Vectors of Pairwise Item Preferences

Gaurav Pandey[1], Shuaiqiang Wang[2], Zhaochun Ren[2] and Yi Chang[3]

[1] University of Jyvaskyla, Finland
gaurav.g.pandey@jyu.fi
[2] JD.com, China
wangshuaiqiang1,renzhaochun@jd.com
[3] Jilin University, China yichang@ieee.org

Abstract. Neural embedding has been widely applied as an effective category of vectorization methods in real-world recommender systems. However, its exploration of users' explicit feedback on items, to create good quality user and item vectors is still limited. Existing neural embedding methods only consider the items that are accessed by the users, but neglect the scenario when a user gives high or low rating to a particular item. In this paper, we propose *Pref2Vec*, a method to generate vector representations of pairwise item preferences, users and items, which can be directly utilized for machine learning tasks. Specifically, *Pref2Vec* considers users' pairwise item preferences as elementary units. It vectorizes users' pairwise preferences by maximizing the likelihood estimation of the conditional probability of each pairwise item preference given another one. With the pairwise preference matrix and the generated preference vectors, the vectors of users are yielded by minimizing the difference between users' observed preferences and the product of the user and preference vectors. Similarly, the vectorization of items can be achieved with the user-item rating matrix and the users vectors. We conducted extensive experiments on three benchmark datasets to assess the quality of item vectors and the initialization independence of the user and item vectors. The utility of our vectorization results is shown by the recommendation performance achieved using them. Our experimental results show significant improvement over state-of-the-art baselines.

Keywords: Vectorization, Neural Embedding, Recommender systems

1 Introduction

Based on neural networks, neural embedding has emerged as a successful category of vectorization techniques in recommender systems [8, 2], among which *word2vec* [22, 23] is a fundamental and effective algorithm. It was initially proposed for natural language processing problems and considers two states 1 or 0 for each word, representing either appearance or absence of the word in documents. It assumes that the words appearing closer to each other would have higher statistical dependence. Given its effectiveness, many variants have been proposed for machine learning problems, such as name speech recognition [25],

entity resolution [19], machine translation [30], social embedding [12, 24] and recommender systems [11, 1]. Several pioneering efforts have been applied to real-world recommendation scenarios with neural embedding like *prod2vec* [11] and *item2vec* [1], that have been proposed by straightforwardly employing *word2vec*, where each user is considered as a document, and each item is simply regarded as a word. Consequently each item can only have two possible states 1 or 0, representing whether the user has performed a particular action (e.g. purchase, click, etc.) on the item or not. Using sets and sequences of items for each user, they learn the vector representations of the items.

Though such representations create good quality item vectors for some tasks, they lack the functionality to capture higher levels of granularities of users' feedback for vectorization. This could lead to incorrect interpretations, as the top-ranked item and low-ranked items would be treated equally. Thus it is expected to severely limit the vectorization quality for many tasks like calculating item similarities for single item recommendations, clustering user or items, etc. Currently, the efforts are limited for neural embedding-based methods, especially for datasets involving ratings. Therefore, we investigate the neural item embedding problem, to create quality vectorization for items using users' historical rating information with higher granularities (e.g. ratings in range 1 to 5).

To solve this problem, we propose *Pref2Vec* which involves three components: (1) The first step transforms the given user-item rating matrix into a users' pairwise preference matrix. On doing this, each pairwise preference of items has one of the two statuses i.e. occurrence or absence, which is similar to the situation of words in *word2vec*. (2) Then we employ neural embedding to create vector representations for pairwise item preferences by maximizing the likelihood estimation of the conditional probability of each pairwise item preference given another one. Using these preference vectors, the vectors of users can be generated by minimizing the difference between users' observed preferences and the product of the user and preference vectors in the second step of *Pref2Vec*. (3) In the last step, using the user vectors, the item vectors are generated similarly by minimizing the difference between items' observed ratings and the product of user and item vectors.

We evaluate the effectiveness of our *Pref2Vec* method in three experimental tasks on movie recommendation datasets to demonstrate its promising performance, where items are the movies for which user ratings are provided. (1) In the first task, we assess the quality of item vectors, by considering the movie genres as ground-truths. We find the similarities between each pair of items, using the generated item vectors and then using the ground truth (genres). The difference between these two similarities for item pairs are considered as the errors, using which we are able to compute RMSE (root mean squared error) and MAE (mean absolute error), as a quality measures for comparison. We contrast the quality of our item vectors with the quality of item vectors of other standard techniques, like: a) item vectors generated using matrix factorization and b) neural embedding item vectorization by using the sets of items that are rated by users as words. (2) In the second task, we run the vectorizations of the user and

item vectors multiple times. We calculate the average variance of the generated values and the mean average covariance of the generated vectors, to establish that our vectorization process is highly independent of initialization. We contrast this with the vectorizations generated by matrix factorization. (3) Moreover, we compare the recommendation ranking generated using *Pref2Vec* with the standard collaborative filtering algorithms using the NDCG measure. Our results for these experimental tasks show performance gains over the comparison partners.

