Product Popularity Predictor: Implementation and evaluation of a sentiment analysis tool for determining popularity on the web

Ugonna E. Nwakama

Abstract Ever since companies have been announcing as-of-yet unreleased products, there has always been a push to try measuring the public's reaction to the product. These days, most of the opinions and criticisms of the public are voiced on the Internet. This project aims to explore the concept of using data obtained from the web to determine the popularity of an unreleased product by collecting data about how much web-media sources mention the product. The purpose of this project is to try to establish a correlation between the intensity of web discussions of a product and the product's success. The proposed application will collect data from social and news media. It is possible to extend this concept to monitor the popularity of an already existing product.

***Index Terms*—popularity prediction, popularity detection, consumer products, sentiment analysis**

I. Overview

The web is full of people seeking information and others giving information. Companies reach out to both current and potential customers to raise awareness about upcoming and existing products. Individuals seek out product reviews to gauge the value of a particular product. All of this kind of activity takes place in social media, blogs, journals or news media sites. However, this entire "buzz" is dispersed into too many different locations on the web.

This project aims to detect the "buzz" surrounding technology product names on various forms of media. As a part of the project's motivations, I want to be able to form a basis to establish a correlation, or lack thereof, between the activity that takes place on the web about a product and the success of that product. For the purposes of this project, we define web media as both social media and more journalistic forms of media.

The motivation for the project came from the desire to monitor media activity before, during and after new product announcements, to be able to determine any trend that could help in predicting the probable success of the product after release. If this trend data proves to be useful, it could indeed be a valuable asset.

This project hopes to be able to accomplish this by monitoring the frequency with which web media references a particular product name for comparative purposes. In this project, I also hope to be able to analyze references to the products to determine where the sentiment polarity of the popularity.

For the scope of this project, I focus on only modern electronic devices and similar technological products. The contributions this paper and project set out to make are:

1. I build a system as a proof-of-concept with the purposes of gathering and visualizing data from web media for analytic and pedagogical purposes.
2. The paper sets out to study the prospect of gathering web data to determine popularity of commercial items and the usefulness of this
3. The project establishes a basis for further studies of the correlation between product popularity and commercial success.

The following section in the paper discusses related work that I researched during this writing. Section III defines the problem this project sets out to solve, Section IV describes the methods taken to solve the problems and Section V and VI show the results obtained and the conclusions respectively.

II. Literature Review Of Previous Work

Multitudes of projects in the past have attempted to exploit web media information to detect trends. Most of

them have occurred in recent times, mostly due to improved access to online information and data. After the advent of social media sites like Twitter[1], there has been a rise in the number of projects and research tackling trend detection and prediction. Particularly, some of these projects include:

- Applications to detect trends in a network of blogs using data mining and NLP algorithms [1]
- Experiments to collate micro-blog posts on specific topics and summarize them using algorithms [2]
- Applications to detect the first post to mention a trending topic on Twitter, and to detect trends on Twitter [3] [4]

In addition to these research projects, many commercial products exist with the purpose of tracking trends on Twitter and other social media. Products like RankMeme[2] and TweetTronic[3] offer its users the chance to see what electronic products are the most popular on Twitter and how they compare with other products.

Most of these projects use Twitter as a source of data because it provides the easiest access to what it calls a "fire hose" of user posts. This rich source of data is useful for detecting trends and performing sentiment analysis.

Similar to this paper, there have also been research projects to mine the web and collate data to maintain a body of knowledge, ontology [4], or detect general trends [5]. In [5], the authors discuss systems that one could use to detect trends in textual data, visualizing and evaluating this data. It is possible to extrapolate this to include textual data from the web, which is similar to the work of this paper.

Characteristics that are notably missing from some or all of these projects stated above are:

- Focused analysis on specific keywords: most of the work referenced focuses on detecting trends in general. This is a very broad approach and does not show how to monitor a particular trend, even when it is no longer trending.
- Broader data scope: most of the referenced work focuses on just one facet of web media mostly social media.
- Quantitative analysis of data: some of the above work attempt to show detected trends in context without providing a means to compare quantitatively two or more trends.

Note that these listed characteristics are not all-inclusive as the mentioned projects have some, but not all of the characteristics.

