TweetBuzz: Identifying Buzzwords in a Domain

Prakhar Panwaria Saurabh Aggarwal
Computer Sciences Department, University of Wisconsin-Madison
{prakhar, saurabha}@cs.wisc.edu

***Abstract* —** Since past few years, social media has become an integral part of modern society. The user generated social data, in the form of tweets (microblogs), wall-postings, blogs, video blogs (vlogs), etc. contains great wealth of information and can be exploited to explore any field including the current affairs worldwide. In this paper, we introduce 'TweetBuzz', an application which uses Twitter [10] data, i.e., tweets, to analyze the current buzzwords, the topics which are being highly discussed, in a particular domain. This system can be applied to any area in general, provided we have its knowledge base; and we also show that having some specific knowledge about the domain can also help in improving the accuracy of the system. In our case, we have chosen 'Hollywood movies' as our area, and using a large set of tweets, we try to find out what all movies in a specific genre are popular among Twitter users.

***Keywords* — social media analysis, information extraction, twitter**

1. INTRODUCTION

With the rapidly growing user generated (and unstructured) data via social media, lot of research is being done around information extraction and data mining. The data from online social networking sites, like Facebook and Twitter, can be used to extract a lot of information about what is going on in the world. Applications have been built around tweets to predict the stock market [1], and even detecting catastrophic events like tsunami [2] or hurricane.

In this paper, we describe our application, TweetBuzz, which can be used to identify the popular topics, in a particular area, being discussed by Twitter users. This application can have various use cases when applied to different fields. For example, if we use 'Computer Science' as our domain with specific research areas, say 'Artificial Intelligence', as one of the concepts in our taxonomy, one can use this application to easily know about the recent developments and current focus of research in the field of AI. TweetBuzz can be used for any domain, but to facilitate explaining the application design, we have chosen a small field of 'Hollywood Movies' as our domain. And, as an output, the application generates a tagcloud to visualize the popularity of the movies of a particular genre requested by the user as his/her query.

1.1 Outline

In this paper, in brief, we try to describe TweetBuzz architecture and how it performs on the real world tweets. In the following section, we describe the related work done around extracting information from the data generated from social media. In Section 3, we discuss how we created a taxonomy of Hollywood Movies and other useful data structures. Then, in Section 4, we describe how to process the tweets and identify the relevant movie mentions out of them. In Section 5, we show how we process the user's query and generate a tagcloud of movies, popular among Twitter users. In Section 6, we present the results of our evaluation and compare those with the baselines and other solutions. We give our conclusions in Section 7.

2. RELATED WORK

According to a 2008 survey on information extraction [5], there is lot of work being done in the area since last two decades. Lot of applications are being developed which are centered around extracting meaningful information from unstructured data. In our application, TweetBuzz, we are using tweets to identify the current buzzwords in a particular domain. There are similar applications like Google Trends [3] in which they are using the search engine data to identify the topics, people are most curious about. But, using Google Trends, it is difficult to know the current trends in a particular field, as it returns only the frequently searched words which are similar to the query. Even Twitter provides an API [17] to get top trends from the tweets, but using it, we can only retrieve top 10 trending topics for a specific WOEID (Yahoo! Where On Earth ID) of a physical location. There has been work around trend detection over Twitter

stream as well. TwitterMonitor [18] is one such application, but it is more focused towards analyzing real-time tweets to find frequently appearing (bursty) words. They are not doing any domain specific analysis, and for that reason, they have not built a knowledge base for any domain or in general.

In this paper, as we have used Hollywood movies as our domain, we looked up for similar applications which can find top movies in a particular genre. RottenTomatoes [11] and IMDB [12] seem to do the similar work, but they completely rely on the ratings given by their users or movie reviewers as feedback. Their data is already structured, and they just need to display according to different genres. So, their problem statement is entirely different.

