COMP 4601

Project Report

Kanto - Music Search Engine

Hrishi Mukherjee | Yoni Kidanemariam

Introduction

Search Engines have become an integral part of life in the 21st Century. The ability to enter in a few words, and be returned with results that are interesting and useful to an individual, has transformed the world. Thus, we are living in an information age. Music, however has been in existence for at least 55,000 years. In fact, is has been found to exist in every known culture, in different forms and manifestations, since its creation. Kanto, the music search engine, dives into these two extraordinary fields, and brings them together to create a tool that helps people in search of music. It takes the intelligence of the search engine and combines it with the boundlessness of music. Imagine, a tool that allows one to perform simple dive and retrieval, from an ocean of songs. Unknown songs, forgotten lyrics, and debates over which lyrics are correct become a thing of the past. Kanto solves all these problems. It achieves this by using a combination of different tools and resources including: data retrieval of songs from the web, storing songs in a database, indexing song data, and storing in a directory for future querying. Users can search any string of words and instantly be displayed with a list of potential songs. Music is food for the soul, and no deserves to go hungry.

Background/Related Work

Currently, you can find various types of search engines with the similar purpose of providing a source to find music. Some of the biggest ones being Shazam and Genius. Shazam solves the exact same problem as Kanto, however it uses a different method. Instead of a user entering the lyrics to the unknown song, Shazam uses the audio of the song currently playing and creates a digital fingerprint of the audio. It then matches the fingerprint against a database of millions of songs. It is very fast and efficient, and incredibly user-friendly. However, the cons are that the song needs to be playing for it to work. Kanto does not require the song to be playing, but relies on the memory of the user inputting the lyrics. Genius, is also a music search engine with a similar goal. Its main purpose is providing music knowledge to people. Genius focuses on hip-hop and explains the deeper meaning behind the lyrics of a song. The problem with Genius is that it focuses more on rap music, while Kanto accepts all genres of music. However, Genius recently expanded their music collection and is trying to include more than just rap and hip-hop. Kanto, Shazam, and Genius all use different methods to the solve problems of unsatisfied music lovers. Shazam uses an audio clip to find a songs information. Genius has a unique goal of providing knowledge to the users, giving a better understanding of the lyrics in a song. Lastly, Kanto uses a string of words provided by the user to help them find what they are looking for, which is the music.

Problem Statement

A common problem that individuals face each day is hearing a song for the first time and enjoying the song, however end up missing the song title and losing the opportunity to hear it again. For example, this happens a great deal in a car listening to the radio or at a restaurant and hearing that song brings out strong emotions, but the name of the song is unknown. One could try to ask around to friends and family if they know what song they heard, giving them a sample through humming or even singing. However, this method is not usually successful nor is it efficient. Individuals encountering a problem like this need a tool that will allow them to find the song they are looking for with little information, like a few lyrics or the artist's name.

Solution/Implementation

The problem stated in the section above occurs regularly. To accomplish solving this problem we developed a music search engine. A music search engine works just like any other search engine, but the difference is that it is used strictly for finding songs.

CRAWLING

Creating a music search engine requires a large amount of data. The first step of development was creating a web crawler. The purpose of the web crawler was to crawl the web, while extracting data. However, this is much easier said than done. First, we needed to find the information we wanted on the web, which required thorough investigation. We searched through the web looking for sites that looked promising, for the data we needed. Once a promising website was found for data extraction, the crawler browsed through the website. The main website for extracting data was "www.azlyrics.com". The structure of the website was the easiest for us to work with, and was used for much of the data extraction.

Kanto uses Crawler4j to crawl the web. Crawler4j is an open source web crawler for Java. It has a very simple interface for crawling the web, which made it the perfect candidate for crawling. We took Crawler4j and made a few adjustments and additions to create our Music crawler. The Music crawler contains 2 classes, the Contoller and MusicCrawler. The Controller class has many functions and parameters that are set, but most importantly it specifies the seed of the crawl, and starts the crawler. The MusicCrawler class performs two methods for the crawler, which is deciding which URLs should be crawled and handling the page that is fetched to be processed for data by us. Together, these two classes make up the Kanto Music Crawler.

