A time-sensitive user-specific recommendation system for Twitter

Shaunak Chatterjee
Computer Science Division
University of California
Berkeley, CA 94720
shaunakc@cs.berkeley.edu

Mobin Javed
Computer Science Division
University of California
Berkeley, CA 94720
mobin@eecs.berkeley.edu

Anupam Prakash
Computer Science Division
University of California
Berkeley, CA 94720
anupamp@cs.berkeley.edu

Abstract

Most modern analysis of social networks and social media assume a static network. The primary reasons for this are better parameter estimation and simpler inference. However, this assumption often results in very erroneous deductions, especially for local (e.g. user-level analysis). The aim of this project is to explore efficient ways to model the temporal variation of influence on social links in Twitter. Using this temporally sensitive model, we hope to perform better at predicting a user's interest in a currently trending topic.

1 Introduction

Social networks like FaceBook [1] and Twitter [2] are an integral part of our lives today. They have become an indispensable platform to disseminate information and keep in touch with family, friends and the world in general. The ability to predict a user's interest in a particular topic or product has tremendous social and commercial impact. Therefore, it comes as no surprise that people have been analyzing these social networks for a while now, with various applications in mind.

Our focus in this work is on Twitter, an online social networking and microblogging service which was created in 2006. Users send and read **text-based posts** of up to **140 characters** known as ***tweets***. Twitter has gained worldwide popularity and has more than 140 million active users (as of 2012) generating over 340 million tweets daily. It has been described as "the SMS of the internet." Users can "follow" users, whose tweets would then show up on their tweet feed.

Since a user who follows many other users would have a lot of tweets showing up on their twitter feed, there needs to be some way of ranking the tweets and showing *the most relevant ones at the current time.* Hence, we need a recommendation system for tweets which is customized for each user, and is also time sensitive, to capture the user's current interests.

A user's interests change over time and hence it is important to consider a user's long-term as well as short-term preferences. Temporal dynamics were also modeled in the now famous Netflix challenge. Koren [10] introduced temporally sensitive parameters in his collaborative filtering model and outperformed several more complex static models.

2 Problem Definition

In this project, we would like to design a collaborative filtering scheme with temporal dynamics for Twitter users. The grand objective is to be able to predict a user's interest in a particular topic which is currently trending.

The formal problem statement (and this is a step towards the grand objective) for the scope of this project is as follows: Given a user u, a time window t, and a set of tweets $\{TW\}$, rank the set of tweets based on the user u's preferences at time t.

To evaluate our recommendation system, we adopt the following scheme. We take a tweet (say tw^*) that a user u^* has actually tweeted in time window t^* and hence obviously finds interesting, and a sample of 9 other random tweets. We then rank these 10 tweets in accordance with that user's preferences during that time window. Since the other tweets are randomly chosen, it is very likely that they will not be as relevant to the user as tw^*. Hence, the predicted rank of tw^* should be quite high (close to 1). The average rank predicted for tw^* over a set of test tweets, will give us a good indication of the accuracy of our algorithm.

3 Data acquisition

We used the MarkLogic-Hadoop connector to query the MarkLogic server for the tweets, XML parse them, filter by English language and store a condensed version of each tweet in the following format.

⟨ created-at, text, user-id, screen-name, statuses-count, friends-count, followers-count, location ⟩

The input to mappers are ⟨ `Nodepath, MarkLogicNode` ⟩ pairs. The MarkLogic-Hadoop connector provides an Xpath interface to generate the MarkLogicNodes. Figure [3] shows a subset of the settings from our XML configuration file. The *mapreduce.marklogic.input.documentselector* and *mapreduce.marklogic.input.subdocumentexpr* properties are used in conjunction to form the MarkLogicNodes, which in our case are the *status* nodes.

⟨ property⟩ ⟨ name⟩ mapreduce.marklogic.input.namespace⟨ /name⟩ ⟨ value⟩ tw,http://www.bid.berkeley.edu/statuses⟨ /value⟩ ⟨ /property⟩

⟨ property⟩ ⟨ name⟩ mapreduce.marklogic.input.documentselector⟨ /name⟩ ⟨ value⟩ fn:collection()⟨ /value⟩ ⟨ /property⟩

⟨ property⟩ ⟨ name⟩ mapreduce.marklogic.input.subdocumentexpr⟨ /name⟩ ⟨ value⟩ //tw:status⟨ /value⟩ ⟨ /property⟩

⟨ property⟩ ⟨ name⟩ mapreduce.marklogic.input.maxsplitsize⟨ /name⟩ ⟨ value⟩ 50000L⟨ /value⟩ ⟨ /property⟩

The querying was extremely slow and we were able to collect about only 2.5GB using 32 mappers running for about 96 hours. Here are some notes from our experiences:

- We tried to increase the number of mappers to make the querying fast, but the MarkLogic-Hadoop connector doesn't allow for that. It seems to split the data by forests (hence 32 mappers). Changing the value of *mapreduce.marklogic.input.maxsplitsize* didn't change anything.

