Twitter Sentiment Analyzer

Alex Susma

June 2016

Contents

[2 Purpose of the project 3](#_Toc452924290)

[3 Design & Implementation 3](#_Toc452924291)

[3.1 Program Flow 3](#_Toc452924292)

[3.2 Implementation 5](#_Toc452924293)

[3.3 Sentiment Analysis 5](#_Toc452924294)

[3.3.1 Examples 6](#_Toc452924295)

[3.4 Design Choices 7](#_Toc452924296)

[3.4.1 String::find vs Regex 7](#_Toc452924297)

[3.4.2 Vector vs unordered sets vs trie 7](#_Toc452924298)

[3.4.3 Static vs Dynamic Dictionaries (learning) 7](#_Toc452924299)

[3.4.4 Multithreading vs Single Threading 7](#_Toc452924300)

[3.4.5 Scheduled to run or run continuously 7](#_Toc452924301)

[3.4.6 32bit 7](#_Toc452924302)

[3.4.7 Searching for stop words 7](#_Toc452924303)

[3.5 Open source usage 8](#_Toc452924304)

[3.5.1 TwitCurl library 8](#_Toc452924305)

[3.5.2 RapidJSON 8](#_Toc452924306)

[3.5.3 Negative, Positive and Stop Words lists 8](#_Toc452924307)

[4 How to run 8](#_Toc452924308)

[4.1 Presence of input files. 8](#_Toc452924309)

[4.2 #define EXECUTION\_PERIOD 10000 8](#_Toc452924310)

[4.3 #define MULTITHREADING 9](#_Toc452924311)

[4.4 #define OUTPUT\_TWEETS 9](#_Toc452924312)

[5 Sample run 10](#_Toc452924313)

[5.1 Sample run multithreading 10](#_Toc452924314)

[5.2 Sample run single threaded 11](#_Toc452924315)

[6 Limitations 12](#_Toc452924316)

[6.1 Spaces in company names 12](#_Toc452924317)

[6.2 100 Tweets Max 12](#_Toc452924318)

[6.3 Sentiment Change 12](#_Toc452924319)

[6.4 Sentiment Analysis 12](#_Toc452924320)

[6.5 Repeated Tweets 12](#_Toc452924321)

[7 Future improvements 12](#_Toc452924322)

[7.1 Static + dynamic dictionary. 12](#_Toc452924323)

[7.2 Weighted words 13](#_Toc452924324)

[7.3 Expressions 13](#_Toc452924325)

[7.4 Add more companies related words to dictionaries 13](#_Toc452924326)

[7.5 Further preprocessing of tweets 13](#_Toc452924327)

[7.6 Hold more results in memory and show a trend 13](#_Toc452924328)

[7.7 Remove repeated tweets 13](#_Toc452924329)

[7.8 Adding more customization for tweets searched 13](#_Toc452924330)

[7.9 Make certain options available via parameters 13](#_Toc452924331)

# Purpose of the project

The purpose of this project is to present a solution to one of the common machine learning problems: text sentiment analysis. The program will read Twitter entries containing mentions about the requested companies and then perform a sentiment analysis. The output of the program is a score showing the overall sentiment about each company, and, if ran continuously, the change in sentiment since last run.

# Design & Implementation

For the scope of this program, several design choices had to be made. First of all, the programming was done using C++ in Visual Studio 2012 on Windows 8. Compromises and tradeoffs had to be made to ensure the project is finished in a decent amount of time. (Section 3.4). Next, several open source libraries and tools were used to help speed up the development. Please see section 3.5 for details.

## Program Flow

The program will read 4 files at startup: a list of negative words, a list of positive words, a list of stop words and the input file containing the companies to be researched.