In our application, we are preparing taxonomy and other useful data structures which can help in improving the accuracy of the system. Lot of work has already been done around creation and maintenance of large-scale knowledge bases [7], but in this paper we have not build it large enough, like getting data from multiple sources, for e.g., Wikipedia or DBPedia, and tried to keep it simple for easy explanation of our application design. After creating the taxonomy, we extract movie mentions from the tweets and try to link them with the ones present in our taxonomy. Similar work has been done in Doctagger [6], in which they extract and link mentions to classify and tag tweets. However, in our case, we are not classifying and tagging tweets, but just identifying if the tweet is alluding to a movie.

3. PREPARING TAXONOMY

The first step of our system is creating a knowledge base of movies in the form of a taxonomy. For efficiently matching the tweets with taxonomy nodes, we also create a prefix map data structure. We also generate other useful data structures using domain specific knowledge, which can help improve the accuracy of the system. These steps are mentioned in detail in the following sub-sections.

3.1 Creating Taxonomy Tree

The taxonomy should be large enough to get appropriate results from this application, and, thus, should be able to get updated dynamically. Hence, we use the TMDB (The Movie Database) API [13] to fetch the movies and their genres to construct the taxonomy.

In a nutshell, the TMDB API calls are as follows:

a) `tmdb.getGenreList()`: Returns a list of all movie genres.
b) `tmdb.getGenreMovies(gID,lang,pg)`: Returns a page of movies of the genre with `gID` and language `lang`. This method is called repeatedly for each genre to fetch multiple pages of movies in each genre in order to build a big taxonomy.

The in-memory data structure that we use to store the taxonomy is a 3-level tree, as shown in Figure (1). The top level is the root, the next level contains all genres, and the third level contains a huge list of movies for each genre. If a movie belongs to a particular genre, it is a direct child of that genre in the taxonomy tree.

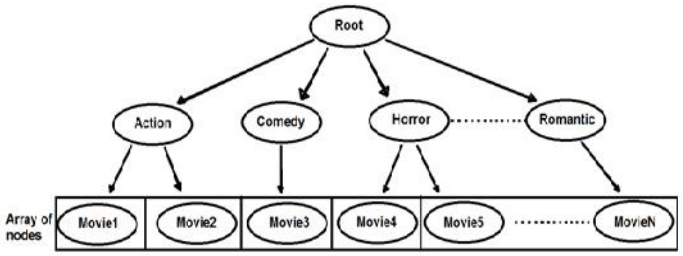

Figure (1) Taxonomy Tree

All nodes of the taxonomy tree are indexed into an array (henceforth, referred to as 'taxonomy node name array'). Now, since building this huge taxonomy using the TMDB API takes lot of time, so we persist the taxonomy in a taxonomy XML in the following format:

```
<taxonomy>
    <node name="Action" id="-1">
        <node name="Movie1" id="1"/>
    </node>
    <node name="Comedy" id="-1">
        <node name="Movie3" id="3"/>
    </node>
</taxonomy>
```

The `id` attribute of the movie node is the movie identifier returned from the TMDB API. This is later used to fetch the movie's cast using TMDP API again. Since this is specific to movie nodes, `id` for genre nodes is set to '-1'.

Once the above XML is built and stored on disk, our system creates the taxonomy tree directly from the XML file on next launch of the application. Whenever the taxonomy tree needs to be updated, we simply regenerate it by making TMDB API calls again.

3.2 Generating Prefix Map

The next step is to make the lookup of the movie names in taxonomy tree faster. We do this the same way as is done in DocTagger [6] using a prefix map. As a black box, prefix map takes in all the movie names, each tweet is run over the prefix map, and the prefix map spills out which movies are found in the tweet text.

The data structure of each entry in the prefix map is `<Key, <NodeId, isLast>>`. Here, `Key` is the prefix of the movie name, `NodeId` refers to the id of the movie node the key refers to and `isLast` signifies whether the current key is the prefix of any other prefix key or not. If `isLast` is true, then the current prefix is not a prefix to any other prefix key.

We use the `insert (String movieName)` method to create the prefix map. This method inserts the movie with name `movieName` into the prefix map by splitting it into tokens separated by whitespaces. For example, for the movie *"Dead Poets Society"*, it will make 3 entries into the prefix map:

i. *<"Dead", <-1, false>>*
ii. *<"Dead Poets", <-1, false>>*
iii. *<"Dead Poets Society", <25, true>>*

(where *25* is the index of the movie *"Dead Poets Society"* in the taxonomy node name array).

