Retweet or not? Personalized Tweet Re-ranking

Wei Feng Jianyong Wang
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
feng-w10@mails.tsinghua.edu.cn, jianyong@tsinghua.edu.cn

ABSTRACT

With Twitter being widely used around the world, users are facing enormous new tweets every day. Tweets are ranked in chronological order regardless of their potential interestedness. Users have to scan through pages of tweets to find useful information. Thus more personalized ranking scheme is needed to filter the overwhelmed information. Since retweet history reveals users' personal preference for tweets, we study how to learn a predictive model to rank the tweets according to their probability of being retweeted. In this way, users can find interesting tweets in a short time. To model the retweet behavior, we build a graph made up of three types of nodes: users, publishers and tweets. To incorporate all sources of information like users' profile, tweet quality, interaction history, etc, nodes and edges are represented by feature vectors. All these feature vectors are mapped to node weights and edge weights. Based on the graph, we propose a feature-aware factorization model to re-rank the tweets, which unifies the linear discriminative model and the low-rank factorization model seamlessly. Finally, we conducted extensive experiments on a real dataset crawled from Twitter. Experimental results show the effectiveness of our model.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering, Retrieval Models, Selection Process*

Keywords

Social Media; Recommender System

1. INTRODUCTION

With 500 million active users and 340 million tweets generated every day[1], Twitter is one of the leading social networking and microblogging service providers in the world. A tweet is a text message of up to 140 characters generated by users. Users can follow their friends, idols and any other kind of information sources to keep up with their latest tweets. Tweets can be shared along the followee-follower links so that more users are informed. The sharing action is referred as retweet. Retweet behavior is the key part behind the information spreading in Twitter.

When a user logs in to Twitter, tweets are ranked in chronological order regardless of their potential interestedness. Users have to scan through pages of tweets to find important information. Thus a more personalized ranking scheme is needed to deal with the overwhelmed updates. Since retweet history can be considered as explicit feedbacks of one's preference for tweets, we can learn a predictive model to find what tweets are likely to attract one's attention. Tweets can be re-ranked according to their probability of being retweeted. This re-ranking scheme will help users find the important tweets that are missed since last login in a short time. Personalized re-ranking can serve as a complementary information filtering tool to the current system.

While some work [2, 3, 7, 12] has been done in studying the retweet behavior, they mainly focused on the global factors, i.e., predicting popular tweets in the whole social network or in the perspective of information spreading[11, 14]. Different from previous work, our work is focused on local factors at the individual level, i.e., building a predictive model that serve for each user.

Our contributions are summarized as follows:

- We propose a novel problem called personalized tweet re-ranking to help users deal with overwhelmed tweets. To the best of our knowledge, we are among the first to study retweet behavior at the individual level.
- We propose a general graph model to analyze retweet behavior. All sources of information can be converted to the feature vectors of nodes and edges. The graph model is fully extensible to new features.
- Based on the graph, we propose a feature-aware factorization model which unifies the linear discriminative and the low-rank factorization models seamlessly.
- We have conducted extensive experiments on a real dataset crawled from Twitter. Our model is proved to be more effective than the existing models.

The rest of the paper is organized as follows: In Section 2, we give a formal definition of our problem and discuss the drawbacks of the existing methods. In Section 3, we introduce our basic framework and all kinds of features. Our feature-aware matrix factorization model is introduced in

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

WSDM'13, February 4–8, 2013, Rome, Italy.

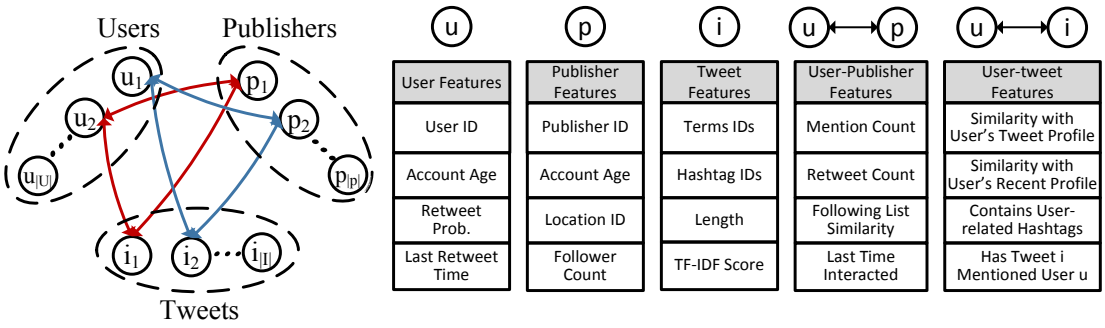

Figure 1: Retweet behavior modeled by a graph. Nodes and edges are represented by features. Some typical features are listed here.

Section 4. Section 5 describes our experimental study on a dataset crawled from Twitter, where we show that our model is more effective than the existing models. In Section 6, we introduce the related work. Finally, we conclude the paper in Section 7.

2. PRELIMINARIES

2.1 Problem Statement

Every time a user signs in to Twitter, she/he will get the latest 20 tweets[2] on the home page. As the user rolls down to the bottom, the client will pull another 20 tweets until the user stops scanning. The whole process is called a session in our paper. Usually only a small number of tweets will be retweeted in a session. The goal of Personalized Tweet Re-Ranking (PTR) is to rank the tweets that are more likely to be retweeted at the front of the list for each session.

Let U, P, I denote the user set, publisher set and tweet set[3], respectively. Since a tweet can be received by multiple users, we define a tweet as a tuple $<u, p, i, t, r_{upi}>$ where

- $u \in U$ is the receiver of the tweet, $p \in P$ is the publisher of the tweet, $i \in I$ is tweet itself.
- t is the timestamp of the session indicating when user u saw tweet i.
- r_{upi} is a binary constant indicating whether this tweet is retweeted by user u. r_{upi} can be viewed as the class label or u's binary rating for tweet i.

Formally, the goal of PTR is to predict r_{upi} for each new tweet. Let $\widehat{r}_{upi}$ denote the predicted rating, there are two ways to judge whether $\widehat{r}_{upi}$ is a good approximation of r_{upi}:

- **Point-wise Approach**. This approach tries to find the best boundary to separate the positive tweets from the negative tweets. When r_{upi} is 1, $\widehat{r}_{upi}$ should also be close to 1.
- **Pair-wise Approach**. This approach tries to find the best $\widehat{r}_{upi}$ so that $\widehat{r}_{upi}$ and r_{upi} can generate the same ranking order. $\widehat{r}_{upi}$ is not limited to have a similar value with r_{upi}.

More details will be discussed in Section 4.3.

[2]The exact number may be different in different clients.
[3]Tweet set can be viewed as item set in traditional recommender systems

2.2 Drawbacks of Matrix Factorization

In this section, we will discuss the reason why existing matrix factorization techniques cannot be directly applied to PTR and what are the differences between PTR and traditional recommendation task.

In the setting of matrix factorization, low-dimension vectors $\mathbf{p}_u$ and $\mathbf{p}_i$ are learned for each user u and item i respectively to approximate the original user-item rating matrix. Let r_{ui} denote user u's rating to item i, r_{ui} is predicted according to the following equation [4]

$$\widehat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^T \mathbf{p}_i \quad (1)$$

where μ is the global bias, b_u and b_i are the biases for user u and item i, respectively. There are two challenges that cannot be solved by the traditional matrix factorization models:

- Tweets have much stronger temporal effects than traditional items like movies and songs. On the one hand, users usually prefer new tweets. On the other hand, most new tweets have barely no feedbacks from users, since other users may have not seen them yet. Moreover, with the low cost of composing tweets, tweets are generated with a much higher frequency than traditional items. In this case, most tweets in the test set may not exist in the training set at all. Experimental results in Section 5.4 empirically prove this.
- The auxiliary information is much richer than the traditional user-item rating data. For example, a tweet can contains URLs, medias and hashtags. All of them can be viewed as indicators to the content quality. The times and frequency that users interact with each other will indicate whether they will keep interacting in the future. The number of followers and the fraction of tweets being retweeted can indicate a user's authority. Traditional matrix factorization lacks support for the auxiliary information.

