Get Zipfy With It

Abigail Jacoby
Dept. of Computer Engineering
University of New Mexico
Albuquerque, NM
jacobya@unm.edu

Vanessa Job
Dept. of Computer Science
University of New Mexico
Albuquerque, NM
vjob@unm.edu

Justin Don Thomas
Dept. of Computer Science
University of New Mexico
Albuquerque, NM
jthomas105@unm.edu

Kellin Rumsey
Dept. of Mathematics and Statistics
University of New Mexico
Albuquerque NM, USA
knrumsey@unm.edu

***Index Terms*—zipfian distribution, zipfs law, power law, sentiment, social media, big data, metropolis hastings**

I. Introduction

Social media data is massive in size, and rich in information. Unfortunately, this wealth of information goes largely unharnessed due to many kinds of bias found in this kind of data. One of the more troublesome forms of bias in social media data is that of *voluntary response*. When performing sentiment analysis on a specific topic, we can only have data from people who have chosen themselves for the sample.

As a motivating example, consider the case where we attempt to use social media data to help predict the outcome of an election. There can be no truly random sample of social media users, since we can only examine those users who have offered their opinion on the subject. Compare two users Alf and Betty, and suppose Alf has specified his opinion once through social media while Betty has specified her opinion 10 times. Under certain sampling conditions, we my be 10 times more likely to include Betty's opinion in our analysis compared to Alf, but we would like to represent them equally in our sample.

II. Prior Research

Several researchers have examined Twitter data with the goal of predicting election outcomes. For instance, in [1], DiGrazia, McKelvey, Bollen and Rojas show that the number of Republican candidate name mentions correlates with the Republican vote margin in 406 U.S. congressional elections from 2010. Another example is the paper of Tusmajan, Sprenger, Sandner, and Welp [4] that finds the number of tweets that mention a political party correlates to the results of the 2009 federal elections in Germany.

Gaurav, Srivastava, Kumar, and Miller [2] develop a model that moves beyond merely counting mentions in tweets. They successfully predict the outcomes of three Latin American presidential elections in 2013 by counting the number of times a candidate's name is mentioned but also include tweets mentioning candidates' common nicknames. They also do some additional filtering to exclude irrelevant tweets.

But as Gayo-Avello, Mextaxas, and Mustafaraj point out in [3], if would be very surprising if Twitter activity predicted the outcome of elections ”given the difference between the demographics of likely voters and social media users.” They analyze data from several elections and conclude that election predictions based on the Twitter metrics described in prior papers are no better than random chance.

Although there has been considerable research on this topic, we believe that we are the first to focus on the problem of voluntary response bias.

III. The Statistical Model

Readers may wish to consult the appendix for details on the distributions discussed in this section.

Consider an arbitrarily large population of *users*, where each user U_i has *sentiment* on a topic. We denote the sentiment S_i and assume the following.

$$S_i \sim Bernoulli(\gamma) \tag{1}$$

We are restricting S_i to be binary, but the ideas presented in this paper can be easily extended to handle multiple categories. In essence, γ is the true proportion of users with *positive* sentiment for the topic. In a political election for example, this could represent the proportion of users that support Candidate A, and it is usually our goal to predict this value. At first glance, the maximum likelihood estimator given in equation (2) is a reasonable choice, and this is the *classical* estimator used in previous research for this topic.

$$\hat{\Gamma}_c = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{2}$$

We will show that this estimator can be badly biased if we attempt to model the Voluntary Response Bias. If we look at a users most recent M *tokens*[1], we realize that some number of these tokens will be topic-related. We denote this number K_i and refer to it as a users *passion*. For the majority of topics, users with low passion will be far more frequent than users with high passion. Thus in a random variable sense, K_i can be modeled with some sort of decay distribution. We have found that for this type of data, the decay is usually of the form $P(K = k) \propto (k+1)^{-\theta 2}$. If this is the case, then we say the data behaves according to *Zipf's Law*, and model the K_i with a *Zipfian Distribution*. For other problems, other decay distributions such as *exponential decay* may be more

[1]We use token in place of platform specific terms such as ”post” or ”tweet”
[2]We use a slightly modified version of the Zipfian Distribution which has support on the set $\{0, 1, 2, ...M\}$ rather than the traditional $\{1, 2, ...M\}$

appropriate, in which case a *truncated geometric* distribution can be used to model passion.

Intuitively, there may be situations where there is a relationship between passion and sentiment. For example, a supporter of a particular candidate may provide us with a handful of positive tokens, but somebody who opposes the same candidate may produce negative tokens far more frequently. Hence we can generalize the model in (1), by allowing γ to be an unknown function of K_i.

$$K_i \sim Zipfian(M, \theta) \tag{3}$$

$$S_i|K_i \sim Bernoulli(\gamma(K_i) \tag{4}$$

$$S_i \sim Bernoulli(\Gamma)) \tag{5}$$

Marginally, the S_i will still be Bernoulli random variables, where the parameter of interest is now Γ.

$$\Gamma = E[\gamma(K_i)] = \sum_{k=0}^{M} \gamma(k)P(K_i = k) \tag{6}$$

Hence the classical estimator given in (2) does not change, and if we were able to sample from these S_i then we would indeed have an unbiased estimator. The problem lies in the fact that we are unable to draw from this distribution. In general, we stream tokens and process each one quickly, storing only those tokens which are related to the topic. Under this sampling scheme it is clear that we have introduced voluntary response bias, since users with high passion are far more likely to be sampled and users with $K_i = 0$ cannot be sampled at all.

