Intelligent Web Report

Ben Carr & Luke Heavens

Introduction

This assignment required the construction of a dynamic web application capable of querying the social web. API's made available from sites such as Twitter, Foursquare and Google Maps have been used in combination with Java Servlets, a web interface and database to allow the collection, storage and visualisation of publicly available data. In this report, each aspect of the system will be detailed in turn, outlining the issues faced, the design choices made, how well the requirements have been met and any limitations that were present. The system produced is able to perform all of the four functions outlined in the specification and more, including using live streaming and storing information in a database.

Documentation

1.1 Querying the social Web

Searching for Tweets

Tracking public discussions on specific topics and performing basic queries is the first feature implemented in the system. One or more search terms (e.g. a keyword or hashtag) are provided by the user and matching tweets are returned. Search results can be restricted to a specific geographic location (geolocation), defined by a latitude, longitude and radius, which can be enabled or disabled using a simple checkbox.

Data is sent to a Java Servlet which in turn performs a Twitter query with the keywords and geolocation (if requested). Matching tweets are ranked using a combination of both popularity and recency. Once returned and displayed, the full profile of a tweet's tweeter can be fetched (using a separate AJAX call triggered by clicking their name) and displayed in a modal window. This process is explained later. In addition, up to 10 retweeters can be fetched using a "See who retweeted this" link and another AJAX call. The link to display retweets is only shown if the tweet has been retweeted.

During the implementation of this feature, it was discovered that Twitter4J does not support the string id field of a tweet object. This caused problems, as Javascript does not support numbers of type 'Long' and as such, tweet id's were rounded down. To correct this, the id's are converted into strings and serialised separately.

This query meets all the requirements set in the specification and performs all tasks as expected. Currently, the only limitation with this system is the hard-coded cap on the number of tweets returned. Given additional time however, the system could be modified to allow the user to request additional tweets to be fetched. Also it could be improved to allow retweeter profiles to become links to full profile modal windows.

Searching for Keywords

The system also allows the tweets of several users to be fetched and examined to determine the most frequent terms used amongst them. The web interface allows the user to enter the ids of up to 10 users, the number of desired keywords and the number of days to search.

Upon receiving these parameters, the Java Servlet makes a query to Twitter to fetch the all the tweets made by each user over the requested time period. Each tweet is then stripped of all non word-characters (i.e. not a-z,A-Z,0-9,_), with the exception of hashes (#), at symbols (@), apostrophes and spaces. This is done to reduce the number of word variants and invalid words. After removing undesirable characters, each tweet is divided into terms. Each term is then checked for an appearance in a stop list. Its presence would indicate that the word is not useful at determining the content of a tweet and ignored. This stop list a vital part of the system as without it, connectives and other context-free words would rank most frequently. Remaining words, unfiltered by the stoplist are stored in a complex inverted index. This allows the frequency and use of the words to be easily compared.

Following the construction of the inverted index, the highest ranking terms are obtained. Both the ranked terms and those terms left unranked are transferred to a simpler data structure to be either sent back to the web page or stored in the database. In addition to the ranked frequent terms, the user objects are returned to the web page to be displayed.

The inverted index is built using a complex data structure comprised of hashmaps. These were chosen for their speed, allowing the system to quickly and repeatedly determine if a given key is already present and to easily obtain the associated values. The stop words are held in a hash set for the same reason.

This query meets all the requirements set in the specification and performs all tasks as expected. Currently the user objects for all the usernames entered are returned along with the terms due to low cap on the number of users that can be queried. If the system was extended to allow more users and more terms, it may become necessary to hold back data until explicitly requested (e.g. holding back user data until the web interface asks for the breakdown of term occurences).

Searching for User Check-ins

It is possible to find all the venues that a user has visited within a specified number of days by providing a Twitter username and the number of days over which to search. In addition to this data, a hidden form field is also given to the Java Servlet which determines whether the user object is needed. This is required only on the first request when streaming, and therefore saves calls to Twitter in subsequent requests.

Visited venues are fetched by first finding all tweets made by the user over the defined time period that contain Foursquare information. If a Foursquare link is found, the URL is expanded and the user and check-in parameters are extracted. These are used in a Foursquare query to find the check-in details and venue details. Valid venues are returned listed and plotted on a Google Map using markers placed using venue latitude and longitude coordinates. Clicking markers on the map displays the name of venue plotted.

Requesting information over zero days triggers the streaming API. After first returning any venues visited today using the normal search API, a stream starts to record new tweets in a list using a listener. A new AJAX call is performed every 20 seconds to check for new venue/check-in data and to update the web page accordingly. If a request is not received for 60 seconds, or the user performs a different query, the stream terminates.

This query implements all requirements but could be improved in a number of ways. Firstly, the venue photo displayed is the most recent user uploaded image which may not represent the venue well. An alternative service could provide better photos such as Google or Flickr. Secondly, the system currently relies solely on Foursquare to determine where a user has been. Other online sources or tweet geolocation information could be used in addition.

Searching for Venues

The final main feature of the system allows venues to be obtained given a name and/or geographic location (latitude/longitude/radius). For each venue found, all the users who have visited in the last X days (where X is supplied by the user) are also returned.

Collecting this information is done by expanding URLs in fetched tweets to find valid Foursquare check-ins and venues. This information is stored in two hashmaps. The first linking venue id to venue information, the second, mapping venue id to tweets with valid check-ins. Both hash maps are serialised and sent back to the page and can be logically connected through the venue id. In addition to displaying the collected venues and check-ins, each venue is plotted on a Google Map and users can be clicked to show their profile. As before, entering zero in the days to search field initiates the streaming API. This works in the same way as the previous feature.