III. Problem Definition

My objectives for this project are mainly to form a basis to establish a relationship between how much people on the web mention a product (its popularity) and how well it does in sales. To achieve this, we would model the gathered data in various ways to optimize the analysis process. Such a study will provide insights as to the usefulness of collating discrete sets of user opinions about varying products to predict a trend.

To define the problem scope, we must define what we mean by popularity of a product on the web for the purposes of this project. The products in this case are consumer products like computer, mobile devices, telecommunications hardware, and home-entertainment hardware. Popularity, on the web, of a product is the amount of times web media references that product; be it social media or journalistic media. Hence, for the purposes of this project, we can define popularity thusly.

Definition of popularity: The level of pervasiveness and mind-share an object has in a given society, regardless of if this object is well liked or not.

To achieve popularity and sales-success prediction, I attempt to gather data that could indicate the current popularity of a particular product, study forecasting of this popularity, and suggest ways in which current sales perhaps correlates with current popularity.

It should be noted that this project isn't an attempt to detect the most popular products on web media, but an attempt to gather data to view the popularity in general of a specified product or set of products.

Although we attempt to classify the kind of popularity in this project, we still generalize popularity as both positive and negative popularity regarding public sentiments.

This project utilizes the following data sources: online news media publications, blogs and social media. I mine a subset of websites for specified keywords and tally all references to that keyword.

IV. Methodologies

For this project, I focused on a small subset of sites to extract data. I explain the approaches to extracting the data in further sub-sections below. For social-media data, I focused on Twitter. With over 500 million registered users and over 200 million active users a month[4], Twitter offers a rich source of public posts and opinion. To extract this data, I utilized Twitter's Search API v1.0. This API allows us to search for all mentions of a term on the website.

For other, more general, web media, I built an automated

[1] https://twitter.com
[2] http://rankmeme.com
[3] http://tweettronic.com
[4] http://en.wikipedia.org/wiki/Twitter#Growth

web crawler. This crawler parses the documents on several known websites and extracts information on the specified product. It also checks the relevance of each document before including it as a data point.

A. *Web data extraction*

To get references to keywords of interest around the web, I focused on a combination of social media and news/journalistic media as sources of both public and professional opinions. For this project, I minimized the scope of social media to cover just Twitter.

Information Retrieval scheme

For the proof-of-concept system, I based the information-retrieval schema on keyword queries in which a keyword represents each product of interest that we intend to track for popularity. Each new keyword query is stored in our database for continuous and periodical data mining. Once a new keyword enters the database, it starts the data mining process, which a *cron* job periodically maintains for seven (7) days.

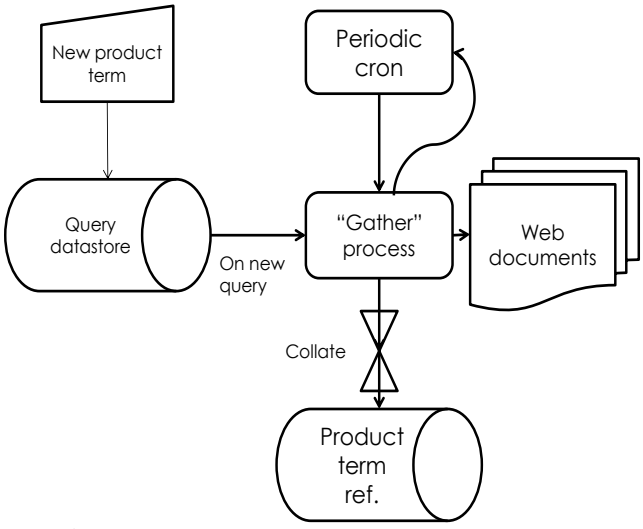

Fig. 1. Information Retrieval scheme representation

The system then collates each keyword reference and stores it in the database. The cron job periodically restarts this process and it will only stop after seven (7) days elapses or it has gathered enough data.

Twitter post extraction

Twitter offers an API for retrieving tweets from their service via a search mechanism[5]. This API returns relevant tweets that match a specified query. It returns a maximum of 1,500 tweets or maximum of one-week-old tweets – whichever comes first – for any queries. For this project, I used the first version (v1) of this API to retrieve public posts that contained tracked keywords. As of the time of this writing, Twitter was deprecating this version by May 7, 2013[6].