PARSING

Now that the data was in place, the next step was searching through each page and retrieving only data necessary for storing. Parsing data from a foreign web page required a very specific formula. The web pages from “www.azlyrics.com” had a very similar structure for each song, however some HTML pages had a few minor differences. Consequently, a specific set of instructions needed to be placed for the crawler and parsing components to work harmoniously. The way the website was set up, the most optimal parsing technique required browsing by the artist’s name. Therefore, a specific artist was chosen, and the information was then hard coded in the required fields of the Constants class before each crawl. The crawler would browse through all the songs of one artist, while simultaneously extracting data from each page.

Kanto uses JSoup to parse the music data. JSoup is an open-source Java library, which provides an API for extracting and manipulating data from HTML page. Thus, when the crawler visited and fetched the song page, we used JSoup to connect and parse all the data necessary. The data included artist name, song name, and lyrics. Each element was found in different areas on the page, therefore we strategically picked out each value needed efficiently and effectively.

STORING

The next step in the data extraction system was storing the retrieved data into a music database. We implemented a Song class which represented a song, along with a SongCollection class which connected to the database, and controlled the activity of the database. The database was used as an intermediate step before the indexation of each song. As stated earlier, first an arbitrary artist is chosen, then the crawler begins browsing over the songs of the chosen artist. While crawling through the song of the artist, the essential data is extracted from each song including: song name, lyrics, and artist's name. A Song document is then created, where the value of each field is stored with the information extracted from the HTML page. Once the Song document is created, the SongCollection class is called and stores the song in the Kanto Database, adding it into the songs collection.

Kanto uses MongoDB for its database. MongoDB is an open-source No-SQL database. It allows users to store a collection of songs easily and securely. This is done in the SongCollection class. This class creates the new database and adds Song documents to the song collection. Mongo supports all the song fields (artist name, song name, lyrics, and id). Thus, the SongCollection class allows songs to be easily stored, and simple retrieval when needed. As mentioned earlier, the database was used as a transitional step between crawling and indexing. Also, it acts as a backup compartment in case the index directory is ever corrupted.

INDEXING

Once we had data to work with, it was time to start indexing. To achieve this, all the songs from the database are migrated and processed through the indexing phase. We were easily able to take the raw data of the songs, transform them with a few steps, before adding them to the index. First, a list of all the Song documents is retrieved from the database. The information of each song is tokenized, normalized, stemmed, and stripped of stop words. The processed information is then stored in a new index document, and stored in the index directory.

Kanto uses Lucene for indexing the songs. Lucene is an open-source high-performance information retrieval software library, written in Java. It is equipped with excellent tools for indexing, which is what we required to build Kanto. We created a Lucene class which performed all the methods required for indexing and querying. Once all the songs were retrieved from the database, a method from the Lucene class is called, which processes the documents to be indexable. This was done by creating a new Lucene document, and storing each variable as a Textfield, which is the indexable field used in Lucene software. This action is done for each song. Finally, the new indexed Lucene song document is added to the index directory for querying.

RESTful WEB SERVICE

We now had all the songs stored in the index directory. Our next development was the searching component, which involved the actions of RESTful web service. The purpose if the web service was to allow the user to perform a query for their desired song. Thus, the Client enters in the search words, the search words are then passed through to the Server. The Server receives the data and makes a call to a method from the Lucene class, which performs a query on the string in the index directory. The method returns a list of the top hits, or top songs, in order from best to worst. The results are then outputted to the user.

Kanto uses RESTful web services to handle the connection between the client and the server. REST is an architectural style which is based on web-standards and the HTTP protocol. The REST architecture fitted the requirements of the Kanto music search engine, and so chosen for implementation. We created a class called Engine, which handled the REST web services between the client and server. The server provides the access to the indexed documents and the client retrieves the resources according the query.

