SENG402 Final Report - Social Media in Software Development

Johann Reiher 17080963
University of Canterbury

Matthias Galster
University of Canterbury

***Abstract*—Using Drupal as a case study, we investigate the use of sentiment analysis on bulk sets of tweets in order to extract useful emotional trends relating to the development/release cycle. We offer extensive filtering of spam and other irrelevant data, automated categorization, and data visualization of various sentiment and categorical trends exhibited by our data set.**

I. Introduction

User and customer feedback is integral to the conception, design, development, and maintenance of any software projects and products [1]. Many of the prevalent methods of gathering feedback (e.g. customer surveys) are not ideal due to a number of factors including low response rate, a lack of real-time feedback, and the need to use incentives to attract users.

A second major issue, particularly in open source software development, is that of globally distributed development teams and users [2]. This, again, is essentially a feedback problem; developers require constant feedback from the rest of their team in order to maintain an understanding of product changes/enhancements/bugs etc. which is heavily affected by being geographically separated. The global spread of users essentially rules out a face-to-face approach to surveying users and necessitates the use of an online method.

II. Background

The use of social media particularly micro-blogging e.g. Twitter in software development as a tool with which to communicate and solicit user-feedback is becoming increasingly widespread. This trend is particularly prominent in the open-source community as it allows for simple and transparent communication within an often geographically scattered team of contributors and users. Given its prevalence simplicity and the ability to share information in almost real-time, the use of Twitter will be the primary focus of this project. Developers are relying more and more on micro-blogging for fast, real-time feedback from globally distributed sources without the need for extra incentives to encourage user feedback. Twitter, in particular, is used heavily by software developers and the general project team to curate technical information [2]. There is evidently a huge amount of valuable information now residing in these micro-blogging communities and although it is already highly useful in terms of immediate feedback and action, information on the overall trends and sentiments of both developers and users would add another major advantage. Research and development of comprehensive sentiment analysis tools has already begun and is outlined in [3], [4], [5]. The target audience consists of software developers, organizations, project and product managers, and marketing teams.

III. Solution Approach

The solution approach will consist of three basic steps: 1) Extraction of data from Twitter based on product-related criteria (e.g., product name): Twitter has a comprehensive REST API which allows for the programmatic reading and writing of Twitter data. This will be used to search for Tweets concerning a particular product or project, with Drupal likely to be used as a case study. A number of wrappers exist for this API, including TwitteR (an R package), and Tweepy (a Python wrapper). For the purpose of removing irrelevant (noise) data, it is necessary to filter the overarching set of tweets. There are a number of options for the method/style of filtering. The simplest option is a basic keyword filter as outlined in [6]. This has the advantage of simplicity and speed of implementation while also being unlikely to give false positives, however it is also likely to result in some relevant tweets being missed if they are fuzzy in their declaration of their topic. This step may also attempt to separate developer accounts from user accounts in order to allow separate analysis of these two distinct groups. Extracting large quantities of Twitter data can be very time-consuming. In order to account for this and also maintain a baseline set of test data, the Tweets gathered via the REST API must be perpetuated. This will likely take the form of a MySQL database. 2) Analysis of Twitter data using comprehensive sentiment analysis: Sentiment analysis will be used in order to programmatically extract emotional data from tweets. A number of sentiment analysis libraries exist and require comparison. Each individual tweet will be categorised as either positive or negative based on the polarity value found through the sentiment analysis. In order to test the validity of the output of the chosen sentiment analysis method, a set of test data with known classification is required. This can be created manually, however this is time consuming as it is necessary to read and label each individual tweet. A method that has been suggested is to search for emoticons and take these as a source of truth i.e. a happy face indicates positive polarity. 3) Presentation of findings in a user-friendly way: Using the classified tweets from the previous step we can now observe trends over time. These trends must be linked

to important features of the development cycle e.g. bugs, features, releases. A possible method is to look for words which occur in a large number of tweets around the same date and compare these to the available information on the development at that point e.g. change-logs. Volume must be visualised, along with overall polarity as areas of high volume are often more indicative than those of high polarity. The project is split into seven work packages, each with a number of tasks associated with it.

A. Work package 1

Retrieve data from Twitter; data must be related to a particular product; this may require a search based on hash-tags and Twitter handles; search results would need to be retrieved and stored in a structured format without losing meta-information (e.g., time, Twitter handle (ID), etc.) and text.

1) Task 1: Research existing methods/libraries/APIs and evaluate their suitability for use in the context of this project.

2) Task 2: Retrieve arbitrary data from Twitter programmatically.

3) Task 3: Design and implement a database to store the retrieved tweets, including all relevant meta-data.

B. Work package 2

Data filtering; noisy data needs to be reduced. Filtering could be done on account (is the particular account relevant to the analysis? Is the account held by a contributor or a user?), on the date posted, or on the message content (is the content actually relevant or did the message just happen to contain search terms?).

1) Task 1: Separate developers from users.

2) Task 2: Filter garbage tweets (spam).

C. Work package 3

Sentiment analysis; extracting emotional information from data. This requires extensive research and review of existing libraries/APIs. Basic analysis and visualization should be used.

1) Task 1: Find and compare existing sentiment analysis libraries.

2) Task 2: Perform basic sentiment analysis on the data set from Work Package 1, adding the output to the database.

3) Task 3: Identify/find/create control set and compare to the output of sentiment analysis from Task 2.

4) Task 4: Graph the polarity of both users and developers over time.

D. Work package 4

Link the exhibited trends from Work Package 3 to development events and categorize these events. Events may include new releases, added/removed features, bugs etc. These can be further categorized (e.g. product, process, feature) and may include dynamic categories.

1) Task 1: Create a development and release time-line for the period covered by the main data set.

2) Task 2: Look for keywords in times of high volume and compare these to documentation to find cause.

3) Task 3: Link sentiment trends to the development/release time-line.

E. Work package 5

The results will be presented to the target audience (developers, analysts, etc.) in a user-friendly way.

1) Task 1: Review and compare visualization libraries/tools.

2) Task 2: Implement visualization of relevant data.

F. Work package 6

Test the newly created tool(s) using different products in order to evaluate effectiveness as well as the general approach.

1) Task 1: Find other products with an appropriately large amount of Twitter usage.

2) Task 2: Run analysis and evaluate outputs.

G. Work package 7

Final report and presentation.