Using this API, we can easily obtain access to a huge amount of social media posts and filter them based on keywords. To utilize this, I created a scheduled *cron* job to periodically fetch tweets every hour and store for further transformation. The system was set to retrieve all posts in only the English language, retrieve the posts in their raw form and strip all extraneous data from the post text.

Web document crawling

To obtain data from other websites, I created a focused [6] [7] and topic-biased crawler. This crawler utilizes a URL crawl frontier [7], which it extracts from the main page of a *seed-URL* [7]. From each seed-URL, the crawler extracts links to articles of interests and populates the frontier with these links. Based on a selection policy I established (see section in this paper for *Determining Relevance of posts/article*), the crawler extracts article text from each of the article URLs in the frontier.

The seed-URLs are links to blogs or news websites I maintain in a database. Using these seed-URLs, the web-crawler performs a breadth-first [8] search on the web pages for a keyword. Because I start every new crawl from the home page of each seed-URL in this breadth first approach, when the crawler encounters an article or post that has a publish date which is before the time of the last crawl on that seed-URL, it treats that article as a boundary and stops the crawling process for that tree-level. This process of URL canonicalization prevents the crawler from mining the same resource twice.

To perform a true breadth-first search based on the site-map of a website, I optimize the crawler to handle paginated websites. In summary, a simplified algorithm, in pseudocode, for the crawler is similar to this:

```
1.  Accept seed-url, u
2.  Initialize URL-frontier
3.  for each article link, l, on home page of seed-URL
4.      check publish date of l
5.      if l.date > last crawl date
6.          break loop
7.      end if
8.      check l against initial selection policy
9.      add l to frontier
10. end for
11. if boundary not met
12.     set u = u.nextpage
```

[5] https://dev.twitter.com/docs/api/1/get/search

[6] https://dev.twitter.com/blog/api-v1-retirement-final-dates

13. recur this function
14. **end if**
15. **for** each *l* in frontier
16. extract *l.article*, set *a* = *l.article*
17. check *a* against selection policy
18. **if** *a* meets policy
19. store *l*, *l.date*
20. **end if**
21. **end for**

The crawler was built with the structure of each seed-URL in mind. I studied the HTML structure of the web pages and built the web crawler to be able to extract only links to articles from each webpage.

The web-crawler crawls in a serial pattern for each seed-URL and each keyword. A more optimized crawler would use a parallel pattern and a better approach than breadth-first to mine data. However, due to time constraints for this project and lack of infrastructural resources, that subject is beyond the scope of this paper.

Web crawling policies

The created web-crawler met up to certain established standards and policies.

1. Before accessing the contents of each link in a crawl, check the link against the *robots.txt* file of the webpage so as not to access restricted content
2. I gathered the list of seed-URLs after carefully reviewing the *Terms of Use* of the websites. If any of them explicitly stated anything against data mining, I do not maintain it as a seed-URL.
3. The web crawler delays each request for a new page so as not to overload the server hosting that web page.
4. In case of any errors when accessing a web page, based on the HTTP error code, the web crawler either backs-off and tries later after some time or goes on to the next link in the frontier.

B. *Determining Relevance of posts/article*

The posts obtained from Twitter did not need to undergo any form of relevance evaluation because Twitter already attempts to fetch the most relevant results according to your search term when using the Search API[7]. In spite of this, for other posts obtained from blogs or news websites using the web crawler, I had to determine if the post was relevant enough to the ideas of the keyword the system is monitoring.

To determine relevance, I restricted my approach to naïve methods. The first step is to gather the frequency at which a web document references the keyword. The web crawler/extractor performs a diacritic and case-insensitive search on the main text in the web document to obtain all keyword references. Because the websites used for this project are known beforehand to provide balanced, unbiased information, there was no need to use the term frequency and inverse document frequency using the term frequency–inverse document frequency (TF-IDF) algorithm. A crawler that attempts to determine relevance from unknown sources could however be required to use TF-IDF [6]. For instance, a collection of documents on law is likely to have the term "*legal*" in every document. Hence, a search term of "legal" in this corpus is mostly irrelevant as its discovery is not an interesting [10] data point.

Nonetheless, for pedagogical purposes, my web crawler uses a TF-IDF weighting scheme to determine the relevance of each post to a keyword. Each keyword reference in a document is calculated and the system normalizes the raw keyword frequency according to the length of the document. If a single sentence document references a product name even once, that document is likely relevant to the product term.