Now, if we try to insert another movie *"Dead Poets"*, the 2nd node created in the above example gets updated to *<"Dead Poets", <10, false>>* (where *10* is the index of the movie *"Dead Poets"* in the taxonomy node name array).

3.3 Using Domain Specific Knowledge

Some tweets do not mention the movie name directly, but there may be an indirect reference to the movie by mentioning their casts. Let's consider following tweet for instance: *'Kristen Stewart & Robert Pattinson in Miami to promote their latest movie.'*. This tweet refers to the movie *'Twilight'* but does not specifically mention the movie name.

Our system keeps a data structure to map the actors with all their movies. Whenever we encounter an actor's name, we add some score to the nodes corresponding to each movie of the actor in the taxonomy. However, the score added is relatively low, as at this point we have less confidence that it is actually referring to those movies. But if there are more than one actor's name in the tweet, then the nodes of the movies common to both actors have a high score. So, the key idea is that the tweets that mention more than one actor significantly add to the movie's score, but a single reference does not.

More specifically, the data structure we use is a trie, as shown in Figure (2). The key of the data structure is the actor's name, and the value is a list of indexes of all movie nodes (in taxonomy node name array) of that actor. So, for instance, if *'Tom Hanks'* is one of the leaf nodes in the trie, and if in the taxonomy, there are two of his movies with the node names *'The Terminal'* (node id: *2*) and *'Caste Away'* (node id: *5*), we would have values *(2, 5)* stored in the trie leaf node. The structure is as follows:

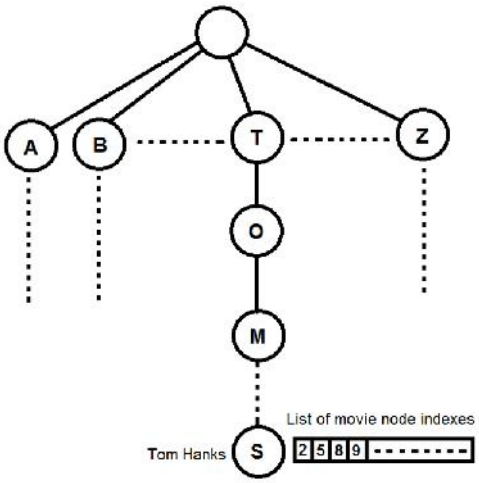

Figure (2) Movie's Cast Trie

4. PROCESSING TWEETS

So, at this point, we have already prepared all the relevant data structures related to taxonomy of movies (like taxonomy node name array, taxonomy prefix map, movies' cast trie), and stored them in memory. Our next step is to start reading the tweets and process each one of them. The output of this step is a big taxonomy node score map `<NodeName, Score>`, a mapping of name of the node in taxonomy and its score. This map is used to get the score of the nodes, closely related to the user's query (see Section 5.1). Processing provided Twitter data turned out to be an expensive process in terms of time, hence it is difficult to process all the tweets for every user's query. Since we have a static data source, we store the taxonomy node score map persistently in a file, so that we don't need to create the data structure again when the application is re-launched. On user's next query, we parse that file to create taxonomy node score map in memory, which we can then use to find out the score of the relevant nodes,

thus preparing the tagcloud as an output. Now, we go through each step of processing the tweet in the following sub-sections:

4.1 Reading a Tweet

For the purpose of our project, we were given around 17GB of static twitter data (with around 7.5 million tweets) as a single huge file. Hence, for our use case, we simply read the file and process tweet one by one. But, our system can be scaled to take real-time tweets as an input, provided we have proportionally scaled hardware resources. The tweets in the file are in JSON format, which we parse to extract the actual text tweeted by the user, using its 'text' tag, and process it.

4.2 Preprocessing a tweet

Next, we preprocess the tweet message. In the preprocessing step, the result is a set of word-level tokens which can be used to identify the movies' mentions. We filter out common characters like white spaces, colon, comma, full stop, question mark, and symbols like *'&'* and *'-'*. We also apply the same filters in the movies mentioned in the taxonomy which makes sure that we are not missing any movies from our taxonomy when we compare the tokens with them. We also remove the words starting from *'@'* and *'RT'*, as we are more focused towards the context of the tweet message rather than a Twitter user.