IV. Work Performed

A. Work Package 1 - Retrieve data from Twitter

1) Task 1: "Research" : A number of libraries/packages for interfacing with the Twitter REST API were investigated. TwitteR - the R package used in "Microblogging in Open Source Software Development" - was the first solution investigated[2]. R is a open-source statistical graphing and data-mining language and environment commonly used in complex mathematical situations. Due to a lack of experience with R, its environment, and its unusual syntax which may have hindered project progress, alternatives were investigated. Python was chosen as the preferred language for working with the Twitter API due to a number of factors:

- The existence of a number of strong candidate libraries.
- Familiarity with the language.
- Ease of text-manipulation.
- Speed of prototyping and development.
- Limited size of the end-product.

Python libraries investigated include tweepy, python-twitter, twython, and TweetPony. All of these libraries offered essentially the same functionality and ease of use for the scope of this project. Tweepy was eventually chosen as it is actively maintained, and works with Python 3.

2) Task 2: "Retrieve": In order to retrieve data from Twitter, it was necessary to create a developer account and a Twitter app. This allows for the generation of the keys required in the authentication process. Initially, tweets were retrieved via the standard search method, however this only allows for the retrieval of one page of tweets at a time (up to 100 tweets). In order to retrieve the maximum number of tweets (~15,000) it was necessary to use a Cursor object as a pagination helper. The cursor takes the API's search function as its first parameter, followed by the search query, and the number of results to return per page. The cursor then iterates through the ~1,500 available pages and retrieves the tweets from each.

3) Task 3: "Store" : A database was required to store Tweets after retrieval via Tweepy. This was especially necessary due to the time overhead of retrieving tweets (~10 minutes). Tweets are returned by Twitter's API as JSON objects. Tweepy provides a wrapper for these responses and automatically makes use of this when retrieving tweets. The result is a Python object with JSON name:value pairs mapped to class attributes. The object also contains the original JSON for convenience. Since the project scope primarily surrounds the tweet text and is less effected by properties of entities like users, a single table will suffice. Initially the intent was to use a mySQL database, however this was determined to be over kill for the intended use. MongoDB and SQLite were both investigated, including a first prototype in MongoDB before eventually deciding on SQLite for its simplicity, portability, and the added benefit of being built in to Python. The initial prototype simply stored the entire JSON of Tweets in a single column table. The next iteration was the SQLite version currently in use. This comprised a single table which stored user_id, follower_count, screen_name, tweet_id, retweeted, favourite_count, tweet_text, created_at. This allows for the select statements required by Work Package 2.

B. Work Package 2 - Filtering

A step was added to this work package as it became clear that retweets would need to be removed. Since all retweets begin with "RT ", they were able to be removed by checking the first three characters of each tweet and removing the tweets matching this pattern.

1) Task 1: "Separate developers from users" : In order to complete this task a list of official Drupal accounts and the accounts of core contributors had to be found. This was based on a combination of the core accounts mentioned in "Microblogging in Open Source Software Development", along with a list of major contributors found at https://www.drupal.org/node/3060/committers[6]. The separation itself is done via database query.

2) Task 2: "Filter spam" : In order to allow the spam filter to adapt to different topics/products in the future, a machine learning approach was desirable. This is done via a Bayesian classification Python library called Bayesian [7]. Naive Bayesian classification has been shown to be highly effective in the recognition and categorization of bodies of text - particularly in the context of Twitter - and was therefore deemed an appropriate algorithm for this task [8], [4]. The program includes a utility function which prompts the user for manual classification of a designated number of tweets and writes these answers to file as its training set. Alternatively, It can then classify unknown examples based on the information collected from these training examples. The higher the number of training examples, the more accurate the spam detection (particularly with a relatively even distribution of positive/negative examples). Testing was done with 20 training examples and showed a good degree of success, however in order to increase the robustness of the spam filter a training set of size 100 will be used in future.

C. Work Package 3 - Sentiment Analysis

1) Task 1: "Research": A number of sentiment analysis libraries were found and compared including SentiStrength, Pattern, Textblob, and NLTK [9], [10], [11], [12]. NLTK, although very powerful, was not ideally suited to the task of classifying tweets as the functionality it offers is more base-level and would have required significantly more work than the other options to attain basic sentiment analysis readings. SentiStrength required awkward in-Python Java handling and careful text manipulation and was decided against on those grounds. Pattern offers out-of-the-box sentiment analysis giving access to both polarity, and mood/modality values. TextBlob is a high level text analysis library which builds on the work seen in both Pattern and NLTK. It provides single-line sentiment analysis, giving polarity and subjectivity values, however it does not include the mood/modality functions seen in Pattern.

2) Task 2: "Sentiment Analysis": Initial sentiment analysis was performed using TextBlob. TextBlob was chosen due to the reasons mentioned in Task 1, along with ease of implementation, and Python 3 compatibility. This provided good results, however it became apparent that more sentiment data, specifically mood and modality, would be required to perform proper analysis. To this end, the decision was made to use Pattern instead. This required back-porting from Python 3 to Python 2. A number of problems with encoding arose during this process, with some libraries requiring utf-8 while others required Unicode. These issues were fixed by specifying the particular encoding required at a number of points in the program. The current implementation gives us values for polarity, subjectivity, mood, and modality.

3) Task 3: "Graphing": Figures 1 to 4 illustrate the summed polarity per day for various subsets of users. There appears to be some degree of correlation in the trends of the different