3. FRAMEWORK

In this section, we first introduce the graph model. Then we introduce our basic framework.

Recall a tweet is defined as a tuple $<u, p, i, t, r_{upi}>$ in Section 2.1. There are three entities involved during the retweet action: user u, publisher p and tweet i itself. It is intuitive to construct a graph to illustrate the involved factors in retweet behavior. The graph is shown in Figure 1. Now we give a formal definition of the graph.

Definition 1. (**Retweet Graph**) Retweet behavior can be represented as a graph G = (V, E) where

- V = $U \cup P \cup I$. Three types of nodes are involved: users, publishers, and tweets.
- E = <u, p>∪<u, i>∪<p, i>. <u, p> represents u's trust to p. <u, i> represents the content interestedness of tweet i to u. <p, i> represents the fact that p is the publisher of tweet i.

As shown in Figure 1, nodes and edges in the retweet graph have rich auxiliary information. To incorporate all sources of information, each node is represented by a feature vector $\mathbf{x}_m$ ($m \in \{u, p, i\}$) and each edge is also represented by a feature vector $\mathbf{x}_{um}$ ($m \in \{p, i\}$)[4]. These features will finally be mapped to node weights and edge weights.

We assume that a retweet action is mainly influenced by two types of factors:

- **Node Weights**. Node weights can be interpreted as whether user u is willing to retweet, whether publisher p is likely to be retweeted, whether tweet i has a high quality, etc. Based on the specific node features, node weights can incorporate multiple meanings.
- **Edge Weights**. Edge weights can describe relations like whether user u and publisher p are close friends, and whether tweet i is interesting to user u. Again, all the information can be contained in the edge feature.

According to the above assumption, we designed the prediction function:

$$\widehat{r}_{upi} = \sum_{m \in \{u,p,i\}} f_m + \sum_{m \in \{p,i\}} f_{um} + \sum_{m \in \{p,i\}} \mathbf{g}_u^T \mathbf{g}_m \qquad (2)$$

where f_m is a function that maps a node feature vector to node weight, f_{um} is a function that maps the edge feature vector to edge weights. $\mathbf{g}_u^T \mathbf{g}_m$[5] is a term representing the factorization part. In this paper, four relations are factorized: user-publisher, user-location, user-term, and user-hashtag. Unlike the user-tweet relation, the above relations are all much denser. But our framework is not limited to the above four relations. It is fully extensible to new relations. The details will be discussed in Section 4.

In the next two subsections, we will introduce the node features and edge features used in f_m ($m \in \{u, p, i\}$) and f_{um} ($m \in \{p, i\}$). Some of the features are designed to be calculated in realtime (incrementally) to catch the temporal effects. All the node features marked by '*' are further used in the factorization term $\mathbf{g}_u^T \mathbf{g}_m$.

3.1 Node Features

3.1.1 Tweet Features

Term IDs/Hashtag IDs*. Although it is not common to use IDs as features, they have special meaning in our model. According to Equation 2, f_i will map term IDs to their latent biases. These latent biases can be viewed as the automatically learned term weights, which can serve as complements to the hand-crafted TF-IDF scores. $\mathbf{g}_i$ will map term IDs to low-dimension latent vectors. Each dimension can be regarded as a latent topic. Each latent vector is a distribution over these topics. $\mathbf{g}_u^T \mathbf{g}_i$ will calculate the interestedness of the terms for user u. More details will be discussed in Section 4.

Has geography/hashtag/multimedia/URL/mention. This set of boolean features is used to indicate the content quality. For example, a tweet with multimedia like an video or an image will be more likely to be noticed. If a tweet mentions the user, the user will be more likely see it.

Length and average TF-IDF score. Both the features describe the content quality.

Retweet count and time span since last time being retweeted. Both the features are calculated in realtime and can be updated incrementally. They are used to capture tweets with a high spreading speed.

Time span since created. This feature measures the freshness of the tweet. Since fresh tweets carry new information, they tend to be more valuable than older tweets.

3.1.2 Publisher Features

Publisher ID*. Like term IDs and hashtag IDs, publisher IDs in f_p will be mapped to latent biases, which represent the prior probabilities of publishers' tweets being retweeted. This reflects the idea that tweets from authorities are more likely to be retweeted than tweets from common publishers. Besides the latent biases, $\mathbf{g}_p$ will map publisher IDs to low-dimension latent vectors, which have the same space with low-dimension vectors of terms or hashtags. Each dimension represents a topic and the value at the dimension represents the publisher's authority on this topic. Finally, user u's trust to publisher p is calculated by $\mathbf{g}_u^T \mathbf{g}_p$. More details will be introduced in Section 4.

Location ID*. Location IDs represents the cities or countries extracted from publishers' profiles. This feature is used to capture the spatial effects like local events and language preferences. Followers that have a strong preference toward a location will also likely to be interested in publishers in this location. f_p maps location IDs to their latent biases, which represent activeness of locations. $\mathbf{g}_p$ maps location IDs to latent vectors. Each vector represents a distribution over the k latent topics. Finally, the topic distribution of a publisher is partially influenced by her/his location.

Prior probability of being retweeted and time span since last time being retweeted. Both the features are used to measure the probability of the publisher's tweets getting retweeted. They can filter publishers that seldom get attentions and capture popular publishers.

Authority score. The authority score is defined as the ratio between the follower count and the followee count. This feature states the social status of the publisher. High authorities are likely to have much more followers than followees.

Is verified. Verified publishers tend to earn more trust.

Mention count. If a publisher is frequently mentioned, she/he is more likely to be popular and has more interactions than other publishers.

Account Age. From this feature we can know how long a user has participated in Twitter. Older accounts tend to be more influential in the social network.

[4]Since <p, i> has little connection with whether user u will retweet tweet i, we will mainly focus on the other edges in this paper, i.e., <u, p> and <u, i>.

[5]The function value of $\mathbf{g}_u$ is a vector. For convenience, we also use $\mathbf{g}_u$ to represent a vector.

3.1.3 User Features

User ID*. Like publisher IDs, f_u will map user IDs to latent biases, which state their willingness to retweet. $\mathbf{g}_u$ will map user IDs to low-dimension latent vectors, which states the user's preferences for each latent topic. User u's trust to publisher p is defined as $\mathbf{g}_u^T\mathbf{g}_p$. Tweet i's interestedness to u will be calculated according to $\mathbf{g}_u^T\mathbf{g}_i$.

Prior probability of retweet and time span since last retweet. As a complement to users' biases, this feature also describes users' willingness to retweet. However, this feature is hand crafted. It is computed by n_{ret}/n_{recv}, where n_{ret} and n_{recv} are the number of tweets retweeted and received respectively before timestamp t. The time span since last retweet will indicate how long a user has not retweeted. These features can filter users that seldom retweet and catch active users that often retweet.

Account Age. This feature is the same with the account age of publishers.

3.2 Edge Features

3.2.1 User-Publisher Features

Similarity of tweet profiles. Users and publishers are represented by terms in their tweets. All the tweets (originally posted and retweeted) are integrated into a single document. The tweet profile is a term vector with TF-IDF weighting scheme of all the tweets. The similarity between a user and a publisher is defined as the similarity of their tweet profiles in the term vector space. This feature is used to measure the user-publisher similarity in the long term.

Similarity of recent tweet profiles. Each user and publisher are represented by their latest ten tweets. This feature can describe whether the user and the publisher focus on the the same topic recently. We want to catch the short term interests. Users' recent tweet profiles are updated incrementally in realtime.

