TriRank: Review-aware Explainable Recommendation by Modeling Aspects*

Xiangnan He[1] Tao Chen[1] Min-Yen Kan[1] Xiao Chen[2]
[1]School of Computing, National University of Singapore
[2]Institute of Computing Technology, Chinese Academy of Sciences
{xiangnan, kanmy, taochen}@comp.nus.edu.sg chenxiao3310@ict.ac.cn

ABSTRACT

Most existing collaborative filtering techniques have focused on modeling the binary relation of users to items by extracting from user ratings. Aside from users' ratings, their affiliated reviews often provide the rationale for their ratings and identify what aspects of the item they cared most about. We explore the rich evidence source of aspects in user reviews to improve top-N recommendation. By extracting aspects (*i.e.,* the specific properties of items) from textual reviews, we enrich the user–item binary relation to a user–item–aspect ternary relation. We model the ternary relation as a heterogeneous tripartite graph, casting the recommendation task as one of vertex ranking. We devise a generic algorithm for ranking on tripartite graphs — TriRank — and specialize it for personalized recommendation. Experiments on two public review datasets show that it consistently outperforms state-of-the-art methods. Most importantly, TriRank endows the recommender system with a higher degree of explainability and transparency by modeling aspects in reviews. It allows users to interact with the system through their aspect preferences, assisting users in making informed decisions.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *information filtering*

Keywords

Reviews, Aspects, Explanable Recommendation, Top-N Recommendation, Tripartite Graph Ranking

1. INTRODUCTION

Recommender systems serve to help users discover choice products to consume, matching users to items of potential interest (*e.g.,* products, businesses, movies, *etc.*). Among the various recommendation techniques, collaborative filtering (CF) is widely used, due to its effectiveness in providing personalized recommendation based on the wisdom of the crowds. By and large, existing CF techniques have focused on modeling user–item relations, such as ratings. As a single rating only indicates a user's overall satisfaction for the item, it is hard to infer the actual rationale of the rating. Different users may care about different aspects of the same item. For example, in the restaurant domain, one user may give a 5-star rating for food quality, while another user may give the same rating, but due to favorable ambiance. Since existing CF techniques largely lack such fine-grained analysis, they can fail to accurately model a target user's interests.

Aside from ratings, most Web 2.0 systems also give users the opportunity to leave reviews on the provided items. These reviews often justify a user's rating, offering the underlying reasons for the rating by discussing the specific properties of the item. Thus they are well-suited as a complementary data source for collaborative filtering. In this work, we leverage on this key observation to address the task of *top-N recommendation*; that is, to produce an ordered list of N items that will be most appealing to a user.

Leveraging user reviews (or interchangably, "comments") is non-trivial for systems, as these reviews are written by users freestyle, exhibiting noise and irrelevant content. Recent research [10, 11, 17, 19] have largely modeled reviews at the word level, distilling them into latent topics to combine with the latent factor model. Although these methods achieve good prediction accuracy, the recommendation process is not transparent and the generated recommendations are not explainable to users, a well-known drawback of the latent factor model [14]. To improve users' experience and trust, transparency and explainability become increasingly important for practical recommender systems [27]. Moreover, improvements in rating prediction (a lower error rate) may not directly translate into improvements in top-N recommendation [3]. As systems usually make only their top suggestions visible, targeting improvement at the top ranks is most beneficial for recommendation in practice.

Instead of modeling reviews at the word level, we propose to model reviews in the level of distilled *aspects*, which are the specific properties of items and expressed in reviews as noun words or phrases [31]. For example, *battery* and *screen* are aspects of digital products, *food quality* and *ambiance* are aspects of restaurants. Given the aspects extracted from reviews, we first model the user–item–aspect relation as a tripartite graph. Then we devise TriRank, a generic al-

*This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

CIKM'15, October 19–23, 2015, Melbourne, Australia.

http://dx.doi.org/10.1145/2806416.2806504.

gorithm based on the graph regularization framework [9, 36] for ranking in tripartite graphs. TriRank ranks vertices by accounting for both the structural smoothness (encoding collaborative filtering and aspect filtering effects) and fitting constraints (encoding personalized preferences). Besides TriRank's superior performance, there are two key properties making it suitable in practice: first, the recommendation process is transparent by explaining its recommendations with respect to aspects, allowing users to customize their recommendations to fit their preferences (*i.e.,* scrutability [27]). Second, TriRank is tolerant of noisy aspects, common in automated aspect extraction, such that porting to new domains can be done without manual effort.

It is instructive to clarify that this work solely relies on the automatically detected item aspects in user reviews, and not any other additional information, such as sentiment (a separate area of study). We believe regardless of a user's sentiment on a reviewed aspect, its mention at least reflects the user's interest in the aspect. We believe by mining such item aspects latent in reviews, we can infer user preference at a finer granularity, thus providing better personalized recommendation.

This paper is organized as follows. In Section 2, we discuss the process of aspect extraction from reviews. We detail our TriRank method in Section 3, and conduct experiments and empirical studies on aspects in Section 4. We review related work in Section 5, before concluding the paper in Section 6.

2. ASPECT EXTRACTION

Aspect extraction, also termed as feature or attribute extraction, has a long history in review mining (see [31]). Aspects can be seen as the components, attributes, or properties of an item. Early seminal work [12] proposed several language rules to extract product features from reviews. The rules have been widely used and extended by later work, *e.g.,* [32] considered specific phrase patterns and sentence patterns. Aside from the unsupervised rule-based methods, supervised sequence labeling techniques such as the *Conditional Random Field* have been adopted to learn aspects [13].

As our focus is to leverage aspects from user reviews, we do not contribute to aspect extraction, but instead seek to maximally exploit technologies that can perform it. As such, we apply an existing state-of-the-art aspect extraction toolkit [33] that constructs a sentiment lexicon from user reviews. It creates entries that are feature–opinion word pairs with an associated sentiment polarity, represented as (F, O, S). For example, an entry such as (*service, excellent, positive*) might be extracted from a restaurant review. The key feature extraction part of the tool is a rule-based system, mainly adopting and extending the rules proposed by [12]. As each feature is a noun word or phrase, representing the item's property that a user comments on, we can directly use it as an aspect.

We apply the tool with its default settings, extracting $6,025$ and $1,617$ distinct features (*i.e.,* aspects) from our datasets culled from Yelp and Amazon Electronics, respectively (datasets described later in Section 4). Table 1 shows top features extracted from the two datasets, ranked by their $tf \times idf$ score. We notice that the tool produces some features that are good but also many noisy features, such as "ive" ("I've"), "picturesmy", "l50", which are quirks of the corpus. Also, some top features are domain-specific stop words (*e.g.*, "food", "restaurant", "product"), which do not represent the specific properties of items. Despite this significant level of noise, we do not perform any post filtering on the extracted aspects to test the robustness of our proposed method to noise.

Table 1: Top automatically extracted aspects.

