Endowing third-party libraries recommender systems with explicit user feedback mechanisms

Riccardo Rubei, Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio, Phuong T. Nguyen
Department of Information Engineering, Computer Science and Mathematics, Università degli studi dell'Aquila, Italy
{riccardo.rubei, claudio.disipio}@graduate.univaq.it, {juri.dirocco, davide.diruscio, phuong.nguyen}@univaq.it

2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-6654-3786-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SANER53432.2022.00099

***Abstract*—During their daily routine, developers often deal with a plethora of resources, attempting to search for relevant artifacts that can be added to the project under development. This kind of information overload may render developers overwhelmed, thus undermining their productivity and efficiency. Recommender systems are an effective means of easing such a burden, providing relevant items for the current programming contexts, e.g., third-party libraries (TPLs), API calls, or code snippets. By focusing on TPLs, there has been no work to allow for the integration of tailored feedback mechanisms with which users can conveniently accept or discard libraries. In this paper, we propose an approach to handle explicit user feedback, including positive, negative, and additive. Thus, further than accepting or discarding the recommended TPLs, users can also endorse libraries that, in their opinion, are relevant for the current context, even though they are not included in the provided recommendations. As a proof of concept, we demonstrate how user feedback generated by the proposed mechanism can change the outcome of a real TPLs recommender system. The results show that our proposed approach helps the considered system retrieve relevant items, under different configurations.**

***Index Terms*—Recommender systems; Third-party recommendation; User feedback**

I. Introduction

During the software development life cycle, developers usually use external tools and artifacts to assist their programming tasks. Third-party libraries (TPLs) are software components developed to provide functionalities to solve a wide range of tasks. The purpose of TPLs is to help developers relieve the burden of creating desired functionalities from scratch. Though being very useful and widely used, TLPs are rather heterogeneous, and thus developers may need to spend a lot of effort to find the right ones. In this respect, the problem of recommending TLPs has attracted attention from both academia and industry [13], [12], [5]. A crucial aspect that could potentially increase the overall precision is leveraging developers' perception of specific libraries. Various studies [15], [3] succeed in improving existing recommender systems by considering both implicit and explicit feedback [20]. Interestingly, no work has been done to address the issue of empowering TPLs recommender systems with user feedback.

In this work, we pesent a novel approach to exploiting user feedback to enhance TPLs recommendations. We address the motivating question: "*Is it possible to increase the relevance of the items provided by a TPL recommender system by augmenting it with user feedback?*" taking into consideration three types of feedback, i.e., positive, negative, and additive. In the scope of this paper, we built an initial prototype based on the Learning to Rank [11] (LTR) model to rearrange the recommended list of items produced by CrossRec [12], a well-founded TPLs recommender system, according to the given user feedback. Furthermore, we also propose a method to enable additive feedback, i.e., a user can endorse a library that was not in the original recommendations.

The preliminary study conducted on CrossRec shows that the approach obtains encouraging results in equipping the system with the proposed feedback mechanism. Moreover, our findings suggest that the envisioned technique can be incorporated into different kinds of recommender systems. We published the replication package to facilitate future research.[1]

II. Motivation and Background

A. Explanatory example

To highlight our contribution, we take a motivating example in Fig. 1. Given a ranked list of TPLs provided by a recommender system and that is supposed to be relevant for the current development context, a user can express three different types of feedback, namely *positive*, *negative* or *additive*. For instance, the system provides as a first item *junit* by relying on its internal mechanism. However, the user may prefer another one, for instance, *mockito*, according to the requested functionalities. To this end, a possible mechanism to *upvote* or *downvote* a library is represented in Fig. 1, where *mockito* has been preferred over *junit* that is, instead, marked with a "dislike." Furthermore, the user can suggest a new item that does not belong to the original recommended list. In the example, the user wants to add *jackson* since it provides similar functionalities of *json* that was already suggested. Such an envisioned feedback mechanism aims to increase the performance of the used recommender systems, e.g., in terms of the relevance of recommended items to the current context.