Similarity of self-descriptions. Users can write self-descriptions to introduce themselves. The similarity is defined as the cosine distance of their description in the vector space model.

Similarity of the following lists. This feature is defined as the Jaccard similarity of the following lists of the user and the publisher. We assume that similar users tend to follow similar publishers.

Are friends/Is same location/same time zone. 'Are friends' refers to whether users and publishers follow each other, while 'Is same location' and 'Is same time zone' are used to find whether they are close in physical world.

Ratio of their authority score/Are both verified. Recall that 'Authority score' is defined as the ratio of follower count and followee count. According to whether a user and a publisher are verified, we have four different combinations, i.e., user is verified but publisher is not verified, etc. We want to know whether users and publishers have comparable influence.

Mention count/Retweet count/Reply count/Time span since last interaction. This set of feature describes the closeness between the user and the publisher. We assume users with many interactions in the past tend to keep interacting in the future. These feature are calculated in realtime and can be updated incrementally.

3.2.2 User-Tweet Features

Similarity of the tweet and the user's tweet profile. Recall that a user's tweet profile is a TF-IDF vector of all his tweets. This feature describes the content similarity between the tweet and the user's tweet profile.

Similarity of the tweet and the user's recent tweet profile. Recall that a user's recent tweet profile is a TF-IDF vector of his lastest ten tweets. This feature describes whether this tweet is related to user's short term interests.

Has mentioned the user/Has hashtags related to the user. If the tweet has mentioned the user, he will be informed to see this tweet. Hashtags related to user u are defined as the hashtags used or retweeted by user u.

3.3 Nodes and Edges as Feature Vectors

In this section, we introduce some details about how these features are used in our algorithm.

There are two types of features involved in this paper:

- **Numeric features**. Numeric features are normalized to have mean equals to zero and standard deviation equals to one.
- **Categorical features**(including boolean features). A categorical feature with k categories is converted to a sparse vector of length k. The i-th entry corresponds to the i-th category. Take user ID as an example. Suppose user ID ranges from 1 to 5. u_2 will be represented by $(0, 1, 0, 0, 0)^T$. For features like term IDs and hashtag IDs, the feature vector is normalized to sum to 1. Take term IDs for example. Suppose the content of tweet i is 'WSDM is coming' and the dictionary is {1:WSDM, 2:is, 3:coming, 4:data, 5:mining}. Tweet i can be represented by $\mathbf{x}_i = (1, 1, 1, 0, 0)^T$. Normalizing $\mathbf{x}_i$ by the number of nonzero entries, i.e., three, we have $\mathbf{x}_i = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0)^T$. Without normalization, term IDs of long tweets will dominate other features.

Missing feature values are specially handled in our task.

- **Numeric features**. If a numeric feature is missing, it will be marked as 'NA' and a new boolean feature is added and set to true. A suitable weight can be found in the training process to replace the missing value. For example, if a user's tweets have never been retweeted, the feature 'last time being retweeted' is missing. So we add new boolean feature $x_{new} = 1$ with weight θ_{new}. During the training process, $\theta_{new}x_{new}$ is used to represent the missing value. With enough training examples, a proper θ_{new} will be learned. For non-missing values, the added boolean feature is false.

- **Categorical features**. If a categorical feature has missing values, a new category is added to represent all the missing values. Take hashtag IDs as an example. For tweets that do not have hashtags, we add a new hashtag 'NULL' to represent the missing hashtag.

4. FEATURE-AWARE FACTORIZATION MODEL

In this section, we will discuss how to incorporate all the features in our model. We will introduce the meaning of Equation 2 in two steps: Section 4.1 introduces the definitions of f_m and f_{um} ($m \in \{p, i\}$). Section 4.2 introduces the definition of $\mathbf{g}_u^T\mathbf{g}_m$ ($m \in \{p, i\}$).

4.1 Feature-based Approach

When only considering f_m and f_{um}, Equation 2 is equivalent to

$$\widehat{r}_{upi} = \sum_{m\in\{u,p,i\}} f_m + \sum_{m\in\{p,i\}} f_{um} \tag{3}$$

where f_m maps a node feature vector $\mathbf{x}_m$ to the node weight and f_{um} maps an edge feature vector $\mathbf{x}_{um}$ to the edge weight. In the simplest case, f_m is defined to be a linear combination of all the node features:

$$f_m = \boldsymbol{\theta}_m^T \mathbf{x}_m \tag{4}$$

where $\boldsymbol{\theta}_m$ is the feature weight vector that stores the importance of each dimension. f_{um} is also defined to be the linear form:

$$f_{um} = \boldsymbol{\theta}_{um}^T \mathbf{x}_{um} \tag{5}$$

where $\boldsymbol{\theta}_{um}$ is the feature weight vector corresponding to edge feature vector $\mathbf{x}_{um}$.

With the definition of f_m and f_{um}, we discuss the meaning of user ID and publisher ID in feature vectors. Suppose user ID and publisher ID both range from 1 to 5. When other features are ignored, u_1 can be represented by $\mathbf{x}_{u_1} = (1,0,0,0,0)^T$ and p_2 can be represented by $\mathbf{x}_{p_2} = (0,1,0,0,0)^T$. According to Equations 4 and 5, we have

$$\widehat{r}_{u_1p_2i} = \boldsymbol{\theta}_{u,1} + \boldsymbol{\theta}_{p,2} \tag{6}$$

where $\boldsymbol{\theta}_{m,k}$ represents the k-th entry of vector $\boldsymbol{\theta}_m$. $\boldsymbol{\theta}_{u,1}$ and $\boldsymbol{\theta}_{p,2}$ represent the general biases of u_1 and p_2. $\boldsymbol{\theta}_{u,1}$ states u_1's willingness to retweet while $\boldsymbol{\theta}_{p,2}$ states the probability of the tweets of p_2 being retweeted. With enough training instances and an effective loss function, appropriate $\boldsymbol{\theta}_{u,1}$ and $\boldsymbol{\theta}_{p,2}$ will be found. These learned biases can be treated as complementary to hand-crafted features like 'prior probability of retweet' and 'prior probability of being retweeted' introduced in Section 3.1.

With the understanding of user ID and publisher ID, it is easy to understand the meaning of term IDs. Suppose the content of tweet i is 'WSDM is coming' and the dictionary is {1:WSDM, 2:is, 3:coming, 4:data, 5:mining}. When only term IDs are considered, tweet feature vector $\mathbf{x}_i$ is $(1,1,1,0,0)^T$. To avoid long tweet dominating the ranking score, we normalize each entry by the text length, which makes $\mathbf{x}_i$ equal to $(\frac{1}{3},\frac{1}{3},\frac{1}{3},0,0)^T$. According to Equations 4 and 5, we have

$$\widehat{r}_{upi} = (\boldsymbol{\theta}_{i,1} + \boldsymbol{\theta}_{i,2} + \boldsymbol{\theta}_{i,3})/3 \tag{7}$$

where $\boldsymbol{\theta}_{i,1}$, $\boldsymbol{\theta}_{i,2}$ and $\boldsymbol{\theta}_{i,3}$ are automatically learned term importance. $(\boldsymbol{\theta}_{i,1}+\boldsymbol{\theta}_{i,2}+\boldsymbol{\theta}_{i,3})/3$ describes the content quality of tweet i. $\boldsymbol{\theta}_{i,k}$ can be considered as a complement to TF-IDF score of term k.

Now we discuss why tweet ID is not a good option for tweet features. Recall that in Section 2.2 we argue that tweets have stronger temporal effects. Since only a small number of recent tweets can attract users' attention, most tweet biases will have no training data at all. Unlike tweets, most users, publishers and terms are likely to exist in both training data and test data.

