CS 224w - Porject Final report *Hooyeon 'Haden' Lee; hlee0 (05382015); 12/09/2011*

Using Twitter to Estimate and Predict the Trends and Opinions

1 Introduction

The most common and conventional way to collect people's opinions has always been random sampling and asking several survey questions by phone. For instance, if a news media is interested in how popular Obama is among Americans, it would probably call up a thousand people and ask how they would rate Obama. The data collected in this manner is considered a gold standard; there has been only few attempts to find a different way to collect such data. However, this approach bears a few shortcoming such as costs, limited targeted people, etc.

In this paper, we attempt to predict people's opinions and trends by analyzing the Web data such as Tweets on Twitter. There are several interesting questions we could ask, given millions of Tweets: how people feel about economy; how people feel about Obama - job approval rate; what people think about the top five contestants left in the TV show American Idol. Collecting polls by asking people in person or by phone is costly and time-consuming, but if we could get the same results from the freely available Twitter data, it could supplement or supplant the conventional way of collecting the data [1].

2 Problem Description

As briefly introduced in previous section, we would like to solve the following problem:

Problem: *Can we analyze the Twitter data in order to estimate and predict the future trends and opinions of people?*

To formalize, let us define a time series variable $p(t)$ that we would like to predict; for instance, Obama Job Approval rate that ranges in 0% and 100% could be such a time series variable. Then our problem simplifies to predicting $p(t)$ using the Twitter data.

Since the Tweets are also time series data, we can denote another time series variable $q(t)$ that can be obtained by some text analysis over the Twitter data (this could be, for instance, a certain word's frequency at time t). Hence, we could try to correlate these two time series $p(t)$ and $q(t)$ via learning models such as linear regression model. Simply put, we would like to first estimate $p(t)$ as some function of $q(t)$:

$$f_\beta(q(t)) \sim p(t+\beta) \tag{1}$$

Where the time parameter β specifies how much of time we want to predict. For instance, $\beta = 7$ days would mean that we want to predict $p(t+7)$ using the data up to time t (hence, this is 7-day forecast). Our problem, then, is to find (or learn) the functions f_β for given β, where we can obtain the time series $q(t)$ from Twitter and can observe $p(t)$ via external polling data such as Gallup.

3 Related Work

Google recently developed an online tool called *Google Flu Trends* [2], which attempts to predict the number of influenza patients based on the search queries people enter on Google.com. The actual number of patients in the United States is reported by the Center for Disease Control (CDC), but the report comes out with a 2-week lag in that it is released two weeks after the time when actual physician visits happened. In this work, the authors attempted to train models and learn the predicting function g_β via a simple linear regression model over log-odds of the following two events: $p(t)$ that is the fraction of influenza-related physician visits among all visits and $q(t)$ that is the fraction of search queries that are influenza-related. This simple model provided a good estimate of the number of flu patients, which matched the actual CDC report that comes out 2 weeks later. In their model, β was set to zero, meaning that the authors only attempted to predict the number of patients of the given day, using the search query fraction data of the same day - and thsi prediction was verified via the CDC rerpots that come out two weeks later.

Another closely related work is done by a group of authors at Carnegie Mellon University in a recent paper called From Tweets to Polls[1]. In this work, the authors attempted to solve a very similar problem using the Twitter data. The authors trained a linear model after applying the sentiment analysis over each Tweet by checking whether a Tweet contains a positive word and/or a negative word; a Tweet could be marked either positive or negative or both, according to the sentiment analysis lexicon they used. For predicting Obama's Job Approval rate, they limited the input space to the set of the Tweets that contain a word 'Obama,' on which the sentiment analysis was performed. A simple linear regression model was successful in estimating the future approval rate, but not by much from the baseline prediction. For predicting other tredns such as the consumer confidence index, the authors report that their model did not work well.