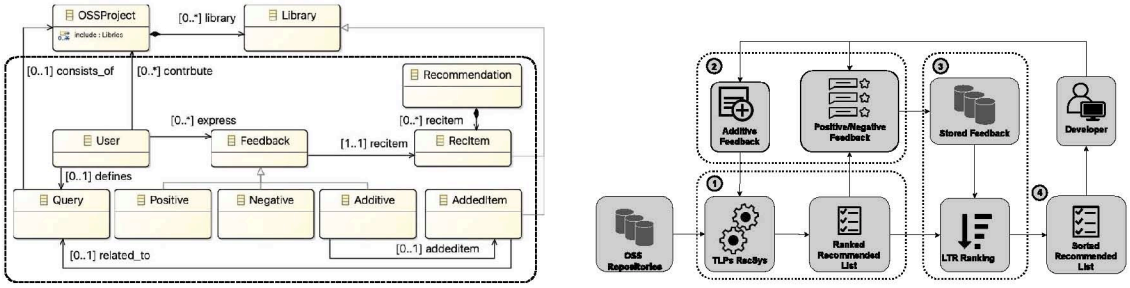
Fig. 1: Explanatory example.

B. Background

Third-party library recommendation. TPLs recommender systems provide developers with third-party libraries that are considered to be relevant to the projects under development [5], [8], [12], [13]. Due to space limitations, this

[1]https://github.com/SANER2022-UserFeedback/RecSys

978-1-6654-3786-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SANER53432.2022.00099



(a) Conceptual model (b) System architecture

Fig. 2: Conceptual model and system architecture of the user feedback mechanism for TPLs recommendations.

section recalls only one of them, i.e., CROSSREC [12], as it is considered as among state-of-the-art TPLs recommenders. CROSSREC works based on the assumption that *"if projects share some third-party libraries, then they will probably share additional libraries."* In particular, CROSSREC encodes the relationships among OSS projects in a graph and utilizes a collaborative-filtering technique [10] to retrieve TPLs. The system returns a ranked list of libraries collaboratively mined from the most similar projects given an input project.

To the best of our knowledge, though several TPLs recommender systems exist, no work has been done to enable them to exploit user feedback to increase the relevance of the recommended items. As detailed in the next section, this paper proposes a novel approach to equip recommender systems with the management of user feedback.

Learning to Rank (LTR). It is a supervised learning technique widely used to cope with the ranking task [2]. Given a query q and a set of documents $D(d_1, .., d_n)$, LTR ranks them according to their relevance with respect to the query. To this end, the feature vectors are extracted from the initial dataset and are used to feed the model using the stochastic gradient descent method. It eventually retrieves a ranked list of documents and the corresponding relevance score.

In this paper, we utilize a particular LTR model, namely the Weighted Approximate-Rank Pairwise (WARP) model [19]. The rationale behind this choice is that it works better in sorting top-K elements of a recommended list. In particular, the WARP model is employed to sort the list of TPLs as recommended by CROSSREC to align it with user preferences that have been previously expressed in terms of feedback.

III. PROPOSED METHODOLOGY

We conceptualize an approach to endow recommender systems with the management of user feedback. Even though the final goal is to define such a mechanism generically, in this paper, we focus on the problem of recommending TPLs and supporting user feedback for them. Figure 2(a) represents a conceptual model covering the concepts of interest. In particular, we focus on open-source projects that depend on a set of TPLs (see the concepts *OSSProject* and *Library*). A developer that is working on a given software project can ask the available recommender system to provide her with a list of further TPLs that might be added to the project under development (see the concept *User* that *defines* a *Query* consisting of the project under development and gets back a *Recommendation* element consisting of different *Items*). Users might want to express their *Feedback* for each returned item to increase the relevance for future similar requests. Users can like or dislike recommended items (see the concept *Positive* and *Negative*, respectively) or can even suggest additional items that were not included in the original recommended list (see the concept *Additive* feedback). It is worth noting that the entities encapsulated in the dashed frame represent the agnostic part of the methodology, meaning that the same concepts can be adapted to any kind of recommendations, e.g., API calls or snippets. However, thoroughly assessing the genericity of the proposed approach is planned as future work.