Yelp	bar, salad, menu, chicken, sauce, restaurant, rice, cheese, fries, bread, sandwich, drinks, patio
Amazon	camera, quality, sound, price, product, battery, pictures, features, screen, size, memory, lens

Table 2: Statistics of aspects extracted from reviews.

		User–Aspect		Item–Aspect	
Dataset	# of Aspects	Avg. # of A / User	Density	Avg. # of A / Item	Density
Yelp	6,025	183.8	3.05%	138.0	2.29%
Amazon	1,617	61.4	3.80%	23.2	1.44%

Note that in terms of manifestation, aspects and tags (in social tagging systems) look alike – they are both usually short noun phrases. However, they differ fundamentally in nature and hence utility. Tags are simple keywords that are directly annotated by users to categorize and manage items. Aspects, on the other hand, describe specific attributes of items, and are implicitly extracted from free-text reviews.

Table 2 summarizes the statistics of extracted aspects on the two datasets. We note that the densities of the user–aspect and item–aspect matrix are much higher than that of the user–item rating matrix (usually less than 1%, see Table 3); a good signal that the aspect matrices contain rich information useful for addressing the sparseness of the original rating matrix.

3. PROPOSED METHOD

We now present our proposed method for review-aware recommendation, by introducing the tripartite graph to model the user–item–aspect ternary relation. We then devise the generic TriRank algorithm for ranking on tripartite graphs. Finally, we detail how to operationalize TriRank for personalized recommendation.

3.1 Data Model and Notation

Let $G = (U \cup P \cup A, E_{UP} \cup E_{UA} \cup E_{PA})$ be a tripartite graph, where U, P and A are vertex sets that represent users, items and aspects, respectively. Let E_{UP}, E_{UA} and E_{PA} be edges that represent user–item, user–aspect, item–aspect relations, respectively. Each input triple $<u_i, p_j, a_k>$ denotes that user u_i has rated item p_j with a review mentioning aspect a_k, is then represented as a triangle with edges e_{ij}, e_{ik} and e_{jk} (as in Figure 1). Each edge carries a weight to denote the strength of two connected vertices; edges with higher weight denote stronger more significant relations between vertices. For example, we can model user u_i's rating on item p_j as an edge with weight of e_{ij}. Without loss of generality, we use the symbol R, Y and X to denote the edge weight matrix of user–item, user–aspect and item–aspect relations, respectively.

3.2 Tripartite Graph Ranking (TriRank)

The goal for item recommendation is to devise a ranking function $f : P \to \mathbb{R}$, which maps each item in P to a real number such that the value reflects the target user u's (predicted) preference on the item. Then, sorting the resultant items by score yields u's personalized item ranking. Since

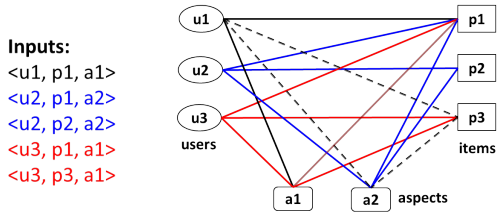
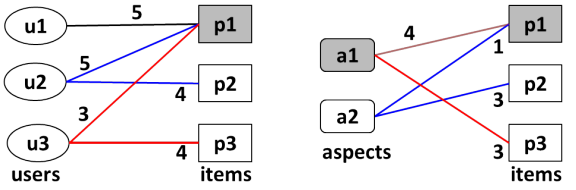


Figure 1: An example tripartite structure of the given inputs (the dashed line illustrates the additional input $<u_1, p_3, a_2>$).



(a) User-Item structure **(b)** Item-Aspect structure

Figure 2: Smoothness constraints on decomposed graphs from Figure 1. Assume $u1$ previously rated item p_1 with mentioning aspect a_1 (shaded vertices).

TriRank induces scores for all vertices in the graph, it has the important side-affect of assigning scores to the aspect and user vertices: these denote u's interest on aspects and similarity with other users, respectively.

In a nutshell, TriRank assigns the ranking score of vertices by enforcing the structural smoothness and fitting constraints of the graph. By smoothness and fitting constraints, we adopt the same definitions as those common to the graph regularization framework [9, 36]:

· *Smoothness* implies local consistency: that nearby vertices should not vary too much in their scores.

· *Fitting* encodes prior belief: that the ranking function should not cause much deviation from the observations.

TriRank seeks to assign each vertex a score such that *the graph is sufficiently smooth* and *the prior belief is retained*. In the following, we first illustrate how the two constraints in the tripartite graph capture the intuition for recommendation, before describing the TriRank algorithm.

3.2.1 *Illustrating Regularization Constraints*

Let us first see how the smoothness works by considering the example in Figure 1. We decompose it into two subgraphs in Figure 2 for ease of exposition. The left subfigure gives the user–item structure, where edge weights denote ratings. Assume we want to recommend items to u_1, who has only rated p_1 with a score of 5. As p_1 is connected more strongly to u_2 than u_3, u_2 is given a higher score than u_3. Finally, since the edge weights of $<u_2,p_2>$ and $<u_3,p_3>$ are identical, we infer that p_2 should receive a higher score than p_3. Such smoothness constraints on the user–item relation alone yields the traditional CF effect.

Considering aspects can provide additional evidence that influences the recommendation process. Let us continue to recommend for u_1 but base our decision on item–aspect structure (Figure 2(b)), where edge weight denotes the number of an item's reviews mentioning an aspect. As u_1 only previously mentions aspect a_1, enforcing smoothness would rank p_3 higher than p_2, as p_3 is more strongly connected to a_1, in contrast to p_2. This example also shows that predicting based on CF and aspect filtering yield different results; and that the *smoothness* constraint on the whole graph to combine them may be beneficial.

The fitting constraint serves as a means to personalize the ranking for each user (*cf.* shaded vertices of Figure 2). For a target user u, the past ratings and reviewed aspects indicate u's prior (known) preference on the vertices. It should guide the ranking process such that the resultant ranking function should be consistent with the prior belief.

3.2.2 *Regularization on Tripartite Graph*

We now define the regularization function to implement the two constraints for ranking vertices.

Smoothness. Similar to the previous work [9] that defines a smoothness regularizer on bipartite graphs, we devise the regularizer on user–item structure as follows:

$$Q_{UP}(f) = \sum_{i=1}^{|U|}\sum_{j=1}^{|P|} r_{ij}\left(\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(p_j)}{\sqrt{d_j^p}}\right)^2, \quad (1)$$

where $f(u_i)$ and $f(p_j)$ denote the final ranking scores (*i.e.,* parameters to learn); r_{ij} is the edge weight between u_i and p_j; d_i^u and d_j^p are the weighted degrees (sum of edge weights) of u_i and p_j, respectively, for normalization. The counterpart user–aspect and item–aspect smoothness regularizers for aspect filtering can be obtained similarly.

