Project Report

**Group Members**: Brady Ericksen, Timothy Anderson, Jordan Ross, Casey Lee Murphy

**Topic: #1** Implementing a Tweet Clustering System

**Contribution of Each Member:**

* **Jordan Ross:** Tweet collection (number of tweets to collect, authentication, querying for tweets), implementation of Twitter API (twitter4j), implementation of a Spring Boot web service to handle http requests from a webpage, user interface using JavaScript, HTML, CSS, JQuery, and Bootstrap, implementation of word cloud through a JQuery library called jqcloud.
* **Brady Ericksen:** Cleaning of tweets (multiple regular expressions, string replacement, trimming and lowercasing using said regular expressions and the built in String functions), creation and editing of Project Report throughout development. Compilation of screenshots to showcase working project.
* **Timothy Anderson:** Creation of project skeleton, cleaning of tweets (multiple regular expressions, string replacement) implementation of HashSet to store tweets and remove duplicates, setting up the groups GitHub, primary presenter during class presentation, creation of the readme, group PowerPoint, java logger, and java Doc.
* **Casey Lee Murphy:** Clustering of tweets (using RxNLP’s Sentence Clustering API), taking in a HashMap of the collected cleaned tweet data and converting it to a form that the API can properly consume, returning resulting JSON data to front-end web application responsible for displaying the cluster information.

**Overall Architecture:** This is a java application. We made it a Spring Boot web service so our UI in the browser could talk to our Java code. We used Maven as our build tool. Our web service accepts HTTP requests (like a servlet). We send an HTTP request containing the word to collect tweets about from the browser to the web service to collect tweets. Specifically, we collect Tweets using the twitter4j API. We pass the tweets to a function to clean the tweets and remove duplicate tweets. Then we send it to a class that handles the clustering and formatting. Finally, we return the JSON containing the tweets to the browser where the JSON is parsed and the tweet clusters are displayed as an interactive word cloud that responds to click events. When you click on a word in the word cloud it displays the cluster sentences associated with it.

**System Specifics:**

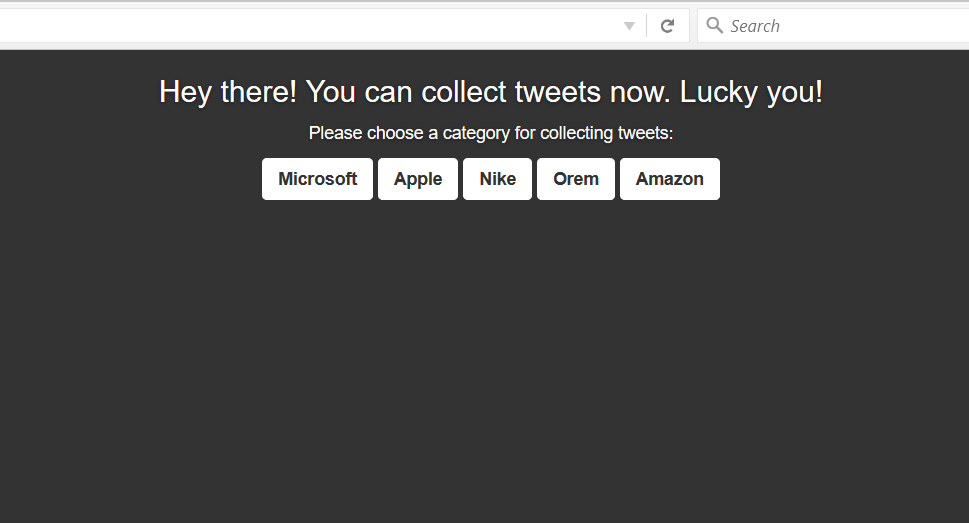
* **Tweet Collection:** Twitter4j was our API of choice to collect tweets. We set up authentication to twitter with a key and access token, then we used their API services to ask for and collect 100 tweets at a time. We call a method to clean the tweets (see below) and then added them to a HashSet.
* **Tweet Cleaning:** Using Regular Expressions and the String’s replaceAll() function, we filter out all Retweets, Usernames, and Hyperlinks. The Regular Expressions in question are as follows: **RT:?\s@[a-zA-Z0-9\_]+:?\s?** – This is to remove the first set of potential retweets within the Tweet. This checks to see if a portion of the string begins with RT: or RT, followed by a whitespace, followed by @ and any username immediately after, followed by one or zero colon characters in order to remove nested retweets, followed by whitespace. We next have the following regular expression: **(http[:/.\w]+)** – This is to remove all hyperlinks. It looks for any string starting with the hypertext transfer protocol, which is then followed by any number of colons, slashes, periods, and words.

