EECS 545: Project Final Report

Querying Methods for Twitter Using Topic Modeling

Fall 2011

Lei Yang, Yang Liu, Eric Uthoff, Robert Vandermeulen

1.0 Introduction

In the area of information retrieval, where one desires to match a query to relevant documents, a new problem has arisen since the advent of social media. Social media websites, unlike typical webpages, contain user generated content (or documents) that are short in length, many of which are only a few words. Examples of this are Twitter "tweets", Facebook status updates, or Reddit.com comments. Retrieving such short documents creates a problem for standard information retrieval techniques that rely on keyword matching. This is because it is rare that a short document will contain all the words in the query, despite the fact that the document may still be relevant. In this report, we present a machine learning approach to solving this problem and apply it to data from Twitter.

Twitter is a microblogging website where users post public messages, called "tweets", that are limited to 140 characters. Every tweet posted is associated with the user who posted it; tweets are not anonymous. Twitter users tend to post (or "tweet") messages many times a day. Given such a huge number of documents, it is natural to assume that people need a way to filter for important or interesting content. Additionally, information retrieval methods can be used to track trends in topics, or current events (like the protests in Egypt this year).

In this approach, we apply a popular generative model for document content that captures the notion of document "topics" ("sports"," movies", and "video games" could be examples of topics). This "topic modeling" approach is a natural description for documents, which are usually about one or more topics. This method may be more appropriate for short documents than keyword matching, since even short documents are usually topical. In the following, we propose an approach to information retrieval, where the topic of a query is matched to the topic of each tweet in a principled way.

2.0 Description of Data Used

The data adopted in this study was obtained through a Twitter supplied API. It was drawn from a 10% sample of all tweets posted in August 2011 and includes roughly 546 million tweets from 17 million users. Due to the limited computational resources, the data was subsampled to a manageable size. Specifically, we randomly sampled 50,000 users from the set of users who have posted at least 50 tweets during that period, leaving us with roughly 6.2 million tweets as our data set. Each tweet was preprocessed to remove common words (like "the" or "is"), and words contained in at least one tweet for more than half of the users. Moreover, words are stemmed by the standard Krovetz stemmer [4]. (e.g. "apples" is converted to "apple".). After preprocessing, the average length of a tweet is 7.1 words. It is important to note that up to the point we have been using the terms "word" and "topic" in the common sense of the terms,

however in the next in the mathematical section below we will apply a technical definition to these terms which are different in meaning but still associate back to the intuitive understanding of the terms.

3.0 Topic Modeling: Mathematical Description

We employed Latent Dirichlet Allocation (LDA) in our study, which is a generative topic model for text. This model reads in a collection of documents and outputs a topic model. In this project, the corpus was a collection of all post-processed tweets in our data set and each document was a collection of all of the tweets from a single Twitter user. From the corpus, we defined a dictionary which is a one-to-one mapping between the natural numbers from 1 to M to all unique text strings. We define a "word" to be an index into this dictionary. For example, the string "dunk" was mapped to the word 12481. The corpus of text D, made up of documents $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_U\}$, U being the number of users. Each document $\boldsymbol{w}_i = (w_1, w_2, \dots, w_{N_i})$ is a sequence of words from a dictionary where N_i is the number of words in the document, and $w_j \epsilon \{1,2, \dots, M\}$ are words. In the LDA model the number of topics, K, is chosen a-priori. A topic $z \epsilon \{1,2, \dots, K\}$ denotes a probability distribution over the words, $f_z: \{1,2, \dots, M\} \to \mathbb{R}$. For example, supposing the 10th topic can be described as "basketball", the word "dunk" is expected to occur with high probability, or equivalently $f_{10}(12481)$ is large compared to $f_{10}(x)$ for most other words x. Using the variables defined, we can describe a generative model to create documents for a given user.