The architecture implementing the conceptual model previously described is shown in Fig. 2(b) and consists of the following components:

▷ *TPLs RecSys:* it is the recommender system, which is able to recommend third-party libraries for the project under development. Given the current development context, the system generates a ranked list of recommended libraries ①.

▷ *User feedback:* the user can express three different types of explicit feedback for each item in the recommended list, i.e., positive, negative and additive ②. Including or removing a TLP from the project under development is mapped to positive or negative feedback, respectively. Additive feedback consists of injecting endorsed libraries into the training data of *TPLs RecSys*. Thus, during the next iteration of the whole process, the system will take into account the new addition. It is worth mentioning that only one injection may not be sufficient to promote the new library for the next recommendation requests;

▷ *LTR Ranking:* LTR is applied to the ranked list provided by the adopted recommender system by considering previously stored user feedback ③. We make use of the *LightFM* Python library [6] which implements the WARP model. Using the feedback collected in the previous phase, a set of feature vectors v are extracted to train the LTR model according to the following format: v=$(1, 0, 0, 1, 0.....1, 0)$, where 1 is a positive rate and 0 is a negative one expressed for each TPL. To feed the ranking model, such vectors need to be transformed into a *scipy* coordinate matrix,[2] i.e., each pair user-library has a rating. Concerning the hyperparameters, we set the learning rate to 0.02 and the number of epochs to 70, given that higher

[2]https://bit.ly/30v9sVD

values seem to have negligible effects on the performance. As final output, this component produces the rearranged list of TPLs by considering positive and negative feedback;
▷ *Sorted Recommended List:* The sorted list of recommended items is eventually presented to the user ④ who decides to either *(i)* accept it or *(ii)* express additional feedback to trigger another recommendation session.

IV. Proof of concept

This section presents the results of the conducted preliminary evaluation to assess the feasibility and the effectiveness of the proposed approach. To this aim we considered CrossRec [12] as TPL recommender system. The rationale behind such a selection is that it is considered as among state-of-the-art library recommender systems, e.g., it achieves a better performance compared to three well-founded baselines, namely LibRec [16], LibFinder [13] and LipCUP [14]. To simulate binary feedback, we introduce a rating mechanism that assigns a vote to each library suggested by CrossRec. Furthermore, we modify the original graph by adding projects and libraries to measure the impact of additive feedback.

A. Experimental parameters

– $\mathcal{L}$ is the set of libraries on which the user expresses feedback, i.e., positive, negative or additive;
– p and p' are the original position and position after the ranking phase respectively of $l \in \mathcal{L}$;
– R_l is the set of user feedback for a specific library $l \in \mathcal{L}$ expressed as a binary rate r, i.e., 0 and 1 for negative and positive feedback, respectively;
– $REC(l)$ is the recommended list provided by CrossRec that includes a specific library $l \in \mathcal{L}$;
– c is the cut-off value (the number of recommended libraries);
– N is the number of positive feedback given to library l;
– K is the set of new OSS projects that includes intentionally seeded libraries to simulate additive feedback.

B. Metrics

To measure the capability of the methodology to upvote/-downvote a given library, we use *hit-rank*$_l$@N [20] defined as: *hit-rank*$_l$@N=$\frac{count_{r \in R_l}(|\Delta p|>0)}{|R_l|}$, where r is the feedback expressed on the library $l \in R_l$ by the user and $|\Delta p|$ is the delta between the original position p of the item and its new position p' after rearranging. This metric represents how many times the considered library $l \in L$ should be upvoted/downvoted to modify its starting position p. For positive feedback, we consider a match if the $p' > p$. Similarly, $p' < p$ means that l was successfully downvoted using negative feedback.

We measure the effectiveness of additive feedback by *hit-count*$_l$@K computed as: *hit-count*$_l$@K=$count_{l \in REC(l)}$, where K is the number of projects that include the new library $l \in \mathcal{L}$ added by the user, and $REC(l)$ represents the recommendations of CrossRec. In other words, *hit-count*$_l$@K measures how many times $REC(l)$ includes the added library l according to the number of additive feedback K.