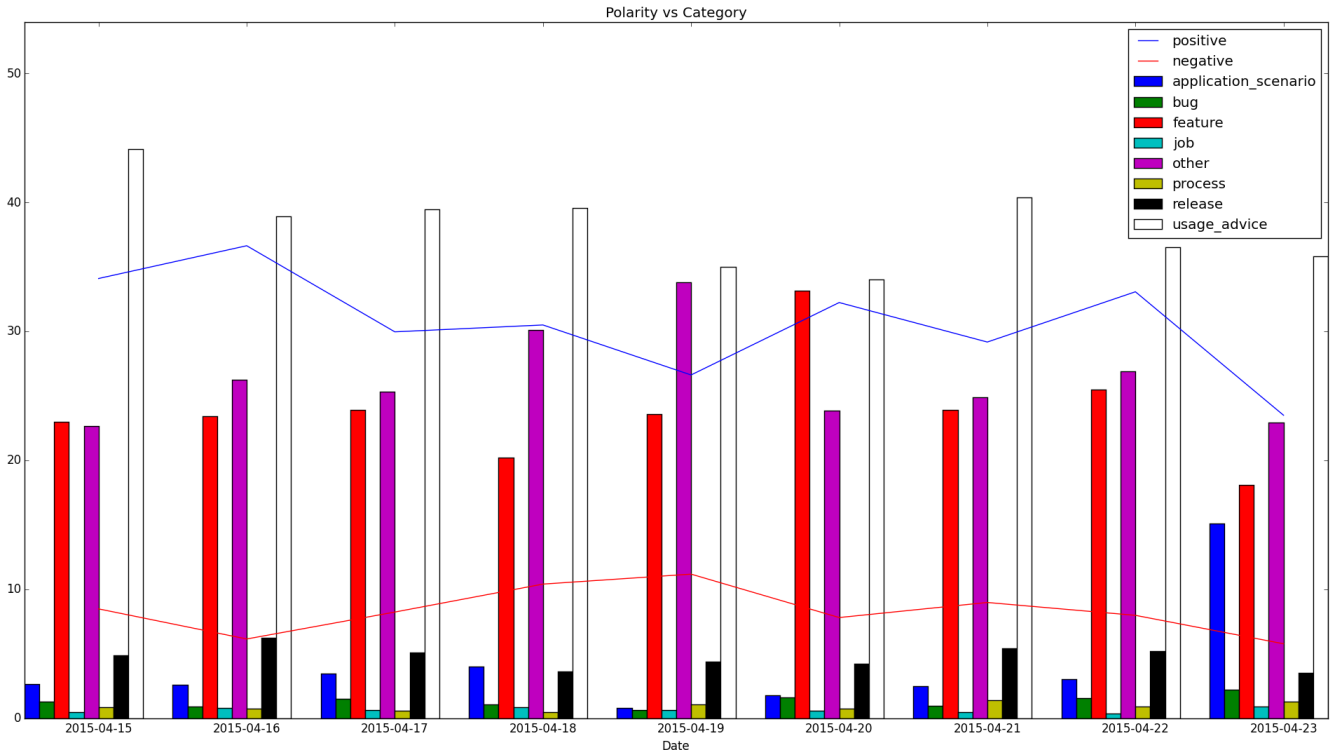

Fig. 1. Polarity and Category Percentages per Day - Iteration 1

groups, however given the small range of time within which all of the tweets occur, this is an unreliable assertion.

D. Work Package 4 - Categorization

This work package has not proceeded as anticipated in the project proposal, however progress has been made. A machine learning approach was decided on for categorization as manual classification is unfeasible for such a large data set. A Naive Bayesian classifier like that used in Work Package 2 was employed. In order to train the classifier, it was necessary to manually classify tweets into categories. The selected categories are Feature, Release, Process, Bug, Application Scenario, Usage Advice, Job, and Other. The same set of tweets was manually classified independently by each of us and then merged to form a consensus. The classifier has a class for each of the categories, as opposed to the binary version used in the spam filter.

Feature	Announcing Features for #Drupal8 - Informative reading from @phase2 http://t.co/FHZMXV0KWe
Release	Excellent summary of the expected features you'll see in #Drupal 8 http://t.co/Dbww2mOWR6 Which are you most looking forward to?
Process	Hey! What do you think of Dummy comment entity necessary for building comment form is expensive to build? I'm tryię http://t.co/PX8DRMMWO4
Bug	Hey! What do you think of Token length is not checked and causes exceptions if it is longer than 50 chars? I'm tryâę http://t.co/YQ2EKVd3YM
Application Scenario	Drupal for Non-profits. See the resource guide: #drupal http://t.co/iS2242oiND
Usage Advice	Creating Joomla Templates, Drupal Themes, Wordpress Themes, DNN Skins, and Blogger Templates all in minutes. http://t.co/jSWmDBlyza
Job	#Empleo #Job Build a Website in either DRUPAL or WORDPRESS (WP preferred) by jasonsbradshaw http://t.co/kOKxgTju97
Other	Turn with respect to the nonpareil pervious cms - drupal: IFOlt http://t.co/YvfCAGTlb0

TABLE I
EXAMPLES OF TWEETS CLASSIFIED INTO EACH CATEGORY

V. Interpreting Sentiment Data

A. Polarity

Polarity values are in the range [-1.0, 1.0] where -1.0 indicates a totally negative sentiment, and 1.0 indicates a totally positive sentiment.

B. Subjectivity

Subjectivity values are in the range [0.0, 1.0] where 0.0 is highly objective, and 1.0 is highly subjective.

C. Mood

Mood refers to the use of auxiliary verbs (e.g., could, would) and adverbs (e.g., definitely, maybe) to express uncertainty. Mood is either INDICATIVE, IMPERATIVE, CONDITIONAL, or SUBJUNCTIVE.