2 Related Work

Vectorization techniques are of great importance in machine learning. Specially in the area of natural language processing, neural embedding techniques for vectorization of words have been used in many applications [27, 29, 2, 8, 28, 30, 25]. Neural embedding techniques assume that the words that occur close to each other in the text are more dependent than the words that are far off. However, vectorization techniques using neural networks were inefficient to train, especially when the size and vocabulary of the dataset increased. But, the widely used word embedding technique *word2vec* that was introduced a few years ago, made creation of vector representations of words very efficient. It employs highly scalable skip-gram language model, that is fast to train and preserves the semantic relationships of the words in their vector representations. This technique for word embedding has recently shown considerable improvement in applications like name entity resolution [19] and word sense detection [3].

The success of *word2vec* has probably lead to the adoption of the neural embedding techniques in domains other than word representations. Djuric et al. [9] used vectorization of paragraphs as well as vectorization of words contained in each paragraph to create a hierarchical neural embedding framework. Also, Le et al. [20] created an algorithm that learns vector representations of sentences and text documents. They represent each document as dense vector that is utilized to predict words in the document. Moreover, Bordes et al. [4] have introduced the approach that embeds entities and relationships of multi-relational data in low-dimensional vector spaces, to be used for text classification and sentiment analysis tasks. Socher et al. [26] attempted to improve this approach by representing entities as an average of their constituting word vectors. Also, there have been recent efforts to learn the vector representations of nodes in graphs [24, 12].

Moreover, several recent recommendation applications have employed neural word embedding. of *prod2vec* and *user2vec* by Grbovic et al. [11]. The *prod2vec* model creates vector representations of products by employing neural embedding on sequences of product purchases, where each product purchase is considered as a word. Whereas, the *user2vec* model considers a user as a global context in order to learn the vector representations of user and products. Similarly, *item2vec* [1] employs neural embedding on sets of items on which the user has taken action (e.g. songs played or products purchased), while ignoring the sequential information. The experimental results for these techniques show their effectiveness.

Although there have been many applications of neural embeddings in various areas including collaborative filtering, to the best of our knowledge, among the

available neural embedding techniques on the rating information, there is no straightforward way to incorporate different levels of item ratings. He et al. [13] have utilized deep neural network frameworks for recommendation, but they also consider items in 1 and 0 state. Besides, there has not been an attempt to generate and utilize preference vectors. Hence, in this paper we attempt to generate preference vectors as an intermediate step, which can be utilized to generate good quality user and item vectors for various data mining tasks.

3 Problem Formulation

In this section, we formulate the neural rating vectorization problem, aiming to create vector representations for users and items by considering users' historical rating preference on items. Since matrix factorization can be actually regarded as traditional preference vectorization technique, let's firstly review its definition.

Consider a set of users U with m users, a set of items I with n items and a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users. Each element $r_{u,i}$ of the uth row and ith column of R is the rating given by a particular user $u \in U$ for the item $i \in I$, where most of the elements in R are unknown as users generally can provide ratings only for a very small number of items. The objective of the rating vectorization problem is to generate a vector $\boldsymbol{u}$ for each user $u \in U$ and a vector $\boldsymbol{i}$ for each item $i \in I$, where the dot product of each user u and item i is close to the corresponding rating $r_{u,i}$ of i by u. Formally, the problem can be defined as follows:

Definition 1 (Matrix Factorization). *Given a set of users U with m users, a set of items I with n items, a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users, the matrix factorization problem aims to create two low-rank dimensional matrices $\boldsymbol{U}$ of dimension $k \times m$ and $\boldsymbol{V}$ of dimension $k \times n$ for users and items respectively by minimizing the following objective function:*

$$\arg\min_{\boldsymbol{U},V} \sum_{u \in U, i \in I} \phi_{u,i} \left(r_{u,i} - \boldsymbol{u}^\top \boldsymbol{i} \right),$$

where $\phi_{u,i} = 1$, if u has rated i; otherwise 0, We define a novel neural rating vectorization problem. It treats the possible ratings on each item i as an intrinsic property of the item, which indicates the quality of i and thus are independent from users. The neural rating vectorization problem aims to generate rating vectors on items by maximizing the likelihood estimation of the conditional probability of each score on item given another one. Formally, the neural item embedding problem can be defined as:

Definition 2 (Neural Item Embedding). *Let U, I and R be a set of users U with m users, a set of items I with n items, and a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users, respectively. The neural item embedding problem aims to create low rank vector representations of*

dimension $k \times m$ for items I by minimizing the following objective function:

$$\arg\min_{\boldsymbol{I}} - \sum_{u \in U} \sum_{i,j \in I, i \neq j} \phi_{u,i} \phi_{u,j} \log Prob(r_i = r_{u,i} \mid r_j = r_{u,j}), \tag{1}$$

where $Prob(r_i = r_{u,i} \mid r_j = r_{u,j})$ is the probability that user u provides a score of $r_{u,i}$ to item i given that the same user u assigns a score of $r_{u,j}$ to another item j. Once we obtain the item vectors by solving the above problem, user vectors can be generated directly by minimizing the difference between items' observed ratings and the product of user and item vectors: $\arg\min_{\boldsymbol{U}} \sum_{u \in U, i \in I} \phi_{u,i} \left(r_{u,i} - \boldsymbol{u}^\top \boldsymbol{i}\right)$

Note that the probability $Prob(r_i = r_{u,i} \mid r_j = r_{u,j})$ in Equation (1) actually involves two aspects of information: (1) the co-occurrence of ratings on each pair of items by same users, and (2) the rating scores or relative preferences of users holds on items. Thus it is extremely hard to be formulated by straightforwardly adapting that in *word2vec* [22, 23] with hierarchical softmax of the vectors.

4 The *Pref2Vec* Algorithm

Pref2Vec solves the neural item embedding problem in Definition 2 in three steps. Firstly, we generate vectors of pairwise item preference. We use these preference vectors in the second step to generate user vectors, that are in turn used to create item vectors in the third step.

4.1 Pairwise Preference Vectorization

To create vectors of pairwise item preferences, we create the pairwise preference matrix and use it to create the sets of positive pairwise preferences for each user. Then, we utilize neural language models to learn representations of positive preferences in lower dimensional space using available positive preference pairs.

Consider a set of users $U = \{u_1, u_2, \ldots, u_m\}$, a set of items $I = \{I_1, I_2, \ldots, I_n\}$ and their corresponding rating matrix R of dimension $m \times n$. Each row of R contains ratings $R_u = \{r_1, r_2, \ldots, r_n\}$ given by a user u for the n items, where most of the elements in R_u are unknown as users generally can provide ratings only for a very small number of items. This allows us to build a set of pairwise preference for each user by using a preference function: $p(i, j) \in \{+1, -1\}$, where $i = 1 \ldots n$, $j = 1 \ldots n$, $i \neq j$ and both r_i and r_j are known. The preference function $p(i, j)$ has a value of $+1$ if $r_i > r_j$ and -1 otherwise.

Now, we create the sets of positive preferences P_u for each user u. Without losing generality, here we only consider the conditions of positive preference pairs, as all of the negative preferences can be straightforwardly transformed into positive ones by reversing the positions of the two items. With n items, we should consider a total of $N = n(n-1)$ unique preference pairs, denoted as $P = \{p_1, p_2, \ldots, p_N\}$. Each users' preferences P_u is a subset of P, formally $P_u \subseteq P$ for any user u. Now, *Pref2Vec* proceeds with learning the vector representations of the preferences on the collection of preference sets $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$ for all of the users.

We consider the *word2vec* framework [22, 23] that generates vector representations of words. They presented the continuous *skip-gram* model, which assumed that for each target word the sequence of its surrounding words are trivial and can be ignored. This is achieved by maximizing the cumulative logarithm of the conditional probability for the surrounding words given each target word in the corpus with neural networks. Our approach is very similar, since we consider our collection of preference sets: $\mathbb{P}$ as the corpus, the preference sets $P_1, P_2, \ldots, P_m$ by the users as the sentences and the preferences $p_1, p_2, \ldots, p_N$ as the words.

However, the key difference in our approach is that we completely ignore the spatial information within the preference sets. This is because unlike words in sentences, the order of the preferences for a user (in a non-temporal setup) is inconsequential. This is the reason why we have a set representation of preferences for a user, as opposed to a sequence representation. Actually this property makes our scenario even better fit the *skip-gram* model than natural language processing, where the preferences have no sequence information and thus the sequence of the "surrounding preferences" can be ignored without any accuracy loss. Therefore, in the *Pref2Vec* framework, we learn the vector representations of the products by minimizing the following objective function over the entire collection $\mathbb{P}$ of preference sets:

$$\underset{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_n}{\arg\min} - \sum_{P_k \in \mathbb{P}} \sum_{(p_i, p_j) \in P_k, i \neq j} \log Prob(p_j \mid p_i), \tag{2}$$

where $Prob(p_j \mid p_i)$ is the hierarchical softmax of the respective vectors of the preference p_j and p_i. In particular, $Prob(p_j \mid p_i) = \frac{\exp(\boldsymbol{i}_o^\top \boldsymbol{j}_t)}{\sum_{p_l \in P_k} \exp(\boldsymbol{i}_o^\top \boldsymbol{l}_t)}$,where $\boldsymbol{i}_o$ and $\boldsymbol{j}_t$ are the initial and target vector representations respectively of preferences p_i and p_j. $\boldsymbol{l}_t$ is the target vector representations of any preference p_l in P_k. From Equation (2), we see that *Pref2Vec* model ignores the sequence of preferences within a user preferences set. The context is set to the level of preference sets, where the preference vectors that fall in the same preference sets will have similar vector representations.

