Learning to Rank Social Update Streams

Liangjie Hong[*] [†], Ron Bekkerman[§], Joseph Adler[§], Brian Davison[†]
[†] Dept. of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
[§] LinkedIn Corp., Mountain View, CA, USA
{lih307,davison}@cse.lehigh.edu, {rbekkerman,jadler}@linkedin.com

ABSTRACT

As online social media further integrates deeper into our lives, we spend more time consuming social update streams that come from our online connections. Although social update streams provide a tremendous opportunity for us to access information on-the-fly, we often complain about its relevance. Some of us are flooded with a steady stream of information and simply cannot process it in full. Ranking the incoming content becomes the only solution for the overwhelmed users. For some others, in contrast, the incoming information stream is pretty weak, and they have to actively search for relevant information which is quite tedious. For these users, augmenting their incoming content flow with relevant information from outside their first-degree network would be a viable solution. In that case, the problem of relevance becomes even more prominent.

In this paper, we start an open discussion on how to build effective systems for ranking social updates from a unique perspective of LinkedIn – the largest professional network in the world. More specifically, we address this problem as an intersection of learning to rank, collaborative filtering, and clickthrough modeling, while leveraging ideas from information retrieval and recommender systems. We propose a novel probabilistic latent factor model with regressions on explicit features and compare it with a number of non-trivial baselines. In addition to demonstrating superior performance of our model, we shed some light on the nature of social updates on LinkedIn and how users interact with them, which might be applicable to social update streams in general.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Theory

[*]Part of this work was done when the first author was on an internship at LinkedIn Corp.

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.

Keywords

Social network, Social Stream, Learning to rank, Collaborative filtering

1. INTRODUCTION

In the current Web ecosystem, social media services are ubiquitous. With the rise of websites like Facebook, Twitter, and LinkedIn, millions of users are connecting and communicating with each other over rich multimedia content, including text, images, video, and audio. The content forms streams of social updates, which allow users to get instantly informed on the latest news. When aggregated, the social update streams become a powerful tool for monitoring the geopolitical situation in various regions. For instance, social update streams have been heavily used to disseminate information during the "Arab spring". In addition, social update services become the ultimate platform for collecting information in natural disasters, such as earthquakes and tsunamis [20].

Although social update streams provide a unique opportunity for users to obtain fresh information, users commonly acknowledge two issues that prevent current social streams from being sufficiently relevant, which causes deterioration of user experience and engagement. First of all, while facing a large number of updates from their social connections, users simply cannot consume them in an effective and efficient way. This is known as the problem of *information overload* (see, e.g., [3, 12]). Furthermore, social updates for a user are usually limited in scope to their social circle, induced from their connections. Thus, it is very difficult for a user to obtain information distributed outside of their circle, even though it might match their interests. In order to obtain relevant information, users spend long hours searching social media (see, e.g., [2, 21]). We call this problem *information shortage*. To address both these problems, social media monitoring systems are being built, which filter and recommend social updates to users based on numerous signals. This area has recently attracted close attention of academic and industrial research communities.

The task of filtering and recommending social updates can be approached from various perspectives. From the Information Retrieval (IR) perspective, constructing personalized social streams can be cast into the classic ranking problem: the task is to rank social updates by descending order of user interest. It may be true that some existing IR techniques could be potentially applied to social stream ranking. However, user's interests in social streams are not represented in terms of a search query. Instead, queries are implicit and have to be inferred. The absence of a search query distinguishes the social stream ranking problem from many classic IR tasks. In addition, social information needs are more diversified compared to traditional IR scenarios. Although traditional IR tools do not appear to be directly applicable to ranking in social streams,

some of recently developed *learning to rank* approaches are very appealing to be used in the new setting.

From the perspective of Recommender Systems (RecSys), building a list of relevant social updates can be viewed as recommending relevant items to users. Thus, many collaborative filtering techniques are applicable to the task of social stream ranking. However, social stream systems are much more dynamic than traditional recommender systems: many new updates can be pushed into the system every second. Therefore, the cold start problem becomes even more severe in social stream systems. The traditional collaborative filtering paradigm needs to be adjusted to ranking social streams.

Surprisingly, little prior research work has been done to tackle the problem of social stream ranking from the point of view of building an effective system. This is partially due to the fact that no real-world dataset of social updates is openly available to the research community – due to obvious reasons related to user privacy. Most commercial ranking algorithms (e.g., the one used by Facebook) are proprietary. Indeed, a successful content ranking system on social streams will not only provide more relevant information to users and improve user engagement, but also shed the light on patterns of user behavior and social trends, which might be strong signals for behavioral targeting in computational advertising – the driving power of most Web 2.0 venues.

In this paper, we start an open discussion on how to build effective systems for social stream ranking (SSR). To the best of our knowledge, we are the first group of researchers to elaborate technical details for such a system. More specifically, we address the problem as an intersection of learning to rank, collaborative filtering and clickthrough modeling, while leveraging ideas from information retrieval and recommender systems. Our contributions are three-fold:

1. Analyze social streams (based on LinkedIn data) and provide some insights on users' behavior;

2. Propose a novel probabilistic latent factor model with regressions on explicit features; integrate the idea of learning to rank and collaborative filtering

3. Demonstrate the superior performance of our model by comparing with several non-trivial real-world baselines.

The rest of the paper is organized as follows. In Section 2, we compare SSR with other related research areas. As we will point out, SSR is a unique setup to which existing techniques cannot be applied directly. In Section 3, we review the problem of social stream ranking in the context of LinkedIn. In Section 4, we introduce our ranking model step by step and compare it with some existing IR and collaborative filtering techniques. In Section 5, we evaluate our model against a number of non-trivial baselines. We conclude our paper in Section 6.

2. RELATED WORK

In this section, we briefly overview three research directions related to social stream ranking: (a) learning to rank, (b) recommender systems, and (c) clickthrough models. Some of the approaches to tackle these problems are relevant to SSR and can be adapted. Along with their similarities to SSR, we also reveal their significant differences from SSR, and discuss the uniqueness of SSR as a new research field.

Learning to Rank (LtoR): In IR, a generic task is to construct a ranked list of documents relevant to a query issued by a user. Although ranking is a fundamental problem in IR and has been studied for decades, it still remains challenging. Instead of proposing carefully designed ranking models based on heuristics or traditional probabilistic principles, a recent trend is to apply machine learning techniques to learn ranking functions automatically, i.e., LtoR [19]. In the standard LtoR setting, a typical training set consists of queries with their associated documents represented by feature vectors as well as corresponding relevance judgements. A machine learning algorithm is employed to learn the ranking model, which can predict the ground truth label in the training set as accurately as possible – in terms of a loss function. In the test phase, when a new query comes in, the learned model is applied to sort the documents according to their relevance to the query, and return the corresponding ranked list to the user as the response to the query. Depending on different hypotheses, input spaces, output spaces and loss functions, approaches to LtoR can be loosely grouped into three categories: point-wise, pairwise, and list-wise.

Although the goal of LtoR is to provide a ranked list of documents (items) for users – a similar aim as of SSR – it is different from SSR in three aspects. First, social stream systems usually do not have explicit queries and users do not have to specify any explicit input in order to obtain relevant output from the systems. Second, each user's social stream is highly dependent on their social context. Therefore, compared to IR, social streams are intrinsically personalized. The second difference leads to the third fundamental distinction between SSR and LtoR: relevance judgements are difficult to obtain in SSR and the notion of relevance can be hard to define. Nevertheless, some strategies and insights developed in LtoR can be borrowed for SSR.