Since, we are only considering Hollywood Movies in our taxonomy, we are assuming only English tweets will mention them. We consider that a tweet is in English if it has more than half of the words in English. In order to efficiently check if the tweet message is in English, we maintain a dictionary of English words in a trie data structure, against which we compare the tokens we get after splitting the message string over the aforementioned delimiters.

In our algorithm, we give more weight to the hashtags, in a sense that if a hashtag mentions a movie, we are more confident that the tweet is talking about that movie. Hence, in that case, we give relatively higher score to that mention. So, we maintain two different type of tweet message tokens:

a) *Hash tokens*: If we find a token starting with '#', we consider it as a hash token.
b) *Normal (non-hash) tokens*: A token which is not a hash token

So, now the set of generated tokens are clean and contain meaningful text which we now can process to identify mention of the movies in the tweet.

4.3 Using Web Context

Going through the tweets in the data source provided, we observed that users generally include one or more web links in their tweet to describe what they are alluding to. So, we have also considered the web context of the tweet as a relevant factor in identifying whether the tweet is talking about any movie in particular. So, we first check if the tweet contains any web link. If it does, we fetch the title of the corresponding web page (using JSOUP API [16]) , and consider it like yet another message string linked to the current tweet, and hence perform the preprocessing over it to find relevant tokens (we refer them as 'web tokens'). Getting web context is a time-consuming process, so we make sure that we find the message context only after we have valid hash tokens and normal tokens in the tweet.

4.4 Extracting and Scoring Mentions

This is the most important step performed while processing a tweet. In this step, we identify any mentions of movies in the tweet. A mention is denoted by $<n, s>$, where n is the node identifier of the movie node mentioned in the tweet and s is score given to that node by our system. Please note that, unlike Doctagger [6], we are not storing the matched token as we do not need any further information from it. What we need is just the score of relevant nodes in the taxonomy. So, the result of this step is a map of $<n, s>$ entries. We try to extract mentions from all the hash tokens, normal tokens and web tokens generated till now for the tweet. To extract the mentions more accurately, we also use some domain knowledge (see Section 3.3) apart from the prefix map generated from the taxonomy.

As explained in Section 3.3, our system maintains a trie with movies' casts, and map them to their movies in the taxonomy tree. We use this trie to lookup for any mentions of the movie actors in the tweets, and if we find any actor name present in the tweet, we add the mentions of all his/her movies present in the taxonomy, along with their scores. After we used the movie casts as our domain knowledge, we observed a decent amount of increase in Recall, showing that use of domain specific knowledge can indeed help improving the accuracy of identifying the tweet context.

We use the prefix map, which we generated after we created the taxonomy (see Section 3.2) to extract the actual mentions in the tweet. Thus, to match the tweet tokens with the taxonomy nodes, we use method `retrieve(String queryString)` of prefix map. This method returns the value for the key `queryString` from the prefix map. The return value, as mentioned before, is of the form: *<NodeId, isLast>*.If the value returned is *null*, we move on to the next word. If it is non-null, and if *isLast* is true, we stop appending text to the query string with subsequent words, otherwise we keep on appending subsequent words to lookup in the prefix map again. For example, for the tweet *'I watched Dead Poets Society'*, queryStrings generated are as follows, considering the prefix map of the example mentioned in Section 3.2:

i. *"I"*: returns *null*
ii. *"watched"*: returns *null*
iii. *"Dead"*: returns *<-1, false>*
iv. *"Dead Poets"*: returns *<10, false>*
v. *"Dead Poets Society"*: returns *<25, true>*

Hence, as a result for this tweet, we get *"Dead Poets Society"* as a movie mention, along with some score given by the system.