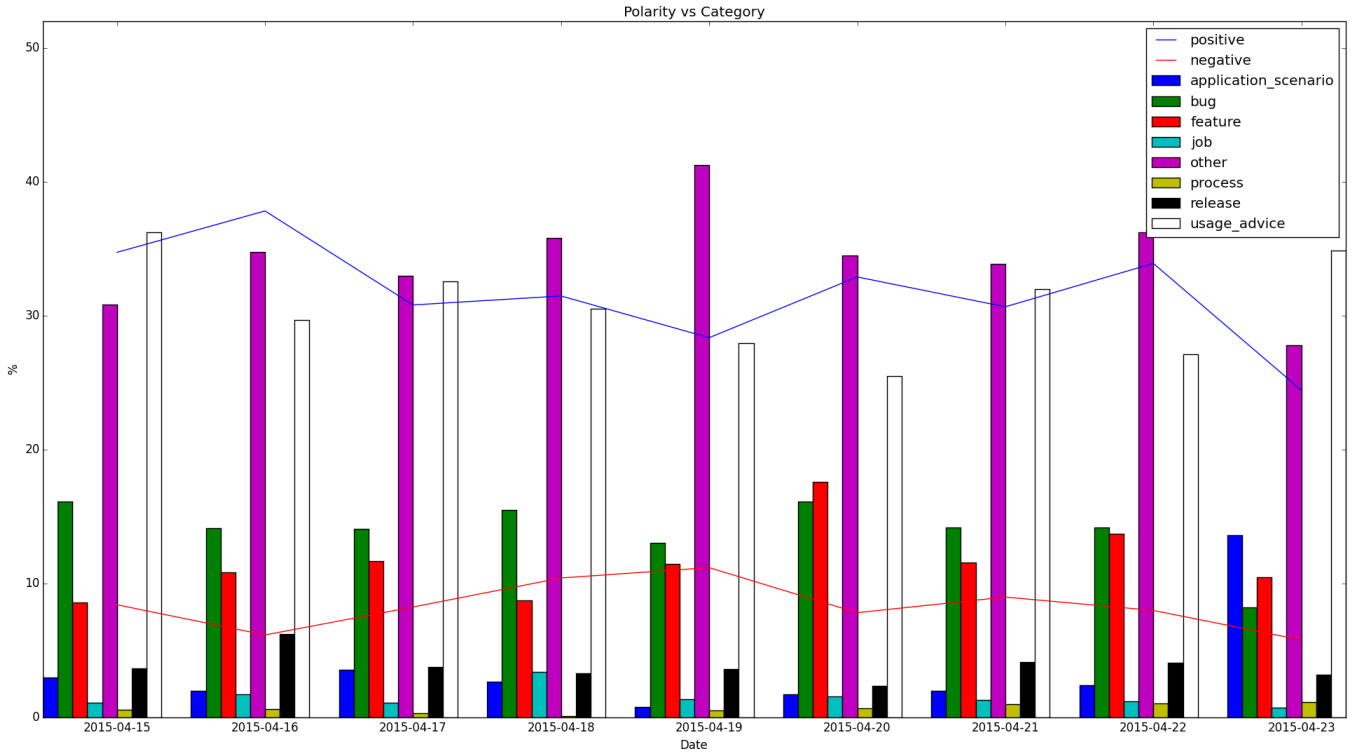

Fig. 2. Polarity and Category Percentages per Day - Iteration 2

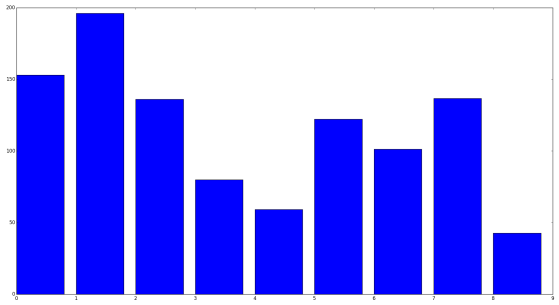

Fig. 3. Summed Polarity per day - All Tweets

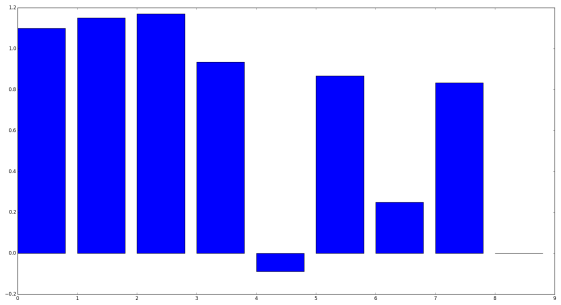

Fig. 4. Summed Polarity per day - Core Contributors

D. Modality

Modality values are in the range [-1.0, 1.0] and represent the level certainty expressed. Values above 0.5 indicate facts.

VI. Coverage of Sentiment and Mood

Of the 9812 tweets which remain after filtering

- 3904 have non-neutral polarity,
- 9767 have non-neutral modality,
- 9394 have non-neutral subjectivity.

The number of tweets with non-neutral polarity was found with the following SQL query:

```
SELECT count(*) FROM
    tweets_with_sentiment_and_mood WHERE (
    polarity != 0.0);
```

The number of tweets with non-neutral modality was found with the following SQL statement:

```
SELECT count(*) FROM
    tweets_with_sentiment_and_mood WHERE (
    modality != 0.0);
```

The number of tweets with non-neutral subjectivity was found with the following SQL statement:

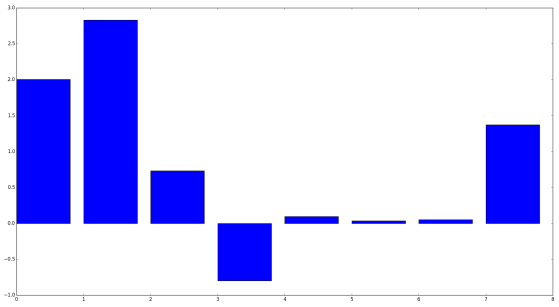

Fig. 5. Summed Polarity per day - Official Drupal Accounts

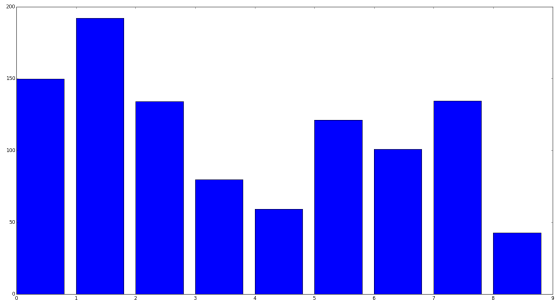

Fig. 6. Summed Polarity per day - Non-Affiliated Accounts

```
SELECT count(*) FROM
    tweets_with_sentiment_and_mood WHERE (
    modality != 0.0);
```

VII. Evaluation

The delivered solution comprises four core elements; pulling tweets, filtering spam and other unwanted content, performing sentiment analysis, and categorising tweets. The solution, to a large degree, was evaluated via a combination of extensive unit testing and data visualization. The visualization in particular assisted greatly in allowing us to pinpoint areas of interest e.g. high/low polarity, unevenly distributed categorization, changes between training iteration, and daily/hourly trends. Test driven development was employed on a number of occasions, however the bulk of testing was performed retroactively. Fleiss' Kappa and Cohen's Kappa were used to evaluate the level

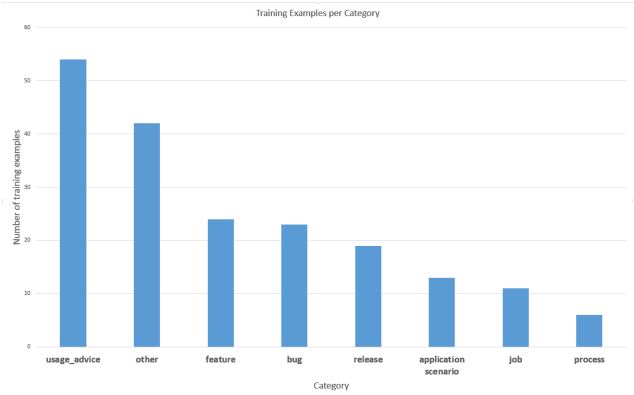

Fig. 7. Training examples per category

TABLE II
Mood Values

Mood	Form	Use	Example
INDICATIVE	none of the below	fact, belief	It rains.
IMPERATIVE	infinitive without *to*	command, warning	*Don't* rain!
CONDITIONAL	*would, could, should, may*, or *will, can* + *if*	conjecture	It *might* rain.
SUBJUNCTIVE	*wish, were,* or *it is* + infinitive	wish, opinion	I *hope* it rains.