For our model, a user $u \epsilon \{1,2, \dots, U\}$ can be represented as a probability distribution over the topics, $g_u: \{1,2, \dots, K\} \to \mathbb{R}$. We can think of each document (all tweets by a single user) as generated by a mixture of topics. A document for user u is generated according to the following model:

1. The number of words for the document is chosen, $N \sim Poisson(\xi)$. N.B. the choice of ξ does not influence the parameter estimation in this model.
2. A vector of topics $\boldsymbol{z} = \{z_i\}_{i=1}^{N}$ are drawn as i.i.d. samples from the distribution g_u
3. For each z_i in $\boldsymbol{z}$ we draw a word $w_i$as a sample from the distribution f_{z_i}independently from the other words.

Notice that g_u can be uniquely represented by a 1 by K vector, θ_u, with $\theta_u^i = g_u(i)$. The θ_u values are sampled from a Dirichelet distribution with an unknown parameter $\alpha \epsilon (0, \infty)^K$.
Under this model, given a set of sample documents, we can apply an iterative E-M algorithm to estimate the model parameters $\{f_z\}_{z=1}^{K}$, $\{g_u\}_{u=1}^{U}$ and α in an unsupervised way [1]. These parameters were estimated for our dataset, using an implementation obtained from http://chasen.org/~daiti-m/dist/lda.

3.1 Mathematical Problem Statement

For any information retrieval method, given a query and a set of documents the query algorithm should rank the documents in order of relevance. Furthermore any document can be considered to be "relevant" or "irrelevant" to the query, this determination is ultimately subjective. Here we will introduce a mathematical statement of the problem using a more general notation not relating to other sections. Let $\boldsymbol{D} = \{\boldsymbol{d}_1, \dots, \boldsymbol{d}_H\}$, a set of H documents and $\boldsymbol{Q}$ be a finite subset of all possible queries. Given a retrieval algorithm let γ be the retrieval function, a set function defined such that $\gamma(q, \{\boldsymbol{d}_1, \dots, \boldsymbol{d}_H\}) = \{\boldsymbol{d}_{\boldsymbol{q_1}}, \dots \boldsymbol{d}_{\boldsymbol{q_n}}\}$ where $\{\boldsymbol{d}_{\boldsymbol{q_1}}, \dots \boldsymbol{d}_{\boldsymbol{q_n}}\}$ are the top n highest ranked documents for query $\boldsymbol{q}$. We can now introduce a relevance set function $\rho: \boldsymbol{Q} \times 2^{\boldsymbol{D}} \to \mathbb{N}$ where $\rho(q, D)$ is the number of elements in D that are "relevant" to query q, again this is subjective. It is desired that an algorithm maximizes $\sum_{q \in \boldsymbol{Q}} \rho(q, \boldsymbol{D})$. We will test for the cases $n = 10$ and $n = 20$, the top ten and top twenty results.

4.0 Proposed Algorithms: Description and Analysis

We propose two general algorithms for using the LDA topic model parameters to return query results. A query is simply text containing search terms. In all the algorithms we describe, the query is preprocessed using the same method as we used on the tweets and converted into the associated word vector. In following sections, a query will be represented as $\boldsymbol{q} = [q_1, \dots, q_M]$ where $q_i \epsilon \{1,2, \dots, M\}$, represents a word in the query. Let $\{t_j\}_{j=1}^{T}$ be the set of all tweets in the corpus, where $t_j = [w_1^j \dots w_{N_j}^j]$ is a tweet (and N_j is the number of words in the tweet). Let the function $\phi: \{1, \dots, T\} \to \{1, \dots, U\}$ be the mapping from each tweet to a user who wrote the tweet. The words in both the query and tweet are the words remaining after using the same preprocessing method explained in section 2.0.

