Hurricane Katrina Narratives - SearchKat

Instructor: Edward A. Fox

Client: Katie Carmichael

By: Matthew Chittum, Tanvir Rahman, Zinai He, Jiarui Li

Virginia Tech

CS 4624 Multimedia, Hypertext, and Information Access

City: Blacksburg, VA

Date: May 6, 2015

Table of Contents

1 Executive Summary/Abstract

SearchKat is a searchable database built on Apache Solr. The database was created with the purpose of being able to search Hurricane Katrina narratives that were gathered from survivors. The project was started in Fall 2015 in CS4624 Multimedia, Hypertext, and Information Access with Dr. Edward Fox as the professor and Dr. Katie Carmichael as the project client.

SearchKat has user specified word groupings which were determined by Dr. Carmichael in order to make the database thematically searchable, with each of these word groupings being considered a theme. The system can both search by these word groupings as well as by generic word associations such as synonyms.

The database has customized fields displayed when a query is performed. We have created a CSV file which stores the filename, line number, and line content for each line through all of the files. We use this information to display this relevant data whenever a search is performed.

2 User's Manual

2.1 Introduction

SearchKat is a database built on Apache Solr. It indexes user specified text files and allows the user to run queries on the indexed files. Specifically it is designed to index and search through transcripts of interviews from Hurricane Katrina victims. SearchKat is designed in this way to meet the specifications of the project client, Dr. Katie Carmichael.

Included in SearchKat there are three main executables:

- Parser/Wordcount (*transcriptCleaner.jar*)
    - This functions as the first step for preparing documents for SearchKat. The parser will detect any text exactly matching "sp" text bounded with braces ("{ }") and remove them. The cleaned file will be outputted to a folder labeled "output" located in the root folder. It will then do word count for each cleaned file and form a CSV file with two columns: narrator (equivalent to filename) and word with number of occurrence in that file. It will ignore words with length less than 5 while performing the word count since these are mostly common words and can be discarded. The before and after effect is shown in Appendix Figure 1 and Figure 2.

- Database CSV File Generating Script (*csv_generator.py*)
    - This Python script creates a single CSV file from a directory of output files generated by the Parser above.

- Solr Database (*start.sh* and *stop.sh*)
    - This is a folder containing all the files for our implementation of SearchKat. We generated one CSV file containing all contents in all txt files provided by our client in order to customize the search results.

2.2 Instructions for Parser/WordCount

The program is a executable jar file. To use, place the jar file in the same directory as the txt files you want to clean and perform a word count on. Then execute *cleaner.sh* to run the program. The cleaned files will be placed into a folder labeled "output" and the word count will be placed into a folder labeled "wordCount".

2.3 Instructions for Database CSV Creation Script

The script which generates the CSV file is a single Python script. In order to execute the script the system must have Python installed. In order to create a database CSV file, line 3 should have the path to the output folder from the parser specified. Once this is specified, the script can be executed from the terminal by typing “Python csv_creator.py”. This will generate a file in the same directory that the script is executed that is labeled data.csv and contains all the information from the output directory in single file.

2.4 Instructions for Database

For simplicity we have included two scripts for SearchKat: A script to start the database (*start.sh*), and a script to close the database (*stop.sh*). Both of these scripts are located inside the directory “*Solr-5.0.0”* and can be executed by double-clicking them in the file system. The copies of the .sh files are included with the submission.

After the start script has finished executing, open up a browser and navigate to: http://localhost:8983/solr/csvCore/browse to access the search UI.

In **case an error occurs**, the database can be started by following these commands (type these in the terminal):

1. Start Solr on certain port of localhost
    a. bin/solr start
    b. Run "http://localhost:8983/solr" on browser of choice.
2. Create a core name csvCore
    a. bin/solr create -c csvCore
3. Post everything in doc/, indexing all the file within that directory (in other word, Add documents for searching)
    a. bin/post -c csvCore "filedirectory"
4. To browse search interface (search box for you to work with)
    a. Go to http://localhost:8983/solr/csvCore/browse

To stop Solr if *stop.sh* fails, you can follow these commands (type these in the terminal):

1. Stop everything
    a. bin/solr stop -all
2. Delete index file from core csvCore
    a. curl http://localhost:8983/solr/csvCore/update --data '<delete><query>*:*</query></delete>' -H 'Content-type:text/xml; charset=utf-8'
    b. curl http://localhost:8983/solr/csvCore/update --data '<commit/>' -H 'Content-type:text/xml; charset=utf-8'

3 Developer’s Manual

3.1 Introduction

SearchKat is a project for creating a thematically searchable database of Hurricane Katrina narratives. This project was started in the spring semester of CS4624: Multimedia, Hypertext, and Information Access with Dr. Edward A. Fox as the Professor and Dr. Katie Carmichael as the project sponsor. This manual specifically relates to the development of SearchKat and is intended to be a resource for any developers who may continue the project.

