applied sciences

MDPI

Article

Leveraging User Comments for Recommendation in E-Commerce

Pang-Ming Chu [1], Yu-Shun Mao [1], Shie-Jue Lee [2,*] and Chun-Liang Hou [3]

[1] Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung 804, Taiwan; pmchu@water.ee.nsysu.edu.tw (P.-M.C.); ysmao@itlm.ee.nsysu.edu.tw (Y.-S.M.)
[2] Department of Electrical Engineering, Intelligent Electronic Commerce Research Center, National Sun Yat-Sen University, Kaohsiung 804, Taiwan
[3] Information Technology Department, Southern Taiwan Business Group, Chunghwa Telecom Co., Ltd., Kaohsiung 800, Taiwan; hjl@cht.com.tw
* Correspondence: leesj@mail.ee.nsysu.edu.tw; Tel.: +886-7-5254141

Received: 8 March 2020; Accepted: 2 April 2020; Published: 7 April 2020

Abstract: Collaborative filtering recommender systems traditionally recommend products to users solely based on the given user-item rating matrix. Two main issues, data sparsity and scalability, have long been concerns. In our previous work, an approach was proposed to address the scalability issue by clustering the products using the content of the user-item rating matrix. However, it still suffers from these concerns. In this paper, we improve the approach by employing user comments to address the issues of data sparsity and scalability. Word2Vec is applied to produce item vectors, one item vector for each product, from the comments made by users on their previously bought goods. Through the user-item rating matrix, the user vectors of all the customers are produced. By clustering, products and users are partitioned into item groups and user groups, respectively. Based on these groups, recommendations to a user can be made. Experimental results show that both the inaccuracy caused by a sparse user-item rating matrix and the inefficiency due to an enormous amount of data can be much alleviated.

Keywords: collaborative filtering; user-item rating matrix; data sparsity; scalability; Word2Vec; self-constructing clustering

1. Introduction

Recommender systems [1–3] are able to analyze the past behavior of customers and recommend the products in which they might be interested. Recommender systems can roughly be categorized into two types: collaborative filtering and content-based filtering. Content-based filtering [4–6] assumes that customers will buy things that are similar to what they have bought in the past. Therefore, detailed information about products and users are required for recommendations. However, the information needed is growing harder to get in the modern age of privacy awareness. Collaborative filtering [7–17] assumes that similar users have similar interest in items and similar items have similar ratings by users. Traditionally, users give their preference ratings for the previously purchased products, and these ratings are maintained in a user-item rating matrix. Based on the content of the matrix, collaborative filtering makes recommendations to a user according to the opinions of other like-minded users on the products. In general, collaborative filtering is simpler and more practical, and it tends to be more appealing in the E-commerce community.

However, there exist some issues with collaborative filtering. Two of them are data sparsity and scalability [18–20]. Data sparsity is related to the sparse ratings in the user-item rating matrix, and it can lead to inaccurate recommendations. Scalability is related to the huge number of products or/and

users involved, which may cause an unacceptably long delay before valuable recommendations are acquired. In this paper, we propose a novel approach to deal with data sparsity and scalability. First of all, we apply Word2Vec [21,22], which is a word semantic tool developed by Google, to analyze the user comments on their previously bought goods. Each word is assigned a unique vector that represents its semantics. A set of item vectors, one item vector for each product, is then developed. Through the user-item rating matrix, the user vectors of all the users are obtained. Principal component analysis and an iterative self-constructing clustering algorithm are applied to reduce the time complexity related to the large numbers of items and users. Then, recommendation work is done with the resulting clusters. Finally, reverse transformation is performed, and a ranked list of recommended items can be offered to each user. With the proposed approach, the inaccuracy caused by the sparse ratings in the user-item rating matrix is overcome, and the processing time for making recommendations from an enormous amount of data is much reduced.

The proposed approach is an extension of our previous work published in [23]. Some major advantages over the previous work are provided. Word2Vec is applied on user comments to find the similarity relationship among products. Therefore, the sparsity problem is alleviated, and the accuracy of recommendation is improved. An iterative self-constructing clustering algorithm is used. The clusters obtained are less dependent on the order of the training patterns presented to the clustering algorithm, leading to more robust recommended results. Finally, clustering is applied not only on the products but also on the users. Consequently, the scalability problem can be further alleviated.

The rest of this paper is organized as follows. In Section 2, collaborative filtering, particularly our previous work [23], is briefly introduced. Some other related work is reviewed and discussed in Section 3. The proposed approach is described in detail in Section 4. Experimental results are presented in Section 5. Finally, a conclusion is given in Section 6.

2. Collaborative Filtering

Suppose there is a set of N users, u_i, $1 \le i \le N$, and a set of M products, p_j, $1 \le j \le M$. A user u_i may express his/her evaluation to a product p_j by providing a rating r_{ij}, which is a positive integer, for p_j. Usually, a higher rating is assumed to indicate a more favorable feedback from the user. If user u_i has not provided a rating for product p_j, $r_{ij} = 0$. Such information can be represented by the following user-item rating matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1M} \\ r_{21} & r_{22} & \cdots & r_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{N1} & r_{N2} & \cdots & r_{NM} \end{bmatrix} \quad (1)$$

which is an $N \times M$ matrix. Traditionally, collaborative filtering recommends products to each user solely based on such a user-item rating matrix. The idea is simple, but issues of data sparsity and scalability may arise.

In [23], we proposed an approach to address the problem of scalability. Each product p_i, $1 \le i \le M$, is represented as a N-vector, which is the ith column in $\mathbf{R}$. A self-constructing clustering algorithm is applied to divide the products into a collection of g clusters $C_1, C_2, \cdots, C_g$, with $g \le M$. Each cluster is regarded as one item group. Therefore, g item groups, denoted $I_1, I_2, \cdots, I_g$, respectively, are obtained. Then a $M \times g$ transformation matrix $\mathbf{T}$ is formed, with the (i, j)th entry t_{ij} denoting the membership degree of product p_i to item group I_j, $1 \le i \le M$, $1 \le j \le g$.

Next, the original user-item rating matrix $\mathbf{R}$ is transformed to a reduced matrix $\mathbf{B}$ by

$$\boldsymbol{B} = \mathbf{RT} \quad (2)$$

which is an $N \times g$ matrix. Based on $\mathbf{B}$, a $g \times g$ correlation matrix is constructed. Then, ItemRank is applied, and a predicted preference list of item groups, $\mathbf{h}_i$, is iteratively derived for each user u_i. Since

recommendation is done with **B**, which is of size $N \times g$, rather than with **R**, which is of size $N \times M$, efficiency is improved. Finally, the reverse transformation is performed:

$$\mathbf{s}_i = \mathbf{T}\mathbf{h}_i \tag{3}$$

which is the predicted preference list of products for user u_i, $1 \leq i \leq N$.

However, collaborative filtering recommender systems may encounter difficulty due to sparsity. Consider Table 1, which is a small user-item rating matrix, consisting of 7 users and 7 items. Clearly, there are many missing ratings in **R**.

Table 1. A small user-item rating matrix **R**.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
u_1	5	0	0	0	1	0	0
u_2	0	5	0	0	0	1	0
u_3	0	0	5	0	0	1	0
u_4	0	0	0	5	0	0	1
u_5	1	0	0	0	5	0	0
u_6	0	1	0	0	0	5	0
u_7	0	0	1	0	0	0	5

Note that, because of sparsity, the rows of **R** are hardly similar to each other. Recommendations based on the similarity between the users will not be accurate at all. In addition, because of sparsity, the columns of **R** are hardly similar to each other. It is impossible to reduce the number of items by clustering techniques based on the similarity between the items. It is also impossible to reduce the number of users based on the similarity between the users. Therefore, data sparsity makes it hard in this case to get recommendations right and efficiently by collaborative filtering through similarity between the users or/and items.