Each algorithm is motivated by a different assumption about how a user generates a tweet. Since tweets are shorter than traditional documents, a revision of the original LDA model was applied to accommodate the difference. In the following, we explain two ways to model tweets and queries as individual short documents. Because we cannot model the relevance for our mathematical model, we assume that relevant documents will be topically similar to the query. For each of the algorithms therefore, we develop a mathematical method for comparing the topical similarity between a tweet and a query.

4.1 Algorithm 1: Single Topic Assumption Search (STS)

In our first algorithm, we model tweet words and query words as i.i.d. realizations from a single topic (one topic per tweet or query). For the query, we assume there is a hidden random variable Y that represents the query topic, which is uniformly distributed over $\{1,2, \dots, K\}$. From this we get $P(\boldsymbol{q}, Y = y) = P(Y = y)P(\boldsymbol{q}|Y = y) = \frac{1}{K}\prod_{i=1}^{L} f_y(q_i)$. We assemble these probabilities into a vector $\tilde{\boldsymbol{v}} = [\, P(\boldsymbol{q}, Y = i)]_{i=1}^{K}$. Using this vector and Bayes' rule we construct

the "posterior distribution vector" $= [\, P(Y = i|\boldsymbol{q})]_{i=1}^{K} = \frac{\tilde{\boldsymbol{v}}}{\|\tilde{\boldsymbol{v}}\|_1}$, so $\boldsymbol{v}^{(i)}$ is the probability that the query is about topic i.

Similarly, for the tweets we assume there is a hidden random vector $\boldsymbol{X} = [X_1, \dots, X_T]$ where X_j represents the topic of the jth twee and is drawn from the distribution $g_{\phi(j)}$ independently. The topic of each tweet is drawn from the topic distribution of the user who wrote the tweet. From this we get $P(t_j, X_j = x) = P(X_j = x)P(t_j|X_j = x) = g_{\phi(j)}(x)\prod_{i=1}^{N_j} f_x(w_i^j)$. We assemble these probabilities into a vector $\widetilde{\boldsymbol{\theta}}_j = \left[P(t_j, X_j = i)\right]_{i=1}^{K}$, and using this vector we construct the "posterior distribution vector" $\boldsymbol{\theta}_j = \left[\, P(X_j = i|t_j)\right]_{i=1}^{K} = \frac{\widetilde{\boldsymbol{\theta}}_j}{\|\widetilde{\boldsymbol{\theta}}_j\|_1}$, so $\boldsymbol{\theta}_j^{(i)}$ is equal to the probability that the jth tweet is about topic i. Notice, that the query model is the same as the tweet model except for the query we do not have a user topic distribution, so we assume that the topics are equally likely.

Given tweet t_j and a query $\boldsymbol{q}$ the tweets that are most relevant to the query are those that minimize $\left\|\boldsymbol{\theta}_j - \boldsymbol{v}\right\|_p : \; p \geq 1$. We experimented with $p = 1$ and $p = 2$.

One interpretation of the vector $\boldsymbol{\theta}_j$ is as a "soft" Bayes classifier. If we wanted to label a tweet t as being about one topic x, we could write the Bayes classifier:

$$f(t) = \underset{x\in\{1,\dots,K\}}{\arg\max}\, P(X = x)P(t|X = x).$$

This is identical to $\underset{i\in\{1,\dots,K\}}{\arg\max}\, \widetilde{\boldsymbol{\theta}}_{\boldsymbol{j}}^{(i)}$ where $\widetilde{\boldsymbol{\theta}}_j$ was described above.