SearchKat is built using Apache Solr. “Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites.” Solr indexes documents via JSON, XML, CSV or binary over HTTP. Querying is done via HTTP GET and results are in JSON, XML, CSV or binary. [3]

SearchKat mostly consists of the database; however, there are two intermediate programs/scripts we created in order to assist with SearchKat. The three parts of the project are:

- Parser/Wordcount
    - This functions as the first step for preparing documents for SearchKat. This file The parser will detect any “sp” and text bounded with braces (“{ }”) and remove them. The cleaned file will be outputted to a folder called “output” located in the root folder. It will then do word count for each cleaned file and form a CSV file with two columns: narrator (equivalent to filename) and word with number of occurrence in that file. It will ignore words with length less than 5 while doing word count since these words won’t be important for client to find out the word grouping.

- Database CSV File Generating Script
    - This Python script creates a single CSV file from a directory of output files generated by the Parser above.

- Solr Database
    - A folder containing all the files for the implementation of SearchKat. It has two modified configuration files(schema.xml, solrconfig.xml) for customizing browsing result, shell scripts files for starting and stopping SearchKat, and a synonyms.txt storing synonym groupings for thematic searching.

Each of these parts of the system have full documentation below with full details for any future work. As a note, the interview files were not submitted to VTechWorks

alongside the project due to the confidential nature of the files. If the project is continued, Dr. Carmichael will need to provide the files.

3.2 Inventory of All Files

Here we list all of the files developed for our project. For simplicity, we don't list a full directory of the Solr files, only files which we feel are important or that we modified. For full details of Solr, see the Solr documentation at http://wiki.apache.org/solr/. The input files used in the database were not included in the submission due to confidentiality. Dr. Carmichael has these files and the files will have to be gathered from her if they are needed.

- Parser/Word Count
  - *transcriptCleaner.jar*
    - Executable jar file that takes one parameter which is the path to the folder with txt files. The program can be executed by running the cleaner.sh or type command line "java -jar transcriptCleaner.jar [directory]" in terminal. This program is written in Java and requires Java to be installed.
  - *cleaner.sh*
    - Bash script file that can be put into the folder with txt files and run to invoke the jar file. This script requires Unix based system to run.
  - *README-transcriptCleaner.txt*
    - Plain text file that describes the usage of *transcriptCleaner.jar*.
  - *commons-io-2.4.jar*
    - The external library to support the source file.
  - *main.java*
    - The source file.

- CSV Generator
  - *csv_generator.py*
    - Single script file that creates a database CSV file from the output generated by the parser. The generated CSV file is indexed by solr and used for searching. The script can be executed by specifying a path to the output file directory in Line 3 and then either executing by double clicking it or executing it in the terminal. This script is written in Python and requires Python to be installed.

- Solr Files
  - *schema.xml*: modified for enabling thematic search
  - *solrconfig.xml*: modified for customizing browsing result

- *synonyms.txt*: storing synonym groupings for thematic searching.
- *start.sh*: run SearchKat at localhost port 8983
- *stop.sh*: stop any running SearchKat nodes

3.3 Details of Parser/ Wordcount

The Parser/ WordCount program is a 184 line Java program. The program takes a single parameter which is the path to the folder containing all txt files. It will generate cleaned (no markings) txt files and output them into a folder in the given path named "output".

Then, it will automatically take the cleaned txt files and perform a word count on them. The word count will be output to a folder in the given path named "wordCount". The word count for files is generated into a single CSV file with two columns. The first column is the narrator's name (extracted from filename), and the second column is the word with the number of occurrence in the file. The words are ordered in descending order.

Any words with a length that is less than 5 are not included since they won't be helpful to client's decision of word grouping.

The purpose of this program was to clean the txt files obtained from client and make them easier to read. The word count file was delivered to the client in order for her to determine the word groupings for thematic searching.