Recommender Systems (RecSys): As we will see, RecSys plays a key role in many online services, improving user experience and engagement. In the simplest form, RecSys aim to present a user with a list of items in the hope that these items would match the user's interests to some extent. Content-based methods and neighborhood methods are widely used in RecSys. Content-based methods convert the problem of recommendation into an IR problem by constructing user profiles and item profiles. A user profile serves as a query to an index of item profiles. Similarity measures (e.g., cosine similarity, Jaccard coefficient) are utilized to match users and items. In contrast, neighborhood methods usually explore the notion of *topical locality*, assuming that the interaction between users and items can be predicted solely upon observations of "neighboring" users or items. That is, a user's interest in an item is approximated by the average of neighboring observations. Although content-based methods and neighborhood methods are popular due to their simplicity, they cannot exploit hidden interactions between users and items. Recently, another class of methods called *latent factor models* has gained increasing attention. These methods are highly accurate and can easily incorporate various biases. However, compared to content-based methods and neighborhood methods, latent factor models are more vulnerable to the appearance of new items and new users, i.e., to the cold-start problem. Therefore, these three approaches are complementary to each other in practice (e.g., [17]). In a general sense, SSR may be considered as RecSys, however, SSR usually does not have item ratings, and user feedback to SSR is often implicit.

Click-through Model (CM): both IR (see, e.g., [14, 9]) and RecSys [25, 31] researchers have noticed that users' feedback is vital for learning a high-quality model. In order to derive users' preferences and model users' clickthrough data, a variety of CMs have been proposed. The most common approach to clickthrough modeling is to construct a generative model aiming to explain the training data (see, e.g., [33, 13]). Other models that derive users'

preferences have been proposed as well (see, e.g., [25, 5, 31, 18]). While generative models are specifically designed to understand clickthrough data, it is difficult to incorporate them into current IR or RecSys frameworks, partially due to the fact that generative models are hard to adapt to optimizing a non-probabilistic objective. Indeed, it is easier to first obtain user preferences from clickthrough data analysis and then adapt existing IR/RecSys tools to using these preferences (e.g., [25, 5, 31]). Our paper is inspired from this idea.

In addition to these three directions, some efforts have been made to directly tackle the problem of SSR. For instance, Chen et al. [8] discussed the problem of recommending content on Twitter by considering many dimensions, including content sources, topic interests of users and social voting. However, their study focused on empirical validations of several features (signals) and the dataset used is significantly smaller than ours. Their later work [7] goes beyond single Tweet recommendation to conversation recommendation. Duan et al. [11] noticed that the ranking problem of Twitter can be treated as an application of learning to rank. Their dataset is also small and relationships between recipients and senders are not explored. As we have discussed, SSR is not simply a LtoR problem. Choudhury et al. [10] argued that SSR should consider the problem of "diversity" and they tested their greedy algorithm on 67 employees from a large technology corporation. Our work differs from all this related work in three significant ways: 1) we test our proposed method on a large-scale, real-world dataset; 2) we propose a principled way to address SSR in the context of LtoR, CF and CM; and 3) we conduct comprehensive evaluation of our model against several models that underlie state-of-the-art recommender systems and report a consistent improvement in performance.

3. OVERVIEW OF LINKEDIN

Founded in December 2002 and launched in May 2003, LinkedIn[1] is primarily used for online professional networking. As of March 2012, LinkedIn has more than 160 million registered users in more than 200 countries and territories. On LinkedIn, user profiles play a central role for establishing professional existence and personal brand. Users can update their professional profiles to include a spectrum of content types (e.g., position descriptions, publications, patents, open source projects, skills, etc.). In addition, LinkedIn offers collaborative platforms to help users consume relevant news stories (e.g., LinkedIn Today[2]), seek answers to questions on professional issues (e.g., LinkedIn Groups[3]), and share useful content (e.g., LinkedIn Signal[4]). On the left-hand side of LinkedIn's homepage, a typical user will see a list of content items that come from their professional connections. This update stream consists of a wide range of types of updates including changes on their profiles (e.g., changes in their employment), shares of information (e.g., news articles, blog posts), and Twitter updates. These updates compose a social stream for the user. A snapshot of a user's homepage is shown in Figure 1.

As we have discussed in previous sections, delivering truly relevant social updates to users is a very difficult task. Information overload is certainly a serious problem for users who have hundreds of connections. In contrast, for newly joined users who do not have a sufficient number of connections, a system could recommend potentially useful updates to make the user adapt to the service more smoothly and quickly.

[1]http://www.linkedin.com
[2]https://www.linkedin.com/today/
[3]https://www.linkedin.com/myGroups
[4]https://www.linkedin.com/signal/

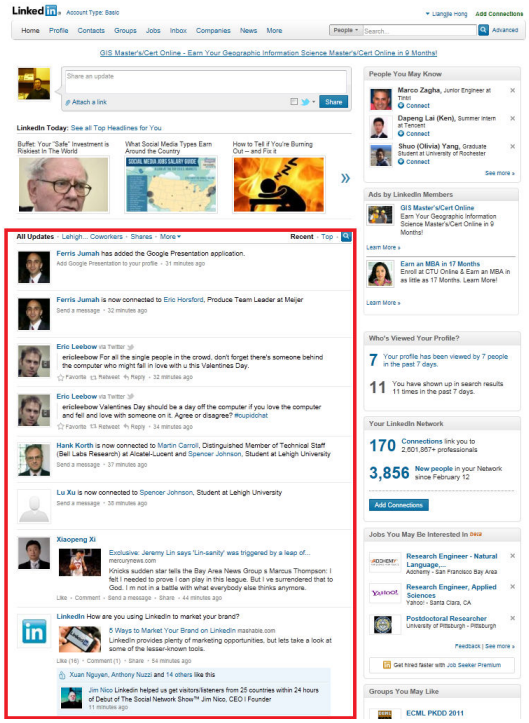

Figure 1: A typical example of LinkedIn homepage where the social stream of a user is highlighted by a red rectangle, shown on the left hand side of the screen.

From the LinkedIn point of view, it is important to attract users to consume their social content and interact with their social streams as it is a clear indicator of a healthy engagement pattern. A steady, but badly delivered social stream may distract the user and make them lose interest in the service. A weak social stream for new users may make them question the benefits of the service. On the other hand, if a user interacts with the social stream frequently, clicking "Like" buttons or making comments to others' updates, the user is likely to become more and more engaged.

4. SOCIAL STREAM RANKING

In this section, we will discuss the ingredients of our proposed model step by step, from a simple linear model to a much more involved tensor factorization based latent factor model. We start our discussion from why the problem of SSR cannot simply be treated as a rating prediction problem, which is a classic setting in RecSys.

4.1 Evaluation Metrics

In traditional RecSys settings where the entities are users and items (e.g., the famous Netflix competition[5]) forming a matrix of users by items, the main goal is to predict or recover missing values in this matrix – if we treat existing ratings as observed values and non-existing ratings as missing values of the matrix. The performance of recommender systems is evaluated by how accurately a system can predict these values. For instance, in the Netflix Prize, Rooted Mean Squared Error (RMSE) is used to measure the accuracy of rating predictions. RMSE is defined as $\sqrt{\frac{1}{N}\sum_i^n (x_i - \hat{x}_i)^2}$ where x_i is the ground-truth rating for rating i, $\hat{x}_i$ is the predicted value and N is the total number of ratings to be tested. Although it

[5]http://www.netflixprize.com/

might be an appropriate evaluation metric for movie recommendation tasks where multiple levels of ratings are available, two crucial issues will arise while using it as an evaluation metric to SSR.