Finally we discuss the meaning of f_{up} and f_{ui} in Equation 3. f_{up} and f_{ui} are edge weights of $<u, p>$ and $<u, i>$, respectively. $<u, p>$ represents how much u trusts p. $<u, t>$ can be considered as the interestedness of t to u. While node weights provide a prior knowledge of each object, edge weights contain more personalized information. Unlike node weight, edge weights are different for each different instances. Thus they are considered to be more effective, which is also empirically proved in Section 5.5.

So far, all the edge features (i.e., similarities between nodes) are hand-crafted. In the next section, we will discuss how factorization techniques can be used to serve as complements to the hand-crafted features.

4.2 Feature-aware Factorization Model

The basic idea of $\mathbf{g}_m$ $(m \in \{u,p,i\})$ is to learn a k-dimension latent vector for each dimension of the node feature, where k is usually a number from 50 to 200. Each latent vector can be viewed as a distribution of preferences over the k latent topics. $\mathbf{g}_u^T\mathbf{g}_p$ represents the similarity of user u and publisher p over k latent topics. $\mathbf{g}_u^T\mathbf{g}_p$ and $\mathbf{g}_u^T\mathbf{g}_i$ can serve as complements to f_{up} and f_{ui}, where similarities are calculated from hand-crafted features.

Like f_m, $\mathbf{g}_m$ is defined to be a linear form:

$$\mathbf{g}_m = \boldsymbol{\Phi}_m \mathbf{x}_m \tag{8}$$

where $\boldsymbol{\Phi}_m$ is a $k \times |\mathbf{x}_m|$ matrix.

$\mathbf{g}_m$ in the special form. We use $\mathbf{g}_i$ as an example to explain the meaning of $\mathbf{g}_m$. According to the node features introduced in Section 3.1, tweet i can be represented by its terms and hashtags, i.e., $\mathbf{x}_i = (\mathbf{x}_i^{term}, \mathbf{x}_i^{tag})^T$. Corresponding to $\mathbf{x}_i^{term}$ and $\mathbf{x}_i^{tag}$, $\boldsymbol{\Phi}_i$ can also be rewritten to two blocks, i.e., $\boldsymbol{\Phi}_i = [\boldsymbol{\Phi}_i^{term}, \boldsymbol{\Phi}_i^{tag}]$. $\mathbf{g}_i$ is computed by

$$\begin{aligned}\mathbf{g}_i &= \left[\boldsymbol{\Phi}_i^{term}, \boldsymbol{\Phi}_i^{tag}\right] \begin{bmatrix} \mathbf{x}_i^{term} \\ \mathbf{x}_i^{tag} \end{bmatrix} \\ &= \boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term} + \boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag}\end{aligned} \tag{9}$$

We can see that $\mathbf{g}_i$ is made up of two components, i.e., $\boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term}$ and $\boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag}$. $\boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term}$ represents the latent vectors inferred from terms while $\boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag}$ represents the latent vectors inferred from hashtags. Suppose the content of tweet i is '#WSDM# #Data Mining# WSDM is coming'. The term dictionary is {1:WSDM, 2:is, 3:coming, 4:data, 5:mining} and the hashtag dictionary is {1:WSDM, 2:Data Mining, 3:Machine Learning}. Accoring to the dictionaries, we have $\mathbf{x}_i^{term} = (\frac{1}{3},\frac{1}{3},\frac{1}{3},0,0)^T$ and $\mathbf{x}_i^{tag} = (\frac{1}{2},\frac{1}{2},0)^T$. Plug $\mathbf{x}_i^{term}$ and $\mathbf{x}_i^{tag}$ into Equation 9, we have

$$\begin{aligned}\mathbf{g}_i &= \boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term} + \boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag} \\ &= \frac{\boldsymbol{\Phi}_{i,1}^{term} + \boldsymbol{\Phi}_{i,2}^{term} + \boldsymbol{\Phi}_{i,3}^{term}}{3} + \frac{\boldsymbol{\Phi}_{i,1}^{tag} + \boldsymbol{\Phi}_{i,2}^{tag}}{2}\end{aligned} \tag{10}$$

where $\boldsymbol{\Phi}_{i,k}$ represents the k-th column of $\boldsymbol{\Phi}_i$. When $\mathbf{x}_i^{term} = (\frac{1}{3},\frac{1}{3},\frac{1}{3},0,0)^T$, only the first three columns of $\boldsymbol{\Phi}_i^{term}$ are retained. '$(\boldsymbol{\Phi}_{i,1}^{term}+\boldsymbol{\Phi}_{i,2}^{term}+\boldsymbol{\Phi}_{i,3}^{term})/3$' in the above equation represents the averaged latent vector for term IDs. '$(\boldsymbol{\Phi}_{i,1}^{tag} + \boldsymbol{\Phi}_{i,2}^{tag})/3$' represents the averaged latent vector for hashtag IDs. $\mathbf{g}_i$ is a combination of these two latent vectors.

$\mathbf{g}_m^T\mathbf{g}_n$ in the special form. Now we explain the meaning of $\mathbf{g}_u^T\mathbf{g}_i$ in Equation 2. According to Equation 8, we have $\mathbf{g}_u = \boldsymbol{\Phi}_u\mathbf{x}_u$. $\mathbf{g}_u^T\mathbf{g}_i$ is computed as the following

$$\begin{aligned}\mathbf{g}_u^T\mathbf{g}_i &= (\boldsymbol{\Phi}_u\mathbf{x}_u)^T(\boldsymbol{\Phi}_i\mathbf{x}_i) \\ &= (\boldsymbol{\Phi}_u\mathbf{x}_u)^T(\boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term} + \boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag}) \\ &= (\boldsymbol{\Phi}_u\mathbf{x}_u)^T(\boldsymbol{\Phi}_i^{term}\mathbf{x}_i^{term}) + (\boldsymbol{\Phi}_u\mathbf{x}_u)^T(\boldsymbol{\Phi}_i^{tag}\mathbf{x}_i^{tag})\end{aligned} \tag{11}$$

where $(\mathbf{\Phi}_u \mathbf{x}_u)^T(\mathbf{\Phi}_i^{term}\mathbf{x}_i^{term})$ represents user u's preference to tweet i's terms, and $(\mathbf{\Phi}_u \mathbf{x}_u)^T(\mathbf{\Phi}_i^{tag}\mathbf{x}_i^{tag})$ represents user u's preference to tweet i's hashtags.

Now we discuss the meaning of $\mathbf{g}_u^T\mathbf{g}_p$. According to publisher features introduced in Section 3.1, a publisher can be represented by $\mathbf{x}_p = (\mathbf{x}_p^{ID}, \mathbf{x}_p^{loc})$, where $\mathbf{x}_p^{ID}$ is a feature vector of publisher IDs and $\mathbf{x}_p^{loc}$ is a feature vector of location IDs. Similar to $\mathbf{g}_u^T\mathbf{g}_i$, $\mathbf{g}_p^T\mathbf{g}_p$ is calculated by

$$\begin{aligned}\mathbf{g}_u^T\mathbf{g}_p &= (\mathbf{\Phi}_u\mathbf{x}_u)^T(\mathbf{\Phi}_p\mathbf{x}_p) \\ &= (\mathbf{\Phi}_u\mathbf{x}_u)^T(\mathbf{\Phi}_p^{ID}\mathbf{x}_p^{ID} + \mathbf{\Phi}_p^{loc}\mathbf{x}_p^{loc}) \\ &= (\mathbf{\Phi}_u\mathbf{x}_u)^T(\mathbf{\Phi}_p^{ID}\mathbf{x}_p^{ID}) + (\mathbf{\Phi}_u\mathbf{x}_u)^T(\mathbf{\Phi}_p^{loc}\mathbf{x}_p^{loc})\end{aligned} \tag{12}$$

where $(\mathbf{\Phi}_u \mathbf{x}_u)^T(\mathbf{\Phi}_p^{ID}\mathbf{x}_p^{ID})$ represents user u's preference to publisher p, and $(\mathbf{\Phi}_u \mathbf{x}_u)^T(\mathbf{\Phi}_p^{loc}\mathbf{x}_i^{loc})$ represents user u's preference to publisher p's location.