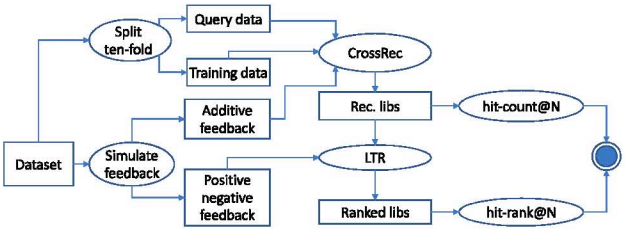


Fig. 3: The evaluation process.

C. Dataset

We use the original CrossRec replication package and dataset made available online.[3] The dataset consists of 1,200 projects with 13,498 libraries. Due to the lack of real user feedback, we mimic explicit feedback by counting the frequencies of the examined libraries, i.e., if a given project p *includes* a certain library l, the value for the pair (p, l) is 1 otherwise it is set to 0. Such an occurrence is mapped to a *positive feedback*, and the rate of a library is its frequency on the whole dataset, i.e., the number of projects that invoke it.

TABLE I: Features of the examined TPLs.

	Library					
Value	avro	guice	mockito	guava	slf4j	junit
Freq.	26	74	213	306	473	969
$\mathbf{f}_{c=10}$	0.007	0.010	0.232	0.294	0.415	0.754
$\mathbf{f}_{c=20}$	0.025	0.043	0.359	0.398	0.478	0.755

We chose six among the libraries that are representative in terms of popularity, i.e., from the least to the most popular libraries, and counted the occurrence of the recommended libraries. Table I describes the libraries, their frequency, and percentage of occurrences in the results, considering two CrossRec cut-off values, i.e., 10, and 20.

D. Methodology

We set up different configurations as follows. Concerning the cut-off values, c=10 and c=20 are chosen to measure the effectiveness of the LTR model when different sizes are considered. We empirically vary the number of positive and negative feedback by varying N, and K, i.e., N=$\{20, 40, 100, 200, 600, 1000\}$, K=$\{0, 20, 50, 100, 200\}$, to assess the contribution of additive feedback. Fig. 3 depicts the evaluation process consisting of three phases, i.e., data preparation, recommendations, and evaluation. Starting from the dataset (see Section IV-C), positive, negative, and additive feedback is simulated according to c, N, and K. We applied the ten-fold cross-validation technique, and the query data is generated from testing projects by removing half of their libraries. To inject an additive feedback for a library l_k with respect to the project context p=$\{l_1, ..., l_n\}$, a new row $\{p, l_1, ..., l_n, l_k\}$ is added to the training data. Given such query data, the system retrieves the list of recommended items. Then, given the first c items, positive and negative feedback, LTR ranks the result provided by CrossRec.

E. Results

We analyze the results using two research questions.

[3]https://github.com/crossminer/CrossRec

TABLE II: Results obtained with positive feedback.

	Lib.	p	hr_{20}	hr_{40}	hr_{100}	hr_{200}	hr_{600}	hr_{1000}
c=10	avro	7	**0.56**	0.44	0.44	0.56	0.44	0.44
	guice	8	0.46	0.53	**0.62**	0.38	0.46	0.61
	mockito	6	0.49	0.52	0.48	0.54	0.47	**0.55**
	guava	6	0.48	0.48	0.48	**0.52**	0.47	0.50
	slf4j	3	0.64	**0.71**	0.61	0.70	0.64	0.70
	junit	3	0.89	0.90	0.89	**0.92**	0.84	0.90
c=20	avro	13	0.20	0.30	0.20	0.27	**0.33**	0.27
	guice	12	0.37	0.37	**0.41**	0.41	0.35	0.33
	mockito	10	0.44	0.44	0.45	0.42	**0.48**	0.48
	guava	10	0.44	0.45	0.46	**0.48**	0.46	0.46
	slf4j	8	**0.57**	0.48	0.53	0.48	0.44	0.54
	junit	6	0.87	0.79	**0.90**	0.88	0.86	0.88

▷ **RQ$_1$**: *How does the positive and negative feedback contribute to rearranging* CROSSREC*'s recommendations?* We measure the impact of positive and negative feedback in upvoting and downvoting a certain library, through a series of experiments using the configurations described in Section IV-A. Table II shows the results obtained by the proposed methodology when positive feedback is considered.[4] Given a library l, we compute the average initial position p by setting the rating of all users to 0 as described by the p column, i.e., l has not been rated by any user yet. Starting from this initial state, we increase the number of *positive* ratings according to N. In this way, we resemble the situation where an unrated library grows in popularity by exploiting explicit feedback.

