



Article

SoftRec: Multi-Relationship Fused Software Developer Recommendation

Xinqiang Xie , Bin Wang * and Xiaochun Yang * 

School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China;
1610560@stu.neu.edu.cn

* Correspondence: binwang@mail.neu.edu.cn (B.W.); yangxc@mail.neu.edu.cn (X.Y.)

Received: 26 May 2020; Accepted: 18 June 2020; Published: 24 June 2020



Abstract: Collaboration efficiency is of primary importance in software development. It is widely recognized that choosing suitable developers is an efficient and effective practice for improving the efficiency of software development and collaboration. Recommending suitable developers is complex and time-consuming due to the difficulty of learning developers' expertise and willingness. Existing works focus on learning developers' expertise and interactions from their explicit historical information and matching them to specific task. However, such procedures may suffer low accuracy because they ignore implicit information, such as (1) developer–developer collaboration relationships, (2) developer–task implicit interaction relationships, and (3) task–task association relationships, etc. To that end, this paper proposes a multi-relationship fused approach for software developer recommendation (termed SoftRec). First, in addition to explicit developer–task interactions, it considers multivariate implicit relationships, including the three types mentioned above. Second, it integrates these relationships based on joint matrix factorization and generates forecast results upon the architecture of deep neural network. Furthermore, we propose a fast update method to address the cold start issue by making online recommendations for new developers and new tasks. Extensive experiments are conducted on two real-world datasets, and a user study is conducted in a well-known software company. The results demonstrate that SoftRec outperforms four state-of-the-art works.

Keywords: developer recommendation; collaboration relationship; joint matrix factorization; deep neural network; fast model update

1. Introduction

Internet-based software development model has greatly promoted the efficiency of software development. Such as online developer collaboration platforms (e.g., GitHub (<https://github.com/>), StackOverflow (<https://stackoverflow.com/>), SourceForge (<https://sourceforge.net/>) and TopCoder (<https://www.topcoder.com/>)) have become a prevalent paradigm for software development. They usually adopt open-collaboration mode [1,2] with a voluntary or competitive mechanism to facilitate self-organizing collaborative software development, in which the productivity and quality greatly depends on the efficiency of developers' collaboration.

For example, in the process of modern code review (MCR) [3], a developer usually submits a code change to a code review system (e.g., Gerrit (<https://www.gerritcodereview.com/>), Rietveld ([https://en.wikipedia.org/wiki/Rietveld_\(software\)](https://en.wikipedia.org/wiki/Rietveld_(software))), Crucible (<https://www.atlassian.com/software/crucible>)) and recommends a set of suitable developers to review the change. Then, the reviewers would check the change and give some useful suggestions. Next, the developer will refine the change according to these suggestions and commit the change to the main branch of the version control system when the reviewers approve it. For some large open source projects, hundreds of contributors may attempt to change the code, adding new features or fixing bugs every day. These changes are submitted

as pull-requests [4,5] subject to review by the whole community. However, such a mechanism makes it very difficult to find suitable developers as reviewers. For example, Thongtanunam et al. found that 4%-30% of the reviews suffer from a code-reviewer assignment problem, and these reviews require an average of 12 extra days to complete [6], and more than 15% of the complaints were filed the delays in completing pull requests [7]. Similar problems are encountered in StackOverflow, one of the most successful question and answering (Q&A) open-collaboration platforms. It has over 9 million users who has posted more than 16 million questions and answers on programming. However, among those questions, only about 70% of them can be answered and closed in time and millions of questions have yet to be answered properly [1]. As a result, an automatic developer recommendation approach is urgently needed for identifying suitable developers to improve developers' collaboration efficiency.

In recent years, many efforts have been made for developer recommendation. Typical approaches such as the expertise-based developer recommendation and the collaborative filtering (CF)-based developer recommendation. Usually, the expertise-based approaches [2,8] recommend developers based on the ratings of their explicit expertise (e.g., skills, contributions, activeness, workload). The CF-based approaches [4,9,10] which make recommendations based on the assumption that developers with similar historic behaviors or social relationship would behave similarly on future tasks. For example, some research [11,12] integrate the explicit social information into the recommender system and recommend developers based on their friendships.

However, due to the difficulty of matching developers' expertise and willingness, together with the sparsity of developer–task explicit interactions, previous approaches have not achieved ideal results. In practice, there are a lot of implicit relationships between developers and tasks that can be leveraged to help find suitable developers. Take GitHub's code review as an example, as shown in Figure 1a–c, where u_i denotes the developer, p_i denotes the pull-request, dots with blank cells denotes the developers' explicit review (or comment) on pull-request, and dots with grey cells denotes the developers' implicit interaction (e.g., star, browse). Since u_2 has implicit interaction on p_6 , we can infer that u_2 has potential preferences for p_6 (denoted as $u_2 \rightarrow p_6$). Since u_1 has a close collaboration relationship with u_2 ($u_1 \leftrightarrow u_2$), we can infer the potential relationship $u_1 \rightarrow p_6$ (red circles in Figure 1a). Similarly, since u_1 has preferences for p_1 ($u_1 \rightarrow p_1$), and p_1 has an association relationship with p_4 ($p_1 \rightarrow p_4$), we can infer the potential relationship ($u_1 \rightarrow p_4$).

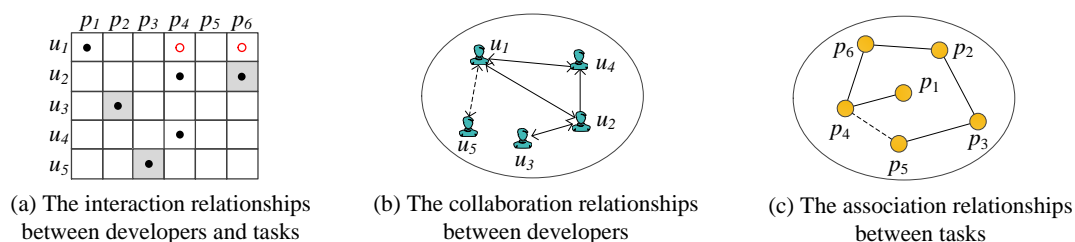


Figure 1. Illustration of the implicit relationships between developers and tasks based on GitHub's code review.

By analyzing GitHub's code review recommendation, we find that developers are more likely to accept a recommended pull-request if (1) they have reviewed code or solved issues of the contributors of the same pull-request (i.e., the collaboration relationships between developers can keep continuity); (2) they have browsed or stared (besides explicitly reviewed) the pull-request (i.e., implicit interaction relationships help improve recommendation accuracy); or (3) they have reviewed a similar pull-request of the recommended one. Such information can be exploited to improve the accuracy of developer recommendation.

To leverage the above mentioned relationships between developers and tasks, we develop a novel multi-relationship fused approach for software developer recommendation (SoftRec). The major contributions of SoftRec are:

- We formally define the multivariate relationships between developers and tasks, including three types: (1) developer–developer collaboration relationships, (2) developer–task interaction relationships, and (3) task–task association relationships.
- We propose a multi-relationship fused approach to recommend developers based on joint matrix factorization and generate forecast results upon the architecture of deep neural network. To our best knowledge, this is the first attempt to integrate these three implicit relationships into developer recommendation.
- We propose a fast approach to update the changes of the model efficiently and improve the recommendation efficiency, which can address the cold start issue.
- To evaluate the effectiveness of SoftRec, we conduct experiments on two real-world datasets: One from GitHub with 2517 developers and 9329 tasks, while the other from a well-known company's GitLab with 590 developers and 15,632 tasks, and we also conduct a user study in this company. By comparisons of four state-of-the-art works, the results demonstrate the advantages of the SoftRec.

The remainder of this paper is organized as follows. We review the related work in Section 2. In Section 3, we present the proposed multi-relationship fused software developer recommendation framework. In Section 4, we present and discuss the result of experiments on real-world datasets and conduct a user study. In Section 5, we discuss the threats to validity of our approach. Concluding remarks with a discussion of some future work are given in Section 6.

2. Related Work

Among previous approaches, collaborative filtering (CF) [13–17] is the most typical approach that achieves great success. CF-based developer recommendations share a similar process, that is, they first calculate the similarity between the given task and other resolved task based on the explicit interaction relationships. Then, the similarity ratings between the given task and each resolved task are assigned to the developers of the corresponding resolved tasks. This way, each developer has a rating to indicate their expertise with respect to the given task. However, CF-based approaches usually suffer from serious sparsity of the developer–task explicit interactions and the cold start issues. For example, by analyzing GitHub datasets, the data sparsity of the developer–task explicit interaction matrix is as low as 0.1351%, which greatly limits the effectiveness of the recommendations.