3. Related Work

Various methods have been proposed to address to address the issues of data sparsity and scalability associated with collaborative filtering. In matrix factorization methods [24,25], an $\boldsymbol{M} \times \boldsymbol{N}$ user-item rating matrix is decomposed into two matrices of size $\boldsymbol{M} \times \boldsymbol{K}$ and $\boldsymbol{K} \times \boldsymbol{N}$, respectively. In this way, $\boldsymbol{K}$ groups are obtained. Bobadilla et al. [18] use a Bayesian non-negative matrix factorization method and apply K-means to get better parameters for matrix factorization. These methods have a drawback in common. When encountering a heavily sparse user-item rating matrix, the groups obtained are not reliable. As a result, the resulting recommendations are not accurate and may not be useful. Zhang et al. [26] introduce the concepts of popular items and frequent raters. They assume that the ratings match some probability model. To overcome the data sparsity and rating diversity, smoothing and fusion techniques are employed. However, the method may encounter difficulties with a user-item rating matrix having a high degree of sparsity.

Clustering-based methods [2,23,27–33] have been proposed to address the scalability issue. Users or/and items are clustered into groups, and thus the numbers of users or/and items are reduced. Park [34] proposes a method that identifies tail items, having only a few ratings, from head items, having an enough number of ratings, according to their popularities. The recommendations for tail items are based on the ratings of clustered groups, while the recommendations for head items are based on the ratings of individual items or groups clustered to a lesser extent. However, this method tends to recommend tail items as a whole to users. Das et al. [30] use the DBSCAN clustering algorithm for clustering the users, and then implement voting algorithms to recommend items to the user depending on the cluster into which it belongs. The idea is to partition the users and apply the recommendation algorithm separately to each partition. Allahbakhsh and Ignjatovic [35] propose an iterative method that regards all the users as one user. The ratings by individual users are integrated together according

to each user's credibility, which is iteratively updated. This can reduce the sparsity difficulty. However, it cannot give personalized recommendations to each user. In our previous work [23], we apply a self-clustering algorithm to cluster products into item groups. However, the users are not clustered. This may encounter the scalability problem when a huge number of users are involved.

Incorporating information from sources other than the user-item rating matrix may help to overcome the sparsity problem [6,20,36–38]. Victor et al. [39] consider the trust and distrust relationship between users when doing rating prediction by collaborative filtering. The information about the trust and distrust relationship is gathered from the "useful" or "not useful" tag in the user text comments on items. If a user gives a useful tag in another user's comments, a positive relationship will be considered to exist between them. Forsati et al. [40] also consider the trust and distrust relationship between users, taken from social networks, to mitigate the sparsity issue. They improve the matrix factorization method by adding a condition to the optimization function in which the trust and distrust information is embedded. Huang et al. [41] apply the user-item rating matrix, user social networks, and item features extracted from the DBpedia knowledge base to cluster users and items into multiple groups simultaneously. By merging the predictions from each cluster, the top-n recommendations to the target users are returned. Zheng et al. [42] propose a matrix factorization method that takes advantages of user comments to items. The comments are converted to item vectors with elements corresponding to the predefined keywords. If a keyword appears in the comments about an item, the corresponding element of the item vector is set to 1; otherwise, it is set to 0. One difficulty of this method is that the keywords have to be manually defined. Barkan et al. [43] use Word2Vec to analyze the metadata of items and construct item vectors. Then, items are clustered, and the top-k similar items can be found. However, for most of these methods, accessing user relationships from social networks is getting harder when the awareness of privacy is soaring nowadays. Besides, the information obtained indirectly through friends in social networks may not always be relevant. For instance, one may have a friend who has a really different lifestyle, contradicting the basic assumption for the trust relationship.

4. Proposed Approach

Suppose, from some other sources, we know that p_1, p_2, p_3, and p_4 in Table 1 are similar to each other, and so are p_5, p_6, and p_7. Then, we can somehow cluster these seven products into two item groups, say, p_1^g and p_2^g, and transform the matrix to something similar to the one shown in Table 2. Assume that the ratings in this table are obtained by taking the maximum of the original product ratings. Clearly, there exist two groups of similar users, namely, $\{u_1, u_2, u_3, u_4\}$ and $\{u_5, u_6, u_7\}$, and recommendations through the similarity between users can be done by collaborative filtering.

Table 2. The user-item rating matrix after a clustering of products.

	p_1^g	p_2^g
u_1	5	1
u_2	5	1
u_3	5	1
u_4	5	1
u_5	1	5
u_6	1	5
u_7	1	5

Motivated by this idea, we propose a novel approach to overcome the inaccuracy and inefficiency caused by data sparsity and scalability. Our approach consists of five steps: developing item and user vectors, grouping for items and users, getting a reduced user-item rating matrix, calculating group rating scores, and deriving individual preference lists. Step 1 aims to overcome the sparsity problem by exploiting extra information from user comments. Item and user vectors are derived. Step 2 aims to deal with the scalability problem. In this step, products and users are clustered into item groups

and user groups, respectively. As a result, the numbers of products and users are greatly reduced. In step 3, the original user-item rating matrix is converted to a reduced user-item rating matrix that involves item groups and user groups. In step 4, a series of random walks are executed on the reduced user-item rating matrix, and a preference list of item groups is derived for each user group. Since low numbers of item groups and user groups are involved, this step can be done efficiently. Finally, in step 5, reverse transformations are performed, and preference lists of individual products are offered to each user. The pseudo-code of the approach is given below. Details will be described later.

The Proposed Approach

/⋆ Step 1: Developing item and user vectors ⋆/
Extract training patterns from the collected user comments;
Train the Word2Vec network and obtain item vectors and user vectors;
/⋆ Step 2: Grouping for items and users ⋆/
Reduce the dimensionality of item and user vectors;
Cluster the item and user vectors into item and user groups;
Obtain transformation matrices of membership degrees;
/⋆ Step 3: Getting the reduced user-item rating matrix ⋆/
Transform the original user-item rating matrix to the reduced matrix;
/⋆ Step 4: Calculating group rating scores ⋆/
Construct the correlation matrix from the reduced user-item rating matrix;
Apply ItemRank to predict preference lists of item groups for user groups;
/⋆ Step 5: Deriving individual preference lists ⋆/
Reversely transform to obtain preference lists of products for users;

4.1. Step 1—Developing Item and User Vectors

As mentioned, it is usually difficult to discover similarity between items or users solely based on the sparse ratings provided in the user-item rating matrix. Extra sources have to be consulted for help. In this step, we apply Word2Vec on user comments to discover semantic relationships between products. Each product is converted to an item vector. Then, the item vectors can be used for clustering products later. In addition, the users can be converted to user vectors, by which the clustering on users can be done.

We address the data sparsity problem by finding the relationship between items from the user comments posted on the public domain. We use these user comments to train a Word2Vec network. The word vectors obtained after training should reveal semantic relationships among items, so it is required that the inputs to the Word2Vec network be items in the training patterns.