We next have a backup regular expression to remove any retweets not caught by the first retweet replaceAll() call – **RT:? [a-zA-Z0-9\_]+:?**Our next replaceAll() is to remove any usernames in the file: **(@)\w+** - as they are not pertinent to our topic based clustering. Finally, we remove any non-ascii characters (for example, Japanese or Chinese text) in order to meet the String conditions that the RxNLP’s Sentence Clustering API requires - **[^\x00-\x7F]**

We also remove tab characters and slashes using **\t** and **\\\\ -** Removing tabs for the purpose of readability and slashes because they can cause problems in the clustering. We follow this with **[!/'"#$%^?+:=&\*~<>()\_-]** which is designed to remove the majority of non-alphanumeric characters. We remove brackets, periods, and commas using **[\[\.\],]** and finally we remove newline and carriage returns using **[\r|\n]** - We return the cleaned String to be placed into our HashSet. The purpose of the HashSet is to prevent the insertion of duplicate Tweets and to provide an easy collection to iterate over during Clustering.

* **Tweet Clustering:** To take care of the task tweet clustering, we created a separate class called TwitterClustering.java. This class serves two main functions: taking in the tweets and converting them into a form that can be easily consumed by the API, then actually calling the RnXLP topic clustering API to do the bulk of the work. In more detail, the method that returns the end result takes in a HashSet of tweets that were collected from a separate class. The HashSet is then passed to another method within the TwitterClustering class which iterates over every item of the HashSet and puts them into a formatted string. This string is then passed back to the API calling method. The API calling method provides a key and then specifies the type of data we would like returned from the API. In our case, the API is returning a string object which contains all of the clustered tweets with their associated subjects. Finally, this string is passed back to the TwitterController class to be formatted in a way that can be displayed to the browser front-end.
* **Display:** These are the components used in the UI: HTML, JavaScript, JQuery, Bootstrap, jqcloud and CSS. To collect tweets we make a POST to our Spring Boot web service. Once the data is received back from the twitter and clustering services, JavaScript takes the data, does a little formatting, and sends it to the jqcloud library to display the word cloud.

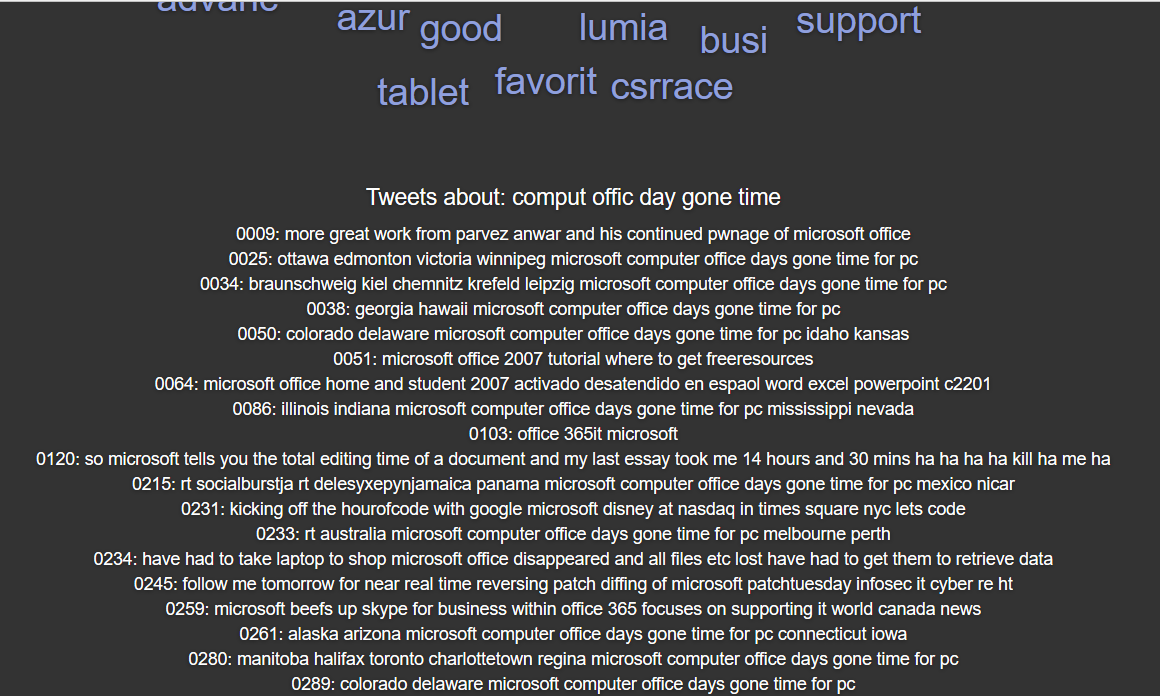
**Types of Topics:** For our demo, we have the following hashtag options to choose from: Microsoft, Apple, Nike, Amazon, and Orem. Any other Twitter Hashtag you would like to search would work as well. We have chosen the previous five topics specifically for our demo in order to prevent strange user input as well as provide a pleasing topic-specific theme in our GUI.