To address these limitations, previous work incorporated various side information into CF [3,4,18–28]. For example, Zheng et al. [4] proposed a CF-based approach (PR-CF) that generates the latent factor models based on the explicit interaction matrix, and then combines the latent factor models with the tasks' neighborhoods. Jiang et al. [19] constructed an explicit social relationship network on GitHub, and then propose a CF-based recommendation approach based on Co-cluster for developers and tasks. Ma et al. [18] proposed a factor analysis approach called SoRec which was based on joint matrix factorization. It integrates the social information into the rating matrix and shares the users' latent vectors in both the recommender system and the social network. Yu et al. [20] proposed an approach named IR+CN that mines social relationship from historical comments for recommend reviewers. Xia et al. [3] proposed a hybrid approach (TIE) which utilizes text mining and file location to find similar tasks and recommends developers based on conditional probabilities. Bosu et al. [22,23] analyze the characteristics of different kinds of social interaction networks between developers and their influence on the impression of developers. They found that the interactions between developers and tasks can help form an accurate perception of expertise. Its log-joint posterior probability distribution is given as follows:

$$\ln p(\cdot) = \frac{-1}{2\sigma_R^2} \|I^R \odot (R - g(U^T V))\|_F^2 - \frac{1}{2\sigma_C^2} \|I^C \odot (C - g(Q^T U))\|_F^2 + Reg, \quad (1)$$

where C and R denote the social relationship and rating matrices, Q , U , and V denote the latent social, users, and item feature matrices, respectively. C , R , U , V , and Q follow Gaussian distributions

with the means of 0 and variances of σ_C^2 , σ_R^2 , σ_U^2 , σ_V^2 , σ_Q^2 , respectively. I^R and I^C denote the indicator matrices, \odot denotes the Hadamard product, $g(x)$ is the logistic function, and Reg is the regularization term. Parameters U , V , and Q can be learned by maximizing Equation (1). The common limitation of CF-based developer recommendation approaches is that they usually suffer from low accuracy caused by the sparsity of explicit developer–task interactions.

In recent years, deep learning approaches have been widely used to build recommender systems in many fields [13,17,29–32]. For example, He et al. [13] proposed a neural network architecture to model the latent features of users and items and devise a general framework (NeuCF) for collaborative filtering. Wang et al. [17] proposed a graph collaborative filtering approach based on graph neural networks, which explicitly encodes the user–item interaction relationships in the form of high-order connectivities with embedding propagation. Xue et al. [29] proposed a deep matrix factorization models (DMF) with a neural network that map the users and items into a common low-dimensional space with non-linear projections. He et al. [31] proposed a deep learning model (NFM) that unifies the strengths of factorization machines and deep neural networks for sparse rating modelling.

Despite prevalence and effectiveness, we argue that previous approaches with side information are insufficient, since they only consider the explicit collaborative similarity (e.g., explicit interactions), which lacks of rich semantics (e.g., various implicit relationships). In real-world applications, there are more implicit relationships between developers and tasks (e.g., mentioned in Figure 1), which particularly help understand developer’s behaviors and preferences. In this paper, we try to integrate both explicit and implicit relationships among developers and tasks to make accurate developer recommendations.

3. The Multi-Relationship Fused Software Developer Recommendation

In this section, we propose SoftRec, a novel multi-relationship fused approach for developer recommendation. Its overall recommendation process is shown in Figure 2, there are three parts: (1) the input is multi-relationships, in which we define the collaboration relationship matrix C , the interaction relationship matrix R and the association relationship matrix S in steps ①, ② and ③ respectively; (2) the fusion of multi-relationships as shown in step ④, in which we share the common developer latent vector U_i in both C and R as well as the common task latent vector V_j in both R and S based on joint matrix factorization; and (3) the output is the developer prediction in step ⑤, in which we propose to project the vectors U_i and V_j into a latent structured space based on the architecture of deep neural network. Furthermore, we propose a fast model update approach for SoftRec.

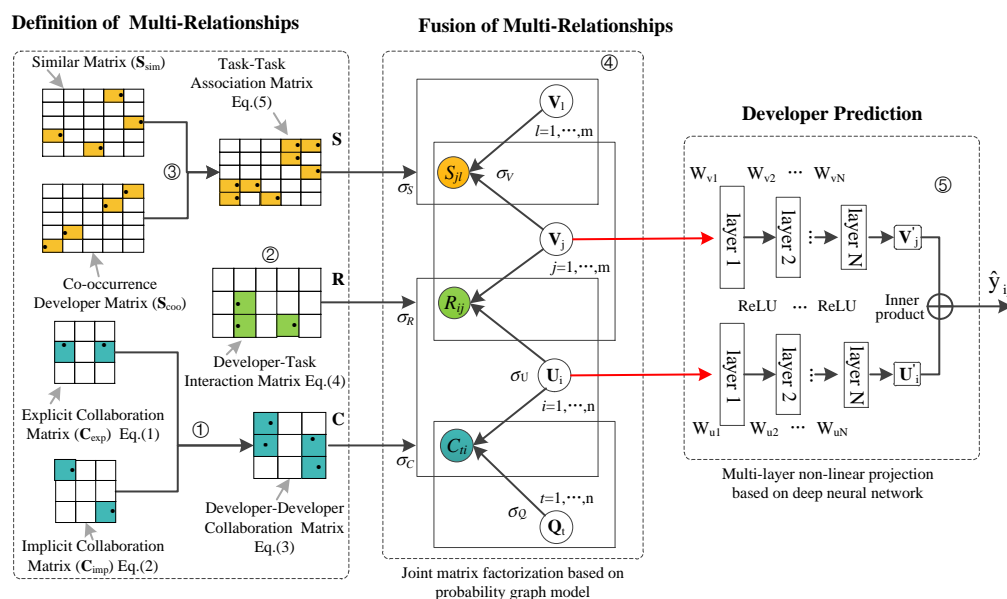


Figure 2. Recommendation process of SoftRec.

3.1. Definition of Multi-Relationships

In this section, we first formally define the developer–developer collaboration relationship, developer–task interaction relationship and task–task association relationship illustrated in Figure 1.

3.1.1. Developer–Developer Collaboration Relationship

Recent research [33] at Google shows that code review usually depends on close working relationships between authors and reviewers. Through a study of Neusoft Corporation's R&D teams (Neusoft Corporation (<https://www.neusoft.com/>) is the largest software service provider in China, with more than 18,000 developers and tens of thousands of commercial customers all over the world), we also found that developers usually prefer reviewers that have close collaboration relationships with them. Therefore, collaboration relationships can be leveraged to improve developer recommendation accuracy. In this paper, we categorize the collaboration relationship into explicit collaboration and implicit collaboration. The former refers to direct interaction between developers and the latter refers to indirect collaboration between developers.

Explicit Collaboration. Suppose \mathcal{O} denotes a set of interaction objects between u and u' , \mathcal{A}_o denotes a set of actions which performed on object $o \in \mathcal{O}$, the explicit collaboration relationship is formalized as:

$$c_{exp}(u, u') = \begin{cases} \frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \sum_{a \in \mathcal{A}_o} \frac{1}{|\mathcal{A}_o|} \sum_{i=1}^s \omega^{s-i} \pi_1(u, u', o, a, i), & \text{if exists;} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where s denotes the number of interactions while interactive object is o and action is a . $\omega \in [0, 1]$ is the decay factor used to weaken the influence of multiple interactions between the same developers. Here, let $\pi_1(\cdot) = \frac{interactTime(u, u', o, a, i) - beginTime}{endTime - beginTime}$, which is used to reflect the influence of time locality. It supposes that the most recent interaction is more important than it was a long time ago. $interactTime(u, u', o, a, i)$ denotes the i th occurrence time of the action a which occurred between u and u' on object o , $i = 1$ denotes the first interaction, $beginTime$ and $endTime$ denote the earliest and latest occurrence time of all interactions in the datasets.

Example 1. Suppose u and u' have the interactive object $\mathcal{O} = \{o_1, o_2\}$, where \mathcal{O} may include many elements (e.g., *pull_requests*, *issue_comments* and *commit_comments* in GitHub dataset). Suppose $o_1 = \text{issue_comments}$, $o_2 = \text{commit_comments}$ and the performed actions $\mathcal{A}_{o_1} = \{a_1\}$, $\mathcal{A}_{o_2} = \{a_2\}$. Suppose $a_1 = a_2 = \text{comment}$, and the performed number of a_1 and a_2 are $s_{a_1} = 1$ and $s_{a_2} = 2$. Suppose the occurrence time of a_1 and a_2 are $t_{a_{11}}$ and $t_{a_{21}}$, $beginTime$ and $endTime$ are t_b and t_e , respectively. We can calculate: $c_{exp}(u, u') = \frac{1}{2} \{ \omega^0 \frac{(t_{a_{11}} - t_b)}{t_e - t_b} + \omega^0 \frac{(t_{a_{21}} - t_b)}{t_e - t_b} + \omega^1 \frac{(t_{a_{22}} - t_b)}{t_e - t_b} \}$.

Implicit Collaboration. Suppose \mathcal{O}' denotes a set of co-occurrence objects (e.g., organization, project, team) between u and u' . The implicit collaboration relationship is formalized as:

$$c_{imp}(u, u') = \begin{cases} \frac{\sum_{o \in \mathcal{O}'} I(u, u', o)}{\sum_{o \in \mathcal{O}'} I(u, u', o) + \psi}, & \text{if exists;} \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

where $I(\cdot)$ is the indicator function. If u, u' are co-occurred in o , its value is 1, otherwise, its value is 0. ψ is a constant, used to regularize the results, similar to its use in [4], its default value is 100.