This smoothness regularizer can be seen as a graph kernel that measures the similarity of vertices. Although there are various kernels [25], we have purposefully chosen this one (originally introduced by [36]) due to its effective encoding of the CF effect in bipartite graph scenarios. To see this, assume we recommend for the target user u. First, minimizing Eq. (1) constrains a vertex's score based on its neighbors – if a user is strongly connected by many high-scoring **items** (*e.g.,* rated items of u), the user will be given a high score (*i.e.,* more similar with the u); likewise, if an item is strongly connected by many high-scoring **users** (*i.e.,* similar users), it will have a high score. Second, the quadratic nature of the normalization suppresses the popularity of highly connected vertices; this property is essential to prevent a ranking from being dominated by popular vertices [1].

Fitting. Let the target user's prior preference on item p_j be p_j^0; then the regularizer to enforce the fitting constraint on items is defined as:

$$Q_P(f) = \sum_{j=1}^{|P|}(f(p_j) - p_j^0)^2. \quad (2)$$

We can similarly achieve such fitting regularizers on users and aspects. This fitting regularizer corresponds to the squared error loss that is commonly used by machine learning models in recommendation. Different with the latent factor model [14, 19] that only optimizes for rated items, Eq. (2) also importantly takes unrated items into account (*cf.* summing over all items). This property is very desirable for the top-N task, as it aims at ranking unrated items [3]. Another option for top-N recommendation is to optimize a ranking-based loss function, such as *AUC* used by *Bayesian Personalized Ranking* [23]. This is an interesting extension to be explored in the future.

Regularization function. To account for the heterogeneous structure of the tripartite graph, we combine the smoothness regularizer on each relation type with the fitting regularizer using different weights for each vertex type:

$$\begin{aligned} Q(f) = \alpha \sum_{i,j} r_{ij} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(p_j)}{\sqrt{d_j^p}})^2 + \beta \sum_{j,k} x_{jk} (\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2 \\ + \gamma \sum_{i,k} y_{ik} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2 + \eta_U \sum_i (f(u_i) - u_i^0)^2 \\ + \eta_P \sum_j (f(p_j) - p_j^0)^2 + \eta_A \sum_k (f(a_k) - a_k^0)^2, \end{aligned} \tag{3}$$

where α, β and γ denote the weight of smoothness on user–item, item–aspect and user–aspect relation, respectively; η_U, η_P and η_A denote the weight of fitting constraint on users, items and aspects, respectively (we discuss how to set the prior preference u_i^0, p_j^0 and a_k^0 later in Section 3.3).

3.2.3 Optimizing the Regularization Function

We now minimize Eq. (3) to derive the final ranking scores (*i.e.,* model parameters). As the objective function is strictly convex, standard optimization techniques find a unique solution regardless of initialization. Two widely used techniques are *stochastic gradient descent* (SGD) and *alternating least squares* (ALS). SGD updates all parameters towards the negative gradients for each training instance, while ALS minimizes the objective function per parameter until a joint optimum is found (*i.e.,* coordinate-wise descent). For this scenario, we adopt ALS over SGD as the objective function can be analytically solved for each parameter, and importantly, it does not need to set the learning rate, which is crucial to SGD's effectiveness. Additionally, it usually yields a faster convergence and is easier to parallelize than SGD.

By differentiating $Q(f)$ with respect to $f(u_i)$, $f(p_j)$ and $f(a_k)$, respectively, and letting the derivatives be 0, we obtain the iterative update rules. Let the ranking vector for items be $\vec{p} = [f(p_j)]_{|P|\times 1}$, and the prior preference vector for items be $\vec{p}_0 = [p_j^0]_{|P|\times 1}$. Let similar definitions follow for $\vec{u}, \vec{u}_0$ for users, and $\vec{a}, \vec{a}_0$ for aspects. The equivalent update rules in matrix form are as follows:

$$\begin{aligned} \vec{u} &= \frac{\alpha}{\alpha+\gamma+\eta_U} S_R \cdot \vec{p} + \frac{\gamma}{\alpha+\gamma+\eta_U} S_Y \cdot \vec{a} + \frac{\eta_U}{\alpha+\gamma+\eta_U} \vec{u}_0, \\ \vec{p} &= \frac{\alpha}{\alpha+\beta+\eta_P} S_R^T \cdot \vec{u} + \frac{\beta}{\alpha+\beta+\eta_P} S_X \cdot \vec{a} + \frac{\eta_P}{\alpha+\beta+\eta_P} \vec{p}_0, \\ \vec{a} &= \frac{\gamma}{\gamma+\beta+\eta_A} S_Y^T \cdot \vec{u} + \frac{\beta}{\gamma+\beta+\eta_A} S_X^T \cdot \vec{p} + \frac{\eta_A}{\gamma+\beta+\eta_A} \vec{a}_0, \end{aligned} \tag{4}$$

where S_R is the symmetric normalized form of matrix R, defined as $[\frac{r_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}}]_{|U|\times|P|}$; S_X and S_Y are defined similarly. We note that a closed-form solution can be obtained analytically, but omit them due to space limitations.

3.3 Personalized Recommendation

Given the general TriRank algorithm, we need to cover how we obtain the initial graph (specifically, edge weights and the target user's prior preference) to concretize the generic algorithm for our review-based recommendation scenario.

Edge weights. *User–item* edge weights from relation R can be set as in traditional CF: in cases with explicit feedback, it can be the rating score; for implicit feedback, whether the user has interacted with or browsed the item (measured as either a binary yes/no, or an integer view count). Our datasets provide explicit user ratings, so we use these ratings as-is.

For the *user–aspect* relation Y and the *item–aspect* relation X, edge weights connote the degree of user interest (item speciality) with respect to the aspect. Once aspects are identified in reviews, we can use either the actual count (number of mentions) within all a user's (item's) reviews, or the review frequency (number of reviews that mention the aspects). As reviews vary in length, an aspect may occur multiple times in long reviews, but may not imply that the user pays more attention to the aspect[1]. As such, we use review frequency in our experiments. As in general IR, we take the logarithm of the review frequency, to dampen the effect of aspects that appear very frequently.

Prior preference. We need to set the prior preference vectors for the three vertex types, with respect to the target user u_i for personalization.

For *items*, the item prior preference vector $\vec{p}_0$ takes a positive value if the target user has interacted with the item, otherwise, 0. Thus we adopt the i^{th} row vector of R as the $\vec{p}_0$ for the target user u_i. Similarly, for *aspects*, the aspect preference vector $\vec{a}_0$ is set as the row vector of user–aspect matrix Y. As the smoothness part of Eq. (3) normalizes the edge weight by a vertex's degree, we also apply the L_1 norm on $\vec{p}_0$ and $\vec{a}_0$ for meaningful combination.

The user preference vector $\vec{u}_0$ should denote the target user's similarity with other users. We can set $\vec{u}_0$ based on a user's social network when it is available. We can also adopt standard user-based CF, and set $\vec{u}_0$ by measuring user similarity from the rating matrix R. In this work, we adopt the most basic approach, simply setting the target user herself as 1, and all other users as 0. Note that this variable does not directly reflect user's preference on aspects or items, so its design is beyond the scope of this work.