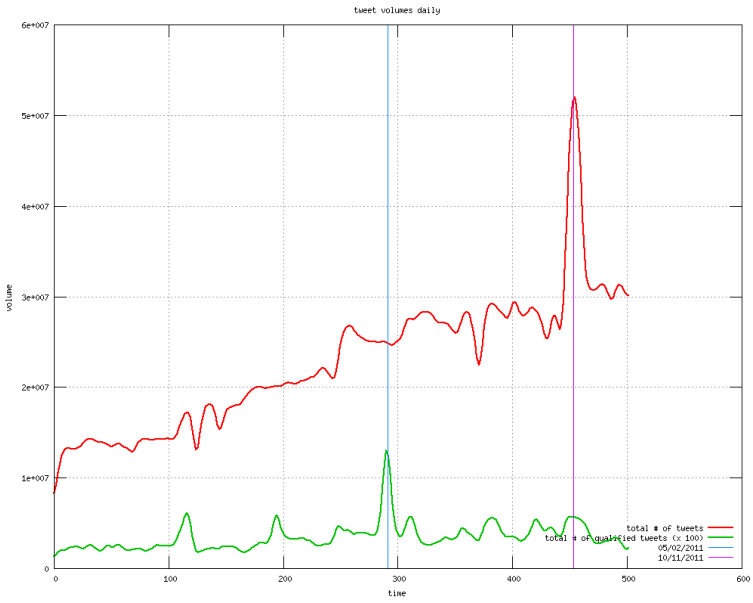

Figure 1: Time vs Tweet Volume

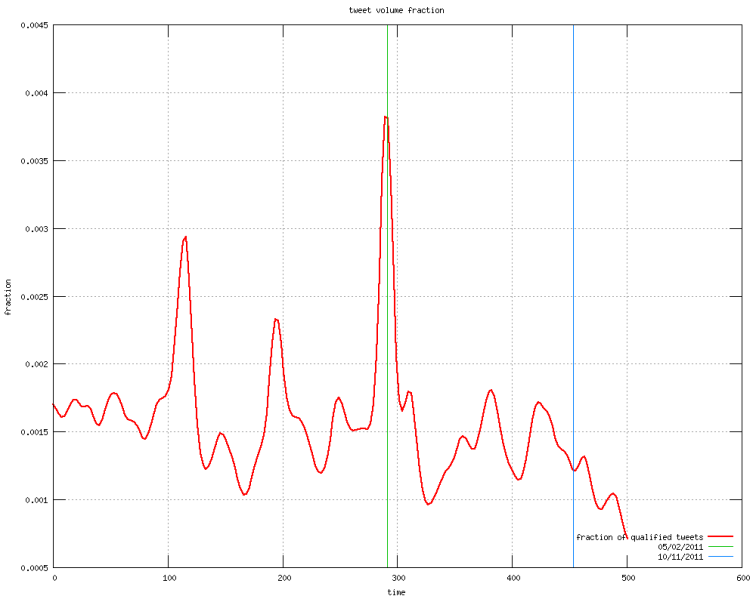

Figure 2: Time vs Qualified Tweet Fraction

4 Data

4.1 Data Collection

We used the publicly available Tweets from Jul'10 to Nov'11, which were crawled via public Twitter APIs and other tools. For the trend data, we collected the following data, but only had time to work with the Obama Job Approval rate data.

(1) Obama Job Approval: Gallup data

(2) American Idol: Top 12 Elimination Polls

(3) Economy: Consumer confidence index

Table 1: Top words with respective train/test errors with $\mu = 3$

Rank	Word	Train Err (Single)	Test Err (Single)	Train Err (Agg)	Test Err (Agg)
1	cong	1.88353	3.44086	1.88353	3.44086
2	nov	1.89899	4.3944	1.7073	3.83679
8	#2011	1.9683	3.92031	1.30232	3.85882
9	#thatawkwardmoment	1.973	5.88304	1.2216	4.41469
11	#nv	1.9891	4.69948	1.16341	5.07407
14	images	1.99633	3.17813	1.1433	3.9654
15	#vote2010	1.9996	5.07752	1.12839	3.5898
17	#whyivote	2.00431	4.94258	1.1247	3.61808

4.2 Data Statistics

In this work, we used all public Tweets from 7/15/2010 to 11/30/2011, inclusive, and the Gallup's Obama Jab Approval polls data from 7/15/2010 to 11/30/1011, inclusive. During this period, we have the following statistics (here, 'Qualified Tweets' refer to the Tweets that contains 'Obama' (case-insensitive) as a substring):

# of days	504
# of Tweets (Total)	11,470,198,016
# of Tweets (Daily)	22,758,329
# of Qualified Tweets (Total)	1,737,394
# of Qualified Tweets (Daily)	34,471