4.2 Algorithm 2: Topic Mixture Search (TMS)

In our second algorithm, we model the tweet words and query words as i.i.d. realizations of multiple topics, which are drawn from the tweet or queries' own topic distribution. This is similar to the assumption made by LDA. Recall, in our LDA implementation, a document was defined as all the tweets from a single user and by using an iterative E-M algorithm, we estimated the distribution of words given a topic z, f_z, the distribution of topics given a user u, g_u (or θ_u), and the Dirichlet prior for user-topic distributions α. In order to develop topic distributions for individual tweets and queries, we treat them as a document generated by a "dummy user", according to the LDA model. Holding f_z and α fixed, we use the maximization (M) step of the E-M algorithm to develop the most likely topic distribution for the "dummy user" associated with a tweet or query. Notice that, in this algorithm, we ignore the user topic distributions g_u computed by LDA. Thus, for a given query $\boldsymbol{q}$, we can compute a topic distribution for $\boldsymbol{q}$, which will call $\boldsymbol{d}$, from f_z, α and the words in the query. Likewise, for a given tweet t_j we can compute a topic distribution $\boldsymbol{\tau}_j$ from f_z, α and the words in the tweet.

Like the previous algorithm, given tweet t_j and a query $\boldsymbol{q}$ the tweets that are most relevant to the query are those that minimize $\left\|\boldsymbol{\tau}_j - \boldsymbol{d}\right\|_p : p \geq 1$ (we used $p = 1$ and $p = 2$).

4.3 Qualitative Algorithm Comparison

Based on the formulations above, we give some qualitative analysis about how the two algorithms behave. In the first algorithm, the product $\prod_{i=1}^{N_j} f_x(w_i^j)$ will be small for any topic where any of the words $[w_1^j, \ldots, w_{N_j}^j]$ are not likely to be generated from the topic. This causes the vector $\boldsymbol{\theta}_j$ will tend to have one large (close to one) entry corresponding to the topic where the words have good agreement. On the other hand, in the second algorithm, there is no assumption that the tweet (or query) is about only one topic. In this case, the resulting $\boldsymbol{\tau}_j$ tends to have larger values in entries that correspond to a topic that matches any word in the tweet. To further elucidate, consider the example queries below, and the corresponding distribution vectors from each method. The queries are "Apple Pie" and "Apple Bacon".

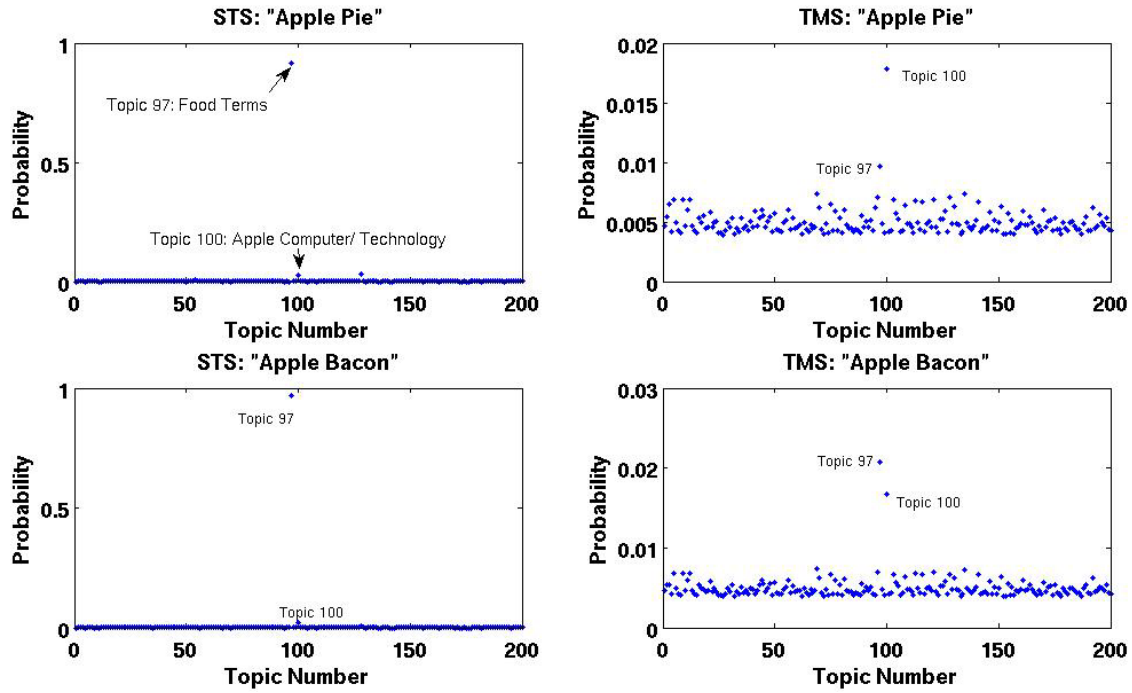

Figure 1: Example Tweet and Topic distributions for STS (left) and TMS (right)

