ELIXIR: Learning from User Feedback on Explanations to Improve Recommender Models

Azin Ghazimatin
Max Planck Institute for Informatics
Germany
aghazima@mpi-inf.mpg.de

Soumajit Pramanik*
IIT Bhilai
India
soumajit@iitbhilai.ac.in

Rishiraj Saha Roy
Max Planck Institute for Informatics
Germany
rishiraj@mpi-inf.mpg.de

Gerhard Weikum
Max Planck Institute for Informatics
Germany
weikum@mpi-inf.mpg.de

ABSTRACT

System-provided explanations for recommendations are an important component towards transparent and trustworthy AI. In state-of-the-art research, this is a one-way signal, though, to improve user acceptance. In this paper, we turn the role of explanations around and investigate how they can contribute to enhancing the quality of generated recommendations themselves. We devise a human-in-the-loop framework, called Elixir, where user feedback on explanations is leveraged for pairwise learning of user preferences. Elixir leverages feedback on pairs of recommendations and explanations to learn user-specific latent preference vectors, overcoming sparseness by label propagation with item-similarity-based neighborhoods. Our framework is instantiated using generalized graph recommendation via Random Walk with Restart. Insightful experiments with a real user study show significant improvements in movie and book recommendations over item-level feedback.

CCS CONCEPTS

• **Information systems → Recommender systems**.

ACM Reference Format:
Azin Ghazimatin, Soumajit Pramanik, Rishiraj Saha Roy, and Gerhard Weikum. 2021. ELIXIR: Learning from User Feedback on Explanations to Improve Recommender Models. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3449848

1 INTRODUCTION

Motivation. Generating explanations for recommendations like movies, music or news, by online service providers, has gained high attention in academic and industrial research [3, 48, 51]. While early methods generated simple explanations such as "you liked x and other users also liked x and y ...", modern approaches have become much more sophisticated (see, e.g., [49]). A key goal is to enhance user trust by transparent and scrutable recommendations, so that users understand how the recommended item relates to their prior online behavior (search, clicks, likes, ratings, etc.) (see, e.g., [4, 13]). Moreover, it is desirable that explanations are *causal* and *actionable*, meaning that i) they refer only to the user's own action history and not to potentially privacy-sensitive cues about other users (see, e.g., [12]) and ii) the user can act on the explanation items by giving confirmation or refutation signals that affect future recommendations. Critique-enabled recommendation models [8, 18, 23, 27] pursue these goals, but are restricted to user feedback on the recommended items and associated content (e.g., text snippets from item reviews), disregarding the *explanation items*. In this paper, we extend this regime of actionable user feedback to the explanations themselves, by obtaining additional cues from users in a lightweight manner and incorporating them into an active learning framework to improve future recommendations.

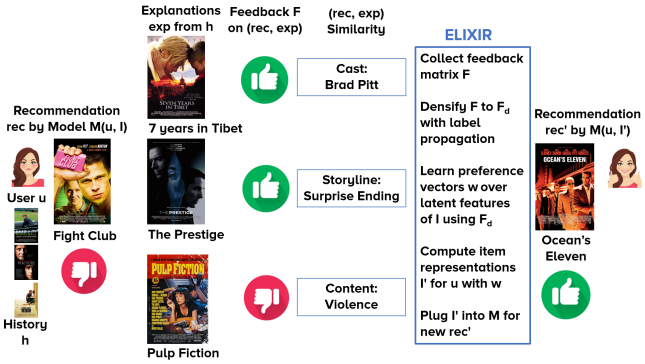


Figure 1: Example illustrating the intuition for Elixir.

Example. Fig. 1 shows an illustrative scenario. Alice receives a recommendation for the movie *Fight Club* (*rec*) based on her online history and factors like item-item similarities. This is accompanied by an explanation referring to three items, all previously liked by Alice and being similar, by some aspects, to *rec*. We have *exp*1: *Seven Years in Tibet*, *exp*2: *The Prestige*, and *exp*3: *Pulp Fiction*. The system generated these three items for explanation because:

- *exp*1 features the actor Brad Pitt who also stars in *rec*,
- *exp*2 has a surprise ending, similar to *rec*,
- *exp*3 contains violent content, like *rec*.

*This work was done while the author was at the MPI for Informatics.

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.
WWW '21, April 19–23, 2021, Ljubljana, Slovenia
© 2021 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.
ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449848

Now suppose that Alice loves Brad Pitt and surprise endings but hates disturbing violence (she likes *Pulp Fiction* for other reasons like its star cast and dark comedy, that dominated her opinion, despite the violence). When receiving *rec* with the above explanation, Alice could give different kinds of feedback. The established way is to simply dislike *rec*, as a signal from which future recommendations can learn. However, this would completely miss the opportunity of learning from how Alice views the three explanation items. Alice could also dislike the explanation as whole, but this would give only a coarse signal, too, and would appear conflicting with the fact that she previously liked $exp1, exp2$ and $exp3$, confusing the recommender system. The best feedback would be if Alice could inform the system that she likes Brad Pitt and surprise endings but dislikes violence, for example, by checking item properties or filling in a form or questionnaire. However, this would be a tedious effort that few users would engage in. Also, the system would have to come up with a very fine-grained feature space of properties, way beyond the usual categories of, say, movie genres.

Problem Statement. The goal in this paper is to leverage user feedback on explanations. This entails two major problems:

- *Feedback:* How can we elicit user feedback on properties of both recommended and explanation items in a lightweight manner, without burdening the user with too much effort?
- *Actionability:* If we are able to obtain such refined feedback, how can the recommender system learn from it to improve its future outputs for the user?

Approach. This paper presents Elixir (**E**fficient **L**earning from **I**tem-level e**X**planations **I**n **R**ecommenders), a novel framework, for leveraging explanations to improve future recommendations.

We address the *Feedback* problem by asking users for a binary like/dislike signal about the similarity of an explanation item *exp* to the recommended item *rec*. This can be thought of as assessing the quality of the *item pair* $\langle rec, exp \rangle$, but it is important that one of these is an explanation item that was faithfully produced by the recommender system specifically to justify *rec*. Our experiments compare this choice against asking for assessments on the similarity of the recommendation with the least relevant items in the user's profile, which turns out to be inferior. As we consider only causal explanations that refer to the user's own history of actions, the user should be reasonably familiar with item *exp*. This kind of feedback is more refined than simply disliking the entire recommendation. The feedback is optional and can be given for any subset of the possible $\langle rec, exp \rangle$ pairs. Most importantly, the user is not burdened with identifying relevant properties of items, to further explain her feedback to the system. So Elixir builds on very lightweight and proactive user feedback.

We address the *Actionability* problem by extending state-of-the-art recommender models with a user-feedback matrix that encodes the like/dislike signals on $\langle rec, exp \rangle$ pairs. Since this matrix is inevitably sparse, Elixir densifies this input by means of label propagation on item neighborhoods [46]. To avoid the huge cost of computing similarities for all item pairs, we employ locality sensitive hashing (LSH) to find the closest items to every $\langle rec, exp \rangle$ tuple, thereby making Elixir efficient and tractable.

The core of our method is the learning of user-specific latent vectors that capture user preferences, by combining the densified feedback matrix and a prior item-item similarity matrix through regularized optimization that models the signals in the feedback matrix as soft constraints. The latent vectors would reflect that Alice loves Brad Pitt and sophisticated plots but dislikes violent movies – without referring to these properties, all by means of learning latent representations from lightweight feedback. The per-user vectors are plugged into the actual recommender system to learn user-specific item representations for future recommendations.

We instantiate the Elixir framework in a popular family of graph-based recommenders: Random Walk with Restart (RWR) (see, e.g., [31]), from which explanations can be generated in a faithful and causal manner (see, e.g., [12, 41]).

Contributions. The salient contributions of this paper are:

- Elixir is, to the best of our knowledge, the first framework that leverages user feedback on explanation items, thus making explanations actionable, whereas prior works only tapped into feedback on recommendation items.
- Elixir elicits lightweight user feedback to learn user-specific item representations, and incorporates these into the recommender model, instantiated with the RWR methodology.
- We report experiments with data from a longitudinal user study in two domains: (i) movies, and (ii) books. The results demonstrate the viability of Elixir and its substantial gains in recommendation quality over item-level feedback.
- The user study data and all code for this project are released as research resources at https://www.mpi-inf.mpg.de/impact/learning-from-feedback-on-explanations and https://github.com/azinmatin/elixir/.