- We couldn't figure out a way to specify the *documentselector* at a granuarlity lesser than the whole database e.g. fn:collection() and greater than a single document, for e.g. doc("/twitter/2011_11_23_tweet21.xml"). Due to this, we were forced to query the full database.

This dataset has approximately 18 million tweets which is about 2% sample of the total number of tweets (640 million) on the Marklogic server. Since, the Marklogic data itself is a 1% sample, this dataset is a 0.02% sample of the original Twitter datastream, and is hence very sparse. This dataset will henceforth be referred to as the "sparse dataset."

To cater to this sparsity, we picked the top 10,000 users with the highest statuses count, and then further filtered this set down to 5000 users by selecting those with the highest number of followers. Next, we used Twitter REST APIs to get the statuses and network information for these 5000 most popular users. For each user id, we query for it's timeline and friend ids using HTTP GET requests [4],[5]. The timeline returns upto 3200 most recent statuses, but is rate-limited to 200 statuses per query. The queries themselves are rate-limited to 150 an hour per IP address.

To use the network information, we also needed to query for the statuses of the friends of these 5000 popular users. But doing so forms an ever expanding set of users for which we need the statuses. In order to limit the friends whose statuses we query for, we use the following strategy: we generated the masterlist of all the friends of these 5000 users and picked the top 5000 which are friends of multiple users. Our final dataset therefore, has 10,000 users in total with 3200 statuses per user. This dataset will be referred to as the "dense dataset."

In the following sections, we will describe two different recommendation frameworks which we designed and implemented — namely, latent space analysis with LSI projections and session-temporal graphs with LDA.

4 *LSI* model

The training dataset was a 0.02% sample of the twitter data-stream over a three month period, the data was heavily imbalanced with respect to time.

Twitter data is extremely noisy with 40% of it being babble, 38% conversation and 4% spam [6]. Moreover filtering tweets according to language is not sufficient as in some countries people combine the local language (eg: Indonesia) with English for tweets. From the initial dataset of 18 million tweets we filtered out tweets containing hyperlinks and those for which half the words were not in the Webster dictionary. The filtered corpus contained about 4 million tweets.

4.1 Temporal variation in topics

With the sparse data set, we can see the evolution of topics in the twitter verse at a coarse level of granularity. The day is the number of days elapsed since the beginning of the data set, Nov 21st 2011, trending words are those that occur at least $\mu + 3\sigma$ times on the given day where (μ, σ) are the mean and variance for the appearance of the word. The words have been selected to show that coarsely the sparse twitter stream captures important events on these days (see Table 1):

Day	**Trending words**
3	lord cooking grandma xmas thanksgiving breakfast america celebrate food family
40	happy years party friends tonight crazy year mom fun parents family
83	music rihanna artist grammys hop houston
85	loving depressed buying gifts pink cute
99	santorum presidential ron baseball ohio
108	international children women apple cook

Table 1: Trending words capture important events

4.2 Feature extraction

The number of words occurring in the filtered twitter corpus is of the order of millions, however the distribution of words follows a power law and has a heavy tail. Eliminating words that occur less than 10 times in the corpus leaves us with a feature set of size about 80000. Stripping off punctuation apart from @ and # which represent user mentions and hash tags and filtering out stop words, the feature set size further reduces to around 40000.

A topic model like Latent Semantic Indexing (LSI) that relies on matrix factorization requires a small set of relevant features. For topic modeling on twitter, the feature set should have the following properties: (i) High coverage: Words from the feature set should occur frequently in tweets. (ii) High relevance: Words in the feature set should have high correlation with words with high information content i.e. occurrence of a word in the feature set should serve as an indicator that the tweet is topic specific rather than babble/conversational.

We used the following heuristic with $(t_1, t_2) = (100, 50)$ to rank features and selected around 3500 of the top features for training the LSI model:

1. Select a threshold t_1, the weight of a word w occurring k times in the corpus is:

$$wt(w) = \begin{cases} \frac{1}{\max(t,k)} & \text{if } w \text{ in dict} \\ 0 & \text{otherwise} \end{cases}$$

2. The weight of a tweet is the total weight of words contained in it. The score of a word is the average weight of tweets containing the word.