First of all, as we see in Figure 1, the final presentation of a social stream is a list of items shown on the computer screen. Due to the limited space on the screen, users can only see a portion of the list which usually only consists of a handful of updates. Although users can always scroll down the list and even go to additional pages, not all of them do that in practice. Thus, even if the accuracy of predictions is important, we certainly wish to have higher-accuracy items on the top of the list rather than to have the whole list slightly more accurate. This ordering information cannot be easily captured in accuracy-based metrics, like RMSE. In addition to the reason that accuracy-based metrics may not be appropriate, in the practical sense, a system optimizing accuracy might fail to produce reasonable results. As we will see, the users rarely interact with the *majority* of the updates. In other words, users are only interested in a small number of items. An accuracy-optimized system may overfit non-interacted items and yield good performance overall but might not match the small portion of clicked updates. Indeed, this drawback of accuracy-based metrics is also discussed in the collaborative filtering literature (e.g., [25, 18, 31]).

Based on this discussion, we adopt rank-based metrics to evaluate the effectiveness of SSR systems. Rank-based metrics are widely used in the IR community. In this paper, we borrow Mean Average Precision from the traditional IR. We define the "precision" at position k (Precision@k) of a social stream as $(\#\text{ of clicks in top positions})/k$. Then, an average measure across all top m positions (Average Precision) for user u is defined as $(\sum_{k=1}^{m} \text{Precision@k} \times l_k)/(\#\text{ of clicks for ranked list of user } u)$ where l_k is a binary indicator whether the position k has been clicked or not and m is the total number of positions evaluated. Note that the Average Precision is evaluated per user. We can average it across all users, resulting in the Mean Average Precision (MAP) measure.

4.2 Dataset

Before discussing the details of our dataset, we introduce the concept of *impression*. Every time a user logs in LinkedIn's homepage, the system generates a list of candidate social updates from many sources, mainly based on the user's social connections. This list of updates can be small or large, depending on the user's social circle. If the number of updates in the list exceeds a certain threshold (e.g., $10 - 15$), these updates cannot be shown on a single screen, and the user will need to scroll down. An impression is a list of updates that a user has actually seen on the screen. Given historical data, we can "replay" the users' activity while analyzing impressions. Note that social updates are not distinct: one specific update produced by a user or a company can be shown to many users.

Since our experimental setting is "simulation" (details will be discussed in Section 4.1), we discard all impressions without any clicks because these impressions do not change our experimental results (as measured by MAP). Note that there is a deeper argument on this decision. Remember that one issue associated with social streams is their sparsity. Indeed, only a small number of impressions attract users and a handful of clicks is performed on such impressions, compared to the large amount of impressions produced in total. Thus, it might be useless for any model to fit these non-interacted impressions. Focusing on impressions that actually matter reduces the training set significantly and produces better results, which we saw in our empirical studies. Thus, only impressions with at least one click remain in our dataset. In our experiments, we also filter out impressions with less than five items.

Data Summary	April, 2011	September, 2011
Impressions	3M-4M	10M-20M
Updates	30M-40M	100M-200M
Clicked Updates	3M-4M	10M-20M
Non-clicked Updates	27M-36M	90M-180M
Distinct Updates	10M-20M	20M-30M
Recipients	1M-2M	4M-5M
Producers	4M-5M	6M-7M

Table 1: The basic statistics about the dataset."M" means million. The numbers are obfuscated due to commercial reason.

We report on two datasets of LinkedIn's social update stream. Both are subsamples of the actual social stream collected by LinkedIn's engineering team. The first dataset was taken from April, 2011 stream, while the second was taken from September, 2011 stream. The basic statistics on the two datasets are shown in Table 1. The reason why we take two datasets that are not consequent in time is to demonstrate that the performance of different algorithms is indeed consistent over different time periods.

4.3 Linear Models

In this section, we will discuss several simple linear models to tackle the problem of SSR. Given a social update, a user can choose to respond to this update or not. For simplicity, we treat all kinds of responses as a "click" event and no response as a "non-click" event. Thus, we focus on binary responses in this work. We denote $\mathbf{y}$ as a vector of responses to all social updates, across all impressions. This way, we concatenate updates from all impressions together and drop the notion of an impression. The ordering of elements in $\mathbf{y}$ does not matter as we only care about the correspondence: the response y_i corresponds to the i-th update in the entire dataset and f_i represents the estimation of y_i from the models described later. In addition, we define the following auxiliary functions: $r(i)$ is the recipient of update i, $s(i)$ is the sender of update i, $t(i)$ is the type of update i, and $c(i)$ is the sender type. Let $\mathcal{R}$ be the set of recipients, $\mathcal{S}$ be the set of senders, $\mathcal{T}$ be the set of types, $\mathcal{I}$ be the set of social updates.

Feature Model (FM): One straightforward model is linear estimation, which predicts the response by a linear combination of features. For a specific update i, we collect their corresponding features. Let $\boldsymbol{\phi}$ be a feature vector – we use subscript to indicate its corresponding type. For instance, $\boldsymbol{\phi}_{r(i)}$ represents the feature vector (e.g., profile) for the recipient user of update i. A simple prediction of y_i – a linear combination of user features and update features – can be defined as:

$$f_i^{(1)} = \boldsymbol{\beta}_{r(i)}^T \boldsymbol{\phi}_{r(i)} + \boldsymbol{\alpha}_{r(i)}^T \boldsymbol{\phi}_i \tag{1}$$

where $\boldsymbol{\beta}_u$ and $\boldsymbol{\alpha}_u$ are per-user coefficients to be learned from the training set. This model is essentially equivalent to the one where user features and update features are combined into a single feature vector and a per-user coefficient to be learned. Note, an even more simpler model could be also considered where a universal coefficient is used, instead of per-user coefficients. However, this model would be too restricted and personalization cannot be applied.

Latent Bias Model (LBM): Here, we introduce a linear model to explain the clicks on items. We start with an assumption that whether a new item i will be clicked by a user depends on the average click rate: $f_i = \mu$ where μ is the average click rate across all items and all users. Certainly, this estimation is too coarse and

inaccurate because, as we mentioned before, the majority of items are not clicked. We can extend this base estimation by incorporating a wide range of biases: 1) type (category) bias, 2) item bias, 3) recipient bias, 4) sender type bias and 5) sender bias. Adding these biases is very intuitive. Certain types of updates (e.g., notifications about changes in user profiles, including changes in job titles etc.) receive more attentions than others. Users tend to respond (e.g., click "Like" button or make comments) more often on these types of updates than on others. In addition, some individual items are more popular than others as their content might be more interesting (e.g., breaking news, unexpected stories). From the perspective of senders and recipients, biases are also significant. For instance, some updates are coming from companies that inform their followers about their new products and services – those updates are far more popular than status updates from individual users. Moreover, certain users are more engaged than others which introduces user biases. Therefore, all these biases (i.e., prior knowledge) can capture a wide range of effects of interactions. Let b denote biases and the subscript to indicate the type of biases. Also, the subscript can be an index for a feature. Therefore, we can have the following linear estimation:

$$f_i^{(2)} = \mu + b_i + b_{t(i)} + b_{r(i)} + b_{c(i)} + b_{s(i)} \tag{2}$$

Note that these biases are generally unknown. We treat them as latent variables to be learned from the dataset. Comparing with LR, which depends on feature vectors some of which might be difficult to calculate and update (e.g., graph-based features, content based features), this model is appealing since no extra information is needed for learning, besides requiring indicators.