TriRank works by enforcing the collaborative filtering and aspect filtering effects, and is summarized in Algorithm 1. For convergence, one can monitor $Q(f)$'s value until it stabilizes or set a maximum number of iterations.

The iterative solution Eq. (4) presents a more transparent view on the ranking process. The scores of items, users and aspects mutually reinforce each other – a score increase in an item will increase the score of its connected users and

Algorithm 1: TriRank for review-aware top-N item recommendation.

Input: User-Item interactions R and reviews.

Offline Training (for all users):
1. Extract aspects from reviews (Section 2).
2. Build item–aspect matrix X and user–aspect matrix Y.
3. TF term weighting: $X = X.tf(); Y = Y.tf()$.
4. Build symmetric normalized matrices S_R, S_X, S_Y.

Online Recommendation (for target user u_i):
5. Build u_i's prior preference vectors $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$.
6. L_1 norm on $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$.
7. Randomly initialize ranking vectors $\vec{p}$, $\vec{a}$ and $\vec{u}$.
8. Iteratively run update rules Eq. (4), until convergence.
9. Recommend top ranked items to u_i, and explain the recommendation using top ranked aspects.

[1]Note that this is the same argument for the analogous document frequency over collection frequency, in general IR.

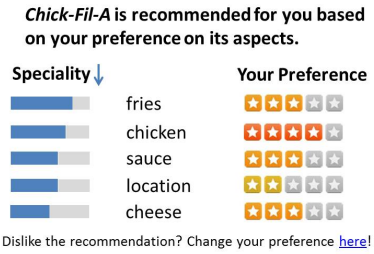


Figure 3: Mock user interface for showing the rationale behind recommending *Chick-Fil-A* to a user.

aspects; similarly, for users and aspects. The overall solution can be seen as a semi-supervised learning process [36] on graphs – with the prior preference as labeled data, the algorithm propagates the labels to other unlabeled vertices.

3.4 Discussion

There are three properties of TriRank that merit a more detailed discussion: explainability, insensitivity to noisy aspects, and structural ambiguity.

Explainability. As TriRank ranks items in an easily explainable way, it provides users more transparency in understanding the system behavior. We can attribute recommendations to the top-ranked aspects matching the target user and recommended item. Figure 3 shows a mock-up interface for explaining recommendation based on aspects, inspired by tag-based explanation [28]. Aspects are sorted by item's speciality by default, but a user can sort according to her predicted preference. This property makes the system scrutable [27], allowing a user to control how the system utilizes her reviews. For example, if a user dislikes a recommendation due to inaccurately-captured aspects or she has updated preference not captured in her reviews, she can edit her aspect preference. TriRank can then encode the new aspect query vector (*i.e.*, $\vec{a}_0$) and return the revised recommendations (shown later in Section 4.3).

This is a major advantage over the recent solutions [19, 34] which integrate reviews using a latent factor model (LFM). Such LFM methods only provide single-shot recommendation where the rationale for the recommendation is opaque. In contrast, the scrutability provided by our method easily allows recommendation to become a cyclical process – a user can iteratively interact with the recommender system, where her actions improve the system's recommendations in turn. This iterative and scrutable nature are becoming increasingly important for real-world recommender systems [27].

Insensitivity to noisy aspects. As mentioned in Section 2, extracted aspects are noisy. For noisy aspects which are outliers (*e.g.,* "picturemy", "150"), they usually occur less frequently in reviews as compared with those from normal aspects. As such, they will have smaller edge weights in the tripartite graph, thus exerting less impact on the ranking (see x_{jk} and y_{ik} of Eq. (3)). For noisy aspects which are domain-specific stop words, although they have high frequency in reviews, they actually distribute evenly for all users and items (*i.e.,* column vector of S_X and S_Y of Eq. (4)). As a result, they will contribute evenly across all items' ranking scores, hence not changing the relative ranking among items. As such, TriRank is relatively insensitive to noisy aspects (either outliers or stop words).

Structural ambiguity. Given a list of triples as inputs, we can uniquely represent them as a tripartite graph, but not vice versa. This is because in the tripartite graph, we cannot attribute a specific edge to an input tuple, as the conversion to the tripartite graph does not represent tuple association. More specifically, let the reviewed aspects of user u_i and item p_j be A_i and A_j, respectively. Assume u_i interacts with p_j, then the tripartite structure can not differentiate the aspects in $A_i \cap A_j$ for the interaction r_{ij}. When such ambiguities occur, they can act like unseen additional inputs, which can complement the actual observed data in a manner similar to transitive reasoning.

Table 3: Statistics of datasets in evaluation.

Dataset	Review#	Item#	User#	Avg	Density
Yelp	114,316	4,043	3,835	29.8	0.74%
Amazon	55,677	14,370	2,933	19.0	0.13%

"Avg" denotes the average number of reviews per user.

4. EXPERIMENTS

We first introduce our experimental settings, and then compare its performance with other methods. We then study the utility of aspects in depth. Finally, we perform a few case studies of TriRank's recommendation output.

Datasets. We experiment with two publicly accessible datasets: Yelp[2] and Amazon Electronics [19].

1. Yelp. This is the Yelp Challenge dataset published on April 2013. It includes 11,537 items, 229,907 reviews and 45,981 users. The dataset is very sparse – 49.6% of users only made one review, making it difficult for evaluation.

2. Amazon. This dataset contains user ratings and reviews on Amazon products of Electronics category, published by [19]. The original dataset contains over 800K users, 80K items and 1.3M reviews. It is more sparse than the Yelp dataset – with 77.9% of users making only one review.

Following the common practice by other works [4, 34] in evaluating recommender algorithms, we filter out the items and users having fewer than 10 reviews. Table 3 summarizes the statistics of the filtered datasets. We split each dataset into three parts for training, validation and testing by time. For each user, we sort her reviews in chronological order. The first 80% are used for training, and the remaining most recent 20% are randomly split as validation set (for parameter tuning only) and test set (for evaluation).

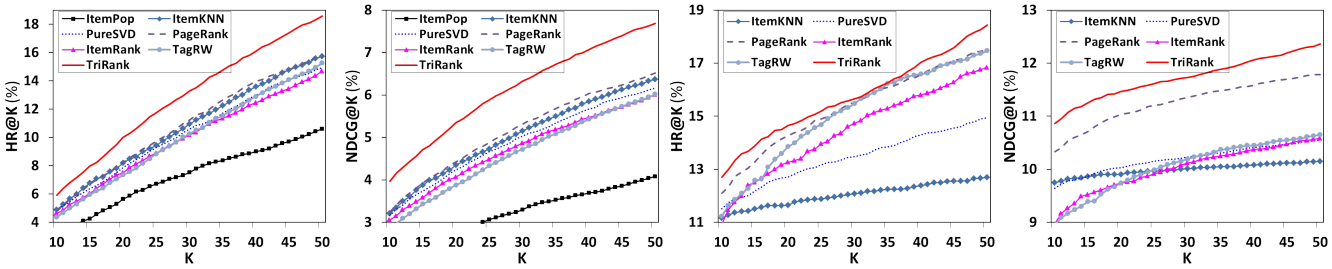
Evaluation Metrics. Given a user, each algorithm produces a ranked list of items. To assess the ranked list with the ground-truth item set (GT), we adopt *Hit Ratio* (HR), which has been commonly used in top-N evaluation [15, 29]. If a test item appears in the recommended list, it is deemed a hit. HR is calculated as:

$$HR@K = \frac{Number\ of\ Hits\ @K}{|GT|}. \tag{5}$$

As the HR is recall-based metric, it does not reflect the accuracy of getting top ranks correct, which is crucial in many real-world applications. To address this, we also adopt *Normalized Discounted Cumulative Gain* (NDCG), which assigns higher importance to results at top ranks, scoring successively lower ranks with marginal fractional utility:

$$NDCG@K = Z_K \sum_{i=1}^{K} \frac{2^{r_i} - 1}{log_2(i+1)}, \tag{6}$$

[2] `http://www.yelp.com/dataset_challenge`



(a) Yelp – Hit Ratio **(b)** Yelp – NDCG **(c)** Amazon – Hit Ratio **(d)** Amazon – NDCG

Figure 4: Performance evaluated by Hit Ratio and NDCG from position 10 to 50 (*i.e.*, K).

where Z_K is the normalizer to ensure the perfect ranking has a value of 1; r_i is the graded relevance of item at position i. We use the simple binary relevance for our work: $r_i = 1$ if the item is in the test set, and 0 otherwise.

For both metrics, larger values indicate better performance. In the evaluation, we calculate both metrics for each user in the test set, and report the average score.

Baselines. We compare TriRank with the following commonly used and competitive methods in top-N evaluation:

1. **Item Popularity (ItemPop)**. Items are ranked by their popularity judged by number of ratings. Although it is not personalized, it is surprisingly competitive in top-N evaluation [3], as users tend to consume popular items.
2. **ItemKNN [24]**. This is standard item-based CF, and has been used commercially by Amazon [16] and MovieLens [28]. We adopt cosine similarity to measure the similarity among items. We test the method with different number of neighbors, finding that using all neighbors works best.
3. **PureSVD [3]**. A state-of-the-art for top-N recommendation, which performs *Singular Value Decomposition* on the whole matrix, thus directly considering all instances. unlike other latent factor methods that optimize against error only on rated instances. This property is important when applying matrix factorization models for top-N evaluation. We follow the implementation in [3], using the package SVDLIBC[3], tuning the number of latent features from 10 to 200, finding the best performance at 30.
4. **Personalized PageRank [8]**. This is a widely used graph method for top-N recommendation, *e.g.,* by [15, 29]. We perform Personalized PageRank on the user–item graph[4], and set the personalized vector same with TriRank's prior item vector $\vec{p}_0$. The damping parameter (*i.e.*, weight of the personalized vector) was respectively optimized to 0.9 and 0.3, for Yelp and Amazon datasets.
5. **ItemRank [6]**. This is another graph based method that recommends based on the item–item correlation graph. Similar to Personalized PageRank, we set the personalized vector identically as $\vec{p}_0$ and tune the damping factor.
6. **TagRW [35]**. This is the state-of-the-art method to model tags for top-N item recommendation. As we have mentioned that aspects are similar with tags in terms of format, we need to compare with such a method to study how tag-aware methods perform on the task of modeling aspects. TagRW enhances ItemRank [6] by incorporating tags into building the item–item graph and performing an additional random walk on user–user graph. We feed aspects (all the same settings with TriRank) as tags into the method, and tune the five parameters of the method in a sequential way, as suggested by their paper.

As the existing review-aware methods [4, 17, 19, 34] are optimized for predicting observed ratings, it is unfair to compare with them for top-N evaluation. We validate this by evaluating the *Hidden Factors and Topics* model [19], which is state-of-the-art for review-aware rating prediction. It achieves poor top-N performance in our settings, worse than Item Popularity. Thus we do not further compare with other methods designed for rating prediction.

TriRank has six regularization parameters to tune – three for the traditional collaborative filtering effect (α, η_U, η_I) and three for the aspect filtering effect (β, γ, η_A). As performing grid-search on all six parameters simultaneously is time-consuming, we separately tune those for CF and those for aspects – first fixing β, γ, η_A as 0, searching for α, η_U, η_I; then performing the reverse with the optimal α, η_U, η_I. Performance was stable across many parameter settings, thus we report results for a selected set.

4.1 Performance Study

Figure 4 plots the performance when K ranges from 10 to 50, and Table 4 shows the concrete scores obtained at position 10 and 50. We first focus on results of the Yelp dataset. From Figure 4(a) and (b), we see that both metrics exhibit the same trend: TriRank performs best, outperforming all other methods with a large margin; followed by PageRank and ItemKNN, where PageRank performs slightly better than ItemKNN. PureSVD, ItemRank and TagRW obtain similar HR scores, while NDCG tells the quality of ranking: PureSVD ranks correct items higher than ItemRank and TagRW. Item Popularity performs the worst, indicating the importance of modeling users' personalized preferences, rather than just recommending popular items.

Surprisingly, TagRW does not always outperform ItemRank, although it utilizes additional aspect information. It shows that their method for integrating tags into recommendation may not be effective for aspects. Analyzing the results, we believe that there are two reasons responsible for TagRW's inferior performance. First, they integrate aspects by transforming to the item–item and user–user similarity graph, which may lead to signal loss. Second, noisy aspects may have an adverse impact on their method, and the impact is highly dependent on the similarity measure they use. Our proposed TriRank mitigates both of these nega-

[3] `http://tedlab.mit.edu/~dr/SVDLIBC`

[4] We also evaluated Personalized PageRank on the user–item–aspect tripartite graph. Even with optimal tuning of each edge and vertex type, performance did not improve; thus we only report Personalized PageRank's performance on the standard user–item graph.

Table 4: Performance of compared methods and TriRank at rank 10 and 50.

Dataset	Yelp				Amazon			
Metric(%)	HR@10	NDCG@10	HR@50	NDCG@50	HR@10	NDCG@10	HR@50	NDCG@50
ItemPop	3.06	1.85	10.61	4.08	2.38	1.36	6.13	2.37
ItemKNN	4.90	3.21	15.72	6.37	11.17	9.75	12.69	10.15
PureSVD	4.79	3.17	14.94	6.16	11.52	9.64	14.94	10.55
PageRank	4.79	3.24	15.90	6.52	12.10	10.33	17.49	11.78
ItemRank	4.64	3.05	14.67	6.01	10.92	8.97	16.84	10.58
TagRW	4.36	2.85	15.25	6.02	11.23	8.96	17.47	10.65
TriRank	**5.92****	**3.97****	**18.58****	**7.69****	**12.71****	**10.86****	**18.44****	**12.36****

"**" denotes the statistical significance for $p < 0.01$.

tive factors by 1) directly modeling aspects into the user–item relation as a tripartite graph, and 2) ranking vertices by regularizing the tripartite graph.