3. Select words sorted by score that occur more than t_2 times in the corpus.

4.3 Latent semantic indexing

The latent semantic indexing LSI model [8] constructs a latent space for tweets by computing the singular value decomposition of the term-document matrix weighted by tf-idf scores. The terms are the feature sets constructed as above while the documents are an aggregation of tweets containing the terms. In addition we added tweets containing popular hashtags and usernames as documents for training the model.

The dimension of the latent space is chosen so that it captures around 0.8 fraction of the Frobenius norm of the tf-idf matrix, about 200 dimensions suffice.

4.4 Ranking using the LSI model

The model was tested on the dense data obtained through the twitter REST api described in section 2. The data was split into training and test sets by aggregating the tweets of users according to day and making an 80:20 split uniformly at random. The test set contains 20% of the tweets by a user on a day.

The prediction task for testing the model was the following: for each test tweet, sample 9 other tweets from different users in the test set and produce a ranking of the tweets specific to the user who posted the test tweet and the time at which the test tweet was posted.

The prediction task is expected to perform well for a reasonable topic model as a few randomly sampled users are likely to be very different in the topic space and should be distinguishable, we used the following algorithm for ranking using the LSI model:

1. Project the test tweet onto the LSI latent space to obtain unit vectors in $\mathbb{R}^k$. ($k = 200$)

2. The training set is stored as a list of unit vectors in the the LSI latent space, the distance between a user and the test tweet is the minimum distance between a user tweet and the test tweet in the LSI space.

3. Rank users according to the distance in step 2.

The performance of the LSI based ranking algorithm on the dense data set for varying values of k is presented in Table 2 in Section 7. An average rank of 4.22 with a standard deviation of 0.088 was obtained.

4.5 Anchor Words

The performance of the ranking algorithm can be improved by considering anchor words. An anchor word is a word unique to the tweets of a given user that occurs on a large enough number of days. Most of the users in the dense data set had a long list of anchor words with high coverage, which would help improve the performance of the ranking algorithm.

The anchor words are mostly @ user mentions, they indicate the strong social connections that the user actively follows over a period of time and are valuable information for a recommendation system.

5 Session Temporal graphs

5.1 Framework

In this section, we describe the basis of our more generic recommendation system. Xiang et al [11] proposed the session temporal graph framework which can combine a user's long-term and short-term preferences. Data in the form of $< user, item, time >$ triples (like we have), can be modeled using a graph as shown in Figure 1(a). Note that the user does not rate the item – the interaction is an implicit indicator of interest. A "user" node represents a user while an "item" node represents the type of object to be recommended (e.g. books, movies). A "session" node is associated with a user and a specific time window. We represent the $< user, item, time >$ triples through $< user, item >$ and $< session, item >$ links by dividing the time into bins and binding the bins with corresponding users. Hence, the session node is a combined node with a user and a specific time bin.

In a conventional time-series model, time is treated as a universal dimension shared by all users. However, in recommendation systems, the authors argue that the time dimension is a local effect that should not be compared cross all users arbitrarily. Correlation of users/items on time is typically not useful while correlation of items on time for a specific user is significant, i.e. items within a user session are somewhat more relevant. For users whose interests fluctuate quickly, the session window will be small and for users with more slowly evolving preferences, the window will be larger.

Formally, a session temporal graph (STG) is a directed bipartite graph $G(U, S, I, E, w)$ where U denotes the set of user nodes, S is the set of session nodes, I the set of item nodes, E denotes the interaction between U or S and I. $w : E \to R$ denotes a non-negative weight function for edges. A user node u connects to all the items the user has interacted with and hence captures her long-term preferences. A session node only connects to the items interacted with during a particular time-window (i.e. session), hence capturing the user's short-term preferences during that time.

5.2 Recommendation on STG

When making recommendations for user u at time t, the user node u and the session node corresponding to $< u, t >$ are injected with user preferences (see Figure 2). Preferences injected into the user node is propagated to the items $N(u)$ that the user interacted with at all times. This then propagates to other unknown items similar to user u's long-term preferences via other users who have interacted with items in $N(u)$. Similarly, preferences injected into the session node will propagate to items $N(u, t)$, which u interacted with during the session containing time t, and it is then propagated to similar items through other users who rated those items. For computational efficiency, we only consider paths of length 3 – all odd-length paths starting from a user or session node, end at an item node. Longer length paths also add more noise. The preference of user u at time t for item i is given by the sum of weights propagated to item i by all paths of length 3.

