerative clustering requires O(n2) memory complexity. With our total of 170000 webpages we had a memory constraint. So this 170000 webpages were divided into 4 groups and are clustered. And each part has its own group of clusters.

**Problems Faced:**

Time and Memory complexity of the algorithms in use is the major problem faced here. A lot of experimentation with different libraries were made to finally cluster the webpages. And few alternative methods as mentioned above had to be used. Agglomerative Clustering using centroid method was taking a long time (close to 6 hours for first 50000 webpages) and crashing at the end and hence not shown in the UI.

**Results:**

Around 30-40 queries were used to test the clustering results to get the parameters right. Below I have mentioned two query results.

**Query: Beethoven songs**

kmeans

http://www.classicfm.com/composers/beethoven/

http://www.gramophone.co.uk/review/beethoven-complete-piano-sonatas-5

http://mp3raid0.com/download/beethoven.html

https://en.wikipedia.org/wiki/Beethoven

https://en.wikipedia.org/wiki/Ludwig\_van\_Beethoven

Single-link

https://en.wikipedia.org/wiki/Beethoven

http://www.classicfm.com/composers/beethoven/

http://www.gramophone.co.uk/review/beethoven-lieder-0

http://www.gramophone.co.uk/review/beethoven-complete-piano-sonatas-5

http://mp3raid0.com/download/beethoven.html

Complete-link

http://www.gramophone.co.uk/review/beethoven-lieder-0

http://mp3-download.tubidy4u.com/search/classical-songs.html

http://www.classicfm.com/composers/beethoven/album-reviews/

http://www.classicfm.com/composers/beethoven/

http://www.gramophone.co.uk/review/beethoven-complete-piano-sonatas-5

Average link

http://www.gramophone.co.uk/review/beethoven-complete-piano-sonatas-5

http://mp3raid0.com/download/ludwig\_van\_beethoven.html

http://www.classicfm.com/composers/schubert/guides/schubert-20-facts-about-great-composer/franz-schubert-gustav-klimt-7/