With respect to the Amazon dataset, TriRank again achieves the best performance on both metrics ($p < 0.01$ in most cases). Focusing on Figure 4(c) that shows the HR scores, TriRank is followed by PageRank and TagRW, which significantly outperform other methods. When K is set to 30–40, the HR differences between TriRank and PageRank and TagRW are small, but the NDCG reveals significant gaps among the three methods, indicating that TriRank successfully orders the correct items more effectively than the other two. Meanwhile, TagRW and ItemRank better PureSVD, as evaluated by HR ($K \geq 15$), but not by NDCG, which indicates the matches of TagRW and ItemRank actually occur at lower ranks. This reinforces our point that a good recall score does not necessarily translate to a high-quality ranking, hence the necessity to evaluate by ranking based measures, such as NDCG. ItemKNN performs worst among all the non-trivial personalized methods. ItemPop performed very weak, and as such, was entirely omitted in the figure to better highlight the performance of the other methods.

Looking at the interesting performance variations across the two datasets, we first notice that ItemPop only performs well on the Yelp dataset. We believe this is caused by consumption behavior differences across the two domains – people may visit popular restaurants or businesses rated in Yelp, but only purchase certain products on demand from Amazon. Similarly, ItemKNN performs strongly on the Yelp dataset (better than PureSVD), but poorly on the Amazon. One possible reason comes from data sparsity: as in Table 3, each item of the Amazon dataset only has 3.9 reviews on average. In such cases, the similarity measure fails in neighbor-based CF. An interesting finding is that PageRank consistently outperforms ItemRank, although both rely on Personalized PageRank with the same personalized vector. We believe the explanation is due to the fact that ItemRank ranks based on the transformed item–item correlation graph. Transforming the user–item graph to an item–item correlation graph will lose signal especially when the data is sparse, *e.g.*, when two items have no common users reviewing them. In such cases, it is more beneficial to directly rank from the user–item graph. Finally, TagRW betters ItemRank only on the Amazon dataset, indicating that the tag-based method to integrate aspects does not lead to consistent improvement. Our proposed TriRank achieves the best performance on the two datasets evaluated by both metrics, demonstrating its superiority in providing personalized item ranking to users by mining aspects in reviews.

4.2 Utility of Aspects

Table 5: Performance of TriRank with different parameter settings at rank 50.

Dataset	Yelp		Amazon	
Settings	HR	NDCG	HR	NDCG
0. All set	**18.58**	**7.69**	**18.44**	**12.36**
1. $\beta = 0$ (no item–aspect)	17.05	6.91	16.23	11.31
2. $\gamma = 0$ (no user–aspect)	**18.52**	**7.68**	**18.40**	**12.36**
3. $\eta_A = 0$ (no aspect query)	18.21	7.51	17.62	12.10
4. $\beta, \gamma, \eta_A = 0$ (no aspects)	17.00	6.90	15.97	11.16
5. $\alpha = 0$ (no user–item)	11.67	4.84	10.32	5.08

There are natural issues about aspects that we also wish to address:

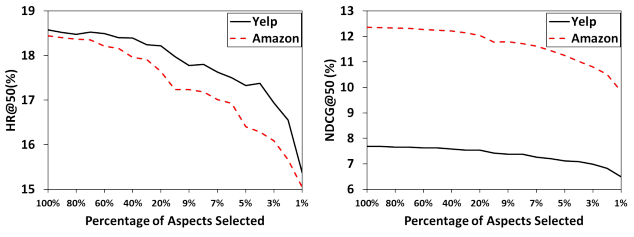
1. How do the aspect-related components (*e.g.,* item–aspect and user–aspect) contribute to the performance?

2. How does the quality of aspects impact the performance? Can TriRank handle the inherent noise in automatically extracted aspects well?

4.2.1 Aspect Importance Study

As TriRank is modular, with parameters for each type of vertices and edges, it is easy to answer the first question by varying the aspect-related parameters: β and γ to control the smoothness for the item–aspect and user–aspect relation, respectively, and η_A for the aspect query vector. Setting a parameter to 0 removes the corresponding effect.

Table 5 shows TriRank's performance with different parameter settings, evaluated at rank 50. In both datasets, when item–aspect smoothness is eliminated by setting $\beta = 0$ (Row 1), performance drops significantly. This indicates the importance of item–aspect relation, and validates our motivation that modeling user preference via decomposed aspects can yield more fidelity over modeling user–item ratings only. In contrast, when user–aspect smoothness is removed (Row 2), the performance remains unchanged. This shows that user–aspect smoothness contributes substantially less to TriRank's performance; however, we note that at least the target user's portion of the user–aspect relation can not be removed in recommendation, as the rated aspects of a user form her aspect query vector needed for recommendation. Row 3, which exhibits low performance, validates this point, as here we have removed the aspect query vector by setting η_A as 0. If we remove the modeling of aspects in its entirety (Row 4), TriRank degrades to the *Bipartite User–*



(a) Hit Ratio versus Aspects **(b)** NDCG versus Aspects

Figure 5: TriRank performance with respect to percentage of top aspects selected.

Item Ranking algorithm [9] on the user–item graph, and performs even worse. This further reveals the importance of modeling aspects for quality recommendation.

To round out our study, Row 5 shows the performance of removing user–item smoothness, which encodes standard CF in the terminology of our regularization approach. The resulting performance is worst among all settings. The user–item relation is still fundamental to model and is most important, followed by the importance of the item–aspect smoothness; user–aspect smoothness contributes least and may be removed. However, the reviewed aspects of a particular target user are still critical for capturing her personalized preference.

4.2.2 *Aspect Quality Study*

For the second question, we first rank aspects by their tf×idf score in the item–aspect matrix, and then select top scoring aspects to build the tripartite graph and inspect TriRank's performance.

Figure 5 shows TriRank's performance with respect to percentage of top aspects selected. As we can see, both datasets show the same trend: when the filtering ratio is moderate, performance remains largely unchanged. The filtering inflection point for both datasets is around 30%. When we filter out aspects beyond this point, performance starts to drop significantly. This indicates that the aspects with $tf \times idf$ play a dominant role in modeling users' preference for quality recommendation. We further validate this conclusion by filtering in the reverse direction (not shown; *i.e.*, dropping the top x% aspects ranked by $tf \times idf$), finding that even a small amount of filtering (1%) leads to significant degradation. We conclude that one can safely filter out the low $tf \times idf$ scoring aspects for efficiency, as they contribute less to recommendation performance.

Interestingly, TriRank's performance does not improve when only high $tf \times idf$ aspects are utilized. Although slight improvements can be obtained with tuning, they are not statistically significant. We further evaluate TriRank with the 124 high-quality aspects selected in [34]'s work on the same Yelp Challenge dataset. Even after all parameters are re-tuned, test performance is not improved. This validates the nice property of TriRank of being relatively insensitive to noisy aspects, which are also expected to have less impact in the ranking outputs as previously explained (Section 3.4). TriRank can effectively utilize the merits in the automatically extracted aspects, without the need to filter out noisy aspects manually. Compared to the *Explicit Factor Model* [34] that integrates only high-quality aspects into a matrix factorization model and then generates recommendations by optimizing the predicted ratings in an opaque manner, TriRank is more transparent in leveraging aspects and also is more tolerant of low-quality aspects.