6 Latent space STG

The STG framework is well-suited for our task because it allows us to design a user-specific time-sensitive recommendation system. However, an important modification is necessary to adapt this to tweets. The STG framework works for **previously seen items**. When working with tweets, a test tweet is most likely going to be distinct from any previously seen tweet. Therefore, we have to rely on similarity of tweets to drive our recommendation algorithm. A popular approach to express similarity between textual entities (like tweets, blogs, news articles) is to project it onto a latent space using a topic model.

Latent Dirichlet allocation (LDA) [7] is a generative topic model, which has been very popular for modeling text corpora. A topic distribution is sampled for each document from a Dirichlet prior. Now, for each word, a topic is sampled from the document-specific topic distribution. Finally, the word is sampled from the topic-specific word distribution. Due to space constraints, we will not delve further into the details of LDA. Once an LDA model has been trained for K topics on a corpus of tweets, given a new tweet (say tw), it outputs a K-dimensional non-negative real vector θ_{tw}, where θ^i_{tw} denotes the expected number of words generated by topic i in tweet tw. Since it denotes an expectation, θ^i_{tw} can be fractional. For our analysis, we consider a normalized version of θ_{tw} ($\sum_i \theta^i_{tw} = 1$). We used David Blei's LDA-C implementation for learning an LDA model on our training dataset.

Our modified STG framework enhances the original STG framework with the projection from the items (tweets) to the latent space (LDA topic space) as shown in Figure 1(b). The edge weights and preference propagation scheme is detailed next.

6.1 Edge weights

The recommendation scheme in the latent space STG is similar to the original STG scheme. Consider the case when we are interested in the preferences of user u at time t. Let $< u, t >$ denote the corresponding session node. Preference weights β and $1 - \beta$ are injected into nodes u

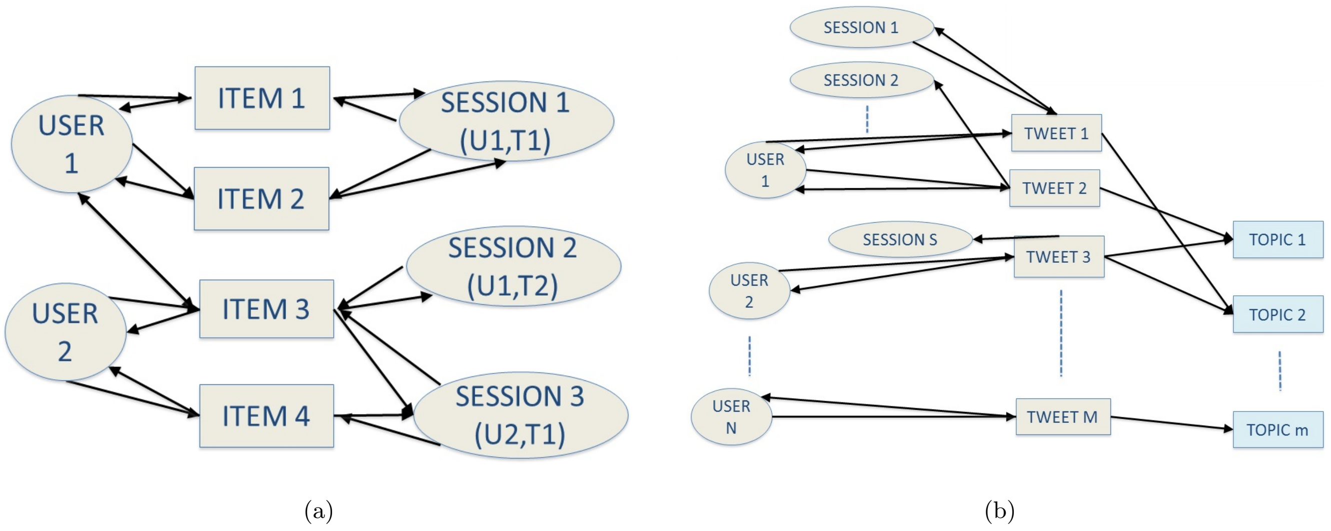

Figure 1: (a) The original STG framework for item recommendation. (b) Our adapted STG framework for tweet recommendation. Since new tweets are most likely distinct from previously seen tweets, we have to project the tweets onto a common latent space to use the information from previous tweets.

and $< u, t >$ respectively. $\beta \in [0, 1]$ is the relative importance of long-term to short-term preference. $\beta = 1$ means that the recommendation will be based solely on long-term preferences of the user, while $\beta = 0$ means only the short-term preferences will be considered.