2 THE ELIXIR FRAMEWORK

In this section, we describe the components of Elixir and present its instantiation for a state-of-the-art recommender, RecWalk [31], that is based on random walks with restart (RWR). Table 1 summarizes concepts and notation discussed in this section.

2.1 Feedback collection

Elixir enables recommender systems to combine individual item ratings with feedback on pairs of recommendation and explanation items. The set of item-level signals H_u refers to the set of individual items that appear in the interaction history of user u. Denoting the universe of all items by $I = \{v_1, v_2, ...v_{|I|}\}$, we have $H_u \subseteq I$ and usually $|H_u| \ll |I|$.

While most recommenders train a user model solely based on H_u, Elixir exploits signals from user feedback on item pairs from recommendations and their explanations. We denote this pair-level feedback by the matrix $F_u \in \{-1, 0, +1\}^{|I|\times|I|}$. The matrix entry $F_u(v_i, v_j)$ represents user u's feedback on recommendation item v_i and explanation item v_j. To collect such feedback, we ask users whether they like/dislike the similarity between items v_i and v_j. We encode a user's liking, no feedback, and disliking with +1, 0 and −1, respectively.

2.2 Feedback densification

As the set of all item-pairs is very large, we expect matrix F_u to be extremely sparse. To mitigate this sparseness, we use the label propagation (LP) algorithm [46] on a graph where nodes are *pairs*

Notation	Concept
u	A single user
v	A single item
$\vec{v}$	Latent vector for item v
d	Number of latent dimensions
U	Set of all users
I	Set of all items
H_u	Interaction history of user u
$F_u(v_i, v_j)$	Feedback on item pair (v_i, v_j) by user u
v_{ij}	Pseudo-item for item pair (v_i, v_j) in LP
W	Affinity matrix for LP
F_u^d	Densified feedback matrix for user u after LP
m	Number of non-zero elements in F_u^d
$sim(\vec{v}_i, \vec{v}_j)$	Similarity of item pair (v_i, v_j)
$\vec{w}_u$	Preference vector for user u
$g(\vec{v}, \vec{w}_u)$	Latent representation of item v for user u
γ	Regularization coefficient for learning w_u
G	Graph on which the RWR recommender is run
N	Set of nodes in graph
E	Set of edges in graph
T_N, T_E	Types of nodes and edges in graph
θ (θ_N, θ_E)	Mapping functions (of graph nodes and edges) to types
A	Matrix of user-item interactions for RWR
S	Item-item similarity matrix for RWR
$\vec{e_u}$	One-hot vector for user u
α	Restart probability in RWR
β	Probability of walking over interaction edges in RWR
rec	Recommendation item
exp	Explanation item

Table 1: Notation for salient concepts in Elixir.

of items, and edges represent the *similarity between item-pairs*. To define such a graph, we introduce the concept of a pseudo-item v_{ij} for each labeled pair of items (v_i, v_j) (that models an item that is like a mixture of the properties of the two items in the pair) where $F_u(v_i, v_j) \neq 0$, with $\otimes$ denoting the element-wise product:

$$\vec{v_{ij}} = (\vec{v}_i \otimes \vec{v}_j)^{\frac{1}{2}} \tag{1}$$

where $\vec{v}_i$ is the feature vector for item v_i. Depending upon the recommender model, item features are either learned by the model [15, 20, 22] or are available from additional sources [10, 31]. More generally, we assume that item features can be cast into a latent representation. Eq. 1 defines the pseudo-item $\vec{v_{ij}}$ as the element-wise geometric mean of $\vec{v}_i$ and $\vec{v}_j$. Compared to the arithmetic mean, the geometric mean is more appropriate for boosting similarities and dampening dissimilarities (higher and lower values in the original vectors, respectively).

The original LP algorithm requires an affinity matrix W which encodes item-item similarities. In our problem, the labels we propagate are feedback points on (v_i, v_j) pairs: so each pseudo-item v_{ij} represents a pair of items and the affinity matrix thus contains pair-pair similarities. This makes W huge ($W \in \mathbb{R}^{|I|^2 \times |I|^2}$) and prohibits full materialization.

Our approach rather is to materialize and populate merely a small subset of W by considering only the k nearest neighbors of each pseudo-item v_{ij} (a labeled feedback point). A naive approach would require the generation of all possible pairs of items in which the nearest neighbors are computed (with complexity $|I|^2$)).

To avoid this bottleneck, we compute an approximate kNN set for each pseudo-item v_{ij} using the following technique. We find the kNN set of v_{ij} rather among the items in I, denoted by the itemset kNN_{ij}^I (the superscript I denotes that this is a set of items and not item-pairs or pseudo-items). Instead of searching in $|I| \times |I|$ for the kNN of v_{ij}, we search in $kNN_{ij}^I \times kNN_{ij}^I$. This computation is made efficient using *locality sensitive hashing (LSH)* to deal with the large number of pairings. This way, the search space for label propagation is reduced from $O(|I|^2)$ to $O(|I|)$.

To determine the kNNs of v_{ij}, an item-item similarity measure is required. Different recommenders use different measures for capturing such similarities: cosine similarity [31], Euclidean similarity [17], weighted inner products [19, 32, 47], and angular similarity [35] are a few common choices. We treat the similarity function as a plug-in module $sim(.,.)$, and instantiate it using cosine similarity when required. Cosine similarity is emerging as a particularly convenient choice when items (and often users, categories, etc.) are represented in a shared latent space. Note that we treat items and pseudo-items uniformly and use the same function $sim(.,.)$ to compute their similarity. The output of this stage is a densified feedback matrix F_u^d.

2.3 Feedback incorporation

Optimization problem. We incorporate matrix F_u^d into the recommender by imposing a soft constraint for learning a user-specific mapping function $g(.,.)$ with $\vec{w}_u$ as its parameter vector. The goal is to learn preference vectors $\vec{w}_u$ for each user that can be combined with existing item representations $\vec{v}$ to produce user-specific item representations and then fed into the underlying recommender model. To learn $\vec{w}_u$, we formulate the following objective function where the signals from the densified feedback matrix are incorporated as a soft constraint:

$$\min_{\vec{w}_u} \frac{1}{m} \sum_{v_i, v_j} F_u^d(v_i, v_j) \cdot (sim(\vec{v}_i, \vec{v}_j) - sim(g(\vec{v}_i, \vec{w}_u), g(\vec{v}_j, \vec{w}_u))) + \gamma ||\vec{w}_u||^2 \tag{2}$$

where $m = |\{(v_i, v_j) | F_u^d(v_i, v_j) \neq 0\}|$. Eq. 2 includes a mapping or transformation function g (common choices would be vector translation and scaling), to be computed by the optimization solver. This serves to map original item representations and re-arrange their positions in the latent space such that their new similarities reflect the user's feedback on item pairs. The underlying intuition is to decrease or increase pairwise similarities whenever $F_u^d(v_i, v_j) = -1$ or $F_u^d(v_i, v_j) = 1$, respectively. The above objective achieves this two-fold (increasing and decreasing) effect in a unified manner. Additional L2 regularization is used on $\vec{w}_u$ to encourage small magnitude, avoiding drastic changes when it is applied inside *g()*.

After learning an (near-) optimal $\vec{w}_u$, each item vector $\vec{v}_i$ is updated to $g(\vec{v}_i, \vec{w}_u)$; we use these user-specific item vectors to generate new recommendations. We posit that such user-specific item representations helps the recommender model to incorporate the *more liked* and *less disliked* latent aspects of similarity for each user, and helps produce improved recommendations.