[13]

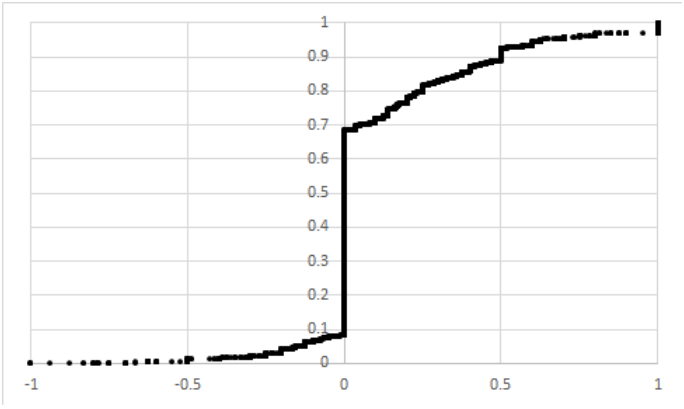

Fig. 8. Cumulative Distribution Function of Polarity Values

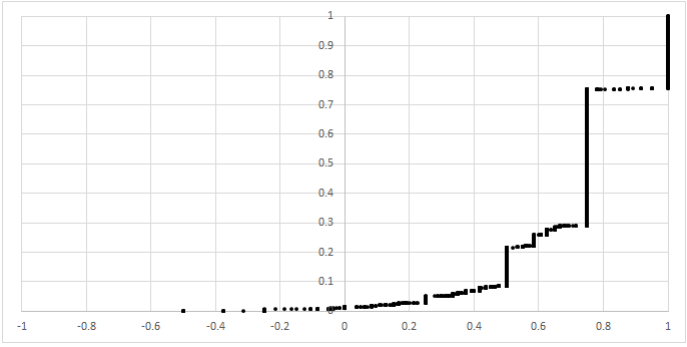

Fig. 9. Cumulative Distribution Function of Modality Values

of agreement between human and machine in regard to the classification of polarity, modality, and category.

A. Pulling tweets

Although the large majority of functionality involved in pulling tweets comes directly from the Tweepy library and as such is mostly beyond the evaluation scope of this project, there are few important points to note regarding issues that arose during the development process. The most configurable part of the tweet pulling process is the formation of the search query. As a result, this search query has a conceivably enormous impact on the other components of the solution. A search query which is too specific may provide clean data, however it can easily miss some relevant material and possibly diminish the overall number of tweets pulled. A search query that is not specific enough (too fuzzy) may result in a large amount of rubbish data being pulled, possibly to the degree that we exceed the maximum available number of returned tweets and therefore miss out on more relevant data. The large impact that the initial data set has on the rest of the project dictated that a great deal of care was taken composing the search query. The chosen query,

```
q="drupal8 OR drupal OR #drupal OR
#drupal8"
```

errs on the side of not being specific enough, pulling every tweet that mentions Drupal at all, however trials found this to be necessary in order to retrieve a satisfactorily large set of tweets for analysis. The breadth of the search query means that extra emphasis is needed on the filtering stage in order to remove junk data, noise, and spam. The use of time-dependent queries was trialed, however their effectiveness was dramatically decreased by the fact that regardless of the time range specified in the query there is a finite number of tweets available for extraction at any given time, e.g. a pull using the query

```
q="(drupal8 OR drupal OR #drupal OR
#drupal8) until:2015-10-03"
```

performed on 2015-10-13 returned zero tweets.

B. Filtering

1) Retweets: The first stage of filtering is to remove the retweets. The rationale behind this is that since tweets already have a "retweet_count" value which indicates the number of times that they have been retweeted, there is little information to be gained from the retweets themselves. The advantage of using the "retweeted" value rather than simply leaving retweets in the data set would be that this allows for more controlled handling of retweets, i.e. the amount of popularity surrounding a tweet can be taken into account when calculating sentiment values, rather than the overall sentiment being skewed by retweets still in the data set. There is, however, an issue with only using the "retweet_count" attribute because this only provides the number of times that the original tweet has been retweeted and does not take into account retweets of retweets. Retweet removal is currently done via a relatively naive method which involves looking for "RT " at the beginning of tweets. This is effective in removing manual retweets, however retweets which use Twitter's retweet button do not exhibit this characteristic and are therefore missed. [14]A better approach would be to combine the current method with the use of the "retweeted_status" attribute native to the Twitter API's JSON Tweet objects, the presence of which indicates that the tweet is a retweet. [15]

2) URLs: The second stage of filtering is removing tweets which are dominated by URLs to an extent that there is a negligible amount of useful information in them. This, once again, uses a text-parsing approach, searching for URLs of both "http://" and "https://" forms. The main negative effect of this approach is that if a tweet contains multiple URLs it will only pick up on the first one. The need for the additional search for "https://" URLs became apparent during unit testing when one of the example tweets was not being removed as expected.

3) Spam: The third, and most important stage of filtering is the Bayesian spam filter. A difficulty with this stage of filtering is that although we wish to filter "traditional" spam (e.g. tweets containing terms such as "viagra", "fast cash", and "this isn't junk"), we also wish to filter legitimate tweets which are contextually irrelevant, along with what seem to be (and will henceforth be referred to as) bot-generated tweets. This difficulty is exacerbated somewhat by the broad initial search query, as noted above. Bot-generated tweets differ from traditional spam in that they often lack the standard spam terms which a spam filter tends to look for and are instead composed of a number of completely legitimate terms which to a human reader are clearly nonsensical, however to the spam filter look like clean tweets.