http://www.classicfm.com/composers/tchaikovsky/news/

http://www.classicfm.com/composers/beethoven/

**Query: Rock bands**

kmeans

http://www.ranker.com/crowdranked-list/the-best-rock-bands-of-all-time

http://www.ranker.com/list/the-best-black-rock-bands/stevie-wonder-on-a-keytar

https://en.wikipedia.org/wiki/List\_of\_Christian\_rock\_bands

http://www.ranker.com/list/artists-who-have-inspired-the-best-covers/rockboy

http://www.ranker.com/list/greatest-musical-artists-of-all-time/ranker-music

Single-link

http://www.ranker.com/list/best-hard-rock-bands-and-artists/reference

http://www.ranker.com/list/the-best-black-rock-bands/stevie-wonder-on-a-keytar

https://en.wikipedia.org/wiki/Category:Lists\_of\_rock\_musicians\_by\_subgenre

https://en.wikipedia.org/wiki/Category:Lists\_of\_bands

http://www.ranker.com/crowdranked-list/the-best-rock-bands-of-all-time

Complete-link

https://en.wikipedia.org/wiki/Category:Lists\_of\_bands

https://en.wikipedia.org/wiki/Category:Lists\_of\_heavy\_metal\_bands

http://www.ranker.com/crowdranked-list/best-hair-metal-bands-of-all-time

http://www.ranker.com/crowdranked-list/best-heavy-metal-bands-that-i-know

http://www.ranker.com/crowdranked-list/20-greatest-singers

Average-link

https://en.wikipedia.org/wiki/Category:Lists\_of\_bands

https://en.wikipedia.org/wiki/Category:Lists\_of\_heavy\_metal\_bands

https://en.wikipedia.org/wiki/Category:Lists\_of\_rock\_musicians\_by\_subgenre

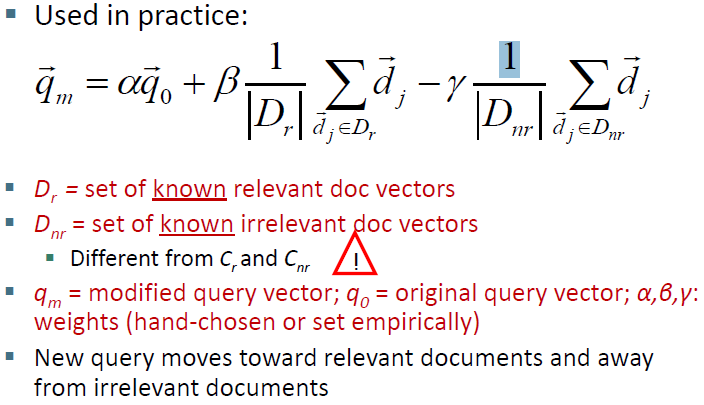
http://www.ranker.com/crowdranked-list/the-best-rock-bands-of-all-time

http://www.ranker.com/list/best-hard-rock-bands-and-artists/reference

**5. Query expansion and relevance feedback – Harshil Shah**

Rocchio Algorithm

The following algorithm was used to test the Rocchio Algorithm:



Following was the approach used to arrive to a set of weights that fetches a more relevant modified query:

1. Get the query and the results from Solr for the results for the query
2. Fetch the the results from Google
3. Label each result from our search query as either relevant for non-relevant based upon the results of Google.
4. Increment beta, decrement gamma keeping alpha constant to get higher ratio.
5. Repeat step 4 for a fixed number of iterations or until weights stabilize
6. Output the modified query from the query vector

The 20 queries selected to test the algorithms:

The queries were selected such that they are either band names, artists, genre, singles or albums, thus covering all major parts of music search queries.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linkin | Beyonce | Beatles | Radiohead | Adele |
| Hey jude | Taylor swift | Hans zimmer | Marshall mathers | Smells like |
| Foo fighters | Santana | Californication | Beethoven songs | beat it |
| Rock bands | Hip hop | Punk | Rap | Jazz |

The results were not promising and a universal set of weights could not be found. Also the running time for convergence was too high. Because of this we shifted to pseudo relevance feedback.

Pseudo relevance feedback:

Here we use local feedback strategies to expand the query with terms correlated to the query terms.

We can find those co-related terms from the local clusters build from the local document set.

Three types of clustering are:

1. Association cluster:

The idea is that stems which co-occur frequently inside documents have a synonymity association

1. Metric cluster:

The idea is that the stems which occur far apart in the document are less co-related to the terms that occur closer like in the same sentence.

1. Scalar cluster:

The idea is that two stems are more co-related if they have similar neighborhood.

The algorithm for this is for each stem in the query add the closest neighbors from the cluster (association /metric /scalar)

Some query expansion results with metric clusters

|  |  |
| --- | --- |
| Initial Query | Expanded Query |
| Linkin | linkin park retriev |
| hey jude | Hey jude nah |
| Jazz | Jazz smooth chart |
| Adele | Adel depp hello |
| rock bands | Rock band metal greatest music |
| Beethoven songs | Beethoven songs van s classic |
| Beat it Michael Jackson | Beat it Michael Jackson want just version |
| Foo fighters | Foo fighter taylor itun |
| Beyonce | Beyonc play download |
| Taylor swift | Taylor swift download |
| Santana | Santana view carlo |
| Hip hop | Hip hop c |

Some challenges:

1. Scalar clustering took too long to run so we can’t perform them on the fly
2. The query words need to spell correctly so as to get the correct stems.
3. The expanded query is of stems so for proper nouns like “ADELE” the stem is “Adel” so the results doesn’t have Adele results.
4. The documents doesn’t have correct spelled words always so for example in the above example of hip hop the expanded query had “c” in it

Collaboration with GUI:

1. The API calls were made to run the query expansion.
2. The input for the API call were the initial query and SOLR output.
3. After the query expansion the expanded query was returned as an output to the API call.

Coding:

1. Python was the language selected to implement.
2. Pythons NLTK WORNET Porter stemmer was used for stemming.

Project Demo Selection:

For the demo, the query “rock bands” was selected because it is a genre search and the user of looking for the back bands from history and the best. The algorithm used for query expansion is “metric clustering”.