An alternative choice of formulating Eq. 2 would be to have only the L2 regularization term as the objective and model the signals from F as hard constraints. With this alternative approach, the constraints would become inequality constraints ($F_u^d(v_i, v_j) \cdot (sim(\vec{v}_i, \vec{v}_j) - sim(g(\vec{v}_i, \vec{w}_u), g(\vec{v}_j, \vec{w}_u))) < 0$) and would require that the KKT (Karush-Kuhn-Tucker) conditions be satisfied for an optimal solution to exist. In practice, experimenting with the hard constraint formulation resulted in null solutions for most cases; hence our soft-constraint-based method.

Implementation. The optimization in Eq. 2 for learning $\vec{w}_u$ is non-convex due to the presence of the cosine function. Stochastic gradient descent (SGD) (available in libraries like PyTorch) is used for computing near-optimal solutions.

2.4 Elixir in recommenders with RWR

Generating recommendations. We incorporate our method into RecWalk [31], a state-of-the-art recommender model based on random walks with restart. The input to this model is a heterogeneous graph (also referred to as a heterogeneous information network, HIN) $G = (N, E, \theta)$ with a set of nodes N, a set of edges $E \subseteq N \times N$ and a mapping θ from nodes and edges to their types, such that $\theta_N : N \mapsto T_N$ and $\theta_E : E \mapsto T_E$, where $|T_N| + |T_E| \geq 2$. Nodes are either of type *user* or *item*, i.e., $N = U \cup I$. Edges capture user-item interactions, denoted by $A \in \{0, 1\}^{|N|\times|N|}$, and node-node similarities presented by $S \in \mathbb{R}_+{}^{|N|\times|N|}$. So we have two types of nodes and two types of edges in this graph.

In RecWalk, the recommendation score of item v_i for user u is computed as $PPR(u, v_i)$. PPR stands for personalized PageRank [14], defined as follows:

$$\vec{PPR}(u, .) = \alpha \cdot \vec{e_u} + (1 - \alpha) \cdot \vec{PPR}(u, .) \cdot [\beta A + (1 - \beta)S] \quad (3)$$

where α is the restart probability, $\vec{e_u}$ is the one-hot vector for user u and β is the probability that a walk traverses an interaction edge. According to Eq. 3, a walk either visits one of its neighbors with probability $1 - \alpha$ or jumps back to user node u. The neighbors are connected either through interaction or similarity edges. Matrix S encodes similarities between nodes of the same type. Without loss of generality, we assume that a user is similar only to herself, i.e., $S(u_i, u_j) = 1$ if and only if $i = j$. The item-item similarity, however, is defined by the $sim(., .)$ function and hence $S(v_i, v_j) = sim(v_i, v_j)$. We simplify the notation and use S to refer only to item-item similarities. Note that RecWalk normalizes matrix S in a certain way to enforce stochasticity. We omit the details for the sake of brevity and refer users to [31] for more information. The item v in $|I|$ that has the highest $PPR(u, v)$ score, is produced as the recommendation *rec* for the user u.

Generating explanations. Suppose item *rec* is recommended to user u. Item-level explanations $\{exp\}$ in RWR-recommenders can be generated using the Prince algorithm [12]. The resulting explanation item sets are minimal and counterfactual: they ensure causality relation using the counterfactual setup that u would not have received *rec* if she had not liked the items in $\{exp\}$ in her history H_u. However, minimality of explanations is not a concern in the current context. Therefore, we take a more straightforward approach to approximate Prince by estimating the contribution score of item $v_j \in H_u$ to the recommended item *rec*:

$$contribution(v_j, rec) = PPR(v_j, rec) \quad (v_j \in H_u) \quad (4)$$

where $PPR(v_j, rec)$ is the PageRank of node *rec* personalized for node v_j. We use the top-k items with highest contributions in H_u as the explanation set $\{exp\}$ for item *rec*.

Incorporating feedback. In RecWalk, item-item similarities are explicitly captured in matrix S, i.e., $S(v_i, v_j) = sim(v_i, v_j)$. Given the items' latent representations (possibly computed by running techniques like NMF or SVD from sparse explicit feature vectors), we define $sim(v_i, v_j)$ as the cosine similarity between v_i and v_j, and hence $S(v_i, v_j) = cos(\vec{v}_i, \vec{v}_j)$. As discussed earlier, to incorporate densified feedback F_u^d, we introduce a user-specific preference vector $\vec{w}_u$ to adjust u's bias with respect to the latent aspects and update the item representations by adding $\vec{w}_u$. The transformation function g is chosen to be a vector translation, shifting universal item representations onto user-specific ones:

$$g(\vec{v}_i, \vec{w}_u) = \vec{v}_i + \vec{w}_u \quad (5)$$

The intuition behind the mapping described in Eq. 5 is to highlight (suppress) the effect of liked (disliked) features through addition of positive (negative) bias values. Plugging the definitions for g and sim and the densified matrix F_u^d into the optimization objective (Eq. 2), we learn $\vec{w}_u$ as follows:

$$\min_{\vec{w}_u} \frac{1}{m} \sum_{v_i, v_j} F_u^d(v_i, v_j) \cdot (cos(\vec{v}_i, \vec{v}_j) - cos(\vec{v}_i + \vec{w}_u, \vec{v}_j + \vec{w}_u)) + \gamma ||\vec{w}_u||^2 \quad (6)$$

Using $\vec{w}_u$, we build a user-specific similarity matrix S_u defined as:

$$S_u(v_i, v_j) = cos(\vec{v}_i + \vec{w}_u, \vec{v}_j + \vec{w}_u) \quad (7)$$

Finally, we update the personalized PageRank recommendation scores accordingly, thereby completing the integration of pairwise feedback into the recommender model:

$$\vec{PPR}(u, .) = \alpha \cdot \vec{e_u} + (1 - \alpha) \cdot \vec{PPR}(u, .) \cdot [\beta A + (1 - \beta)S_u] \quad (8)$$

Scenario	#Item feedback	#Pair feedback	Sessions	Hours
Phase 1	50	–	1	2
Phase 2	30	300	5	10
Phase 3	72	–	1	2
Total	152	300	7	14
Total (All users)	≃ 4000	7500	175	350

Table 2: Annotations per user over stages of the study (spanning a total of 350 person-hours) in each domain.

3 USER STUDY FOR DATA COLLECTION

Elixir operates in a unique framework of user judgments on similarities between recommendation and explanation pairs. It hinges on longitudinal observations of the same users providing: i) original profiles, ii) feedback on *rec* items and iii) feedback on $\langle rec, exp \rangle$ pairs, as well as iv) item-level assessments on the final recommendations. Thus, a study involving real users was imperative to demonstrate the practical viability of our proposal. To this end, we recruited 25 volunteers, who were all Masters' students of the Computer Science Department at the authors' Institute, with payment

per hour comparable to that of master workers on a crowdsourcing platform like Amazon Mechanical Turk.

3.1 Statement on ethics

Participants' privacy was fully respected: all personally identifying information concerning participants was kept private during the course of the study, and deleted after its completion. All data was stored locally, with encryption, firewall protection and other measures of this sort. During the course of the study, users had to provide ratings on individual as well as pairs of movies and books. While this is not personally sensitive per se, we recognize that the data reflects users' personal preferences. All participants signed a consent document that they agree to this data being used for research purposes and that it can be released with anonymized identifiers. The user study and the inclusion of results in this paper were approved by the Ethics Review Board at the authors' Institute.

The annotation sessions were conducted over online video conferencing, so that participants' browser activity could be monitored. To respect users' privacy, no video recordings were made. A one-hour training session was conducted, where participants were made aware of the goals of the study and their exact tasks, and were guided through examples.

3.2 Setup

The user study was conducted in two domains: (i) movies (restricted to Hollywood, because of its popularity and the users' familiarity), and (ii) books. Over the course of six weeks (three weeks for each domain), each user annotated individual as well as pairs of movies and books for a total of 28 hours. The payment was 10 Euros per hour, with the total cost amounting to $25 \times 28 \times 10 = 7,000$ Euros. For each domain, the annotations were collected in three phases that lasted three weeks.