Notice that TMS returns vectors that are less heavily weighted to a single topic. We found that this performed much better in general because it allowed more nuance in the differentiation of topics. For example, if we replace the word "pie" with bacon in the query, STS produced a vector which is very close to the original query. Here, "bacon" and "pie" are considered

topically identical as food terms. TMS on the other hand is able to retain subtle associations with non-food topics. For this reason, we discarded STS as a viable query algorithm, but it may have utility as a single topic classifier.

Through experimentation it was determined that using the 1-norm for the TMS algorithm worked better than the 2-norm and this norm was used in the algorithm evaluation. We suspect that the 1-norm outperforms the 2-norm because the 1-norm emphasizes the subtle topic associations in TMS, i.e. those that are small in probability. This is opposed to the 2-norm which prefers to minimize large differences in individual entries in the vectors. TMS has a strong tendency to rank extremely short tweets (one to two words) highly. To obviate this we added a regularization term to penalize short tweets. Instead of ranking tweets according to those which minimize $\|\boldsymbol{\tau}_j - \boldsymbol{d}\|_p$ we instead rank them to minimize $\|\boldsymbol{\tau}_j - \boldsymbol{d}\|_p + \lambda \frac{1}{L_j}$, where L_j is the number of words in tweet j. Interestingly this regularizer penalizes simplicity rather than complexity. We call this regularized TMS algorithm RTMS.

5.0 Evaluation and Results

In the following section we detail our method of evaluating the RTMS algorithm against more traditional information retrieval methods. In the LDA model, we decided to use K=200 topics. This was the largest number of topics we could feasibly model due to computational constraints. For our RTMS algorithm we set λ= 0.05; this parameter was chosen through experimentation.

5.1 Competitor Algorithms

The problem of querying for relevant documents has been a problem long studied in the field of information retrieval. There are two classic approaches for calculating the numerical relevance between a document and a query: vector space model and language modeling approach. In this section we will introduce two classic algorithms, which correspond to each of the two approaches.

The following notation will be employed for mathematical explanation this section alone:

- d denotes a document, q represents a query, and w indicates a word.
- $c(w, d)$ is the count of word w in the document d.
- $c(w, q)$ is the count of word w in the query q.
- N is the total number of documents in the document collection.
- $df(w)$ is the number of documents that contain the term w.
- $|d|$ is the word length of the document d.
- $\overline{d}$ is the average word length of all documents in the collection.

5.1.1 Pivoted Normalization Method (PNM)

In vector space model, both the documents and queries are organized as a vector of dimension $|V|$, which is the size of the dictionary that records all unique words in the data collection. Under such representation of documents and queries, various methods could be developed to evaluate the distance between these vectors. Such distance between document vectors and query vectors are then employed to rank documents to queries. Among the algorithms belonging to this approach, pivoted normalization method [2] is one of the best performing methods. According to [2], the pivoted normalization method is formulated as the following:

$$\sum_{w \in \{q \cap d\}} \frac{1 + \ln[1 + \ln(c(w,d))]}{(1-s) + s\frac{|d|}{\overline{d}}} \cdot c(w,q) \cdot \ln \frac{N+1}{df(w)}$$

Where $s \epsilon [0,1]$ is a user chosen parameter. In this algorithm, s being small will tend to rate longer documents as more relevant, and s being large will rate short documents as more relevant.