After input, the program will establish a connection to Twitter and start searching for each company. Once the results come back, the software will parse the JSON reply and extract the tweets. Tweets are preprocessed to remove unwanted characters, words and other unnecessary parts. With the bare bones tweets, an analysis of sentiment is performed via simple matching with the positive or negative words. A count is held for each tweet which will provide the final sentiment for that tweet. For a company, many tweets are analyzed and the total sentiments are accumulated and displayed at the end. While a complex problem, the solution chosen is straight forward. The program is mostly linear, however the actual search and analysis of tweets is done in parallel via multithreading, 1 thread per query. This provides a considerable increase in execution speed. See section 5 for sample runs in multithreading vs single threading execution. The following diagram explains the program flow.

Figure 1 - Program Flow

## Implementation

To begin, the very first task done was to put down the basic requirements of the program and analyze them. After the core idea of the requirements was understood, a basic flow was drawn on paper.

At this point, a few tasks were identified:

1. Connection to Twitter
2. Searching and parsing Twitter responses
3. Finding a way to analyze sentiment

The first part was to find a way to connect to Twitter from C++. TwitCurl library was found and used to get a connection. While it seemed simple at first, a few problems arose while trying to implement it. First of all, getting all the include files and libraries to link properly was not as trivial as it may have seemed. A few extra steps were required to set up an application on the Twitter Dev site and give access to it via OAuth keys. Due to the implementation of TwitCurl, certain options in the Twitter queries were not available. The code taking care of sending the query was changed in TwitCurl and a newer version of the library was generated. The extra features allow the use of certain flags to limit the tweets returned to English only and to remove entities from the JSON response for less data transferred.

The 2nd part of the problem was to find a C++ JSON parser. RapidJSON seemed like a reasonable choice and it was quite easy to implement. Reading through the tutorials proved helpful, and shortly after, the program was able to search and parse the text from the Twitter responses.

Moving forward the decision was made to come up with a simple algorithm to figure out the sentiment of each tweet and then cumulate the sentiments from all the tweets of a specific company (Section 3.3).

If left running, the program will run every X minutes as per the requirements. When a company analysis is made, if previous results are available, a comparison is made to see if there is any change in sentiment.

## Sentiment Analysis

The approach chosen is to have dictionaries containing positive and negative words. Each word in the tweet is searched for in the dictionaries and if a match is present, the count for that specific sentiment is increased.

After some research, it was decided that the preprocessing of tweets before analysis is a good step to take. Code was written to remove parts of tweets like: urls, stop words, @usernames, #, numbers and other special characters. The tweets are made lowercase to speed up comparisons of words.

After each tweet is processed, it is analyzed and a sentiment is assigned to it based on the absolute difference in positive and negative words matched. A count of 0 is neutral. After each tweet is analyzed, a company total per sentiment is generated by simply adding up the total number of positives, negatives and neutrals.

Given the nature of tweets and the limited dictionaries, most of the tweets end up being neutral. To avoid having to show that all companies have a neutral sentiment, a system was put in place.

The algorithm will take into consideration the total number of tweets. If the difference between positives and negatives is more than 10% of the total number of tweets analyzed, the system will choose the sentiment as being the higher one. For example, it can be stated that the sentiment for a company with the following results is mostly positive if we do not consider the neutral tweets. Out of 100 tweets, the algorithm will check if either positives or negatives are larger by 10% (10 tweets) than the other. In this case the difference between positives and negatives is larger by 25, so the following company is rated positive.

Positive tweets 30

Neutral tweets 65

Negative tweets 5

If the results would resemble the following set, a neutral rating would be given since there is no clear difference(less than 10% difference) between positives and negatives.

Positive tweets 18

Neutral tweets 70

Negative tweets 12

### Examples of individual tweets analysis

Raw tweet:

***RT @alexsusma I like my #Samsung phone however I do have 3 problems with it. Check this link: http://www.yyy.zzz/image***

Preprocessed tweet:

**i like my samsung phone however i do have problems with it check this link**

Positive words matched: “like”. Positive count = 1

Negative words matched: “problems”. Negative count = 1

Total sentiment = 0 => Neutral.