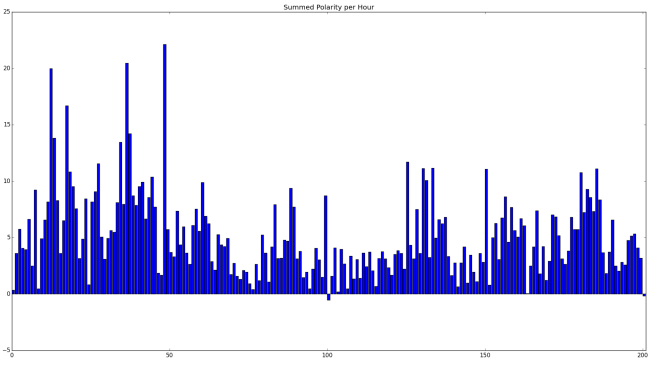

Fig. 10. Summed polarity per hour

a) Bot-generated tweet example: "Financial remuneration Drupal sodium hyposulfite in order to sapid acquisitions lead planning function: UNieSr" : The result of this is that bot-generated tweets are far more difficult for the spam filter to

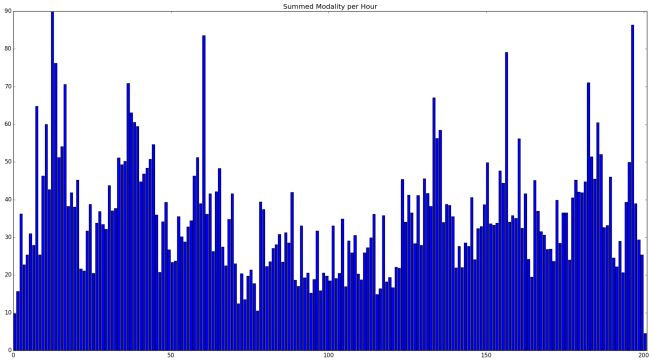

Fig. 11. Summed modality per hour

detect as they lack the standard terms and keywords which are fundamental to the success of the naive Bayesian classifier. In this respect, the Bayesian spam filter may not be ideal, however it does perform very well in terms of avoiding false positives which is extremely important in a scenario such as this where every tweet contains potentially useful information. Paul Graham's 2003 work showed that false positive rates of less than 0.06% were possible with a correctly implemented Bayesian spam filter. [16] This testing, however, was performed on emails rather than tweets. The big difference here is size, with the maximum length of a tweet being significantly smaller than many emails. The smaller size gives the spam filter less to go on when performing classification and therefore may result in a higher false negative rate, allowing more spam to pass through the filter. While investigating polarity extremes via relating trends exhibited in data visualization back to the tweets that caused these trends it was discovered that although no traditional spam was observed, a large number of legitimate yet unwanted tweets still existed in the data set. These tweets turned out to be the main cause of all of the largest polarity spikes as they tended to exhibit more extreme polarities than their more desirable counterparts. In theory Twitter's "Botmaker" filter dramatically lowers the incidence of bot-generated spam, however there is evidently still a substantial amount of spam circumventing this filter. [17]

C. Sentiment analysis

The functionality dealing with sentiment analysis essentially comprises no more than wrapper functions for the Pattern API, however investigation showed that the library itself can be carefully tweaked to give customized results. Although the Pattern library is known to provide good results across a broad range of domains, it does not take into account the jargon of any specific domain. As a result of this, there is definite room for improvement through the use of customized sentiment values for domain-specific terms e.g. giving "automation" a high positive polarity in a software engineering scenario. While investigation into this area was carried out (successfully adding the aforementioned example to the list of functional sentiment words) it has not been fully fleshed out in the delivered solution. Due to the lack of extensive domain-specific addition/reclassification of sentiment terms, the overall accuracy is ~75%. [13] Fleiss' Kappa showed a gradually increasing level of agreement between our two human classifiers and Pattern, with the kappa value steadily increasing from 0.45 (moderate agreement) to 0.61 (substantial agreement) between the first and final iteration. [18] Given that the two human classifiers ended up with a kappa value of 0.9 (almost perfect), this level of agreement for the three-way comparison is essentially what should be expected given the ~75% accuracy of Pattern's sentiment analysis.

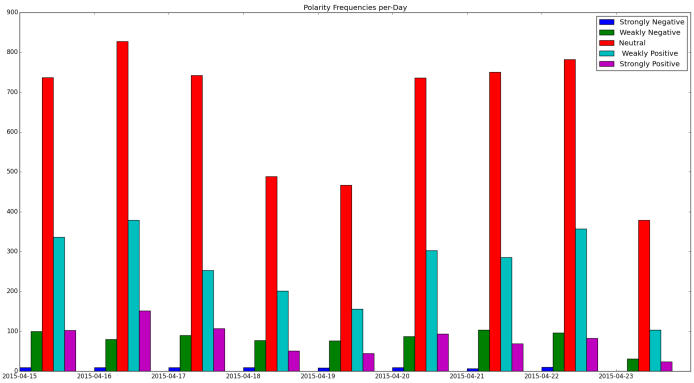

Fig. 12. Polarity frequencies per day

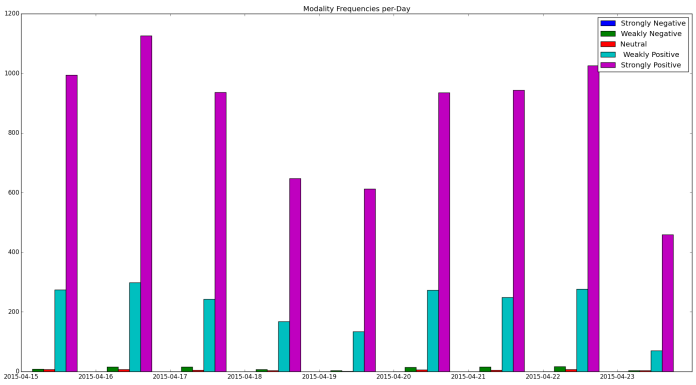

Fig. 13. Modality frequencies per day

D. Categorization

Categorization is performed using the same Bayesian classifier as the spam filter, however the number of categories is increased to eight to reflect the eight distinct groups of tweets which we are interested in analysing. The far larger number of categories means a significant increase in the amount of manually labeled training data required in order for the classifier to achieve similar results to that of the one used in the filtering stage. The classifier was initially trained using a set of 100 tweets. Evaluation of this first iteration gave a kappa value of 0.53 (moderate agreement), with the human classifiers

achieving a kappa of 0.85 (almost perfect). Although the three-way level of agreement was not particularly high, it was better than expected given such a small amount of training data. The second iteration saw the addition of another 100 tweets to the training set. Rather than increasing the kappa as expected, this increase in training data caused a slight decrease in kappa, from 0.53 to 0.50. One possible reason for this is that the training data is heavily skewed towards certain categories as a result of their prevalence in the two sets of tweets used to create the training set (illustrated in Figure 7). A slightly more likely reason is that since the data set used in the testing of the first iteration was then used to further train the classifier, a new set of test data had to be used for the testing of the second iteration to avoid testing on training data.