We can show however, that the distribution of passion under this scheme is now proportional to $P(K = k) \propto k(k+1)^{-\theta}$. We will call this an *Inflated-Zipfian Distribution*, and note that a similar distribution can be obtained for the Truncated Geometric case. Following the sampling scheme described above, the following model is far more appropriate.

$$\mathcal{K}_i \sim Inflated\text{-}Zipfian(M, \theta) \tag{7}$$

$$\mathcal{S}_i|\mathcal{K}_i \sim Bernoulli(\gamma(\mathcal{K}_i)) \tag{8}$$

$$\mathcal{S}_i \sim Bernoulli(G) \tag{9}$$

Unless $\gamma(\mathcal{K}_i)$ is a constant function, the desired proportion Γ will not be the same as G, hence the classical estimator Γ_c will be unbiased for the wrong parameter. To remedy this problem, let us return to equation (6). We skip some details here, but Γ can be split into two parts yielding a weighted average.

$$\Gamma = \omega\gamma(0) + (1 - \omega)\Gamma^* \tag{10}$$

The weighting term ω depends exclusively on the decay distribution and its parameters. If we assume Zipf's law holds, then $\omega = 1/H(M+1, \theta)$ where $H(n, s)$ is the n^{th} *generalized harmonic number*.[3]. The parameter M is known, and the parameter θ can be estimated using Bayesian Inference with the Metropolis-Hastings algorithm. This implies that a reasonable estimate for ω can be obtained.

[3]If we assume exponential decay, then $\omega = (1 - \theta)/(1 - \theta^{M+1})$

More importantly, by construction we have $\Gamma^* = E[\gamma(K_i)|K_i > 0]$. Hence Γ^* depends only on information which is attainable, and indeed we are able to provide an unbiased estimator for this parameter.

$$\hat{\Gamma}^* = \frac{\sum_{i=1}^{n} \mathcal{S}_i \mathcal{K}_i^{-1}}{\sum_{i=1}^{n} \mathcal{K}_i^{-1}} \tag{11}$$

The intuition behind 11, is that we are weighting each users "vote" by it's inverse passion. This enforces that a user who has produced 20 tokens gets only a twentieth of the weight as a user who has produced a single token which counteracts the fact that we were 20 times more likely to include the first user in the sample to begin with.

Finally we consider the appearance of $\gamma(0)$ in equation (10). This can be interpreted as the proportion of users with positive sentiment given that they haven't produced a single token on the topic. It should be evident that this value will be difficult or impossible to estimate. In a moment, we will offer a possible approach if an estimate is desired, although we must realize that in many situations we cannot hope to estimate this value reliably. Fortunately, in application spaces such as product sentiment, we may decide that users with $\mathcal{K}_i = 0$ are irrelevant and in these cases the goal is simply to estimate Γ^*. Therefore equation (11) is sufficient to provide adequate analysis here.

If we deem it necessary to estimate $\gamma(0)$, we can offer a reasonable approach. First we estimate $\hat{\gamma}(k)$ for $k = 1, 2, ...M$ and then we can preform regression in an attempt to estimate $\hat{\gamma}(0)$.[4] Equations (12) and (13) describe a method for estimating these values.

$$\mathcal{I}_k = \{i; \mathcal{K}_i = k\} \tag{12}$$

$$\hat{\gamma}(k) = \frac{\sum_{i \in \mathcal{I}_k} \mathcal{S}_i}{|\mathcal{I}_k|} \tag{13}$$

In summary, we have broken Γ into two terms yielding three quantities to estimate. Both ω and Γ^* can be estimated reliably from data. Our ability to estimate $\gamma(0)$ is heavily restricted, limiting us to applications where Γ^* is the parameter of interest or cases where the relationship between $\gamma(\mathcal{K}_i)$ and $\mathcal{K}_i$ is apparent.

In conclusion, we propose the following *Inverse Passion Adjusted* (IPA) estimator for Γ, with $\hat{\omega}$, $\hat{\gamma}(0)$ and $\hat{\Gamma}^*$ as they are described above.

$$\hat{\Gamma} = \hat{\omega}\hat{\gamma}(0) + (1 - \hat{\omega})\hat{\Gamma}^* \tag{14}$$

IV. Simulation Study

To illustrate the method laid out in the previous section, we simulated a population of 255,000 users, with passion values following Zipf's Law with $M = 20$ and $\theta = 1.7$. We also defined a linear relationship[5] between passion and sentiment according to (15) for $k = 1, 2, ...20$. In the end, we obtained approximately 1 million topic related tokens with

[4]It is recommend that a wide range of regression techniques such as local, weighted, non-linear or combinations of the aforementioned be considered.