We take the user review text of an item and insert the item identity into the text repeatedly. Suppose that the window size is $2s + 1$. The item identity is inserted in the middle of every $2s$ words in the original text. Training patterns, each consisting of an input–output word pair (x, y), are extracted from the windows. For each window, the central word is taken as input x, and each of its context words is taken as output y to form training patterns. Assume that a window contains $2s + 1$ words, $w\prime_{r+1} \cdots w\prime_{r+s} w\prime_{r+s+1} w\prime_{r+s+2} \cdots w\prime_{r+2s+1}$, where $w\prime_{r+s+1}$ is the central word and the other words are context words. Then, there are $2s$ training patterns, $(w\prime_{r+s+1}, w\prime_{r+1}), \ldots, (w\prime_{r+s+1}, w\prime_{r+s})$, $(w\prime_{r+s+1}, w\prime_{r+s+2}), \ldots, (w\prime_{r+s+1}, w\prime_{r+2s+1})$, extracted from the window. Let the review text of an item, with its item identity being 0000013714, be:

> *I bought this for my husband who plays the piano He is having a wonderful time playing these old hymns The music is at times hard to read because we ...*

For $s = 3$, the user review text after insertion is shown below, ignoring punctuation marks:

I bought this **0000013714** *for my husband who plays the* **0000013714** *piano He is having a wonderful* **0000013714** *time playing these old hymns The* **0000013714** *music is at times hard to* **0000013714** *read because we . . .*

Note that the item identity 0000013714 appears in the middle of each non-overlapping window of seven words. For instance, the first window consists of the following words:

I bought this **0000013714** *for my husband.*

Then, we extract training patterns from each window. Considering the first window above, six training patterns are extracted:

(0000013714, I), (0000013714, bought), (0000013714, this),

(0000013714, for), (0000013714, my), (0000013714, husband).

For the second window "who plays the 0000013714 piano He is", another six training patterns are extracted:

(0000013714, who), (0000013714, plays), (0000013714, the),

(0000013714, piano), (0000013714, He), (0000013714, is).

Note that the first words in the training patterns are the item identity.

After extracting the training patterns from the collected user comments, the training patterns are fed into the Word2Vec network [21,22], as shown in Figure 1, for training.

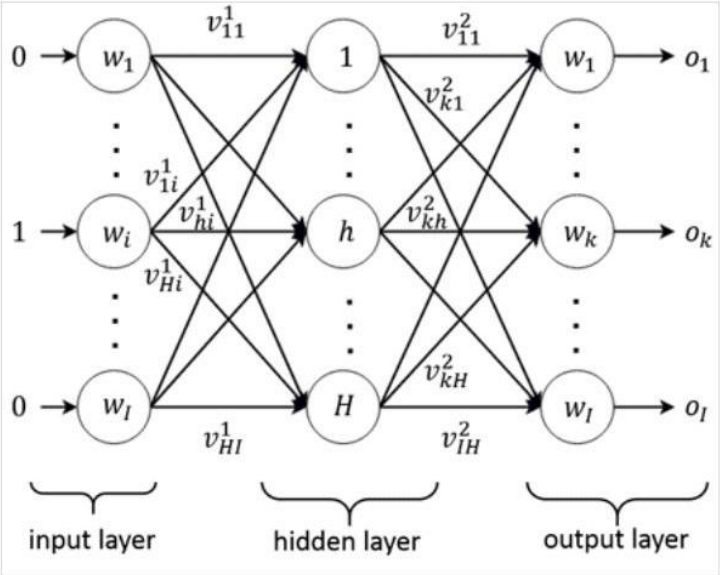

Figure 1. The Word2Vec network.

The neural network is trained as follows. Suppose (w_i, w_k) is a training pattern involving two words, w_i and w_k. Word w_i corresponds to node i in the input layer, and word w_k corresponds to node k in the output layer, with input weights $\mathbf{v}_i^1 = \left[v_{1i}^1 \cdots v_{Hi}^1\right]^T$ and output weights $\mathbf{v}_k^2 = \left[v_{k1}^2 \cdots v_{kH}^2\right]^T$. Note that the input value at node i is 1, while those at the other nodes in the input layer are 0. In addition, the desired value at node k is 1, while those at the other nodes in the output layer are 0. The network output at node k is

$$o_k = \frac{1}{1 + e^{-\mathbf{v}_i^1 \cdot \mathbf{v}_k^2}} \tag{4}$$

where '$\cdot$' is the inner product operator. Let J be $J = -\log_e o_k$. By steepest descent, $\mathbf{v}_i^1$ and $\mathbf{v}_k^2$ are adjusted as

$$\mathbf{v}_i^1 \leftarrow \mathbf{v}_i^1 - \eta \nabla_{\mathbf{v}_i^1} J = \mathbf{v}_i^1 - \eta(o_k - 1)\mathbf{v}_k^2, \tag{5}$$

$$\mathbf{v}_k^2 \leftarrow \mathbf{v}_k^2 - \eta \nabla_{\mathbf{v}_k^2} J = \mathbf{v}_k^2 - \eta(o_k - 1)\mathbf{v}_i^1 \tag{6}$$

where η is the learning rate. During the learning, if two training patterns (w_i, w_k) and $\left(w_j, w_k\right)$ occur frequently in the training documents, $\mathbf{v}_i^1$ and $\mathbf{v}_j^1$ will be close to each other. When the learning is completed, the input weights of the network are taken to be item vectors. That is, $\mathbf{v}_1^1$ is the word vector of word w_1, $\mathbf{v}_2^1$ is the word vector of word w_2, etc. Note that item vectors are word vectors of the item identities. For M products, we have M item vectors in total, $\mathbf{p}_1$, $\mathbf{p}_2, \cdots$, $\mathbf{p}_M$, as shown below:

$$\mathbf{p}_i = \begin{bmatrix} p_{1i} \\ p_{2i} \\ \vdots \\ p_{Hi} \end{bmatrix} = \begin{bmatrix} v_{1i}^1 \\ v_{2i}^1 \\ \vdots \\ v_{Hi}^1 \end{bmatrix} \tag{7}$$

for $i = 1, \cdots, M$. Clearly, each item vector contains H components. In this work, we set $H = 100$. Note that if two items are semantically similar, then their corresponding item vectors are close to each other.

Now that item vectors are available, we can derive user vectors, which can then be used for similarity computation. First, we normalize each row in the user-item rating matrix $\mathbf{R}$ as follows:

$$a_{ij} = \frac{r_{ij}}{\sum_{k=1}^{M} r_{ik}}, \ 1 \le j \le M \tag{8}$$

for $1 \le i \le N$. Then, user vectors, $\mathbf{u}_k$, $1 \le k \le N$, are derived as follows:

$$\mathbf{u}_k = \begin{bmatrix} u_{1k} \\ u_{2k} \\ \vdots \\ u_{Hk} \end{bmatrix} = \sum_{j=1}^{M} a_{kj} \mathbf{p}_j. \tag{9}$$

Note that each user vector also contains H components.

4.2. Step 2—Grouping for Items and Users

This step aims to deal with the scalability problem. First of all, the dimensionality, H, can be too large for effective processing. PCA (Principal Component Analysis) [44,45] is applied to reduce the dimensionality of the item and user vectors. We choose q_p and q_u as principal components so that they are as small as possible, and the cumulative energy is above a certain threshold θ, e.g., 0.7. Therefore, there are q_p, instead of H, values in $\mathbf{p}_i$, $i = 1, \cdots, M$, and there are q_u, instead of H, values in $\mathbf{u}_i$, $i = 1, \cdots, N$.