To give a better sense of the Twitter data, Figure 1 shows the time vs. volume of Tweets plot while Figure 2 shows the fraction of Qualified tweets. The green line in Figure 1 (the number of Qualified Tweets) is scaled up by multiplying by 100. The left-most vertical line represents the date on which Osama Bin Laden was killed; we see that the number of qualified Tweets bursts out on that data while the total number of Tweets did not change that much. On the other hand, during the week of 10/11/2011, the 'Occupy Wall Street' movement/riot was prevailing around the globe, and the total number of Tweets spiked during this week as shown with the right-most vertical line in Figure 1, while the fraction of qualified tweets did decrease by a bit (because not all bursting tweets mentioned Obama). These two plots together indicate that people do share their opinions via Twitter and we can easily see from the statistics that such trends could be captured if enough people are tweeting about the event or person.

5 Method

To solve the problem of predicting the trends and opinions, we first consider the following assumptions and pre-processing step.

5.1 Assumptions and Pre-Processing

First of all, given a prediction problem (or a topic), we must admit that not all Tweets are about the given topic; in fact, the majority of Tweets would be 'noise' for most topics. Hence, it is important to limit our input space (or the data) to a proper set of Tweets that are talking about the given topic. This introduces a different but relevant problem called 'topic analysis'; given a Tweet, can we identify what topic(s) this Tweet belongs to? Since this problem itself is an interesting, difficult problem, we do not attempt to solve this problem in this work, but we would simply filter the Tweets as a pre-processing step. That is, given a topic, we will only consider the Tweets that contain a certain word (or certain words) related to the topic. For instance, for the Obama job approval prediction, we could simply consider the Tweets that contain 'Obama'.

Second of all, as the authors in [1] pointed out, Twitter has its own language in the sense that people post Tweets using hash tags, URLS, Internet slangs, etc., in order to express their opinions within 140 characters. Considering that each Tweet consists only of 11 words on average, this is barely a short sentence. Hence, a simple sentiment analysis may be work well for analyzing the twitter data. In this work, we propose that the term frequency count (single terms, bi-grams = two adjacent words, etc.) and aggregated term frequency count (top k single terms, top k bi-grams, etc.) could be used in learning the time series prediction model.

5.2 Learning Models

We used linear regression model (on the top of log-odds of the two time-series data) where the features are word frequencies and target variables are Obama's Job Approval rate.

To be more formal, let us define the following: n is the total number of days, in our case it is 504. In our Vocabulary (English words and hash-tags), we have $m = |V| = 346,880$ words. For each word w_i, the corresponding feature vector $x^{(i)} \in \mathbb{R}^n$ is defined as: $x_j^{(i)}$ is the word frequency of w_i on day $j = 1, 2, \ldots, 504$. Henceforth, the superscript $^{(i)}$ denotes the i-th training example. In particular, $\sum_{i=1}^{m} x_j^{(i)} = 1$ for all j (i.e. the sum of word frequency equals to one on any given day). Finally, $\vec{y} \in \mathbb{R}^n$ is defined as the Gallup's poll data on each day.

5.2.1 Linear Regression - Single Word

We can easily think of a linear regression model to learn the parameters $\theta^{(i)} \in \mathbb{R}^2$ using just a single word w_i by solving the famous normal equation:

$$\theta^{(i)} = [(x^{(i)})^T x^{(i)}]^{-1}(x^{(i)})^T \vec{y} \tag{2}$$

For convenience, we define $\theta_0^{(i)}$ to be the intercept term, and we would modify the $x^{(i)}$ vector above to be a $n \times 2$ matrix such that its first column is all 1's, and the second column is $x^{(i)}$. Solving this normal equation will yield the best-fitting parameters $\theta^{(i)}$ for word w_i.

5.2.2 Linear Regression - Aggregate Words

Now, suppose we have learned $\theta^{(i)}$ for all m words. We can sort them by the training error to get the best k words that work best on the training set, and use these top k words altogether to learn a new linear regression model. In this case, the parameters $\Theta^{(k)}$ we want to learn would be in dimension $k+1$ as opposed to dimension 2 in the single word case. That is, $\Theta_0^{(k)} + \sum_{i=1}^{k} \Theta_i^{(k)} x_j^{(i)}$ would be our prediction on day j, using the top k words. Learning Θ is identical to learning θ in that we just have to solve the normal equation:

$$\Theta^{(k)} = [(X^{(k)})^T X^{(k)}]^{-1}(X^{(k)})^T \vec{y} \tag{3}$$

where X is the design matrix containing the top k feature vectors as rows.