[5]Piecewise and quadratic $\gamma(k)$ functions were also simulated, providing similar results

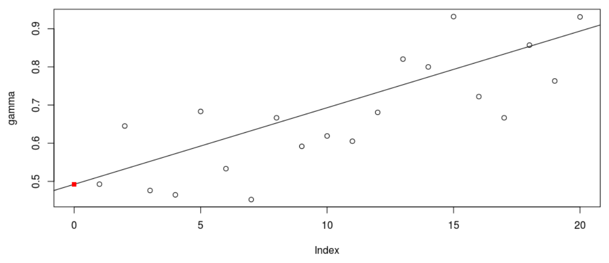
which to do the analysis. Using a process which is reminiscent of Bootstrapping, we select a large subset of this data and compute $\hat{\Gamma}_c$, $\hat{\Gamma}^*$ and the IPA $\hat{\Gamma}$. As in Monte Carlo simulation, we repeat this process many times and construct sampling distributions for each statistic.

$$\gamma(k) = 0.5 + \frac{0.4}{20}k \tag{15}$$

Using Metropolis-Hastings algorithm based on the log-posterior of the Inflated-Zipfian Distribution with a gamma prior on θ, we are able to accurately infer the correct value of θ. This implies that $\hat{\omega}$ will also be a reasonable estimate.

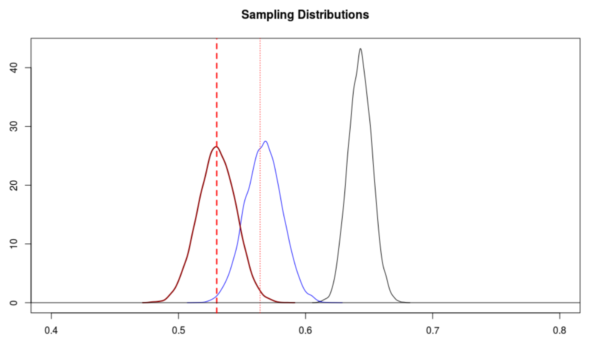
As discussed in the previous section, we can also be confident that we are correctly estimating Γ^*. The difficulty arises when we attempt to estimate $\gamma(0)$. For the purposes of our simulation study, we were able to estimate this value reasonably well with simple linear regression. Figure 1 demonstrates the process of estimating γ for a single Monte Carlo sample. Figure 1 shows that we are estimating well if we

Fig. 1: Estimating $\hat{\gamma}(0)$



recall the true value $\gamma(0) = 0.50$ according to (15).

Fig. 2: Sampling Distributions



The sampling distributions in Figure 2 are enlightening. The bold dashed line represents Γ, and the light dotted line represents Γ^*. Indeed, the classical estimator $\hat{\Gamma}_c$ shown in black is badly biased for both parameters of interest. The estimator $\hat{\Gamma}^*$ shown in blue is unbiased for it's target, and in this particular case the IPA shown in red is unbiased for the primary parameter of interest Γ. We must remember however that this estimator is sensitive to our estimate $\hat{\gamma}(0)$ which may be hard to obtain in practice.

Finally, we note that our estimators do appear to have a larger variance than the classical estimator, but the elimination of the voluntary response bias heavily outweighs this disadvantage.

Before demonstrating this method on a real data set, we will devote two sections to the technical details behind data collection, and sentiment classification of tweets.

V. Data Collection

The goal of our data collection was to collect timelines from a sampling of Twitter users. We were looking for the the last 20 tweets in the timelines of these users from inside our target data collection range. The range we were interested in was between August 1st 2016 and November 7th 2016. There were two phases of timeline collection. In the first, we collected timelines of random users to analyze the frequency of political tweets, and in the second we collected the timelines of people with political tweets, hoping to weight the relevance of their tweets with information derived from the first phase. This section will explain how the data was collected and stored for processing as well as the problems we ran into collecting timeline data from twitter.

A. Collection Process

The data collection was performed in five steps:

1) Collect User IDs.
2) Collect User Timelines.
3) Trim Timelines.
4) Collect More Timelines.
5) Collection Output.

These tasks were completed using Python 2.7 with Twitter interactions handled using Twython [10], a Python wrapper for the Twitter API [9]. Collection was handled through a Python module that was loaded in interactive mode. Our 'twythonInterface.py' contained methods that the user could call to perform the steps of the data collection and formatting with parameters to be set by the user as required. This interface was used to call the both the Twython function implementations and serve as a front end for the file manipulation on the scraped data. The interface modules can be found at the Github repository, https://github.com/cannoness/CS567-Big-Data-Project, under the folder 'twythonInterface'. The individual steps are described in detail below.