Remarks: Our approach is also inspired by *item2vec* [1], that uses a straightforward application of *word2vec* by considering a set of items (accessed by a user) as a sentence and the individual items as words. Similar to *Pref2Vec*, *item2vec* also ignores the sequential information of items in a set. *item2vec* has been efficiently used in scenarios where we have a simple sequence of items, e.g. products purchased, videos watched, etc. In such cases, for each user the items are in 0 or 1 state. However, if the user feedback is provided in higher granularities (e.g. user ratings), then simply considering the sequence of items rated by the user and treating them equally, is expected to severely limit the quality of vectors. On the other hand, *Pref2Vec* enables the utilization of rating information by incorporating pairwise item preferences in the vectorization process.

4.2 User Vector Generation

However, the preference vectors generated in the previous section cannot be utilized directly for recommendation tasks, that often require good quality user and

item vectors as an input. In this section we describe the second step of *Pref2Vec* and aim to find vectors corresponding to the m users, given the preference vectors for each pair of items and known ground truths for the preferences.

For a particular user let $\boldsymbol{p}_1, \boldsymbol{p}_2 \dots \boldsymbol{p}_r$ be the preference vectors, each of length k, for which the respective values of preference function are $p_1, p_2, \dots, p_r \in \{+1, -1\}$. The corresponding user vector can be achieved by minimizing the cumulative difference between users' each observed preference p_i and the product of the user and preference vectors $\boldsymbol{u}^\top \boldsymbol{p}_i$. Thus we can formulate this problem as linear classification, where $\boldsymbol{p}_1, \boldsymbol{p}_2 \dots \boldsymbol{p}_r$ are training instances, the values of the preference functions $p_1, p_2, \dots, p_r$ are ground truth. With consideration of a bias b, we aim to predict the coefficients of a linear classification model, which is the user vector $\boldsymbol{u}$. In this study, we use Logistic regression [16] to solve the problem. The loss function with L2 norm is : $\arg\min_{\boldsymbol{u},b} \sum_{i=1}^{r} \log(1 + \exp(-p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))) + \frac{\lambda}{2}||\boldsymbol{u}||^2$, where $\boldsymbol{u}$ is a vector of length k, b is a number and λ is the tuning parameter for L2 norm. We use the gradient descent method for optimization. Given a learning rate α, the update formulas are derived as follows:

$$\begin{aligned} \boldsymbol{u} &\leftarrow \boldsymbol{u} - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))} \boldsymbol{p}_i + \lambda \boldsymbol{u}\right) \\ b &\leftarrow b - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))}\right) \end{aligned} \tag{3}$$

The generated user vectors $\boldsymbol{u}$ corresponding to each of the m users, form a user matrix U of dimension $m \times k$.

4.3 Item Vectors Generation

The last step of *Pref2Vec* is to find item vectors given the rating matrix $R_{m \times n}$ and the user matrix $U_{m \times k}$ generated in the previous section. For this we optimize matrix $I_{n \times k}$, by minimizing the difference between items' observed ratings and the product of user and item vectors, i.e. $UI^\top \approx R$. The n rows of I would be the item vectors. We minimize the loss function: $\arg\min_I ||R - UI^\top||^2 + \frac{\lambda}{2}||I||^2$, where λ is the tuning parameter for L2 normalization. We use the gradient descent method for optimization. Given a learning rate η, the update formula is:

$$I \leftarrow I - \eta(-2(R - UI^T)^\top U + \lambda I) \tag{4}$$

5 Experiments

The following three research questions guide the remainder of the paper.
RQ1 Is the quality of item vectors generated using the *Pref2Vec* approach better than state-of-the-art vectorization algorithms? (See Section 5.1)
RQ2 Are the outputs of the proposed *Pref2Vec* algorithms independent from their initialization? (See Section 5.2)

RQ3 Can the vectorization results be utilized to improve the performance of recommender systems? (See Section 5.3)

Datasets. We use three MovieLens[4] data sets in our experiments: MovieLens-100K, MovieLens-1M and MovieLens-10M. MovieLens-100K dataset contains 100,000 ratings given by 943 users on 1682 movies. MovieLens-1M dataset is larger with 1,000,000 ratings given by 6040 users on 3952 movies. Movielens-10M is the largest dataset used, with 10 million ratings given by 69878 users on 10681 movies. In MovieLens-100K as well as MovieLens-1M the ratings are given as integers from 1 to 5. In MovieLens-10M, the ratings are given in the range 0.5 to 5 with an increment of 0.5. In these datatsets there are 18 movie genres, a movie can belong to one or more of them. For all the three datasets we randomly assign 10 ratings for each user for testing and the rest for training. We have used the vector length of 10 for all the vectorization methods.