Combining Equation 11 with Equation 12, we can find that four matrices (relations) are factorized in our problem: user-term matrix, user-hashtag matrix, user-publisher matrix, and user-location matrix.

$\mathbf{g}_m$ in the general form. Now we discuss $\mathbf{g}_m$ ($m \in \{u, p, i\}$) in a more generalized setting. Suppose node m has k_m features, then we have $\mathbf{x}_m = (\mathbf{x}_m^1, \mathbf{x}_m^2, ..., \mathbf{x}_m^{k_m})^T$ and $\mathbf{\Phi}_m = (\mathbf{\Phi}_m^1, \mathbf{\Phi}_m^2, ..., \mathbf{\Phi}_m^{k_m})$. According to Equation 8, $\mathbf{g}_m$ is calculated according to the following equation

$$\begin{aligned}\mathbf{g}_m &= \left[\mathbf{\Phi}_m^1, \mathbf{\Phi}_m^2, ..., \mathbf{\Phi}_m^{k_m}\right]\left[\mathbf{x}_m^1, \mathbf{x}_m^2, ..., \mathbf{x}_m^{k_m}\right]^T \\ &= \sum_{q=1}^{k_m}\mathbf{\Phi}_m^q\mathbf{x}_m^q\end{aligned} \tag{13}$$

where $\mathbf{\Phi}_m^q\mathbf{x}_m^q$ is a latent vector for node feature vector $\mathbf{x}_m^q$. $\mathbf{g}_m$ is a combination of these latent vectors.

$\mathbf{g}_m^T\mathbf{g}_n$ in the general form. According to Equation 13, we $\mathbf{g}_m^T\mathbf{g}_n$ is calculated by

$$\begin{aligned}\mathbf{g}_m^T\mathbf{g}_n &= (\sum_{q=1}^{k_m}\mathbf{\Phi}_m^q\mathbf{x}_m^q)^T(\sum_{p=1}^{k_n}\mathbf{\Phi}_n^p\mathbf{x}_n^p) \\ &= \sum_{q=1}^{k_m}\sum_{p=1}^{k_n}(\mathbf{\Phi}_m^q\mathbf{x}_m^q)^T(\mathbf{\Phi}_n^p\mathbf{x}_n^p)\end{aligned} \tag{14}$$

The above equation states that similarity between node m and node n is the summation of pair-wise dots of node m's latent vectors and node n's latent vectors.

Connections with traditional matrix factorization models. Our feature-aware factorization model can be considered as an extension of the traditional matrix factorization model. In the simplest case, our model can be degraded to the matrix factorization model defined by Equation 1.

We can illustrate the connections with an example. Suppose user ID and publisher ID all range from 1 to 5. When only user ID and publisher ID are considered, node feature of u_1 is $\mathbf{x}_{u_1} = (1, 0, 0, 0, 0)^T$ and node feature of p_2 is $\mathbf{x}_{p_2} = (0, 1, 0, 0, 0)^T$. According to Equation 2, the similarity term $\mathbf{g}_u^T\mathbf{g}_p$ is calculated by

$$\begin{aligned}\mathbf{g}_u^T\mathbf{g}_p =& (\mathbf{\Phi}_u\mathbf{x}_{u_1})^T(\mathbf{\Phi}_p x_{p_2}) \\ =& \mathbf{\Phi}_{u,1}^T\mathbf{\Phi}_{p,2}\end{aligned} \tag{15}$$

where $\mathbf{\Phi}_{i,j}$ represents the j-th column of matrix $\mathbf{\Phi}_i$. Plug $\mathbf{g}_u^T\mathbf{g}_p$ and $f_u^T f_p$ into Equation 2 and ignore other features, where $f_u^T f_p$ for u_1 and p_2 has been explained in Equation 6, we have

$$\widehat{r}_{u_1p_2i} = \boldsymbol{\theta}_{u,1} + \boldsymbol{\theta}_{p,2} + \mathbf{\Phi}_{u,1}^T\mathbf{\Phi}_{p,2} \tag{16}$$

The above equation has the same form as Equation 1. This means feature-aware matrix factorization can be degraded to matrix factorization if only user ID and item ID are used as features.

To summarize, our feature-aware factorization model learns latent biases and vectors for each node feature. In this paper, we only consider five types of node features: user ID, publisher ID, location ID, term IDs, and hashtag IDs. Our model is fully extensible to new features, as long as the model has not over-fitted the problem.

4.3 Loss Function

In the previous section we have defined parameters $\boldsymbol{\theta}_m$, $\boldsymbol{\theta}_{um}$ and $\mathbf{\Phi}_m$ in Equations 4, 5, and 8. The ranking score $\widehat{r}_{upi}$ is dependant on these parameters. To find the best parameter, we need a good loss function to measure whether $\widehat{r}_{upi}$ is a good approximation of r_{upi}. Two types of loss functions are considered in this paper:

Point-wise Loss. Point-wise approach is similar to a binary classification task. Whenever the real label r_{upi} is 1, the predicted score $\widehat{r}_{upi}$ should be close to 1. First we choose the logistic function to transform $\widehat{r}_{upi}$ to (0, 1) interval:

$$\widehat{r}'_{upi} = \sigma(-\widehat{r}_{upi}) \tag{17}$$

where $\sigma(x) = 1 \,/\, (1 + e^{-x})$. The loss function is defined as

$$\begin{aligned}l(\boldsymbol{\theta}_m, \boldsymbol{\theta}_{um}, \mathbf{\Phi}_m) = -r_{upi}\log\widehat{r}'_{upi} + (r_{upi} - 1)\log(1 - \widehat{r}'_{upi}) \\ + regularzation\end{aligned} \tag{18}$$

The $regularzation$ term will be introduced at the end of the section. Now we explain the meaning of the loss function. Suppose $r_{upi} = 1$, the above loss function will be $\log(\widehat{r}'_{upi})$. If $\widehat{r}'_{upi}$ is close to 1, the loss is zero. If $\widehat{r}'_{upi}$ is close to 0, the loss will be close to positive infinite. Our goal is to minimize the loss function so that $\widehat{r}'_{upi}$ is always close to r_{upi}.

Pair-wise Loss. Pair-wise loss focuses on the relative ranking order instead of the difference between $\widehat{r}_{upi}$ and r_{upi}. When a negative example is ranked higher than a positive example, a loss is generated. AUC is such a metric that measures the probability of ranking a positive example higher than a negative example. Let T^+ and T^- denote the indices of positive tweets and negative tweets, respectively. AUC is defined as

$$AUC = \frac{\sum_{p=1}^{|T^+|}\sum_{q=1}^{|T^-|}\mathbb{I}(\widehat{r}_{u_pp_pi_p} - \widehat{r}_{u_qp_qi_q})}{|T^+||T^-|} \tag{19}$$

where $\mathbb{I}(x)$ is 1 when $x > 0$, otherwise $\mathbb{I}(x)$ is 0. Since sigmoid function can be considered as a smoothed version of $\mathbb{I}(x)$ and is differentiable, we replace $\mathbb{I}(x)$ with sigmoid function. The final pair-wise loss function is defined as

$$\begin{aligned}l(\boldsymbol{\theta}_m, \boldsymbol{\theta}_{um}, \mathbf{\Phi}_m) =& \frac{\sum_{p=1}^{|T^-|}\sum_{q=1}^{|T^+|}\sigma(\widehat{r}_{u_pp_pi_p} - \widehat{r}_{u_qp_qi_q})}{|T^+||T^-|} \\ &+ regularzation\end{aligned} \tag{20}$$

Note that we have changed the order of positive examples and negative examples in the above equation, since we want

to minimize the loss function. Now it represents the probability of ranking a negative example higher than a positive example, which can be viewed as the loss.