Raw tweet:

***Wow this Apple phone is #amazing. It has the greatest battery life.***

Preprocessed tweet:

***wow this apple phone is amazing it has the greatest battery life***

Positive words matched: “amazing” and “greatest”. Positive count = 2

No negative words matched. Negative count = 0

Total sentiment = 2 => Positive.

## Design Choices

### String::find vs Regex

Most of the tweets preprocessing, searching and replacing could have been done using regex, however, string::find was the simpler and faster choice.

### Vector vs unordered sets vs trie

All the dictionary words are stored in vectors of strings. Since dictionaries are static and do not change, a vector made sense. Searching is done using Binary Search for a log(n) performance. Given the relatively small number of words, this was considered acceptable versus the more complex or space inefficient alternatives.

### Static vs Dynamic Dictionaries (learning)

The dictionaries used are static and loaded at runtime. The files are sorted so that binary search makes sense on them. For the scope of this project it was decided this was good enough. The alternative would have involved providing a set of tweets from which the system would learn and generate its own dictionary.

### Multithreading vs Single Threading

Given the fact that multiple companies need to be analyzed using the same method, parallelizing the search and analysis was a good decision. As seen in the sample output, it provides a considerable increase in performance. Since no shared data is really accessed, synchronization was not an issue.

### Scheduled to run or run continuously

There were 2 approaches to making the program run multiple times. First one was to keep the program running and make it pause for X minutes then perform another analysis. The second approach was to schedule the program and make it start from the beginning every X minutes. It was decided to go with the first approach. The program will keep on running and perform the analysis until stopped.

### 32bit

While in today’s world most applications are 64bit, due to the compilation complications of the open source code, I had to run the program in 32 bit release (with debug info).

### Searching for stop words

Right now the program will check if a stop word is present in the stop words list and if it is not it will be searched for in the positive or negative lists. The fact that we check for the stop words before the main lists does not eliminate a search, however it might increase performance. The stop words list is much smaller than positives/negatives which means a stop words search will take on average less time than searching for unwanted words in the larger lists(positive and negatives).

## Open source usage

To help with the development of the program, several open source components were used.

### TwitCurl library

As per the description from the website, TwitCurl is a C++ library for Twitter API. The library uses cURL to handle http requests. The library supports the latest Twitter API v1.1, SSL and JSON. This library was used to connect to Twitter via OAuth and to search for Tweets.

Available at <https://github.com/swatkat/twitcurl>

### RapidJSON

RapidJSON is a fast C++ JSON parser. Easy to use, this parser helped in the parsing of Twitter JSON responses.

Available at <http://rapidjson.org/index.html>

### Negative, Positive and Stop Words lists

These are public lists of common negative, positive and stop words in English language. They were used to identify different sentiments in the tweets.

Stop Words: <http://www.ranks.nl/stopwords>

Positive and Negative Words: [https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon](https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html%23lexicon)

# How to run

Several files, defines and values need to be provided for the program to run as desired.

## Presence of input files.

Make sure the files required for input are present in the directory. If you run the program compiled from Release folder, please copy the 4 files in Release. If the program is compiled and ran from Visual Studio, keep the files in the main project folder.

Stop-words.txt, Negative-words.txt, Positive-words.txt and inputCompanies.txt

The file inputCompanies.txt contains the list of companies to be researched. Please input one company per line and use the very first line to define how many tweets to include in the analysis. As per the Limitations sections of this document, please keep the number of tweets between 1 and 100.

The program will use a hardcoded twitter account for testing.

## #define EXECUTION\_PERIOD 10000

By default, the program will run every 10 seconds. This can be changed in the file MLSentiment.cpp

