**The University of Texas at Dallas**

**CS 6322**

**Information Retrieval**

**Spring 2016**

**Class Project Proposal**

*Project TITLE: Search Engine for Movies*

*Group: No 3*

*Students: Ekal Golas,* [*exg140230@utdallas.edu*](mailto:exg140230@utdallas.edu)

*Asha Mary Thomas,* [*axt143530@utdallas.edu*](mailto:axt143530@utdallas.edu)

*Navaneeth Venugopala Rao,* [*nbv140130@utdallas.edu*](mailto:nbv140130@utdallas.edu)

*Saumya Ann George,* [*sxg151030@utdallas.edu*](mailto:sxg151030@utdallas.edu)

*Kevin George,* [*kag140330@utdallas.edu*](mailto:kag140330@utdallas.edu)

**Problem Statement: Search Engine for Movies**

In order to implement our solution, we chose the following 5 sites – movies.com, metacritic.com, imdb.com, rottentomatoes.com and wikipedia.org.

The responsibilities for the project are as follows –

**Crawling** – Navaneeth Venugopala Rao

**Indexing and Relevance** – Saumya Ann George

**Clustering** – Kevin George

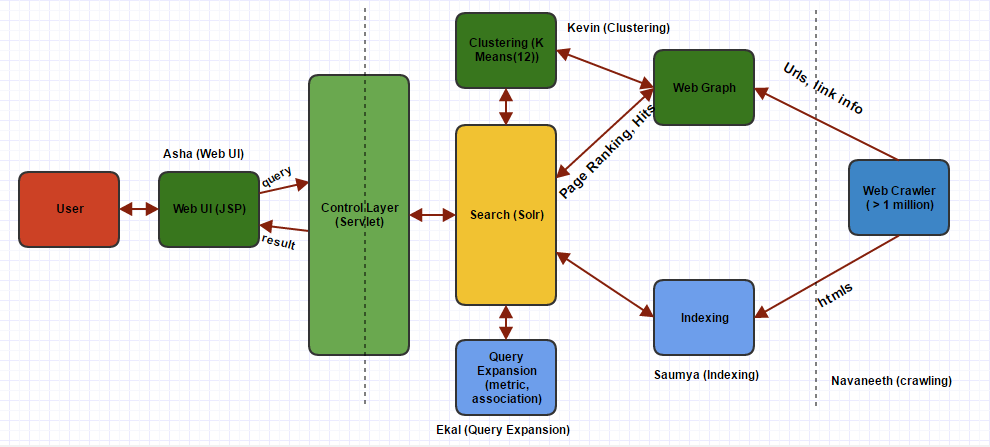
**UI** – Asha Mary Thomas

**Query** **Expansion** – Ekal Golas

**Navaneeth Venugopala Rao – Crawling**

We crawled using Apache Nutch which fed input to the Solr for indexing and created webgraphs for further processing.

**Application Architecture:**



**Movie Search Engine Architecture**

|  |
| --- |
| **Lucene:** Java library for indexing and searching, powerless and fast, etc.  **Solr:** http-based index and seach server, Web-based administration panel, etc.  **Nutch:** Internet search engine software, web Crawler, powerful for vertical search engine, etc. |

**Key Learnings:**

1. Implementing Open Source software.
2. Handling large amount of data.

**Navaneeth Venugopala Rao – Crawling**

**Technology Stack**: Apache Nutch.

We crawled around 1,022,375 web pages.

The number of outgoing URLs is 2,129,447.

The number of fetched web pages is 536,458.

We used Apache Nutch to crawl movies.com, metacritic.com, imdb.com, rottentomatoes.com and wikipedia Site.

We gave around 100 URLs as seed URL to start out crawling. These URLs were those of movies, actors, directors, producers, reviews.

The sample list of seed URLs are as follows:

http://www.movies.com/search

http://www.rottentomatoes.com/celebrity/danny\_boyle

https://en.wikipedia.org/wiki/Godzilla\_(1998\_film)

http://www.imdb.com/name/nm3053338/?ref\_=tt\_ov\_st\_sm

http://www.imdb.com/title/tt0172495/?ref\_=nv\_sr\_1

https://en.wikipedia.org/wiki/Black\_Hawk\_Down\_(film)

http://www.movies.com/deadpool/m67637

http://www.imdb.com/title/tt0436697/?ref\_=nm\_knf\_t1

http://www.metacritic.com/movie/harry-potter-and-the-goblet-of-fire

http://www.imdb.com/chart/moviemeter?ref\_=nv\_mv\_mpm\_7

http://www.imdb.com/name/nm0447695/?ref\_=nv\_cel\_dflt\_1

http://www.movies.com/actors/morgan-freeman/morgan-freeman-movies/p191454

http://www.rottentomatoes.com/top/

http://www.imdb.com/title/tt0050083/?ref\_=nv\_sr\_1

http://www.metacritic.com/movie/gods-of-egypt

http://www.rottentomatoes.com/celebrity/tom\_hanks

http://www.imdb.com/title/tt1735898/?ref\_=nv\_sr\_1

http://www.imdb.com/genre/action

http://www.imdb.com/name/nm0544718/?ref\_=nv\_sr\_6

http://www.imdb.com/title/tt0145487/?ref\_=nm\_knf\_t3

http://www.rottentomatoes.com/m/toy\_story\_3/

http://www.imdb.com/title/tt4833824/

https://en.wikipedia.org/wiki/Jaws\_(film)

http://www.imdb.com/title/tt1409024/?ref\_=nm\_flmg\_act\_11

https://en.wikipedia.org/wiki/Indecent\_Proposal

http://www.rottentomatoes.com/top/bestofrt/top\_100\_action\_\_adventure\_movies/

http://www.metacritic.com/browse/movies/score/metascore/all/filtered?sort=desc

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

http://www.movies.com/actors/helen-mirren/helen-mirren-movies/p286223

http://www.imdb.com/chart/top?ref\_=nv\_mv\_250\_6

http://www.imdb.com/name/nm0451321/?ref\_=fn\_al\_nm\_1

http://www.imdb.com/name/nm0000190/?ref\_=nv\_sr\_1

**Navaneeth Venugopala Rao – Crawling**

https://en.wikipedia.org/wiki/Steven\_Spielberg\_filmography

http://www.imdb.com/name/nm0362766/?ref\_=nv\_sr\_1

http://www.metacritic.com/person/scarlett-johansson

https://en.wikipedia.org/wiki/Demi\_Moore

http://www.imdb.com/title/tt0266915/?ref\_=nm\_knf\_t3

http://www.imdb.com/name/nm0000226/?ref\_=tt\_ov\_st\_sm

http://www.rottentomatoes.com/m/steve\_jobs\_2015/

http://www.imdb.com/title/tt0414387/?ref\_=nv\_sr\_2

http://www.rottentomatoes.com/m/godfather/

http://www.metacritic.com/movie

https://en.wikipedia.org/wiki/Ridley\_Scott

http://www.metacritic.com/browse/movies/score/metascore/year/filtered?sort=desc&year\_selected=2005

http://www.rottentomatoes.com/top/bestofrt/top\_100\_horror\_movies/

https://en.wikipedia.org/wiki/Matt\_Damon\_filmography

https://en.wikipedia.org/wiki/Ashton\_Kutcher

http://www.metacritic.com/browse/movies/score/metascore/90day/filtered

http://www.rottentomatoes.com/m/star\_wars\_episode\_vii\_the\_force\_awakens/

http://www.metacritic.com/movie/kung-fu-panda-3

http://www.imdb.com/search/title?genres=action&title\_type=feature&num\_votes=1000,&sort=user\_rating,desc

http://www.metacritic.com/movie/the-social-network

http://www.rottentomatoes.com/m/slumdog\_millionaire

http://www.imdb.com/title/tt0993846/?ref\_=nm\_knf\_t2

http://www.rottentomatoes.com/top/bestofrt/top\_100\_comedy\_movies/

http://www.metacritic.com/movie/the-jungle-book-2016

http://www.imdb.com/name/nm0000545/?ref\_=nv\_sr\_1

https://en.wikipedia.org/wiki/Woody\_Harrelson

http://www.metacritic.com/movie/match-point

http://www.imdb.com/name/nm0000329/?ref\_=nv\_sr\_2

Nutch generates 3 folder when it crawls:

1) **CrawlDb** - It contains all the link parsed by the Nutch.