Regularization. Regularization is used to punish big parameters so that the model is not over-fitted. We use L2-norm regularization. Suppose node m have k_m feature components, the regularization term is defined as

$$regularization = \sum_{m\in\{u,p,i\}} \lambda_m ||\boldsymbol{\theta}_m||^2 + \sum_{m\in\{p,i\}} \mu_m ||\boldsymbol{\theta}_{um}||^2 + \sum_{m\in\{u,p,i\}} \sum_{q=1}^{k_m} \lambda_{m,q} ||\boldsymbol{\Phi}_m^q||^2 \quad (21)$$

where λ_m, $\lambda_{m,q}$ and μ_m are regularization parameters that control the the sensitiveness to big parameters, which are often set empirically. [10] has proposed an efficient method to find the optimal regularization parameters.

4.4 Parameter Learning

The parameters are learned by minimizing the loss function with stochastic gradient descent. The basic idea of stochastic gradient descent is to calculate the gradient with respect to each training instance and move a tiny step along the descent direction according to the gradient. The step size is controlled by a parameter called *learning rate*. As long as the *learning rate* is not too large, parameters are guaranteed to converge to a global or local optima. Now we briefly list the gradient of Equation 18 and Equation 20.

Let $\boldsymbol{\omega}$ denote the parameter set, i.e., $\boldsymbol{\omega} = \{\boldsymbol{\theta}_m, \boldsymbol{\theta}_{um}, \boldsymbol{\Phi}_m\}$. The gradient of Equation 18 is

$$\frac{\partial l}{\partial \boldsymbol{\omega}} = (r_{upi} - \widehat{r}'_{upi}) \frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\omega}} \quad (22)$$

The gradient of Equation 20 is

$$\frac{\partial l}{\partial \boldsymbol{\omega}} = \frac{\sum_{p=1}^{|T^-|} \sum_{q=1}^{|T^+|} \frac{\partial \sigma(err_{pq})}{\partial \boldsymbol{\omega}}}{|T^+||T^-|} \left(\frac{\partial \widehat{r}_{up_pi_p}}{\partial \boldsymbol{\omega}} - \frac{\partial \widehat{r}_{u_qp_qi_q}}{\partial \boldsymbol{\omega}}\right) \quad (23)$$

where $err_{pq} = \widehat{r}_{up_pi_p} - \widehat{r}_{u_qp_qi_q}$ and $\frac{\partial \sigma(err_{pq})}{\partial \boldsymbol{\omega}} = \sigma(err_{pq})[1 - \sigma(err_{pq})]$. Both the above equations need to compute $\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\omega}}$. According to Equation 2, $\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\omega}}$ is computed as follows

$$\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\theta}_m} = \mathbf{x}_m + 2\lambda_m \boldsymbol{\theta}_m \quad (24)$$

$$\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\theta}_{um}} = \mathbf{x}_{um} + 2\mu_m \boldsymbol{\theta}_{um} \quad (25)$$

$$\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\Phi}_u^k} = \boldsymbol{\Phi}_p \mathbf{x}_p + \boldsymbol{\Phi}_i \mathbf{x}_i + 2\lambda_{u,k} \boldsymbol{\Phi}_u^k \quad (26)$$

Since $\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\Phi}_p^k}$ and $\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\Phi}_i^k}$ have similar form with $\frac{\partial \widehat{r}_{upi}}{\partial \boldsymbol{\Phi}_u^k}$, we do not list them. With gradients with respect to each parameter in $\boldsymbol{\omega}$, $\boldsymbol{\omega}$ is updated according to the following iterative equation

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} - lr * \frac{\partial l(\boldsymbol{\omega}^{(t)})}{\partial \boldsymbol{\omega}^{(t)}} \quad (27)$$

where lr is the *learning rate* that controls how far to move along the descent direction and is set empirically.

4.5 Complexity Analysis

Suppose the dimension of node features and edge features are respectively n_{node} and n_{edge}. The dimension of $\boldsymbol{\Phi}_m$ ($m \in \{u, p, i\}$) is set to $k \times n_{node}$. Since node contains features like user ID and publisher ID, n_{node} can represent the data size. According to edge features introduced in Section 3.2, n_{edge} does not grow with the data size. Thus n_{edge} can be considered as a constant and $n_{edge} \ll n_{node}$.

Space Complexity. All the parameters stay in main memory during the training process. The time complexity of $\boldsymbol{\theta}_m$ ($m \in \{u, p, i\}$) and $\boldsymbol{\theta}_{um}$ ($m, \in \{p, i\}$) is $O(c_1 n_{node} + c_2 n_{edge})$, where c_1 and c_2 represent the number of types of nodes and edges. The space complexity of $\boldsymbol{\Phi}_m$ ($m \in \{u, p, i\}$) is $O(c_1 k n_{node})$. Since training data is read line by line and each line can be dropped immediately after the gradients are calculated, they are not accounted. So the total space complexity is $O(k n_{node} + n_{edge})$. The space grows linearly with the data size.

Time Complexity. First we discuss the time complexity of the training process. Suppose we have n training instances. For each instance, the parameters corresponding to non-zero features are updated. For example, for a user node with user ID=1, only the first column of $\boldsymbol{\Phi}_u$ will be updated. Let F denote the average number of non-zero node features ($F \ll n_{node}$), the time complexity of updating $\boldsymbol{\theta}_m$, $\boldsymbol{\theta}_{um}$ and $\boldsymbol{\Phi}_m$ parameters is $O(c_1 F + c_2 n_{edge} + c_1 kF)$. Since kF dominates $c_2 n_{edge}$, the cost for updating parameters is O(kF). Suppose we need R rounds to converge. The total time complexity is O(nkFR). Since k and F are small constants, the final complexity of training is O(nR). The time complexity of prediction is the same with the complexity of updating parameters, i.e., $O(kF)$. Since both k and F are small constants, the time complexity of prediction is O(1).

5. EXPERIMENTAL STUDY

5.1 Dataset

We crawled Twitter with a breadth-first strategy on the user graph using Twitter's REST API[6]. The dataset in this paper was crawled from April to June, 2012. Each user's latest 3200 tweets[7], profile and following list were crawled. Once a user's following list is crawled, every user on the following list are further crawled. Finally we are able to simulate users' browsing history (i.e., what tweets were received and what tweets were retweeted). All the terms are lowercased and stemmed.

Since users do not have time to see all the tweets, we split the browsing history into sessions to filter the missed tweets. Suppose tweet i has been retweeted by user u. Since there are 20 tweets per page, a session is defined to be a tweet set made up of three parts: (1) tweet i itself (2) fifteen tweets before i (3) five tweets after i. Sessions that have overlaps will be merged into one. Finally, the statistics of our dataset is shown in Table 1.

We split the dataset into training set and test set at time point of May 14th, 2012. The ratio of training set and the test set is about 3:1. The statistics of overlaps between Test Set and Training Set are shown in Table 2. From the table we can see that only 2% tweets in the test set occurs

[6]https://dev.twitter.com/docs

[7]This number is limited by Twitter.

Table 1: Dataset Statistics

Users	Tweets	Sessions	Terms	Tags	Locations
28,420	2,132,533	119,206	554,820	148,476	8,255

Table 2: Overlap between Test Set and Train Set

Users	Tweets	Terms	Hashtags	Locations
93.6%	2%	67%	43%	93.8%

in the training set. This indicates that performing matrix factorization on user-tweet matrix cannot work. The rest of the columns are more suitable for matrix factorization since they are much more denser.

