Sports Central *

Brian Beer
Department of Computer Science
University of Illinois at Urbana-Champaign
bbeer2@illinois.edu

Christopher Choi
Division of General Studies
University of Illinois at Urbana-Champaign
cdchoi3@illinois.edu

ABSTRACT

Many people continuously try to access the latest and greatest information on sports using the Internet. Our CS 410 group project centered around building a website/search tool providing a foundation of sports information that contains articles from several different sports websites. As of writing this report, this includes a few thousand (recent) news articles including ESPN, Yahoo! Sports, Bleacher Report, and NBC Sports. Also, it includes several hundreds sports-related tweets from Twitter, a social networking service. Users can utilize this tool in order to search and examine sports articles related to queries and tweets on specific topics. Through this searching tool, one can view sports trends and other peoples' interests and responses to sports-related events. This paper serves to introduce the background of why we created this tool, methods used to create it and ideas for future improvements.

Keywords

sports, text information, social network, news

1. INTRODUCTION

The development of improved broadcasting systems and ultimately the Internet, allow people to experience and follow sports, as they happen, on the other side of the globe. This, coupled with the increasing popularity of social networking sites such as Facebook and Twitter have made it easier for people to express and share their responses to sports-related events immediately. The fast-paced communication surrounding sports calls for speedy access to information. In order to fulfill sports fans' desire for faster access to information, for our CS 410 project, our group decided to make a website or tool that provides quick views of people's responses and news articles surrounding sports. The tool is composed of five fundamental parts: crawler(s), SNS integration, the inverted index generator, back-end, and front-end.

*Submitted as the final project for CS 410 Summer 2014.

The crawler's main task is to pull articles from sports websites, comprised of the Wget Linux terminal command, and scraping part, which is implemented using Python programming language. It would crawl the articles from sports websites and return resulting txt files, which are accepted by the Lucene inverted index generator. The social network integration part crawls tweets from Twitter using Twitter Search API and Java programming language. Twitter Search API allows one to pull certain tweets related to predefined queries, and, using Twitter4J libraries, Java application output "tweets.txt," several tweets delimited by newline character. This resulting file would be used by the inverted index generator. The building index part consists of Apache Lucene, which provides fast search libraries and a server to generate inverted index using tweets and articles. The back-end query service is composed of Apache Solr, a Java-based full-text search server using Lucene to do the querying. The front-end is designed using HTML and CSS and implemented using JavaScript and AJAX. This allows the users to input searching queries and displays the result by processing the resulting JSON file from Apache Solr. Finally, the content-update automation, implemented using Java, connects each part in order to minimize manual work towards updating the database. As a result, the website or searching tool has more than a few thousand (recent) news articles from sports websites including ESPN, Yahoo! Sports, Bleacher Report, NBC Sports, and CBS Sports, while also providing several hundred sports-related tweets from Twitter social networking service.

2. RELATED WORK

Various pieces of the tool exist, split into different websites. ESPN, for example, has sports writers that write "topical" pieces, providing some insight into their opinions and predictions on sports. Google news provides means of a vertical search engine to pool together recent news (including sports articles). Social media accounts of major news outlets, provide play-by-play updates immediately, significantly before any article could be written. However, we believe that while there are different websites that are useful in some of these aspects, there is no central site that captures the trending information from the Social Sites coupled with the a good aggregated set of sports news. The idea is to answers people's calls for getting an overall sense of: "What is happening in Sports?" While Google news seems to provide the best interface for aggregated sports news, they don't provide any notion of how the general public feels or what they are saying about the same topics that you are interested in. Wanting

to know "What are people talking about Sports?" at a quick glance and overall how people feel about a win/loss, showing both sides on the same, easy to use page. Our goal is to fill in this gap.

3. PROBLEM DEFINITION

As stated above many sports websites exist providing up-to-the minute updates of your favorite sporting events, from basketball to football to soccer and so on. Today, articles are distributed between several different websites, including ESPN, Bleacher Report, Yahoo! Sports and many more. Each site provides it's own spin on predictions of upcoming games, analysis of teams and players and are subject to the biases of the journalists and newscasters themselves. The challenge to overcome, is to get an understanding of the difference in opinions, sporadic information spread between multiple sites and have the most up-to-date sports information at the hands of the users without having to spend hours on end digging through each web page. This problem can be tackled by crawling these web pages, collecting articles, comments, tweets, etc. and aggregating the data into a common index.