Let $w_{u,tw}$ be the weight of the edge from user u to tweet tw. The **sum of the outgoing edges** from any node is **1.0**. If a user u has T_u tweets, then $w_{u,tw} = \frac{1}{T_u}$. The session-tweet edge weights are also similarly determined. If $T_{<u,t>}$ is the number of tweets in session $< u, t >$, then $w_{<u,t>,tw} = \frac{1}{T_{<u,t>}}$. The tweet-topic edge weights are the normalized LDA-vectors θ^i_{tw} (as described previously).

The reverse edge probabilities are more computation-intensive (due to the normalization). For the topic-tweet edges, the weight $w_{i,tw} = \theta^i_{tw} / \sum_{tw'} \theta^i_{tw'}$. The tweet-user edge between tweet tw and user u is given by $w_{tw,u} = w_{u,tw} / (\sum_{u'} w_{u',tw} + \sum_{<u',t'>} w_{<u',t'>,tw})$, while the tweet-session edge weight is given by $w_{tw,<u,t>} = w_{<u,t>,tw} / (\sum_{u'} w_{u',tw} + \sum_{<u',t'>} w_{<u',t'>,tw})$.

Finally, the preference propagation works as follows. Initially, some preference is injected into the specific user and session nodes. The incoming preference at any node is distributed among all the outgoing edges, *proportional to the edge weights*. This ensures that if two users or sessions have significant overlap in tweet topics, then the preference propagation would use this information in an analogous way to two users rating the same item (in the original STG setup). Further computational simplification is possible by marginalizing out the tweets, and the simplified model is shown in Figure 2.

The tweet marginalization is based on the observation that once some weight (say w^*) reaches a topic node, its outward propagation into user and session nodes (via tweets) and then back to the topic nodes (again via tweets) happens in the same way irrespective of which user and session we are interested in. Let ϕ_{ij} denote the weight propagated to topic node j (via tweet-user/session-tweet) if unit weight reaches topic node i. Computing *phi* is straightforward and hence the details are skipped. This reduced framework is completely analogous to the original STG framework, with the items replaced by topics.

Upon completion of preference propagation (for paths of length 3 in the STG formulation in Figure 2), the preferences that reach the topics (let us call that vector $\theta_{u,<u,t>}$) reflect the preference vector of user u for session $< u, t >$. Now for a tweet tw^*, the recommendation score of tw^* for user u at time t is $\theta^{\top}_{u,<u,t>} \theta_{tw^*}$. To rank a set of tweets, we compute the recommendation score and rank them in descending order.

6.2 Model extensions

The STG framework can also be extended to account for user interactions. In Twitter, since a user u could be following another user u', we can add a link from u to u' (e.g. link shown between user 2 and user 3 as shown in Figure 2). Although we did collect the network information for our dense dataset, we ran out of time to incorporate this into the analysis before this submission. This is something we are currently working on.

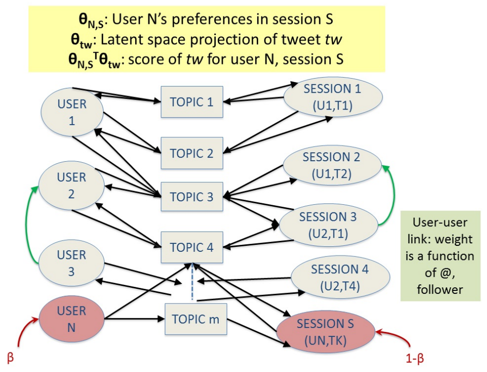
$\theta_{N,S}$: User N's preferences in session S
θ_{tw}: Latent space projection of tweet *tw*
$\theta_{N,S}^T\theta_{tw}$: score of *tw* for user N, session S
USER 1
USER 2
USER 3
USER N
TOPIC 1
TOPIC 2
TOPIC 3
TOPIC 4
TOPIC m
SESSION 1 (U1,T1)
SESSION 2 (U1,T2)
SESSION 3 (U2,T1)
SESSION 4 (U2,T4)
SESSION S (UN,TK)
User-user link: weight is a function of @, follower
β
1-β

Figure 2: The latent space STG framework with the tweets marginalized out.