Now, we apply an iterative self-constructing clustering algorithm [46], which is an iterative version of that used in [23], to partition the item and user vectors into item and user groups, respectively. We feed the item vectors, $\mathbf{p}_i$, $1 \le i \le M$, as the training patterns, to the clustering algorithm. For each pattern $\mathbf{p}_i$, $i = 1, \cdots, M$, the membership degree of the pattern to each existing cluster C_j, with center $\left[c_{1j}^p \cdots c_{q_p j}^p\right]^T$ and deviation $\left[d_{1j}^p \cdots d_{q_p j}^p\right]^T$, is calculated as

$$G_j(\mathbf{p}_i) = \prod_{k=1}^{q_p} \exp\left[-\left(\frac{p_{ki} - c_{kj}^p}{d_{kj}^p}\right)^2\right]. \tag{10}$$

If $G_j(\mathbf{p}_i) < \rho$ for all existing clusters, a new cluster is created, and $\mathbf{p}_i$ is included in the new cluster. Otherwise, $\mathbf{p}_i$ is added into the cluster with the largest membership degree. Note that Gaussian is adopted to describe the data distribution of each cluster, and $G_j(\mathbf{p}_i)$ lies in the range of $(0, 1]$. Suppose we have M_g clusters $C_1^p, C_2^p, \cdots, C_{M_g}^p$ at the end, each having its own center and deviation. Each cluster is regarded as one item group. Therefore, we have M_g item groups, denoted $p_1^g, p_2^g, \cdots, p_{M_g}^g$, respectively. Let

$$t_{ij}^p = G_j(\mathbf{p}_i) \tag{11}$$

for $1 \le i \le M$ and $1 \le j \le M_g$. Then, we form the transformation matrix

$$\mathbf{T}_p = \begin{bmatrix} t_{11}^p & t_{12}^p & \cdots & t_{1M_g}^p \\ t_{21}^p & t_{22}^p & \cdots & t_{2M_g}^p \\ \vdots & \vdots & \ddots & \vdots \\ t_{M1}^p & t_{M2}^p & \cdots & t_{MM_g}^p \end{bmatrix} \tag{12}$$

which is an $M \times M_g$ matrix, containing all the membership degrees of the products p_i, $1 \le i \le M$, to the item groups p_j^g, $1 \le i \le M_g$.

Next, we feed the user vectors, $\mathbf{u}_i$, $1 \le i \le N$, as the training patterns, to the clustering algorithm. Suppose we have N_g clusters C_1^u, $C_2^u, \cdots$, $C_{N_g}^u$ at the end. Each cluster is regarded as one user group. Therefore, we have N_g user groups, denoted u_1^g, $u_2^g, \cdots$, $u_{N_g}^g$, respectively. Let t_{ij}^u denote the membership degree of $\mathbf{u}_i$ to cluster C_j^u, computed as

$$t_{ij}^u = G_j(\mathbf{u}_i) = \prod_{k=1}^{q_u} \exp\left[-\left(\frac{u_{ki} - c_{kj}^u}{d_{kj}^u}\right)^2\right] \tag{13}$$

for $1 \le i \le N$ and $1 \le j \le N_g$. Then, we form the transformation matrix

$$\mathbf{T}^u = \begin{bmatrix} t_{11}^u & t_{12}^u & \cdots & t_{1N_g}^u \\ t_{21}^u & t_{22}^u & \cdots & t_{2N_g}^u \\ \vdots & \vdots & \ddots & \vdots \\ t_{N1}^u & t_{N2}^u & \cdots & t_{NN_g}^u \end{bmatrix} \tag{14}$$

which is an $N \times N_g$ matrix, containing all the membership degrees of the users u_i, $1 \le i \le N$, to the user groups u_j^g, $1 \le j \le N_g$.

4.3. Step 3—Getting a Reduced User-Item Rating Matrix

In this step, we transform the original user-item rating matrix $\mathbf{R}$ to the reduced matrix $\mathbf{R}_g$. We first convert $\mathbf{R}$ to a matrix $\mathbf{B}$ by

$$\mathbf{B} = \mathbf{R}\mathbf{T}_p \tag{15}$$

which is an $N \times M_g$ matrix, indicating the ratings for the M_g item groups by the N users. Next, we convert $\mathbf{B}$ to $\mathbf{R}_g$ by

$$\mathbf{R}_g = \{\mathbf{T}_u\}^T\mathbf{B} = \{\mathbf{T}_u\}^T\mathbf{R}\mathbf{T}_p = \begin{bmatrix} r_{11}^g & r_{12}^g & \cdots & r_{1M_g}^g \\ r_{21}^g & r_{22}^g & \cdots & r_{2M_g}^g \\ \vdots & \vdots & \ddots & \vdots \\ r_{N_g1}^g & r_{N_g2}^g & \cdots & r_{N_gM_g}^g \end{bmatrix} \tag{16}$$

which is an $N_g \times M_g$ matrix. Note that $\mathbf{R}_g$ contains the ratings for the M_g item groups by the N_g user groups. We call $\mathbf{R}_g$ the reduced user-item rating matrix.

4.4. Step 4—Calculating Group Rating Scores

In this step, one preference list of item groups is derived for each user group. Since low numbers of item groups and user groups are involved, this step can be done efficiently. Firstly, a correlation graph is built from the reduced user-item rating matrix $\mathbf{R}_g$. Then, a series of random walks based on ItemRank [9] are performed.

Based on $\mathbf{R}_g$, we construct a correlation graph that shows the inter-relationship among the item groups. Each item group is regarded as a node in the graph, and thus we have M_g nodes in total. The weight m^g_{ij} on the edge between node p^g_i and node p^g_j, $1 \le i,\ j \le M_g$ is properly defined [23]. When the correlation graph is completed, we have the following correlation matrix:

$$\mathbf{M}_g = \begin{bmatrix} m^g_{11} & m^g_{12} & \cdots & m^g_{1M_g} \\ m^g_{21} & m^g_{22} & \cdots & m^g_{2M_g} \\ \vdots & \vdots & \ddots & \vdots \\ m^g_{M_g1} & m^g_{M_g2} & \cdots & m^g_{M_gM_g} \end{bmatrix} \tag{17}$$

which is a $M_g \times M_g$ matrix. Then, we normalized each column by:

$$m^g_{ij} \leftarrow \frac{m^g_{ij}}{\sum_{k=1}^{M_g} m^g_{kj}},\ 1 \le i \le M_g \tag{18}$$

for $1 \le j \le M_g$.

After creating the correlation matrix $\mathbf{M}_g$, we proceed with ItemRank to predict a preference list of item groups for each user group. Consider any user group u^g_i, $1 \le i \le N_g$. Let $\mathbf{h}_i(0)$ be

$$\mathbf{h}_i(0) = \begin{bmatrix} 1/M_g \\ 1/M_g \\ \vdots \\ 1/M_g \end{bmatrix} \tag{19}$$

which is a vector with M_g elements. The following operation

$$\mathbf{h}_i(t+1) = \alpha \mathbf{M}_g \mathbf{h}_i(t) + (1-\alpha)\mathbf{r}^g_i \tag{20}$$

is performed iteratively for $t = 0,\ 1,\ 2,\ \cdots$ until convergence. Note that $\mathbf{r}^g_i$ is the transpose of the ith row of $\mathbf{R}_g$ and $\alpha \in [0, 1]$ is a user-defined constant. A common choice for α is 0.85. Usually, convergence is reached after 20 iterations [9]. Let $\mathbf{h}_i$ be the converged result, having M_g components. Then, $\mathbf{h}_i$ is the predicted preference list of item groups for the user group u^g_i, $1 \le i \le N_g$.