5.2 Evaluation Metric

We use Mean Average Precision (MAP) to measure the performance. Suppose we have a session that contains n tweets. The Average Precision (AP) of this session is calculated by

$$AP = \frac{\sum_{k=1}^{n}(P@k \times isRetweeted(k))}{number\ of\ retweeted\ tweets} \tag{28}$$

where k represents the position from 1 to n, $isRetweeted(k)$ is 1 when the $k{-}th$ tweet is retweeted, otherwise $isRetweeted(k)$ is 0. MAP is the mean of the APs for all sessions.

5.3 Models for Evaluation

Recommendation with Social Regularization (**SocRS**). This method is proposed in [5] to incorporate social networks into traditional matrix factorization. When adapted to our problem, we minimize the following loss function:

$$\min_{U,V} L_2(R,U,V) = \frac{1}{2}\sum_{i=i}^{m}\sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\beta}{2}\sum_{i=i}^{m}\sum_{f\in F^+(i)}^{n} Sim(i,f)||U_i - U_f||^2 + \lambda_1||U||^2 + \lambda_2||V||^2 \tag{29}$$

where β is weight of social factors, U is the set of latent vectors for users and publishers, $F^+(u)$ is the set of friends of u, $Sim(i,f)$ is the similarity between u_i and u_f. u_i and u_f are represented as a vector of their retweeted tweets, respectively. $Sim(i,f)$ is defined to be the Jaccard similarity between u_i and u_f. The final rating $\widehat{r}_{upi}$ is $U_i^T U_p$. The method assumes that friends tend to have similarity interests, thus their latent vector U_i and U_f should be similar. This baseline is used to prove that matrix factorization on user-tweet matrix will not work even when social networks are incorporated to address the sparsity of user information.

Non-personalized Retweet Prediction (**RP**). [3] and [7] explored tweet features and publisher features to predict whether a tweet will get retweeted regardless of which user retweets it. This baseline can be viewed as a special form of our model where only tweet features and publisher features are considered. This baseline is used to prove the need for personalization.

Feature-based Model (**Feat**). This method only considers hand-crafted node and edge features and thus is a simplified version of our method. $\mathbf{g}_u^T\mathbf{g}_p$ and $\mathbf{g}_u^T\mathbf{g}_i$ are ignored in Equation 1. This method is used to measure the contributions from hand-crafted features.

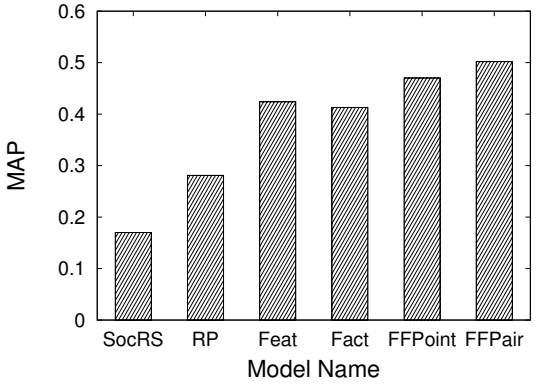

Figure 2: Comparision of all models

Table 3: Comparison of all models

Model	SocRec	RP	Feat	Fact	FFPoint	FFPair
MAP	0.176	0.281	0.424	0.413	0.47	0.502

Factorization-based Model (**Fact**). This method only considers automatically learned latent biases and vectors. Thus this is another simplified version of our method. All the hand-crafted feature are ignored. Only features corresponding to latent biases and latent vectors are retained in Equation 2. This method is used to measure the contributions from the factorization part.

Feature-aware Factorization Model with Point-wise Loss (**FFPoint**). This is our proposed model with point loss function. Compared with $Feat$ and $Fact$, this model can prove the advantages of combining feature-based model and factorization-based model.

Feature-aware Factorization Model with Pair-wise Loss (**FFPair**). This method is used to prove the advantages of using pair-wise loss function. Compared with point-wise loss function, pair-wise loss function further employs the session context information.

All the experiments were conducted on a server with Intel Xeon E5405 2.00GHz CPU and 10G memory. The algorithms are implemented in Java with the support of matrix library jblas[8] for fast matrix/vector manipulation.

5.4 Overall Results

The overall results are shown in Figure 2 and Table 3. First we analyze why $SocRS$ and RP cannot solve the problem. Recall that according to Table 2, only 2% tweets in the test set also exist in the training set. So it is not surprised $SocRS$ has the the worst performance. By using publisher features and tweets features, RP is able to outperform $SocRS$. However, its performance is much worse than the personalized model. This indicates that the interestedness of a tweet varies from user to user. Only considering publisher's authority and tweet's quality is not enough. Personalization plays an important role in the retweet behavior.

Now we discuss some indications by comparing $Feat$ with $Fact$. With hand-crafted user-publisher features and user-tweet features, $Feat$ has a big improvement over RP. A major difference between PTR and traditional matrix factorization models can be found here: PTR has rich features

[8]http://jblas.org/

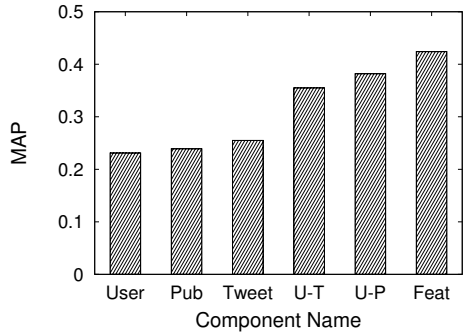

Figure 3: Contribution of each component in Feat

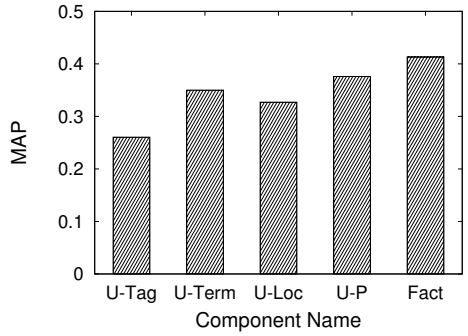

Figure 4: Contribution of each component in Fact

to indicate the similarity between a user and an item[9] while traditional matrix factorization only has a user-item rating matrix. In fact, the MAP of $Feat$ is even slightly higher than $Fact$. However, this does not indicate $Feat$ is better than $Fact$. The main advantage of $Fact$ is that it does not need any feature engineering work. All the latent biases and vectors are automatically learned from the data. The similarity between a user and a publisher is directly calculated by $\mathbf{g}_u^T\mathbf{g}_p$. So is for user-tweet similarity. Moreover, the improvement of $Feat$ over $Fact$ is quite marginal.

Finally, we analyze the results of $FFPoint$ and $FFPair$. By combining features with factorization models, the MAP of $FFPoint$ is further improved. This indicates that feature-based model and factorization-based model can be complementary to each other. Comparing $FFPair$ with $FFPoint$, we can find that the MAP is improved again by replacing point-wise loss function with pair-wise loss function. Since the loss is minimized according to each session instead of each single tweet, the pair-wise loss function is able to catch the context of user's choice. Thus using pair-wise loss function can lead to a better performance.

5.5 Contribution of Each Component

To measure the contribution of each component, we compare models trained for each component of $Feat$ and $Fact$.

Components of Feat. According to Equation 3, $Feat$ is made up of node features and edge features. The performance of models corresponding to each component is shown in Figure 3, where 'U-T' and 'U-P' represent user-tweet edge features and user-publisher edge features, respectively. Since node features does not consider personalized information and are mapped to basic biases, each of them has relative poor performance when used alone. On the contrast, user-tweet features and user-publisher features play an important role in this model. User-publisher features are more effective than user-tweet features. Since a tweet has only up to 140 characters, it is difficult to find explicit features to measure the interestedness of the tweet to the user. By resorting to user-publisher features, the user's preference toward the tweet is more predictable. Finally, when all features are combined, we get the $Feat$ model analyzed in Section 5.4.