The results show that the mechanism is able to promote a library. Such an improvement is more evident for the most popular libraries, i.e., *junit*, *slf4j-api*. In fact, the number of positive rates needed to upvote a library is not the same for all libraries, i.e., the LTR module reduces the popularity impact. For instance, assigning 100 positive rates to *guice* improves its ranking 0.62 of the time with c=10. Meanwhile, *mockito* reaches the maximum value at $hit\text{-}rank_l@1000$, though it is more popular with the frequency of 213.

While the mechanism works better in upvoting popular libraries, i.e., $hit\text{-}rank_l@N$ reaches around 0.90 of effectiveness for *junit*, the most popular library, it suffers from degradation of performance when a less popular item is considered, e.g., the system improves the ranking for *avro* only 0.56 of the time with $c = 10$. The performance is negatively affected when increasing c from 10 to 20 for almost the libraries, except *junit*. This can be reasoned by referring to Table I. Since *junit* is the most popular item, it appears in the CROSSREC top rank items in almost all the tests.

We conduct a similar experiment to measure the impact of a *negative* feedback to downgrading a popular library. The different configurations and the corresponding results are shown in Table III. In this setting, the p column represents the initial position of the library when its rate is equal to $1,200$, i.e., every user upvotes the library. In such a way, we are able to simulate negative feedback by decreasing the votes using the same threshold defined for the previous experiment.

Downvoting a popular library is more difficult as the corresponding $hit\text{-}rank_l@N$ scores are lower for all libraries, compared to the results in Table II. Generally speaking, the negative feedback successfully downvotes the target library only 0.30 of the attempts. The better results are reached with *avro*, i.e., $hit\text{-}rank_l@40$ =0.33 with c=10 and $hit\text{-}rank_l@600$ =0.46 with c=20. The findings suggest that it is easier to downvote a less popular library than a most used one. This claim is confirmed by the results obtained with *junit* since $hit\text{-}rank_l@1000$ is extremely small for all the configurations, i.e., the maximum value is 0.07. This is expected since users usually follow the *wisdom of the crowd* during their development activities, i.e., they tend to select libraries used by the majority of the community [7].

[4]For the sake of presentation, hr_N and hc_K stand for $hit\text{-}rank_l@N$ and $hit\text{-}count_l@K$, respectively.

TABLE III: Results obtained with negative feedback.

	Lib.	p	hr_{20}	hr_{40}	hr_{100}	hr_{200}	hr_{600}	hr_{1000}
c=10	avro	5	0.11	**0.33**	0.11	0.11	0.22	0.11
	guice	6	**0.15**	0.15	0.15	0.00	0.15	0.00
	mockito	4	**0.25**	0.23	0.22	0.21	0.18	0.24
	guava	5	0.19	0.16	0.19	**0.24**	0.19	0.19
	slf4j	3	0.16	0.15	0.14	**0.17**	0.13	0.16
	junit	2	0.08	0.04	**0.19**	0.04	0.08	0.07
c=20	avro	12	0.27	0.20	0.27	0.13	**0.43**	0.20
	guice	12	**0.22**	0.18	0.18	0.22	0.14	0.18
	mockito	7	**0.29**	0.28	0.28	0.28	0.28	0.27
	guava	8	0.21	0.26	0.21	0.22	0.24	0.27
	slf4j	7	**0.29**	0.24	0.24	0.28	0.23	0.23
	junit	2	0.07	0.07	0.11	0.06	**0.14**	0.05

TABLE IV: Results obtained with additive feedback.