During the development of this query, it was learned that Foursquare only allows venue managers to see which users have visited their venue resulting in the implementation described above. Whilst it would be more efficient to access the users from the venue directly, the API has these limitations in the interest of user privacy. A second issue concerning the Twitter streaming API was also uncovered whereby multiple listener filters could not be used as each filter (e.g. search term/geolocation) is treated as a logical OR. Therefore, the system filters by geolocation if available and then matching tweets are filtered afterwards separately.

This query meets all the requirements and returns all necessary information. The limitations are identical to the previous query described above concerning the reliance on Foursquare.

User Profile Modal Windows

Information about users can be viewed through the use of modal windows. Elements from their Twitter profile are fetched using an AJAX call containing the user id (which is stored as a data tag in the display user profile links). Information fetched includes profile picture, name, username, description, follower/following counts, banner image and location. A separate AJAX call is used to get the users 100 most recent tweets. The two calls are separate as sometimes some information is already known to the webpage and once HTML is generated, subsequent AJAX calls do not need to be made. In addition, the tweets are also iterated through to find the latest geolocation, which is displayed if present.

1.2 Storing Information

Whenever the system receives some user, location or frequent term data, it will be stored in the system database. Using a second web interface it is possible to view data that has been collected. All the information requested in the requirements is stored and is accessible either by entering a username or venue name.

1.3 Web Interface

The system is controlled via a browser based interface and Java Servlets. The browser interface comprises of two HTML files (queryInterface.html & databaseInterface.html) which display the forms and the results of queries. Javascript and jQuery are used to dynamically change content on the page, and all results are displayed in the “pageContent” div element. Twitter Bootstrap is used for CSS styling and various Javascript plugins such as the forms accordion. jQuery handles all the AJAX calls to the server and processes all the results, including displaying any errors. The calls are directed to the Servlet which in turn determines which queries to run.

1.4 Quality of the Solution

The solution to the assignment has brought together a number of advanced technologies such as AJAX, JSON, Java Servlets and Threads with standard ones such as MySQL, JavaScript, HTML and CSS. In addition to these technologies several widespread libraries have been used such as jQuery, Bootstrap, Gson and Twitter4J. The blend of these technologies allows a system with improved the usability and quality to be made.

All requests to the servlet are performed using AJAX via jQuery and all data responses are serialised as JSON strings using the Gson Java package. This allows communications to and from the server to be made whilst keeping the user on the same page i.e. to update content dynamically. This is particularly useful when the streaming API is used. JSON allows complex data objects to be serialised for communication between the client and server. Together, these systems allow for a cleaner user experience.

Validation of user entries has been done using a combination of client (Javascript) validation and server (Java) validation. This reduces server errors, improves the user experience, reduces the chance of security breaches, prevents unnecessary traffic and provides helpful error messages.

All queries to the social web return some data which has been identified by the requirements as desirable to store in a system database. Multithreading has been used whenever the system wishes to communicate with the database. This means that rather than wait for the database communications to terminate before completing a query and responding to the AJAX call, the servlet can return the requested data whilst the database communications continue on the server. The primary motivation for this is the decrease in time that the user is waiting for the page response.

Errors are handled throughout the system and a thrown and caught at different depths in the program. If an error cannot be recovered from, the HTTP status code is set to 500 and the web page will convey this to the user. The native Java logging system has been implemented to record the details of any errors that occur. By adjusting the server settings, the location of the log output can be altered. This approach also allows log entries can also be made at different levels, for example ‘severe’, ‘warning’ or ‘info’.

Finally, all communication with the database is performed using prepared statements. This protects against SQL injection attacks.

1.5 Additional Features

As well as the required functionality, the system includes a few additional features. The first main feature is nearby popular venues. When a venue search is performed and geolocation data is provided, the system will look for any other venues in the area that the user may be interested in. This is done using Google Places and a third-party package for Java. The Google Places API takes a latitude, longitude and radius and returns a list of venues. This list is ordered by prominence by default, which is ranked using Google’s place indexing and the location importance, and will provide the user will relevant suggestions of other venues to visit.

Conclusion

This assignment shows that the web, particularly the social web, is a huge source of information in determining human behaviours and events. The biggest problem with these sources lie in the connection between sources of data and understanding it. Getting a venue image from Google Images is trivial, but ensuring it is the same venue as one mentioned on Foursquare for example is less so. Another difficulty is analysing text. Simply mentioning the name of a location in a tweet does not necessitate that the tweeter is there. Despite this, it can be seen from this system that all sorts of useful and interesting information is obtainable.

Extra Information

No extra configuration is needed to use our system. Some good good examples of test queries can be seen in appendix A.

Appendix A

Although the system is intuitively easy to use, it is worth noting some test data to get the best results from the system. This test data is displayed below:

Form	**Data**	**Comments**
Tweet Form	query = “#sheffield” OR query = “from:bbcnews”	The second query will return tweets which are more likely to have been retweeted
Keyword Form	usernames = “bbcnews channel4news itvnews” keywords = 10 days = 2	This will return the most common words used in news tweets in the past 2 days
User Check-in Form	username = “stevewoz” days = 2	This gets check-ins for a frequent Foursquare user
Venue Form	name = “meadowhall” lat = 53.41391389 lon= -1.4125872 radius = 5 days = 2	This will get venues and check-ins at or near Meadowhall and will also show popular venues in the area