Combining FM and LBM: It is straightforward to consider combining FM and LBM together. Thus, the combined model will enjoy the freedom that different parts of the model will explain a variety of user behaviors. The combined model is simply:

$$f_i^{(3)} = f_i^{(1)} + f_i^{(2)} \tag{3}$$

Note, this is essentially a linear feature model with biases decomposed into many aspects.

Incorporating Temporal Effects: Social streams are temporally sensitive in nature. Users focus on fresh information and interact with them while they often do not bother to look at old updates. We model the temporal effects of updates by a simple feature: $t_{\text{recency}} = t_{\text{imp}} - t_{\text{upt}}$ where t_{imp} is the numerical time when a user sees a particular impression and t_{upt} is the numerical time when an update is produced. These feature values would be different for different updates. Even for the same update, depending on when recipients access their update streams, the feature value can vary greatly. Incorporating this feature into our estimation can be as follows:

$$f_i^4 = f_i^{(*)} + \zeta \times t_{\text{recency}} \tag{4}$$

where $f_i^{(*)}$ can be any estimator defined in Equations (1, 2 and 3) and ζ is a free parameter, indicating the importance of recency of updates. Note that ζ can be learned from the training set and can also be treated as another personalized parameter. However, we fix it across all users and manually tune this parameter mainly because of two reasons. Firstly, since not all users interact with social streams regularly and new users are coming all the time, we need to provide reasonably relevant social updates for these users. Under these circumstances, explicit features might not be available (e.g., new users) and biases are not learned reliably from the training set (e.g., not enough interactions before). Therefore, a safe choice is to rank items chronologically. In addition to that, users who are heavily engaged with their social streams are familiar with existing ranking schemes which are primarily based on recency. We do not want to suddenly change their expectations on their future impressions. Hence, we set ζ such that the temporal effect can always be an important factor and indeed the learned coefficients and biases will dominate the estimation only when it is necessary.

Since responses are binary, we impose a logistic loss on predictions and true values, yielding learning procedures of the logistic regression flavor. All these linear models can be learned effectively by minimizing the following objective function for each update i:

$$l_1(y_i, f_i^{(*)}) = \log\Big[1 + \exp(-y_i f_i^*)\Big] \tag{5}$$

where $y_i \in \{\pm 1\}$ is the ground-truth response and f_i^* is the estimated response from the models defined above. In common practice, in order to avoid overfitting the training set, we also use L_2 regularizer to shrink all parameters towards zero. Taking Equation (4) as an example, the final objective function is:

$$L_1 = \sum_i l_1(y_i, f_i^4) + \lambda_1 \Big(\sum_i ||b_i||^2 + \sum_{t(i)} ||b_{t(i)}||^2 + \sum_{c(i)} ||b_{c(i)}||^2 + \sum_{r(i)} ||b_{r(i)}||^2 + \sum_{s(i)} ||b_{s(i)}||^2 \Big) + \lambda_2 \sum_u \Big(||\boldsymbol{\beta}_u||_F^2 + ||\boldsymbol{\alpha}_u||_F^2 \Big)$$

where λ_1 and λ_2 are two regularization parameters to be manually tuned. Many methods are available to optimize the objective function above. Here, we adopt the Stochastic Gradient Descent (SGD) method, a widely used learning method for large-scale data, to learn parameters. SGD requires gradients, which can be effectively calculated as follows:

$$\frac{\partial L_1}{\partial b_*} = -\sum_i \Big[1 - \sigma(y_i f_i^4)\Big] y_i + 2\lambda_1 \sum_* b_*$$

$$\frac{\partial L_1}{\partial \boldsymbol{\beta}_{r(i),k}} = -\sum_i \Big[1 - \sigma(y_i f_i^4)\Big] y_i \boldsymbol{\phi}_{r(i),k} + 2\lambda_2 \boldsymbol{\beta}_{r(i),k}$$

$$\frac{\partial L_1}{\partial \boldsymbol{\alpha}_{i,k}} = -\sum_i \Big[1 - \sigma(y_i f_i^4)\Big] y_i \boldsymbol{\phi}_{i,k} + 2\lambda_2 \boldsymbol{\alpha}_{i,k}$$

where b_* represents any bias terms, $\boldsymbol{\beta}_{r(i),k}$ and $\boldsymbol{\alpha}_{i,k}$ represent k-th element of coefficient for user $r(i)$ and update i respectively. Note that $\sigma(x) = \frac{1}{1+\exp(-x)}$.

4.4 Latent Factor Models

Although linear models are efficient, they are usually oversimplified and cannot capture interactions between different effects. Latent Factor Models (LFM) are widely used in recommender systems (e.g., [17, 29, 31, 30]) and have proven effective in many scenarios (e.g., [17]). Specifically, LFM can model the interactions between different types of entities such as user-user and user-item, discovering their latent relationships. In this section, we discuss two types of LFM: matrix factorization and tensor factorization, and see how they can be applied to the task of SSR.

Matrix Factorization: In traditional CF, matrix factorization techniques are used to exploit user-item interactions. A straightforward idea would be to directly apply matrix factorization methods to user-update matrix. However, this idea is not practical. As discussed above, social streams have new updates arriving all the time, and existing updates are only consumed by a small number of users. Thus, cold-start problems are much more severe here, compared to traditional CF where the user base and the item base are relatively stable. Here, we impose a latent factor $\boldsymbol{\eta}_u \in \mathbb{R}^k$ for each recipient and producer and factorize the recipient-producer matrix to predict

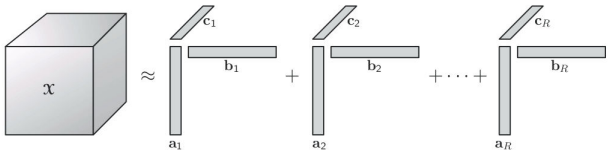

Figure 2: CANDECOMP/PARAFAC decomposition of a tensor, a three-way array. The dimensionality of three latent factors is the same.

the actions on updates. We describe the model in a probabilistic way:

$$\boldsymbol{\eta}_u \sim P(\boldsymbol{\eta}_u \,|\, \mathbf{0}, \sigma^2\mathbf{I}) \;\; u \in \{\mathcal{R}, \mathcal{S}\}$$
$$y_i \sim P(y_i \,|\, \boldsymbol{\eta}_{r(i)}, \boldsymbol{\eta}_{s(i)}, b_*, \mu)$$

where b_* is any biases introduced in Section (4.3), $\mathcal{R}$ and $\mathcal{S}$ are the set of recipients and producers respectively. For $P(\boldsymbol{\eta}_u \,|\, \mathbf{0}, \sigma^2\mathbf{I})$, it is usually assumed to be Gaussian or Laplace, corresponding to L_2 or L_1 regularization on latent factors respectively. Here, we use a multivariate Gaussian assumption. For $P(y_i \,|\, \boldsymbol{\eta}_{r(i)}, \boldsymbol{\eta}_{s(i)}, b_*, \mu)$, we assume:

$$y_i \sim P(y_i \,|\, f_i)$$
$$f_i = \mu + b_i + b_{t(i)} + b_{r(i)} + b_{c(i)} + b_{s(i)} + \boldsymbol{\eta}_{r(i)}^T \boldsymbol{\eta}_{s(i)} + \epsilon$$

where ϵ allows a Gaussian distribution. Thus, the final generative process also follows a Gaussian distribution. This formalism is similar to the one introduced in [17]. The model described here is very intuitive. Whether a user u_1 is going to click on an update from a user/company u_2 depends on u_1's and u_2's affinity.