	Lib.	hc_0	hc_{20}	hc_{40}	hc_{100}	hc_{200}	hc_{600}	hc_{1000}
c=10	avro	9	86	172	368	511	711	**758**
	guice	15	122	221	377	514	695	**740**
	mockito	322	449	520	640	732	836	**864**
	guava	382	471	541	631	691	759	**777**
	slf4j	536	564	569	609	625	659	**672**
	junit	**995**	986	993	985	985	983	992
c=20	avro	31	146	264	470	615	784	**822**
	guice	55	200	341	492	619	768	**798**
	mockito	491	618	680	782	838	930	**944**
	guava	518	594	668	733	781	822	**829**
	slf4j	616	637	637	669	693	704	**726**
	junit	997	989	**998**	990	989	988	997

Answer to RQ$_1$. Either positive or negative feedback has a clear impact on the recommendation results. The effectiveness strongly depends on the popularity of the considered libraries.

▷ **RQ$_2$**: *How does the additive feedback impact on the original recommended list?* We study the influence of additive feedback by simulating the addition of new projects relying on libraries recommended by users. Such additions induce the modification of the CROSSREC's original matrix and allow us to measure the number of additions needed to promote the user suggested library to make it appear in the recommended list. The results obtained for additive feedback are shown in Table IV. Column hc_0 contains data that is obtained without operating any change to the CROSSREC's original matrix. Subsequent columns instead, contain values that are obtained after adding fake projects including the library of the corresponding row. For instance, the cells *[avro,*hc_{20}*]* contain how many times out of 1,200 queries, the library *avro* has been recommended after having added 20 artificial projects containing it. The table shows a dominant trend: the more projects are seeded, the more often the endorsed library gets recommended. While this is more visible with less frequent libraries, e.g., *avro*, it is less evident with popular libraries, such as *junit*, for instance, with

c=20, from hc_0 to hc_{1000}, there is only a small fluctuation in the number of recommendations. Altogether, we see that similar to results with positive feedback, the additive ones have a remarkable impact on the less popular libraries, i.e., they increase their probability of being suggested by CROSSREC.

Answer to RQ$_2$. Introducing additive feedback helps increase popularity of the endorsed libraries. The more frequent a library is (e.g., *junit*), the less the additive feedback impacts on the number of times it is recommended.

V. RELATED WORK

The BRAID framework [20] has been built on top of an existing API recommender system to allow for the specification of a user-based query. Furthermore, it combines LTR and Active Learning techniques to adapt the original recommendations according to the user query. Elixir [4] supports user feedback expressed on items' explanations using pairwise learning. The list of pair ratings-explanations is encoded to user-specific latent vectors that have been used to improve a well-founded recommender system based on random walk with restart technique. Wang *et al.* [18] proposed Active Code Search to incorporate explicit user feedback. Given a recommended list, the user can iteratively express feedback on the proposed items to rearrange the final rank. Then, a *refined engine* is employed to embody the collected feedback using NLP techniques. Similarly, WebAPIRec [17] employs a personalized ranking model to recommend an ordered list of web APIs given the project profile. The model is fed with historical data of APIs usages to retrieve the ranked list of APIs. Recently, several studies employed federated learning to collect user feedback by preserving sensitive data [1], [9].

Ouni *et al.* [13] make use of multi-objective algorithm by relying on library usage history. Given a library, the system recommends new items by maximizing the semantic similarity. LibCUP [14] uses a clustering approach to identifying co-usage patterns and suggests similar TPLs. Similarly, LibD [8] provides libraries to mobile apps using a clustering technique.

To the best of our knowledge, the possibility of providing positive, negative, and additive feedback in the context of TPLs recommender systems is still underinvestigated. Our approach allows users to provide feedback on recommended TPLs. The experiments performed on CROSSREC are encouraging, and we plan to consider other TPLs recommender systems.