This 'aggregated linear regression' approach mimics what Ginsberg, et al. did in their work [2].

5.2.3 Time Parameters

Notice that the above linear regression models try to use the Tweets to predict the polls data on the same day. However, it would be more useful and practical if we could predict the polls result t days ahead of time. For instance, the Tweets posted today may be a good indicator of the true polls data in 3 days or a week. Hence, we introduce the time parameter β that indicates how far in the future we would like to predict. In our work, we would consider the values $\beta = 0, 1, \ldots, 30$ ($\beta = 0$ means predicting the same-day trends, and $\beta = 7$ means predicting the trends 7 days ahead).

Then, the above models would be modified accordingly when we introduce the parameter β as follows.

For single-word version, we use the data $x^{(i)}$ on day j to predict $y_{j+\beta}$:

$$||y_{j+\beta} - (\theta_0^{(i)} + \theta_1^{(i)} x_j^{(i)})||^2 \tag{4}$$

The term in parenthesis is our prediction, and the above expression is the squared error of the prediction for day $j + \beta$.

For top-k word aggregate model, we get similar formula:

$$||y_{j+\beta} - (\Theta_0^{(k)} + \sum_{i=1}^{k} \Theta_i^{(k)} x_j^{(i)})||^2 \tag{5}$$

6 Experiments and Findings

6.1 Experiment Setup

Single-word Model

For each experiment, we have to fix a set of parameters. The time parameter β is set to an integer value between 0 and 30. Since we have 504 days of data, we train our model on days $[t_s \ldots t_e)$ and test our model on days $[t_e \ldots 504)$ (our days are 0-indexed). With these three parameters fixed, we learn single-word linear regression models for each word in our vocabulary V (recall that $|V| = 346,880$), and we sort the words by how well they fit the target variables (Obama Job Approval rates).

Aggregate Model

Based on this measure, we pick the best k words (where $k = 1 \ldots 100$) and train an aggregate-word linear regression model as explained earlier. Our findings focus on using these aggregate-word models, and our single-word models play a role as a feature selector or filter to learn ggregate-models.

6.2 Results

Table 1 summarizes the train/test errors when $\beta = 3$, showing some of the top 20 words due to the limited space. Notice that we see some politically relevant words such as 'cong' (congress), '#vote2010', '#whyivote', etc. Also notice that in aggregated model, the train error continues to decrease because we are introducing more features to fit the data (however, since it will eventually overfit, the train error of this model would decrease and then increase).

Figure 3: Training Set Size vs Error

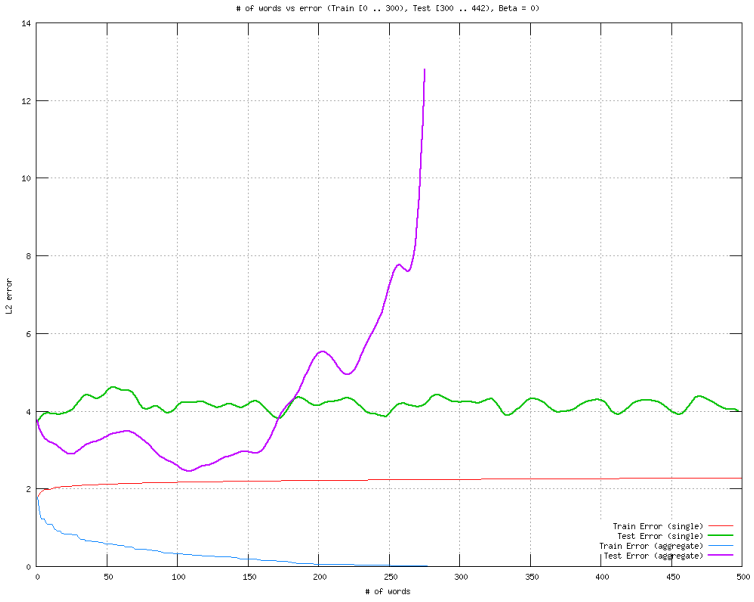

Figure 4: Aggregate Model Size vs Error

However, we delve into our data sets and results in depth in order to understand exactly what happens with our models. As we vary a particular parameter while set others fixed, we can understand the effects of the varying parameter. The following sections will discuss what each of the parameters tells us about the data.