2) **LinkDB** - It contains for each URL the outgoing and the incoming URLs.

3) **Segment** - It contains the list of URLs to be crawled or being crawled.

Nutch crawl can be broken into the following steps:

1) Inject seed URLs into Nutch crawl database. Nutch Crawl Database contains all the links that has been parsed or is to be parsed by Nutch.

2) Then generate segments for the list of injected URLs. This step ensures whether the link given for crawling has already been parsed or not and if it follows the regex of allowed URLs.

3) Nutch then start to crawl the links in the segment .In this step Nutch parses the webpage to look for the contents as well as any outgoing URLs in the page.

4) After Nutch has crawled a set of URLs those URLs are merged with the crawl database. For each URL parsed Nutch makes an entry to the link database of the URLs a particular URL is referring to.

**Navaneeth Venugopala Rao – Crawling**

Nutch command used to crawl:

./crawl -i -D Solr.server.URL=http://localhost:8983/Solr/ URLs/ TestCrawl/ 3

crawl [-i|--index] [D “key=value”] <Seed Dir> <Crawl Dir> <Num Rounds>

-i|- - index Indexes crawl results into a configured indexer

-D A Java property to pass to Nutch calls

Seed Dir Directory in which to look for a seeds file

Crawl Dir Directory where the crawl/link/segments dirs are saved

Num Rounds The number of rounds to run this crawl for

We used Apache Solr to index documents crawled by Nutch .We used one of the Nutch command mentioned above which crawls and directly send the contents parsed from URL by Nutch to Solr for indexing. In order to allow Nutch to directly send its contents to Solr we have to change Solr&#39;s schema.xml for the collection we were using in Solr. The command to directly send crawled contents by Nutch directly to Solr is as follows:

./crawl -i -D Solr.server.URL=http://localhost:8983/Solr/ URLs/ TestCrawl/ 3

In the above command -i tells Nutch to sends it’s parsed content to the assigned indexer which in Solr in our case.

To provide URLs for PageRank and hits algorithm we generated a dump out of linkdb of Nutch.

http:// http://devour.com/video/jurassic-park-without-the-dinosaurs/ Inlinks:

fromURL: http://www.movies.com/movie-news/today-movie-culture-deadpool-goes-39back-to-future39-39jurassic-park39-without-dinosaurs-more/20193 anchor: via Devour

http:// http://schedule.sxsw.com/2012/events/event\_FS12415 Inlinks:

fromURL: http://www.movies.com/movie-news/sxsw-60-seconds-lsquosleepwalkrsquo-lsquochasing-icersquo-picked-up-matthew-lillard-cheers-indonesian-midnight-madness/6989 anchor: Fat Kid Rules the World

http://http://www.movies.com/movie-news/when-can-i-watch-that-with-my-kids-archive Inlinks:

fromURL: http://www.movies.com/movie-news/hermione-granger-role-model/10847 anchor: When Can I Watch That with My Kids?

We implemented a java program to read the above dump and generate for each URL parsed a URL id and to create a graph from it .We parsed the dump to write two files one named docIDURL.txt and the other adjFile.txt.

Command for generating dump from linkdb is as follows:

Command pattern - ./Nutch readlinkdb <linkdb> -dump <out\_dir>

Command used - ./Nutch readlinkdb test/linkdb -dump linkdbDump

docIDURL.txt - Contains the URL and URL id mapping.

adjFile.txt - It contains the directed graph that is computed out of the dump.

**Navaneeth Venugopala Rao – Crawling**

The above 2 files were used to generate PageRank and hits score.

In order to implement the relevancy model and for clustering of URLs parsed we needed the contents of each URL parsed. The contents of each URL parsed in the segments folder of Nutch can be dumped by using readseg command of Nutch .The command is as follows:

Command pattern: /Nutch readseg -dump <segment\_dir> <output>

Command used: /Nutch readseg -dump test/segments/ (segment Num) segmentParsedDump

As while crawling we configured it for 3 runs it generated 3 segment folders .So we ran the above command for each segments to create the dump. But as those dump was not in a good format to be used we had to preprocess it for further use. So we wrote a java program to read the dump. For each URL and its content in dump we created a separate file .The file content looked as follows:

URL: https://en.wikipedia.org/wiki/Batman\_v\_Superman:\_Dawn\_of\_Justice

Parsed Content :: <Contents>

We have like 1,022,375 files created at the end of this process. The above files created were used for clustering purpose.

**Saumya Ann George – Indexing and Relevance**

**Indexing and Relevance**

Apache Solr is used for indexing the crawled database.

The output of the crawler contains 3 directories crawldb, linkdb and segments. These are fed as input to Apache Solr using nutch command as shown below:

Bin/nutch index -DSolr.server.url= http://ec2-54-191-183-57.us-west-2.compute.amazonaws.com:8983/Solr/ TestCrawl//crawldb -linkdb TestCrawl//linkdb -dir TestCrawl//segments/

where TestCrawl is the folder which contains crawled data.

TF-IDF based relevance model is provided by Apache Solr for scoring the web pages and we made use of this score as vector space relevance model.

To develop relevance model based on Page Ranking, we used inbuilt Nutch command. The damping factor used is 0.85.

Following were the steps we followed in order to push PageRank scores into crawled database so that query results will display web pages in the descending order of page ranking values.

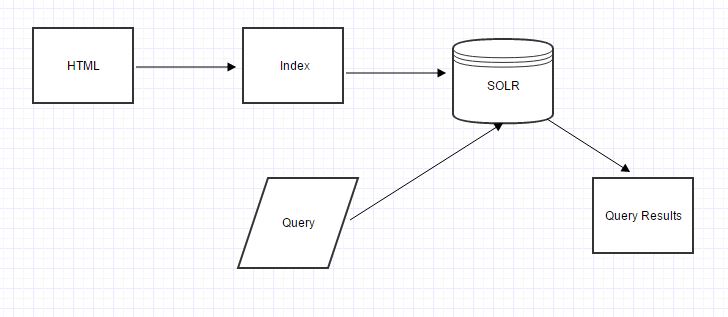


Fig. Flow for Indexing HTML files and Querying

Create a web graph from the set of URLS fetched by the crawler

bin/nutch org.apache.nutch.scoring.webgraph.WebGraph -segmentDir TestCrawl/segments/ -webgraphdb crawl/webgraphdb

1. Run PageRank algorithm iteratively until the score converges

bin/nutch org.apache.nutch.scoring.webgraph.LinkRank -webgraphdb crawl/webgraphdb/

1. Update the page rank score into the crawled database

bin/nutch org.apache.nutch.scoring.webgraph.ScoreUpdater -crawldb crawl/crawldb -webgraphdb crawl/webgraphdb/

1. Finally, index the crawled html files using Solr and nutch command:

bin/nutch index -DSolr.server.url=http://localhost:8983/Solr/ TestCrawl//crawldb -linkdb TestCrawl//linkdb -dir TestCrawl//segments/

**Saumya Ann George – Indexing and Relevance**

Total Number of links = 860527

Total Number of nodes = 258565

The largest number of ingoing links = 82570

The largest number of outgoing links = 65

We implemented our own program for calculating HIT score which includes hub score and authority score. Based on the query provided by the user, root set of web pages are collected based on page rank results. The root set is converted to a base set by adding out link and inlinks of the the root set. After which, HIT score calculation is done and based on authority web pages are sorted and returned to the User Interface for display.