4.5. Step 5—Deriving Individual Preference Lists

Now, we have obtained $\mathbf{h}_1,\ \mathbf{h}_1,\ \cdots,\ \mathbf{h}_{N_g}$ for user groups $u^g_1,\ u^g_2,\ \cdots,\ u^g_{N_g}$, respectively. Each $\mathbf{h}_i$, $1 \le i \le N_g$, is a preference list in which M_g item groups are involved. However, we are interested in recommending individual products to each user. In this step, we do reverse transformation to get a predicted preference list of products for each user.

Firstly, we find a predicted preference list of products for each user group. We compute vector $\mathbf{y}_i$ by

$$\mathbf{y}_i = \mathbf{T}_p \mathbf{h}_i \tag{21}$$

for $1 \le i \le N_g$. Clearly, $\mathbf{y}_i$ is of length M, and it is the predicted preference list of products for user group u_i^g. Next, we compute vector $\mathbf{s}_i$ as

$$\mathbf{s}_i = t_{i1}^u \mathbf{y}_1 + t_{i2}^u \mathbf{y}_2 + \cdots + t_{iN_g}^u \mathbf{y}_{N_g} \tag{22}$$

for $1 \le i \le N$, which is the predicted preference list of products for user u_i. Similar to ItemRank, the products can be recommended to user u_i in the order according to the magnitudes of the elements in $\mathbf{s}_i$.

5. Experimental Results

To evaluate the performance of our proposed approach, we conduct a set of experiments on several benchmark datasets. For convenience, we call our approach CFUCC (Collaborative Filtering based on User Comments and Clustering) in the remainder of this section. We also compare our CFUCC approach with some other collaborative filtering recommender systems.

Two metrics are adopted for comparison of recommendation accuracy, mean absolute error (MAE) and root mean squared error (RMSE) [47]. Let $\mathcal{P}$ be the set of all products, and $\mathcal{L}_i$ and $\mathcal{T}_i$ be two sets containing the products user u_i has rated in the training set and in the testing set, respectively. Note that it is required that none of $\mathcal{L}_i$ is empty, i.e., $\mathcal{L}_i \neq \varnothing$, $1 \le i \le N$. To compute MAE or RMSE, we have to convert predicted preferences to corresponding predicted scores. Let $\hat{r}_{ij}$ be the predicted score corresponding to the predicted preference of product p_j for user u_j. We compute $\hat{r}_{ij}$ by

$$\hat{r}_{ij} = r_i^a + \frac{\sum_{k=1 \wedge k \neq i}^{N} \left[\mathrm{Sim}(u_i, u_k) \times \left(r_{kj} - r_k^a \right) \right]}{\sum_{k=1 \wedge k \neq i}^{N} \mathrm{Sim}(u_i, u_k)} \tag{23}$$

where r_i^a is the average of the ratings in $\mathcal{L}_i$, r_k^a is the average of the ratings in $\mathcal{L}_k$, r_{kj} is the rating for product j in $\mathcal{L}_k$, and $\mathrm{Sim}(u_i, u_k)$ indicates the similarity between user u_i and user u_k defined by the cosine of the predicted preference lists $\mathbf{s}_i$ and $\mathbf{s}_k$:

$$\mathrm{Sim}(u_i, u_k) = \frac{\mathbf{s}_i \cdot \mathbf{s}_k}{\sqrt{\mathbf{s}_i \cdot \mathbf{s}_i} \sqrt{\mathbf{s}_k \cdot \mathbf{s}_k}} \tag{24}$$

with '$\cdot$' being the inner product operator for vectors. Then, MAE and RMSE are defined as

$$\mathrm{MAE} = \frac{\sum_{i=1}^{N} \sum_{j=1 \wedge r_{ij} \in \mathcal{T}_i}^{M} \left| r_{ij} - \hat{r}_{ij} \right|}{\sum_{i=1}^{N} |\mathcal{T}_i|} \tag{25}$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} \sum_{j=1 \wedge r_{ij} \in \mathcal{T}_i}^{M} \left(r_{ij} - \hat{r}_{ij} \right)^2}{\sum_{i=1}^{N} |\mathcal{T}_i|}}. \tag{26}$$

Note that a low MAE or RMSE value indicates the superiority of a recommender system.

A 5-fold cross-validation is adopted for our experiments. In each experiment, the entries in a dataset are split randomly into five different subsets. Then, five runs are performed. Each time, four of the five subsets are used for training, and the remaining one is used for testing. Then, the results of the five runs are averaged. Note that in each of the following experiments, the predicted preference lists are derived from the training set and all MAE and RMSE are measured on the testing set. In addition, no overlapping exists between the training set and the testing set in any experiment.

In the following experiments, we work with three datasets, Amazon Digital Music, Amazon Video Game, and Amazon Apps for Android. They were collected by Julian McAuley from Amazon.com over a period of 18 years (2014–1996) and were described in [48,49]. Each piece of data contains the information as shown in Figure 2. Several fields of information were recorded.

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the
piano.  He is having a wonderful time playing these old hymns.
The music  is at times hard to read because we think the book
was published for singing from more than playing from.  Great
purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


Figure 2. Original user review.

However, we are interested in the fields of 'reviewerID' (user ID), 'asin' (item ID), 'reviewText' (user text comments to the item), and 'overall' (user's rating to the item). Among them, 'asin' and 'reviewText' are used by Word2Vec to form item vectors, while 'reviewerID' and 'overall' are used in forming the user-item rating matrix. Each non-zero entry in the matrix is represented as a triple $\left(u_i, p_i, r_{ij}\right)$ where $r_{ij} \in \{1, 2, 3, 4, 5\}$. The Amazon Digital Music dataset contains 64,706 reviews, with 5541 users and 3568 items. The sparsity of this dataset is $1 - \frac{64706}{5541 \times 3568} = 99.7\%$. The Amazon Video Game dataset contains 231,780 reviews, with 24,303 users and 10,672 items. The sparsity of this dataset is 99.9%. The original Amazon Apps for Android dataset contains 752,937 reviews, with 97,281 users and 13,209 items and a sparsity of 99.93%. To run other systems easily, we randomly take 385,482 reviews, with 35,001 users and a sparsity of 99.92%. The characteristics of these datasets are summarized in Table 3. As can be seen, sparsity and scalability are very serious in all of these datasets.

Table 3. Characteristics of the datasets used for experiments.

	# Of Users	# Of Products	# Of Ratings	Sparsity
Amazon Digital Music	5541	3568	64,706	99.7%
Amazon Video Game	24,303	10,674	231,780	99.9%
Amazon Apps for Android	35,001	13,209	385,482	99.92%

We show the effectiveness, both regarding the accuracy and efficiency, of our approach, CFUCC, by comparing it with some other collaborative filtering methods, including SCC (Self-Constructing Clustering) [23], CFCB (Collaborative Filtering by Clustering Both users and items) [2], BiFu (Biclustering and Fusion) [26], and ICRRS (Iterative method for Calculating Robust Rating Scores) [35]. SCC applies clustering on products, but users are not clustered. BiFu applies K-means to cluster the users and products, and ICRRS uses a reduction technique that decouples the credibility assessment of the cast evaluations from the ranking itself. CFCB clusters users based on users' ratings on products. For a fair comparison, we wrote programs for these methods. All the programs were written in Python 3.6, running on a computer with Intel(R) Core(TW) i7-4790K CPU, 4.00 GHz, 32GB of RAM, and 64 bits windows 10.