Tensor Factorization: As social streams have different entities like recipients, producers and categories, it would be natural to directly model the multi-way data interaction, rather than concentrating on two-way relationships. It has been shown that high-order relational modeling can improve the performance of CF systems in many scenarios, for instance in social tag recommendations [24, 26].Here, we focus on one particular three-way relationship: recipient-producer-category of the update. We associate latent factors $\boldsymbol{\eta}_x \in \mathbb{R}^k$ for these three types of entities. Similar to the matrix case, we define the following generative procedures:

$$\boldsymbol{\eta}_x \sim P(\boldsymbol{\eta}_x \,|\, \mathbf{0}, \sigma^2\mathbf{I}) \;\; x \in \{\mathcal{R}, \mathcal{S}, \mathcal{T}\}$$
$$y_i \sim P(y_i \,|\, f_i)$$

where f_i is defined as:

$$f_i = \mu + b_i + b_{t(i)} + b_{r(i)} + b_{c(i)} + b_{s(i)} + \sum_k \boldsymbol{\eta}_{r(i),k} \boldsymbol{\eta}_{s(i),k} \boldsymbol{\eta}_{t(i),k} + \epsilon$$

where $\boldsymbol{\eta}_{*,k}$ represents the k-th element in the vector and ϵ again follows a Gaussian distribution. This particular form of tensor decomposition is known as CANDECOMP/PARAFAC (CP) decomposition [16], depicted in Figure 2. There are two important properties about CP decomposition. Firstly, it is a direct analogue to factorization methods in two-way array (matrix) data where latent factors share the same latent space. Secondly, CP decomposition has a nice yet surprising property that it has a unique solution of decomposition where matrix factorization does not enjoy this result [16]. This property indeed provides a theoretical guarantee to the decomposition and may be a reason for better performance.

We are aware of other forms of tensor factorization as well. For instance, Tucker decomposition [16], where the dimensionality of different latent factors varies, is widely used in many applications and applied to social media as well (e.g., [24, 29]). However, we do not choose the Tucker decomposition for our settings because not only it requires to pre-specify the dimensionality of all factors separately, but also does not guarantee uniqueness of the decomposition result.

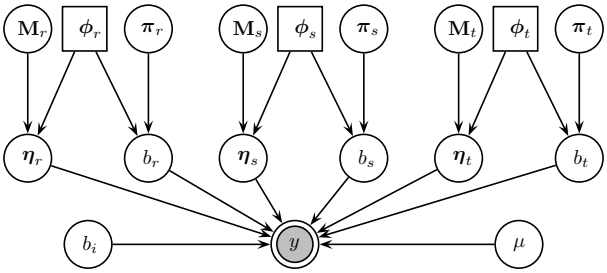

Figure 3: A graphical representation of regression-based tensor factor model. Square nodes represent features and circled nodes represents unknown variables. The response y in the middle is observed in the training set but to be predicted in the test set.

Incorporating Features: Both matrix factorization and tensor factorization discussed above do not directly incorporate explicit features. Here, we introduce features into the model by employing two levels of regression models. The basic idea is that latent features will depend on explicit features and final responses are derived from latent features. The first level regression models are defined as:

$$\boldsymbol{\eta}_{x(*)} = \mathbf{M}_x \boldsymbol{\phi}_{x(*)} + \epsilon_x \qquad x \in \{\mathcal{R}, \mathcal{S}, \mathcal{T}\}$$

where $\boldsymbol{\phi}_{x(*)}$ represents a feature vector for entity x and $\mathbf{M}_x$ is a transformation matrix to be learned. If ϵ_* follows a zero-mean k-dimensional Gaussian distribution, latent factors $\boldsymbol{\eta}_*$ indeed follow multivariate Gaussian distribution with the mean of a transformation of explicit feature vectors. This way, explicit feature space is mapped to latent feature space.

In addition to binding latent factors to explicit features, other biases may also have the same prior distributions:

$$b_{x(*)} = \boldsymbol{\pi}_x^T \boldsymbol{\phi}_{x(*)} + \epsilon_{b_x}$$

where $\boldsymbol{\pi}_x$ is a regression coefficient for entity x. If the error term ϵ_{b_x} follows a Gaussian distribution, biases $b_{x(*)}$ will also follow a Gaussian distribution centered at a transformation from explicit features. Note that this two-level regression scheme can be applied to matrix factorization as well as tensor factorization. The idea to use regression priors for matrix factorization has been explored by [1, 30] but not yet discussed on multi-way data relations like tensors. The final graphical representation of the model is shown in Figure 3.

In addition to the method described here to incorporate features, we are aware of other possibilities, such as [23] where latent factors and explicit features are treated as same set of features.

Like linear models from Section 4.3, LFM (with features) can also be learned through a Maximum A Posterior (MAP) estimation. Taking Tensor Factorization with Features as an example, the problem of minimizing the negative log posterior of the model boils

down to the following objective:

$$L_2 = \sum_i L_1(y_i, f_i) + \sum_{x\in\{\mathcal{R},\mathcal{S},\mathcal{T}\}} \lambda_x \sum_{x(i)} ||\boldsymbol{\eta}_{x(i)} - \mathbf{M}_x\boldsymbol{\phi}_{x(i)}||_F^2 + \sum_{x\in\{\mathcal{I},\mathcal{R},\mathcal{S},\mathcal{T},\mathcal{C}\}} \lambda_{b_x} \sum_{x(i)} ||b_{x(i)} - \boldsymbol{\pi}_x^T\boldsymbol{\phi}_{x(i)}||^2 + \sum_{x\in\{\mathcal{I},\mathcal{R},\mathcal{S},\mathcal{T},\mathcal{C}\}} \Big(\lambda_{\pi_x}||\boldsymbol{\pi}_x||_F^2 + \lambda_{M_x}||\mathbf{M}||_F^2\Big)$$

where all constant terms are ignored and all λ terms are manually tuned regularization parameters. For both matrix factorization and CP decomposition, a number of techniques are available to solve the objective function. For instance, the alternating least squares (ALS) method is the "workhorse" [16] for both matrix and tensor factorization. However, here, we still adopt a SGD method, which can scale to the dataset with which we are working. In order to use SGD, the gradients of latent factors can be derived. Firstly, we focus on latent factors:

$$\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\eta}_{r(i),k}} &= -\sum_i \Big[1-\sigma(y_i f_i)\Big] y_i \boldsymbol{\eta}_{s(i),k}\boldsymbol{\eta}_{t(i),k} \\
&\quad + 2\lambda_r\Big(\boldsymbol{\eta}_{r(i),k} - \mathbf{M}_{r[k]}\boldsymbol{\phi}_{r(i)}\Big) \\
\frac{\partial L_2}{\partial \boldsymbol{\eta}_{t(i),k}} &= -\sum_i \Big[1-\sigma(y_i f_i)\Big] y_i \boldsymbol{\eta}_{r(i),k}\boldsymbol{\eta}_{s(i),k} \\
&\quad + 2\lambda_t\Big(\boldsymbol{\eta}_{t(i),k} - \mathbf{M}_{t[k]}\boldsymbol{\phi}_{t(i)}\Big) \\
\frac{\partial L_2}{\partial \boldsymbol{\eta}_{s(i),k}} &= -\sum_i \Big[1-\sigma(y_i f_i)\Big] y_i \boldsymbol{\eta}_{r(i),k}\boldsymbol{\eta}_{t(i),k} \\
&\quad + 2\lambda_s\Big(\boldsymbol{\eta}_{s(i),k} - \mathbf{M}_{s[k]}\boldsymbol{\phi}_{s(i)}\Big)
\end{aligned}$$