Among the web pages we crawled,

1. highest Hub score was assigned to url: <http://www.movies.com/movie-news/cine-latino-seven-badass-actors-over-40/9828>
2. highest authority score was assigned to url: <https://twitter.com/moviesdotcom>

Maximum HubScore =0.31443

Maximum Authority score = 0.28223

Topic Based Page Ranking

For the purpose of testing Topic based page ranking, I chose 2 topics among movies:

1. Action movies
2. Horror movies

The results for **Action movies** are as follows from our search engine: (included corresponding movie title and PageRank score for the webpage)

http://www.imdb.com/title/tt4790268/

*"title": "Never Back Down: No Surrender (2016) - IMDb",*

*PageRank score = 0.28449315 🡪 Highest page rank for action movies*

http://www.imdb.com/title/tt3991302/

*"title": "Rumble (2016) - IMDb"*

*PageRank score = 0.27642596*

http://www.imdb.com/title/tt3387710/

*"title": "Number One Contender (2014) - IMDb"*

*PageRank score = 0.26670334*

http://www.imdb.com/title/tt1155076/

*"title": "The Karate Kid (2010) - IMDb"*

*PageRank score = 0.26580232*

http://www.imdb.com/title/tt0458339/

*"title": "Captain America: The First Avenger (2011) - IMDb"*

*PageRank score = 0.25817215*

http://www.imdb.com/title/tt0478970/

*"title": "Ant-Man (2015) - IMDb"*

*PageRank score = 0.25817215*

The results for **Horror movies** are as follows from our search engine: (included corresponding movie title and PageRank score for the webpage)

http://www.imdb.com/title/tt1786503/

*"title": "Haunted Maze (2013) - IMDb"*

**Saumya Ann George – Indexing and Relevance**

*PageRank Score = 0.4814272 🡪highest page rank for horror movies*

http://www.imdb.com/title/tt3702668/

*"title": "Axeman 2: Overkill (2016) - IMDb"*

*PageRank Score= 0.43667218*

http://www.imdb.com/title/tt4424292/

*"title": "Crazy Lake (2016) - IMDb"*

*PageRank Score =0.43629628*

http://www.imdb.com/title/tt5006566/

*"title": "Dragon No. 6 (2016) - IMDb"*

*PageRank Score=0.41692817*

http://www.imdb.com/title/tt3249124/

*"title": "Hitchhiker Massacre (2016) - IMDb"*

*PageRank Score = 0.40799996*

http://www.imdb.com/title/tt2245119/

*"title": "Scars (2012) - IMDb"*

*PageRank Score=0.40184653*

In collaboration with the person responsible for UI and clustering, we tested 60 movie names and observed that the results of the query gets better after applying HIT score and clustering.

**Kevin George – Clustering**

**Clustering**

Performed K-Means clustering with k as 12.

In order to perform clustering of html documents, urls of the documents and its contents were required. These details are fetched from solr, parsed and subjected to K Means clustering .On clustering based on contents in the documents, each URL will be uniquely assigned to one cluster. The url and the cluster number of every document is stored in a flat file before searching starts.

Clustering identifies documents which are similar based on their content. Hence documents in one cluster are similar within themselves and much different from documents in other clusters.

The results of clustering are incorporated in the relevance models. Based on results from solr which are ordered on the basis of higher page ranks, documents are further evaluated for their clusters. The most relevant cluster is identified and documents which belong to the same cluster are pushed up when clustering results where showed up. We had like 207303 records created after crawling. The above files created were used for clustering purpose. We have used only one flat clustering method and its k -means clustering

Accessing Clustering Information: We maintained our clustering information in a list of files which contained URL and cluster number of that URL. So for all the urls in the result we collected the cluster id associated with it. For each cluster id generated we have maintained a list of urls near the centroid so we just include those results in our result set if they are not there.

**K Means Algorithm Description**

Given a set of observations (x1, x2, …, xn), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) setsS = {S1, S2, …, Sk} so as to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

\underset{\mathbf{S}} {\operatorname{arg\,min}}  \sum_{i=1}^{k} \sum_{\mathbf x \in S_i} \left\| \mathbf x - \boldsymbol\mu_i \right\|^2 

where μi is the mean of points in Si.

**Choosing – K**

The correct choice of k is often ambiguous, with interpretations depending on the shape and scale of the distribution of points in a data set and the desired clustering resolution of the user. In addition, increasing k without penalty will always reduce the amount of error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster. Better results were achieved when K value was 12.

**Kevin George – Clustering**

Pseudo code for Extracting contents from index files of solr

1. Specify path to file and read contents from file
2. While until there are no more documents
   1. Read every record and look for ‘id’ section and save it
   2. Extract the details in ‘content’ section and save it
   3. Process the content to remove html tags
   4. Save a record with <id> , <content>
3. End the program

Pseudo code for performing K means clustering

1. Read the file with records saved in format <id> , <content>
2. While Until there are no more records in file
   1. Perform k means clustering on the content of data with k value of 12 and divide the content into clusters.
   2. Save the clusters with <id> , <cluster number> format
3. End the program

Testing

20 queries were used to test the results of clustering. These are the queries I used.