VIII. Discussion

A. Objectives

The primary objective of the project was to produce a tool which gives users the ability to analyse a given Twitter feed and draw correlation/causation information from the output in order to discover which changes most strongly effect both user and developer satisfaction. This objective was only partially fulfilled as a number of the later planned work packages were not able to be fully completed. Essentially the entirety of Work Package 4 could not be completed after initially being slowed down by filtering issues and unanticipated difficulties in gathering the appropriate release/commit information, and then hitting time constraints at the end of the project. Although this was an important work package for the project, it's absence does not detract much from the overall functionality of the tool due to the fact that this missing information could not easily be extracted programmatically and therefore would have been a manual process rather than part of the tool itself. This does however diminish our ability to completely evaluate the tool and general viability of the approach.

B. Learning

A number of important lessons were learned, or at times re-learned throughout the project. The biggest lesson was that no matter how well a project seems to be planned at the beginning, it is impossible to foresee the changes and challenges that will inevitably occur as the early stages steadily increase one's ability to properly assess the later stages. This ties in with doing as much research as possible early in the project to minimize surprises later on, and with resisting the urge to implement new functionality as soon as possible without having attained a sufficient grasp of the underlying concepts. Another big lesson was the need to thoroughly very investigate an API before choosing to use it not just in with regard to the obvious things - like checking whether it is current, well supported, known to be accurate and bug-free etc - but looking right down to the documentation of each class and function. Many (particularly Python) APIs look extremely well documented at face value but are severely lacking in the kind of depth that the native Java APIs have.

IX. Future work

It is clear from our analysis that there are a number of areas in which even relatively minor improvements could have a significant impact on the overall legitimacy of the project as a standalone solution.

A. Pulling tweets

A viable method of gathering a larger initial set of tweets is to perform multiple pulls at arbitrary time intervals (~once every 7 days) with the "since" query parameter used to ensure that the same tweets are not pulled multiple times. This, given enough time, will allow for a far larger number of tweets to be used in the analysis which could have a sizable effect on the ongoing success of the project.

B. Filtering

Given the amount of noisy junk data which is still passing through our spam filter, and the flow-on effects that this has on the rest of the project, it is clearly necessary to take steps to make improvements in this area. Re-trialing the current classifier with a significantly expanded training set would require the least input of the possible methods of improvement and still has the potential to yield good results. In the event that this does not satisfactorily improve spam filtering, different methods will need to be investigated. The first decision then will be whether to carry on with a different supervised learning approach or to branch out into unsupervised learning, or another non-machine-learning alternative. In the area of supervised learning, support vector machines (SVMs) have been shown to be highly effective in filtering spam, while an approach utilizing K-Means clustering can work well for an unsupervised alternative. [19], [20] Minimum description length (MDL), a principal based on Occam's razor, is the basis for yet another highly effective spam removal method. [21]

X. Conclusion

We have made significant progress towards our initial goal of providing a tool with which users can analyse a given Twitter feed and draw development-related information from the output in order to discover which changes most strongly effect both user and developer satisfaction. Although it falls short of this objective this project brings together Twitter, filtering, sentiment analysis, and automated tweet categorization in a rigorously tested and easy to use Python module, while also allowing for visualization of sentiment trends. We propose some possible direction for future work through which our initial objective may be pursued and successfully brought to fruition.

Acknowledgment

References

[1] X. Yang, D. Hu, and D. M. Robert, "How Microblogging Networks Affect Project Success of Open Source Software Development," in *2013 46th Hawaii International Conference on System Sciences (HICSS)*, Jan. 2013, pp. 3178–3186.

[2] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (R) Evolution of Social Media in Software Engineering," in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 100–116. [Online]. Available: http://doi.acm.org/10.1145/2593882.2593887

[3] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment Analysis of Twitter Data," in *Proceedings of the Workshop on Languages in Social Media*, ser. LSM '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 30–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=2021109.2021114

[4] A. Go, L. Huang, and R. Bhayani, "Twitter sentiment analysis," *Entropy*, vol. 17, 2009.

[5] A. Mudinas, D. Zhang, and M. Levene, "Combining Lexicon and Learning Based Approaches for Concept-level Sentiment Analysis," in *Proceedings of the First International Workshop on Issues of Sentiment Discovery and Opinion Mining*, ser. WISDOM '12. New York, NY, USA: ACM, 2012, pp. 5:1–5:8. [Online]. Available: http://doi.acm.org/10.1145/2346676.2346681

[6] X. Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald, "Microblogging in Open Source Software Development: The Case of Drupal and Twitter," *IEEE Software*, vol. 31, no. 4, pp. 72–80, Jul. 2014.

[7] "Bayesian 0.3.1 : Python Package Index." [Online]. Available: https://pypi.python.org/pypi/Bayesian/0.3.1

[8] "Choosing the right estimator scikit-learn 0.16.1 documentation." [Online]. Available: http://scikit-learn.org/stable/tutorial/machine_learning_map/

[9] "SentiStrength - sentiment strength detection in short texts - sentiment analysis, opinion mining." [Online]. Available: http://sentistrength.wlv.ac.uk/

[10] "Pattern | CLiPS." [Online]. Available: http://www.clips.ua.ac.be/pages/pattern

[11] "TextBlob: Simplified Text Processing TextBlob 0.9.1 documentation." [Online]. Available: https://textblob.readthedocs.org/en/dev/

[12] "Natural Language Toolkit NLTK 3.0 documentation." [Online]. Available: http://www.nltk.org/

[13] "pattern.en | CLiPS," Dec. 2010. [Online]. Available: http://www.clips.ua.ac.be/pages/pattern-en#article

[14] "FAQs about Retweets (RT)." [Online]. Available: https://support.twitter.com/articles/77606

[15] "Tweets." [Online]. Available: https://dev.twitter.com/overview/api/tweets

[16] "Better Bayesian Filtering." [Online]. Available: http://www.paulgraham.com/better.html

[17] L. Hutchinson, "How Twitter's new "BotMaker" filter flushes spam out of timelines," Aug. 2014. [Online]. Available: http://arstechnica.com/information-technology/2014/08/how-twitters-new-botmaker-filter-flushes-spam-out-of-timelines/

[18] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, Mar. 1977. [Online]. Available: http://www.jstor.org/stable/2529310

[19] K.-B. Duan and S. S. Keerthi, "Which is the best multiclass SVM method? An empirical study," in *Multiple Classifier Systems*. Springer, 2005, pp. 278–285. [Online]. Available: http://link.springer.com/chapter/10.1007/11494683_28

[20] F. Qian, A. Pathak, Y. C. Hu, Z. M. Mao, and Y. Xie, "A case for unsupervised-learning-based spam filtering," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38. ACM, 2010, pp. 367–368. [Online]. Available: http://dl.acm.org/citation.cfm?id=1811090