5.1. Experiment 1—Amazon Digital Music Dataset

We work with the Amazon Digital Music dataset in this experiment. Table 4 shows comparisons on MAE, RMSE, and Time(s) among CFUCC, SCC, ICRRS, CFCB, and BiFu. In this table, the value obtained by the best method for each case is shown in boldface. For CFUCC, we use $\rho = 0.7$ for item clustering, $\rho = 0.6$ for user clustering, $v_0 = 0.5$, and $\theta = 0.7$ for PCA. CFUCC clusters products into 35 groups and users into 32 groups, SCC clusters products into 204 groups, CFCB clusters products into 35 groups, and BIFU divides into 35 groups for items and users, respectively. As can be seen from

Table 4, CFUCC is the best among the methods. CFUCC has the lowest value, 0.709, in MAE and the lowest value, 0.996, in RMSE, and runs most efficiently, 51.9 s in the CPU time.

Table 4. Performance comparisons among different methods for Experiment 1.

	CFUCC	SCC	ICRRS	CFCB	BiFu
MAE	**0.709**	0.835	0.998	0.830	0.837
RMSE	**0.996**	1.168	1.419	1.235	1.202
Time(s)	**51.9**	945.3	358.5	373.7	660.2

5.2. Experiment 2—Amazon Video Game Dataset

We work with the Amazon Video Game dataset in this experiment. Table 5 shows comparisons among different methods. CFUCC clusters products into 32 groups and users into 15 groups, SCC clusters products into 19 groups, CFCB clusters products into 35 groups, and BIFU divides into 35 and 20 groups for items and users, respectively. As can be seen from this table, CFUCC performs the best. CFUCC has the lowest value, 0.791, in MAE and the lowest value, 1.224, in RMSE, and runs most efficiently, 111.7 s in the CPU time.

Table 5. Performance comparisons among different methods for Experiment 2.

	CFUCC	SCC	ICRRS	CFCB	BiFu
MAE	**0.791**	0.957	0.954	1.757	1.789
RMSE	**1.224**	1.323	1.374	1.948	2.041
Time(s)	**111.7**	2495.5	4187.9	8348.9	13492.2

5.3. Experiment 3—Amazon Apps for Android Dataset

We work with the reduced version of the Amazon Apps for Android dataset in this experiment. Table 6 shows comparisons among different methods. CFUCC clusters products into 25 groups and users into 38 groups, SCC clusters products into 153 groups, CFCB clusters products into 35 groups, and BIFU divides into 25 and 35 groups for items and users, respectively. In this experiment, CFUCC performs the best in every case. CFUCC has the lowest value, 1.024, in MAE, the lowest value, 1.331, in RMSE, and it runs most efficiently, 194.2 s in the CPU time.

Table 6. Performance comparisons among different methods for Experiment 3.

	CFUCC	SCC	ICRRS	CFCB	BiFu
MAE	**1.024**	1.141	1.169	1.218	1.126
RMSE	**1.331**	1.555	1.650	1.641	1.516
Time(s)	**194.2**	6154.9	7507.9	18826.3	24075.3

5.4. Experiment 4—Impact of Imputation

In this experiment, we show how imputation has an impact on the recommended results. Several strategies have been proposed to deal with missing values in the user-item rating matrix. Among them, four popular ones are:

- No imputation. If user u_i has not provided a rating for product p_j, r_{ij} =0. That is, a missing value is treated as 0.
- Imputation by least. Each missing value is replaced with the least positive score. For the case of scores being 1–5, a missing value is replaced with 1.
- Imputation by mid. Each missing value is replaced with the middle of positive scores. For the case of scores being 1–5, a missing value is replaced with 3.

- Imputation by mean. Each missing value is replaced with the mean value of all the positive ratings in the given user-item rating matrix.

Table 7 shows the performance comparison of these four strategies for the Amazon Digital Music dataset. As can be seen, CFUCC is hardly affected by imputation. The reason is that CFUCC uses user comments, instead of rating scores in the user-item rating matrix, to generate and cluster item vectors. Therefore, it is more resistant to the noise caused by imputation strategies. For the other methods, the recommended results depend on the rating scores in the matrix, and thus, they behave differently with different imputation strategies.

Table 7. Impact of imputation on performance for different methods.

		CFUCC	SCC	ICRRS	CFCB	BiFu
No imputation	MAE	**0.996**	1.168	1.419	1.235	1.202
	RMSE	**0.709**	0.835	0.998	0.830	0.837
Imputation by least	MAE	**0.996**	1.028	3.398	3.325	3.305
	RMSE	**0.709**	0.746	3.222	3.139	3.120
Imputation by mid	MAE	**0.996**	1.514	1.630	1.610	1.635
	RMSE	**0.709**	1.381	1.485	1.464	1.568
Imputation by mean	MAE	**0.996**	1.176	1.418	1.174	1.218
	RMSE	**0.709**	0.842	0.998	0.796	0.804

5.5. Experiment 5—Different Ways of Producing User Vectors

We compare three methods of producing user vectors in CFUCC. The three methods are:

- Method 1. This is the method CFUCC adopts, as described previously.
- Method 2. The user identity shown in the "reviewerID" field is inserted repetitively into the review text in the "reviewText" field, as the way we do for generating item vectors. By training a Word2Vec network, the user vectors are produced.
- Method 3. All the review texts of a user are collected in one document. Then, Doc2Vec [50] is applied to obtain a document vector for the document, and the obtained document vector is regarded as the user vector for this user.

Table 8 shows the performance comparison of these methods for the Amazon Digital Music dataset.

Table 8. Performance comparisons among different methods of generating user vectors.

	Method 1	Method 2	Method 3
MAE	**0.709**	1.526	1.530
RMSE	**0.996**	1.723	1.742

As can be seen, Method 1, which we adopt in CFUCC, gets the best results. In general, a review given by a user provides comments on items instead of on users. Therefore, the user vectors obtained by Method 2 are less effective. In addition, the collected document for a user is usually too short to represent himself/herself in a distinctive way. Worst of all, a user may not have given any comments. Consequently, Method 3 performs less effectively.

5.6. Experiment 6—Impact of User Comments

We show the effectiveness of using user comments for clustering products and users in CFUCC.

- Method 1. This is the method that CFUCC adopts. That is, it applies Word2Vec on user comments to get item vectors from which user vectors are obtained, and then clusters products and users based on the item and user vectors.

- Method 2. User comments are not used. Products and users are clustered directly based on the rating scores provided in the given user-item rating matrix.

Table 9 shows the performance comparison of these methods for the Amazon Digital Music dataset. Note that both methods use identical values for the parameters involved in the recommendation process.

Table 9. Performance comparisons between using and without using user comments.

	MAE	**RMSE**	**Time(s)**
Method 1	**0.709**	**0.996**	**51.9**
Method 2	1.180	1.410	3248.0

As can be seen, Method 1, which we adopt in CFUCC, performs better in terms of both accuracy and efficiency. Method 2 uses the user-item rating matrix for clustering. This causes the sparsity problem and makes the recommended results less accurate. Clearly, both the MAE and RMSE of Method 2 are worse than those of Method 1. Secondly, Method 2 clusters users directly. Since there are 3568 products in Amazon Digital Music dataset, each pattern is a 3568-dimensional vector to be considered in the clustering process. In Method 1, by PCA and Word2Vec, each pattern is only a vector of size less than 100. Therefore, users can be clustered faster by Method 1, and recommendation can be done more efficiently.