Navigating a large collection of data is a daunting task without a user-friendly interface. Such an interface would require precise results and an easy to use querying method. Both search results for articles and for social media comments/tweets should be returned in a well-organized, yet simultaneous fashion and be on the same topic. In terms of social media, Twitter is a tool, popularly used to share many forms of opinions, discussions, feelings, predictions, etc. in all aspects of life. It has been consistently growing in popularity over the years. Crawling and performing sentiment analysis, presents it's own unique set of challenges, specific to Twitter. These include handling slangs, abbreviations, smiley faces (emoticons), punctuation and other unique traits that these messages are composed of. In order to avoid confusing a user, organization of the output information needs to be kept in mind. Preserving most - recent information poses a significant challenge when aggregating across many pages. Optimized automation would be required to continuously be updating the articles, social media, etc. and make for difficult challenge to keep pace with a user's thirst for the latest and greatest information.

4. METHODS

4.1 Crawling

I (Christopher Choi) implemented the fully working web crawler. The crawler has two main parts, the crawling part and scrapping part. For the crawling portion, it iterates through all links in the targeted websites and gathers the suitable links that contain sports news articles. The scrapping part scrapes the articles from those locally saved articles and compiles them into a txt.

A challenge I ran into was implementing the crawling part because I have not used JavaScript (JS) extensively recently or other programming languages usually used for crawling well enough. I originally thought to try the JS console method shown during the lecture. This method uses a JS console built in Google chrome to gather the website URL according to the website's HTML structure, to grab the URL inside certain HTML tags. However, I could not save the crawled URLs into txt file from Google Chrome, a necessary requirement for this project. I then tried to use JS's ActiveXObject object to save the string into a txt file from Google Chrome, but it did not work. In addition, I could not determine a way to call another function which would accept strings of URLs as an input to Google Chrome, in order to provide the automation necessary for this step.

Therefore, I moved on to implementing the code in a JS file instead. I could then crawl some URLs from ESPN websites and print them out in the terminal windows, but still could not save it into txt file. The problem turned out to be that the ActiveXObject object only supports Windows and Internet Explorer, but my development system were limited to Linux on EWS machine or Mac OS X on my MacBook.

Another try was to combine the crawler and scraper from methods used in MP3, in class. For crawling part, I tried to use the PhantomJS scraper. I copied and pasted the JS codes from the JS file, created before (mentioned above), into PhantomJS scraper and connected them. However, there was a problem with the ESPN website, the websites themselves seemed to contain errors, which prevented PhantomJS from being able to properly scrape them.

Consequently, given no other options I began looking for an alternative. Scrapy for Python seemed like it offered good promise, but I could not implement it correctly because of syntax problems and complexity of the tool itself. Crawling extensions for Chrome, Beautiful Soup for Python, and crawler4j didn't yield any better results. Finally though, I had a breakthrough after I decided to use GNU Wget, a common terminal command that would crawl the website with several options. Ultimately the command I used was

wget -c -r [-l1 optional] –reject jpg,pdf,gif [starting website URL] [2].

A quick explanation of what this means, -c flag would allow it to continue from the previous Wget command, if it was interrupted or not completed the last time. -r flag made Wget crawl the whole website recursively by going through all the links in the website. -l1 flag limited its recursion one level deep, which means that it would only crawl the links in the starting page, and it would only be used to update the articles after the full websites had been crawled already. Lastly, –reject flag would cause Wget to not download the files that have certain types (jpg, pdf, and gif in this case). Overall, the whole Wget command would recursively crawl the entire websites and their structures. Problems were with Wget can be encountered when it downloads everything from the website including all the repositories. Therefore, some websites, like ESPN and Yahoo Sports, have their HTML article files inside the sub-repositories, so I had to move all the HTML files into one repository in order to process them. First, I tried to move files manually, but quickly realized that was not practical and proposed the following solution [2].

find [source directory] -type f -print0 | xargs -0 mv -t [target directory] [8].