5.1 Evaluation of Item Quality

Ground-truth. Since the datasets provide genre information for all of the items (movies), we use the genre similarity as the ground truth. In particular, the genres of each movie are provided (or can be transformed) in the form of binary values. A value of 1 signifies that the movie belongs to a particular genre and 0 signifies the contrary. A movie can belong to more than one genre. So, let us consider that genre vectors derived from the meta-data are: $(\boldsymbol{G}_i \dots \boldsymbol{G}_j)$, which correspond to our item vectors $\boldsymbol{I}_i \dots \boldsymbol{I}_j$. Since, the genre vectors are binary vectors, to find similarity between them we use: Jaccard similarity [6], an efficient and popular measure for binary similarity. Jaccard similarity between two binary vectors $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$ is simply calculated as: $jacSim(\boldsymbol{v}_a, \boldsymbol{v}_b) = \frac{F_{11}}{F_{01}+F_{10}+F_{11}}$,where F_{11} is the number of features for which both $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$ have value 1. F_{01} is the number of features for which $\boldsymbol{v}_a$ has value 0 and $\boldsymbol{v}_b$ has 1. And, F_{10} is the number of features where $\boldsymbol{v}_a$ had the value 1 and $\boldsymbol{v}_b$ has 0.

For the item vectors $\boldsymbol{I}_1, \boldsymbol{I}_2, \dots, \boldsymbol{I}_n$ (calculated in Section 4.3), the similarity can be calculated for each pair of item vectors $(\boldsymbol{I}_i, \boldsymbol{I}_j)$ as: $cosSim(\boldsymbol{I}_i, \boldsymbol{I}_j) = \frac{\boldsymbol{I}_i^\top \boldsymbol{I}_j}{|\boldsymbol{I}_i| \times |\boldsymbol{I}_j|}$, where $|\boldsymbol{I}_i|$ and $|\boldsymbol{I}_i|$ are the length of the vectors $\boldsymbol{I}_i$ and $\boldsymbol{I}_j$.

Evaluation Metrics. In order to evaluate the quality of item vectors we use the RMSE (root mean squared error) and MAE (mean absolute error) measures. To calculate these, we calculate the similarities between each pair of item vectors and the similarities between their corresponding pairs of ground truths. Since the item in our experiments are movies, the genre information about the movies (available from metadata) is considered as ground truth. The differences between the two similarities for each item are considered as errors, that are in turn used to calculate RMSE and MAE. We use these measures because for good quality item vectors, the vectors that are similar should also have similarity based on their relevant meta data information. Therefore, the lower the values of RMSE and MAE, the better is the quality of vectors.

[4] http://grouplens.org/datasets/movielens/

Table 1. Quality of Generated Item Vectors against Baselines

Algorithm	ML-100K		ML-1M		ML-10M		Ref.
	RMSE	MAE	RMSE	MAE	RMSE	MAE	
RM-Vectors	0.8674	0.8163	0.8790	0.8292	0.8563	0.8151	-
IS-Vectors	0.8238	0.7508	0.7018	0.5886	0.5864	0.4758	[1]
MF-Vectors	0.6844	0.6431	0.6904	0.6478	0.6478	0.6071	[18]
P2V-Vectors	**0.4770**	**0.3846**	**0.5165**	**0.4305**	**0.4456**	**0.3695**	This paper

To calculate the errors we need: the difference between the similarities of two item vectors and the similarities between the corresponding two genre vectors. The errors are calculated for all pairs of items: $e_{i,j} = cosSim(\boldsymbol{I}_i, \boldsymbol{I}_j) - jacSim(\boldsymbol{G}_i, \boldsymbol{G}_j)$. Though cosine similarity and Jaccard similarity are different measurements, their difference used here is expected to be highly indicative of the error. There would be $n(n-1)/2$ such errors. Now, $RMSE = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} e_{i,j}^2}{n(n-1)/2}}$ and $MAE = \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} |e_{i,j}|}{n(n-1)/2}$, where the function $|\cdot|$ gives the absolute value of the parameter.

Baselines. We choose the following methods to evaluate the quality of the item vectors that are generated by the *Pref2Vec* framework, i.e. P2V-Vectors.