1) Collect User IDs: The process for the two phases of data collection differed in how we collected user ids.

1) For the first phase, we streamed all tweets from a region without filter to get a collection of ids from the general twitter population. We used a bare bones extension of the TwythonStreamer running on its own thread to listen to twitter and dump user ids into a deque. Another thread was used to collect those ids from the other end of the deque and write them to an output file. This was handled through our interface with the streamIDsTo method, which allows the user to define an output file name and path as well as a bounding box string in the format defined by the Twitter API.

2) We got user ids for the second phase from a database of political tweets. Since the database of tweets was from within our target range, we also logged the tweet id with the user id to have a starting point for grabbing from their timeline.

In both cases, this collection of user ids was reduced to a list of unique users.

2) Collect User Timelines: From our list of ids we proceeded to use the API's `get_user_timeline()` query to get each user?s last 100 tweets. The Twitter API sends each timeline in JSON format, as a list of individual tweets. From these tweets we saved fields that we would need for further processing as a significantly smaller JSON file. These fields were:

1) User ID string.
2) Text, stripped of non alphanumeric characters.
3) Date as an integer.
4) Tweet id number.

From each timeline returned from the Twitter API we would wind up with a list of these truncated tweets. Each file would contain a list of approximately 300 user timeline lists. This was all handled through the interface's `runTimelineGrabber` method. This method takes a user ID file name, a starting point in file and then calls Twitter's `get_user_timeline` every 15 minutes. It then logs the results.

3) Trim Timelines: From the collected timelines we would try to reduce each timeline to twenty tweets inside our target time frame. Each tweet was checked to see if it fell within our target time frame, from the newest to the oldest. If it was too new, it would be dropped from the list. Tweets with a valid date would be retained in the list until we had collected twenty valid tweets, or until a tweet that was too old was found. If a tweet was too old, it would be ignored and not be logged for follow up. If the total number of tweets saved for each timeline was smaller than 20, and an old tweet was not discovered, the user id and id of the oldest tweet would be logged for follow up in the next step. This is handled with our trim tweets method, which generates a list of filenames to trim based on a user supplied range and generates a follow up list as well as the output timeline files.

4) Collect More Timelines: The log of user ids and oldest tweets would be used to step back through the timeline of each user that came up short, searching for 20 that were within our target timeframe. The 'Collect User Timeline' and 'Trim Timeline' steps would be repeated using the logged user ids and tweet ids. It is necessary to step back through the timeline to search for valid tweets because the API does not provide a way to search a timeline based on date. These steps would be repeated as long as we had users that had too few tweets in the target time frame.

5) Collection Output: Once the target number was found, the files would be output as a .csv consisting of the fields `'user id'`, `'date'`, and `'text'`. The .csv would be constructed user by user, tweet by tweet from newest to oldest, so that each user's timeline would appear as consecutive lines in the file. This is done with our `timelinesToCsv` method which uses the Python csv module to create the output. Once in .csv format, the tweets were then sent on for semantic analysis.

B. Problems Encountered With Collection

We encountered three main problems collecting data with the Twitter API:

1) The Twitter API limit of 300 queries of 100 tweets every 15 minutes artificially slowed down our ability to collect timeline information from thousands of users. ?
2) The Twitter API does not provide a convenient way to collect tweets from a timeline in a certain date range. This means we would have to spend queries trying to find where in the timeline our target range of dates began and ended.?
3) The JSON format is not very useful for big data because json libraries require that the entire file is loaded into memory to parse, access, and manipulate it.?

Problem 3 can be worked around by using a database to store individual tweets, rather than storing groups of timelines as large files, allowing access to individual tweets without loading an entire file into memory. This workaround would have the added benefit of not having to convert the JSON format into .csv for use with the hdfs. Instead, a .csv could be constructed from the database entries, or the semantic analysis could be run off of the database itself. The artificial limitations from problems 1 and 2 suggest the Twitter API is not suited to analyzing large numbers of individual user?s behavior over time. Problem 1 slows down the collection and makes it the single largest bottleneck in the process. Problem 2 prevents us from efficiently collecting tweets from inside the selected time frame, because time is wasted trying to find a starting point to collect valid tweets. If we could specify a time period of a user's timeline to grab, we could simply collect 20 tweets that we could use and forgo grabbing large numbers and backtracking through the timeline. The Twitter API limits put the data collection for this project well within the capabilities of even a modest computer, and would make it very difficult to perform real time analysis of the data. The 15 minute delay between grabbing sets of 300 timelines quickly adds up when we are trying to get data from more than a few thousand users. This type of analysis could still be a valid big data problem with access to large archives of user timelines.

VI. Tweet Context Analysis

To accurately apply the Zipfian distribution to Twitter feed data, the data we gathered through the Twitter API has to be fed through a combination of classifiers and combiners. Our classifiers are divided into categories two categories Classifier C1 calculates each user's ”passion” for politics. Classifier C2 calculates the sentiment of the user's political tweets.