To accomplish this normalization, I used the normalized term frequency equation defined as:

$$tf_{norm_{i,d}} = \frac{n_{i,d}}{\sum_k n_{k,d}} * 10^6$$

This shows that the normalized term frequency for the term i in document d is equal to the number of times the term i appears in d divided by the total number of words, k, in document d multiplied by a scaling factor of 10^6.

The TF-IDF calculation then relies on the normalized term. To obtain the TD-IDF, and hence, the term relevance, I calculate the IDF using the following formula:

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

We calculate the IDF of the key-term, represented by t, in the corpus, represented by D, as the base-10 logarithm of the number of documents in the corpus divided by the number of times t appears in the any of the documents in the corpus. Because we cannot realistically expect to obtain all documents in a given website, I take the corpus to be only the articles or posts in the first page of the website. Hence, in my experiments, $|D| = 10$. In any situation that the term does not appear in any other document in the corpus, to prevent a divide-by-zero situation, the algorithm defaults the denominator to 1.5. With this calculation, the IDF of a rare term is likely to be high and the IDF of a frequent term would be low.

After the IDF is calculated, we obtain the full TF-IDF using the equation:

$$tfidf(t, d, D) = tf_{norm_{i,d}} \times idf(t, D)$$

[7] https://dev.twitter.com/docs/using-search

which shows that the full TF-IDF is calculated by the normalized term-frequency of the keyword multiplied by the inverse-document frequency of the keyword. This gives us a value that we check against an arbitrary threshold to determine if the keyword is relevant to that web document.

After the web crawler gathers posts containing our product term from the website, it considers only the posts with an appropriate normalized term frequency for use as data-points. As earlier stated, I perform the TF-IDF calculation solely for pedagogical purposes. A focused web-crawler that knows the sites it crawls and that knows the search terms beforehand does not require any TF-IDF calculations as the sites can be guaranteed to be non-biased and the keyword guaranteed to be non-frequent.

C. Data manipulation

After the system performs a process of web mining, the keyword database stores each keyword against its reference date and the URL or text that references it. This storage model is chosen so that the text is maintained for sentiment analysis (see section on *Sentiment Analysis*) and to make it possible to perform a process of *extraction*, *transformation* and *loading* (ETL) when we need the data for analysis.

Table 1 An example illustration of the data schema used for keyword reference storage

keyword	date_referenced	url/text

Table 2 An example illustration of the data schema after the extraction and transformation process

from_datetime	no_of_references	to_datetime

When we need the data for analysis, the ETL process transforms it to obtain period groups with the number of references against that period (see Table 2).

D. Data Analysis

As shown above, we currently model data obtained from our data sources for storage by quantitatively collating it. In essence, the system counts and groups posts from certain time-periods by product term. This will help us establish and store the number of references made to each particular product term at each particular period. The context of each post itself is not relevant to us and, hence, the system could discard it after it performs sentiment analysis.

After the system aggregates the popularity count for each term, any action to recall this data for analysis will start an *extraction*, *transformation* and *load* process.

The aim of this transformation is to be able to generate a popularity chart similar to the sample shown below.

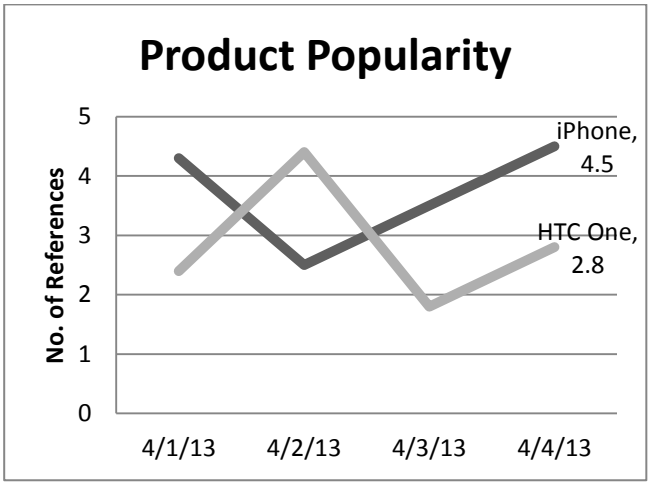

Fig. 2. Sample chart showing popularity of products over time

This visual representation of the data allows us to apply any number of analytical methods to it

E. Sentiment Analysis

For this project, I intended sentiment analysis to be an addendum to the main goal of the project. The goal of the sentiment analysis was to attempt to differentiate between the types of popularity that a product was receiving. Not all references to a product may be positive. For instance, it is possible for a product to receive negative backlash from thousands of unsatisfied customers. The PPP system will normally detect this trend as high popularity, but could still affect the sales of the product negatively.