[21] T. A. Almeida and A. Yamakami, "Advances in Spam Filtering Techniques," in *Computational Intelligence for Privacy and Security*, ser. Studies in Computational Intelligence, D. A. Elizondo, A. Solanas, and A. Martinez-Balleste, Eds. Springer Berlin Heidelberg, 2012, no. 394, pp. 199–214, dOI: 10.1007/978-3-642-25237-2_12. [Online]. Available: http://link.springer.com.ezproxy.canterbury.ac.nz/chapter/10.1007/978-3-642-25237-2_12

Appendix


```
{
  "contributors": null,
  "favorite_count": 0,
  "in_reply_to_status_id": null,
  "retweeted": false,
  "favorited": false,
  "coordinates": null,
  "place": null,
  "in_reply_to_user_id": null,
  "id_str": "582064962199117824",
  "lang": "en",
  "user": {
    "profile_background_tile": false,
    "name": "Brian Brown, Ph.D.",
    "url": "http://t.co/DpnqMLJ4Yn",
    "profile_location": null,
    "profile_background_image_url_https": "https://
        pbs.twimg.com/profile_background_images
        /458613392022855680/ezZHUJpt.png",
    "profile_sidebar_fill_color": "DDEEF6",
    "default_profile_image": false,
    "verified": false,
    "profile_link_color": "000000",
    "profile_use_background_image": true,
    "created_at": "Fri Sep 02 19:25:41 +0000 2011",
    "profile_background_image_url": "http://pbs.
        twimg.com/profile_background_images
        /458613392022855680/ezZHUJpt.png",
    "description": "My main Website: http://t.co/
        QWP9z87LDZ I am a professional #webmaster I
        have been a #geek for over 50 years! #
        WordPress #websites",
    "statuses_count": 908518,
    "utc_offset": -25200,
    "geo_enabled": false,
    "id_str": "366757582",
    "is_translator": false,
    "notifications": false,
    "protected": false,
    "following": false,
    "lang": "en",
    "profile_banner_url": "https://pbs.twimg.com/
        profile_banners/366757582/1408466227",
    "profile_text_color": "333333",
    "location": "",
    "friends_count": 153,
    "listed_count": 240,
    "profile_image_url": "http://pbs.twimg.com/
        profile_images/501771376307355648/
        ZB2ldpQA_normal.png",
    "default_profile": false,
    "entities": {
      "description": {
        "urls": [
          {
            "display_url": "brianbrown.net",
            "indices": [
              17,
              39
            ],
            "expanded_url": "http://www.brianbrown.
                net",
            "url": "http://t.co/QWP9z87LDZ"
          }
        ]
      },
      "url": {
        "urls": [
          {
            "display_url": "brianswebworks.com",
            "indices": [
              0,
```

```
            22
          ],
          "expanded_url": "http://www.
              brianswebworks.com/",
          "url": "http://t.co/DpnqMLJ4Yn"
        }
      ]
    }
  },
  "id": 366757582,
  "followers_count": 1454,
  "favourites_count": 0,
  "follow_request_sent": false,
  "screen_name": "BriansWebWorks",
  "profile_sidebar_border_color": "FFFFFF",
  "time_zone": "Pacific Time (US & Canada)",
  "profile_image_url_https": "https://pbs.twimg.
      com/profile_images/501771376307355648/
      ZB2ldpQA_normal.png",
  "is_translation_enabled": false,
  "contributors_enabled": false,
  "profile_background_color": "FFFFFF"
},
"metadata": {
  "result_type": "recent",
  "iso_language_code": "en"
},
"geo": null,
"retweet_count": 0,
"in_reply_to_status_id_str": null,
"in_reply_to_screen_name": null,
"truncated": false,
"entities": {
  "hashtags": [
    {
      "indices": [
        93,
        107
      ],
      "text": "drupalhosting"
    }
  ],
  "user_mentions": [],
  "urls": [
    {
      "display_url": "DrupalHosts.org",
      "indices": [
        0,
        22
      ],
      "expanded_url": "http://DrupalHosts.org",
      "url": "http://t.co/5HhH8NakPS"
    },
    {
      "display_url": "brbr.co/190xZkw",
      "indices": [
        68,
        90
      ],
      "expanded_url": "http://brbr.co/190xZkw",
      "url": "http://t.co/8kxr6PhVEZ"
    }
  ],
  "symbols": []
},
"id": 582064962199117824,
"in_reply_to_user_id_str": null,
"possibly_sensitive": false,
"text": "http://t.co/5HhH8NakPS Awards Top 3 Cheap
    Drupal Hosting for 2015   http://t.co/8
    kxr6PhVEZ\n #drupalhosting",
"created_at": "Sun Mar 29 06:21:15 +0000 2015",
"source": "<a href=\"http://www.ajaymatharu.com/\"
    rel=\"nofollow\">Tweet Old Post</a>"
}
```


```
CREATE TABLE basic_tweets
                                    (user_id integer,
                                        follower_count
                                        integer,
                                        screen_name text
                                        , tweet_id
                                        integer,
                                    retweeted integer,
                                        favourite_count
                                        integer,
                                        tweet_text text,
                                         created_at text
                                        )

CREATE TABLE tweets_with_sentiment
                                        (user_id integer
                                            ,
                                            follower_count
                                             integer,
                                            screen_name
                                            text,
                                            tweet_id
                                            integer,
                                        retweeted
                                            integer,
                                            favourite_count
                                             integer,
                                            tweet_text
                                            text,
                                            created_at
                                            text,
                                            polarity
                                            real,
                                            subjectivity
                                             real )

CREATE TABLE tweets_with_sentiment_and_mood
                                        (user_id integer
                                            ,
                                            follower_count
                                             integer,
                                            screen_name
                                            text,
                                            tweet_id
                                            integer,
                                        retweeted
                                            integer,
                                            favourite_count
                                             integer,
                                            tweet_text
                                            text,
                                            created_at
                                            text,
                                            polarity
                                            real,
                                            subjectivity
                                             real, mood
                                            text,
                                            modality
                                            real)

CREATE TABLE
    tweets_with_sentiment_and_mood_and_category
                                        (user_id integer
                                            ,
                                            follower_count
                                             integer,
                                            screen_name
                                            text,
                                            tweet_id
                                            integer,
```

```
retweeted
    integer,
    favourite_count
     integer,
    tweet_text
    text,
    created_at
    text,
    polarity
    real,
    subjectivity
     real, mood
    text,
    modality
    real,
    category
    text)
```