A. Classifier C1

Classifier C1 involves calculation of a user's ”passion” for politics, i.e., how often they mention something political in their tweets. A portion of their political tweets are fed to a classification algorithm algorithm implemented in PySpark. These results are saved by userid (UID) and combined with a simple Python script.

C1 separates tweets by political and non-political context. The classification of political content is a simple string matcher that identifies terms that are deemed to be political in 99% of the tweets in which they appear. Those tweets which happen to be about Hillary or Trump (via any of their matching nicknames, scandals, or hashtags) all pulled out and saved in a comma separated value (.csv) file along with their UIDs.

The algorithm follows the general pseudo-code below.

```
For each tweet:
 If same as previous user:
   For each word in tweet:
     If word appears in context_words*:
       political_count+=1
       output user and tweet
       tweet_count+=1
 Else:
   Save ratio, start new user
```

This classifier produces two separate .csv files. In the first, each line consists of a UID and the user's passion ratio, (political tweet count / total tweets). The second file consists of each tweet that mentions Hillary or Trump, along with the UID of the person who made the tweet.

B. Classifier C2

Context analysis of the tweets deemed political by classifier C1 involves a multi-step process.

First, a selection of approximately 7000 tweets from New York City were randomly selected for political content. Of those, approximately 233 were read and then hand labeled as being in favor of Hillary Clinton (1.0) or in favor of Donald Trump (0.0). Each tweet and its classification were written to a .csv file. This .csv was upload on to a Hortonworks Virtual Machine [8] for the testing phase, along with the .csv listing users and their individual tweets.

1) C2: Training Step: The next step is the training step. The method for training, Term Frequency - Inverse Document Frequency (TF-IDF) is similar to the method search engines use to determine a web page's relevancy to particular search terms. The method applies the following algorithm:

$$TF = \frac{f_{i,j}}{\sum_k f_{k,j}} \tag{16}$$

$$IDF = log \frac{|D|}{|(d : t_i \in d)|}$$

$$TF - IDF = log(f + 1) \times log(D/df)$$

Here TF is *term frequency*, IDF is *inverse document frequency*, $TF - IDF$ is the Spark TF-IDF and

$$\begin{aligned} f_{i,j} &= \textit{frequency of the word } j \textit{ in the corpus } i \\ D &= \textit{the number of documents. (In our case,} \\ &\quad\ \textit{documents are tweets.)} \\ df &= \textit{the number documents (tweets) containing} \\ &\quad\ \textit{the word being looked at in the tweet.} \end{aligned}$$

This is done simply in Python by first parallelizing on a Spark context and then passing the labeled set to the TF-IDF trainer [5]. Given the format of our labels and that we wished to have our labels be the categorization, we split the data up into a Labeled Point RDD and passed this to the `NaiveBayes` modeler in Spark as a training set. We also only allowed terms appearing more than twice to be considered in classification. Naive Bayes classifies the term frequency by applying Bayes Prediction. Naive Bayes is a probabilistic model used in statistics that find the probably the word c will appear in a document given that word x has already appeared in the document. Note, this is not the best text classification algorithm [6], but we used it for simplicity and times sake. The equation for Naive Bayes is

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{17}$$

where

$$\begin{aligned} P(c|x) &= \textit{posterior probability} \\ P(x|c) &= \textit{likehood the term occurs} \\ P(c) &= \textit{prior probability of the term} \\ P(x) &= \textit{the term frequency prior probability} \end{aligned}$$

Applying this in Spark is as simple as passing the transformed TF-IDF set to the `naivebayes` function as a training set. The Spark versions of Naive Bayes is by default a multinomial naive Bayes modeler. It is likely our classification would be more accurate if we used a binomial rather than multinomial model.

The next step was to check the accuracy of our prediction model by running the same testing set through the model predictor and comparing the predicted sentiment to the sentiment labels applied to the training data set. We received only 2 errors in our set of 233 tweets, which is roughly 99% accuracy. Training sets generally do need to be much larger and we understand that this was a limitation of our model. With a larger training set, we expect to have more accuracy.

2) Testing Step: With the training model set up in Spark, the testing data is uploaded into a Labeled Point RDD with the labels now being the UIDs of the individual users. We zip the function into the model predictor using the UIDs as the 'actual" label in this case. Zipping the labels using a lambda function prior to passing the information through the modeller is extremely important as there is no order in an RDD inside of Spark. This is the only way to keep track of which sentiment is being calculated.

Finally, this is published into a .csv file, which is done by copying the entire collection using CSV reader and saving as a file to Hadoop's distributed file system, (`dfs`). This file will be subjected to another step to prepare the cleaned data with sentiment for the final analysis.