## #define MULTITHREADING

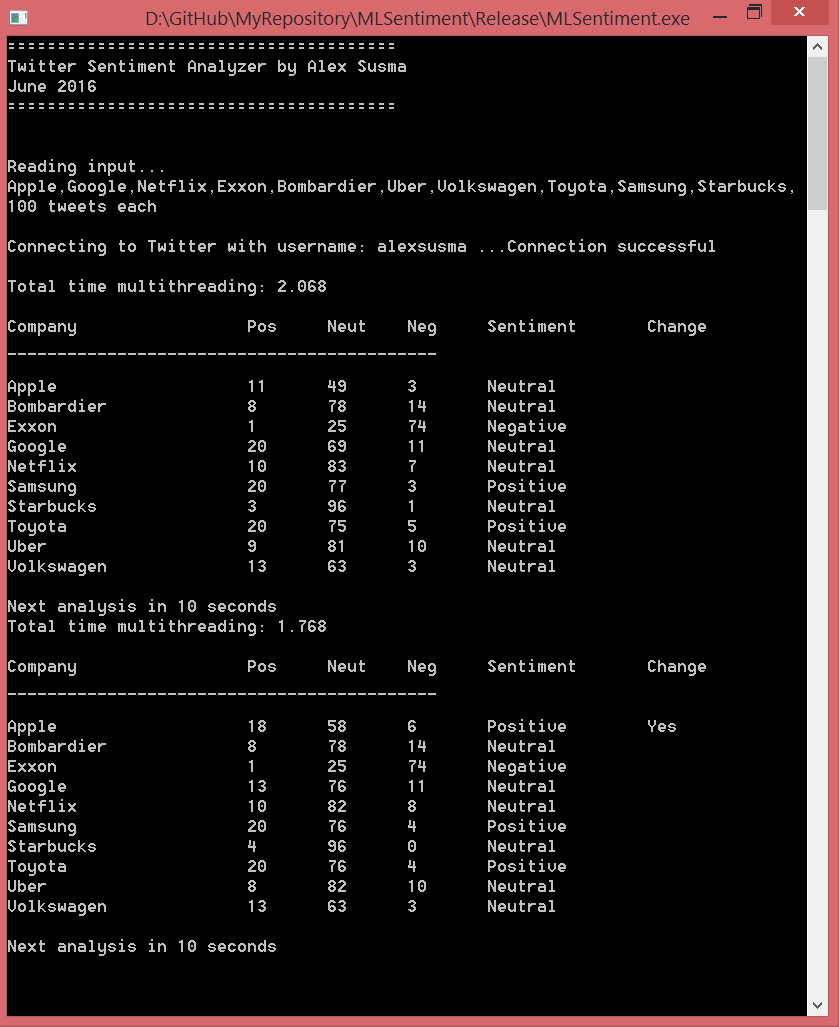
By default, multithreading is defined. If this define is removed, the program will run the queries sequentially. This value is in MLSentiment.cpp.