Example 2. Suppose u and u' participated in a same project p , in addition, they belong to the same technical organization org . We can get $\mathcal{O}' = \{p, org\}$, and $c_{imp}(u, u') = \frac{1+1}{1+1+100}$.

Based on Equations (2) and (3), we formalize the final collaboration relationship between u and u' as:

$$c(u, u') = \alpha \cdot c_{exp}(u, u') + (1 - \alpha) \cdot c_{imp}(u, u'), \quad (4)$$

where $\alpha \in [0, 1]$ is a weight which used to balance the explicit and implicit collaboration. Based on Equation (2) to Equation (4), we can obtain the explicit collaboration matrix, the implicit collaboration matrix and the final collaboration matrix, denote as C_{imp} , C_{exp} and C respectively.

The strength of the collaboration relationship is measured based on the developers' interactive behaviors for the first time. From the viewpoint of quantitative analysis, it not only considers the influence of the time window for the interactions but also the multiple interactions between the same developers. From the viewpoint of qualitative analysis, the collaboration relationship might be specific and targeted to reflect the intimacy between developers in their development process than traditional generalized social relationships (as shown in Section 4.2.1).

3.1.2. Developer–Task Interaction Relationship

The strength of the interaction relationship reflects the degree of developers' preferences for tasks. It means that the closer developer u interacts with task v , the more likely u suits for v (evidences in recent investigations [10]). Next, we formally define the interaction relationship between developer and task.

Interaction Relationship. Similar to Equation (2), suppose \mathcal{A}_v denotes a set of actions (e.g., browsed, stared, explicitly comment) which performed on task v by developer u , the interaction relationship $r(u, v)$ is formalized as:

$$r(u, v) = \begin{cases} |\mathcal{A}_v|^{-1} \sum_{a' \in \mathcal{A}_v} \sum_{i=1}^s s^{-1} \omega^{s-i} \pi_2(u, v, a', i), & \text{if exists;} \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

where $\pi_2(u, v, a', i) = \frac{\text{interactTime}(u, v, a', i) - \text{beginTime}}{\text{endTime} - \text{beginTime}}$, and $\text{interactTime}(u, v, a', i)$ is the occurrence time of the action a' which performed on task v by developer u , beginTime , endTime and ω are similar to the definitions in Equation (2). Based on Equation (5), we can calculate the interaction relationship matrix R .

Example 3. Suppose u has the interactive actions $\mathcal{A}_v = \{a_1, a_2\}$ on task v . Suppose $a_1 = \text{comment}$ and $a_2 = \text{star}$ (e.g., thumbs-up) and the performed number of a_1 and a_2 are $s_{a_1} = 2$ and $s_{a_2} = 1$, the occurrence time of a_1 and a_2 are $t_{a_{11}}, t_{a_{12}}$ and $t_{a_{21}}$, beginTime and endTime are t_b and t_e , respectively. We can calculate: $r(u, v) = \frac{1}{2} \{ (\omega^1 \frac{(t_{a_{11}} - t_b)}{t_e - t_b}) + \omega^0 \frac{(t_{a_{12}} - t_b)}{t_e - t_b}) + \omega^0 \frac{(t_{a_{21}} - t_b)}{t_e - t_b} \}$.

Different from the existing methods [17,20], that usually set $r(u, v)=1$ by default while the interaction exists. Here, we quantitatively calculate $r(u, v)$ based on the developers' interactive behaviors and consider that the strength of the interaction will be affected by the time locality and interaction numbers. In addition, there may be many interactive objects between developers in Equation (2), but for Equation (5), the interactive object defaults to task v .

3.1.3. Task–Task Association Relationship

Intuitively, developers are more willing to accept the tasks which are similar to those they have done before. Therefore, how to measure the similarity relationship between tasks is an important issue. Existing item-based collaborative filtering methods [14,16,34] only consider the collaborative similarity relationship (i.e., the item similarity evidenced by user interactions like ratings and purchases), which lacks of concrete semantics. In real-world applications, there typically exist multiple relationships between tasks that have concrete semantics, and they are particularly helpful to understand developer behaviors. For example, some tasks may have similar titles, file paths or related

similar source code, and others although have no such explicit similar features, they share the same contributors or reviewers, etc. In the context of this research, we define two types of similarity: (1) co-occurrence developer similarity between tasks, and (2) text similarity between tasks. The association relationship is formalized as:

$$s(v, v') = (1 - \beta) \cdot co_dev(v, v') + \beta \cdot tex_sim(v, v'), \quad (6)$$

where $co_dev(v, v') = \frac{|\mathcal{N}_v \cap \mathcal{N}_{v'}|}{|\mathcal{N}_v \cup \mathcal{N}_{v'}|}$ denotes the co-occurrence relationship of developers in v and v' , where \mathcal{N}_v and $\mathcal{N}_{v'}$ denotes the corresponding developer sets associated with v and v' respectively. β is a weight which used to balance the text similarity and the developer co-occurrence relationship. $tex_sim(v, v') = \frac{\mathbf{e}_t \cdot \mathbf{e}_{t'}}{\|\mathbf{e}_t\| \|\mathbf{e}_{t'}\|}$ denotes the cosine similarity of v and v' , where \mathbf{e}_t and $\mathbf{e}_{t'}$ denote the text vectors of v and v' , here, we learn \mathbf{e}_t and $\mathbf{e}_{t'}$ based on Doc2Vec [35], a popular text vector representations model, which can learn vector representations for variable length of texts such as sentences and documents. Here, we implement Doc2Vec upon the model of gensim PV-DBOW (<https://radimrehurek.com/gensim/models/doc2vec.html>) due to its simplicity and ease of implementation. The calculate process is that, for each task, there are some text description (e.g., title, tags, abstract, content and code file paths), we first concatenate each task's text description and generate the concatenated text vectors (i.e., $\mathbf{e}_t, \mathbf{e}_{t'}$). Then we calculate the cosine similarity of those concatenated text vectors. Based on Equation (6), we can calculate the similar relationship matrix S .

3.2. Fusion of Multi-Relationships

In this section, we integrate the collaboration relationship and association relationship with the interaction relationship based on joint matrix factorization [18].

Suppose $C_{n \times n} \in \mathbb{R}^{n \times n}$, $R_{n \times m} \in \mathbb{R}^{n \times m}$, $S_{m \times m} \in \mathbb{R}^{m \times m}$, $Q \in \mathbb{R}^{d \times n}$, $U \in \mathbb{R}^{d \times n}$ and $V \in \mathbb{R}^{d \times m}$ follow Gaussian distributions [18,36] with the means of 0 and variances of σ_C^2 , σ_R^2 , σ_S^2 , σ_Q^2 , σ_U^2 and σ_V^2 , respectively. Here, Q , U and V denote the latent features of the collaboration relationships, the developers and the tasks, respectively. d is the vector size.

As shown in Figure 2 middle subfigure, for each developer u_i , the latent collaboration feature vector is denoted by $Q_t \in \mathbb{R}^{d \times 1}$, and the latent developer feature vector is denoted by $U_i \in \mathbb{R}^{d \times 1}$. They are calculated by factorizing $C_{n \times n}$. Similarly, for each task v_j , the latent task feature vector is denoted by $V_j \in \mathbb{R}^{d \times 1}$, and the latent association feature vector is denoted by $V_l \in \mathbb{R}^{d \times 1}$. They are calculated by factorizing $S_{m \times m}$. Here, U_i is the shared latent developer feature vector, calculated by factorizing C_{ti} and R_{ij} . V_j is the shared latent task feature vector, calculated by factorizing R_{ij} and S_{jl} .

To learn U , V and Q , we design a log-joint posterior probability distribution function as shown in Equation (7):

$$\begin{aligned} \ln p = & \frac{-1}{2\sigma_R^2} \|I^R \odot (R - g(U^T V))\|_F^2 - \frac{1}{2\sigma_C^2} \|I^C \odot (C - g(U^T Q))\|_F^2 \\ & - \frac{1}{2\sigma_S^2} \|I^S \odot (S - g(V^T V))\|_F^2 - \frac{1}{2} \left(\frac{1}{\sigma_U^2} \|U\|_F^2 + \frac{1}{\sigma_V^2} \|V\|_F^2 + \frac{1}{\sigma_Q^2} \|Q\|_F^2 \right) \\ & - \frac{1}{2} (\ln \sigma_R^2 \|I^R\|_F^2 + \ln \sigma_C^2 \|I^C\|_F^2 + \ln \sigma_S^2 \|I^S\|_F^2) - \frac{d}{2} (m \ln \sigma_U^2 + n \ln \sigma_V^2 + m \ln \sigma_Q^2), \end{aligned} \quad (7)$$

where I^R , I^C and I^S denote the indicator matrices. \odot denotes the Hadamard product of two matrices. $g(x) = \frac{1}{1 + \exp(-x)}$ is a logistic function [18] used to limit the predicted value of $U_i^T V_j$ to the interval of $[0, 1]$. The model parameters U , V and Q can be learned by maximizing the log-joint posterior in Equation (7) which is equivalent to minimizing the objective function in Equation (8):