6.2.1 Effects of Training Set Size

For any fixed β and x where x is the number of words used in aggregate model, as we vary $t_e - t_s$ (which is the size of training set), we would expect that the larger training set will help our model learn the target trends. For instance, training on 5 days of data would make our model too simple, while training on 200 days of data would yield a more reasonable model. Figure 3 shows how train/test errors change when we vary t_e from 100 to 300 (x-axis) while we fix parameters $\beta = 0, t_s = 0, x = 20$. In this plot T represents the split-point of train set and test set, which is essentially t_e. As we see, the test error (in bold green) has a decreasing trend in that our models benefit from more data, while train error (in red) increases and decreases but does not fluctuate much.

6.2.2 Effects of Size of Aggregate Model

For any fixed β, t_s, and t_e, an aggregate model is built on top of the best x words measured by single-word models. By varying this x, the size of aggregate model, we can see the effects of having fewer or more words in aggregate model. Naturally, we would expect that our models will fit the training data better as we have more words because that means we are having more features - hence, we can over fit the training data set. However, for the train-error, it will initially decrease as our model can capture the trend better, but eventually it will blow up due to the overfitness.

In Figure 4, we can clearly see this effect. As we add more words to the aggregate model, we see that the train error continues to decrease (in blue), while the test error (in purple, bold) is minimized at around 105 words, and then starts to blow up; after having 270 words, the aggregate model's test error was too huge to fit the plot. The red line represents the train error of each word in single word model, and this is monotone-

increasing because we sorted the words by their train errors. The green line represents the test error of a single word, which appears to be a bit random as single word is clearly not a good predictor.