4.5 Using Go-Words

In order to improve the accuracy of the system, we can also use another form of domain specific knowledge, which we refer to as 'go-words'. Basically, it is the opposite of 'stop words' in a sense that more the stop words in the tweet, less contextual information it contains, but more the go-words in the tweet, more contextual information it has, and hence, more relevant the tweet is. Browsing through 200 tweets in our golden data set and reading movie blogs on internet, we manually generated a list of go-words, and their value. Their value denotes the confidence of the tweet referring to a movie in our taxonomy if that go-word appears in the tweet. For instance, we include words like *'watch'*, *'releas'* and *'trailer'* in our go-words list. Please note that we have stemmed the go-words (like, we have used *'releas'* instead of *'release'*, *'released'*, or *'releasing'*) so that we can simply check if the words in the tweets start with any of the go-words..

For matching the tweet tokens with go-words efficiently, we again use trie data structure to store the go-words.

1) We create a trie of all the go-words, with the go-words as the key, and their value in the leaf node.
2) For each token in the tweet, we run it across this go-word trie. If there is a match, we set the tweet scaling factor as the go-word's value. In case of match with another token in the same tweet, we set it to the maximum of current scaling factor and recently matched go-word's value.
3) If there is no go-word found in the tweet, we assume that the tweet might not be alluding to a movie, hence, we give a scaling factor of `(1 - MAX_FACTOR)`, where `MAX_FACTOR` is the maximum scaling factor given to a go-word.

After, we have calculated the scaling factor of the tweet, we apply it to all the mentions generated till now for the tweet, thus increasing or decreasing the scores of the movies' nodes in the taxonomy, according to the confidence of the movie being referred to in the tweet.

4.6 Filtering Mentions

At this point, we will have a number of movie node mentions and their score. In order to filter out the mentions that might not be relevant (for example, the mentions which got score on the basis of their cast (actors) being mentioned in the tweet), we drop the mentions which have a score less than a threshold.

4.7 Updating Global Score Map

This is the last step performed while processing each tweet. At this point, we will have a set of relevant movie mentions for the tweet. So, we update our global taxonomy node score map, which will be used while processing user's query (see Section 5.1), with these mentions. We get the score of the nodes present in the current mentions, and simply add it to the existing score of an entry with the corresponding node id in the global map. So, after we are done with processing all the tweets, we will have a big taxonomy node score map containing the movie mentions from all the tweets. As mentioned before, since we have static twitter data, we store the global taxonomy node score map persistently in a file, and we use it directly whenever we restart our system for answering user's query. In case our twitter data set changes, we recreate the global map and use it again for further queries.

5. PROCESSING USER'S QUERY

Now, we are ready to take user's query as an input, and output a tag cloud featuring the names of the movies, related to user's query, which have been the buzzwords in the tweets we have as static data. The output may also

represent the movies that have been highly famous, or infamous, among the users (at least those on twitter). Since, we have a small taxonomy (a three level tree with only 3896 movies in it), we limit the user's query to be one of the genres, which are essentially 2nd-level nodes of the taxonomy (also shown in Figure (1)). Following are the steps used in answering user's query.

5.1 Getting Scores of Relevant Nodes

First of all, we get the user's genre query as an input and find the corresponding genre node in the 2nd-level nodes of taxonomy tree. Now, since we want to find out the movies of that particular genre which have been discussed in the tweets, we have to get the scores of those movie nodes. We can easily get those by iterating through the children (3rd-level movie nodes) of queried genre node in the taxonomy tree, and getting the score of the node with corresponding node id in the global map.

5.2 Preparing TagCloud

After we have scores of the movies of the queried genre, we shortlist, for simplicity, top 30 of them according to their scores. Then, we give each of them a tag cloud level on 1 to 10 scale according to the minimum and maximum score of those shortlisted movies. We, then, create the tag cloud by preparing a well formed HTML using a custom style sheet. For instance, if the query is *'Action'* and *'Musical'*, we generate tagcloud shown in Figure (3a) and (3b).

Figure (3a) TagCloud for Query - 'Action'

Figure (3b) TagCloud for Query - 'Musical'

6. EVALUATION

In this section, we present evaluation of our system. We basically discuss how our application stands on runtime, scalability and accuracy.

6.1 Runtime

The code runtime for each part of the algorithm is as follows:

Let number of genres in the taxonomy = *G*
Let number of movies in the taxonomy = *M*
Let number of characters in a tweet = *N (~= 143)*
Let average number of words in a tweet = *W*
Let number of characters in a web document title = *T*
Let number of mentions found in one tweet = *X*

Preparing Taxonomy and Trie:

1) Creation of Taxonomy XML = *G API calls*
   The bottleneck in this step is the network bandwidth & TMDB API response-time.
2) Reading taxonomy XML = *M*
   Single parse step for each movie.
3) Creating taxonomy tree = *3* x *M*
   Because the tree is a 3-level tree, so each movie insertion takes 3 steps.
4) Creation of Movie Cast XML = *M API calls*
   The bottleneck in this step is again the network bandwidth & TMDB API response-time.
5) Reading taxonomy XML = *5* x *A*
   Because we fetch top 5 casts for every movie.
6) Creating movie cast trie = 20 x 5 x M = *100* x *M*
   Assuming average length of a movie cast's name to be 20 characters long, insertion in a trie takes as many steps as the number of characters in the name.

Processing Tweets (complexities per tweet):

1) Preprocessing = *N*
   Linear lookup of non-alphabetic characters and doing preprocessing steps accordingly.
2) Tokenize tweet = *N*
   Lookup of whitespace and other separators in a tweet.
3) Web context = *N + 1 + T*
   Finding hyperlinks in the tweet takes N steps. JSOUP [16] API call (which is the bottleneck) takes 1 step to get the title of web document. T steps to tokenize the title returned.
4) Extracting movie mentions = *W*
   Linear in the number of words in tweet (using prefix map).
5) Extracting movie cast mentions = *N*
   Trie lookup is linear in the number of characters in the tweet.
6) Go words lookup = *N*
   Trie lookup again, linear in the number of characters in the tweet.
7) Filtering mentions = *X*
   Linear in the number of mentions found.

Finding final results:

1) Preparing tag cloud = *U(X)*
   Where *U(X)* stands for the set union of mentions found in each tweet.

Overall Complexity:

Since we prepare the taxonomy just once and then persist the Movie & Movie cast XML's, that overhead is incurred just once. The main complexity lies in the tweet processing step.

So, Total complexity = $5N + W + T + X + 1$

Now, since number of characters in a tweet is much greater than the number of words and mentions in a tweet or words in a title,

$$N >> W, T, X$$

So, overall complexity per tweet = $5N$, where N is the number of characters in a tweet.

6.2 Scalability

Here, we try to evaluate whether our system is scalable enough to process the tweets coming from live Twitter stream instead of reading it as static data from a huge file.

Number of tweets in the provided Twitter data	~7.5 million
Time taken by TweetBuzz to process all the tweets	~10 minutes
Average number of tweets processed by TweetBuzz	~12500 / second
Maximum number of tweets observed in live Twitter stream	~9000 / second [ref. 15]

Table (1) Scalability Evaluation

Thus, from Table (1) and the value of average number of tweets processed per second, we can see that our system is capable of processing the data coming from live Twitter stream.

6.3 Accuracy

We ran our system over the golden data set containing 200 tweets. We present the results of the system's accuracy evaluation over the golden data set in terms of Precision (*P*), Recall (*R*) and a function (*F*) with the value $2PR/(P + R)$, on the scale of 0 to 1. In our case, precision means the ratio of the number of correct movie mentions identified by TweetBuzz over the golden data to the total number of movie mentions identified by TweetBuzz, and recall means the ratio of the number of correct movie mentions identified by TweetBuzz to the actual correct movie mentions identified manually. In Table (2) below, TweetBuzz includes use of movie cast and go-words knowledge, and also web context in the system design. Then, we compare our results with the solutions having less features, like without web context, without movie cast, and, then, without any of these features (i.e., simple matching with the prefix map).