[9]Items are tweets and publishers in our setting.

Table 4: Training Time per Tweet (ms)

Loss Func	Feat	U-P	U-Term	FeatFact
Point-wise	0.34	0.46	11.40	13.12
Pair-wise	1.02	0.67	13.43	16.08

Table 5: Predicting Time per Tweet (ms)

Feat	U-P	U-Term	FeatFact
0.18	0.16	2.58	3.22

Components of Fact. According to $\mathbf{g}_u^T\mathbf{g}_i$ defined in Equation 11 and $\mathbf{g}_u^T\mathbf{g}_p$ defined in Equation 12, four relations are factorized in $Fact$: user-hashtag, user-term, user-publisher, and user-location relation. When trained alone, the performance is shown in Figure 4, where 'U-Tag', 'U-Term', 'U-Loc' and 'U-P' represent user-tag, user-term, user-location, and user-publisher relation, respectively. According to Table 2, the overlap of hashtags in training set and test set is the smallest. Thus about half of the latent biases and latent vectors of the hashtags cannot get trained. Thus the contribution of user-tag relation is the smallest. user-term and user-location relation are proved to be more effective. This indicates users tend to prefer tweets of some certain topics and areas. Finally, like the 'U-P' component in $Feat$, user-publisher relation in $Fact$ is also proved to be the most effective component. Although overlaps of users and locations in the training set and test set are both very high, user-publisher relation is considered as a finer-grained interaction than user-location relation. Thus user-publisher relation is more effective than user-location relation.

5.6 Efficiency Issue

In this section, we first analyze the training time and predicting time of each component in Equation 2. Then we compare the time cost of point-wise loss function and pair-wise loss function. The training time and prediction time of each component are shown in Table 4 and Table 5, respectively. 'Feat' represents the feature-based component made up of f_m and f_{um} in Equation 2. 'FeatFact' is the complete model defined by Equation 2. Since 'U-Loc' and 'U-Tag' have similar training time and predicting time with 'U-P', we do not list them in the tables.

Efficiency of Each Component Comparing training time of different components in the 'point-wise' row, we find that feature-based component and 'U-P' are much more efficient than 'U-Term'. Since a tweet can contain up to 140 different terms but only one publisher, the training time and predicting time of 'U-Term' is 1-140 times faster than 'U-P'. Although 'U-Term' has dominated most of the time and seems to be costly, the final predicting time of 'FeatFact' is 3.22 ms per tweet. This is considered to be fast enough for online response. Note that all the original terms are considered in our model. Since a tweet may contain many meaningless terms, keyword extraction or simple TF-IDF based filtering

can be performed to find out potentially important terms to represent the tweets. Once the tweet is shortened, the efficiency will be further improved.

Point-wise Loss vs. Pair-wise Loss. The predicting time of point-wise approach and pair-wise approach are the same since they both use Equation 2 for prediction. So we only compare the training time. Comparing the 'Point-wise Loss' row with 'Pair-wise Loss' row in Table 4, we can find that pair-wise approach is slower than point-wise but the difference is quite small. Suppose a session contains two positive tweets and eight negative tweets. For point-wise approach, the loss is calculated on ten tweets. However, for pair-wise approach, the loss is calculated on each pair of positive tweets and negative tweets, i.e. $2 \times 8 = 16$ tweets. In our dataset, most sessions only contain one or two positive tweets. Thus pair-wise approach is just slightly slower than the point-wise approach.

6. RELATED WORK

Many work have been done in studying retweet behavior in the macro perspective [2, 3, 7, 8, 12]. Boyd[2] studied some basic issues about retweet behavior: how people retweet, why people retweet and what people retweet. He found that retweet provides a way to let users make conversations with each other. Hong[3] studied how to predict the popularity of messages measured by the number of future retweets. In their work, content features, temporal information and metadata of tweets and publishers are explored. Suh[12] and Petrovic[7] explored tweet features like URLs and hashtags, and publisher features like followers/followees count and account age. Compared with our model, these work tried to find useful node features to predict whether a tweet will be retweeted regardless of who will retweet the tweet. To the best of our knowledge, personalized tweet re-ranking has very limited work. [6] and [13] are considered to be most relevant to our problem. Macskassy[6] claimed that the majority of users do not retweet that tweets similar to their own tweets. When content similarity is considered, the predicting performance can be improved. Uysal[13] further explored user-publisher and user-tweet features. Both of them belongs to pure feature-based approach and mainly focus on finding hand-crafted node features and edge features. Compared with our model, the automatically learned latent biases and vectors are dropped. Thus they are very similar to our baseline 'feature-based model'.

In terms of feature-aware factorization models, [1] and [9] are considered to be most relevant. Agarwal[1] proposed a regression-based prior for the latent vectors, which has similar form with our definition of g_m. The regression-based prior in [1] is mainly based on numeric features. In our work, the latent vectors are learned for each categorical features so that more relations can be incorporated in the factorization model. The practical meaning is quite different. Rendle[9] proposed a general factorization machine, which performs all pair-wise interactions between node features. However, the general form does not exist in our problem. Publisher-tweet interaction is empirically found to have little connection with whether user u will retweet tweet i.

7. CONCLUSIONS

In this paper, we proposed a novel problem called personalized tweet re-ranking. We modeled retweet behavior as a graph made up of users, publishers and tweets. To incorporate all sources of information, nodes and edges are represented by feature vectors. According to the graph model, we designed feature-aware factorization model that can fully explore all the information in the graph for prediction. We aim to propose a general prediction framework. Like SVM for classification task, users only need to specify the node features and edge features. Our feature-aware factorization model will build a predicting model based on the features.

8. ACKNOWLEDGMENTS

This work was supported in part by National Natural Science Foundation of China under Grant No. 61272088 and 60833003, National Basic Research Program of China (973 Program) under Grant No. 2011CB302206. This work is partially done when the authors visited SA Center for Big Data Research hosted in Renmin University of China. This Center is partially funded by a Chinese National "111" Project "Attracting International Talents in Data Engineering and Knowledge Engineering Research".

9. REFERENCES

[1] D. Agarwal and B.-C. Chen. Regression-based latent factor models. In *KDD*, pages 19–28, 2009.

[2] D. Boyd, S. Golder, and G. Lotan. Tweet, tweet, retweet: Conversational aspects of retweeting on twitter. In *HICSS*, pages 1–10, 2010.

[3] L. Hong, O. Dan, and B. D. Davison. Predicting popular messages in twitter. In *WWW (Companion Volume)*, pages 57–58, 2011.

[4] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[5] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[6] S. A. Macskassy and M. Michelson. Why do people retweet? anti-homophily wins the day! In *ICWSM*, 2011.

[7] S. Petrovic, M. Osborne, and V. Lavrenko. Rt to win! predicting message propagation in twitter. In *ICWSM*, 2011.

[8] D. Ramage, S. T. Dumais, and D. J. Liebling. Characterizing microblogs with topic models. In *ICWSM*, 2010.

[9] S. Rendle. Factorization machines with libfm. *ACM TIST*, 3(3):57, 2012.

[10] S. Rendle. Learning recommender systems with adaptive regularization. In *WSDM*, 2012.

[11] D. M. Romero, W. Galuba, S. Asur, and B. A. Huberman. Influence and passivity in social media. In *ECML/PKDD (3)*, pages 18–33, 2011.

[12] B. Suh, L. Hong, P. Pirolli, and E. H. Chi. Want to be retweeted? large scale analytics on factors impacting retweet in twitter network. In *SocialCom/PASSAT*, pages 177–184, 2010.

[13] I. Uysal and W. B. Croft. User oriented tweet ranking: a filtering approach to microblogs. In *CIKM*, 2011.

[14] J. Yang and S. Counts. Predicting the speed, scale, and range of information diffusion in twitter. In *ICWSM*, 2010.