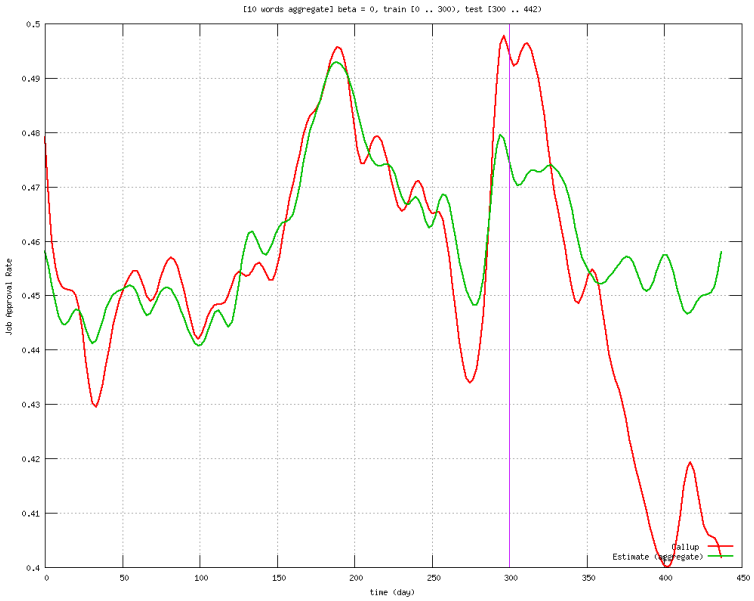

Figure 5: Prediction Using 10 Words

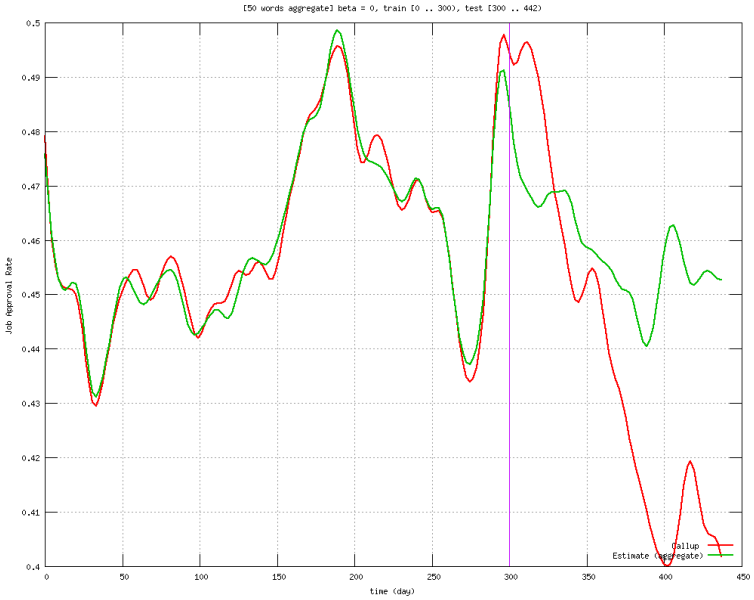

Figure 6: Prediction Using 50 Words

For comparison, Figure 5 and Figure 6 show how our aggregate models predict the trends when we keep all parameters the same except for the number of words used in aggregate model. In these two plots, the red curve is the target trend (Gallup data), and the green curve is our model's prediction. For these plots, we fixed $\beta = 0, t_s = 0, t_e = 300$, but the numbers of words used are 10 and 50, respectively.

Notice that Figure 5 does capture the trains but not quite even in training set, while Figure 6 captures the trends in training set almost perfectly, while it still manages to capture ups-and-downs-of-trends in training set – even though towards the right, it is slightly off, it does capture the moment when a hump comes in at around day 400, and a drop at around day 410.

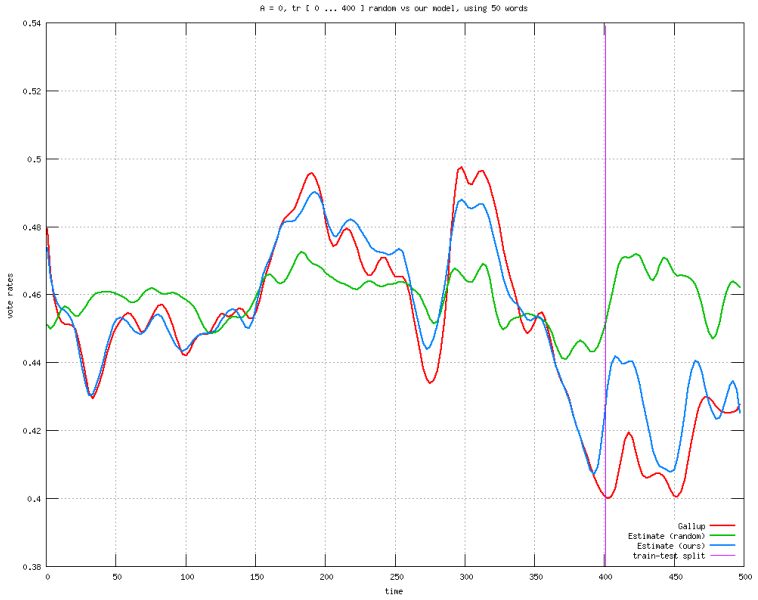

Figure 7: Our Model vs Random Model #1

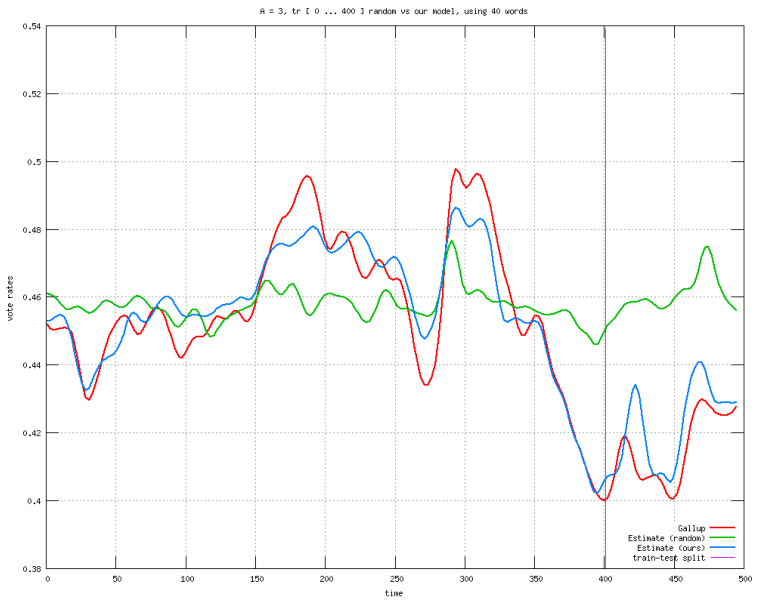

Figure 8: Our Model vs Random Model #2

6.2.3 Comparison with Random Models

From previous results, we confirmed that aggregate models do work better as we use more words, which begs the following question: what happens if we just randomly pick words from our V and train aggregate models? Would these model fit the trend curve just as perfectly as our models? If the answer is yes, then our models are not any better than randomly picking a set of words and learn linear regression models. Here, we fix all the parameters β, t_s, t_e, and x while we train our aggregate models with a different set of words: the best x words (our model) and the random x words (random model).

Figure 7 clearly shows that our model fits the test data very well while the random model cannot even fit the training set very well. The difference is more obvious in the test set (on the right side of the vertical line). Here, the parameters used are $\beta = 0, t_s = 0, t_e = 400, x = 50$. This confirms our assumption that filtering the features (words) by how well they fit the target trend individually, via single-word modes, is a good feature selection process (because we used the same aggregate linear models just with different set of features). Similarly, for parameters $\beta = 3, t_s = 0, t_e = 400, x = 40$, Figure 8 shows similar results.

6.2.4 Weights of Words

Along with the previous section, there is a different way to check whether our selection of words in aggregate model makes sense. That is to look at the actual weights (or θ values in our equations) in aggregate model and see how these weights vary. Intuitively, if our word set is bad or random, these weights should fluctuate a lot as we vary the size of aggregate model while keeping other parameters fixed, because in such a set, the words are not meaningful features to predict the trends. On the other hand, if our word set is a good predictor, then we should see a rather stablized weights across the words, such that some words are positively correlated to the trend while others negatively.

Figure 9 and Figure 10 show how the weights of the top 10 words change while we change the aggregate model size from 1 to 50. Hence, the x-axis represents the number of words used in aggregate model, and there are 10 curves shown in each plot. For instance, in Figure 9, the top red curve is for the top word 'bump', whose weight is pretty much stablized after about 12 words, while the navy curve at bottom for the 8-th word '#wa' starts at $x = 8$ (because it is not included in smaller aggregate models) and its weight does not fluctuate much either. Figure 9 was generated with parameters $\beta = 0, t_s = 0, t_e = 300$. We see similar results in Figure 10 where the parameters were $\beta = 3, t_s = 0, t_e = 400$.