$$\begin{aligned} \mathcal{L}_1 = & \frac{1}{2} \|I^R \odot (R - g(U^T V))\|_F^2 + \frac{\lambda_C}{2} \|I^C \odot (C - g(U^T Q))\|_F^2 + \frac{\lambda_S}{2} \|I^S \odot (S - g(V^T V))\|_F^2 \\ & + \frac{1}{2} (\lambda_U \|U\|_F^2 + \lambda_V \|V\|_F^2 + \lambda_Q \|Q\|_F^2), \end{aligned} \quad (8)$$

where $\lambda_C = \frac{\sigma_R^2}{\sigma_C^2}$, $\lambda_S = \frac{\sigma_R^2}{\sigma_S^2}$, $\lambda_U = \frac{\sigma_R^2}{\sigma_U^2}$, $\lambda_V = \frac{\sigma_R^2}{\sigma_V^2}$, $\lambda_Q = \frac{\sigma_R^2}{\sigma_Q^2}$. Based on Equation (8), we calculate the partial derivative $\frac{\partial \mathcal{L}_1}{\partial U_i}$, $\frac{\partial \mathcal{L}_1}{\partial V_j}$ and $\frac{\partial \mathcal{L}_1}{\partial Q_t}$ as shown in Equation (9):

$$\begin{cases} \frac{\partial \mathcal{L}_1}{\partial U_i} = \sum_{j=1}^m I_{ij}^R g'(U_i^T V_j) (g(U_i^T V_j) - R_{ij}) V_j + \lambda_C \sum_{t=1}^n I_{it}^C g'(U_i^T Q_t) (g(U_i^T Q_t) - C_{it}) Q_t + \lambda_U U_i, \\ \frac{\partial \mathcal{L}_1}{\partial V_j} = \sum_{i=1}^n I_{ij}^R g'(U_i^T V_j) (g(U_i^T V_j) - R_{ij}) U_i + \lambda_S \sum_{l=1}^m I_{jl}^S g'(V_j^T V_l) (g(V_j^T V_l) - S_{jl}) V_l + \lambda_V V_l, \\ \frac{\partial \mathcal{L}_1}{\partial Q_t} = \lambda_C \sum_{i=1}^n I_{it}^C g'(U_i^T Q_t) (g(U_i^T Q_t) - C_{it}) U_i + \lambda_Q Q_t. \end{cases} \quad (9)$$

To calculate the feature vectors of U_i , V_j and Q_t , we utilize the momentum-based stochastic gradient descent (MSGD) method [37] to update the parameters of SoftRec to accelerate its convergence as shown in Equation (10):

$$\begin{cases} \hat{U}_i \leftarrow \nu \cdot \hat{U}_i + \eta \cdot \frac{\partial \mathcal{L}_1}{\partial U_i}, & U_i \leftarrow U_i - \hat{U}_i, \\ \hat{V}_j \leftarrow \nu \cdot \hat{V}_j + \eta \cdot \frac{\partial \mathcal{L}_1}{\partial V_j}, & V_j \leftarrow V_j - \hat{V}_j, \\ \hat{Q}_t \leftarrow \nu \cdot \hat{Q}_t + \eta \cdot \frac{\partial \mathcal{L}_1}{\partial Q_t}, & Q_t \leftarrow Q_t - \hat{Q}_t. \end{cases} \quad (10)$$

where $\nu \in [0, 1]$ is the momentum parameter and $\eta > 0$ is the learning rate. According to previous work [37], the default values of ν and η are usually set to be 0.8 and 0.05, respectively. \hat{U}_i , \hat{V}_j , and \hat{Q}_t are the temporary variables which used to update the momentum. When the parameters U_i , V_j , and Q_t converge or the number of iterations reaches its maximum, we can obtain the feature vectors U_i , V_j and Q_t .

3.3. Developer Prediction

Next, we present the developer prediction based on the feature vectors U_i and V_j calculated in Equation (10). Due to the linear inner product might limit the expressiveness of MF [13,31], here, we project the feature vectors U_i and V_j into a non-linear space upon the architecture of deep neural network [38].

As shown in Figure 2 right subfigure, for developer u_i and task v_j , the input feature vectors are U_i and V_j respectively. Suppose $W_U^{(t)}$ and $W_V^{(t)}$ are the t_{th} ($t=1,2,\dots,N$) layer weighting matrices for U_i and V_j respectively. The developer u_i and task v_j are finally mapped to a low-dimensional vector space as:

$$\begin{cases} U'_i = f^{(N)}(\dots f^{(2)}(W_U^{(2)} f^{(1)}(W_U^{(1)} U_i)) \dots), \\ V'_j = f^{(N)}(\dots f^{(2)}(W_V^{(2)} f^{(1)}(W_V^{(1)} V_j)) \dots), \end{cases} \quad (11)$$

where $f^{(t)}(\cdot)$ denotes the activation function, we implement $f^{(t)}(\cdot)$ with *ReLU*, which is formalized as $f^{(t)}(x) = \max(0, x)$. In this paper, we have two multi-layer networks to transform the representations of u and v respectively. Based on the output vectors U'_i and V'_j in Equation (11), we can calculate the predicted value as follows:

$$\hat{y}_{ij}(u_i, v_j | \Theta) = U_i'^T V'_j \quad (12)$$

where $\Theta = \{W_U^{(t)}, W_V^{(t)}\}_{t=1}^N$ are the parameters. To learn Θ , we opt for the pairwise BPR loss [39], which has been widely used in training recommender systems [40]. It assumes that the observed

interactions, which indicate more user preferences, should be assigned higher prediction values than unobserved ones. The objective function is as:

$$\mathcal{L}_2 = \sum_{(i,j,j') \in R} -\ln \sigma(\hat{y}_{ij} - \hat{y}_{ij'}) + \lambda \|\Theta\|_2^2, \quad (13)$$

where $(i, j) \in R^+$ denotes the set of observed interaction relationships, $(i, j') \in R^-$ denotes the set of unobserved interactions, and $\sigma(\cdot)$ is the sigmoid function. $\lambda \|\Theta\|_2^2$ is the regularizer used to prevent the overfitting. Similar to Equations (9) and (10), we can calculate the partial derivative of \mathcal{L}_2 as well as other model parameters in Equation (13).

3.4. Fast Model Update

How to update the model is a critical problem. Previous update approaches based on large matrix factorization [4,29,41] usually update the whole system offline and periodically. This approach is referred to as fullUpdate. However, on developer collaboration platforms, a lot of explicit and implicit relationships are produced on a daily basis. Frequent full updates are expensive, especially in recommendation scenarios that involve large-scale matrices and multiple relationships. They limit the update frequency and, consequently, lower the timeliness of the recommendation results.

However, updating in time may help address the cold start issue (as will be evaluated and demonstrated in Section 4.2.4), i.e., making recommendations for new developers and new tasks. In practice, the information about new developers and new tasks are very limited, and timely update of every new piece of information about new developers and new tasks can improve their recommendation results significantly. Therefore, in SoftRec, for new developers and new tasks, the latent feature vectors U_i , V_j and Q_t are updated online. Upon the arrival of a new developer or a new task, SoftRec first updates R , C , and S , i.e., $R \leftarrow R \cup \{R_{ij}\}$, $C \leftarrow C \cup \{C_{ti}\}$, and $S \leftarrow S \cup \{S_{jl}\}$, and then performs $U_i \leftarrow U_i - \hat{U}_i$, $V_j \leftarrow V_j - \hat{V}_j$, and $Q_t \leftarrow Q_t - \hat{Q}_t$ with Equations (9) and (10). On the other hand, for old developers and old tasks, because their historical information plays the dominant role during the recommendation process, minor updates may hardly affect the recommendation results within a short period of time. Therefore, SoftRec updates old developers and old tasks' latent feature vectors U_i , V_j , and Q_t with offline fullUpdate only.

Based on the above principles, we propose a novel fast update approach for new developers and new tasks, called fastUpdate. Take the update of feature vector U_i as an example, its pseudo code is presented in Algorithm 1. For developer u_i , let $\phi(u_i) = \bar{\Theta}(R_{i*})$ denote u_i 's known interaction set in R_{i*} , $\bar{\Theta}(\cdot)$ denote the set of known entities. The probability of performing the operation $R \leftarrow R \cup \{R_{ij}\}$ is defined as follows:

$$p(\text{update}|R_{ij}) = \varphi^{|\phi(u_i)|}, \quad (14)$$

where the parameter $\varphi \in [0, 1]$ denotes the decay factor. The probability of update will gradually decrease as $|\phi(u_i)|$ increases. Then, based on Equations (9) and (10), the update of U_i is described in Algorithm 1.