where $\boldsymbol{\eta}_{*,k}$ is the k-th element of the vector and $\mathbf{M}_{*[k]}$ is the k-th row of the matrix. For all biases, gradients $\frac{\partial L_2}{\partial b_{*(i)}}$ are as follows:

$$-\sum_{i\in b_{*(i)}} \Big[1-\sigma(y_i f_i)\Big] y_i + 2\lambda_{b_*}\Big(b_{*(i)} - \boldsymbol{\pi}_*^T\boldsymbol{\phi}_{*(i)}\Big)$$

where $*$ means a particular type of bias and $i \in b_{*(i)}$ represents the updates those bias type match the type of interests. The gradients $\frac{\partial L_2}{\partial M_{x[k,m]}}$ for matrix M_x can be derived as:

$$2\lambda_x \sum_{x(i)} \Big(\boldsymbol{\eta}_{x(i),k} - \mathbf{M}_{x[k]}\boldsymbol{\phi}_{x(i)}\Big)\Big(-\boldsymbol{\phi}_{x(i),k}\Big) + 2\lambda_{M_x}\mathbf{M}_{x[k,m]}$$

where $\mathbf{M}_{x[k,m]}$ is the (k,m)-th element in the matrix $\mathbf{M}_x$. And finally, the gradients for regression coefficients $\frac{\partial L_2}{\partial \boldsymbol{\pi}_{x,k}}$ can be computed as:

$$2\lambda_{b_x} \sum_{x(i)} \Big(b_{x(i)} - \boldsymbol{\pi}_x^T\boldsymbol{\phi}_{x(i)}\Big)\Big(-\boldsymbol{\phi}_{x(i),k}\Big) + 2\lambda_{\pi_x}\boldsymbol{\pi}_{x,k}$$

4.5 Pairwise Learning

So far, we have demonstrated two different types of models: linear models and latent factor models. Both of them minimize certain errors in the learning process. As discussed in Section 4.1, a ranking-based evaluation metric, MAP is used in our experiments. Thus, it is more reasonable to directly optimize this ranking metric. However, it is difficult to optimize ranking measures directly [32, 6, 22] due to their discrete nature. Although some techniques (e.g., [6, 22]) have been developed to derive smoothed surrogate functions to approximate these ranking measures, including MAP, they are usually complicated and expensive to apply to large scale scenarios. Here, we use a much simpler approach: derive pairwise preferences from users' impressions and learn a pairwise ranking function.

First, let $\mathcal{O}_i$ be the set of updates in the impression i, $\mathcal{O}_{i,+}$ be the set of updates clicked by the user and $\mathcal{O}_{i,-}$ be the set of updates not clicked by the user. Remember that we eliminate impressions without any clicks (see 4.2). Therefore, we guarantee that the method described here can be applied to all impressions in our dataset. For any pair of updates (m,n) where $m \in \mathcal{O}_{i,+}$ and $n \in \mathcal{O}_{i,-}$, we can always construct a preference label $l_{m,n} = 1$, meaning that update m is favored over update n in impression i. Under this setting, we have the new objective function for impression i:

$$l_2(i,\mathbf{f}) = \frac{1}{|\mathcal{O}_{i,+}||\mathcal{O}_{i,-}|} \sum_{m\in\mathcal{O}_{i,+}} \sum_{n\in\mathcal{O}_{i,-}} \sigma\Big(f_m - f_n\Big) \quad (6)$$

where σ is a logit function. This new objective function is no longer to fit a single observed label (click or not) but to optimize a pairwise preference induced from impressions. Similar ideas are also explored in [25, 31, 18]. With this new objective function, we can replace the original loss function defined in Equation (5) in both linear model learning and factor model learning. Gradients are omitted due to space limits.

4.6 Summary & Discussion

We discussed several issues related to our proposed methods in this sub-section: 1) parameter tuning, 2) scalability and 3) feature treatment. For parameter tuning, while it is not a significant problem for Linear Models, as they can be trained efficiently, it might be prohibitively expensive to tune a Latent Factor Model. In this work, we do not heavily tune parameters and only wish to see whether these proposed approaches work in principle. One way to deploy a "parameter-free" model might be to consider a Bayesian treatment of Latent Factor Models, like [27, 29]. However, the sheer amount of data and its continuous nature prevent us to explore Bayesian treatment in this work and leave it to the future work. In terms of scalability, we conduct experiments on a single machine in this work but we do notice that SGD can be paralleled [34]. Thus, we can scale our model to even larger datasets. The way we integrate explicit features and latent factors is through regression models. However, this is not the only way to deal with this kind of problem. For instance, matrix co-factorization (see, e.g., [28]) and tensor co-factorization can be another paradigm of combining explicit features and hidden features.

5. EXPERIMENTAL RESULTS

In this section, we demonstrate the effectiveness of our model, through a comprehensive comparison with non-trivial baselines. The dataset used in our experiments is described in Section 4.2. Before we go into the details of the experimental results, we first discuss our experimental setting in Section 5.1 and then all developed models in Section 5.2.

5.1 Experimental Setting

Two standard settings are available to evaluate the effectiveness of systems for SSR. One is to test each model against an existing system in a online setting where both systems run in parallel for a similar audience in a reasonable period of time. After this, the effectiveness of both systems can be calculated using certain measurements, like error rate or ranking metrics. This is a classic A/B testing scenario. The advantage of A/B testing is obvious: it provides a real comparison between models. However, it might be

Models	Comments
Baseline (BL)	LinkedIn
Feature Model (FM)	Section 4.3
Latent Bias Model (LBM)	Section 4.3
Feature Bias Model (FBM)	Section 4.3
Matrix Factorization (MF)	Section 4.4
Tensor Factorization (TF)	Section 4.4
Matrix Factorization with Features (MF2)	Section 4.4
Tensor Factorization with Features (TF2)	Section 4.4

Table 2: All models used in our experiments.

Features	Comments
Seniority	the seniority level of a user
Visiting	how frequently a user visits LinkedIn
PageRank	discretized PageRank scores
Connectedness	how well a user is connected to others
Social strength	how tight a user's connections is
Professional	how professional an update's language is
Recency	the freshness of an update (see Section 4.3)

Table 3: All features used in our experiments.

Training/Testing	BL	FM	LBM	FBM
4_01(Tr.)/4_08(Te.)	0.5278	0.5317	**0.5943**	0.5520
4_08(Tr.)/4_15(Te.)	0.5435	0.5509	**0.6040**	0.5574
4_15(Tr.)/4_22(Te.)	0.5218	0.5246	**0.5823**	0.5235
9_01(Tr.)/9_10(Te.)	0.4829	0.4911	**0.5457**	0.4984
9_10(Tr.)/9_18(Te.)	0.4779	0.4798	**0.5432**	0.4915
9_18(Tr.)/9_25(Te.)	0.4768	0.4803	**0.5329**	0.4886

Table 4: The comparison between linear models. The best performance is shown in bold.

Type Description	Bias b_t
Job Seeker Product Update	0.5765
Joining Sub-Group	0.5407
Company News	0.4592
Joining Group	0.2625
Profile Picture Update	0.2516
Initiating Direct Ads Campaign	0.2253
Profile Update	0.1394

Table 5: Example of highly ranked types of updates

time-consuming and even impossible to compare many models in a batch. In addition, some models require tuning parameters, which may risk the business of the service a company offers. Thus, we do not use A/B testing in this paper and leave it to the future work.