Three sample queries

Rush Hour

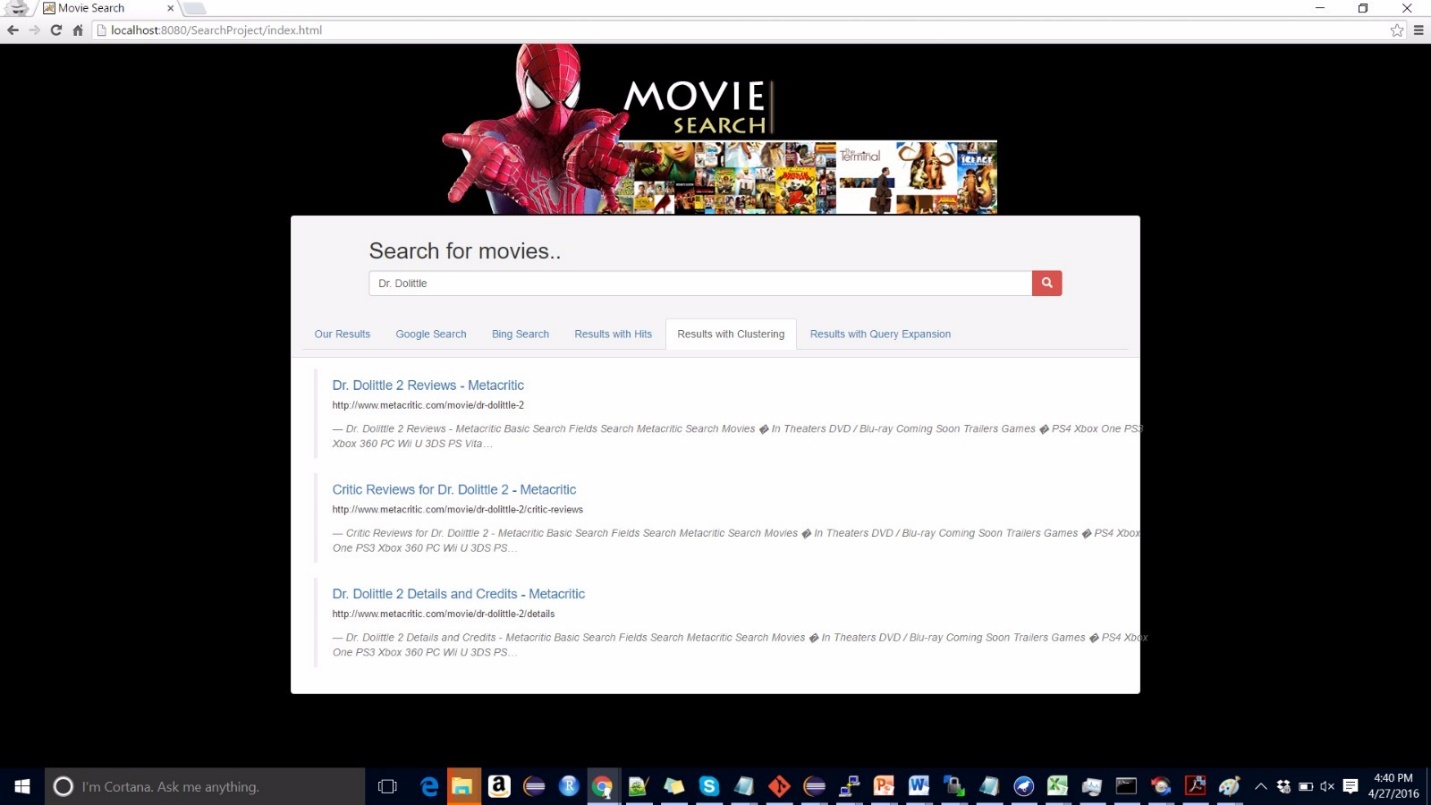
Titanic

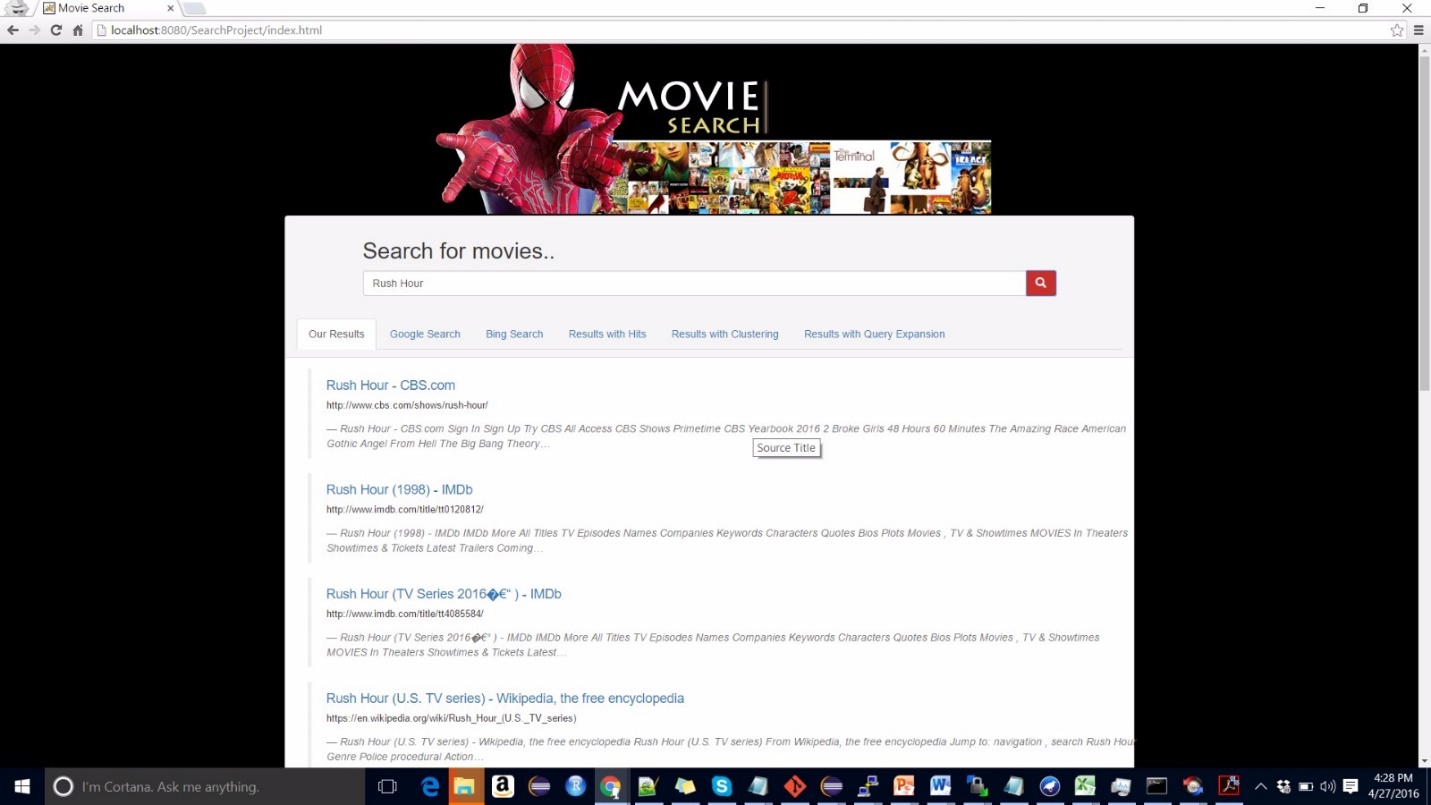
Batman

The criteria for selecting the movie are based on popularity of movies.