Algorithm 1: fastUpdate for U_i

Input: $\phi(u_i)$, θ , φ , ν , η , R_{ij} , \mathcal{L}_1 , R , $U_i = [U_{i1}, U_{i2}, \dots, U_{id}]$;

Output: U_i ;

```

1 if  $p(\text{update}|R_{ij}) > \theta$  then
2    $R \leftarrow R \cup \{R_{ij}\}$ ; initialize  $\hat{U}_i$ ;
3   for  $s$  from 1 to Iter do
4     for  $j$  from 1 to  $d$  do
5        $\hat{U}_{ij} \leftarrow \nu \cdot \hat{U}_{ij} + \eta \cdot \{\frac{\partial \mathcal{L}_1}{\partial U_i}\}_j$ ;  $U_{ij} \leftarrow U_{ij} - \hat{U}_{ij}$ ;
6 return  $U_i$ ;
```

where θ is the empiric constant and $Iter$ is the number of iterations before an early stop is applied, its value defaults to 100.

Similar to the update of U_i in Algorithm 1, Q_t and V_j can be updated as well. We can see that the time complexity of fastUpdate approach is $O(\{|\bar{\Theta}(C_{*i})| + |\bar{\Theta}(S_{j*})| + |\bar{\Theta}(R_{i*})|\} \cdot d \cdot Iter)$, while the time complexity of the traditional fullUpdate approach that updates the entire model is $O(\{|\bar{\Theta}(C)| + |\bar{\Theta}(S)| + |\bar{\Theta}(R)|\} \cdot d \cdot Iter)$. This way, the proposed fastUpdate approach as a supplement to the fullUpdate can address the cold start issue by making online recommendations for new developers and new tasks (as shown in Section 4.2.4). After updating U_i and V_j , we can calculate the value of \hat{y}_{ij} based on Equations (11) and (12).

4. Experiment and Discussion

We evaluate the proposed approach on two real-world datasets, one is collected from the GitHub projects, another is collected from Neusoft Corporation. We aim to answer the following research questions:

- RQ1 How does SoftRec perform compared with state-of-the-art CF-based recommendation methods?
- RQ2 How does SoftRec perform when tackling different data sparsity?
- RQ3 How do developer–developer collaboration relationships and task–task association relationships (i.e., λ_C and λ_S) affect SoftRec?
- RQ4 How does SoftRec perform when tackling model update and does it help solve the cold start issue?
- RQ5 Can SoftRec have practical value and be recognized by enterprise users in practice?

4.1. Experimental Settings

4.1.1. Data Collection and Preprocessing

To collect data for our experiments, we crawl five popular GitHub projects with GitHub API (<https://developer.github.com/v3/>), including symfony/symfony, akka/akka, elasticsearch, netty/netty and ipython/ipython, and the collected data is from March 2015 to November 2018. These five projects are highly popular and have 2760 watches, 39,340 stars and 16,860 forks on average on GitHub. In addition, we select five popular commercial projects from the Neusoft Corporation's GitLab platform, including Workflow, DI, DataViz, ACAP and APM, and the collected data is from January 2017 to June 2019. In GitHub projects, we recommend code reviewers for the given pull-requests, and in GitLab projects we recommend assignees for the given issues.

Next, we filter out the tasks (i.e., pull-requests or issues) with less than two different reviewers or assignees according to previous works [4]. The statistics of preprocessed data are shown in Table 1, and the initial data density of the interaction matrix are $\rho_1 = 3.57\%$ and $\rho_2 = 6.01\%$, respectively.

Table 1. Statistics of the collected data.

Datasets	Projects	Tasks#	Developers#	Interactions#
GitHub	symfony	1023	1331	23,970
	akka	1312	207	12,464
	elasticsearch	2751	251	26,272
	netty	819	307	5364
	ipython	3424	421	19,158
GitLab	Workflow	6210	104	1035
	DI	2193	150	19,737
	DataViz	2107	108	12,642
	ACAP	2311	125	25,421
	APM	2811	103	11,244

4.1.2. Relationship Mapping

We calculate the three types of relationships based on the definitions in Section 3.1. Table 2 shows the mappings of interactive objects and interactive actions in the collected datasets. Take the GitHub dataset as an example. The collaboration relationship defined in Equation (1) are calculated based on the interactive object set $\mathcal{O}=\{\text{issue_comments, commit_comments, pull_request_comments}\}$ in each GitHub project. The developers' interactive action set \mathcal{A} performed on \mathcal{O} is $\{\text{comment, commit, reaction}\}$, where the interactive action reaction represents a series of emojis (<https://developer.github.com/v3/reactions/>).

Table 2. Relationship mapping

Datasets	Relationships	Objects	Actions
GitHub	developers' collaboration relationship	issue_comments, commit_comments, pull_request_comments	comment, reaction, commit
	developer–task interaction relationship	pull_requests, pull_request_comments	comment, reaction
	tasks' similarity relationship	pull_requests, pull_request_comments	-
GitLab	developers' collaboration relationship	merge_requests, issues, issue_comments	comment, reaction, commit
	developer–task interaction relationship	issues, issue_comments, commit_comments	comment, reaction
	tasks' similarity relationship	issues, issue_comments	-

4.1.3. Approaches for Comparison

In this experiment, we compare SoftRec with five state-of-the-art approaches:

- **PR-CF** [4]: a typical CF-based hybrid approach that generates the latent factor models based on the developer–task explicit interaction matrix, and then combines the latent factor models with the tasks' neighborhoods to capture the similarity between developers and tasks.
- **IR+CN** [20]: This approach recommends developers based on their social relationships. By mining historical comments, it constructs a weighted graph called comment network (CN) to model developers' social relationships.
- **DMF** [38]: a typical matrix factorization model with neural network architecture to learn a common low dimensional space for the representations of users and items.
- **NFM** [31]: a typical deep learning model that unifies the strengths of factorization machines and deep neural networks for sparse rating modelling.

4.1.4. Parameter Settings

Similar to [17], we use a grid search for parameters: the decay factor $\omega \in [0, 1]$ in Equations (2) and (5) are searched in $\{0.6, 0.7, 0.8, 0.9\}$, and α and β in Equations (4) and (6) are searched in $\{0.4, 0.5, 0.6, 0.7\}$. The vector size d is tuned among $\{16, 32, 48, 64\}$. λ_C and λ_S in Equation (8) are searched in $\{1, 5, 10, 20\}$. λ_U , λ_V and λ_Q in Equation (8) are searched in $\{0.001, 0.005, 0.01, 0.1, 0.5\}$. θ and φ in Algorithm 1 are searched in $\{0.0001, 0.0005, 0.001\}$ and $\{0.4, 0.5, 0.6, 0.7\}$ respectively. λ in Equation (13) is tuned among $\{0.01, 0.03, 0.05\}$.

4.1.5. Scenario Description

To test the performance of SoftRec in tackling the interaction sparsity (i.e., RQ2) and model update (i.e., RQ4), we design two test scenarios: (1) interaction sparse scenario; (2) new developer cold start scenario. To simulate the interaction sparse scenario, we design different ratios of data density by removing the known elements from developer–task explicit interaction matrix R_{exp} according to their time slice. To simulate cold-start scenarios with new developers, (1) we first change the developers to new developers by removing their related information from the initial database tables; (2) second, we recover the removed information into the corresponding database tables and recalculate various relationships according to their time slice (i.e., recover one day’s information at a time), in this process, the fastUpdate is performed online, but the fullUpdate is performed periodically (i.e., performed after seven day’s recoveries).

4.1.6. Performance Evaluation

We adopt three representative performance metrics: precision@k, recall@k [4], and ndcg@k [42] for performance evaluation, as shown in Equations (15)–(17). Where precision@k and recall@k are widely-used metrics that don’t consider the ranking position, ndcg@k is a popular ranking-based metrics in recent years, which considers the ranking position, where a higher position is assigned with a higher score. By default, we set $k = 5$, and randomly divided the datasets into 10 groups, where 80% as the training set and 20% as the testing set, and the evaluation was conducted based on cross-validation. We use the task’s actual developers as the ground-truth results and the reported performance was averaged over 20 repetitions.

$$precision@k = \frac{|ActualDevelopers \cap RecommendedDevelopers|}{|RecommendedDevelopers|}, \quad (15)$$

$$recall@k = \frac{|ActualDevelopers \cap RecommendedDevelopers|}{|ActualDevelopers|}, \quad (16)$$

$$ndcg@k = Z_k \sum_{i=1}^k \frac{2^{r_i} - 1}{\log_2(i + 1)}, \quad (17)$$

where Z_k is the normalizer to ensure that the perfect ranking has a value of 1. r_i is the relevance of developer u at position i , if u exists in the test, we set $r_i = 1$, otherwise $r_i = 0$.