Figure 6: Training reviews of a sampled Yelp user.

4.3 Case Studies

While macro-level empirical analysis are useful, it is also instructive to examine actual results to better understand the outputs of TriRank. To this end, we give two case studies drawn from the Yelp dataset to demonstrate its explainability and scrutability.

4.3.1 *Explainability*

Figure 6 shows four training reviews of a sampled user[5]. From the first two reviews, we can see the user is interested in "chicken", although she gives low ratings to the two businesses. In the heldout test set, she reviews the business *Chick-Fil-A* with a comment "*I love Chick-Fil-A… the spicy chicken sandwiches [sic], the lemonade, the soup, the brownies*", which further validates her preference for "chicken". As expected, TriRank ranks *Chick-Fil-A* highly (6^{th} position), mainly due to chicken being a top aspect of this business (3 of its 7 training reviews mentioned "chicken"). Examining the top items returned by PageRank, none have "chicken" as a top aspect, and most of them are popular items with more than 100 reviews. This is because random walk models are easily biased to popular items, as reported by [1]. Moreover, the third and fourth reviews show that the user is also interested in "shrimp". As a result, TriRank ranks the seafood restaurant *Red Lobster* highly in the 3^{rd} position. Although it is evaluated as a loss as the test set does not contain the item, when we checked her complete history in Yelp.com, we found she actually reviewed this restaurant later (outside of the dates in the Yelp Challenge dataset), mentioning "shrimp". Again, the recommended *Red Lobster* is not a popular item with only 7 training reviews. This case study demonstrates TriRank's capability of recommending more relevant and personalized items (not just popular items) according to a user's reviewed aspects.

4.3.2 *Scrutability*

Another key property of our TriRank instantiation is the encoding of aspect query vector $\vec{a}_0$, serving as the gateway to edit a target user's preference. We simulate the process on a sampled user[6].

For this user, 9 of the 14 training reviews mentioned "service", which is the top aspect, followed by "beer". However in the test set, he reviews the business "*Total Wine & More*", whose top aspects are "wine" and "liquor". In this case, both

[5]User ID "*8fTTvS499XCz4oP49kxq8A*". Only part of each review is shown as the original is long.
[6]User ID "*omoEjYFKVV7e-DtnezeUOw*".

TriRank and PageRank fail to recommend the correct item, and all top items returned do not have wine as a specialty. We simulate user feedback by editing the aspect query vector to set "wine" to a higher value, and re-run TriRank with all other parameters unchanged. In the updated ranked list, 8 of the top-10 items have "wine" as the top aspect, and the correct item "*Total Wine & More*" is ranked in 2^{nd} position.

5. RELATED WORK

While collaborative filtering systems perform well in general, their performance suffers when the amount of user feedback is insufficient (*i.e.,* in cold-start). Another important shortcoming is that they do not capture the rationale for user's rating, and thus can not accurately capture a target user's preference. To overcome these weaknesses, various forms of side information have been incorporated into CF, including tags [35], geo-location [11] and user reviews [19]. In this section, we first study the area of review-aware methods, and then examine graph-based recommendation techniques as TriRank falls into this category.

5.1 Review-aware Recommendation

User reviews have been utilized to assist recommender systems in many domains, for movies [4], hotels [20], restaurants [5] and e-commerce [19]. Regardless of domain, we can categorize the approaches based on how reviews are integrated into the recommender: 1) word-based, 2) sentiment-based, and 3) aspect-based methods.

Word-based. These approaches directly factorize the review words into CF. [26] used words to measure similarity, whereas [11] modeled each word as a latent vector within the latent factor model. As the original word space is large and sparse, dimension reduction techniques have been adopted. McAuley and Leskovec [19] employed *Latent Dirichlet Allocation* (LDA) [2] to winnow down the word space, and combined with the standard latent factor model. Subsequently, [17, 30] adopted a full Bayesian treatment to combine topics and latent factors for rating prediction.

Sentiment-based. These approaches utilize the user's explicitly mentioned opinions on items. [21] proposed to fill in the missing ratings with a predicted sentiment score before applying neighbor-based CF. [22] built a user–item opinion matrix, where each entry was the aggregated sentiment score of a review, and then applied traditional CF on the opinion matrix. More recently, [4] proposed an integrated graphical model to jointly model the sentiment and ratings for movie recommendation.

Aspect-based. Our work falls into this category. Early work [5] along this line manually annotated six aspects in the restaurant domain (*e.g.,* service, ambiance, *etc.*), and classified sentences with respect to these aspects. Their regression method validated the usefulness of aspects for rating prediction. Musat *et al.* [20] built topical profiles of users and items from reviews, and predicted ratings at the topic level. They tested two ways to extract topics — LDA and opinion word frequency — finding the latter produced higher quality topics. Recently, Zhang *et al.* [34] jointly factorized the user–item rating matrix by inserting aspects, decomposing it into item–aspect and user–aspect matrices, where the aspects were automatically extracted.

Several hybrid methods have also integrated aspect and sentiment [4, 5, 34] as they are closely related. In contrast, our work focuses on integrating aspect into CF for explainable recommendations, thus we forgo incorporating sentiment, to minimize the reliance on sentiment analysis accuracy. Compared to the above review-aware works, our method explores a graph model to integrate aspects, which has not been previously been investigated. Moreover, our proposed TriRank affords the recommender a finer degree of user interaction — aspect preference — allowing for both more accurate and transparent recommendation.

5.2 Graph-based Recommendation

Graphs form a natural representation for modeling the relationship among data objects. In recommender systems, graph models have been used widely and commercially (*e.g.,* by Twitter [7] and YouTube [1]), due to their good interpretability in generating recommendations. A typical workflow is first representing items as vertices of a graph, and then admitting recommendation as a vertex-ranking problem. For example, in YouTube video recommendation, [1] built a user–video co-view graph for video items, adopting label propagation for selecting important videos.

Besides directly working on the heterogeneous user–item graph, another family of approaches [6, 18, 35] projects the user–item graph to an item–item graph (as the ranking target is item), and then applies homogeneous graph ranking techniques, such as personalized PageRank [8]. Specifically, [6] recommended based on an item correlation graph, where entries denoted the likelihood that two items are co-rated. [18] proposed an item preference graph, where entries denoted the strength that users prefer one item over another.