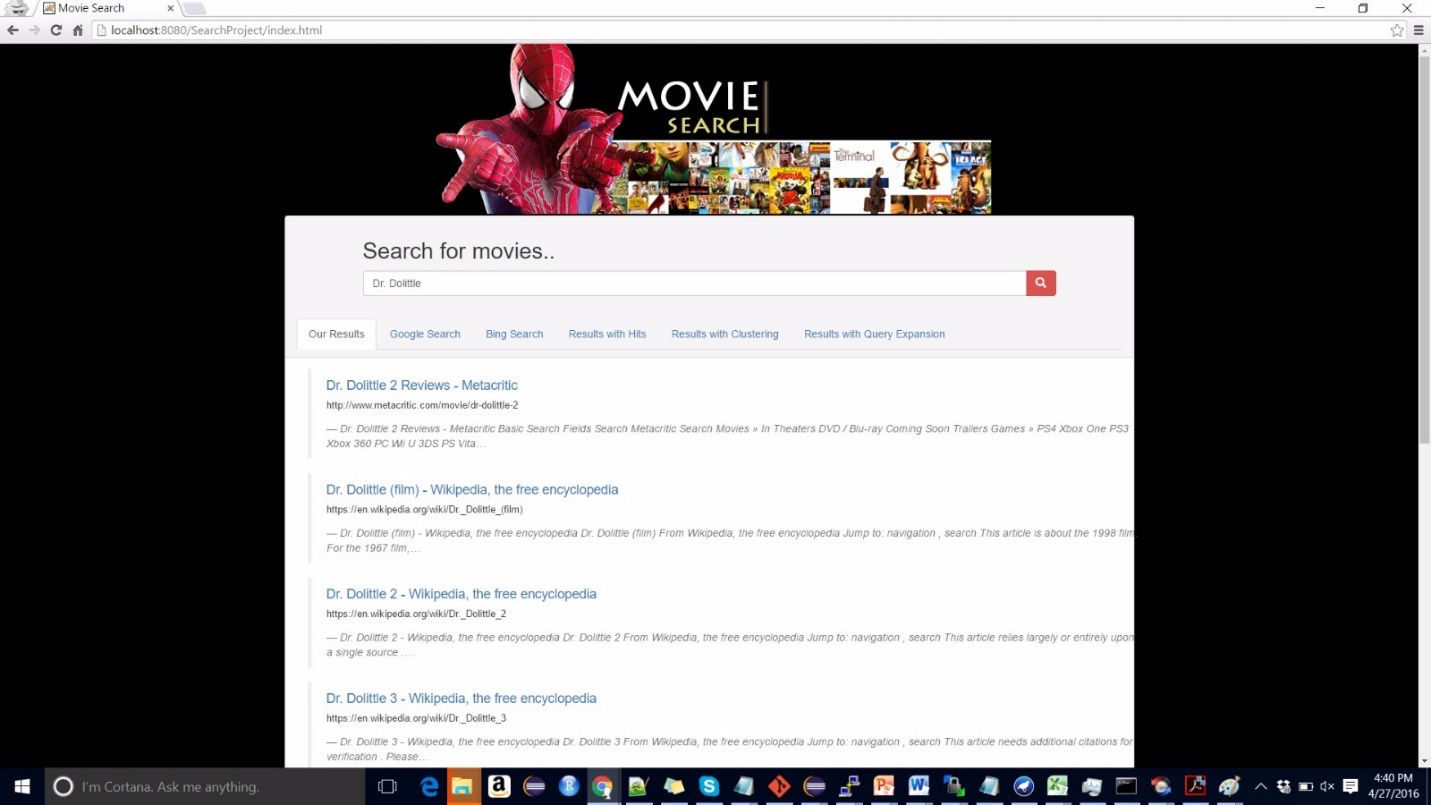
After clustering top ranked result pages belonging to same cluster were displayed and it was relevant.

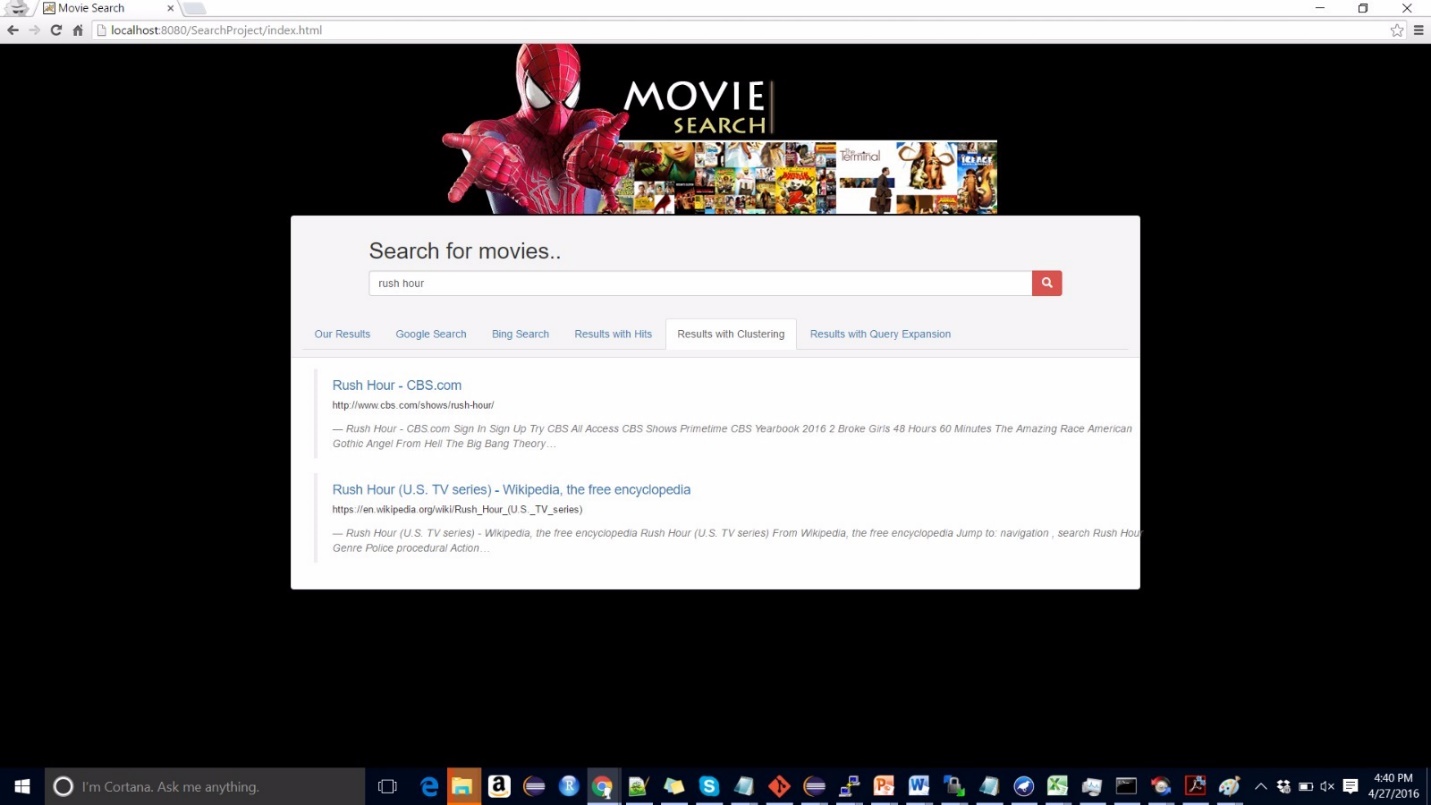
**Kevin George – Clustering**





**Kevin George – Clustering**

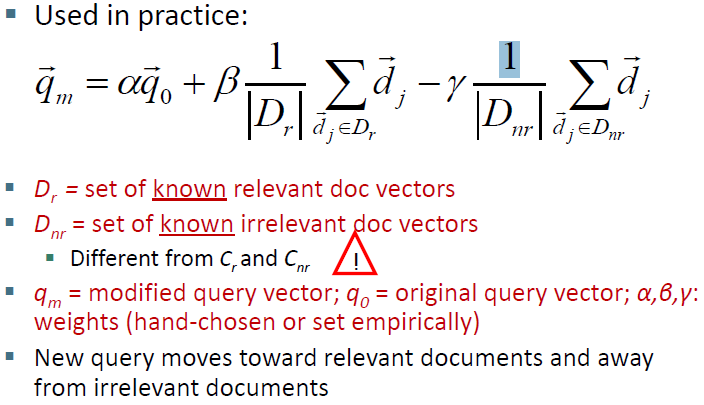




**Ekal Golas – Query Expansion**

# Rocchio Algorithm

Following equation was used to test the Rocchio algorithm:



In the above image, 20 queries were fed as q0 and the equation was tested by modifying various values of alpha, beta and gamma in the equation so that a set of weights could be found that can separate relevant documents from the irrelevant documents.