5.7. Experiment 7—Impact of Parameter Settings

The energy threshold θ determines the degree of reduction on the dimensionality of the vectors produced by PCA. Table 10 compares the performance of four different values of θ for the Amazon Digital Music dataset. The columns "URdim" and "IRdim" indicate the reduced dimensionality of user vectors and item vectors, respectively, after PCA. Note that the original vectors have 100 dimensions. When $\theta = 0.7$, the reduced user vectors have nine dimensions, while the reduced item vectors have 28 dimensions. As expected, a larger θ results in a higher dimensionality of the reduced vectors. Note that CFUCC with $\theta = 0.7$ performs the best. However, the performance is pretty stable with values near 0.7.

Table 10. Performance comparisons among different values of θ.

	URdim	**IRdim**	**MAE**	**RMSE**
$\theta = 0.6$	6	18	0.716	1.012
$\theta = 0.7$	9	28	**0.709**	**0.996**
$\theta = 0.8$	18	43	0.712	0.998
$\theta = 0.9$	37	62	0.730	1.216

Next, we show the impact of different values of ρ on the recommended results obtained by CFUCC. Note that a different ρ value can lead to a different number of clusters produced. A higher ρ will produce more clusters. Table 11 compares the performance of different values of ρ for the Amazon Digital Music dataset. In the left part of the table, the ρ for clustering user vectors is kept at 0.6, while the ρ value varies for clustering item vectors. In the right part of the table, the ρ for the clustering item vectors is kept at 0.7, while the ρ value varies for clustering user vectors. As can be seen, the value of ρ does affect the performance of CFUCC, but not significantly.

Table 11. Performance comparisons among different values of ρ.

ρ=0.6 **For Users**			ρ=0.7 **For Items**		
ρ For Items	**MAE**	**RMSE**	**ρ For Users**	**MAE**	**RMSE**
$\rho = 0.6$	0.723	0.997	$\rho = 0.5$	0.710	0.997
$\rho = 0.7$	**0.709**	**0.996**	$\rho = 0.6$	**0.709**	**0.996**
$\rho = 0.8$	0.712	1.000	$\rho = 0.7$	0.713	0.999

6. Conclusions

We have presented an extension of [23] to address the inaccuracy and inefficiency caused by the data sparsity and scalability in collaborative filtering recommendation. Word2Vec is applied on user comments to assign an item vector to each product. Through the user-item rating matrix, the user vectors of all the users are also produced. Then, the products and users are clustered into item groups and user groups, respectively. Based on these item groups and user groups, recommendations to a user can be made. Experimental results have shown that both the accuracy and efficiency of recommendation are improved by the proposed approach.

Some research work will be investigated for the system in the future. Doc2Vec or other embedding techniques may be used for developing item vectors. Item descriptions, such as those available in the Amazon datasets, can be used for creating item or user vectors. We may use the reviews to learn word embeddings and take each item to be the aggregation of the embedded vectors of the words that are in the reviews. We have only adopted MAE and RMSE to evaluate the quality of the predictions. Other measures such as the precision, recall, and nDCG may be used to evaluate the quality of recommendations. Recently, neural collaborative filtering has been proposed for recommendation [51,52]. It will be interesting to investigate the effectiveness of incorporating it in our system.

Author Contributions: Data curation, P.-M.C., Y.-S.M. and S.-J.L.; Formal analysis, S.-J.L.; Funding acquisition, S.-J.L.; Methodology, P.-M.C.; Project administration, S.-J.L.; Resources, C.-L.H.; Software, P.-M.C. and Y.-S.M.; Validation, Y.-S.M. and C.-L.H.; Writing – original draft, P.-M.C.; Writing – review & editing, S.-J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was financially supported in part by the grants MOST-106-2321-B-037-003, MOST107-2221-E-110-065 and MOST-107-2622-E-110-008-CC3, MOST, by MOST-107-EPA-F-012- 001, EPA, by NSYSU-KMU JOINT RESEARCH PROJECT (#NSYSUKMU 108-P042), and by the Intelligent Electronic Commerce Research Center, NSYSU.

Acknowledgments: The anonymous reviewers are highly appreciated for their comments which were very helpful in improving the quality and presentation of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Resnick, P.; Varian, H.R. Recommender systems. *Commun. ACM* **1997**, *40*, 56–58. [CrossRef]
2. Gong, S. A collaborative filtering recommendation algorithm based on user clustering and item clustering. *J. Softw.* **2010**, *5*, 745–752. [CrossRef]
3. Smith, B.; Linden, G. Two decades of recommender systems at Amazon.com. *IEEE Internet Comput.* **2017**, *21*, 12–18. [CrossRef]
4. Debnath, S.; Ganguly, N.; Mitra, P. Feature weighting in content based recommendation system using social network analysis. In Proceedings of the 17th International Conference on World Wide Web, Beijing, China, 21–25 April 2008; pp. 1041–1042.
5. De Gemmis, M.; Lops, P.; Semeraro, G.; Basile, P. Integrating tags in a semantic content-based recommender. In Proceedings of the 2008 ACM Conference on Recommender Systems, Lausanne, Switzerland, 23–25 October 2008; pp. 163–170.