3) Final Document Cleansing: In order to do analysis on the data, the sentiment files need to be combined with their respective passions. A final pass through the data is done using the Python data analysis library Pandas [7] to zip the two .csv files together on the UIDs so that we get out the average number of political tweets out of each users last N tweets, along with the average sentiment of each tweet. We used two methods here. First, we analyzed only the first tweet we saw for its political sentiment. Then we looked at all of each user's tweets and analyzed the sentiment of each tweet they made, took the average of the sentiments, and rounded to one digit. So for example, if a user's average sentiment was 0.63, it was rounded to 1 and they were they were declared a Hillary supporter. If their average sentiment was 0.32, this rounded to 0 they were declared a Trump supporter.

VII. Application

In this section, we repeat the method used in the simulation study on the 2016 political election data set. Let us list the following assumptions made by our statistical model in (7)-(9).

1) Each user has at least M tokens during time-frame T.
2) Sentiment is binary.
3) For a given user, sentiment remains constant over all of their topic related tokens.
4) We make no classification error.

The first assumption is not particularly troublesome, since we can choose M such that the majority of users have at least that many tokens[6]

The second and third assumptions may be somewhat violated here, but we have discussed our solution in the previous sections. The final assumption is the most troublesome, and the assumption which is most heavily violated. The classification error can be modeled through the use of Binomial random variables, but this complicates things immensely and we decided not to attempt this here.

The assumptions which were obviously met in the simulation study are possibly violated in our data set. So we emphasize that this discussion focuses on *how* to apply the method and find the IPA.

A. Preliminary Findings

Early on, we streamed tweets from New York City keeping not just the tweets which were flagged as political, but all tweets during our streaming window. The sample included approximately 7,000 unique users and for each user we pulled their timelines and classified each tweet as political or non-political according to C1. In practice, this is a monumental waste of resources since over half of the data we collect is completely unusable, but this allowed us to sample from the original decay distribution rather than the inflated-distribution in order to check our Zipf's Las assumption. Figures 3 and 4 show histograms of the passion with fits based on the posterior mean estimates for θ for Zipfian and exponential decay respectively.

Fig. 3: Zipfian Fit to Twitter Data

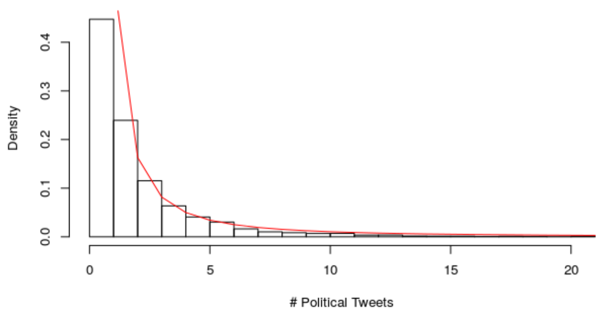
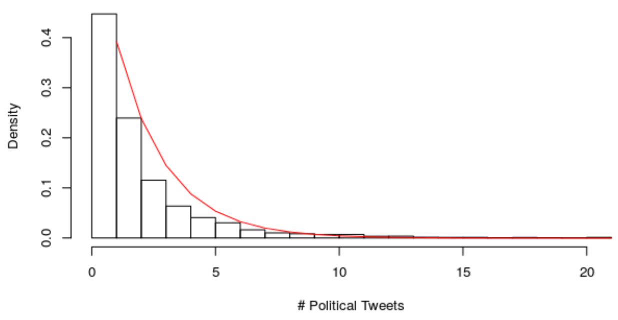


Fig. 4: Exponential Fit to Twitter Data



While neither fit is perfect, it is apparent that the Zipfian distribution appears to fit the data better than the Truncated Geometric distribution, a result that is backed by a simple χ^2 test. Thus the Zipf's Law assumption has been validated.

B. Constructing the IPA

To construct our final data set, we sampled tweets from the state of Colorado in the usual way by streaming tweets and keeping only those which were flagged by C1 as political. Using the methods described in earlier sections, we obtain passion and sentiment values for each user in the sample. The model now dictates that the passion values have been drawn from an inflated decay distribution. Figures 5 and 6 provide histograms and the fits based on posterior mean estimates for θ for the Inflated-Zipfian and Inflated-Truncated-Geometric distributions respectively.

This time it is even more apparent that Zipfian decay is far more appropriate than exponential. The Metropolis Hastings algorithm provides a posterior mean of $\hat{\theta} = 2.744$ here. The fit provided in Figure 5 is very encouraging, and provides evidence that the model is appropriate. Hence we estimate ω according to equation (18)

[6]Pull a small pilot sample of users, and let λ the average number of tokens produced during T. Compute M such that $\Gamma_\lambda(M+1)/M! \leq \alpha$. Where $\Gamma_b(x)$ is the incomplete gamma function. This will ensure that approximately $(1-\alpha)\times 100\%$ of the users will have at least M tokens during T.