Therefore, my objectives were to research the possibility to differentiating between an item's popularity and its infamy from a stream of text. Because of time constraints, I limited my sentiment analysis to single sentence posts from Twitter.

First attempt at sentiment analysis

For my first attempt to achieve this, the system used a number of natural language processing techniques. The open-source Python Natural Language Toolkit (www.nltk.org) library provided the natural language functionality. These following processes outline the steps I took for my first attempt at sentiment analysis:

1. The Twitter search API could return text with special symbols. The scripts perform initial text normalization to remove all HTML character entities or convert them to their Unicode equivalents. "`<`" becomes "<", for instance.
2. The system continued preprocessing each sentence by tokenizing the sentence into their various parts. Each sentence was broken down into a list of its individual words.
3. The final stage of preprocessing was to tag the different parts of speech (POS) in the list of words that made up a sentence. Hence, a list of words such as `['iPhone', 'with', 'a', 'weak', 'battery']` will become `['N', 'P', 'AR', 'ADJ', 'N']` where the POS tags stand for noun,

preposition, article adjective, adjective and noun again respectively.

4. Create and identify a dictionary of positive and negative words.
5. With this dictionary, we tag our list of words in a sentence with its POS and its sentiment if it matches any word in the dictionary. With this tagging process, the sentence 'iPhone with a weak battery' would become:

```
[([‘iPhone’], [‘N’]), ([‘with’], [‘P’]), ([‘a’], [‘AR’), ([‘weak’], [‘ADJ’, ‘negative’]), ([‘battery’], [‘N’])].
```

   This structure is a Python list of a tuple of two lists, which represent the original word in the sentence, its part of speech and an additional sentence. The sentiment is only added if the word is found in either the negative or positive dictionary
6. The final step is to get a sentiment score using a very naïve method of counting the number of negative and positive tags found in our structure.

As is obvious from the above method, this procedure is very naïve and negligent of some basic assumptions of sentimental sentences. One of the aspects neglected is negating or inverting expressions in a sentence. As an example, one could consider the following sentence as a negative sentence:

"My iPhone has a terrible battery"

The naïve system above will also consider it as negative, assuming it has the word "terrible" in its negative dictionary. However, one may not consider the following sentence a negative one:

"My iPhone does <u>not</u> have a terrible battery"

Unfortunately, the system will flag this as negative statement due to the presence of the negative word. Hence, the system could take into account negating and inverting expressions like "not" and change the polarity of a sentiment when a negating/inverting expression immediately precedes it.

It is also possible for emoticons [12] in sentences to immediately dictate what sentiment the sentence portrays.

Second attempt at sentiment analysis

As had been noted, the initial attempt at sentiment analysis gave poor results due to its naïveté and simplicity. I then attempted some reworked improvements to the existing procedure. In this new procedure, the system utilized the *Naïve Bayes classifier* [13]. The Naïve Bayes classifier is a feature classifier that works based on probability and the Naïve Bayes theorem. In this case, it uses the probability of features in a training data set to learn the probability of new data falling into a certain class.

Before the classifier works, however, there is a different set of preprocessing techniques applied.

1. Gather a list of positive and negative sentences to use as a test set. In contrast to gathering positive/negative single words in the first method, we gather whole positive/negative sentences
2. Remove all stop words from test set. In this case, stop words include all words with two or less characters.
3. Extract a list of all word features and their frequency from the test set. A sample features list could contain:

```
[(‘this’, 102), (‘iPhone’, 52), (‘hate’, 41), (‘battery’, 12), (‘excellent’, 28)...]
```

   with the first word in each tuple corresponding to a feature and the number in the tuple corresponding to how many times that feature appears in the entire corpus.
4. With this list of word features, apply the features to the test set of sentences using the NLTK library to get a list-like structure that has the features. For example, if the first sentence in the test set is "I love my iPhone", the new structure obtained could be:

```
[({‘this’: False}, {‘iPhone’: True}... {‘love’: True}, {‘excellent’: False}..., ‘positive’)...]
```

   where each tuple contains a Python dictionary of each feature, a Boolean value indicating if that feature appears in the sentence, and the final element in the tuple indicating the sentiment of the sentence from the test set. This final structure is the training set with which we can train the classifier.
5. Use the Naïve Bayes classifier from the NLTK library to get a trained classifier