• **RM-Vectors:** Rating matrix $R_{m\times n}$ contains ratings by m users for n items, and its columns are the simplest (and readily available) form of item vectors.

• **IS-Vectors:** Neural embeddings of items are created by considering the set of items rated by users as sentences and items as words (similar to *item2vec* [1] approach). Comparison with this method would validate the importance of using preference information for vectorization in *Pref2Vec*.

• **MF-Vectors:** In matrix factorization [18] user and item vectors are created by randomly initializing matrices $U_{m\times k}$ and $I_{n\times k}$ and then minimizing the difference between their product and the rating matrix (i.e. $R - UI^\top$).

Results. In Table 1, we compare the item vector qualities using RMSE and MAE. For MovieLens-100K dataset, for both RMSE and MAE, P2V-Vectors perform the best, followed by MF-Vectors. IS-Vectors are the third and the RM-Vectors are the worst performing ones. The trend is same for the dataset MovieLens-1M for RMSE. For MovieLens-1M in terms of MAE as well as for MovieLens-10M (both RMSE and MAE), though P2V-Vectors are still the best performing ones, the second best are IS-Vectors, followed by MF-Vectors and then RM-Vectors. The improvement shown by P2V-Vectors is significant.

Pref2Vec firstly generates preference vectors, and then creates user vectors with the generated preference vectors, and finally produces item vectors with the generated user vectors. Since each step is an approximation process with certain accuracy loss, the preference and user vectors should be more accurate than the item vectors. Thus although we cannot assess the quality of user and preference vectors resulting from lack of corresponding ground-truth information, we can still claim that the quality of the preference, user and item vectors generated by our *Pref2Vec* method can significantly outperform our baselines.

5.2 Evaluation of Initialization Independence of Generated Vectors

Firstly, we describe the measurements to evaluate the independence of generated vectors from their initialization. Let us consider that x different runs (resulting from different initializations) of a vector generation method generate: user matrices $U^{(1)} \dots U^{(x)}$ and the corresponding item matrices $I^{(1)} \dots I^{(x)}$. Each user matrix is of dimension $m \times k$ with the rows corresponding to m user vectors, each of length k. Similarly each item matrix is of dimension $n \times k$ with the rows corresponding to n item vectors, each of length k. Since the features of the vectorization results might be in a different order by different runs of algorithms, we sort the generated features according to their cumulative values among all of the users. The independence of these vectors from the initialization can be measured using (a) variance of the elements of the U and I matrices and (b) correlations between the user and item vectors generated in different runs. These measures are explained in detail as follows.

Variance Calculation. Let $U^{(1)}_{i,j}, U^{(2)}_{i,j} \dots U^{(x)}_{i,j}$ be the x values in the user matrices at ith row and jth column from x different runs of a vectorization algorithm. Their variances can be calculated as: $var_U(i,j) = \frac{1}{x} \sum_{l=1}^{x} \left(U^{(l)}_{i,j} - \overline{U}_{i,j} \right)^2$, where $\overline{U}_{i,j}$ is the average of $U^{(1)}_{i,j}, U^{(2)}_{i,j} \dots U^{(x)}_{i,j}$. With $m \times k$ dimensions of the user matrix U, we can get $m \times k$ variance, and the mean variance would be: $MVU = \frac{1}{m \times k} \sum_{i=1}^{m} \sum_{j=1}^{k} var_U(i,j)$.

Similarly, the variance of the item matrices at the ith row and jth column $I^{(1)}_{i,j}, I^{(2)}_{i,j} \dots I^{(x)}_{i,j}$ from x different runs of a vectorization algorithm can be calculated as: $var_I(i,j) = \frac{1}{x} \sum_{l=1}^{x} \left(I^{(l)}_{i,j} - \overline{I}_{i,j} \right)^2$, where $\overline{I}_{i,j}$ is the average of $I^{(1)}_{i,j}, I^{(2)}_{i,j}, \dots, I^{(x)}_{i,j}$. The mean variance of the generated item vectors can be calculated as: $MVI = \frac{1}{n \times k} \sum_{i=1}^{n} \sum_{j=1}^{k} var_I(i,j)$

A lower value of the mean variance is indicative that the generated values that comprise the user or item vectors do not vary much with different initializations.

Correlation of Vectors. The independence of the vectors from the initialization of the generation technique can also be estimated by the correlation between the vectors generated in different runs. We use Pearson correlation coefficient [15] to calculate correlation $\rho(x, y)$ between variables x and y.