Fig. 5: Inflated-Zipfian Fit

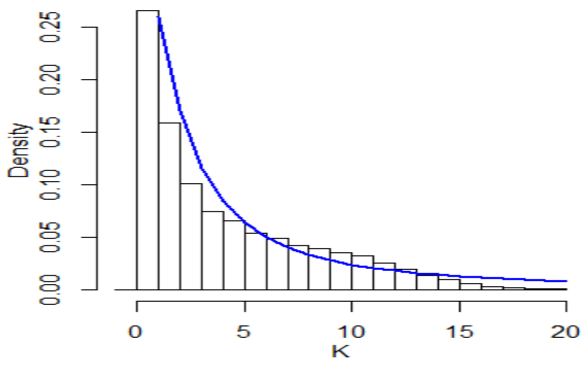


Fig. 6: Inflated-Truncated Geometric Fit

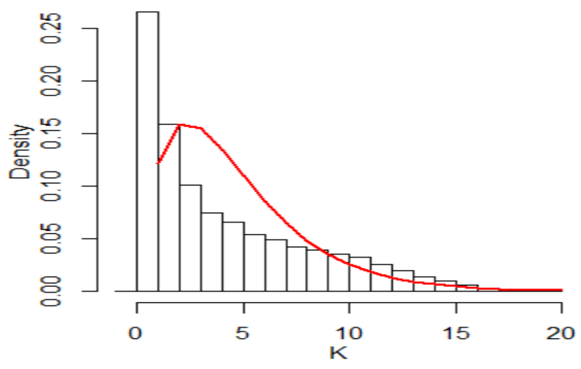


Fig. 7: Comparison of Weights in Local Regression

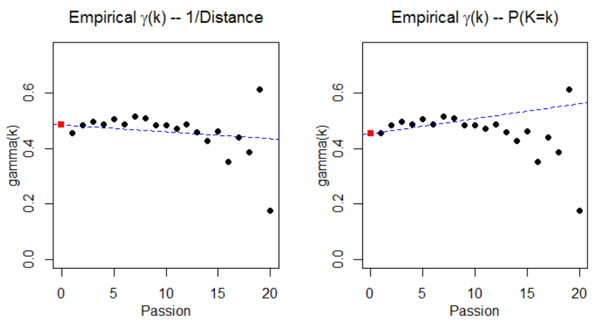
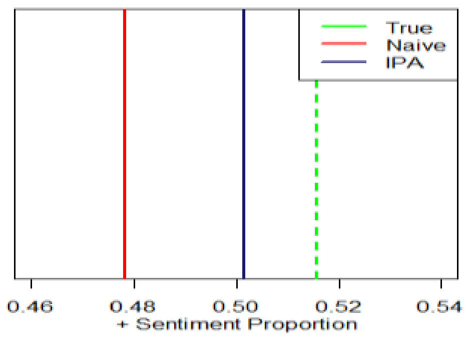


Fig. 8: Comparison of Estimators



$$\hat{\omega} = \frac{1}{H(M+1,\hat{\theta})} = \frac{1}{H(21, 2.744)} = 0.79 \qquad (18)$$

Computing $\hat{\Gamma}^*$ as in (11) is straightforward. As previously discussed, finding $\hat{\gamma}(0)$ will be much more difficult if not impossible. For one, the method proposed in section III is heavily dependent on our chosen regression method. After plotting the estimates in (13) vs k, no pattern was obvious. We decided to use weighted linear regression. Figure 7 demonstrate the sensitivity of $\hat{\gamma}(0)$ to the choice of weights. The left panel uses inverse distance weights, and the right panel uses weights based on the Zipfian probability mass function, and they yield immensely different results.

In the end, we settled on using the square root of the inverse distance as a weight vector, but this choice was somewhat arbitrary. Now that we have estimated the three parts of the IPA, we can construct it for the data. Figure 8 illustrates a "true" Γ, and the classical estimator $\hat{\Gamma}_c$ from (2) and the IPA $\hat{\Gamma}$ from (14).

From Figure 8, the results are encouraging. While the IPA has not eliminated the voluntary response bias, it has reduced it. However, we must acknowledge two very important caveats to the success of this method. First, the "true" value of Γ is quite possibly not the same as the target value of the parameter we have been discussing. The value is 0.5158, and was calculated as Clinton votes over the sum of Clinton and Trump votes. This metric is so that we ignore third party votes in accordance to assumption 2. Technically, our estimator is just predicting the average sentiment of twitter users in Colorado. This parameter may not correspond to the dashed line in Figure 8.

Secondly, we must stress that this method is heavily dependent on $\hat{\gamma}(0)$ which can be seen in part from the calculation in (18). Using a different weight vector in the weighted regression can provide different results for the IPA.

VIII. Conclusion

In summary, we have shown that the passion of twitter users behave according to Zipf's Law, and that sampling in the usual way produces users with passion values that follow an Inflated-Zipfian distribution. We have also shown that ignoring this information can lead to a badly biased estimate if there is a relationship between passion and sentiment.

We used a simulation study as a proof of concept, to show that we can produce unbiased estimators of both Γ and Γ^* if certain, albeit strict, assumptions are met. The application to real data showed that the model is applicable, although there are several limitations that arise with applying the method.