We point out that a key advantage of retaining the user–item structure is that additional information can be easily incorporated by adding new types of vertices. However, existing ranking algorithms do not cater to heterogeneous structures, as they have primarily focused on homogeneous graphs [8, 36] or bipartite graphs [9]. Thus, a corresponding algorithm must be devised to suit the specific heterogeneous graph and ranking purpose. For example, [29] modeled long-term and short-term user preference by introducing session nodes, ranking vertices by propagating user preference via *Breadth-First-Search*; [15] incorporated contexts (*e.g.*, location, time) as vertices in the user's side, and adjusted PageRank for ranking in such a mixed bipartite graph. Similar to above works, our method retains the user–item structure, extending it to a user–item–aspect tripartite graph for modeling aspects. One major difference is that we specifically consider the ternary relationship between user, item and aspect, which has not been studied before.

6. CONCLUSION

We have studied how to utilize item aspects in user reviews for top-N recommendation. We model the user–item–aspect relation as a tripartite graph, and propose TriRank, a generic algorithm for ranking the vertices of tripartite graph by regularizing the smoothness and fitting constraints. We employ TriRank for review-aware recommendation, where the ranking constraints directly model the collaborative and aspect filtering, and also personalization. TriRank achieves state-of-the-art performance over two public review datasets, even with automatically extracted aspects that have significant noise. We validate TriRank as being largely insensitive to low-quality aspects, a desirable property when porting to other domains as it avoids manual efforts in filtering out noisy aspects. Most importantly, TriRank's incorporation

of aspects provides users with more transparency into the recommender system behavior and affords user interaction to further improve recommendations.

Our introduction to TriRank is in its basic form, which has already shown significant utility. TriRank can be further extended in many ways, for example by adopting a more suitable loss function, or by extending to more general n-partite graphs. We hope to extend TriRank to such cases, by jointly modeling the additional information latent in user reviews that may be useful for recommendation: temporal factors [9], category taxonomies [10] and sentiment [21]. Another open issue is in optimal parameter settings. Our current work reports results when the regularization parameters are set uniformly for all users; however in some exploratory work, we found that a specific parameter setting for a subset of users improves performance. This reveals the potential for improvement by setting parameters individually or for groups of similar users based on their preference on aspects.

Acknowledgement

We thank Yongfeng Zhang, Haochen Zhang and the Tsinghua Information Retrieval group for developing and sharing their aspect extraction tool. We would also like to thank the anonymous reviewers for their valuable comments.

7. REFERENCES

[1] S. Baluja, R. Seth, and D. Sivakumar. Video suggestion and discovery for Youtube: Taking random walks through the view graph. In *Proc. of WWW '08*, pages 895–904, 2008.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of RecSys '10*, pages 39–46, 2010.

[4] Q. Diao, M. Qiu, C.-Y. Wu, A. J. Smola, J. Jiang, and C. Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *Proc. of KDD '14*, pages 193–202, 2014.

[5] G. Ganu, N. Elhadad, and A. Marian. Beyond the stars: Improving rating predictions using review text content. In *Proc. of WebDB '09*, 2009.

[6] M. Gori and A. Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In *Proc. of IJCAI '07*, pages 2766–2771, 2007.

[7] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. In *Proc. of WWW '13*, pages 505–514, 2013.

[8] T. H. Haveliwala. Topic-sensitive PageRank. In *Proc. of WWW '02*, pages 517–526, 2002.

[9] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama. Predicting the popularity of web 2.0 items based on user comments. In *Proc. SIGIR '14*, pages 233–242, 2014.

[10] X. He, M.-Y. Kan, P. Xie, and X. Chen. Comment-based multi-view clustering of web 2.0 items. In *Proc. of WWW '14*, pages 771–782, 2014.

[11] L. Hu, A. Sun, and Y. Liu. Your neighbors affect your ratings: On geographical neighborhood influence to rating prediction. In *Proc. of SIGIR '14*, pages 345–354, 2014.

[12] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proc. of KDD '04*, pages 168–177, 2004.

[13] W. Jin and H. H. Ho. A novel lexicalized hmm-based learning framework for web opinion mining. In *Proc. of ICML '09*, pages 465–472, 2009.

[14] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer US, 2011.

[15] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee. Random walk based entity ranking on graph for multidimensional recommendation. In *Proc. of RecSys '11*, pages 93–100, 2011.

[16] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, Jan 2003.

[17] G. Ling, M. R. Lyu, and I. King. Ratings meet reviews, a combined approach to recommend. In *Proc. of RecSys '14*, pages 105–112, 2014.

[18] N. N. Liu and Q. Yang. Eigenrank: A ranking-oriented approach to collaborative filtering. In *Proc. of SIGIR '08*, pages 83–90, 2008.

[19] J. McAuley and J. Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proc. of RecSys'13*, pages 165–172, 2013.

[20] C.-C. Musat, Y. Liang, and B. Faltings. Recommendation using textual opinions. In *Proc. of IJCAI '13*, pages 2684–2690, 2013.

[21] N. Pappas and A. Popescu-Belis. Sentiment analysis of user comments for one-class collaborative filtering over ted talks. In *Proc. of SIGIR '13*, pages 773–776, 2013.

[22] Š. Pero and T. Horváth. Opinion-driven matrix factorization for rating prediction. In *Proc. of UMAP '13*, pages 1–13. 2013.

[23] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt. Bpr: Bayesian personalized ranking from implicit feedback. In *Proc. of UAI '09*, pages 452–461, 2009.

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of WWW '01*, pages 285–295, 2001.

[25] A. Smola and R. Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines*, volume 2777 of *LNCS*, pages 144–158. Springer, 2003.

[26] M. Terzi, M. Rowe, M.-A. Ferrario, and J. Whittle. Text-based user-knn: Measuring user similarity based on text reviews. In *Proc. of UMAP '14*, pages 195–206. 2014.

[27] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*, pages 479–510. Springer US, 2011.

[28] J. Vig, S. Sen, and J. Riedl. Tagsplanations: Explaining recommendations using tags. In *Proc. of IUI '09*, pages 47–56, 2009.

[29] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proc. of KDD '10*, pages 723–732, 2010.

[30] Y. Xu, W. Lam, and T. Lin. Collaborative filtering incorporating review text and co-clusters of hidden user communities and item groups. In *Proc. of CIKM '14*, pages 251–260, 2014.

[31] L. Zhang and B. Liu. Aspect and entity extraction for opinion mining. In *Data Mining and Knowledge Discovery for Big Data*, volume 1, pages 1–40. Springer, 2014.

[32] L. Zhang, B. Liu, S. H. Lim, and E. O'Brien. Extracting and ranking product features in opinion documents. In *Proc. of COLING '10*, pages 1462–1470, 2010.

[33] Y. Zhang, H. Zhang, M. Zhang, Y. Liu, and S. Ma. Do users rate or review?: Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proc. of SIGIR '14*, pages 1027–1030, 2014.

[34] Y. Zhang, M. Zhang, Y. Zhang, Y. Liu, and S. Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proc. of SIGIR '14*, pages 83–92, 2014.

[35] Z. Zhang, D. D. Zeng, A. Abbasi, J. Peng, and X. Zheng. A random walk model for item recommendation in social tagging systems. *ACM Transactions on Management Information Systems*, 4(2):1–24, 2013.

[36] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, pages 321–328, 2004.