3.4 Details of Database CSV File Generating Script

The CSV generating script is a 31 line Python script. The script takes no input and generates an output file in the directory it is executed in. Instead of taking input, it must have the path to the folder containing output from the Parser specified in line 3.

The file it generates is a CSV file that has four columns: filename, line-number, line-content, and full lines. The filename and line-number are self-explanatory, the line-content is the content for the exact line and full lines is the line content with the previous and next line added (if available).

The purpose of this script was to preprocess the data for our Solr database in order to associate more info with the raw data. By indexing the data as a CSV file instead of a directory of files, we can associate the columns in the CSV file as fields and specify which fields we want to have displayed when we perform a search. This allows us to more closely tailor our results to what our client wanted displayed.

3.5 Details of the Solr Database

SearchKat is an implementation Solr 5.0.0. SearchKat uses cores to manage the file index and to manage searching. "csvCore" is the core we created to manage searching our indexed CSV file.

SearchKat's root folder is "solr-5.0.0*"*, all command should be executed under that directory.
Two configuration file were modified for our project:

- *schema.xml* is located at "*/solr-5.0.0/server/solr/configsets/sample_techproducts_configs/conf/*". Line 455, "*<filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>*" is uncommented to enable synonym searching.
- solrconfig.xml is at "*/solr-5.0.0/server/solr/csvCore/conf*". At line 855, the requestHandler "/browse" is modified to enable a filtering param "fl", which specifies what fields to display on the search result (SearchKat retrieve fields from the CSV file)

Since Solr is Java based it requires the **Java JDK** to be installed. Also proper file permissions must be set for the *start.sh* and *stop.sh* scripts to be executed.

3.6 Conclusion

In the developer's manual we have listed the various files that were used in the steps of constructing the database SearchKat. This manual is for anyone that wants to examine the source files for the database and also meant for those that wishes to continue and expand on SearchKat in the future. In the appendix we have included a few screenshots with descriptions that depict the different phases of the project. All of the mentioned files and more can be found in VTechWorks.

4 Lessons Learned

This section of the report details the steps the team went through in order to complete the database. Here you will find the the team's objectives for the project, the planned steps to complete the database, the timeline showing the planned task completion and actual task completion dates, and the problems and issues the team encountered during development along with the steps taken to resolve them.

4.1 Project Plan and Timeline of Project

Phase 1: Clean the transcripts by removing unnecessary tokens.

The first phase is the initial step needed to begin the project and heavily ties into phase two. We need to first remove these tokens from the documents in order to both make them more readable and to make pulling the word counts easier. The transcripts include various meaningless tokens that were picked up by the voice recording instrument that the client used when transcribing from audio to text. These meaningless tokens are to be removed. This is to be done without losing any meaning of the original speaker and without removing culture-/region- specific language and phonetics. To remove the meaningless tokens we will write a small Java program which will parse through each line in a file and remove lines which contain an unnecessary token. See Appendix Figure 1 and Figure 2 for a sample.

Phase 2: Obtain a frequency count of the words in the documents.

Phase 2 is another step necessary to begin the implementation of the database search engine, as this contributes to our searchable database(SearchKat's) thematic search feature. After cleaning the documents, the next step is to obtain a word frequency for the documents. This may be done as either an aggregate count across all documents or as an individual count for each document depending on Dr. Carmichael's preference. The word frequencies will be calculated by a small Java program which will parse each document and count the words. Once SearchKat is fully implemented, it will be able to provide the word frequency of a given document(s) as an added function.

Once the word frequencies have been gathered, the results will be delivered to the client, Dr. Carmichael. She will look through these to determine what she thinks are meaningful themes/groupings of words and will deliver these back to the group. These themes will be used for the thematic searchable database in phase 3.

Phase 3: Searchable database of the documents

Phase 3 is the actual implementation phase of our project and where the majority of the work will be done. We will learn to utilize the Solr search engine's built in features to execute thematic search with a focus on themes indicated by Dr. Carmichael. The search should give a list of all of the documents related to the search key. The search will not only give the full document as the search result but the portion of the document(s) (snippet) that are related to the search key. Related words, phrases, names, etc. (i.e,. themes), decided by Professor Karmichael, will give similar search results. For instance, searching a word that is a part of a word grouping will return the results of searching for all words of the word grouping.

We needed to provide a graphical interface for users to interact by executing searches on the database. As for our interface's "search engine", we will use Solr, an open source enterprise search platform built on Apache Lucene. It will be running behind our interface, and execute search commands as users interact with the application.

4.2 Steps to Completion of SearchKat

Step 1: Prototyping

Step 2: Basic Implementation

a. Single word search
b. Phrase search

Step 3: Testing/Evaluating

a. Check for keywords that are known to return specific valid results
b. Check if all word groupings are properly searchable
c. Check for keywords that are known to return no results
d. Check that time taken for results is within expected parameters
e. Check that results are ordered by the greatest search hits (i.e., the first search result has the greatest number keyword/phrase matches in the document) (or some other criteria)

Step 4: Final Implementation Including Thematic Searching

a. When the user enters a search query that is part of a similarly themed word group, it will return results similar to the results of any of the queries in the same themed group. See example in Appendix Figure 6

Step 5: Final Evaluation of System

a. Verified that features such as single word/phrase search, thematic searching function properly.
b. Ensure our client is comfortable with using the system.

Step 6: Deliver the following items to Dr. Carmichael

a. A search engine that enables single word/phrase and thematic searching within all Katrina reports

b. A user manual including an installation manual and usage manual
c. Possible contact of technical support for Katies during for the near future

Table 1: Timeline of Task Details as of May 6th, 2015

Task	Percentage Completed	Status	Day Started	Day to Be Completed	Actual Completion Date	Task Assignment	Milestone
Parsing	100%	Completed	2/1	3/31	3/6	Client	Yes
Word Count	100%	Completed	2/1	3/31	3/9	Client	Yes
Basic Searching	100%	Completed	3/13	4/15	4/12	Client	No
Thematic Searching	100%	Completed	4/12	4/30	4/28	Client	Yes
Final Deliverable	100%	Completed	2/9	5/6	5/6	Client	Yes

4.3 Problems Encountered and Solutions

While working through the planned steps to complete the project we came across a few problems. Some of these problems were easy to address while others took more time and effort to resolve. These issues often required us to rethink the problem and view it from a different angle. For some of the problems we needed to seek assistance from experts outside of our group. This included our professor for the class, Dr. Fox, and Mohamed Magdy Gharib Farag, a graduate student working with Solr and referred to by Dr. Fox.

- Problem: Our parsed interviews did not have any data associated with it in order to search on. This meant we did not have any context when performing a search.
    - Solution: We fixed this issue by preprocessing the text files before indexing them into Solr. This was done by the csv_generator script discussed above. The purpose of this was to concatenate all of the interviews into a single file, and to associate a line number and line content with each of the returned results.

- Problem: When searches were performed using Solr default settings, the results contained information that was not useful to typical users. The default results contained the narrator (the name of the actual file), the Id that Solr has indexed the file as, the _version that Solr has assigned to the file, the search text, the number of times it's in the file, and a score that Solr has calculated. See Figure 4 in the Appendix for an example. This gives the user very little insight on their search text. It tells the user in what file to look in but nothing else.
    - Solution: This was fixed by editing the fields that Solr displayed when displaying the results. The fields were changed to display the name of the file where a match was found, the line number the match was found at, and the line of text associated with it. See Figure 5 in the Appendix for an example.

- Problem: With our preprocessed CSV file, we were only reporting the exact line the match was found in. This was an issue due to some lines consisting of only a single word and therefore did not have very much meaning associated with it.
    - Solution: We fixed this issue by making revisions to the CSV generator script. Initially the script only had three columns for the filename, line number, and line content. The revised script added a fourth column which contained the previous and next lines alongside the current line (if available). To use this new file, we searched on the third column which only contained the exact line and displayed the fourth column which contained the line matched with the previous and next lines concatenated with it. See Figure 6 in the Appendix for an example.

4.4 Future Work

Currently there are no future plans for the project. Currently the team feels that all objectives currently for the project have been completed. There are no new features currently planned to be added to the project.

5 Acknowledgements

We appreciate all of the help Dr. Fox and Mohamed Magdy Gharib Farag provided us with. They assisted us with figuring out how to use some of the features of Apache Solr and directed us to various resources documenting Solr features. We thank Dr. Carmichael for trusting us with the Hurricane Katrina interview transcripts and for meeting up with us several times to discuss the specifications for the database.