3.2.1 Phase 1: Building users' profiles. It is essential to keep the assessment setup natural: if users were asked to rate arbitrary items and pairs that they are unfamiliar with, the judgments would be unreliable. Thus, as the first step of the study for each domain, we asked users to provide us with 50 movies and books each, that they liked, to build a true history for each user, that would create subsequent recommendations for her. Since movie or book titles can often be ambiguous, users were asked to provide us with MovieLens (https://movielens.org/home) and Goodreads (https://goodreads.com/) URLs in individualized spreadsheets. with the unique ID appearing in its URL. For each domain, we conducted this phase in a session spanning two hours which provided us with $50 \times 25 = 1,250$ user actions (i.e., likes).

3.2.2 Phase 2: Collecting feedback on items and pairs. The obtained user profiles were plugged into the RWR-based recommender model RecWalk [31], where every liked item contributes an interaction edge to the network. The union of all items rated by the 25 users forms our universe of items now, from where we generated the top-30 recommendations for each user. Along with each recommendation, we generated the top-5 explanation items $\{exp\}$ using the approximated version of the Prince algorithm [12] (Eq. 4). To investigate the role of faithful explanations in pairwise feedback, we also identified the five items $\{rand\}$ in the user's profile that are the *least similar* to the recommendation item. These serve as a proxy for pairing the recommendation with random items; they are drawn from the user's profile to ensure familiarity. The similarity is computed as the cosine between the item vectors.

The users are now presented with three tasks: (i) rate the generated recommendations (like/dislike); (ii) rate the similarity of each $\langle rec, exp \rangle$ pair (like/dislike); (iii) rate the similarity of each $\langle rec, rand \rangle$ pair (like/dislike). The two kinds of feedback, item-level in (i), and pair-level in (ii)+(iii), have very different semantics, and users were appropriately briefed and guided. Item-level feedback is straightforward, where they comment whether they liked or disliked an item. Rating an item pair, though, needs a bit more reflection on the possible similarities between the two items (two movies or two books), deciding on the most important factor in case of multiple such aspects, and providing the binary preference assessment. Participants entered their ratings in individualized spreadsheets we prepared for them. Each sheet contained several blocks where each block corresponded to one recommendation item followed by ten different explanation items for it (five $\{exp\}$ and five $\{rand\}$). To avoid any position bias, we randomly shuffled the explanation items in each block.

While the feedback remains lightweight due to the user's potential familiarity with the items, we provided some help to cue their memory. For instance, we presented each movie with its title and its corresponding MovieLens URL, where the user could see the movie's summary and key properties. Moreover, MovieLens provides a rich set of tags on actors, directors, genre, storyline and content; for item pairs we displayed the intersection set of top tags for the two movies. Users could nevertheless browse the MovieLens pages or other background sources at their discretion. Book recommendations were also presented together with some auxiliary information including their descriptions, authors, top genres as listed on their Goodreads pages and their corresponding URLs. Similar to the movie domain, we facilitated users' judgments on pairs of books by listing their common properties such as genres and authors.

Note that the assessment of $\langle rec, exp \rangle$ or $\langle rec, rand \rangle$ is decoupled from the fact whether the user likes rec, exp or $rand$ individually. To make this distinction clear, the users were walked through several reference annotations during the training session. For qualitative analysis, we also asked users to optionally articulate the dimension that was the basis of their similarity feedback. We report on this in the experimental section.

At the end of this stage, each user provided us with 30 item-level ratings (rec), and $30 \times 5 \times 2 = 300$ pair-level ratings (five pairs for each of $\langle rec, exp \rangle$ and $\langle rec, rand \rangle$). Therefore, for each domain, we had a total of 750 distinct item-level feedback points and 7,500 distinct pair-level feedback points, for a total of 25 users. This phase required ten hours from each user: to avoid task fatigue, this was spread over five two-hour sessions.

3.2.3 Phase 3: Collecting feedback on final recommendations. In the last phase of the longitudinal user study, the collected feedback is incorporated into the Elixir framework to produce improved recommendations for every user. *Item-level feedback* is cast into additional interaction edges for the original graph recommender; *pair-level feedback* is incorporated using the procedure described

in Sec. 2. In addition, we experimented with *combined feedback* incorporating both item-level and pair-level. These are the three top-level configurations in our experimental evaluation.

For incorporating pair-level feedback, there are two possibilities of using either *exp* or *rand*, altogether resulting in five variations of the recommender model. These models were each made to produce 30 recommendations, leading to 180 items to be rated by each user. However, there were overlaps in the *rec* sets across configurations. At the end, a total of $\simeq 1,800$ ratings were collected from 25 users in each domain ($\simeq 72$ per user). This phase took two hours per user, on average, and was completed in a single session at the end of the third week.

4 EVALUATION

4.1 Configurations

We evaluate Elixir for different configurations, including the baseline of exploiting solely item-level feedback on *rec* items. But before we can go to the results, we need to explain the basic setup of the experimental framework. Latent vectors necessary to initialize item representations are learnt by running non-negative matrix factorization (NMF) on the sparse matrix of movie-tag memberships from MovieLens and book-genre memberships from Goodreads using the Nimfa Python library (http://ai.stanford.edu/~marinka/nimfa/, default settings). The number of latent dimensions d is chosen to be 20 which was guided by observations on the reduction of sparsity from the original matrix. We use the SciPy library (https://bit.ly/35lVV10) for subsequent label propagation with the cosine kernel. The number k for LP was chosen to be 10, which means that for each pseudo-item, we find the 10 nearest items, and hence $\binom{10}{2} = 45$ pseudo-items. We tried two other values, $k = 5$ and 20, and observed similar results. For LSH, we used NearPy (https://pixelogik.github.io/NearPy/) with random binary projection as its hash function. LSH assigns each item vector v_i to a bucket where its approximate nearest neighbors lie. While a large number of buckets decreases the probability of neighbors to be assigned to the same bucket, a small number reduces the efficiency of kNN queries. Considering the choice of k in LP ($k = 10$), we chose the number of buckets to be 8 (corresponding to 3 random binary projection vectors). With this number of buckets, we reduce the failure rate of LSH to 15%, i.e., for only 15% of the kNN queries with $k = 10$, LSH returns less than 10 neighbors. We use our own implementations of RecWalk and Prince for generating recommendations and explanations, respectively.

The following five feedback configurations are compared:

- *Item-level feedback.* This baseline model only absorbs users' binary preferences on individual items *rec*. Such item-level feedback adds interaction edges to the RWR graph.
- *Pair-level feedback with explanations.* The model captures only the judgments on pairs of $\langle rec, exp \rangle$ items. Such pair-level signals update the similarity matrix used in RWR.
- *Pair-level feedback with random items.* This is similar to the previous configuration, except that the explanation items here are replaced by the least relevant items from the user's history ($\{rand\}$).
- *Item + pair-level feedback with explanations.* The model exploits both individual and pairwise feedback.
- *Item + pair-level feedback with random items.* This is similar to the previous configuration except that the explanation items $\{exp\}$ are replaced by $\{rand\}$.

4.2 Metrics

We evaluate the quality of recommendations generated after feedback incorporation using three metrics: i) Precision at the top-k ranks (P@k), ii) Mean Average Precision at the top-k ranks (MAP@k), and, iii) normalized discounted cumulated gain at the top-k ranks (nDCG@k, computed with binary non-graded relevance assessments). While P@k is a set-based metric considering the top-k items analogous to a *slate of recommendations*, the latter two are sensitive to the ranks of relevant items in the lists. All metrics are computed at three rank cut-off values k: 3, 5, and 10. The relatively low values of cut-off ranks are chosen to show the effectiveness of Elixir in introducing highly selective items into the top recommendations for individual users.

4.3 Initialization

To fairly compare different configurations, we train RecWalk using the same set of parameters. The restart probability α is set to 0.15 as shown effective in prior works. To highlight the effect of similarity edges, we choose $\beta = 0.1$ (a higher β indicates a higher likelihood of sampling an interaction for the random walker). A walker thus traverses similarity edges in G with probability 0.9. Using smaller values for β is also suggested in the original model of RecWalk [31]. The interaction graph G built for movies had 25 users, 621 movies, $1.3k$ interaction edges, and $11k$ similarity edges. For books, this graph was larger and denser, with 868 books, $1.3k$ interaction edges and $41k$ similarity edges. To compute PageRank scores, we use the power-iteration method with a maximum of 500 iterations.