**Screenshots:** This is the primary screen you see after running the program and navigating your browser to localhost:8080

After clicking ‘Nike’:



After clicking on a Cluster such as “comput office day gone time”, a list of tweets in that cluster are listed:



**Modules Used:** Apache Maven (brings in Twitter Streaming API – twitter4j which we use for acquiring the Tweet libraries and reading in said Tweets, Spring, and the RxNLP’s Sentence Clustering API which is used to cluster the cleaned Tweets) as well as built-in libraries.

**Classes Created from Scratch:** The majority of our classes are created from scratch, with calls to the various API’s we used. **Twitter.java, TwitterClustering.java, TwitterController.java,** **Application.java**, and **ApplicationInitializer.java** were all custom made. As stated, only the API’s we used listed above were not-created from scratch.

**Problems Faced:**

* **Cleaning Tweets:** We faced issues in the regular expressions when trying to remove retweets. It turns out that retweets have a few different structures, so we included a way to check for each variation of a retweet (single, nested, included whitespace), which we then removed. We also had issues occur when Tweets with non-english characters were read in. We ended up using regular expressions to remove non-ASCII characters.
* **Installing Maven:** It took us multiple tries on multiple computers to get Apache-Maven working for each of us. Environment variables didn’t work correctly on Mac. Windows worked better, but still had issues with User Permissions, environment variables, and Path information.
* **Version Control:** We faced issues installing and configuring Git, configuring Eclipse to use Git (eventually we all ended up using IntelliJ), and pushing and pulling updates correctly. The .gitignore file had a few issues when we started. Now the git project is easily pull-able and runnable. The dependencies are that you must have Java and Maven both installed and in your system variables.
* **Twitter API:** Since the Twitter API would only give back 100 tweets at a time we kept track of the smallest tweet id from the last 100 tweets collected and set that as the max tweet id for the next call to the API. The next call to the API would give the last 100 tweets that came before the tweet with that tweet id. If we did not do this we would have received the same 100 tweets with each API call.
* **Tweet Clustering:** When we began programming for the clustering of the cleaned tweets, it seemed quite simple from a high level view. However, once we got down into the specifics of the API implementation, some problems arose. The main issue came when we tried to send the cleaned tweets to the API for the first time: we would constantly get a “Bad Request” reply. The reply didn’t contain any specific information as to why it was a bad request, it simply didn’t like the way we had formatted the strings sent to it. After some investigation, we found that the API didn't allow for line breaks in the string object sent to it. After removing the line breaks in the tweet cleaning portion of the code, we were able to send off the request and get a good reply from the API server…sometimes. Again, we were a bit confused as to what was causing the problems because sometimes it would work and other times it wouldn’t. This problem was solved once more by applying more in-depth cleaning of the tweets by removing non alphanumeric characters, and most non-essential information for the tweet. Once we arrived at this point, the clustering code functioned (and continues to function) as expected.

**Extra Credit:** We implemented a user interface with a word cloud featuring the ‘top’ words for each cluster. This provides a clean user interface for the users, which is easier to read and understand for the average person than XML code. The words in the word cloud are clickable to drill down to view the cluster sentence about that word.