7 Results

7.1 Experiments and prediction performance

The evaluation scheme of the recommendation framework detailed above, is the same as for the LSI analysis. For each tweet tw in the test set, we also randomly pick 9 other tweets. For tw, we also know the user u and session $< u, t >$ information about the tweet. We now rank these 10 tweets after computing $\theta_{u,<u,t>}$. We used the same datasets that were used for the LSI analysis.

Since there are 10 tweets to be ranked, the average rank (which is also the performance of a random recommender) is 5.5, since ranks range from 1 to 10. If our recommender can predict an average rank of better than 5.5, then it is catching some signal at least.

For the "sparse dataset" (as described in the Dataset section), the performance numbers were quite weak. For five different values of $\beta \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$, we ran 20 simulations each (the randomness in the experiment comes from the random selection of the 9 other tweets for each test tweet). However, it should be noted that the performance of STG with LDA projections was **consistently better than random** (see Table 2). The LSI projections do not work well with the STG framework since the STG semantics are designed for non-negative projection weights (a property of LDA and probabilistic LSI [9], but not of LSI). Squaring the LSI coefficients and renormalizing the vectors, did not improve the random performance.

For the "dense" dataset, the performance numbers were much better. Detailed results are shown in Table 2. We removed all words whose count in the entire corpus was either over 10000 (too common, hence like stopwords) or below 100 (too rare to be informative). Total remaining tokens were around 66000. In the 25 topics LDA model, all topics learnt seemed too similar, which was why the performance was akin to a random predictor.

Method	Mean rank	Std dev	Comments
Random	5.50	-	*Baseline*
LSI-latent space	**4.22**	0.09	Best predictor
LSI-STG sparse	5.51	0.04	Quite random
LDA-STG sparse	5.30	0.05	25 topics
LDA-STG dense	5.48	0.02	25 topics
LDA-STG dense I	4.42	0.02	100 topics
LDA-STG dense II	5.19	0.05	200 topics

Table 2: Overall performance of different models

However, when we learnt 100 LDA topics, the topics learnt were more discriminative and the predictive performance improved significantly to 4.42. In the 200 topic model, we had to reduce the number of tokens considered to around 30000 (done by further constraining the allowed range of word counts) due to memory considerations. This model did not perform that well.

Varying the value of β in the set mentioned above, did not affect predictive performance. This could be due to the short time frame of the "dense dataset", which meant only short-term preferences were captured in the training data.

Another variations we tried was changing the number of tweets to be ranked from 10 to 100. As expected, the performance numbers did not change much (modulo an appropriate scaling factor).

7.2 Interpreting the results

The bad performance on the sparse dataset is not very surprising. Since about 40% of all tweets are meaningless babble and about 38% is conversational, it is quite difficult to predict interest for these two types of tweets. The "sparse dataset" has only about 20 to 50 tweets in a 3-month period for the densest users. Hence, the signal is indeed very weak.

With the denser dataset, the significantly better performance numbers indicate that there is some promise in this framework. However, the superior performance of the simplistic latent space distance metric over the STG framework shows that the weighted random walk mechanism needs further refinement. Also, using probabilistic LSI projections with the latent STG framework might yield better results.

8 Conclusion

In summary, we have proposed two ways to build a recommendation framework for twitter, which is time-sensitive and user-specific. Also, the latent STG framework is flexible enough to allow for additional information sources like network links ("follows" in Twitter). There are also additional sources of user-user interaction in Twitter (via retweeting, sharing hashtags and directing a tweet at a particular user). The experiments so far suggest that sparse data will be difficult to learn enough about a user and hence make good predictions. However, with dense data, the predictions are significantly better.

Looking ahead, we are planning to incorporate the network information into the framework and see if that leads to better prediction. Also, a more careful study of the distance metrics in the various latent spaces should lead to not only a better understanding of the appropriateness of a metric, but also to performance improvements by choosing a good one.

Acknowledgements

We are thankful to the instructors for providing us with the sparse Twitter dataset. This helped us develop our initial ideas. We are also thankful to Prof. Canny for some interesting insights into topic models and for assisting us in concretizing the problem.

References

[1] http://www.facebook.com.

[2] http://www.twitter.com.

[3] http://community.marklogic.com/products/hadoop.

[4] https://dev.twitter.com/docs/api/1/get/statuses/user-timeline.

[5] https://dev.twitter.com/docs/api/1/get/friends/ids.

[6] http://en.wikipedia.org/wiki/Twitter.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[8] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990.

[9] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM.

[10] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[11] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 723–732, New York, NY, USA, 2010. ACM.