VI. CONCLUSION AND FUTURE WORK

The introduction of user feedback in recommender systems has brought considerable benefits in different domains. This paper conceived a novel approach to incorporate user feedback into TPLs recommender systems. The proposed methodology supports binary and additive feedback by relying on a well-founded LTR model to rearrange the delivered items. The preliminary evaluation of an existing TPLs recommender system using simulated feedback confirms the approach's feasibility. For future work, we plan to integrate the technique in one of the existing IDEs e.g., Eclipse or VSCode, before conducting a user study to collect real feedback. Moreover, further than improving the adopted LTR model by tuning the hyperparameters, we will investigate the application of other ranking models e.g., vector space and probabilistic ones. Lastly, we will extend the proposed methodology to different TPLs recommender systems.

ACKNOWLEDGMENT

The research described in this paper has been partially supported by the AIDOaRT Project, which has received funding from the European Union's H2020-ECSEL-2020, Federal Ministry of Education, Science and Research, Grant Agreement n° 101007350.

REFERENCES

[1] V. W. Anelli, Y. Deldjoo, T. Di Noia, A. Ferrara, and F. Narducci, "Federank: User controlled feedback with federated recommender systems," in *Advances in Information Retrieval*, 03 2021, pp. 32–47.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," ser. ICML. Bonn, Germany: ACM Press, 2005, pp. 89–96.

[3] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *ACM Comput. Surv.*, vol. 44, p. 1, 01 2012.

[4] A. Ghazimatin, S. Pramanik, R. S. Roy, and G. Weikum, "ELIXIR: learning from user feedback on explanations to improve recommender models," ser. WWW. ACM / IW3C2, 2021, pp. 3850–3860. [Online]. Available: https://doi.org/10.1145/3442381.3449848

[5] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, "Diversified third-party library prediction for mobile app development," *IEEE Trans. on Software Engineering*, pp. 1–1, 2020.

[6] M. Kula, "Metadata embeddings for user and item cold-start recommendations," in *Procs. the 2nd CBRecSys, co-located with RecSys 2015, Vienna, Austria, September 16-20, 2015.*, vol. 1448. CEUR-WS.org, 2015, pp. 14–21.

[7] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *EMSE.*, vol. 23, no. 1, p. 384–417, feb 2018.

[8] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "Libd: Scalable and precise third-party library detection in android markets," ser. ICSE, 2017, pp. 335–346.

[9] G. Lin, F. Liang, W. Pan, and Z. Ming, "Fedrec: Federated recommendation with explicit feedback," *IEEE Intelligent Systems*, vol. 36, no. 5, pp. 21–30, 2021.

[10] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, Jan. 2003.

[11] T.-Y. Liu, *Learning to Rank for Information Retrieval.* Springer, 2011.

[12] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, "CrossRec: Supporting software developers by recommending third-party libraries," *J. Systems and Software*, vol. 161, p. 110460, 2020.

[13] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue, "Search-based software library recommendation using multi-objective optimization," *IST Journal*, vol. 83, pp. 55–75, 2017.

[14] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *J. Systems and Software*, vol. 145, pp. 164 – 179, 2018.

[15] G. Salton and C. Buckley, "Improving retrieval performance by relevance feedback," *J. the American Society for Information Science*, vol. 41, no. 4, pp. 288–297, 1990.

[16] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *2013 20th Work. Conf. on Reverse Eng. (WCRE)*, 2013, pp. 182–191.

[17] F. Thung, R. J. Oentaryo, D. Lo, and Y. Tian, "Webapirec: Recommending web apis to software projects via personalized ranking," *IEEE TETCI*, vol. 1, no. 3, pp. 145–156, 2017.

[18] S. Wang, D. Lo, and L. Jiang, "Active code search: Incorporating user feedback to improve code search relevance," ser. ASE, 2014, p. 677–682. [Online]. Available: https://doi.org/10.1145/2642937.2642947

[19] J. Weston, H. Yee, and R. J. Weiss, "Learning to rank recommendations with the k-order statistic loss," ser. RecSys. ACM, 2013, p. 245–248.

[20] Y. Zhou, H. Jin, X. Yang, T. Chen, K. Narasimhan, and H. C. Gall, "BRAID: an API recommender supporting implicit user feedback," in *ESEC/FSE*, 2021, pp. 1510–1514. [Online]. Available: https://doi.org/10.1145/3468264.3473111