The 20 queries selected were as follows:

1. 4 queries with one keyword as the query for data that was crawled
   1. kung
   2. avengers
   3. deadpool
   4. godfather
2. 4 queries with two keywords as the query for data that was crawled
   1. kung fu
   2. batman superman
   3. marvel avengers
   4. black hawk
3. 4 queries containing stop words, special characters, spelling mistakes
   1. kun fu
   2. batmn
   3. batmn superman
   4. blck hwk

**Ekal Golas – Query Expansion**

1. 4 queries with keywords on the data that was not crawled in the domain
   1. khiladi
   2. casablanca
   3. dr dolittle 4
   4. eega
2. 4 queries outside the domain of movies
   1. fifa
   2. friends
   3. doom
   4. harry potter book

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

1. Fetch the results from Solr
2. Fetch results from Google and Bing
3. For each result in our search engine in the first page, check if it exists in the Google and Bing results. If yes, get the rank of the result, else get 0.
4. Keeping alpha as a constant, increment beta and decrement gamma to have a higher ratio. Obtain a modified query vector and test it
5. Repeat step 4 for a fixed number of iterations until weights stabilize
6. Output the final modified query from the query vector

The approach was used for the 20 queries mentioned above. The results obtained were not very promising and universal set of weights could not be arrived. Following were some of the findings:

1. Some queries fetched no documents – like “khiladi”, “eega” etc. which meant that the data we crawled did not contain neither the pages nor any reference to these words. In such cases, relevance model did not work
2. Some queries fetched all irrelevant documents – like “harry potter book” fetched results for harry potter movie. So the modified query was “harry potter” / “harry potter movie”
3. Some queries fetched partial results – like “doom” which is in the domain of games, but fetched results on trailers etc. for the game but also irrelevant results like “doctor doom”. For queries like these a set of weights were obtained and a more relevant query was fetched, but the set of weights differed for every query in this set. Modified query in this case was “doom PC”.
4. Some queries fetched very relevant results – like “kung fu” which fetched the kung fu panda and kung fu cowboy movies. In such cases, weights stabilized too early and so the modified query was not much relevant

Other challenges were:

1. Stemming did not work accurately with spelling mistakes. (WordNet Stemmer)
2. Performance issues as too much time was taken for some queries to get the stabilized weights, hence could not be performed on the fly
3. Each query needed its own unique set of weights

**Ekal Golas – Query Expansion**

Due to the above issues, it was decided not to include the code for Rocchio algorithm for expanding the query, rather the results of the observations were used in the pseudo relevance feedback approaches to filter out the stop words, avoid spelling mistakes etc.

# Pseudo relevance feedback

Based upon the observations made, a list of 54 queries were obtained as follows:

1. 18 queries with one keyword
   1. 6 queries for association clusters
   2. 6 queries for metric clusters
   3. 6 queries for scalar clusters
2. 18 queries with two keywords
   1. 6 queries for association clusters
   2. 6 queries for metric clusters
   3. 6 queries for scalar clusters
3. 18 queries with stop words, two or more keywords, ambiguous terms
   1. 6 queries for association clusters
   2. 6 queries for metric clusters
   3. 6 queries for scalar clusters

Following were the results obtained for each unique query (Each query run 3 times with 3 different cluster methods as mentioned above):

|  |  |  |  |
| --- | --- | --- | --- |
| **Query** | **No. of relevant docs in first page** | **Most relevant expanded query** | **Cluster method used** |
| Kung | 10 | Kung discuss cowboy | Metric, association |
| Kung fu | 10 | Kung fu trailer panda | Metric, association, scalar |
| Deadpool | 10 | Deadpool trailer tbd | Metric, scalar |
| Batman | 9 | Batman superman justice | Association |
| Batman superman | 2 | batman superman dawn affleck trailer justice ben | Metric |
| Avengers | 8 | Avengers marvel official | Scalar, Association |
| Marvel avengers | 5 | marvel avengers entertainment heroes comics top alliance | Metric |
| Marvel the avengers | 5 | marvel the avengers entertainment comics press alliance infinity swamp | Scalar |
| The avengers | 8 | The avengers Ultron | Association |
| Swades | 10 | Swades backdrop bbc | Metric |
| Godfather | 9 | Godfather advertisement crew | Metric |
| Black hawk | 4 | black hawk swan seahawk made noted actress | Metric, Scalar |
| Harry potter | 9 | harry potter crime lennix debbie alan chamber | Metric, Association |
| Harry potter book | 1 | harry potter book shocks jungle download culver itwiki chamber | Metric |
| Doom | 9 | Doom game | Association |
| Justice league | 10 | justice league dawn monsters brothers god | Metric |
| Khiladi | 0 | khiladi | Metric, association, scalar |
| Dr strange | 0 | dr strange roommate firewatch | Metric |

Based upon the above results, it was decided that the metric cluster method is the best for implementation of the pseudo relevance feedback.

# Collaboration with GUI (Asha)

Following was the design of the query expansion implementation:

1. Firstly, Asha needs to have stop words file in which each stop word should be in a separate line
2. Secondly, Asha needs to have a WordNet dictionary folder in her local machine which contains the definition of WordNet corpus and terms
3. Also, she needs to include two 3rd party libraries, apache-commons-io.jar, java-json.jar and edu.mit.jwi\_2.4.0.jar in the build path of the project for the query expansion to work.
4. Asha needs to call the getExpandedQueryString() method in the QueryExpansion class which takes in the query string she wishes to expand, provide a local path for a stop words file and provide a local path for the WordNet dictionary.
5. After the call to getExpandedQueryString() the following happens:
   1. The method creates the URL for a select query on Solr and fetches the output from the execution of that query in a JSON format
   2. The JSON string is read using 3rd party libraries into an array which is then passed to the Parser class
   3. The Parser class then tokenizes all the content and creates a dictionary with a token map for frequency in the document and position of occurrence in the document
   4. This token map created is the passed to the Stemmer class
   5. The stemmer class uses the WordNet dictionary to stem words and create a stem map which maps stems to all tokens contained in the token map.
   6. The token map created is then used to get a distance matrix for each pair of terms
   7. This distance matrix is then used to derive the stem correlation matrix with use of token and stem map
   8. The stem correlation matrix is normalized
   9. For each query word, the top 3 stems who have the highest values are chosen
   10. These stems are then parsed to remove duplicates and arranged in a string in the order of their score in the matrix
   11. The final query is then formed with the use of query given and these stems

**Ekal Golas – Query Expansion**

1. Asha needs to the take the final expanded query returned from the method and display it on another tab as the query expansion results. Also, she needs to search with the expanded query and get the more relevant results and display them as well.

# Project Demo selection

For the project demonstration, the query “kung fu” was selected because:

1. Kung as a query gets some irrelevant results which the query expansion improves upon and provides much better results
2. “Kung fu” as a query is still better than first but not all relevant results are fetched. The query expansion provides a more relevant expanded query for which the results are more relevant.

Hence overall, choosing this query demonstrates the working and advantage of the query expansion module very effectively.

**Asha Mary Thomas – User Interface**

**User interface and comparisons with Google and Bing:**

The front end is rendered using HTML/CSS/Java script and it is powered by Java Servlets in the backend. We also used AJAX to make asynchronous calls to the back end. Bootstrap CSS library is used to design the page layout.

The search functionality enables to query for movies. We have shown the result of the query in five different tabs on the same page: First tab contains the documents retrieved by vector similarity and then arranged based on the page rank. The second tab contains the results after applying the HITS algorithm. The next tab has the data after applying clustering to increase the relevance. The next two tabs contain the results of the query from Google and Bing respectively.