## #define OUTPUT\_TWEETS

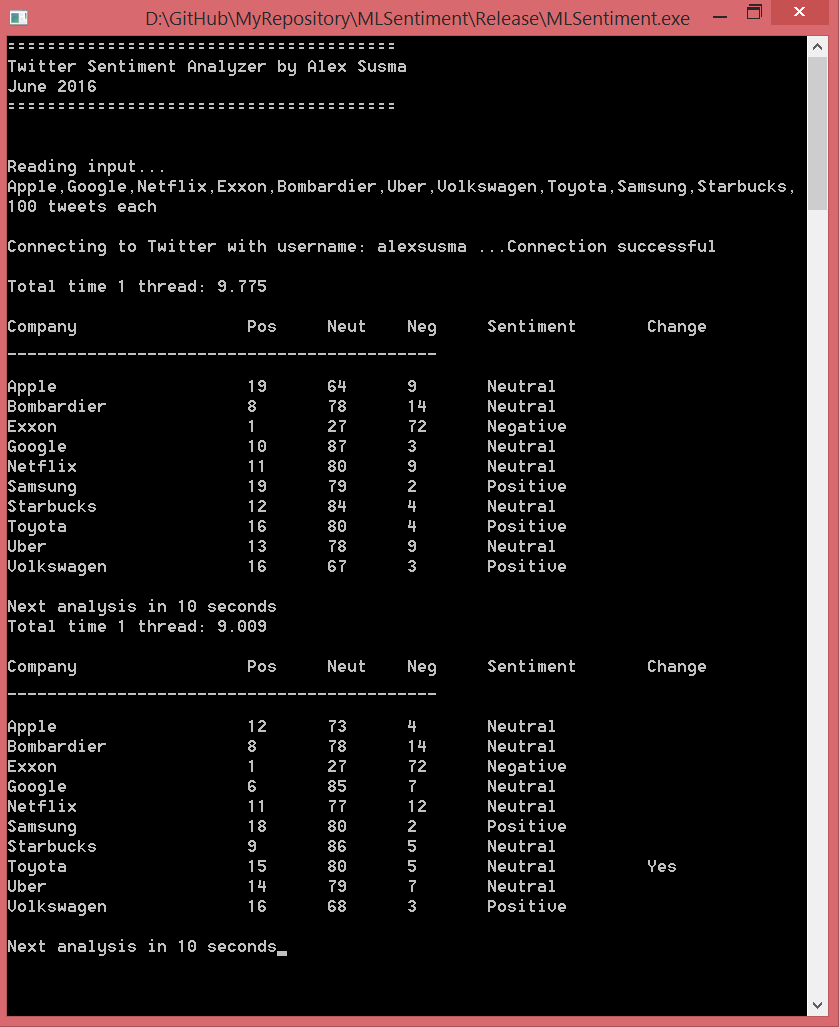
If this is defined, a text file named Tweets.txt will be generated. The file contains the raw tweets for each company and the sentiment associated to each one.

# Sample run

## Sample run multithreading



## Sample run single threaded



# Limitations

Due to different constraints, the program has several limitations

## Spaces in company names

The program was designed to accept spaces in company names. The software will identify them and replace them with %20 which is the required format for http requests. However, for unknown reasons, the Twitter API returns an error when %20 is passed in the url.

## 100 Tweets Max

It seems that TwitCurl and the Twitter API can return a maximum of 100 tweets per query. While this does not seem to be documented, it is the case. Furthermore, sometimes, the query can return less than the asked number of tweets.

## Sentiment Change

The program will determine if the sentiment for a company has changed after comparison with the last set of results. The program won’t show a trend of sentiments over time. The program will not be able to make comparisons with previous results once shut down. The program will store in memory only the last set of results.

## Sentiment Analysis

The nature of the tweets and the fact that the sentiment analysis algorithm is quite result in most of the tweets being neutral. However, a sentiment can be determined if one looks at the positive and negative counts.

## Repeated Tweets

Sometimes the query returns the same tweet many times. This can bias the results.

# Future improvements

If more work were to be put in this project, a few key improvements could be made.

## Static + dynamic dictionary.

A set of training tweets clearly identified as positive/negative could be passed to the program to help it learn several keywords.

## Weighted words

Each word in the dictionary can be assigned a weight. For example Good = 1, Awesome = 2 etc.

This would put more weight on certain words and allow the correct classification of tweets.

## Expressions

While a more complex approach, analyzing expressions could lead to better results. For example, mixing weight with expressions we could differentiate between something that is “Good” vs something that is “Very good”.

## Add more companies related words to dictionaries

Right now the dictionaries contain regular words. Since we are performing company sentiment analysis, more words should be added to reflect sentiment expressions for companies. This could include more terms related to financial and managerial performance. This was partly implemented manually.

## Further preprocessing of tweets

Removal of more special characters could lead to improved performance since those words would not be included in the search.

## Hold more results in memory and show a trend

For example the program could be made to hold more past results in memory and show how the sentiment changed over the last X runs.

## Remove repeated tweets

This would require checking every tweet against all the previously read tweets to make sure doubles are not permitted.

## Adding more customization for tweets searched

Right now the tweets we search are limited to English language only and they are a mixed of recent and old popular tweets. Further improvements could make other flags available through the TwitCurl library. Flags of importance could be the location of the tweets or the period of time when they were sent.

## Make certain options available via parameters

For example the sleep time between executions could be passed as a parameter instead of a #define