In this command, find [source directory] would find several items, such as directory, file, and sockets, in the source directory, and -type f flag would limit targets to be found as regular file. And, -print) would print the file names

on the standard output, followed by null character. xargs would read items from standard input, and -0 flag would specify the file names in the standard input by clarifying that the names are followed by null character. mv would move items to targeted directory specified after -t flag. Overall, the whole command moved only regular files in the directory and its sub-directories to the target directory [8]. After I figured out how to move all the HTML article files into one directory for each sports website, I found that I should scrape the article parts from each HTML files and store them to a common txt file. I implemented the scraper in Python programming language, which would go through each file and find the articles according to HTML tags and get rid of all the HTML tags, Javascript codes, newline characters, and tab characters. Those articles are saved into a buffer string followed by a newline character and the string is written into a txt file. The resulting txt files have several hundred to several thousand lines, and each line represents one article. I formatted the resulting txt file this way because the index generation program accepts txt file formatted in this way to generate the Inverted Index. Given that the different websites have different HTML tags and formats, each website has a separate scraper. I struggled while implementing the scraper for certain websites because they caused additional newlines, even after I removed all the newline characters. I was able to discuss this problem with my CS 241 instructor, Yang Xu, and he told me that the carriage return character can cause newlines in txt files, and I can check it by using Vim Log Viewer. I found carriage characters in resulting txt files and I edited Python based scraper to remove them as depicted in Figure 1.

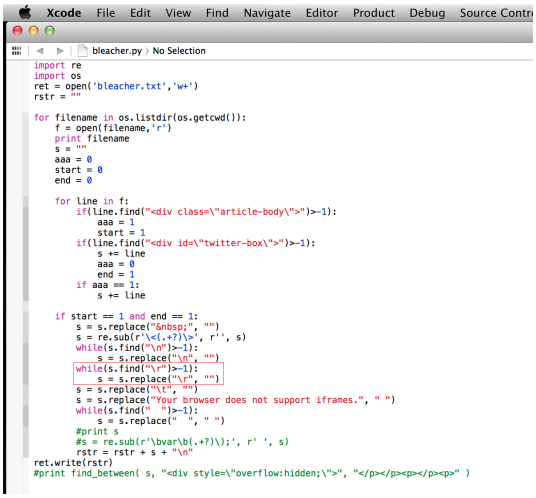

Figure 1: Python based scraper for Bleacher Report news articles including carriage return character removal

As a result, the crawler and scraper worked seamlessly together to generate txt files that include articles separated by newline characters. Brian connected the crawler and scraper by implementing an automatic updating program using a shell script.

4.2 Social Networking Integration

Our group's efforts were focused on Twitter for its prominent use in sporting events and how easily information is shared with the general public, rather than only with a smaller group of friends. Ideally we would like our tool to capture the highlights of sporting events across the world. Users would then tune their interest in what sport and/or query term they interested in. After narrowing the social media site down to Twitter, we researched the available methods for extracting the data from their website. Like most social media sites, Twitter provides one or more APIs to access the social messages or tweets that are being shared. For accessing only the tweets, the main two methods are Search API and Streaming API. Search API provides access to a large quantity of tweets to be downloaded in bulk, using a search query. Streaming API on the other hand provides a rate-limited flow of tweets based off of keywords/query as a live-stream, updating as new tweets come in. Given that ultimately we decided to do sentiment analysis on a bigger dataset, we selected Search API, as this allowed more flexibility in post-processing for what we do with the Twitter documents/"tweets". Search API allows a user to build a query and request tweets via HTTP GET commands. Queries have user-definable parameters to include either more recent tweets, more popular/trending tweets (or both), geolocalized results, language specific tweets, and how to iterate through the results. This is all part of the Twitter REST API. In order to leverage these pre-existing APIs, Twitter4J is designed specifically for Java applications that would like to pull/crawl tweets. In order to implement a program using Search API, we first needed to create a dummy Twitter account that can be provide access to application "tokens", allowing the program to log into the account and pull the tweets. This happened to be one of the most challenging issues to overcome. The documentation and process wasn't particularly straight forward, and without access/keys for a Twitter account, the application cannot connect to Twitter. Eventually we were able to work through the challenges and made a specific test account for this project and gave it access. After the Java application was able to successfully communicate with Twitter servers,, a single query or set of queries needed to be defined. We decided to build a list of predefined twitter queries ("baseball," "football," "basketball," etc.) and pulled this out as a user-configurable text file, "twitter.txt." Twitter4J then uses the search function in the Search API, by making an HTTP GET request with the specified queries. An ArrayList of QueryResults built based off the response. Each QueryResult contains a set of tweets, which have several fields, including text, names, retweet count, geo-location, etc. Given text content of the tweet is what we are looking for, tweet.getText() is then written out to a file. Similar to the web crawlers, this Java application dumps the documents into a plain-text output file, by default "tweets.txt," delimited by a new line. The Java application was then packaged in a JAR file with the necessary Twitter4J libraries, to provide a stand-alone application, with a user-adjustable set of queries. Essentially, this concludes the responsibilities of the Twitter/social networking integration [5].

4.3 Building Index

Apache Lucene [3] is a free software Java-based package allowing for indexing and search of text information. It provides a customizable set of functions for advanced anal-

ysis/tokenization of input text. Additionally Lucene can create an inverted index of a set of input documents. The output is a proprietary index for use with Lucene or programs/applications written to read Lucene indices. One of the primary functions of Lucene is the capability to query the index, with customizable scoring functions, allowing a user to either use a default/built-in function or implementation of their own.
Lucene was the obvious choice for our project, given its ease-of-use, capabilities, and familiarity with its use in class. File input begins by reading in a list of files (each containing many documents, newline delimited). The filenames are currently hard coded. The application will iterate over all documents, using a predefined analyzer/tokenizer. This includes the output of social media site Twitter. Each document is added to the index, one-by-one until all documents are accounted added to the inverted index. Tweets are stored in the same index, however they are using a different field "tweets," rather than "content" (used by regular articles from the crawled web pages). This is accomplished using the addDoc() and addTweet() methods respectively.
The output files of the combined steps above, is a proprietary Lucene inverted index. From this point the index is done being built. However, there is an additional built-in functionality in our indexing application that allows us to do test queries without the need for a particular front-end or back-end application. The test-query is by default disabled, but can be activated by uncommenting out a single line to execute the function test Query (accepting a String query) and print the output to the console. Additionally the query and field ("content" for articles and "tweets" for tweets) can both be changed to whatever value is desired to test the index. Currently these are hard coded, but they could obviously be pulled out command-line parameters fairly easily. The validity of the index is now checked.

4.4 Back-end Query Service

The back-end was a particularly challenging design decision that involved a significant amount of deliberation before reaching a decision. Selecting a framework for back-end services was difficult, because many options would require a significant amount of heavy-lifting/coding in order to handle the web requests for a query. Additionally this would have to send a response that could be understood by the requestor (ex. XML, JSON objects, etc.) Lucene was initially evaluated to perform the querying/scoring of the keywords provided by the front-end, and building a web service around this that would act as the intermediary between Lucene (Java) and the front-end (primarily HTML and JavaScript). There was a high-level of complexity and code would be needed in order to support the incoming requests and service a proper and timely response. Due to the these challenges, alternative solutions were investigated. Custom solutions involving JavaScript calling the Java code using a Applet/Servlet was first checked out, however it appeared to involve a lot of work and presented its own possible challenges. These include compatibility with Lucene libraries (required for querying the Lucene inverted index) as well as possible permissions issues accessing a Java application.
Consequently Apache Solr [4] was found to be an excellent choice, because this uses Lucene to do the querying and provides built-in back-end server capabilities out of the box. Solr is a Java-based full-text search server that provides all the indexing and search functions of Lucene. In fact Lucene is actually running under the hood of the Solr application. We selected this as our back-end server for these reasons.
Solr setup was fairly straightforward, however documentation is somewhat sparse and setting an ideal configuration can be challenging, where as just getting it running, in general, is much less of a challenge. First the inverted index created by Lucene in the previous section needed to be copied to the Solr package's data/index directory, including deleting/replacing any existing previously existing index files. Next the "schema.xml" file (a core Solr configuration file) needed to be updated to reflect the format of the new inverted index. Specifically the field names needed to be updated to match the content of this index. The main, important fields were then changed to "content" (again for sports articles) and "tweets" (for Twitter posts). Once these changes were made the front-end files are then copied to the "solr-webapp/webapp/" directory, in order to host these files externally. Once this setup is completed, the Solr service can then be started by opening the "start.jar" in the JRE [10].

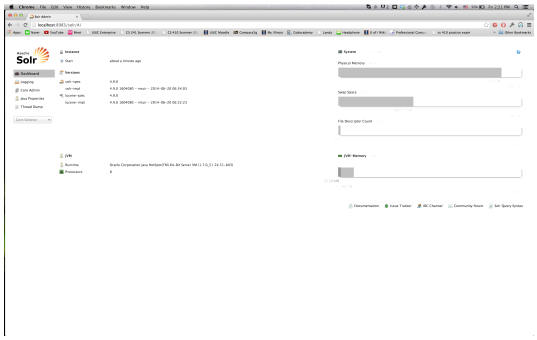

Figure 2: Solr administration page

By default search queries completed by the back-end are completed using the TF-IDF scoring function. This is the same functionality to standalone-Lucene. Queries can be completed and/or tested by using a HTTP request such as

"http://localhost:8983/solr/select?q=content:football&wt=json"

This query statement will request Solr to return results in the form of JSON objects for query football, looking on in the "content" field. After verifying that the back-end is returning results, the full system could then be tested. The front-end was then exercised by using the on-screen query text box. Results are then displayed to the screen using cleaned up formatting. JSON objects are converted to plain HTML, more details on this in the front-end section below. Likewise Twitter results are displayed on a separate pane to right of the main articles. Currently this is using the same query as the article search in order to provide a complementary articles and social media results. The complete system has now been demonstrated as fully functional.

4.5 Front-end

I (Christopher Choi) implemented the front-end part of the project. Front-end is the Graphical User Interface (GUI) part that provides graphical views of the software for users, so they can easily understand the function of the program and use it.

For our project, I decided to use HTML and CSS in order to design the front-end. I thought that using another language to build up the graphic interface would take more time than HTML and CSS, and I am very used to using HTML and CSS to design the website. In addition, I used a Twitter Bootstrap, which is a collection of designed CSS and JavaScript files, to make it easier and faster to implement the website, and it did actually save a lot of time. The website basically consists of two parts, the top jumbotron (or large image) consisting of the search box and the result boxes located on the bottom of the website.

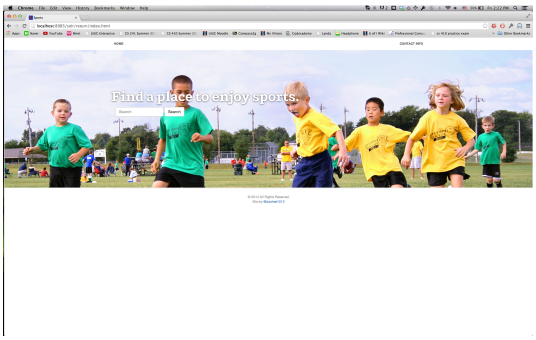

Figure 3: Front-end website design/structure before searching query

Figure 4: Front-end website design/structure after searching query

The search box accepts the query and the AJAX part in the HTML file would find the query result JSON file in server directory:

http://localhost:8983/solr/select?q=content:[query]&wt=json.

The AJAX code would take the content and tweet parts from the result file and store them into string. Then it would display them in the result boxes by adding html tags around the articles and tweets. I had a hard time figuring out the method to display both news articles and tweets, since I first implemented the AJAX function to print only the articles, which is called by HTML when the user clicks the search button. So, I tried to make another AJAX function to print tweets, but the HTML can only call one function, since the AJAX function requires returning false in order to complete its process, which would ignore the second function call. Therefore, I added a tweet-printing part to the article displaying function, and it took a long time to figure out how to put them together. Finally, I could display both articles and tweets in the result box, and the texts are highlighted when the mouse pointer hovers over the text [7].

4.6 Automated Update

A challenge introduced by the constantly changing nature of the websites latest content, providing an aggregated, up-to-date, view of the latest sports news requiring a method for producing quick automatic updates. How we tackled this was to ensure all of our different applications are portable and could be initiated via a script. In this case a simple shell (sh) script was created. This could easily be scheduled as a batch process and triggered at a specific intervals, example: kickoff updates every 30 minutes.

The update itself clears the old documents. Then moves on to crawl the websites using the wget methods above. It was challenging crawling multiple sites in parallel, so currently the script is written so that it completes crawling each website before moving on to the next. The script then generates the complete set of documents/articles from the crawled web page by using the associated python script for the particular site. This process is repeated for each of the websites. The output of this step will be a set of text files (one document/article on each line) for each website, ready to be indexed.

After completing crawling web pages, the tweets document also needs to be updated. The pre-packaged Twitter JAR then reads in "twitter.txt" (set of queries) as outline in more details above and outputs "tweets.txt" Finally the last major step is to update/overwrite the inverted index. The corresponding Lucene JAR is then called, using the various website documents as input, as well as the "tweets.txt" file. The final output is a Lucene index that can be copied into the back-end Solr service. Currently the script stops short of replacing the Solr index itself due to issues in replacing the index of a running search engine. Further details can be seen below in the future work section.

5. EVALUATION

For our evaluation, we manually searched 5 random sports-related queries and calculated the precision at top 10 results in news articles and tweets. The relevance of the results were judged by all the members in the project group.

According to Table 1, the total precision at 10 for both news articles and tweets are very high (0.96 and 0.90 respectively). Even though the query term "Christiano Ronaldo" had spelling error (Ronaldo's first name does not contain the letter "r"), it returned relevant articles and tweets with high precision@10 of 0.8 and 1.0 respectively. In addition, the precision@10 for tweets for the query terms "football" and "soccer" are low because they included tweets in Japanese (depicted in Figure 5).

From these results, we learned that the precision can be im-

Query	Precision@10 for News Articles	Precision@10 for Tweets
LeBron James	1.0	1.0
Christiano Ronaldo (note: incorrect spelling)	0.8	1.0
Football	1.0	0.8
Baseball	1.0	1.0
Soccer	1.0	0.7
Total	0.96	0.90

Table 1: Table for Precision@10 for News Articles & Precision@10 for Tweets

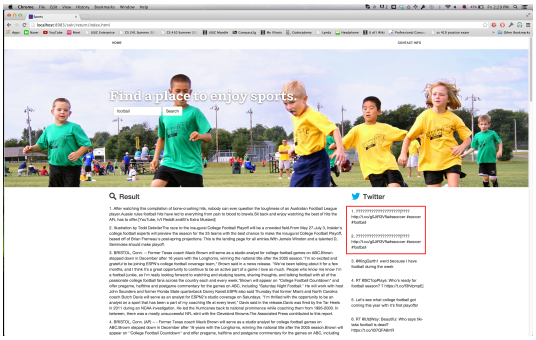

Figure 5: Resulting tweets include foreign language (Japanese)

proved by entirely filtering out news articles and tweets in foreign languages. Subjectively, we tested several additional queries on recent sporting news (such as "LeBron James") recent decision to move to Cleveland. We were presently surprised of the relevance and breadth of data we were presented on our site, when compared to pre-existing similar sites.

6. FUTURE WORK

6.1 Crawling

Even though the crawler and scraper are connected by an update-automation program, it could be improved if I implemented the crawler that only crawls and scraps the article parts of websites using a more focused program, rather than Wget. It would take a longer time to crawl the whole website using Wget, since it goes through all the links in the website, and is not limited to just the articles. Therefore, if the crawl can only crawl and scrap the articles from the sports new website, it would save a lot of time.
Also, it would be better to add more websites to crawl and scrap from. Currently, we have crawled four sports websites [1, 6], including

1. Yahoo! Sports - `http://sports.yahoo.com`
2. ESPN - `http://espn.go.com`
3. Bleacher Report - `http://bleacherreport.com`
4. NBC sports - `http://www.nbcsports.com`

These six websites provide many articles for our database. However, I believe the search results can definitely be improved by adding more websites to be crawled. This is a list of websites, which could be updated in our database later:

1. CBS Sports - `http://www.cbssports.com`
2. FOX Sports - `http://msn.foxsports.com`
3. Sports Illustrated - `http://www.si.com`
4. NFL - `http://www.nfl.com`
5. MLB - `http://mlb.mlb.com`
6. NBA - `http://www.nba.com`

Another thing that can be changed is that the scraper can scrap the article's URL and title as well so that the front-end can display them as well.

6.2 Social Networking Integration

For improvement in the social networking integration we would like to look introduce a few improvements across different areas:

1. Sentiment Analysis on Tweets
2. Inclusion of Twitter Stream API
3. Diversity in Social Networking sites (e.g. add Facebook, etc.)

Sentiment analysis is an area of text information and natural language processing centered around understanding the sentiment (or opinion) of a given document or statement. The intention is to gain more knowledge by looking at multiple user's opinions over some person, place, event, object, etc. Fields such as marketing or risk management have a very strong need and see great benefit from sentiment analysis. Such analysis of opinions allow these companies to alter their decision based off of the additional knowledge. Being a relatively new NLP (natural language processing) and text information field, it is currently under constant improvement and breakthroughs. Challenges include that sentiment analysis requires a deep understanding of semantic relationships between emotional expressions and the topic, coupled with an extensive knowledge in linguistics. Without the required knowledge, it can be difficult for humans to determine whether a sentence contains positive or negative opinions towards a subject.
Specifically for Twitter, sentiment analysis can provide an overall picture of user's feelings towards a particular player, game, tournament, extrapolated prediction of outcome, etc. Given this makes Twitter a clear candidate for sentiment analysis, it presents its own unique challenges. Challenges exist in the language used (slang, abbreviations, etc.), characters and punctuation used (#, , etc.) and generally short length of social messages. Emoticons, emotions formed by multiple characters in the form of smiley faces, sad faces, etc. are an additional piece of information that needs to be handled and understood. Machine learning would be the primary method for gauging sentiment, given its ability to be "trained" off a small set of human-judge data. Many researchers have started with Naive Bayes. This would be the

best baseline/starting point for working sentiment analysis on tweets [9].
Streaming API's ability to provide a live-stream of tweets could liven up the social networking aspect of Sports Central. Rather than only updating the tweets index at the same time as crawling the other pages, making use of Search API, this would allow constant new tweets to come in as they are posted, without delay. We would propose to include this in the Twitter pane of the front-end website, in place of pre-existing Search API data. From there the Search data could be repurposed to focus more on the aforementioned sentiment analysis/large data mining effort. Streaming API would be used to improve the speed of delivering the latest and greatest information to the users, as they are delivered.
Perhaps one of the most obvious improvements is to diversify the social networking sites to more than just Twitter. While Twitter has some obvious benefits outlined earlier in this report (easy to access APIs, short messages that are generally to the point, tweets visible to general public rather than only friends, etc.), diversifying to other social networks like Facebook or even more region specific ones like Sina in China (similar to Twitter). This could shift the content to either more geographically relevant information for people in a certain geo-location or perhaps focus the messages and opinions around what your friends are talking about, instead of a less personal general public. Many opportunities exist here, however the starting point is making a list of potential sites and slowly go through the and identify how easily each could be implemented using whatever APIs are provided by the respective companies.

6.3 building index

In terms of building the index, there is little room for improvement over using the current method. Lucene does a very good job of indexing the set of sports documents and tweets. Likely one of the few reasons that this should change would be if the amount of data is drastically increased, then further optimizations or alternatives could be explored such as Apache Hadoop (see future ideas for back-end query service below for more details). While no clear improvements seem to be required, further customizations on the text analyzer could be performed during the parsing/tokenization phase. Also the testQuery method could be "user friendly" by allowing a user to set runtime flags for test mode and the associate query and field to be tested.

6.4 back-end query service

There are several potential optimizations for improving the back-end. While we have shown through evaluation, that the results are desirable, there is always room for improvement to better serve end-users of Sports Central. Such optimizations may include using Hadoop's cluster/distributed system to maintain fast query times as the number of documents and websites indexed increases, while user-base simultaneously increases. Additional improvements could be made on the scoring function, through use of different models.
Apache Hadoop provides a distributed system for breaking the the task of creating the inverted index and querying into smaller tasks. These tasks involve Map-Reduce, a two step approach to reduce a large problem into several smaller problems, processed across multiple machines. As the number of documents increases and the load on the query server increases, both the time interval at which the web page/inverted index can be updated and the query times will increase. This will help by parallelizing both of these tasks. The end result would be a better user experience, aggregated news sooner to when it came out and faster query times.
By default Solr (and Lucene) use the TF-IDF vector space model for scoring document's relevance. Given that there are so many alternatives both in VS (Vector Space) models and LM (language model) models, it would make sense to implement and evaluate these scoring functions. As an example OkapiBM25 is an alternative method for scoring, including a document length normalization, which will penalize longer documents. Okapi also has several tunable parameters to improve performance on a given dataset. At minimum, Pivoted Length Normalization and Okapi should be tested to cover alternatives in vector space models. Likewise, language/probabilistic models should be evaluated in order to understand whether VS-models or LM-models are better suited for this type of data. This tuning can easily be achieved, since Lucene is the underlying framework in Lucene handling the queries and Lucene itself provides good means to control the scoring function.