**Getting Google and Bing Results:**

We used REST calls to fetch the Google and Bing search results. Both Bing and Google supports custom search REST APIs. Authentication keys are required to access these REST APIs.

Account and Key configurations were done through the following links

Google : https://console.developers.google.com/apis/api/customsearch

Bing : https://datamarket.azure.com

**Accessing index and relevance model:** The index generated for our project is stored in Solr. The index is accessed from the front end by opening a HttpURLConnection provided by the java.net package. The relevance model based on TF-IDF is provided by the solr when we execute the query. The page rank score is also calculated and stored back to solr for efficient access during document fetching. Few parameters like which fields to query on, how many results needs to be fetched, the sorting order etc. are set just before firing a query using the URLConnection. The HITS score is calculated using a java program on the fly. i,e. it is calculated after the query is fired and the initial result set is retrieved.

Following steps are performed when user searches for a query:

1. Reads the query given by the user
2. Call google api and bing api to fetch the results from google and bing respectively
3. Open HttpURLConnection to SOLR using for fetching results
4. Set necessary solr parameters and append the query string received from the user in the opened connection and fetch the result
5. The retrieved result from solr is ordered based on a combined score that is calculated from query and document vector similarity as well as the page rank
   * 1. The page ranks for each url is pre-calculated and is available in the solr. The solr then calculates the score parameter using the query and document vector similarity model and the already available page rank.
6. Call the QueryExpansion Java program to compute the expanded query and its results
7. Invoke HITS algorithm and clustering algorithm by passing the data being retrieved in step 5.
8. Display all the result

**Asha Mary Thomas – User Interface**

*Note: The steps are not presented in the exact order as many steps would be invoked simultaneously. Eg: work being done simultaneous is:*

* *Step 6 and Step7 are called together just after step5 and all three would be computed*

*simultaneously.*

* *The display in step 8 is done in multiple parts as and when the corresponding data in a tab is available.*

We have done extensive tested using more than 70 movie names. The initial testing was done in collaboration with the team mate responsible for indexing to make sure that the pages fetched are relevant. We have done this testing for a constant set of 15 movies by using the different relevance models. First we tested using only the tf-idf value being calculated in the solr. Next we tested the impact and improvement when we used page rank along with tf-idf. Next we tested for the result test changes that happen when applying Hits score. By using a fixed set we could compare the results from the different relevance models to validate whether the results are getting better or not. Similar test has been done with the clustering and pages fetched after query expansion. For calculating the HITS score, the web graph is maintained as an adjacency list in a file which is read at runtime to perform the calculations.

The other part of the testing was done by selecting variety of movie names from different categories are year.

**Accessing Clustering Information**: We have maintained the clustering information in a file which contained url and the corresponding cluster number of that url. The top results from the solr are passed to the cluster based reordering algorithm. In the program, the most relevant cluster is identified and the documents which belong to the same cluster are brought up and the result is retrieved.

**Google/Bing Comparison**: The results showed by bing and google varies from the results we got, at least in the ordering of the urls. This can be attributed to our limited capability to crawl the famous movie pages like imdb. For example, in google the IMDB pages might be ranked higher whereas our result might not have that page or will not have high relevance score. The lower score is due to of the lesser number of pages from IMDB domain which in turn would have resulted in lesser number of links associated with it and hence getting a lesser page rank score. For some of the movies we could get very highly relevant pages which were on par with the bing/google results.

**Query selection for demo:** The queries selected for the demo are from the testing queries we have used. We have picked up the movie names that give good and relevant results on multiple modules of our project.

The three demo queries which we have used are:

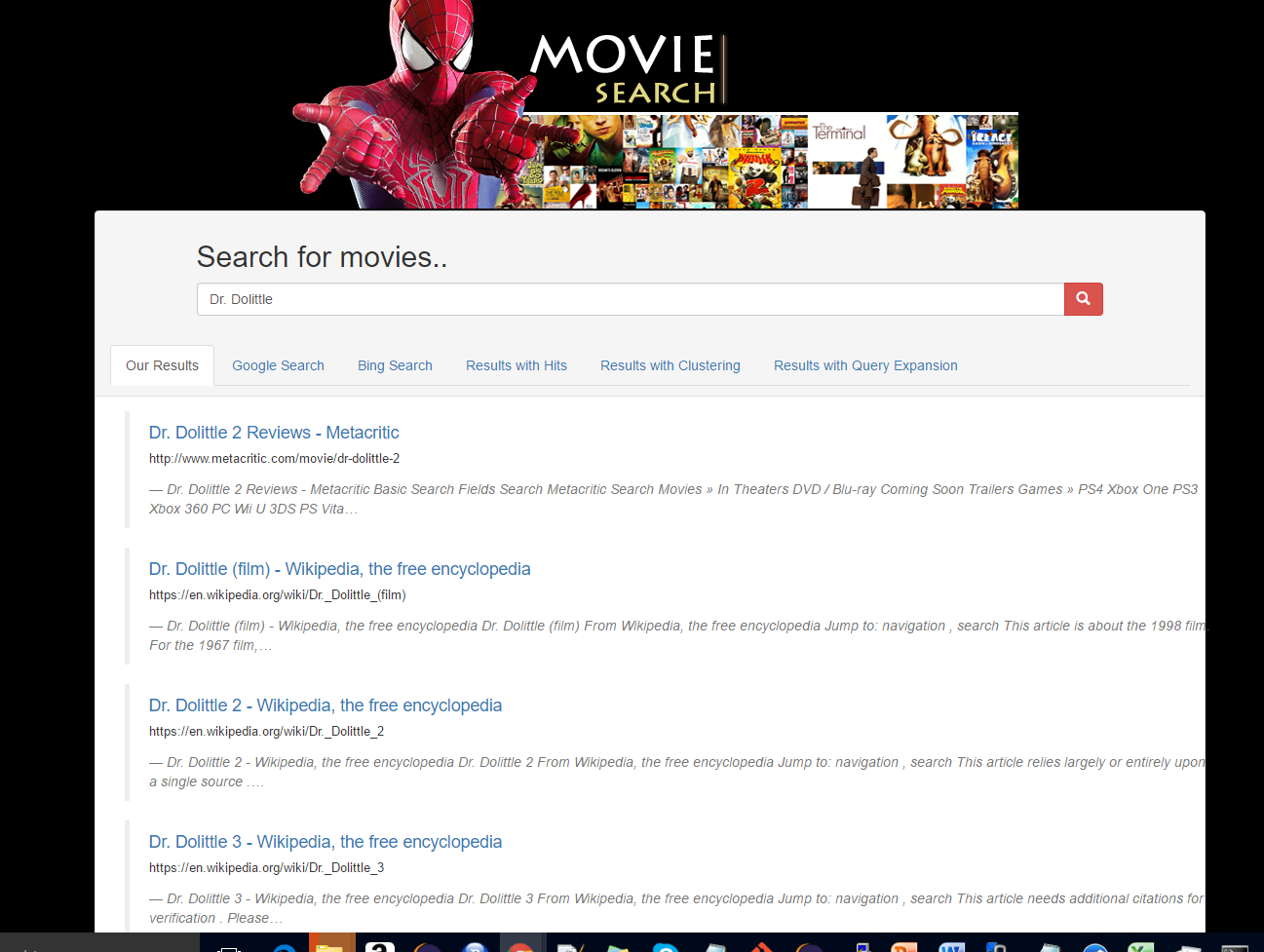
* The Godfather

**Asha Mary Thomas – User Interface**

* Rush Hour
* Dr. Dolittle

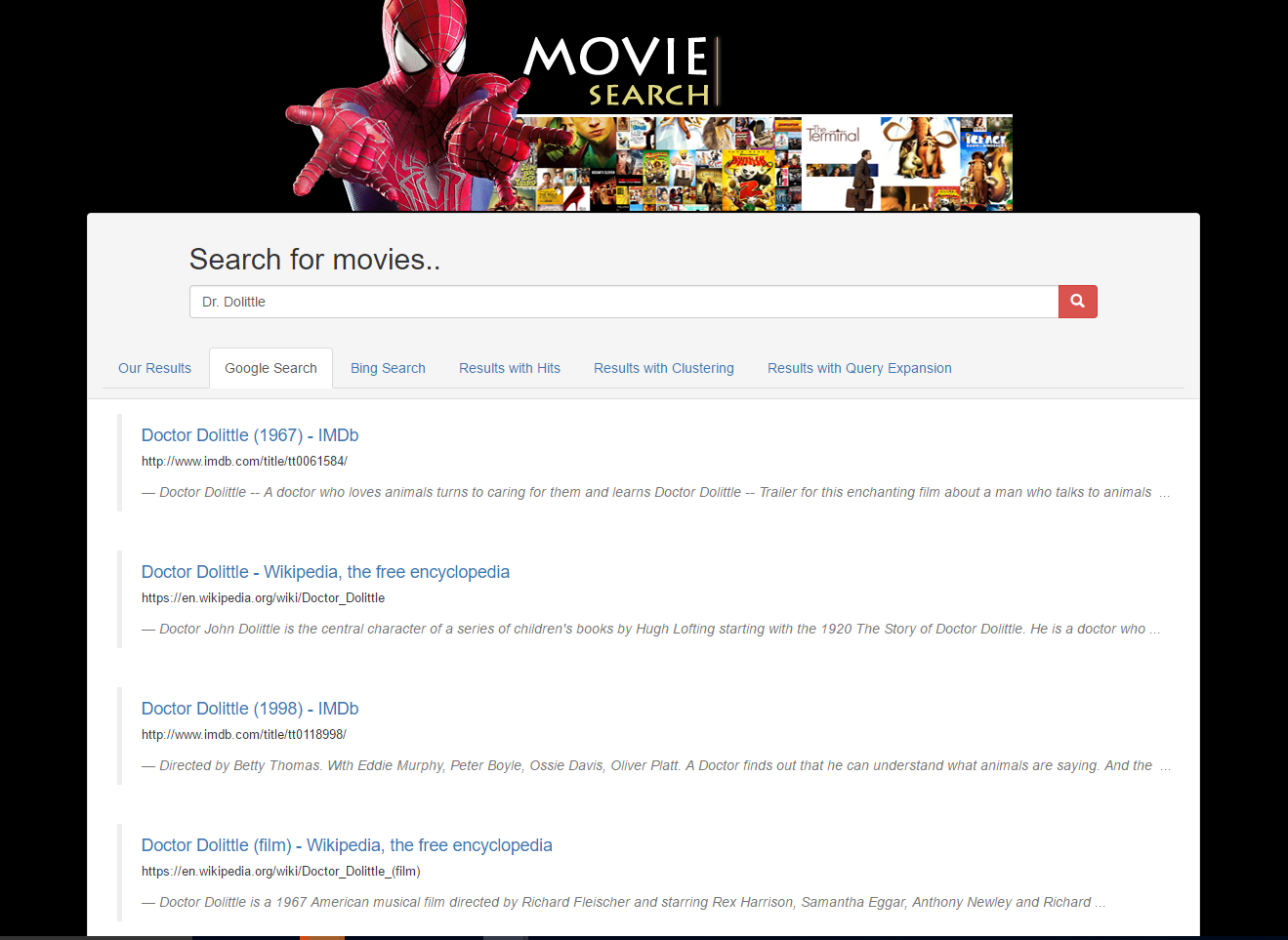
|  |  |  |  |
| --- | --- | --- | --- |
| Query | Our Results | Google Results | Bing Results |
| The Godfather | http://www.imdb.com/title/tt0068646/ | https://en.wikipedia.org/wiki/The\_Godfather | http://www.imdb.com/title/tt0068646/ |
| https://en.wikipedia.org/wiki/The\_Godfather\_(film\_series) | http://www.imdb.com/title/tt0068646/ | https://en.wikipedia.org/wiki/The\_Godfather |
| http://www.movies.com/godfather/m44776 | http://www.amazon.com/Godfather-Collection-Coppola-Restoration-Blu-ray/dp/B000NTPDSW | http://www.rottentomatoes.com/m/godfather/ |
| http://www.movies.com/godfather/godfather-news/m44776 | https://www.kabam.com/games/the-godfather | http://www.moviefone.com/movie/the-godfather/5180/main/ |
| http://www.movies.com/godfather/godfather-pictures/m44776 | http://www.amazon.com/Godfather-Coppola-Restoration-Marlon-Brando/dp/B0019L770A | http://www.amazon.com/The-Godfather-Marlon-Brando/dp/B005DNPFUE |
| Rush Hour | http://www.cbs.com/shows/rush-hour/ | http://www.imdb.com/title/tt0120812/ | http://www.imdb.com/title/tt0120812/ |
| http://www.imdb.com/title/tt0120812/ | http://www.cbs.com/shows/rush-hour/ | https://en.wikipedia.org/wiki/Rush\_Hour\_(1998\_film) |
| http://www.imdb.com/title/tt4085584/ | http://www.imdb.com/title/tt4085584/ | http://www.rottentomatoes.com/m/rush\_hour/ |
| https://en.wikipedia.org/wiki/Rush\_Hour\_(U.S.\_TV\_series) | https://en.wikipedia.org/wiki/Rush\_Hour\_(U.S.\_TV\_series) | http://www.rottentomatoes.com/m/rush\_hour/ |
| http://www.metacritic.com/movie/rush-hour-3 | http://www.rushhour.nl/ | https://en.wikipedia.org/wiki/Rush\_hour |
| Dr. Dolittle | http://www.metacritic.com/movie/dr-dolittle-2 | http://www.imdb.com/title/tt0061584/ | https://en.wikipedia.org/wiki/Dr.\_Dolittle\_(film) |
| https://en.wikipedia.org/wiki/Dr.\_Dolittle\_(film) | https://en.wikipedia.org/wiki/Doctor\_Dolittle | http://www.imdb.com/title/tt0118998/ |
| https://en.wikipedia.org/wiki/Dr.\_Dolittle\_2 | http://www.imdb.com/title/tt0118998/ | http://www.tcm.com/tcmdb/title/73264/Doctor-Dolittle/ |
| https://en.wikipedia.org/wiki/Dr.\_Dolittle\_3 | https://en.wikipedia.org/wiki/Doctor\_Dolittle\_(film) | http://www.rottentomatoes.com/m/Doctor\_Dolittle/ |
| http://www.metacritic.com/movie/dr-dolittle-2/critic-reviews | http://www.amazon.com/Doctor-Dolittle-Rex-Harrison/dp/B00004TS0D | https://en.wikipedia.org/wiki/Doctor\_Dolittle |

**One screen shot:**



**Asha Mary Thomas – User Interface**

Google:



**Asha Mary Thomas – User Interface**

Bing:

