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

1.1.1 Tracking Discussions

Overview & Issues

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

Motivation

Data from the form is sent to a Java Servlet via AJAX, which performs a Twitter query using the requested keywords and optional geolocation information. The motivations for using these technologies are explained in section 1.4. The Twitter API allows the ranking algorithm of matching tweets to be altered. Our implementation organises tweets using a combination of both popularity and recency. This method has been chosen as mere popularity may stagnate results and using newness alone may not return the most relevance tweets. After obtaining tweets the servlet serialises the tweets into a JSON string which is returned to the client-side JavaScript. This enables complex data structures to be easily transmitted and for data to be rendered on-screen. Each tweet is displayed in a style which resembles the Twitter interface in an attempt to provide users with a sense of familiarity. Clicking on a user's name or profile picture, displays their full profile in a modal window (after fetching data in a separate AJAX call). This process is explained in more detail in 1.1.5. In addition, details of up to 10 retweeters are fetched if the user clicks a "See who retweeted this" link via another AJAX call. The link to display retweets is only shown if the tweet has been retweeted at least once to avoid confusion.

Requirements

The requirements specified that the system should be able to track public discussions on different topics. By allowing the user to input search terms, hashtags and geolocation data, the system is able to search for tweets which are relevant to their search. This feature therefore meets all the requirements set in the specification and performs all tasks as expected.

Limitations

The only limitation with this implementation is that currently, only a maximum of 10 tweets are returned by the query. Given additional time however, the system could be modified to permit the user to request additional tweets to be fetched. This could be implemented in the form of a "Load more tweets" bar at the bottom of the page for example. When clicked, this could instruct the system to obtain additional results.

1.1.2 Frequent Keywords Amongst Users

Overview & Issues

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. The desired keywords will then be returned and displayed, ranked on their total occurrence count. One issue, realised early-on, concerned the abundance of connectives and other context-free words that were highly ranked. A stop-list was introduced so that these words were filtered out, leaving keywords that are able to provide useful information to the user. This stop-list was reused from the Text Processing module COM 3110.

Motivation

Using the form parameters and AJAX, the Java Servlet queries 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 in a-z, A-Z, 0-9), with the exception of hashes (#), at symbols (@), apostrophes, underscores and spaces. This is done to reduce the number of word variants and invalid words whilst being sympathetic to Twitter conventions. After removing undesirable characters, each tweet is split 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. Remaining words, unfiltered by the stop-list are stored in a complex inverted index. This allows the frequency and use of the words to be easily compared. The inverted index is built using a complex data structure comprised of hash-maps. 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-list words are held in a hash-set for the same reason. 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 stored. The ranked frequent terms along with the user objects are then returned to the web page using JSON and AJAX. Individual user occurrence counts are displayed in a collapsible summary below each term. These can be expanded individually or all at once.

Requirements

The requirements specified that the system should be able to identify the topics users are discussing within a given time period. Using the system, users can see the top keywords for the group of users specified, along with their individual and combined occurrences. This makes it easy for users to discover commonly discussed topics. The number of IDs that can be inputted are capped to 10 and the days must be greater than 1 as per the requirements. This feature therefore meets all the requirements outlined in the specification and performs all tasks as expected.

Limitations

Currently, the user objects for all the usernames entered are returned alongside the terms. 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 occurrences). Due to a low cap on the number of users that can be queried (outlined in the requirements), this implementation is suitable for the current system. In addition, the number of keywords that can be returned has been limited to 99. A similar extension to that described in 1.1.1 could be introduced to permit the user to “Load more keywords”. A final limitation concerns the Twitter API’s tweet fetching speed. Whilst the processing time is fast, query results have to be obtained in batches. This prolongs the users waiting time.

1.1.3 Places a User Has Visited

Overview & Issues

It is possible to find all the venues that a user has ‘checked’ into within a specified number of days. The user provides a Twitter username and the number of days over which to search. Any Foursquare check-in information that appears in the specified time frame is extracted and used to produce the results. New data can also be streamed live and appended to the page.

The main issue encountered with this feature concerned a 7 day search restriction within Twitter’s API. This prevented tweets older than approximately seven days from being obtained through a Twitter search. To overcome this, the system accesses a specific user’s timeline and, using paging, fetches the user’s tweets in blocks of 100 until the defined date limit is reached. The second issue was due to the Foursquare CompactVenue objects returned by user check-ins. These objects lacked information about a venue such as its description. The system is therefore forced to perform a second call to retrieve the venue’s CompleteVenue object.

Motivation

In addition to user-supplied data, a hidden form field is also passed to the Java Servlet which determines whether the Twitter user object is needed. This field is included to prevent the user object - only required on the first request - from being sent multiple times during streaming mode. This therefore reduces calls to Twitter in subsequent requests.

Visited venues are fetched by first finding all tweets made by the Twitter user over the defined time period. If a Foursquare link is found within a tweet, the URL is expanded to enable the extraction of Foursquare user and check-in parameters. These are in turn used in a Foursquare query to obtain check-in details and basic venue details. For reasons explained in the Issues section, Foursquare is again queried for the respective CompleteVenue object. Collecting additional venue specifics from Foursquare enables the system user to see much more useful information.

As well as displaying and listing returned venues, their locations are plotted on a Google Map using markers and the venue's latitude and longitude coordinates. The map provides an excellent visual representation of the venues and their whereabouts. Clicking markers on the map also reveals a pop-up box displaying the name of that venue. Foursquare has been chosen as it provides a link between a check-in and a venue's information.

If the user requests information from 'the past zero days', two separate queries are performed. The first query checks for any venues visited in the current day. The second query utilises the Twitter Streaming API to open a stream which subsequently begins to record new tweets 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. A trade-off between server load and detection time is made but, considering the frequency of the average user's tweets, this delay was deemed acceptable. If an AJAX request is not received for 60 seconds or the user performs a different query, the servlet will terminate the stream. This prevents unnecessary server work.

Requirements

The requirements dictated that the user should be able to discover the venues visited by a Twitter user in a specified time period. The results in this system are clearly displayed both as a list and also visually on a map. Using the streaming feature also enables live check-ins to be detected by the system and added to the results. Therefore, the feature implemented in this system meets all the requirements given.

Limitations

This implementation could be improved in a number of ways. Firstly, the venue photo displayed in the results is simply the most recent user uploaded photo to Foursquare for that venue. Being a user uploaded image, it may not represent the venue accurately. Alternative services, such as Google and Flickr, could be used in future revisions to provide more relevant photos. 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. Unlike Foursquare, uncertainties arise as to whether a user actually visited a venue rather than simply the surrounding geographic area. Using data from multiple services would also require advanced techniques to ensure correct venue correspondences. Finally due to the numerous Foursquare calls required to collect detailed venue information, results can take a lengthy period to return.

1.1.4 Users Who Have Visited A Place

Overview & Issues

The fourth feature of the system allows venues to be obtained given a name and/or geographic location. For each venue found, a list of users who have visited it in the last X days are also returned. During the development of this query, it was learned that Foursquare only allows venue managers to see the users that have visited their venue. Whilst it would be more efficient to access the users from the venue directly, this API restriction means that check-ins must be extracted from a Twitter search. A second issue, concerning the Twitter streaming API, was also uncovered which prevented the use of multiple listener filters. This was due to the fact that each filter (e.g. search term/geolocation) is treated as a logical OR. The system therefore filters by geolocation first, if available, and then filters by venue afterwards.

Motivation

Information is collected by expanding URLs in fetched tweets to find valid Foursquare check-ins and venues. This information is stored in two hash-maps to increase the speed of data retrieval. The first hash-map links the venue ID to its detailed information, whilst the second links the same venue ID to tweets with valid check-ins. Both hash-maps are serialised into JSON strings and sent back to the page using AJAX, where they are displayed in a list. In addition, each venue is plotted on a Google Map so users can better visualise the results. Clicking on a user's name or profile picture will reveal a modal window of their profile, where their description and tweets can be displayed. Entering zero in the "days to search" field initiates the streaming API. This works in the same way as the previous feature in section 1.1.3.

Requirements

The requirements specify that the system should display who is visiting venues or have done so in the past X days. The system allows users to search by venue name and/or location, displaying results both in a list and on a map. Therefore, this feature meets all the requirements.

Limitations

All the limitations in section 1.1.3 are also shared with this feature, in particular, those concerning the reliance on Foursquare. In addition, when the system is streaming and a new check-in is identified at an existing location, it is added to the top of the list creating a duplicate entry. A final limitation is that only a maximum of 15 venues can be retrieved. This is to prevent overloading Foursquare with queries but could be extended by adding a "load more venues" bar.

1.1.5 User Profile Modal Windows

Information about Twitter users can be viewed in modal windows throughout the site by fetching their profile data using separate AJAX calls. These windows resemble the Twitter profiles by displaying the user's profile picture, name, username, description, follower/following counts, banner image and location. This provides the system user with a sense of familiarity. Another AJAX call is used to get the users 100 most recent tweets. These two calls are separated since information may already be known to the webpage. In addition, the tweets are also iterated through to find the latest geolocation, which is displayed if present. The previously mentioned "load more tweets" buttons could be introduced here too.

1.2 Ontology and Triple Store

Overview

Whenever the system receives some user, location or frequent term data, it is stored in an RDF file using the Jena API. This follows a custom ontology based on the schema.org data structure.

Motivation

The schema.org data structure was chosen for two reasons; Firstly, it can represent most of the data in this system and secondly, because it is already used by major search engines. Additional classes and properties in the ontology use the URI of the servlet as a namespace to ensure that they are unique. Twitter users are represented by their Twitter profile page URI, and similarly venues by their Foursquare venue page URI, enabling cross-site correspondence. The data is stored in an RDF file in the servlet directory where it can be easily accessed via a separate HTML interface. This returns the data in the HTML5+RDFa 1.1 format explained in section 1.3.

Requirements

The requirements stated that the system should store data using RDF triples according to an RDFS ontology. The data should also be accessible in both HTML and RDFa. This system has met these requirements and allows the retrieval of users and venues using a separate interface.

Limitations

The only limitation of this system is that by using a file to store data, only one connection can be opened at a time. This may cause concurrency issues if multiple instances of the system are running on one server and trying to access the datastore simultaneously.

1.3 Web Interface

Implementation

The system is controlled via a browser based interface and Java Servlets. The browser interface comprises of two HTML files (queryInterface.html & datastoreInterface.html) which allow data to be queried from the web or from a datastore respectively. JavaScript and jQuery are used to dynamically change page content whilst Twitter Bootstrap is used for CSS styling and various JavaScript plugins such as the forms ‘accordion’. jQuery handles all the AJAX calls to the servlet which in turn executes queries and either returns results or encountered error information.

RDFa

Data retrieved from the RDF triple store is returned in the HTML5+RDFa 1.1 format readable by both humans and robots. This allows information to be displayed for interpretation by a human user but also for RDFa parsers. Data objects are defined using the “about” and “typeof” attributes. Properties use the “property” attribute and are used to represent relations to literals and also between two objects. These conform to the RDFS ontology and the namespaces used.

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, JavaScript, HTML and CSS. In addition to these technologies, several widespread libraries have been used such as jQuery, Bootstrap, Gson and Twitter4J.

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 permitting the user to remain 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 is 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. Multithreading has been used whenever the system wishes to communicate with its datastore. This means that rather than wait for the communications to terminate before completing a query and responding to the AJAX call, the servlet can return the requested data whilst the datastore 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 are thrown and caught at different depths in the program. If the system cannot recover from an error, the HTTP response status code is set to 500 and the web page will convey that an internal server error has occurred 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’.

1.5 Additional Features

As well as the required functionality, the system includes an additional ability to display nearby popular venues. When a venue search is performed and geolocation data is provided, the system will look for other venues in the area that the user may be interested in. This is done using Google Places and a Java API. The Google Places API takes a latitude, longitude and radius and returns a list of venues. This list, by default, is ordered by prominence. This means results are ranked using a combination of Google’s place indexing and the location’s importance.

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 lies in connecting these different sources of data together. 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 visiting that location. 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 for our system. Example test queries are given in appendix A.

Appendix A

A working copy of the system can be found at the following location:

http://tomcat.dcs.shef.ac.uk:41032/aca11bmc/queryInterface.html

The RDF file is deployed with the server and is therefore stored with the live server files. The data can be retrieved with the system but if the file needs to be accessed directly, it can be found at the following location:

{aca11bmc_root}/mytomcat/webapps/aca11bmc/TripleStore.rdf

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

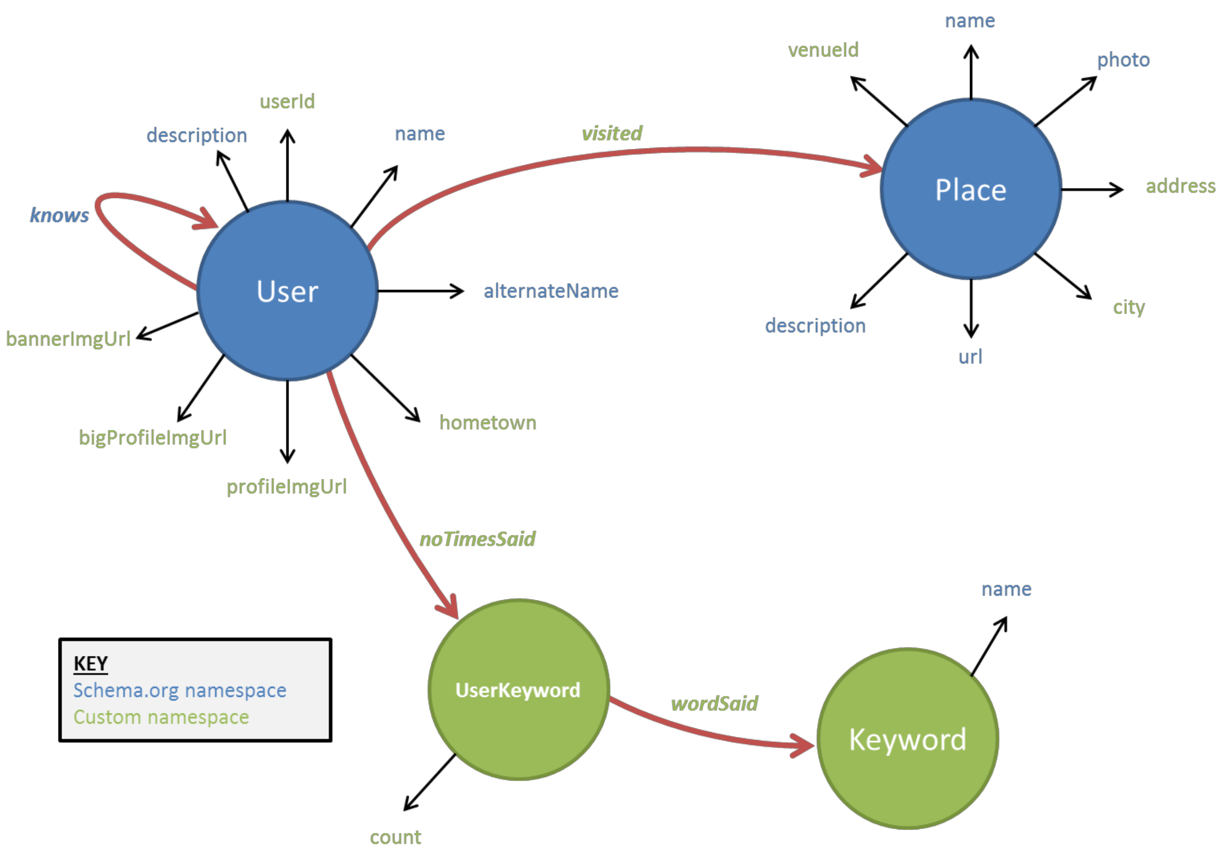
Keyword Form	usernames = “bbcnews channel4news itvnews”
	keywords = 10
	days = 2
		This will return the most common words used in news tweets in the past 2 days

User Check-in Form	username = “stevewoz”
	days = 2
		This gets check-ins for a frequent Foursquare user

Venue Form	name = “meadowhall”
	AND/OR
	lat = 53.41391389
	lon= -1.4125872
	radius = 10
	
	days = 2
		This will get venues and check-ins at or near Meadowhall and will also show popular venues in the area

Appendix B

Below is the RDFS schema used in this system.


userId
description
name
visited
knows
User
alternateName
bannerImgUrl
bigProfileImgUrl
profileImgUrl
hometown
noTimesSaid
venueId
name
photo
Place
address
description
url
city
name
KEY
Schema.org namespace
Custom namespace
UserKeyword
wordSaid
Keyword
count