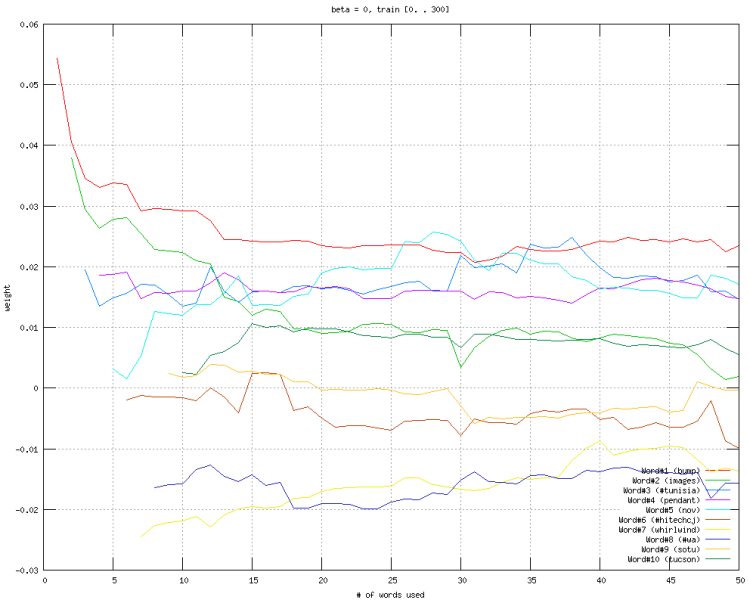

Figure 9: Aggregate Model Size vs Weights #1

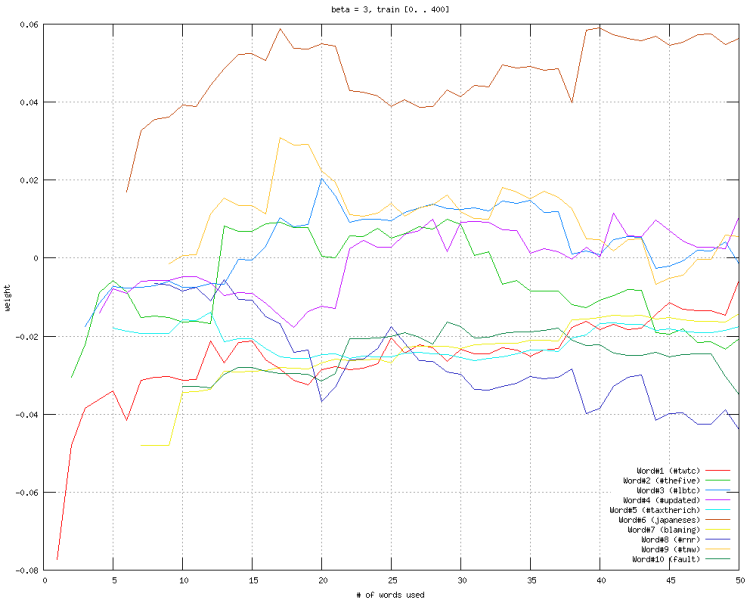

Figure 10: Aggregate Model Size vs Weights #2

7 Summary and Conclusion

In the previous sections, we learned that our aggregate models benefit from a larger data set (larger training set) and a larger set of features (more words in model). This is expected and reasonable because our models should collect enough data in order to learn the target trends, and larger set of features helps fitting the target trends as well.

Furthermore, Section 6.2.3 and Section 6.2.4 confirmed an important fact that our feature selection process via single-word model is a promising process as our aggregate model makes a significantly better prediction that any aggregate model with randomly picked words. We were able to evaluate our model againt the random aggregate models, and confirmed that our model outperforms.

Although we only showed a handful number of plots for a small set of parameters in this paper, the results were consistent across all values of parameters $\beta \in [0 \dots 30], t_s \in [0 \dots 300], t_e \in [t_s + 50 \dots 504], x \in [1 \dots 100]$. However, for large values of β, our aggregate models started predicting less accurately, and this indicates that predicting the far-future is more difficult than the near-future using Twitter data, which confirms that Twitter data is volatile (which counts for day-to-day trends, in some sense) as the authors in [1] claims in their work, too.

There were of course issues with our models as well. As discussed briefly earlier, our best-performing words in single-word models include junk words that are irrelevant to the topic. Our on-going work indicates that this could be fixed if we use bi-grams instead of uni-grams (and this is going to be discussed in Section 8).

Due to the limited time and resources, many of our initial ideas were not implemented or some of the experiments are still running, and these shall be addressed in the next section.

8 Future Work

There are a number of ideas that could improve the performance of our model.

8.1 Natural Language Procesing

In Natural Language Processing, it is often helpful to use bi-grams (two adjacent words) instead of uni-grams (a single word), and our models could easily adopt the bi-gram features because our models do not rely on the uni-gram-ness. We are currently running the experiments with bi-grams, and the pre-processing step (counting the frequency of bi-grams) is partially completed. We selected bi-grams such that each word must come from our vocabulary V for the uni-gram model (this ensures that any bi-gram consists of two important uni-gram words), and the most frequent bi-grams are all relevant to our topic, which is different from our uni-gram case. Some of such bi-grams are 'president obama', 'barack obama', 'white house', etc. Recall that in uni-gram model, the most frequent words are 'rt', 'i', 'the', 'to', 'you', etc. in order. Hence, bi-gram model seems to filter our meaningless words automatically, and thus can possibly improve our models significantly. Stemming of words could also help, but we did not seek into this direction.

8.2 Different Trends

A different direction is to predict different trends, such as the popularity vote of American Idol contestants. This is a different problem because it is now a binary classification (i.e. eliminated or stayed). Hence, for the final version, we attempt to implement better learning models and to apply them on at least two different time series data (Obama job approval rate and American Idol contestant popularity).

References

[1] Bryan R. Routledge Noah A. Smith Brendan O'Connor, Ramnath Balasubramanyan. From tweets to polls: Linking text sentiment to public opinion time series. AAAI Conference on Weblogs and Social Media, 2010.

[2] Rajan S. Patel Lynnette Brammer Mark S. Smolinski Larry Brilliant Jeremy Ginsberg, Matthew H. Mohebbi. Google flu trends. Nature, 2009.