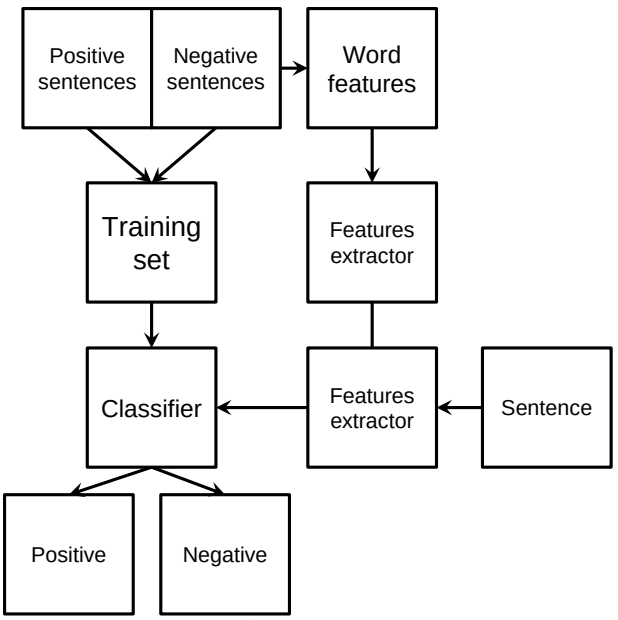

Fig. 3. Summary diagram of sentiment analysis

Drawbacks of sentiment analysis application

My approach to applying sentiment analysis to sentences from posts was not without its limitations. One of the biggest problems was the lack of enough data to use for the classification training. The time-consuming nature of manually collecting thousands of sentences and labeling them as positive or negative discouraged me from collecting enough data.

Due to this, the sentiment analysis system was highly inaccurate and unreliable, to the point that it was discarded mid-way into the project and not applied into the final results. It is possible to use data that had already been gathered. However, due to the nature of technology products, sentiments towards these products do not always translate as easily as sentiments towards other objects. People are more likely to complain or praise a particular feature of a product ("I hate the battery of the iPhone") than complain about or praise the product as a whole ("I love iPhone"). In spite of this, this work is still open to future studies any further enhancements to my sentiment analysis approach could yield results that are more helpful.

V. Experimental Results & Findings

In this section, I reiterate my objectives, evaluate my methodology and mention the findings throughout this project. Because the secondary goal was to learn and understand web-mining techniques enough to apply them to predictive modeling, I expanded my scope widely. However, I extended the scope too widely and attempted to cover a wider breadth on the subject of web-mining rather than depth on a particular web-mining topic. This approach is tedious and discouraged.

Regardless of this, my goal was to be able to study sales report on a particular product, study the correlation between the sales report and the online popularity of the product overtime, and see if we can apply the correlation to predicting the sales of future products based on its online popularity. I consider my work to form the basis of such research as the stepping-stone to detecting popularity online, not just from social media, but also from all other forms of internet media.

Due to the timeliness of the project, the tracked products did not have any sales report from their respective companies. Notwithstanding, the system was built and deployed on the Google App Engine[8] platform and was able to gather data over time until the end of the project.

Using the captured data, I was able to transform it into groups based on periods. This was helpful in allowing me be able to get a reference to a particular time and check how many references the Internet made to a particular product that my system was tracking. Using these time references, I could plot a graph showing the trend of popularity over the entire time my system collected data. At the time of this writing, the system's website displays sample graphs of popularity of tracked products.