Solutions	P	R	F
TweetBuzz	0.07	0.82	0.13
No Web Context	0.07	0.82	0.13
No Movie Cast	0.20	0.33	0.25
Naive	0.20	0.33	0.25

Table (2) Accuracy Evaluation

Following are some of our observations and lessons learned while doing accuracy evaluations:

- Using the movie cast to increase the score of a movie node works well to increase the recall of the system (from 0.33 to 0.82), but the precision goes down (from 0.20 to 0.07). This is because a cast or even a pair of casts can be in more than one movie. For example, a tweet mentioning *"Robert Downey Jr."* may refer to *"Iron Man"* or *"Sherlock Holmes"*.
- The reason why web context doesn't seem to improve accuracy of our system over our golden data set can be that we only lookup the title of the web document so that the tweet processing is fast enough. Looking at the whole web document is very time consuming and may also return false positives For example, if the link points to a news website, then we may discover news articles on the same page that are not relevant to the tweet.
- We also noticed that using go-words feature did not add much to the recall. This is probably because we manually added the list of go-words and intuitively assigned them scores. This can be better handled by reverse learning the words that are used in the tweets which are about movies. But this would need much larger training data (golden data) set, in terms of number of tweets, than what we currently had.
- The precision and recall of the system varies a lot depending on the source of tweets. The initial golden data we tried to collect was from the twitter handle *"@Movies"*. That turned out to be a bad idea because all the tweets had the movie names perfectly mentioned in the tweet. In such cases, where simple string matching over the movie name works, naive algorithm's accuracy comes out to be higher.

- Our final golden data was collected as a combination of different types of tweets, some which directly mentioned the movie name, some which seemed like they mentioned the movie name but they did not, and some that indirectly talked about a movie. This was done using Twitter4J API [14] and searching for tweets with different keywords.

7. CONCLUSION

In this paper, we have discussed the design of our application, TweetBuzz, which can take Twitter data as its input and generate a tagcloud of buzzwords in a particular domain. We used 'Hollywood Movies' as our domain, so the application outputs a tagcloud of the movies which are highly discussed in the Twitter world. We also show how using domain specific knowledge (for instance, Movie Casts and Go-Words) can improve accuracy of the system. There can be various use cases of this application when applied to different fields.

REFERENCES

[1] Bollen, Johan, Huina Mao, and Xiaojun Zeng. "Twitter mood predicts the stock market." *Journal of Computational Science* 2.1 (2011): 1-8.

[2] eddine Dridi, Houssem. "An approach for analyzing textual data in microblogs."

[3] Choi, Hyunyoung, and Hal Varian. "Predicting the present with google trends". *Economic Record* 88.s1 (2012): 2-9.

[4] Mangold, W. Glynn, and David J. Faulds. "Social media: The new hybrid element of the promotion mix." *Business horizons* 52.4 (2009): 357-365.

[5] Sarawagi, Sunita. "Information extraction." *Foundations and trends in databases* 1.3 (2008): 261-377.

[6] Doan, AnHai. (2013). "Entity Extraction, Linking, Classification, and Tagging for Social Media: a Wikipedia Based Approach". Unpublished manuscript.

[7] Doan, AnHai. (2013). "Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches". Unpublished manuscript.

[8] Wong, Felix Ming Fai, Soumya Sen, and Mung Chiang. "Why watching movie tweets won't tell the whole story?." *Proceedings of the 2012 ACM workshop on Workshop on online social networks*. ACM, 2012.

[9] Asur, Sitaram, and Bernardo A. Huberman. "Predicting the future with social media." *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference* on. Vol. 1. IEEE, 2010.

[10] Twiiter. http://www.twitter.com/.

[11] Rotten Tomatoes - Top Movies. http://www.rottentomatoes.com/top/.

[12] IMDB - Genres. http://www.imdb.com/genre/.

[13] The Movie Database. http://www.themoviedb.org/.

[14] Twitter4J. http://twitter4j.org/en/.

[15] YearInReview - Tweets Per Second. https://blog.twitter.com/2011/yearinreview-tweets-second.

[16] JSOUP. http://jsoup.org/.

[17] Twitter - Get Trends API. https://dev.twitter.com/docs/api/1/get/trends/%3Awoeid

[18] Mathioudakis, Michael, and Nick Koudas. "Twittermonitor: trend detection over the twitter stream." *Proceedings of the 2010 international conference on Management of data*. ACM, 2010.