4.2. Experimental Results and Discussions

4.2.1. Overall Performance Comparison (RQ1)

Table 3 compares the overall performance of the five approaches. We have the following observations. First, IR+CN achieves poor performance on both datasets. This indicates that considering only the explicit social relationships does not suffice to capture the potential interactions between developers and tasks. Compared to IR+CN, PR-CF generally achieves better improvements in most cases. This indicates the importance of considering the latent factor similarity between tasks. The reason we chose PR-CF and IR+CN as the comparison algorithm is that both PR-CF and IR+CN are typical developer recommendation approaches and similar to our approach. Where PR-CF utilizes the implicit similarity relationship between tasks to improve the recommendation accuracy, while IR+CN utilizes the social relationship between developers to improve the recommendation accuracy. Second, compared to DMF, NFM achieves a better accuracy in most cases. The reason might be that NFM combines the linearity of a factorization machine in modelling second-order feature interactions and the non-linearity of neural network in modelling higher-order feature interactions, which is more expressive than DMF. Third, among all approaches in comparison, SoftRec achieves the highest accuracy across all different cases. For example, for the GitHub dataset, the precision, recall and ndcg

obtained by SoftRec are 0.4673, 0.7633 and 0.4727, respectively. It outperforms other state-of-the-art approaches by an average of 23.26%, 6.69% and 34.56% in precision, recall and ndcg, respectively. For the GitLab dataset, SoftRec's precision, recall and ndcg are 0.4929, 0.7701 and 0.5109, respectively, outperforming other state-of-the-art approaches by an average of 29.75%, 5.33% and 23.56%, respectively. The reasons might be that (1) SoftRec can fully explore the explicit and implicit multi-relationships which is helpful to improve recommendation accuracy, unlike NMF and DMF which employ the explicit interactions only; (2) SoftRec can project the feature vectors U_i and V_j into a non-linear space upon the architecture of deep neural network, which helps capture the non-linear and complex feature of real-world data, unlike PR-CF and IR+CN model feature interactions into a linear space only.

Table 3. Comparison of different approaches ($k = 5$).

Methods	GitHub Dataset			GitLab Dataset		
	Precision	Recall	ndcg	Precision	Recall	ndcg
PR-CF	0.3791	0.7154	0.3513	0.3799	0.7299	0.4135
IR+CN	0.3248	0.6411	0.3361	0.3145	0.6377	0.3789
DMF	0.3399	0.6998	0.3205	0.3615	0.6819	0.4106
NFM	0.3413	0.6918	0.3327	0.3369	0.7311	0.3704
SoftRec	0.4673	0.7633	0.4727	0.4929	0.7701	0.5109

4.2.2. Performance in Tackling Data Sparsity (RQ2)

We test SoftRec in scenarios 1) with different sparsity of developer–task interactions (described in Section 4.1.5 Scenario Description). Figure 3a,b show the compared results of precision, recall and ndcg for the GitHub and GitLab datasets, where ρ_1 and ρ_2 denote the initial data density of interaction matrices in GitHub and GitLab datasets, respectively. We have the following observations.

When the data density decreases from ρ_1 and ρ_2 to 0, the precision, recall and ndcg values of all approaches decrease quickly. However, SoftRec significantly and consistently outperforms than other approaches. For example, for GitHub dataset, when the data density is $0.6 * \rho_1$, the values of precision, recall and ndcg of SoftRec are 43.55%, 47.17% and 28.71% higher than other compared approaches on average. For GitLab dataset, when the data sparsity is $0.6 * \rho_2$, SoftRec's precision, recall and ndcg are 25.29%, 22.66%, 24.78% higher than other compared approaches on average. Besides, as the data density gradually decreases, the performance of SoftRec decreases more slowly than other compared methods. This phenomenon indicates that SoftRec performs better when tackling data sparsity. Another discussion is that why SoftRec performs better than other compared approaches in GitHub dataset than in GitLab dataset. The reason might be that the GitHub dataset is more sparse than the GitLab dataset (e.g., in GitHub dataset $\rho_1 = 3.75\%$, in GitLab dataset $\rho_2 = 6.01\%$), and the compared approaches do not fully mine various implicit relationships, making its performance worse when the interactions become more and more sparse. However, SoftRec takes advantage of various implicit relationships, and this effectively alleviates the sparsity of interaction and improves the recommendation accuracy.

4.2.3. Effects of the Multi-Relationship (RQ3)

In SoftRec, the parameters λ_C and λ_S determined the effects of developer–developer collaboration relationship and task–task association relationship on the recommendation results. Now let us discuss how these relationships affect SoftRec and how to determine their values. From Equations (8)–(10) we can see that SoftRec uses λ_C and λ_S to balance the collaboration relationships and the association relationships with the developer–task interaction relationships. When $\lambda_C = 0$, it is equivalent to completely ignoring the influence of collaboration relationships. As λ_C increases, it means that the collaboration relationships are leveraged to make recommendations with a higher priority. We also

demonstrate it in our experiments, as shown in Figure 4a,b, as λ_C and λ_S increases from 0.01 to 5 and 0.01 to 10, SoftRec's precision, recall and ndcg values increase, while when λ_C and λ_S exceed 5 and 10 respectively, these values decrease gradually.

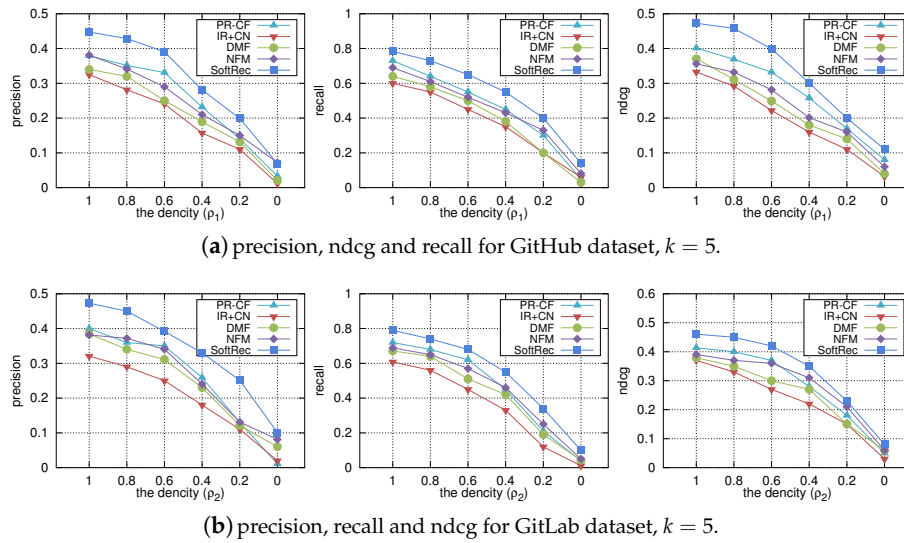


Figure 3. Comparison of the recommendation accuracy (precision, recall and ndcg) with different density of developer–task interaction matrix (e.g., the density ranges from $1 * \rho_1$ to $0 * \rho_1$ in subfigure (a)) in GitHub and GitLab datasets.

Furthermore, to compare the performance of SoftRec with and without the multi-relationship, we set $\lambda_C = \lambda_S = 0$. Table 4 shows the compared results, where SoftRec' denotes the SoftRec without the multi-relationship. For example, for GitHub dataset, the values of precision, recall and ndcg of SoftRec with multi-relationship are 50.11%, 32.93% and 59.10% higher than the values of SoftRec without multi-relationship.

This phenomenon presented in Figure 4 and Table 4 shows that by leveraging of the developer–developer collaboration relationships and task–task association relationships can help improve the recommendation accuracy effectively. Because the optimal values for those parameters are domain-specific, we set the parameters through trial-and-error in the experiments.

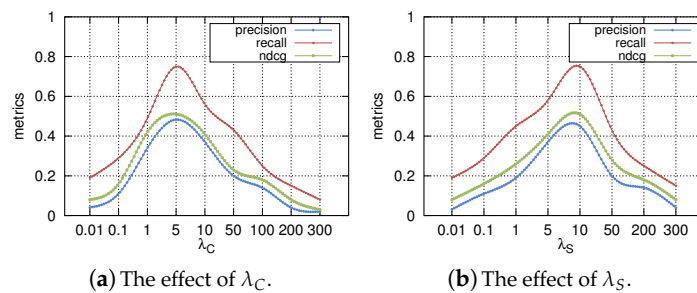


Figure 4. Take GitHub dataset as an example, analyze the effect of developer–developer collaboration relationship and task–task association relationship on the performance by adjusting the parameters λ_C and λ_S , where $k = 5$.

Table 4. Comparison of the SoftRec with and without the multi-relationship fusion ($k = 5$).

Methods	GitHub Dataset			GitLab Dataset		
	Precision	Recall	ndcg	Precision	Recall	ndcg
SoftRec	0.4673	0.7633	0.4727	0.4929	0.7701	0.5109
SoftRec'	0.3113	0.5742	0.2971	0.2935	0.5707	0.3419

4.2.4. Performance in Tackling Model Update (RQ4)

Now let us discuss whether SoftRec can alleviate the cold start issue and its performance in tackling update. The advantage of SoftRec is that its fastUpdate approach as a supplement to the fullUpdate can accommodate new developers and new tasks promptly, thus effectively alleviating the cold start issue. To demonstrate the performance of fastUpdate, we take the new developers cold start (described in Section 4.1.5 Scenario Description) as an example, as shown in Table 5, with the proposed fastUpdate, for GitHub dataset, SoftRec's average precision, recall and ndcg values increase by about 4.25% to 4.1%, 2.88% respectively; for the GitLab dataset, its average precision, recall and ndcg values increase by about 4.17% to 7.14%, 3.70% respectively. It shows that SoftRec's fastUpdate can properly address the cold start issue by making online recommendations for new developers.

Table 5. Compare the performance of model update by analyze the fullUpdate and fastUpdate approaches.