In this paper, we simulate real settings of SSR, conducting offline experiments. More specifically, we gather historical data from LinkedIn user logs, which capturing all impressions users have consumed. Since we know which updates are clicked in each impression, it is easy to replay all these impressions and reorder the updates. Thus, we can produce a "new" impression for users in the dataset. The drawback of this approach to experiments is that we cannot show "new" ordering of impressions that are not clicked by users at all because whether users would have clicked them or not is impossible to test. This is another reason why we drop all impressions without any clicks (Section 4.2).

The dataset for one month is divided into weeks. We train our models on one week and test them on the following week. This setting results in more than 70% items being new in test data each week, which is an evidence to the fact that SSR is different from RecSys and IR.

5.2 Models & Features

We compare several models in our experiments. All models used in the following experiments are shown in Table 2. The baseline is a proprietary system currently deployed in the product of LinkedIn homepage. FM, LBM and their combination (FBM) are examples of simple linear models while MF, TF with their feature enhanced extensions (MF2 and TF2) are examples of latent factor models. For all models, a point-wise loss function (Equation (5)) and a pairwise loss function (6) are both tested. Without stating it explicitly, all models include the temporal effect feature discussed in Section (4.3) while the parameter ζ, the balance between recency and relevance, is manually tuned. All regularization parameters are simply set to 1. We understand that this may not be an optimal choice. For tuning a reasonable learning rate in SGD and ζ, we use the first day in the test week as a "validation" day and choose the parameter setting that can provide the optimal performance on the day. We fix the parameters for the remaining days in the test week.

Some of linear models and latent factor models require explicit features. In this paper, we include several important features to enhance our models. Note that we are aware of many other possible features. However, it is not our goal to study the effectiveness of a particular feature in this work. All features are shown in Table 3. "Seniority" measures the seniority level of a user's job title. "Visiting" measures how well engaged a user is (our assumption is that a frequent visitor is likely to interact with his/her social stream). "PageRank" (details in [4]) and "Connectedness" measure how a user connects with other users. Presumably, a highly respected and well connected user can attract others to interact with their update streams. "Social strength" is a proprietary product used in LinkedIn, measuring the connection closeness between two users. "Professional" measures how likely an update is similar in its language to professional profiles of LinkedIn users (i.e. how professional an update is). The assumption is that users may favor professional updates over non-professional updates on LinkedIn because it is a professional social network.

5.3 Results on Linear Models

In this sub-section, we focus on the comparison between the baseline and all linear models. In this Subsection, we focus on the comparison between the baseline and all linear models. The results are shown in Table 4. The first column indicates how models are trained and tested. For instance, the first number "4_01" means the models are trained on the week of April 1st and tested on the week of April 8th (8-th is the date for validation of parameter tuning) where "Tr." and "Te." are shorthand for "Training" and "Testing", respectively. We conduct experiments on two separate months to avoid some seasonal fluctuations on the data. The numbers shown on the right part of the table are MAP scores.

Our first observation is that the baseline of MAP in September is lower than its in April, implying that updates in the lower positions in the lists get clicked more often over time. One possible explanation is that users become familiar with their social streams and start to find interesting updates manually, looking at more items down the list. The second observation is that all linear models, including FM, LBM and FBM, perform better than the baseline, consistently on two-month datasets. However, for FM, which only depends on explicit features, the performance is very close to the baseline. This is reasonable because only a handful of features are used in our experiments and we do expect that these features are not likely discriminative. On the other hand, LBM, a model only depending on implicit feedback, has consistently $5\% - 6\%$ absolute improvements on MAP over the baseline. This confirms that it is vital

Training/Testing	MF	TF	MF2	TF2
4_01(Tr.)/4_08(Te.)	0.5955	0.6258	0.5951	**0.6336**
4_08(Tr.)/4_15(Te.)	0.6079	0.6228	0.6088	**0.6535**
4_15(Tr.)/4_22(Te.)	0.5962	0.6014	0.5991	**0.6312**
9_01(Tr.)/9_10(Te.)	0.5511	0.5766	0.5523	**0.6003**
9_10(Tr.)/9_18(Te.)	0.5412	0.5833	0.5449	**0.6109**
9_18(Tr.)/9_25(Te.)	0.5359	0.5799	0.5362	**0.5992**

Table 6: The comparison between latent factor models. The best performance is shown in bold.

to exploit different aspects of users' feedbacks and capture them through bias modeling (e.g., [17, 15]). Indeed, FBM gives the most improvements over the baseline in all our experiments. The idea is simple and easy to implement. For the combination of a pure feature-based model FM and a pure implicit-feedback-based model LBM, FBM does perform as someone might expect. It is significantly worse than LBM and almost identical to FM, which might indicate that simply integrating explicit features with biases may not be a good choice and more sophisticated approaches are needed.

LBM can also reveal some interesting patterns from the dataset, which might not be easily identified by other methods. For instance, we can figure out the effective popularity of different types of updates by looking at the values of b_t. The positivity or negativity of these values indicate whether a particular type of update correlates with clicks or non-clicks, while the magnitude of these values means how strongly this correlation might be. We show some samples of top ranked types in Table 5, which are positively correlated with clicks. From the table, we see that job related updates, company-related updates are comparatively attractive. In addition, users pay attention to profile changes of their connections and new connections established by their friends. Note that the value shown in the table is "automatically" normalized in the sense that SGD only updates corresponding parameters when the algorithm meets such type of updates. Also, the ratio of positive examples of a particular type will drive the parameter to strong positive numbers. Thus, no post-processing steps are required. This is an example of how our model can be used in simple data analysis tasks.

5.4 Results on Latent Factor Models

As we discussed earlier, latent factor models are widely used and have been proven to be superior to linear models. We conduct the same experiments as linear models and show their results in Table 6. Here, we compare between pure factorization-based models (MF and TF) and feature-enhanced factor models (MF2 and TF2). Note that all these models are built upon LBM and therefore the performance should at least match LBM. The second column of the table shows the results from MF, which is essentially to factorize the recipient-sender matrix and uncover latent structures. Unfortunately, the gain of performance of MF over LBM is marginal and even not observable. On the other hand, TF offers significant improvements and leads another $3\% - 4\%$ absolute boost for MAP on average. As we discussed before, social stream data is much more complex than traditional recommender system data (in various CF scenarios). Users may interact with certain updates because their senders are famous people or because the type of updates (e.g., news or twitter updates) is of a particular interest. Thus, tensor factorization can model multi-way relationships better than matrix factorization models them via a decomposition to multiple two-way relationships. Indeed, bias terms of recipients and senders in LBM might capture the basic relationship between them and a matrix factorization does not provide any additional benefits. Furthermore, we have already discussed why we do not employ the user-item

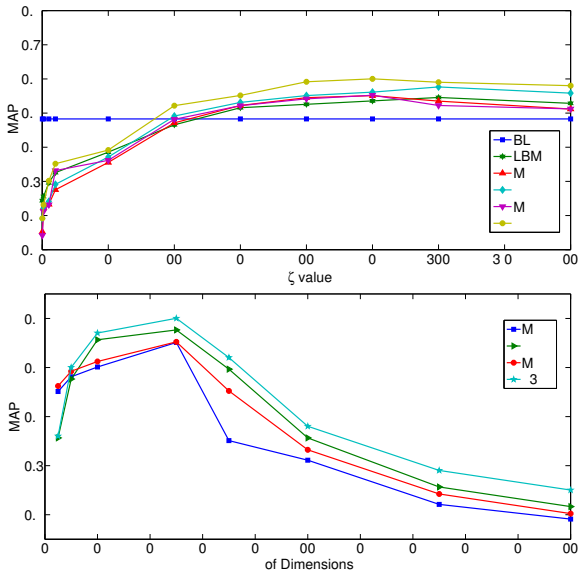

Figure 4: Parameter Sensitivity Analysis: 1) the effect of ζ (top) and 2) # of dimensions in latent factor models (bottom).