To construct the similarity matrix S for RecWalk, we employ LSH again with a similar configuration as discussed for densification. To avoid too many edges in the graph, we only connect items with large similarity. For this, we use threshold 0.7, i.e., $S(v_i, v_j) = 0$ if $cos(v_i, v_j) < 0.7$. Matrix A is built from item-level user feedback: we define $A(u_i, v_j) = 1$ if u_i likes item v_j, and zero otherwise. Regularization parameter γ in Eq. 6, and the learning rate lr in SGD for finding an optimal $\vec{w_u}$ were tuned on a development set containing 20% of each user's ratings. We tested 10 different values for γ $(1, 2, \ldots 10)$ and 3 values for the learning rate ($[0.001, 0.01, 0.1]$) and chose the values with best performance on the development set: $\gamma = 3$ and $lr = 0.01$.

4.4 Key findings

Our key findings are presented in Table 3, where different feedback absorption strategies are evaluted over data from the user study. We make the following salient observations:

- **Pair-level feedback improves recommendations.** The most salient observation from the results in both domains is that including pairwise feedback (all table rows except the first one for each domain) on the similarity of item pairs results in substantial improvements over solely considering item-level feedback. This confirms our hypothesis that procuring feedback on pairs provides refined and highly

Setup [Movies]	P@3	P@5	P@10	MAP@3	MAP@5	MAP@10	nDCG@3	nDCG@5	nDCG@10
Item-level	0.253	0.368	0.484	0.323	0.380	0.448	0.244	0.327	0.422
Pair-level (Exp-1)	0.506*	**0.592***	0.580*	0.566*	0.599*	0.625*	0.496*	**0.557***	0.565*
Pair-level (Exp-3)	0.506*	0.568*	**0.596***	**0.630***	**0.624***	**0.634***	**0.504***	0.547*	**0.575***
Pair-level (Exp-5)	0.480*	0.512*	0.568*	0.567*	0.579*	0.591*	0.463*	0.490*	0.536*
Pair-level (Rand-5)	0.453*	0.504*	0.560*	0.536*	0.537*	0.596*	0.451*	0.486*	0.532*
Item+Pair-level (Exp-5)	**0.533***	0.544*	**0.596***	0.563*	0.571*	0.603*	0.500*	0.517*	0.562*
Item+Pair-level (Rand-5)	0.453*	0.488*	0.572*	0.486*	0.520*	0.578*	0.426*	0.458*	0.526*
Setup [Books]	**P@3**	**P@5**	**P@10**	**MAP@3**	**MAP@5**	**MAP@10**	**nDCG@3**	**nDCG@5**	**nDCG@10**
Item-level	0.253	0.336	0.436	0.309	0.343	0.417	0.233	0.296	0.379
Pair-level (Exp-1)	0.506*	0.528*	0.54*	0.603*	0.620*	0.654*	0.493*	0.512*	0.527*
Pair-level (Exp-3)	0.506*	0.536*	0.58*	0.573*	0.596*	0.612*	0.484*	0.511*	0.551*
Pair-level (Exp-5)	**0.586***	**0.600***	**0.656***	**0.726***	**0.701***	**0.692***	**0.602***	**0.607***	**0.644***
Pair-level (Rand-5)	0.480*	0.504*	0.504	0.593*	0.604*	0.588*	0.468*	0.488*	0.494*
Item+Pair-level (Exp-5)	0.493*	0.456*	0.560*	0.543*	0.566*	0.566*	0.482*	0.458*	0.529*
Item+Pair-level (Rand-5)	0.386*	0.416*	0.516*	0.469*	0.509*	0.536*	0.376*	0.399*	0.475*

Table 3: Comparison of different modes of feedback incorporation with results from the user study. The best value in every column (for each domain) is marked in bold. * denotes statistical significance of all methods over item-level feedback, with p-value ≤ 0.05 using the Wilcoxon signed rank test [43].

beneficial user signals that cannot be captured through aggregated item-level feedback.

- **Pair-level feedback is more discriminative than item-level.** Next, we compare the effectiveness of pair-level feedback vis-á-vis item-level. To have a fair comparison with respect to volume of feedback, we introduce a new setup, Pair-level (Exp-1), where the volume of pair-level feedback is similar to that of the Item-level setup. For this, we only consider users' feedback on the most relevant explanation item. This results in incorporation of 30 pairs with distinct recommendation items, that is compared to item-level feedback on these 30 items. We observe that in both domains, Pair-level (Exp-1) significantly outperforms the Item-level setup. This demonstrates the efficacy of pair-level feedback in capturing users' fine-grained interests. For completeness, we also provide results when we use top-3 and top-5 explanations (Exp-3 and Exp-5, respectively).
- **Using explanations for pair-level feedback is essential.** We set out with the goal of making explanations actionable towards model improvement. This is validated by the observation that item+pair-level for explanations are consistently and substantially better than item+pair-level for random item instead of top-ranked explanation items.

4.5 Analysis

Our longitudinal study on collecting pairwise judgments opens up the possibility of gaining insights on several issues on user behavior and feedback:

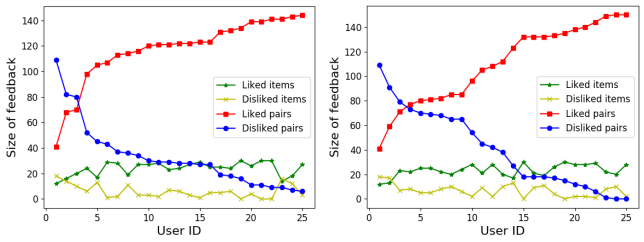
Q1. How different are positive and negative feedback by volume?
Q2. Are users who are more likely to provide negative item-level ratings also biased towards more dislikes on item pairs?
Q3. What are the most common aspects influencing similarity feedback on item pairs?
Q4. How does performance improvement correlate with the diversity of the original profile?
Q5. How do the volumes of positive and negative feedback correlate with performance improvement?

We address these questions in the subsequent analysis. At the end, Fig. 7 shows four representative examples from the user study.

To see size distribution of different feedback types (Q1), we plot the number of item-level and pair-level feedback points provided by each user in Fig. 2. Users vary in their proportions of positive and negative feedback. Overall, users enter much more positive feedback than negative. Reasonable numbers for all four types of judgments (item-/pair-level × like/dislike) show that, overall, users are willing to provide the effort towards improving their recommendations. Monitoring feedback assessments over online sessions showed that pairwise feedback is indeed lightweight effort (measured by time taken for completion) and does not impose severe cognitive load on users.

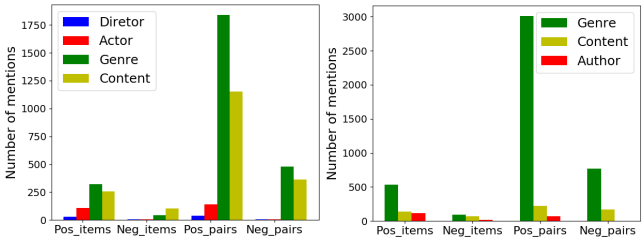
In particular, we find that users who may be biased towards more negative item-level feedback often provide substantial volumes of negative feedback on pairs. The corresponding Pearson correlations in movie and book domains are 0.53 and 0.51, respectively. Therefore, we can conclude that user behavior carries over from items to pairs (Q2). This leads to an extremely crucial insight: negative signals on items cannot be harnessed in graph recommenders, and are simply discarded as bad recommendations, never to be used again. Yet eliciting negative feedback on certain pairs of bad recommendations and good explanation items, can lead to substantial benefit for the recommender system: similar bad recommendations become less likely to be recommended to the user.