5.1.2 Dirichlet Prior Method (DPM)

Language modeling approach [3] assumes that each document in the collection has a language modeling that assigns a probability to each of the $|V|$ words in the dictionary. Given a query *q*, language modeling approach will rank the document *d* based on the probability that query *q* is generated by the language modeling of document *d*. Dirichlet prior method is one of the best performing language modeling approaches which ranks documents according to the probability that the query is generated from each document's language model. The Dirichlet prior method is formulated as the following:

$$\sum_{w \in \{q \cap d\}} c(w,q) \cdot \ln\left(1 + \frac{c(w,d)}{\mu \cdot p(w|C)}\right) + |q| \cdot \ln \frac{\mu}{|d| + \mu}$$

5.2 Evaluation Methods

To evaluate the performance of our algorithms, we developed a set of test queries. These queries were generated by our group as well as people not involved in the project. Next we decided on criterion for choosing whether query results were relevant to the tweet. For each query we collected the top 20 results from our query algorithm as well as the two competitor query algorithms. Finally we counted the number of tweets which were relevant for each algorithm. We report the number of relevant tweets in the top 10 and top 20 results from each algorithm.

5.3 Summary of Evaluation Results

Table 1. Quantitative Evaluation Results

Query: "Basketball Wives"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	10	9	10
# Relevant from top 20	20	19	19
Criterion: Specifically about TV show "Basketball Wives"			
Query: "NFL Lockout"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	1	9	10
# Relevant from top 20	2	18	20
Criterion: About NFL lockout event			
Query: "Mad Men"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	3	8	10
# Relevant from top 20	3	16	18
Criterion: Referencing the "Mad Men" TV show			
Query: "Fast Food Sick"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	3	2	1
# Relevant from top 20	4	2	4
Criterion: About fast food making a person sick or about fast food being physically repulsive			
Query: "Apple Accessories"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	7	6	10
# Relevant from top 20	7	11	11
Criterion: Referencing an accessory for an Apple product			
Query: "Funny Animal"	**RTMS**	**PNM**	**DPM**
# Relevant from top 10	8	4	7
# Relevant from top 20	8	7	8
Criterion: Any tweet containing a reference to humor and an animal			

5.3 Discussion of Results

From the results, TMS ranged from slightly more effective than classical methods to significantly worse. TMS seemed to work better when the query was vague but topical in nature, for example "funny animal", rather than about a specific thing like "NFL Lockout". We believe that this is likely due to the lack of granularity in the topic model. For example, while the topic model groups words about football together well, it does not separate terms related to the NFL Lockout. We discussed this earlier about our first algorithm in section 4.3.

It is worthwhile to mention the differences in the types of tweets returned by each query in the tests above. For example, for the query "Funny Animal", the TMS algorithm returns the tweet "dogs are funny" as highly ranked. On the other hand, key word search returned: "talking

animals aren't funny, I can only think of a handful of movies where theres humans w/ a talking pet and it was funny -_- ". In the TMS example, it is clear that our algorithm is capable of retrieving a tweet that did not contain both of the search terms by associating "dog" with "animal". However, the key word search returns a longer (and possibly more desirable) tweet. TMS tended in general to pick shorter tweets than keyword search. We believe this is partially due to the fact that short tweets match short queries better in the space of the topic distribution vectors compared.

5.4 Qualitative Discussion of Algorithm and Future Work

The results of the test queries not discussed above reveal several limitations and interesting features that are not captured in the quantitative results. In other experimental queries, we discovered that certain words with double meanings can become associated with the wrong topic, yielding "unexpected" results. For example, the query, "Rotten Tomatoes Harry Potter" was suggested by a peer, referencing the ire over a controversial movie rating given by the website RottenTomatoes.com. The words "rotten" and "tomato" are strongly associated with food, while "Harry" and "Potter" are strongly associated with the movie "Harry Potter". The result was tweets which had food terms and Harry Potter terms. For example two highly relevant tweets were "Voldemort & the chocolate factory" and "Harry Potter and the order of the pizza". This ability of topically matching tweets to a query without common keywords, should be of interest for various applications. One way to leverage this without the topic confusion issue just mentioned is to use more topics in the LDA modeling. One would hope that with enough topics, "Rotten Tomatoes" would become strongly associated with a movie reviews topic. Another possible improvement can be a hybrid method that incorporates topic modeling and keyword matching at same time.