Training/Testing	LBM	MF	MF2	TF	TF2
4_01(Tr.)/4_08(Te.)	0.6169	0.6033	0.6151	0.6358	**0.6532**
4_08(Tr.)/4_15(Te.)	0.6188	0.6168	0.6188	0.6528	**0.6641**
4_15(Tr.)/4_22(Te.)	0.5897	0.6104	0.6191	0.6014	**0.6402**
9_01(Tr.)/9_10(Te.)	0.5644	0.5716	0.5723	0.5966	**0.6207**
9_10(Tr.)/9_18(Te.)	0.5593	0.5621	0.5607	0.5999	**0.6183**

Table 7: The effects of pair-wise learning. The best performance is shown in bold.

matrix in our setting: new items are much more common in social streams, compared to other recommender system setups. Thus, it is very interesting to see that well-established matrix-based factor models do not work equally well on social streams as they do in traditional CF scenarios. We believe that a more thorough investigation on this issue is desired.

For latent factor models with features, it is noticeable that MF2 failed to outperform LBM again while TF2 has gained additional $2\% - 3\%$ absolute improvement over TF consistently. This is a yet another signal that matrix factorization does not work well in SSR. For TF2, the increase of MAP can be explained by the two-level regression structure, that explicit features will help create latent features when new items or new users come into the system, effectively mitigating the cold-start problem. The performance of TF2 also validates that regression-based latent factor models are an effective approach to integrate explicit features with latent features.

We also study the sensitivity of parameters, especially the temporal balance weight ζ and the number of dimensions in latent factor models. Taking the first week of September as an example, the effects of both parameters are shown in Figure 4. The first observation is that both parameters are vital to the final performance and they are very sensitive if they are out of certain ranges. For ζ, the optimal performance is achieved when it is around $250 - 300$ while for dimensionality, the results suggest that a reasonable number $(20 - 50)$ is the key to success.

5.5 Results on Pairwise Learning

The results demonstrated so far focus on point-wise learning pro-

cedure. In other words, the objective function imposed by models is still error-based loss function. Here, we focus on how to improve performance by switching the objective function to pairwise preferences learning. More specifically, we conduct similar experiments as previous ones and report results on LBM, MF, TF MF2 and TF2, shown in Table 7. Other models are omitted due to their poor performance. First of all, we notice that almost all models can benefit from pairwise learning, even for the methods which did not show significant gains in previous experiments, such as MF and MF2. However, on another perspective, the overall improvement of a pairwise learning is not huge, usually yielding $1.5\% - 2\%$ improvement on MAP. One possibility is that the pairwise learning here is still naïve. More sophisticated session enabled learning procedures (e.g., [31]) are to be investigated in the future.

6. CONCLUSION & FUTURE WORK

In this paper, we investigate the problem of ranking social updates from a unique perspective of LinkedIn, the largest professional social network in the world. More specifically, we address the task as an intersection of learning to rank, collaborative filtering and click-through modeling, leveraging ideas from information retrieval and recommender systems. We propose a novel probabilistic latent factor model with regressions on explicit features, comparing a number of non-trivial baselines and gaining an approximately 10% improvement on MAP over the baseline. In addition to superior performance demonstrated in the paper, we shed some light on social updates on LinkedIn and how users interact with them, which might be applicable for social streams in general. For future work, it is interesting to see whether it is possible to develop efficient Bayesian treatment of latent models. In addition, other models might be explored as we demonstrate that state-of-the-art CF models do not provide comparable success in SSR. We also wish to extend our work by considering the diversity of information users wish to consume.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant Numbers IIS-0545875 and IIS-0803605.

7. REFERENCES

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

[2] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *SIGCOMM 2009*, pages 49–62.

[3] C. Borgs, J. T. Chayes, B. Karrer, B. Meeder, R. Ravi, R. Reagans, and A. Sayedi. Game-theoretic models of information overload in social networks. In *Workshop on Algorithms and Models for the WebGraph 2010*, pages 146–161.

[4] S. Budalakoti and R. Bekkerman. Bimodal invitation-navigation fair bets model for authority identification in a social network. In *WWW 2012*, pages 709–718.

[5] B. Cao, D. Shen, K. Wang, and Q. Yang. Clickthrough log analysis by collaborative ranking. In *AAAI 2010*.

[6] O. Chapelle and M. Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13:216–235, 2010.

[7] J. Chen, R. Nairn, and E. Chi. Speak little and well: recommending conversations in online social streams. In *CHI 2011*, pages 217–226.

[8] J. Chen, R. Nairn, L. Nelson, M. Bernstein, and E. Chi. Short and tweet: experiments on recommending content from information streams. In *CHI 2010*, pages 1185–1194.

[9] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM 2008*, pages 87–94.

[10] M. De Choudhury, S. Counts, and M. Czerwinski. Identifying relevant social media content: leveraging information diversity and user cognition. In *Hypertext and Hypermedia 2011*, pages 161–170.

[11] Y. Duan, L. Jiang, T. Qin, M. Zhou, and H.-Y. Shum. An empirical study on learning to rank of tweets. In *COLING 2010*, pages 295–303.

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

[13] B. Hu, Y. Zhang, W. Chen, G. Wang, and Q. Yang. Characterizing search intent diversity into click models. In *WWW 2011*, pages 17–26.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *KDD 2002*, pages 133–142.

[15] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *RecSys 2011*, pages 165–172.

[16] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51:455–500, August 2009.

[17] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM TKDD*, 4:1:1–1:24, January 2010.

[18] Y. Koren and J. Sill. OrdRec: an ordinal model for predicting personalized item rating distributions. In *RecSys 2011*, pages 117–124.

[19] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[20] S. Muralidharan, L. Rasmussen, D. Patterson, and J.-H. Shin. Hope for haiti: An analysis of facebook and twitter usage during the earthquake relief efforts. *Public Relations Review*, 37(2):175 – 177, 2011.

[21] M. Naaman, J. Boase, and C.-H. Lai. Is it really about me?: message content in social awareness streams. In *CSCW 2010*, pages 189–192.

[22] T. Qin, T.-Y. Liu, and H. Li. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval*, 13:375–397, 2010.

[23] S. Rendle. Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57:1–57:22, 2012.

[24] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD 2009*, pages 727–736.

[25] S. Rendle, C. Freudenthaler, Z. Gantner, and S.-T. Lars. BPR: Bayesian personalized ranking from implicit feedback. In *UAI 2009*, pages 452–461.

[26] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM 2010*, pages 81–90.

[27] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML 2008*, pages 880–887.

[28] A. P. Singh and G. J. Gordon. A unified view of matrix factorization models. In *ECML 2008*, pages 358–373.

[29] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM 2010*, pages 211–222.

[30] S.-H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha. Like like alike: joint friendship and interest propagation in social networks. In *WWW 2011*, pages 537–546.

[31] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *SIGIR 2011*, pages 295–304.

[32] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR 2007*, pages 271–278.

[33] Y. Zhang, W. Chen, D. Wang, and Q. Yang. User-click modeling for understanding and predicting search-behavior. In *KDD 2011*, pages 1388–1396.

[34] M. Zinkevich, A. Smola, M. Weimer, , and L. Li:. Parallelized stochastic gradient descent. In *NIPS 2010*, pages 1–9.