Results

We are proudly able to say that our music search engine was a success. A user can search for a desired song, with highly accurate results. The search focuses solely on the lyrics of the song. However, the user can enter the name of the artist and will be displayed with a list of their songs. By performing multiple tests, we were able to better understand the accuracy of the search, and see the methods of searching that gives better or worse results. First we discovered that querying words that are rarer, or more abundant in the lyrics of a song, will yield better results. Also, the more information that the user gives about a specific song, the service will return with better results for the user. Additionally, we noticed that querying lyrics with the chorus tends to give more accurate search results. This is because the chorus is usually repeated multiple times throughout a song.

The quality of the music search engine directly depends on the amount of data that it contains. Therefore, the more data, the better. If a user is looking for song, which has not been indexed then they will never be able to find the song. This means that, for the search engine to be successful, it needs to be constantly storing music at the same rate as the artists releasing their music. Otherwise, users will be searching, but will not find what they are looking for. Currently, we have extracted and stored a grand total of songs. This is nowhere near the amount of songs that a music search engine should have, since there are at least 97 million songs that have been released officially in the world. This means Kanto has a long way to go in regards to quantity of music data, however currently it works very well with the music it contains. It will be interesting to see how much the accuracy of the query will be affected as the amount of songs in the collection grows.

The most difficult challenges we faced with developing Kanto was crawling the web, finding good websites to parse data, and parsing the essential information while ignoring the rest. Finding websites to crawl was a difficult task we faced at the beginning of development. Some of the problems we encountered were finding a website with a structure that was easy for us to dive into, understanding how it is setup so that we may extract data efficiently. However, most of the websites we encountered were very difficult to understand in terms of its architecture, and parsing data seemed like a nightmare. When finally settling on a good website to use, and tested out our Music crawler on it to see how it works. The crawler was browsing through the web pages at a rate past the threshold according to the website's server, which ended up getting us blocked. To avoid this problem, the crawler's politeness was set to 20 seconds. Once we had the crawler fetching pages, parsing information became the next difficulty. Our search engine requires only a few fields for indexing, however the web pages contain an abundance of unnecessary data, useless for our specific needs. Therefore, we created a method to parse only the data needed, while ignoring the rest of the data.

Conclusion and Future Work

The goal of Kanto was to create a solution for the music hungry souls. By combining different tools and resources together this goal was accomplished. From crawling the web to parsing data from a web page. Storing data into a database, which grew inch by inch every day. Using Lucene to index the songs and store into an index directory. Creating services that allow a user to perform a query and be presented with the top hits. By bringing these components together, we created a very effective tool that helps solve a simple problem, but can still make a big impact on an individual's day.

Kanto is a successful project, but still has a massive road ahead. Building a Music search engine requires an enormous amount of data. This is the biggest challenge we faced, and will face in the future. Our future goal should be finding the most efficient way to obtain music via web. Also, optimizing the retrieval system so that it always returns very accurate top results.

References

- Genius – About Genius. (n.d.). Retrieved April 09, 2017, from https://genius.com/Genius-about-genius-annotated
- Shazam. (n.d.). Retrieved April 09, 2017, from https://www.shazam.com/company
- Apache Lucene Core. (n.d.). Retrieved April 09, 2017, from https://lucene.apache.org/core/
- Jsoup Java HTML Parser, with best of DOM, CSS, and jquery. (n.d.). Retrieved April 09, 2017, from https://jsoup.org/
- Y. (2017, March 30). Yasserg/crawler4j. Retrieved April 09, 2017, from https://github.com/yasserg/crawler4j
- History of music. (2017, April 04). Retrieved April 09, 2017, from https://en.wikipedia.org/wiki/History_of_music
- J. (n.d.). The Origin of Music. Retrieved April 09, 2017, from http://www.ancient-origins.net/ancient-places-europe/origin-music-00972
- Lars Vogel, (c) 2009, 2016 vogella GmbH. (n.d.). Quick links. Retrieved April 09, 2017, from http://www.vogella.com/tutorials/REST/article.html#rest_overview