Next, we show factors influencing pairwise similarity assessment in Fig. 3 (Q3). To compare and contrast, we asked users to mention their reasons for both item- and pair-level feedback. Qualitative analysis reveals that in the movie domain, genres play the biggest role in feedback, followed by content, actor, and then director. We observe that genre and content (the latter includes storylines like plot twists, alien movies, medieval movies, etc.) are much more likely to influence user preferences than presence of specific actors or directors. This underlines the necessity of latent representation of



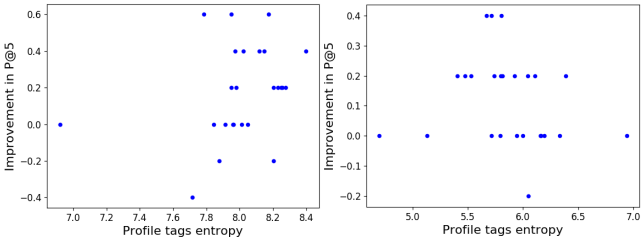
(a) Movies (b) Books

Figure 2: Per-user volume of feedback by type.



(a) Movies (b) Books

Figure 3: Key influencers behind feedback assessments.



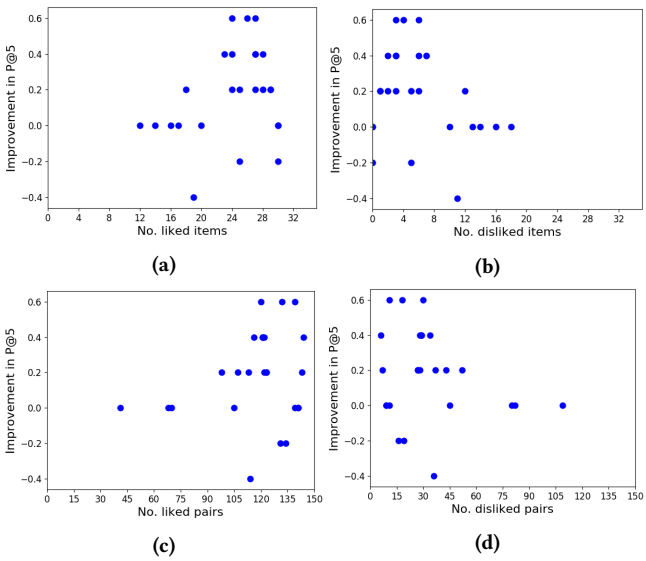
(a) Movies (b) Books

Figure 4: Connecting gains via Elixir with profile diversity.

item properties, as storylines are hard to capture in explicit feature models. Similar trends are observed for books, i.e., genre is the most frequent mention, followed by content and author. The interesting observation is that users are systematic in their behavior: in both domains, histograms have the same relative distribution for item- and pair-feedback.

We also investigate whether all users are equally likely to benefit from Elixir. Since profile sizes are kept constant to control for other factors, we try to see if performance improvements from item+pair-level feedback is connected to the *diversity* of their original profiles (Q4). To quantify diversity, we measure the entropy of the distribution of tags associated with the 50 items that were used to initialize profiles of the users (higher entropy is higher diversity). Plots for movies and books are shown in Fig. 4. Our observation for the movie domain is that Elixir helps users with relatively high interest diversity (right half) slightly more (top right) than the users with more particular interests (languishing towards the bottom left corner). The corresponding Pearson correlation is 0.29 which indicates moderate positive correlation between profile diversity and improvement level. For books, however, the Pearson correlation is −0.17 implying a small negative correlation between diversity of profiles and effectiveness of Elixir.

Next, we investigate if different volumes of feedback on the four possibilities (item-level like/dislike; pair-level like/dislike) lead to



(a) (b) (c) (d)

Figure 5: P@5-improvement w.r.t. feedback size (Movies).

notably differing performance improvements (Q5). We show the effect of feedback size on improvement levels in Fig. 5 and 6, where the two plots in the top row of each figure correspond to item-level feedback, and the bottom to pair-level. The scales and limits of x-axes within rows (and all y-axes) are kept the same for easy comparison. Here we note that dots along the same row (level) of precision correspond to the same users. The notable observation from these figures is that users who provide more positive feedback are likely to see higher improvements. This correlation is particularly more pronounced for positive item-level feedback. While the benefit of sending more positive signals as more actionable is understandable, a certain part of the blame may lie on the graph recommender itself, where "negative edges" cannot be included easily: presence or absence of edges is the standard model. This suggests further research to explore Elixir with other families of recommenders like matrix or tensor factorization, that can more easily incorporate negative feedback.

Finally, we show anecdotal examples from our user study in Fig. 7. For movie recommendation (Fig. 7a), incorporating user feedback on pairwise similarity introduces new items into the top-10 recommendations (*The Chronicles of Narnia*, *The BFG*) for their respective users. These new recommendations possess the similarity aspects liked by the user (*fantasy* for *Narnia*, *based on a book* for *The BFG*), and lack the dimensions that the user has implicitly disliked (*crime* for the first anecdote, *fiction* for the second one). Similarly, we present two instances of improvement in book recommendation in Fig. 7b, where incorporation of pair-level feedback results in reducing the relevance score of disliked items (*Memoirs of a Geisha*, *DotCom Secrets*) and bringing up more relevant items (*The Iliad*, *Mindset: The New Psychology of Success*) for the respective users in the ranked list of recommendations.

Limitations imposed by resource constraints. One limitation of this work is the scale of the user study. Evaluating ELIXIR on a larger scale would incur substantially more monetary cost and require design and implementation of a large-scale system suitable for orchestrating and monitoring the longitudinal process of

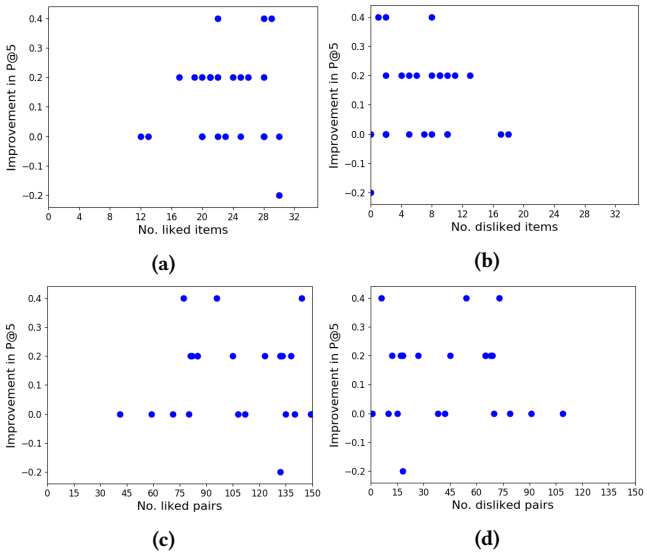


Figure 6: P@5-improvement w.r.t. feedback size (Books).

user-system interactions. Resource constraints also impact the possibility of full exploration of the parameter space in this work, such as a thorough search for the best number of latent dimensions d, as that might require the repetition of the whole study. Nevertheless, we evaluated another value $d = 10$ in the movie domain to verify the robustness of Elixir. Trends were very similar: P@5 values for item-level, pair-level (top explanations), and item+pair level (top explanations) came out to be 0.520, 0.712, and 0.712, respectively, retaining previously observed statistically significant trends of superiority of configurations involving pair-level feedback over item-level only.

5 RELATED WORK

5.1 Explaining recommendations

Explaining recommendations is a key component towards transparency, user satisfaction and scrutability [3, 38]. Herlocker et al. [16] presented the first work on explainability for collaborative-filtering based recommenders by locating people with similar profiles (neighbors). With the introduction of the Latent Factor Model (LFM) by Koren [21], research on interpretability of latent features gained attention [11, 33, 50]. To mitigate the sparseness of ratings in standard collaborative filtering, hybrid recommenders based on Knowledge Graphs (KGs) were introduced. Explanations in these models are mostly path-based, that is, a recommendation item is explained by finding the most relevant paths from the user node [1, 30, 39, 42, 45]. Explainability is particularly important in sophisticated neural models, that mostly use the attention mechanism over words [36], reviews [6], items [11] or images [9] to learn their importance for a given recommendation. Another line of work focuses on generating post-hoc explanations for models that lack transparency [12, 29, 33, 40]. For example, Ghazimatin et al. proposed a method for generating minimal and counterfactual explanations for recommenders with random walk at their core [12]. Yet another example is in LFMs where post-hoc explainability can be approached via using association rule mining [33].