** Note: The following information is valid as of May 6th, 2015

Edward A. Fox

- Professor in the Department of Computer Science at Virginia Tech
- Office in 2160G Torgersen Hall
- Director of Digital Library Research Laboratory (DLRL)
    - Torgersen Hall 2030, x3615
- Office in 2160G Torgersen Hall
- Address: Dept. of CS, 114 McBryde Hall, Mail Code 0106, Virginia Tech, Blacksburg, VA 24061
- phone: 540-231-5113
- email: fox@vt.edu
- WWW: http://fox.cs.vt.edu/

Katie Carmichael

- Assistant Professor in the English Department at Virginia Tech
    - College of Liberal Arts and Human Sciences
- Office in 407 Shanks
- Address: 181 Turner St NW, Blacksburg, VA 24061, USA
- Phone: 540-231-7712
- email: katcarm@vt.edu

Mohamed Magdy Gharib Farag

- Graduate student at Virginia Tech with Dr. Fox and Riham Mansour as advisors.

Appendix

Figure 1: Original Text before Parsing:

Sample of the original text. Specifically the markings that are being removed are “sp”, “{BR}”, and “{SL}” in this sample. There are other markings in the full text.

```
{SL}
{BR}
So went to Lafayette
sp
very good friends
sp
we had
sp
maybe five households of people
sp
friends and
sp
and their relatives that
sp
came to stay with them in Lafayette
sp
um we ended up being there couple of weeks
{BR}
watching the coverage
sp
on television was excruciating
sp
they were not showing St. Bernard
sp
nor St. Tammany
sp
they pretty much were showing the Super Dome
sp
and every now and then youÕd get a little something
sp
and there were
sp
you couldnÕt use the phone and you couldnÕt have found people to use the phone if you couldÕve
sp
um I mean you could use the phone in Lafayette but not to
sp
```

Figure 2: Processed Text After Parsing:

```
So went to Lafayette
very good friends
we had
maybe five households of people
friends and
and their relatives that
came to stay with them in Lafayette
um we ended up being there couple of weeks
watching the coverage
on television was excruciating
they were not showing St. Bernard
nor St. Tammany
they pretty much were showing the Super Dome
and every now and then you'd get a little something
and there were
you couldn't use the phone and you couldn't have found people to use the phone if you could've
um I mean you could use the phone in Lafayette but not to
```

Figure 3. Administration page of the Solr search engine.

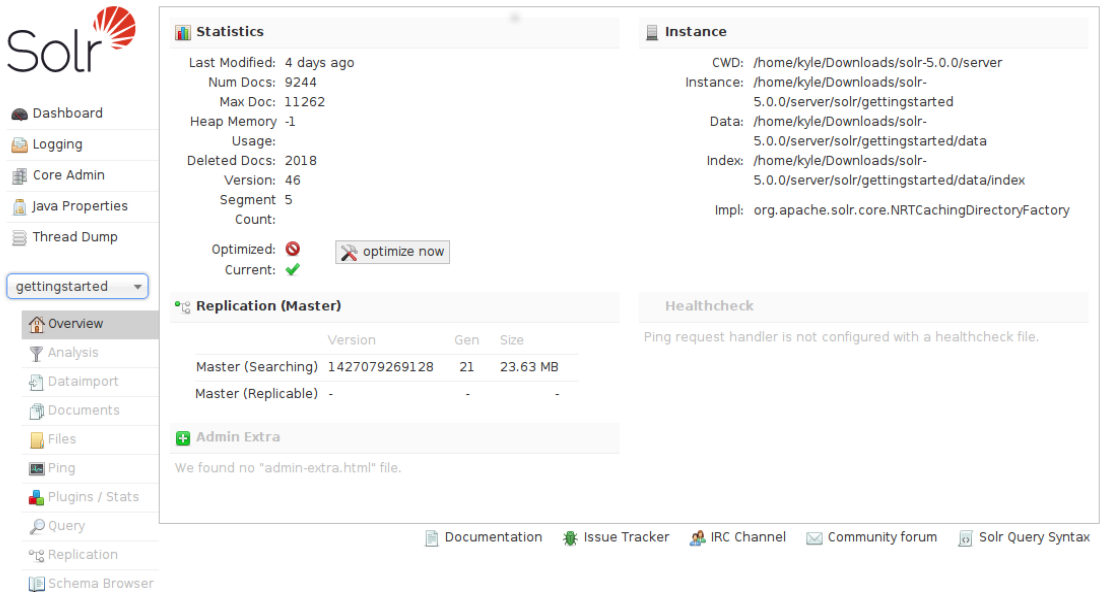


Figure 4. Default results when searching with Solr.