Datasets	k	precision		recall		ndcg	
		fullUpdate	fastUpdate	fullUpdate	fastUpdate	fullUpdate	fastUpdate
github	1	0.6991	0.7103	0.2801	0.2912	0.2911	0.3133
	2	0.6622	0.6819	0.3106	0.3262	0.283	0.3123
	3	0.5523	0.5756	0.5344	0.5445	0.4391	0.4423
	4	0.4632	0.4911	0.6732	0.6819	0.5057	0.5212
	5	0.4301	0.4673	0.6903	0.7433	0.4851	0.4727
gitlab	1	0.7439	0.7811	0.251	0.281	0.6784	0.6903
	2	0.6312	0.6624	0.3643	0.3878	0.6701	0.6832
	3	0.5782	0.5901	0.5687	0.5911	0.5689	0.5811
	4	0.4613	0.4781	0.6433	0.6792	0.391	0.4032
	5	0.4697	0.4829	0.7013	0.770	0.4581	0.4909

4.3. User Study and System Design (RQ5)

To further demonstrate the effectiveness of SoftRec, we conduct a user study at Neusoft Corporation. The scenario is that in Neusoft's project and process management, hundreds of issues (e.g., defects or new features) are created by developers (e.g., project managers, software testers or tech supports) every day. To improve the cooperation efficiency, these issues need to be assigned to the appropriate developers as soon as possible. Current manual assignments are usually time consuming and inefficient and automatic assignment mechanism is urgently required.

In this study, we choose five typical projects in Neusoft Corporation, including Workflow, DI, DataViz, ACAP and APM. For each project, we filter 100 open state issues and try to recommend developers for them based on the SoftRec framework and evaluate the results by means of online questionnaires.

4.3.1. Effectiveness of Our Approach

The results of the questionnaires are shown in Figure 5, We can see that for all surveyed projects, SoftRec's recommendation results are significantly superior to other approaches. For SoftRec, The average values of precision, recall and ndcg in these study projects Workflow, DI, DataViz, ACAP and APM are 0.4721, 0.7868, 0.5102, respectively. Compared with other four approaches, the values of precision, recall and ndcg of SoftRec are 26.33%, 12.64% and 15.63% higher on average, respectively.

Because we recommend developers within the scope of each project, the developer–developer collaboration relationships, developer–task interaction relationships, and task–task association relationships are very close. This leads to higher recommendation accuracies compared with those achieved on the sparse datasets.

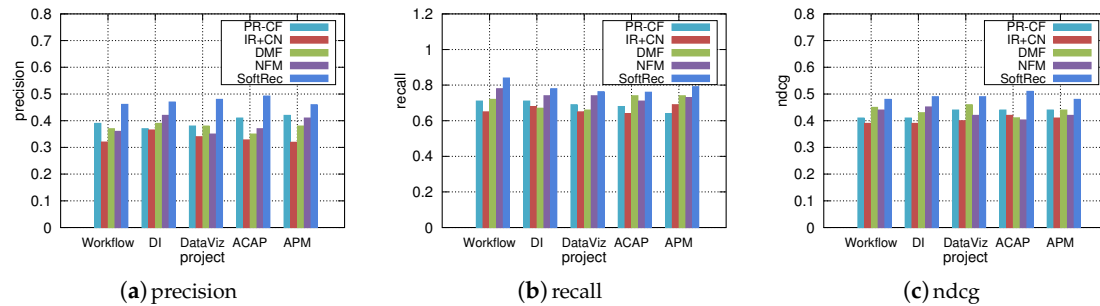


Figure 5. User study analysis (in Neusoft Corporation, $k = 5$).

4.3.2. User Interview

We interviewed 25 developers about their opinions of the recommendations made by SoftRec versus those made by the conventional expertise-based recommendation approaches tested in our experiments. The participants of the interview consisted of 20 male developers and 5 female developers, with an average work experience of 5 years. After inspecting the recommended tasks, 16 of them felt highly confident that they would accept the recommended tasks, 7 of them felt confident about the recommended tasks, and the other 2 may or may not have accepted the recommended tasks. The interview results indicate that more than 92% of the interviewees are satisfied with the recommendations. These interview results empirically show that SoftRec can improve their collaboration efficiency by making accurate developer recommendations.

4.3.3. System Design

To facilitate the practice of DevRec in the real environment, we have designed a feasible technical framework at Neusoft Corporation. As shown in Figure 6, its functions include real-time data collection, distributed data preprocessing, distributed data indexing, distributed data storage, multi-relationship computing, model training and prediction, etc.

In this technical framework, we exploit the open source tool chains as much as possible. For example, we collect data through open source technologies, such as Crawler (<https://www.npmjs.com/package/crawler>), GitHub API (<https://developer.github.com/>), Logstash and Beats (<https://www.elastic.co/cn/downloads/beats>), etc. To optimize the data transmission efficiency, we integrate a message queue middleware (Apache Kafka (<http://kafka.apache.org/>)) into the framework. The data preprocessing (e.g., data extraction, data transformation, data cleaning) are implemented based on Logstash. The data indexing and storage are implemented based on Elasticsearch (<https://github.com/elastic/elasticsearch>). The multi-relationship computing and model training are based on Spark (<http://spark.apache.org/>) and Tensorflow (<https://www.tensorflow.org/>). The advantages of our technical framework are as follows:

- From the viewpoint of software development, we design the framework based on a series of open source tool chains as much as possible, which follows the idea of open source software and can shield the underlying complexity and improve the development efficiency.
- From the viewpoint of system availability, the framework supports distributed data storage and parallel computing for developer recommendation, which makes it have better performance in big data environment and provide a valuable technical reference for system practice. As far as we know, this technical framework has been adopted by Neusoft Corporation and will be integrated

into their commercial product (<https://platform.neusoft.com/allproducts/acap>) as part of their DevOps tool chains.

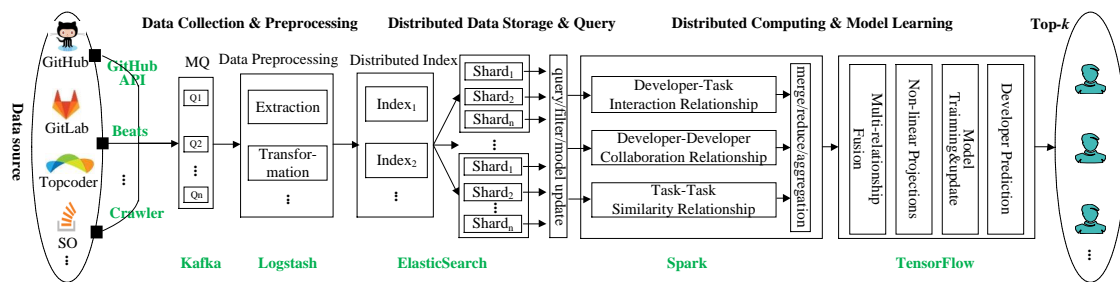


Figure 6. Overall technical framework design for DevRec at Neusoft Corporation.

5. Threats to Validity

First, our experiments are performed on five popular projects in GitHub dataset and five large commercial projects in Neusoft Corporation. We cannot claim that the same results would be achieved with other projects or other periods of time. Moreover, the results of the application of SoftRec to other platforms, e.g., Bitbucket, StackOverflow, TopCoder, might not be exactly the same. As future work, we plan to extend our evaluation on more universal open-source and industrial projects.

Second, SoftRec is a relationship-aware developer recommendation approach aiming to recommend suitable developers based on the idea of collaborative filtering. We use the actual developers of tasks as the ground truth and do not consider their expertise, reputation and workloads, etc. Thus, there is a risk that the recommended developers might not be the best ones of all. To mitigate this threat, we plan to extend our framework by measuring the developers' abilities (e.g., skills, expertise, reputation, contribution and workload).

6. Conclusions and Future Work

In this paper, we proposed SoftRec, a novel multi-relationship fused approach for developer recommendation. In SoftRec, three types of implicit relationships are utilized, including the collaboration relationships between developers, the interaction relationships between developers and tasks, the association relationships between tasks. Furthermore, a novel fast model update approach was proposed to address the cold start issue. To our best knowledge, this is the first attempt to systematically integrate the developers' collaboration relationships and tasks' association relationships into developer recommendation.

Form the viewpoint of theory innovation, we propose to utilize joint matrix factorization to project developers' and tasks' features and their relationships into a low dimensional latent vector space. It leverages not only the common developer latent vectors in both the interaction matrix and the developer collaboration matrix, but also the common task latent vectors in both the interaction matrix and the task association matrix, and we refine the latent vectors based on deep neural network. It effectively solves the issues of interactive sparseness and cold start in traditional collaborative filtering. From the viewpoint of practice, we conduct extensive experiments on two real-world datasets and we also conduct a user study in a well-known software company. The results demonstrate the high performance of SoftRec. Furthermore, we design a feasible technical framework and exploit the open source tool chains as much as possible, which helps to facilitate the practice of SoftRec in real environment.