Primarily, we need a bigger data set and more accurate classifiers. In theory, the method should be easily scalable to bigger data. The cost of applying our method is just the cost of the usual method multiplied by M, and M need not be large for the method to work. In practice however, we found that limitations on Twitter prevented us from efficiently observing a users passion.

On the classification side, the binary sentiment assumption was too restrictive for this particular problem. Given time, the classifiers may improve, but choosing a problem better suited for the method may allow the classifiers to improve considerably with little work.

We believe that there is plenty of room for future work. This includes applying the problem to different application spaces, improving the classifiers and accounting for the inevitable error in the statistical model using Binomial distributions.

IX. Appendix

A. *The Bernoulli Distribution*

$$P(X=x)=\begin{cases} p & x=1 \\ 1-p & x=0 \\ 0 & \text{else} \end{cases}$$

$$p \in (0,1)$$

B. *The Zipfian Distribution*

$$P(X=x)=\frac{(x+1)^{-\theta}}{H(M+1,\theta)}$$

$$\theta > 0,\ x=0,1,2,...M$$

$$H(n,s)=\sum_{k=1}^{n} k^{-s}$$

When $s>1$, we also have the following,

$$\lim_{n\to\infty} H(n,s)=\zeta(s)$$

where $\zeta(s)$ is the *Riemann zeta function*

C. *The Truncated Geometric Distribution*

$$P(X=x)=\frac{\theta^{-k}}{C(M,\theta)}$$

$$\theta > 0,\ x=0,1,2,...M$$

$$C(n,s)=\sum_{k=1}^{n} s^{-k}$$

When $s>1$, we also have the following,

$$\lim_{n\to\infty} C(n,s)=\theta/(\theta-1)$$

D. *The Inflated-Zipfian Distribution*

$$P(X=x)=\frac{x(x+1)^{-\theta}}{H^*(M,\theta)}$$

$$\theta > 0,\ x=1,2,...M$$

$$H^*(n,s)=\sum_{k=1}^{n} k(k+1)^{-s}$$

E. *The Inflated-Truncated Geometric Distribution*

$$P(X=x)=\frac{x\theta^{-x}}{C^*(M,\theta)}$$

$$\theta > 0,\ x=1,2,...M$$

$$C^*(n,s)=\sum_{k=1}^{n} ks^{-k}$$

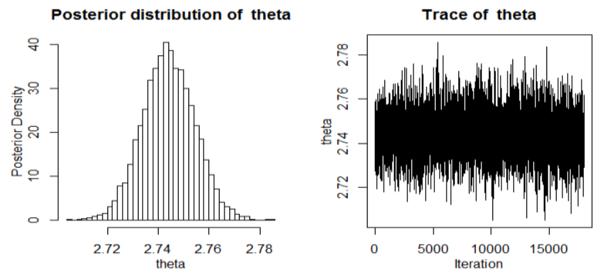
F. *The Gamma Distribution*

$$f(x)=\frac{\beta^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{\beta x}$$

$$\alpha,\beta > 0,\ x > 0$$

G. *Metropolis Hastings*

Fig. 9: Metropolis Hastings Results



Acknowledgment

Some of the data analyzed in this paper came from a MongoDB database built by Rudy Martinez. The authors would like to thank Rudy Martinez for the use of his database.

References

[1] J. DiGrazia, K. McKelvey, J. Bollen and F. Rojas. ”More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior,” [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2235423, [Accessed: 18-Sep-2016].

[2] M. Gaurav, A. Srivastava, A. Kumar, S. Miller, August 2013, ”Leveraging Candidate Popularity on Twitter to Predict Election Outcome”, Proc. of the 7th Workshop on Social Network Mining and Analysis, August, 2013.

[3] D. Gayo-Avello, P. Metaxas, and E. Mustafaraj, ”Limits and Predictions Using Social Media Data,” Proc. of the Fifth International AAAI Conference on Weblogs and Social Media, AAAI Press, January 2011, pp. 490-493.

[4] A. Tumasjan, T. Sprenger, P. Sandner, I. Welpe, ”Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment,” Proc. of the Fourth International AAAI Conference on Weblogs and Social Media, Association for the Advancement of Artificial Intelligence, 2010, pp. 178-185

[5] http://spark.apache.org/docs/latest/mllib-naive-bayes.html [Accessed: Dec. 1, 2016]

[6] http://www.slideshare.net/lucasshen73/spark-application-on-ec2-cluster, [Accessed: Dec. 1, 2016]

[7] ”Python Data Analysis Library ”, http://pandas.pydata.org/ , [Accessed: Dec.1, 2016]

[8] ”Get Started Today with Hortonworks Sandbox”, http://hortonworks.com/products/sandbox/, [Accessed: Dec.1, 2016]

[9] https://dev.twitter.com/docs, ”Twitter Developer Documentation”, [Accessed: Dec.1, 2016]

[10] ”Twython”, https://twython.readthedocs.io/en/latest/, [Accessed: Dec.1, 2016]