6. Xiu, Y.; Lan, M.; Wu, Y.; Lang, J. Exploring semantic content to user profiling for user cluster-based collaborative point-of-interest recommender system. In Proceedings of the 2017 International Conference on Asian Language Processing (IALP), Singapore, 5–7 December 2017; pp. 268–271.
7. Linden, G.; Smith, B.; York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.* **2003**, *7*, 76–80. [CrossRef]
8. George, T.; Merugu, S. A scalable collaborative filtering framework based on co-clustering. In Proceedings of the 5th IEEE International Conference on Data Mining, Houston, TX, USA, 27–30 November 2005; pp. 625–628.
9. Gori, M.; Pucci, A.; Roma, V.; Siena, I. Itemrank: A random-walk based scoring algorithm for recommender engines. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; Volume 7, pp. 2766–2771.
10. Adomavicius, G.; Kwon, Y. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Trans. Knowl. Data Eng.* **2012**, *24*, 896–911. [CrossRef]
11. Liu, Q.; Chen, E.; Xiong, H.; Ding, C.H.; Chen, J. Enhancing collaborative filtering by user interest expansion via personalized ranking. *IEEE Trans. Syst. ManCybern. B* **2012**, *42*, 218–233. [CrossRef] [PubMed]
12. Gopalachari, M.V.; Sammulal, P. Personalized collaborative filtering recommender system using domain knowledge. In Proceedings of the 2014 International Conference on Computer and Communications Technologies (ICCCT), Hyderabad, India, 11–13 December 2014; pp. 1–6.
13. Guan, X. On Reducing the Data Sparsity in Collaborative Filtering Recommender Systems. Ph.D. Thesis, University of Warwick, Warwick, UK, 2017.
14. Li, G.; Zhang, Z.; Wang, L.; Chen, Q.; Pan, J. One-class collaborative filtering based on rating prediction and ranking prediction. *Knowl.-Based Syst.* **2017**, *124*, 46–54. [CrossRef]
15. Nilashi, M.; Ibrahim, O.; Bagherifard, K. A recommender system based on collaborative filtering using ontology and dimensionality reduction techniques. *Expert Syst. Appl.* **2018**, *92*, 507–520. [CrossRef]
16. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.-S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.
17. Liang, D.; Krishnan, R.G.; Hoffman, M.D.; Jebara, T. Variational autoencoders for collaborative filtering. In Proceedings of the 27th International Conference on World Wide Web, Lyon, France, 23–27 April 2018; pp. 689–698.
18. Bobadilla, J.; Bojorque, R.; Esteban, A.H.; Hurtado, R. Recommender systems clustering using bayesian non-negative matrix factorization. *IEEE Access* **2018**, *6*, 3549–3564. [CrossRef]
19. Guo, G.; Qiu, H.; Tan, Z.; Liu, Y.; Ma, J.; Wang, X. Resolving data sparsity by multi-type auxiliary implicit feedback for recommender systems. *Knowl.-Based Syst.* **2017**, *138*, 202–207. [CrossRef]
20. Yin, H.; Wang, W.; Chen, L.; Du, X.; Nguyen, Q.V.H.; Huang, Z. Mobi-SAGE-RS: A sparse additive generative model-based mobile application recommender system. *Knowl.-Based Syst.* **2018**, *157*, 68–80. [CrossRef]
21. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
22. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–8 December 2013; Volume 2, pp. 3111–3119.
23. Liao, C.-L.; Lee, S.-J. A clustering based approach to improving the efficiency of collaborative filtering recommendation. *Electron. Commer. Res. Appl.* **2016**, *18*, 1–9. [CrossRef]
24. Hernando, A.; Bobadilla, J.; Ortega, F. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowl.-Based Syst.* **2016**, *97*, 188–202. [CrossRef]
25. Wu, H.; Zhang, Z.; Yue, K.; Zhang, B.; He, J.; Sun, L. Dual-regularized matrix factorization with deep neural networks for recommender systems. *Knowl.-Based Syst.* **2018**, *145*, 46–58. [CrossRef]
26. Zhang, D.; Hsu, C.-H.; Chen, M.; Chen, Q.; Xiong, N.; Lloret, J. Cold-start recommendation using biclustering and fusion for large-scale social recommender systems. *IEEE Trans. Emerg. Top. Comput.* **2014**, *2*, 239–250. [CrossRef]
27. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web, Hong Kong, China, 1–5 May 2001; pp. 285–295.

28. Sarwar, B.M.; Karypis, G.; Konstan, J.; Riedl, J. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In Proceedings of the 5th International Conference on Computer and Information Technology, Dhaka, Bangladesh, 27–28 December 2002; Volume 1, pp. 291–324.
29. Jiang, J.-Y.; Liou, R.-J.; Lee, S.-J. A fuzzy self-constructing feature clustering algorithm for text classification. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 335–349. [CrossRef]
30. Das, J.; Mukherjee, P.; Majumder, S.; Gupta, P. Clustering-based recommender system using principles of voting theory. In Proceedings of the 2014 International Conference on Contemporary Computing and Informatics (IC3I), Mysore, India, 27–29 November 2014; pp. 230–235.
31. Zahra, S.; Ghazanfar, M.A.; Khalid, A.; Azam, M.A.; Naeem, U.; Prugel-Bennett, A. Novel centroid selection approaches for kmeans-clustering based recommender systems. *Inf. Sci.* **2015**, *320*, 156–189. [CrossRef]
32. Liu, H.; Wu, J.; Liu, T.; Tao, D.; Fu, Y. Spectral ensemble clustering via weighted k-means: Theoretical and practical evidence. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 1129–1143. [CrossRef]
33. Yang, L.; Huang, W.; Niu, X. Defending shilling attacks in recommender systems using soft co-clustering. *IET Inf. Secur.* **2017**, *11*, 319–325. [CrossRef]
34. Park, Y.-J. The adaptive clustering method for the long tail problem of recommender systems. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 1904–1915. [CrossRef]
35. Allahbakhsh, M.; Ignjatovic, A. An iterative method for calculating robust rating scores. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 340–350. [CrossRef]
36. Yang, B.; Lei, Y.; Liu, J.; Li, W. Social collaborative filtering by trust. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 1633–1647. [CrossRef] [PubMed]
37. Musto, C.; Lops, P.; de Gemmis, M.; Semeraro, G. Semantics-aware recommender systems exploiting linked open data and graph-based features. *Knowl.-Based Syst.* **2017**, *136*, 1–14. [CrossRef]
38. Han, J.; Zheng, L.; Xu, Y.; Zhang, B.; Zhuang, F.; Yu, P.S.; Zuo, W. Adaptive deep modeling of users and items using side information for recommendation. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 737–748. [CrossRef]
39. Victor, P.; Verbiest, N.; Cornelis, C.; Cock, M.D. Enhancing the trust-based recommendation process with explicit distrust. *ACM Trans. Web* **2013**, *7*, 6. [CrossRef]
40. Forsati, R.; Mahdavi, M.; Shamsfard, M.; Sarwat, M. Matrix factorization with explicit trust and distrust side information for improved social recommendation. *ACM Trans. Inf. Syst.* **2014**, *32*, 17. [CrossRef]
41. Huang, S.; Ma, J.; Cheng, P.; Wang, S. A hybrid multigroup coclustering recommendation framework based on information fusion. *ACM Trans. Intell. Syst. Technol.* **2015**, *6*, 27. [CrossRef]
42. Zheng, X.; Luo, Y.; Sun, L.; Chen, F. A new recommender system using context clustering based on matrix factorization techniques. *Chin. J. Electron.* **2016**, *25*, 334–340. [CrossRef]
43. Barkan, O.; Koenigstein, N. Item2Vec: Neural item embedding for collaborative filtering. In Proceedings of the 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), Vietri sul Mare, Italy, 13–16 September 2016; pp. 1–6.
44. Wold, S.; Esbensen, K.; Geladi, P. Principal component analysis. *Chemom. Intell. Lab. Syst.* **1987**, *2*, 37–52. [CrossRef]
45. Fan, J.; Chow, T.W.S. Exactly robust kernel principal component analysis. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 749–761. [CrossRef] [PubMed]
46. Wang, Z.-Y. Some Variants of self-Constructing Clustering. Ph.D. Thesis, National Sun Yat-Sen University, Gaoxiong, Taiwan, 2017.
47. Herlocker, J.; Konstan, J.; Terveen, L.; Reidl, J. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **2004**, *22*, 5–53. [CrossRef]
48. Wan, M.; McAuley, J. Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Hyderabad, India, 11–13 December 2016; pp. 489–498.
49. McAuley, J.; Yang, A. Addressing complex and subjective product-related queries with customer reviews. In Proceedings of the 25th International World Wide Web Conferences Steering Committee, Montreal, QC, Canada, 11–15 April 2016; pp. 625–635.
50. Le, Q.; Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 1188–1196.

51. Dacrema, M.F.; Cremonesi, P.; Jannach, D. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In Proceedings of the 13th ACM Conference on Recommender Systems, Copenhagen, Denmark, 16–20 September 2019; pp. 101–109.
52. Ludewig, M.; Mauro, N.; Latifi, S.; Jannach, D. Performance comparison of neural and non-neural approaches to session-based recommendation. In Proceedings of the 13th ACM Conference on Recommender Systems, Copenhagen, Denmark, 16–20 September 2019; pp. 462–466.