6.0 Summary and Conclusions

In this project, we attempted to solve an information retrieval problem for tweets. We desired an algorithm which ranked tweets by relevancy to a query and thus developed two methods of measuring the similarity between a tweet and a query as a substitute for relevancy. Unlike the keyword search algorithms of previous works, our method represents documents and queries by the topical content of the words in the document rather than the words themselves by using a popular topic modeling technique, LDA. As a result we were able to highly rank tweets which were relevant to a query without containing any of the words in the query. Additionally, we were successful in finding short tweets (tweets that are one or two words in length) that were relevant to the query. Despite these successes, we were not able to create a query system which performed as well as standard keyword search methods by our quantitative measure. One issue was matching queries which were very specific in their meaning, for example

"NFL Lockout", because there was not enough granularity in the topic groupings created by the LDA algorithm (e.g. "movie ratings" is not a different topic from "movies"). Due to computational restrictions, we were only able to run LDA for 200 topics. We suspect that if more topics were modeled this problem would be alleviated. Additionally, it seems that our method fails to find relevant tweets which are longer than the query, and we believe that this is because query and tweet topic distribution vectors are more similar if the tweet and query are of the same length. Though our regularizer in RTMS attempts to counter this issue, it was not generally effective and this remains an unsolved issue.

In this project, we saw that topic modeling could be a powerful tool for both modeling short documents and for associating them with words not contained in the document. However, we learned that it is difficult to translate this ability into an effective query system. Words tend to have many meanings, so it is difficult to avoid strange misinterpretations of the query terms, such as in our "Rotten Tomatoes Harry Potter" example. Additionally, though we want to find relevant documents that do not contain the words of the query, it is intuitive to assume that a user would prefer that results which contain some keywords are ranked highest. There is nothing in our algorithm to enforce this, and if two words are topically identical, it will not prefer the occurrence one word over the other in the ranking. Therefore it may be that a combination of TMS and traditional keyword search methods is required to out-perform current information retrieval methods.

7.0 Individual Work

Eric Uthoff: Developed Matlab code to experiment with algorithms. Worked with group to conceptualize new query algorithms and suggest improvements. Worked significantly on writing documents, due to language considerations.

Robert Vandermulen: Developed mathematical motivation and justification for algorithms described within. Worked with Eric to implement algorithms in Matlab. Worked significantly on writing documents, due to language considerations.

Lei Yang: Gettting Twitter stream data by Twitter API, data preprocessing, implementing classic information retrieval methods, running experiments in retrieving Tweets using classic information retrieval methods. Assisted with writing documents.

Yang Liu: Getting Twitter stream data by Twitter API, data preprocessing, data sampling, query sampling, algorithm discussion. Assisted with writing documents.

References:

[1]. Blei, David M.; Ng, Andrew Y.; Jordan, Michael I (January 2003). Lafferty, John. ed. "Latent Dirichlet allocation". Journal of Machine Learning Research 3 (4–5): pp. 993–1022

[2]. A. Singhal. Modern information retrieval: A brief overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 24(4):35–43, 2001.

[3]. C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In Proceedings of SIGIR'01, pages 334–342, Sept 2001.

[4]. R. Krovetz, 1993: "Viewing morphology as an inference process," in R. Korfhage et al., Proc. 16th ACM SIGIR Conference, Pittsburgh, June 27-July 1, 1993; pp. 191-202.