The default Solr fields are Narrator, Id, _version, word, and score.

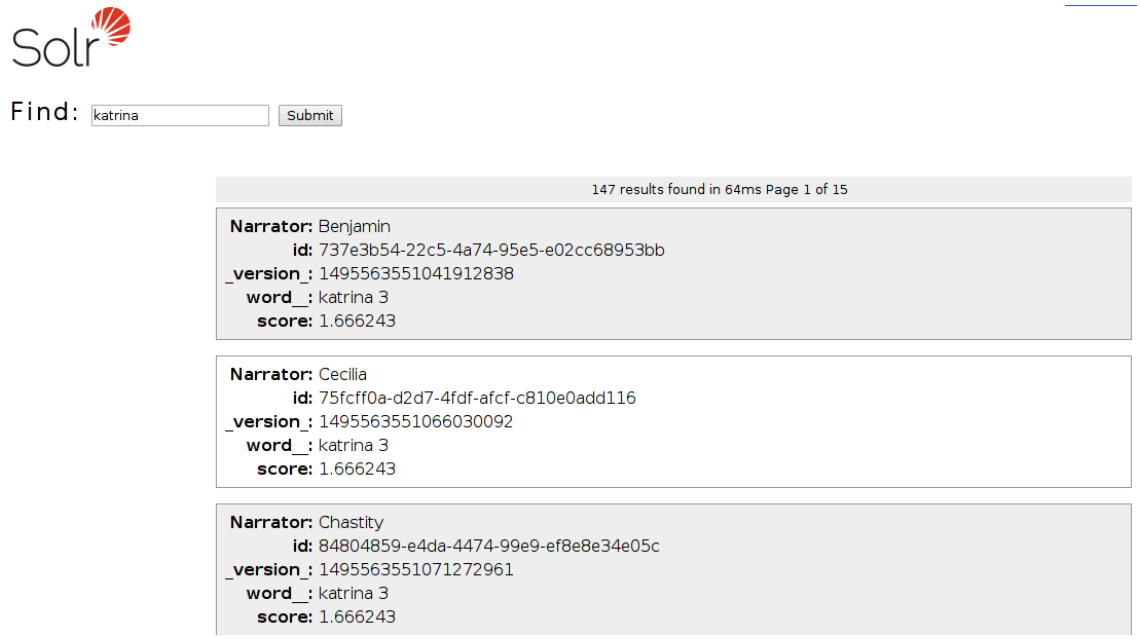


Figure 5. Customized fields for search results

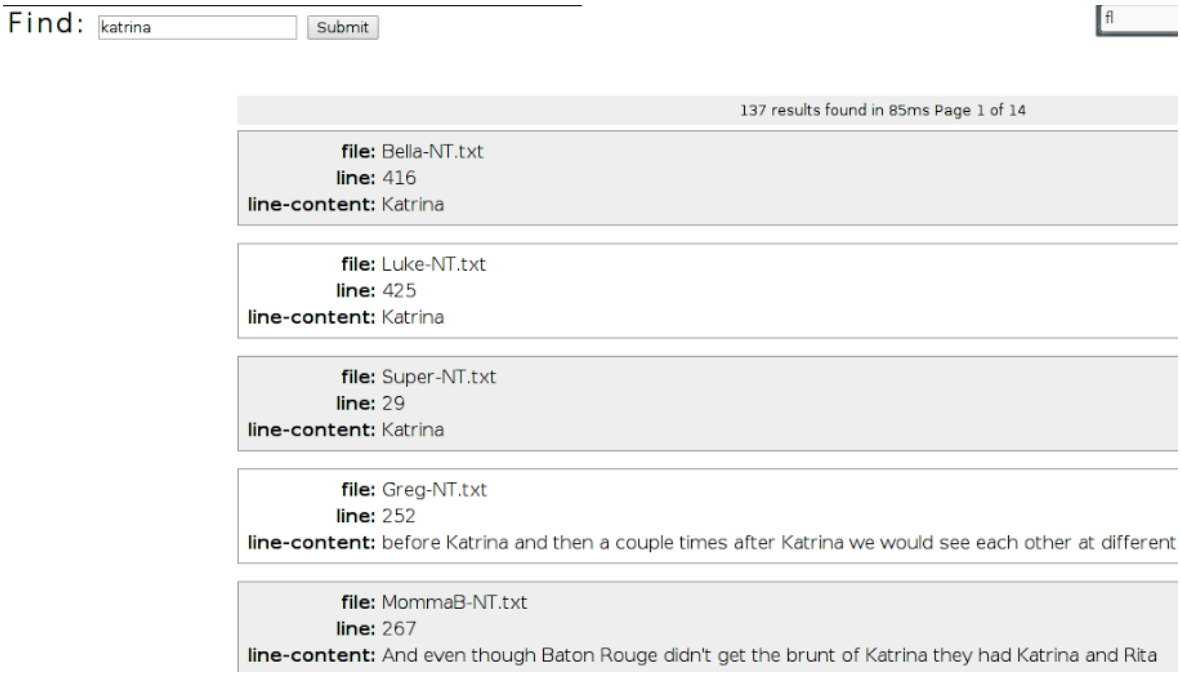
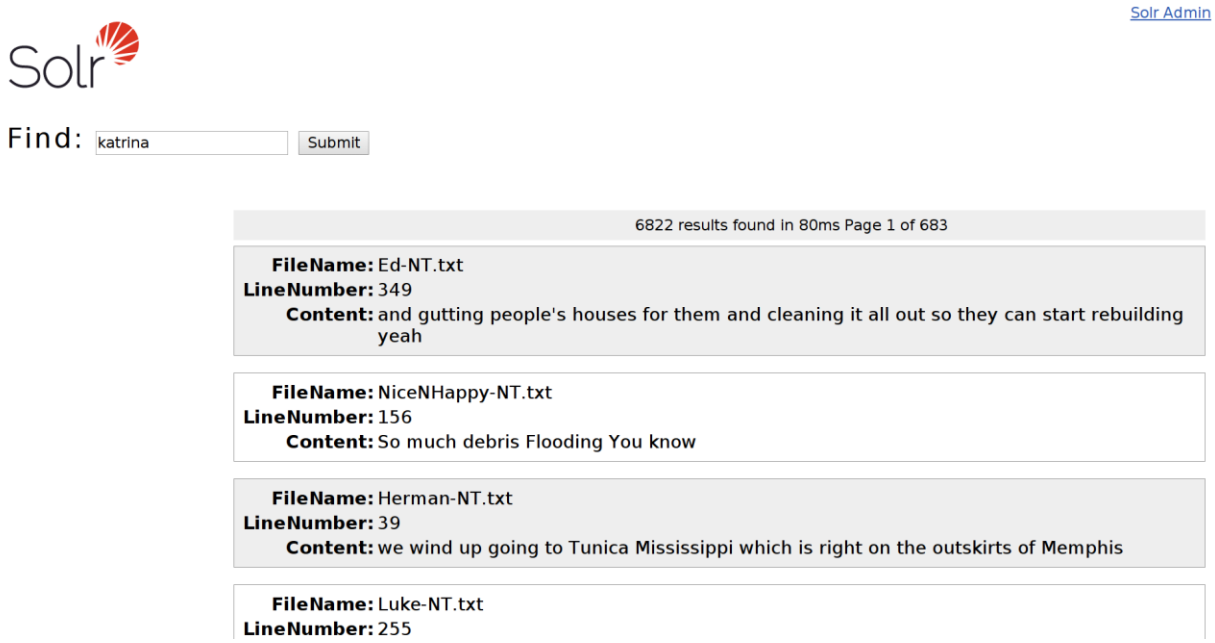


Figure 6. Completed Thematic Searching

The word groupings specified by the client can be found in file *Themes.docx*



References

1. Mishra, Sanjeev. "HowTo? Solr: Building a Custom Search Engine." *Tech 360: HowTo? Solr: Building a Custom Search Engine: Part 1*. N.p., 26 Dec. 2013. Web. 27 Feb. 2015. <http://pixedin.blogspot.com/2012/05/howto-solr-building-custom-search.html>.

2. Vittal, Chiradeep, and Kishan Kavala. "Design Document Template." *Design - Apache Cloudstack - Apache Software Foundation*., 08 Sep. 2014. Web. 28 Feb. 2015. <https://cwiki.apache.org/confluence/display/CLOUDSTACK/Design+Document+Template>.

3. "Solr Is the Popular, Blazing-fast, Open Source Enterprise Search Platform Built on Apache Lucene™." Apache Solr. 2015. Web. 15 Mar. 2015. <http://lucene.apache.org/solr/>