A user matrix $U^{(j)}$ generated in the j^{th} run, contains m user vectors : $\boldsymbol{u}^{(j)}_1 \dots \boldsymbol{u}^{(j)}_m$. For a particular user, the average of pairwise correlations between the vectors generated in the x runs would be: $AC(i) = \frac{\sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{u}^{(j)}_i, \boldsymbol{u}^{(l)}_i)}{x(x-1)/2}$ And, the mean of these average correlation for all the m user vectors can simply be calculated as: $MAC = \frac{\sum_{i=1}^{m} \sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{u}^{(j)}_i, \boldsymbol{u}^{(l)}_i)}{m \times x(x-1)/2}$

Similarly, the mean average correlation for the item vectors, $\boldsymbol{i}^{(j)}_1 \dots \boldsymbol{i}^{(j)}_n$ generated in x runs ($j = 1 \dots x$), can be calculated as: $MAC = \frac{\sum_{i=1}^{n} \sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{i}^{(j)}_i, \boldsymbol{i}^{(l)}_i)}{n \times x(x-1)/2}$ A high value if MAC mean that the vectors generated during different runs are close to each other and hence have high level of independence to initialization.

Table 2. Initialization Independence of Generated Vectors

Algorithm	User Vectors		Item Vectors	
	MVU	MAC	MVI	MAC
MF	0.0730	0.0015	0.0773	0.0315
P2V	**0.0015**	**0.8592**	**0**	**0.7881**

Results. In Table 2, we show the results evaluating the initialization independence of user and item vectors generated using *Pref2Vec* (shown as P2V) and comparing them with the vectors generated by matrix factorization (shown as MF). On the dataset MovieLens-100K we run both the methods 5 times, resulting in creation of 5 different pairs of user and item vectors for both of them. We calculate MVU, MVI and MAC (for user and item vectors) for the vectorization results generated by P2V and matrix factorization (MF).

The values of MVU and MVI of our algorithm are merely 0.0015 and 0 for user and item vectors, which are sharply lower than that of the matrix factorization method. Note that although the values of matrix factorization are smaller than 0.1, they are still large because the values in the user and item matrices are very small, and most of them are less than 1. Also, the values of MAC are very high for P2V for both item and user vectors, especially in comparison with the respective values for MF. This again shows that the user and item vectors generated by P2V in different runs are highly correlated to each other.

5.3 Ranking Prediction based on Generated Vectors

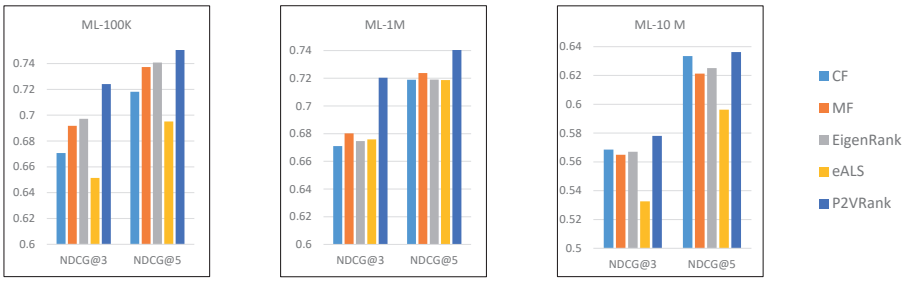
Ranking Model using User Vectors. Here we describe the method to generate rankings for items with unknown ratings for user using the available *Pref2Vec* preference and user vectors. This is done by firstly predicting the preference values $\hat{p} \in \{+1, -1\}$ for the preference vectors corresponding to the items with unknown ratings. Then we employ a greedy order algorithm to derive approximately optimal ranking of the unrated items.

In Section 4.2 we showed the process that generates the user vector $\boldsymbol{u}$ and the value b after optimization. Since the optimization process directly employs the Logistic regression loss function, this allows us to also directly use Logistic regression classification to predict pairwise preferences for a user. More specifically, for a user with user vector $\boldsymbol{u}$ and accompanying value b, the user's preference $\hat{p}_u$ can be predicted as: $\hat{p}_u = +1, \text{if } \boldsymbol{u}^\top \boldsymbol{p} + b > 0; -1$ otherwise.

Hence, for a particular user, if there are q items with unknown rankings $I_1, I_2 \ldots I_q$, the values for the preference function $\hat{p}(I_i, I_j) \in \{+1, -1\}$, can be predicted. Since the values for pairwise preference function are not a direct format to get the rankings, we use the greedy order algorithm proposed by Cohen et al. [7, 21], that efficiently finds an approximately optimal ranking for the target user u. It is showed that based on reduction of cyclic ordering problem [10], the determination of optimal ranking is a NP-complete problem and the algorithm can be proved to have an approximation ratio of 2 [10].