In the future, we will extend SoftRec in three ways. The first one is to employ deep learning to solve the multiple implicit relationship fusion for developer recommendation. The second one is to further solve the boundary of the fast update for SoftRec through the theoretical or experimental analysis. The third one is to further investigate the usefulness of SoftRec and consider integrating the developers'

abilities (e.g., skills, expertise, reputation, contribution and workload) into SoftRec. Moreover, we plan to provide a set of developer recommendation tools that can be used in real environments. We hope to provide free plugins or service APIs for websites such as GitHub, StackOverflow and Topcoder, etc.

Author Contributions: Conceptualization, X.X.; Data curation, X.X.; Investigation, X.X.; Writing—original draft, X.X.; Writing—review and editing, B.W. and X.Y.; Resources, X.Y. All authors have read and agreed to the published version of the manuscript.

Funding: The work is partially supported by the National Natural Science Foundation of China (Nos. U1736104, 61991404, 61532021), Ten Thousand Talent Program (No. ZX20200035), Liaoning Distinguished Professor (No. XLYC1902057).

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

References

1. Gousios, G.; Storey, M.A.; Bacchelli, A. Work practices and challenges in pull-based development: the contributor's perspective. In Proceedings of the IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, 14–22 May 2016; pp. 285–296.
2. Hannebauer, C.; Patalas, M.; Stünkelt, S.; Gruhn, V. Automatically recommending code reviewers based on their expertise: An empirical comparison. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 3–7 September 2016; pp. 99–110.
3. Xia, X.; Lo, D.; Wang, X.; Yang, X. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, 29 September–1 October 2015; pp. 261–270.
4. Xia, Z.; Sun, H.; Jiang, J.; Wang, X.; Liu, X. A hybrid approach to code reviewer recommendation with collaborative filtering. In Proceedings of the IEEE International Workshop on Software Mining (SoftwareMining), Urbana, IL, USA, 3 November 2017; pp. 24–31.
5. Liu, Z.; Xia, X.; Treude, C.; Lo, D.; Li, S. Automatic Generation of Pull Request Descriptions. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; IEEE: New York, NY, USA, 2019; pp. 176–188.
6. Thongtanunam, P.; Tantithamthavorn, C.; Kula, R.G.; Yoshida, N.; Iida, H.; Matsumoto, K.I. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In Proceedings of the IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2–6 March 2015; pp. 141–150.
7. Alami, A.; Cohn, M.L.; Wasowski, A. Why does code review work for open source software communities? In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; IEEE: New York, NY, USA, 2019; pp. 1073–1083.
8. Yan, J.; Sun, H.; Wang, X.; Liu, X.; Song, X. Profiling developer expertise across software communities with heterogeneous information network analysis. In Proceedings of the Tenth Asia-Pacific Symposium on Internetwork, Beijing, China, 16 September 2018; pp. 1–9.
9. Ye, L.; Sun, H.; Wang, X.; Wang, J. Personalized teammate recommendation for crowdsourced software developers. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; pp. 808–813.
10. MirsaeediFarahani, S. Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution. Ph.D. Thesis, Concordia University, Montreal, QC, Canada, 2019.
11. Li, R.; Lin, H.; Shi, Y.; Wang, H. SocialST: Social Liveness and Trust Enhancement Based Social Recommendation. In Proceedings of the 2019 IEEE International Conference on Web Services (ICWS), Milan, Italy, 8–13 July 2019; pp. 139–145.
12. Ye, B.; Wang, Y. Crowdrec: Trust-aware worker recommendation in crowdsourcing environments. In Proceedings of the 2016 IEEE international conference on web services (ICWS), San Francisco, CA, USA, 2–7 July 2016; pp. 1–8.
13. 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.

14. Kabbur, S.; Ning, X.; Karypis, G. Fism: factored item similarity models for top-n recommender systems. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 11–13 August 2013; pp. 659–667.
15. Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; pp. 426–434.
16. Xin, X.; He, X.; Zhang, Y.; Zhang, Y.; Jose, J. Relational collaborative filtering: Modeling multiple item relations for recommendation. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Paris, France, 21–25 July 2019; pp. 125–134.
17. Wang, X.; He, X.; Wang, M.; Feng, F.; Chua, T.S. Neural Graph Collaborative Filtering. *arXiv* **2019**, arXiv:1905.08108.
18. Ma, H.; Yang, H.; Lyu, M.R.; King, I. Sorec: social recommendation using probabilistic matrix factorization. In Proceedings of the 17th ACM Conference on Information and Knowledge Management, Napa Valley, CA, USA, 26–30 October 2008; pp. 931–940.
19. Jiang, J.; Yang, Y.; He, J.; Blanc, X.; Zhang, L. Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development. *Inf. Softw. Technol.* **2017**, *84*, 48–62. [[CrossRef](#)]
20. Yu, Y.; Wang, H.; Yin, G.; Wang, T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Inf. Softw. Technol.* **2016**, *74*, 204–218. [[CrossRef](#)]
21. Ahmed, T.; Bosu, A.; Iqbal, A.; Rahimi, S. SentiCR: A customized sentiment analysis tool for code review interactions. In Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, USA, 30 October–3 November 2017; IEEE: New York, NY, USA, 2017; pp. 106–111.
22. Bosu, A.; Carver, J.C. How do social interaction networks influence peer impressions formation? a case study. In *IFIP International Conference on Open Source Systems*; Springer: Berlin, Germany, 2014; pp. 31–40.
23. Bosu, A.; Carver, J.C. Impact of peer code review on peer impression formation: A survey. In Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, 10–11 October 2013; pp. 133–142.
24. Ouni, A.; Kula, R.G.; Inoue, K. Search-based peer reviewers recommendation in modern code review. In Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, USA, 2–7 October 2016; pp. 367–377.
25. Liao, Z.; Wu, Z.; Li, Y.; Zhang, Y.; Fan, X.; Wu, J. Core-reviewer recommendation based on Pull Request topic model and collaborator social network. *Soft Comput.* **2020**, *24*, 5683–5693. [[CrossRef](#)]
26. Shin, D. How do users interact with algorithm recommender systems? The interaction of users, algorithms, and performance. *Comput. Hum. Behav.* **2020**, *26*, 106344. [[CrossRef](#)]
27. Shin, D.; Zhong, B.; Biocca, F.A. Beyond user experience: What constitutes algorithmic experiences? *Int. J. Inf. Manag.* **2020**, *52*, 102061. [[CrossRef](#)]
28. Lin, P.; Song, Q.; Wu, Y. Fact checking in knowledge graphs with ontological subgraph patterns. *Data Sci. Eng.* **2018**, *3*, 341–358. [[CrossRef](#)]
29. Zhang, S.; Yao, L.; Sun, A. Deep learning based recommender system: A survey and new perspectives. *arXiv* **2017**, arXiv:1707.07435.
30. Xie, F.; Chen, L.; Ye, Y.; Zheng, Z.; Lin, X. Factorization machine based service recommendation on heterogeneous information networks. In Proceedings of the 2018 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 2–7 July 2018; pp. 115–122.
31. He, X.; Chua, T.S. Neural factorization machines for sparse predictive analytics. In Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval, Tokyo, Japan, 7–11 August 2017; pp. 355–364.
32. Wang, S.; Li, C.; Zhao, K.; Chen, H. Context-aware recommendations with random partition factorization machines. *Data Sci. Eng.* **2017**, *2*, 125–135. [[CrossRef](#)]
33. Sadowski, C.; Söderberg, E.; Church, L.; Sipko, M.; Bacchelli, A. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering (ICSE): Software Engineering in Practice, Gothenburg, Sweden, 27 May–3 June 2018; pp. 181–190.

34. He, X.; He, Z.; Song, J.; Liu, Z.; Jiang, Y.G.; Chua, T.S. NAIS: Neural attentive item similarity model for recommendation. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 2354–2366. [[CrossRef](#)]
35. Le, Q.; Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the International Conference on Machine Learning, Beijing, China, 22–24 June 2014; pp. 1188–1196.
36. Mnih, A.; Salakhutdinov, R.R. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2008; pp. 1257–1264.
37. Srinivasan, V.; Sankar, A.R.; Balasubramanian, V. ADINE: an adaptive momentum method for stochastic gradient descent. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, Goa, India, 11–13 January 2018; pp. 249–256.
38. Xue, H.J.; Dai, X.; Zhang, J.; Huang, S.; Chen, J. Deep Matrix Factorization Models for Recommender Systems. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17), Melbourne, Australia, 19–25 August 2017; pp. 3203–3209.
39. Rendle, S.; Freudenthaler, C.; Gantner, Z.; Schmidt-Thieme, L. BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, 18–21 June 2009; pp. 452–461.
40. He, X.; He, Z.; Du, X.; Chua, T.S. Adversarial Personalized Ranking for Recommendation. In Proceedings of the SIGIR'18 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, Ann Arbor, MI, USA, 8–12 July 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 355–364. [[CrossRef](#)]
41. Huang, Y.; Cui, B.; Jiang, J.; Hong, K.; Zhang, W.; Xie, Y. Real-time video recommendation exploration. In Proceedings of the ACM International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 35–46.
42. He, X.; Chen, T.; Kan, M.Y.; Chen, X. Trirank: Review-aware explainable recommendation by modeling aspects. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 19–23 October 2015; pp. 1661–1670.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).