5.2 Critique-based recommenders

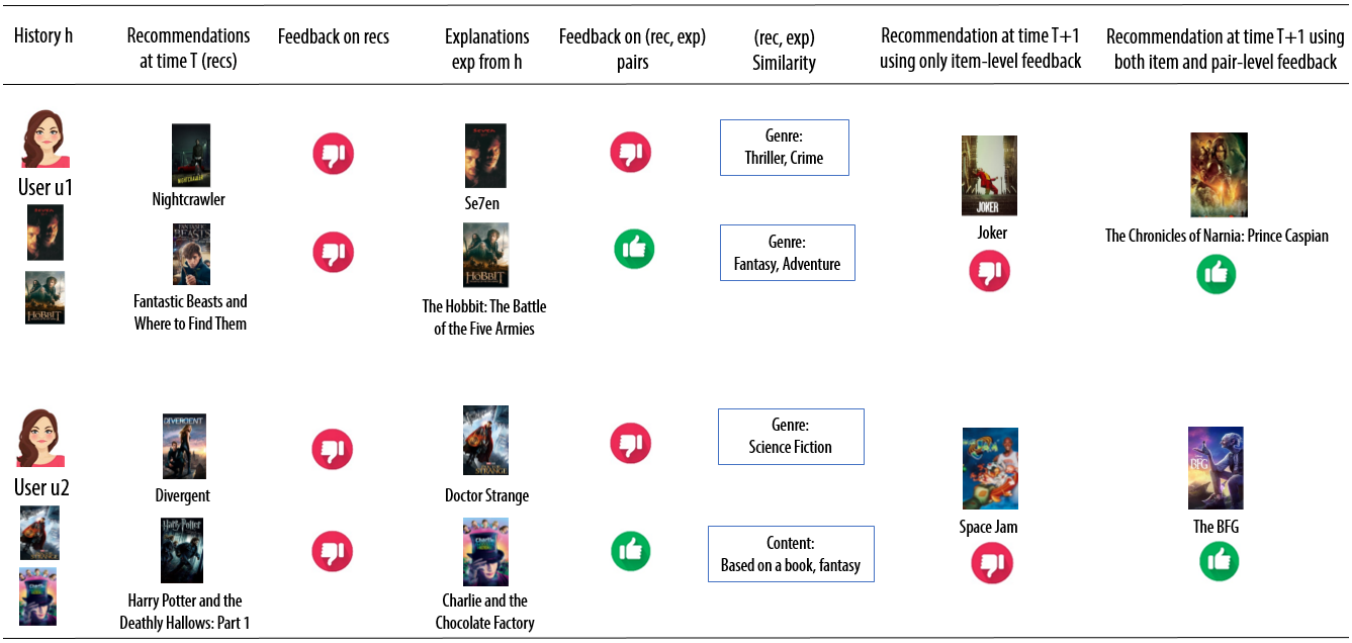
In most of the prior works on explainable recommendation, the role of explanations is limited to providing users with insights into the recommendation model. This limits scrutability as users might not have a clue how to correct the system's reasoning. To increase user control over the recommendation process, critique-based recommenders were introduced [7, 8]. Critiquing is a method for conversational (a.k.a. sequential and interactive) recommendation that adapts recommended items in response to user preferences on item attributes. Incremental critiquing/tuning [23, 28, 34] was thus proposed to improve recommendation quality over successive recommendation cycles. However, this restricts users to critique/tune based on explicit item properties which are hard to generalize. Recent works in the form of Deep Language-based Critiquing (DLC) [26, 44] address this challenge by accepting arbitrary language-based critiques to improve the recommendations for latent factor based recommendation models. In [27], Luo et al. improve the complexity of the existing critiquing frameworks by revisiting critiquing from the perspective of Variational Autoencoder (VAE)-based recommendation methods and keyphrase-based interaction. Existing critique-enabled recommenders mostly focus on negative feedback on concrete features of individual recommendation items. In Elixir, we address this limitation by enabling users to give both positive and negative feedback on pairs of recommendation and explanation items.

5.3 Set-based preference

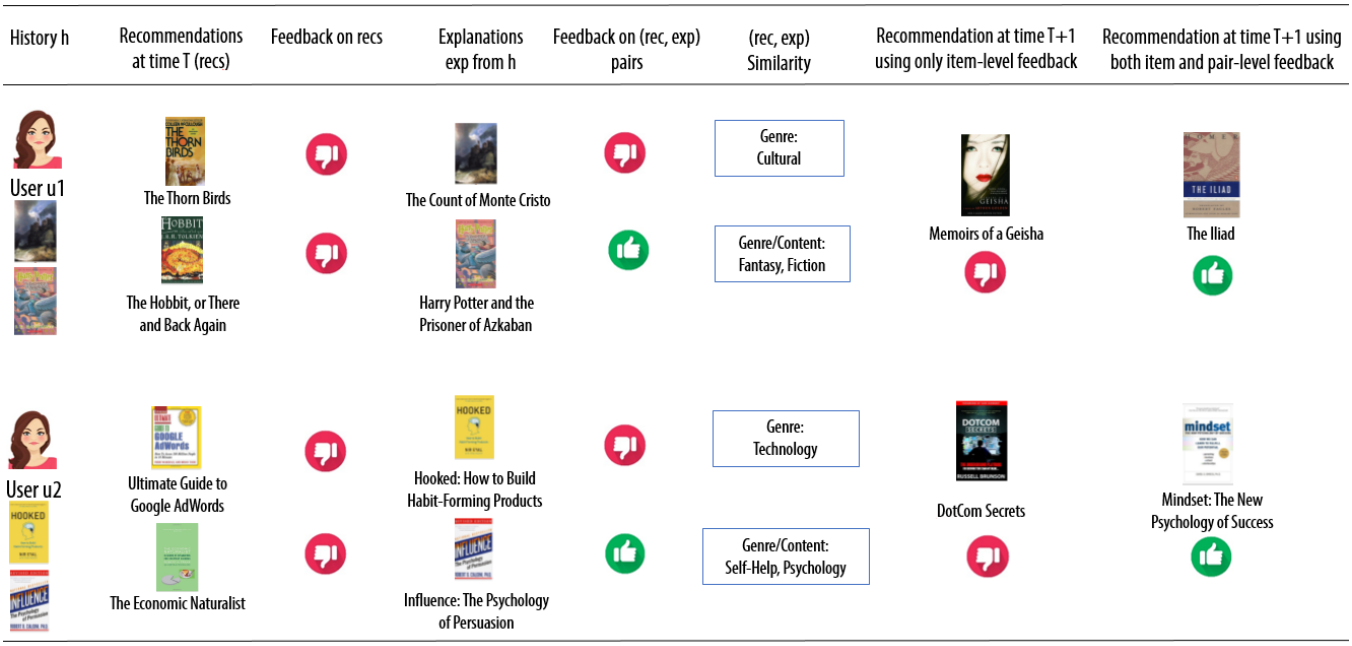
Most recommendation approaches rely on signals provided by users on individual items. Another mechanism of eliciting preference is to ask users for feedback on itemsets. Such set-based preference annotations help in faster learning of user interests, especially in cold start situations [5]. Moreover, users who are not willing to provide explicit feedback on individual items due to privacy concerns may agree to provide a single rating to a set of items, as it provides a certain level of information abstraction. At the same time, from the given set-based rating, some information regarding item-wise preference can be inferred. In the same vein, in [37], authors gathered users' preferences on itemsets and developed a collaborative filtering method to predict ratings for individual items in the set. Apart from understanding user profiles, set-based learning is also useful in works that have focused on recommending lists of items or bundles of items to users such as recommendation of music playlists [2], travel packages [24], and reading lists [25]. Elixir reinforces this viability of set-based feedback.