Using this data, I made several discoveries:

1. Future projects of this kind should localize and regionalize the idea of popularity of a product. Popular products in one region may not be popular in other regions. If one takes the popularity of a product with varying popularity levels in different regions as whole sum, the data may show untrue patterns.
2. Many opinions on social media show a "herd-mentality" towards products. Many people will always repeat already popular opinions. This increases references to the product.
3. References to objects on the internet, especially on social media, follows the inherent popularity of that object. As an example, the iPhone is a vastly popular product. In fact, its popularity is not directly proportional to its sales. When many think of smart-phones, they think of the iPhone. Because of this, references to that product are multitudes of times higher than references to any other product in its class.
4. To correlate possible sales of a product and its web popularity, it is important to be able to differentiate between negative and positive sentiments towards it. As an example: although the Blackberry Z10 is a popular product on Twitter, many of the opinions expressed towards it are negative. The sales of the phone might reflect this negativity.

We can evaluate the performance of my web-mining techniques by studying the data. We will notice that because the data gathering process is periodic, we cannot account for any gaps in time between two periods. Because of this, the quality of the results is dependent on the frequency at which the system mines data, which is in turn dependent on the infrastructural resources available.

A. Limitations

I encountered various limitations that could affect the accuracy, completeness and/or usefulness of this project:

1) Twitter has eliminated its access to anonymous API calls with its introduction of its new API, v1.1. This has also limited the number of times the API can be used.
2) Twitter also limits the number of results that can be fetched at a time to 1500. This means the built system has to periodically poll Twitter to collect enough data for use
3) The Terms of Use of some sites restrict data-mining attempts on the site.

VI. Conclusions

As earlier stated, any future work could extend the results discussed in this project for further studies of the

[8] Google App Engine https://developers.google.com/appengine/

correlation between a product's popularity and its sales success. As this paper proposed, gathering data to determine product popularity on the web is possible with the right infrastructure and resources. Much work has gone into detecting popularity prior to this project, but most of them are either too focused in their data source or fail to quantify the level of popularity and just ranks the most popular products or trends.

APPENDIX

A showcase of the system website and its data results is temporarily available at the following address: www.app-hrd.appspot.com.

I have dedicated about 80-odd hours to the completion of this project, despite its large scope.

REFERENCES

[1] M. H. T. T. N. Glance, "Blogpulse: Automated Trend Discovery for Weblogs," in *WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, Pittsburgh, PA, 2004.

[2] M. H. J. K. B. Sharifi, "Experiments in Microblog Summarization," in *NAACL-HLT 2010*, Los Angeles, 2010.

[3] M. M. P. B. Castillo C., "Information credibility on Twitter," in *Proceedings of the 20th international conference on World wide web,*, ACM, 2011.

[4] M. Mathioudakis and N. Koudas, "Twittermonitor: trend detection over the twitter stream," in *Proceedings of the 2010 international conference on Management of data*, ACM, 2010.

[5] M. Ashburner, C. A. Ball, B. A. Blake, D. Botstein, H. J. Butler, M. Cherry, A. P. Davis and G. Sherlock, "Gene Ontology: tool for the unification of biology.," *Nature genetics*, vol. 25, no. 1, pp. 25-29, 2000.

[6] A. Kontostathis, L. M. Galitsky, W. M. Pottenger, S. Roy and D. J. Phelps, "A survey of emerging trend detection in textual data mining," in *Survey of Text Mining*, New York, Springer, 2004, pp. 185-224.

[7] S. Chakrabarti, "Focused Web Crawling".*Encyclopedia of Database Systems [Online]*.

[8] M. Ester, G. Matthias and H.-P. Kriegel, "Focused Web crawling: A generic framework for specifying the iser interest and for adaptive crawling strategies," in *Proceedings of 27th International Conference on Very Large Data Bases*, Roma, Italy, 2001.

[9] J. Cho, H. Garcia-Molina and L. Page, "Efficient crawling through URL ordering," *Computer Networks and ISDN Systems*, vol. 1, no. 30, pp. 161-172, 1998.

[10] D. Hiemstra, "A probabilistic justification for using tf × idf term weighting in information retrieval," *International Journal on Digital Libraries*, vol. 3, pp. 131-139, 2000.

[11] A. Silberschatz and A. Tuzhilin, "What makes patterns interesting in knowledge discovery systems," *Knowledge and Data Engineering. IEEE Transactions*, vol. 8, no. 6, pp. 970-974, 1996.

[12] J. B. Walther and K. P. D'Addario, "The impacts of emoticons on message interpretation in computer-mediated communication.," *Social Science Compter Review*, vol. 19, no. 3, pp. 324-347, 2001.

[13] B. Pang, L. Lee and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques.," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, Pennsylvania, 2002.