6.5 front-end

While we believe the front-end is in good shape and provides an excellent interface for users, there are still several things that could be improved. The front-end can display the most frequently searched queries on the top of the website. And it can show five to ten most viewed articles from each sport by tracking users' search terms and clicking on the article. In addition, the title and URL of the articles can be displayed following the first few lines of articles, so the users do not have to search the article on Google to find it, and the title briefly shows the user what the article is about. In addition, the rating system (or like system by Facebook) can be used to show the most popular tweets and articles related to certain sports topics. Finally, the overall website design and structure can be improved by making it more responsive, according to today's website trends, and more interactive as well. A majority of these changes are minor and simple to implement, but incremental improvements that can be implemented to improve the experience of the end-user.

6.6 update automation

There are currently three primary challenges in automation that further improvements could be made. The first being improved error handling, second being inclusion of parallelism and the last being automating the full-system by replacing the inverted index in Solr.
Error-handling is non-existent in the current implementation for the shell script. Simple checks should be added, such as if the output files are even valid, files exist, etc. Infinite loop/timeout checks should be performed on the web pages that were crawled. Once these issues are identified, the automation should either re-try or flag the issue and avoid replacing the inverted index until the issue is resolved.
Parallelism could be very helpful in ensuring that entire process is able to happen within a relatively short amount of time. Separate processes/threads could be created for each

website, so that all websites are crawling at one time (or at least several at any one time). Complexity would be increased, because the next step of creating the inverted index would be heavily dependent on all website that are being crawled being completed. Network resources also be fairly limited and running several crawlers in parallel may also quickly find a bottleneck. Regardless, parallel crawling should be investigated.

Lastly, for the full-system to be automated end-to-end, the automation should go one step further and replace the inverted index of the Solr instance that is running, with the newly created index by lucene. The primary challenge here is doing so without interrupting service on Solr. When an index is manually replaced, the Solr server needs to be manually restarted. Further analysis would be required to understand how to get around this limitation, whether there is a feature for swapping the index that Solr supports or it involves swapping between two Solr instances. Swapping Solr instances might look like there is one hosting for users and the other one for updating the index, then the newly updated Solr instance becomes the host to users and the former host now waits to be updated.

7. CONCLUSION

Even though the users can use other search engines, such as Google and Yahoo!, to look up sports articles related to specific queries, it would be hard to also see people's reactions to these news articles or topics and capture trending information about those queries unless the article itself contains users' comments. However, our project also captures sports-related trending information from Twitter and has the potential to integrate many other great social networking sites. In addition, our tool produces the results very quickly, since the inverted index, provided by the back-end services, have been generated before the users search. The clean user interface, provides a side-by-side look at the articles written by journalists and professional sportscaster with the views of the general public. The relevance of the resulting articles and tweets have been evaluated using precision@10 for both news articles and tweets. This shows the strong And, the evaluation results show that our tool has high precisions of .96 and .90 for news articles and tweets respectively. Overall the Sports Central project was successful in doing what it set out to do, provide a one-stop-shop for sports articles and social media, spread across several sites. Areas of further improvement on taking this project to the next level have been identified, with the future possibilities of this site endless.

APPENDIX

A. DIVISION OF LABOR (FOR GRADING)

Person 1: Brian Beer: Implemented Java application to pull Tweets using a set of queries and store to file, created inverted index using crawled web pages and tweets, set-up/configured Solr for use with Lucene-made inverted index and created script in order to automate process of crawling to dumping new inverted index.

Person 2: Christopher Choi: crawl and parse news article and store to file, designed and implemented front-end graphic interface and search engine functions, wrote project report and made project presentation video.

B. REFERENCES

[1] Top 10 most popular sports websites in the world.

[2] Gnu wget 1.15 manual, 2011.

[3] Apache lucene, 2012.

[4] Apache solr, 2012.

[5] Getting started, August 2012.

[6] Top 15 most popular sports websites, August 2014.

[7] Yonik Seeley. Json response writer, March 2011.

[8] Anwar Shah. How do i move all files from one folder to another using the command line?, August 2012.

[9] S. Shahheidari, Hai Dong, and M.N.R. Bin Daud. Twitter sentiment mining: A multi domain analysis. In *Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on*, pages 144–149, July 2013.

[10] Kelvin Tan. Solr in 5 minutes, 2014.