6 CONCLUSION

In this work, we have shown how explanations for recommendations can be made actionable by incorporating user feedback on pairs of items into recommender systems. Elixir is a human-in-the-loop system that proactively elicits lightweight user feedback on the similarity of recommendation and explanation pairs. Elixir subsequently densifies this feedback using a smart combination of label propagation and locality sensitive hashing, learns user-specific item representations by means of a soft-constraint-regularized optimization, and seamlessly injects these learned signals into the underlying recommender. We instantiated this framework with one



(a) Movies



(b) Books

Figure 7: Anecdotal examples showing the effectiveness of Elixir in introducing relevant recommendations to the top-10.

major family of recommender models, based on random walks with restart and exemplified by the RecWalk method. Our experimental evaluation, based on a longitudinal user study, showed major gains in recommendation quality. This demonstrates the power of the proposed Elixir framework to learn more discriminative latent features about user preferences, which are disregarded in traditional item-level ratings.

Future work would naturally focus on extending Elixir to other families of recommenders such as matrix/tensor factorization or

neural methods, exploring alternative strategies for absorbing pairwise feedback, and investigating the effectiveness of ELIXIR for long-tail users with sparse profiles.

ACKNOWLEDGMENTS

This work was supported by the ERC Synergy Grant 610150 (imPACT).

REFERENCES

[1] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms* 11, 9 (2018).

[2] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. 2012. Build your own music recommender by modeling internet radio streams. In *WWW*.

[3] Krisztian Balog and Filip Radlinski. 2020. Measuring Recommendation Explanation Quality: The Conflicting Goals of Explanations. In *SIGIR*.

[4] Krisztian Balog, Filip Radlinski, and Shushan Arakelyan. 2019. Transparent, scrutable and explainable user models for personalized recommendation. In *SIGIR*.

[5] Shuo Chang, F Maxwell Harper, and Loren Terveen. 2015. Using groups of items for preference elicitation in recommender systems. In *CSCW*.

[6] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Neural attentional rating regression with review-level explanations. In *WWW*.

[7] Li Chen and Pearl Pu. 2007. Hybrid critiquing-based recommender systems. In *IUI*.

[8] Li Chen and Pearl Pu. 2012. Critiquing-based recommenders: Survey and emerging trends. *User Modeling and User-Adapted Interaction* 22, 1-2 (2012).

[9] Xu Chen, Hanxiong Chen, Hongteng Xu, Yongfeng Zhang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2019. Personalized fashion recommendation with visual explanations based on multimodal attention network: Towards visually explainable recommendation. In *SIGIR*.

[10] Xu Chen, Zheng Qin, Yongfeng Zhang, and Tao Xu. 2016. Learning to rank features for recommendation over multiple categories. In *SIGIR*.

[11] Weiyu Cheng, Yanyan Shen, Linpeng Huang, and Yanmin Zhu. 2019. Incorporating interpretability into latent factor models via fast influence analysis. In *KDD*.

[12] Azin Ghazimatin, Oana Balalau, Rishiraj Saha Roy, and Gerhard Weikum. 2020. PRINCE: Provider-side Interpretability with Counterfactual Explanations in Recommender Systems. In *WSDM*.

[13] Azin Ghazimatin, Rishiraj Saha Roy, and Gerhard Weikum. 2019. FAIRY: A Framework for Understanding Relationships between Users' Actions and their Social Feeds. In *WSDM*.

[14] Taher H. Haveliwala. 2003. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *IEEE Trans. Knowl. Data Eng.* 15, 4 (2003), 784–796.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.

[16] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. 2000. Explaining collaborative filtering recommendations. In *CSCW*.

[17] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *WWW*.

[18] Yucheng Jin, Wanling Cai, Li Chen, Nyi Nyi Htun, and Katrien Verbert. 2019. MusicBot: Evaluating critiquing-based music recommenders with conversational interaction. In *CIKM*.

[19] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored item similarity models for top-n recommender systems. In *KDD*.

[20] Wang-Cheng Kang, Mengting Wan, and Julian McAuley. 2018. Recommendation through mixtures of heterogeneous item relationships. In *CIKM*.

[21] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*.

[22] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[23] Benjamin Charles Germain Lee, Kyle Lo, Doug Downey, and Daniel S Weld. 2020. Explanation-Based Tuning of Opaque Machine Learners with Application to Paper Recommendation. *arXiv preprint arXiv:2003.04315* (2020).

[24] Qi Liu, Yong Ge, Zhongmou Li, Enhong Chen, and Hui Xiong. 2011. Personalized travel package recommendation. In *ICDM*.

[25] Yidan Liu, Min Xie, and Laks VS Lakshmanan. 2014. Recommending user generated item lists. In *RecSys*.

[26] Kai Luo, Scott Sanner, Ga Wu, Hanze Li, and Hojin Yang. 2020. Latent Linear Critiquing for Conversational Recommender Systems. In *WWW*.

[27] Kai Luo, Hojin Yang, Ga Wu, and Scott Sanner. 2020. Deep Critiquing for VAE-Based Recommender Systems. In *SIGIR*.

[28] Kevin McCarthy, Yasser Salem, and Barry Smyth. 2010. Experience-based critiquing: Reusing critiquing experiences to improve conversational recommendation. In *ICCBR*.

[29] Cataldo Musto, Marco de Gemmis, Pasquale Lops, and Giovanni Semeraro. 2020. Generating post hoc review-based natural language justifications for recommender systems. *User Modeling and User-Adapted Interaction* (2020).

[30] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. 2019. Linked open data-based explanations for transparent recommender systems. *International Journal of Human-Computer Studies* 121 (2019).

[31] Athanasios Nikolakopoulos and George Karypis. 2019. RecWalk: Nearly uncoupled random walks for top-n recommendation. In *WSDM*.

[32] Athanasios N Nikolakopoulos, Vassilis Kalantzis, Efstratios Gallopoulos, and John Garofalakis. 2019. EigenRec: Generalizing PureSVD for effective and efficient top-N recommendations. *Knowledge and Information Systems* 58, 1 (2019).

[33] Georgina Peake and Jun Wang. 2018. Explanation mining: Post hoc interpretability of latent factor models for recommendation systems. In *KDD*.

[34] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. 2004. Incremental critiquing. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*.

[35] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*.

[36] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. 2017. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *RecSys*.

[37] Mohit Sharma, F Maxwell Harper, and George Karypis. 2019. Learning from sets of items in recommender systems. *TiiS* 9, 4 (2019).

[38] Nava Tintarev and Judith Masthoff. 2012. Evaluating the effectiveness of explanations for recommender systems. *User Modeling and User-Adapted Interaction* 22, 4-5 (2012).

[39] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM*.

[40] Xiting Wang, Yiru Chen, Jie Yang, Le Wu, Zhengtao Wu, and Xing Xie. 2018. A reinforcement learning framework for explainable recommendation. In *ICDM*.

[41] Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2020. Learning and Reasoning on Graph for Recommendation. In *WSDM*.

[42] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable reasoning over knowledge graphs for recommendation. In *AAAI*.

[43] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics*.

[44] Ga Wu, Kai Luo, Scott Sanner, and Harold Soh. 2019. Deep language-based critiquing for recommender systems. In *RecSys*.

[45] Yikun Xian, Zuohui Fu, S Muthukrishnan, Gerard De Melo, and Yongfeng Zhang. 2019. Reinforcement knowledge graph reasoning for explainable recommendation. In *SIGIR*.

[46] Zhu Xiaojin and Ghahramani Zoubin. 2002. Learning from labeled and unlabeled data with label propagation. *CMU Technical Report* (2002).

[47] Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. 2019. Relational collaborative filtering: Modeling multiple item relations for recommendation. In *SIGIR*.

[48] Yongfeng Zhang, Xu Chen, et al. 2020. Explainable Recommendation: A Survey and New Perspectives. *Foundations and Trends® in Information Retrieval* 14, 1 (2020).

[49] Yongfeng Zhang, Xu Chen, Yi Zhang, Min Zhang, and Chirag Shah. 2020. EARS 2020: The 3rd International Workshop on ExplainAble Recommendation and Search. In *SIGIR*.

[50] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *SIGIR*.

[51] Guoshuai Zhao, Hao Fu, Ruihua Song, Tetsuya Sakai, Zhongxia Chen, Xing Xie, and Xueming Qian. 2019. Personalized reason generation for explainable song recommendation. *TIST* 10, 4 (2019).