Remarks. Alternatively, we could have directly used the user matrix U (Section 4.2) and the item matrix I (Section 4.3) to generate the ratings matrix ($R =$

Fig. 1. Ranking Performance of *Pref2Vec* against baselines



UI^T), that could be used to generate ranking of unrated items. However, since we follow sequential steps by first generating U from preference vectors, then using U to create I and thereafter using U and I to create R; there is accuracy loss at each step. On the other hand, our ranking model avoids such additional inaccuracies by directly using preference vectors and U to generate rankings.

Baselines. We use the following baselines to access the performance of our simple recommendation method P2VRank:
• **CF:** CF [5] is a memory-based collaborative filtering algorithm that uses the Pearson correlation coefficient to calculate the similarity between users.
• **MF:** Given a raking matrix R, in matrix factorization [18] the user matrix U and the item matrix I are optimized in order to minimize the difference: $R - UI^\top$.
• **EigenRank:** EigenRank [21] uses greedy aggregation method to aggregate the predicted pairwise preferences of items into total ranking.
• **eALS:** Element-wise Alternating Least Squares (eALS) [14] efficiently optimizes a MF model with variably-weighted missing data. As eALS is an implicit feedback algorithm, we consider only higher ratings (≥ 4) as positive feedback.

Results. The performance is evaluated using the standard ranking accuracy metric NDCG [17] @3 and @5. In Fig 1, we see that P2VRank outperforms all comparison partners. Also, we also observed strong statistical significance ($\alpha = 0.05$) on comparing P2VRank against MF for all the three datasets.

6 Conclusion

We proposed *Pref2Vec* to generate vector representations of pairwise item preferences. We also presented the method to generate user and item vectors using preference vectors. Also, our experimental results demonstrated that the quality of item vectors generated by *Pref2Vec* is better than that of the standard techniques. We also verified that the generated user and item vectors are highly independent of the initializations. In addition, we presented the technique to generate rankings of items, using the generated user vectors, and showed that it outperforms the standard recommendation techniques. Currently we only consider the preference of one item over another for the creation of *Pref2Vec* and in future we would like to consider the magnitudes of these preferences.

References

1. Barkan, O., Koenigstein, N.: Item2vec: Neural item embedding for collaborative filtering. In: MLSP. pp. 1–6 (2016)
2. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
3. Bhingardive, S., Singh, D., V, R., Redkar, H.H., Bhattacharyya, P.: Unsupervised most frequent sense detection using word embeddings. In: HLT-NAACL. pp. 1238–1243 (2015)
4. Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS. pp. 2787–2795 (2013)
5. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI. pp. 43–52 (1998)
6. Choi, S., Cha, S., Tapper, C.C.: A survey of binary similarity and distance measures. J. Systemics, Cybernetics and Informatics **8**(1), 43–48 (2010)
7. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. J. Art. Int. Res. **10**(1), 243–270 (1999)
8. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
9. Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M., Bhamidipati, N.: Hierarchical neural language models for joint representation of streaming documents and their content. In: WWW. pp. 248–255 (2015)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1990)
11. Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., Sharp, D.: E-commerce in your inbox: Product recommendations at scale. In: SIGKDD. pp. 1809–1818 (2015)
12. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: SIGKDD. pp. 855–864 (2016)
13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: WWW. pp. 173–182 (2017)
14. He, X., Zhang, H., Kan, M.Y., Chua, T.S.: Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In: SIGIR. pp. 549–558 (2016)
15. Herlocker, J., Konstan, J.A., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. ACM Trans. Inf. Syst. **5**, 287–310 (2002)
16. Hosmer Jr, D.W., Lemeshow, S., Sturdivant, R.X.: Applied Logistic Regression, vol. 398. John Wiley & Sons (2013)
17. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)
18. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
19. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. In: HLT-NAACL. pp. 260–270 (2016)
20. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: ICML. pp. 1188–1196 (2014)
21. Liu, N.N., Yang, Q.: Eigenrank: A ranking-oriented approach to collaborative filtering. In: SIGIR. pp. 83–90 (2008)

22. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS. pp. 3111–3119 (2013)
24. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: SIGKDD. pp. 701–710 (2014)
25. Schwenk, H.: Continuous space language models. Computer Speech Lang. **21**(3), 492 – 518 (2007)
26. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: NIPS. pp. 926–934 (2013)
27. Socher, R., Lin, C.C., Ng, A.Y., Manning, C.D.: Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In: ICML (2011)
28. Turian, J., Ratinov, L., Bengio, Y.: Word representations: A simple and general method for semi-supervised learning. In: ACL. pp. 384–394 (2010)
29. Turney, P.D.: Distributional semantics beyond words: Supervised learning of analogy and paraphrase. TACL **1**, 353–366 (2013)
30. Zou, W.Y., Socher, R., Cer, D.M., Manning, C.D.: Bilingual word embeddings for phrase-based machine translation. In: EMNLP. pp. 1393–1398 (2013)