recoXplainer: A Library for Development and Offline Evaluation of Explainable Recommender Systems

Ludovik Coba, Koa Health, Spain
Roberto Confalonieri, Free University of Bozen-Bolzano, Italy
Markus Zanker, Free University of Bozen-Bolzano, Italy, and University of Klagenfurt, Austria

***Abstract*—As recommender systems today play an important role in our online experience and are involved in a wide range of decisions, multiple stakeholders demand explanations for the corresponding algorithmic predictions. These demands—together with the benefits of explanations (e.g., trust, efficiency, and sometimes even persuasion)—have triggered significant interest from researchers in academia and industry. Nonetheless, to the best of our knowledge, no comprehensive toolkit for development and evaluation of explainable recommender systems is available to the community yet. Instead, researchers are frequently faced with the challenge of re-implementing prior algorithms when creating and evaluating new approaches. Aiming to address the resulting need, this paper introduces recoXplainer, an easy-to-use, unified and extendable library that supports the development and evaluation of explainable recommender systems. recoXplainer includes several state-of-the-art black-box algorithms, model-based and post-hoc explainability techniques, and offline evaluation metrics to assess the quality of the explanation algorithms.**

***Index Terms*—Explainable Recommender Systems, Model-based Explanations, Post-hoc Explanations, Evaluation, Software library.**

I. Introduction

Over the last few years, recommender systems (RSs) have increasingly been deployed in everyday real-world applications, making RSs de-facto a fundamental part of our daily decision-making in many contexts and domains. At the same time, there have been growing societal demands for providing explanations on how recommendations or, more generally, predictions have been generated. In addition, the European Union's General Data Protection Regulation (GDPR) stipulates a right to 'meaningful information about the logic involved' in automated decision-making—commonly interpreted as a 'right to an explanation'.[1] In the case of RSs, these explanations are meant to inform the user about the rationale behind generated recommendations. Above and beyond meeting regulatory requirements, explanations likely make automated systems more transparent, efficient, and even persuasive by ultimately increasing users' trust [1]. Together, the societal demand, the regulatory requirements, and the commercial benefits led to a surge in activity in the field of eXplainable Recommender Systems (XRSs) [2], [3].

However, the practice in the field explainability in RSs hitherto has not only been addressed on the level of mathematical formulations but also lacks working implementations in many cases. In particular, to the best of our knowledge, no comprehensive toolkit for XRSs is available to the research community. Therefore, researchers frequently have to re-implement and replicate previous work in order to evaluate new research ideas. This often leads to implementation and/or bench-marking related problems [4], [5], [6]. The state of affairs is different for eXplainable AI, for which a plethora of tools have been developed (see e.g., [7], [8], [9], [10], [11], [12]).

In order to support efficient scientific processes we therefore put forward a library for the development and evaluation of explainable recommender systems. Its focus is on extensibility, and support for code re-use, replication and reproduction of best practices. Hence, this paper presents recoXplainer (Fig. 1), a Python library for aiding research in the emerging field of XRSs. recoXplainer aims at accommodating the needs of different stakeholders, by providing a unified, flexible, and easy-to-use library that supports the development and evaluation of explainable recommender systems.[2]

recoXplainer has the following distinctive features:

- Ease-of-use: The library implements a pipeline for the development of XRSs consisting of data preprocessing, model training, recommendation, explanation, and evaluation.
- Unified: The library includes several state-of-the-art recommendation algorithms, explanation techniques, and implements standard evaluation protocols.
- Extensibility: The library provides an interface similar to other popular machine learning and deep learning frameworks to be easily extended.

Corresponding author: roberto.confalonieri@unibz.it
L. Coba is with Koa Health, Spain. When this work was started, he was with the Free University of Bozen-Bolzano, Italy. E-mail: ludovik.coba@koahealth.com
R. Confalonieri, and M. Zanker are with the Free University of Bozen-Bolzano, Italy. E-mail: {name.surname}@unibz.it

[1]Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) [2016] OJ L119/1.

[2]recoXplainer was demoed at AAAI-21 [13]. The code is available at https://github.com/ludovikcoba/recoxplainer and it is released under the MIT license.

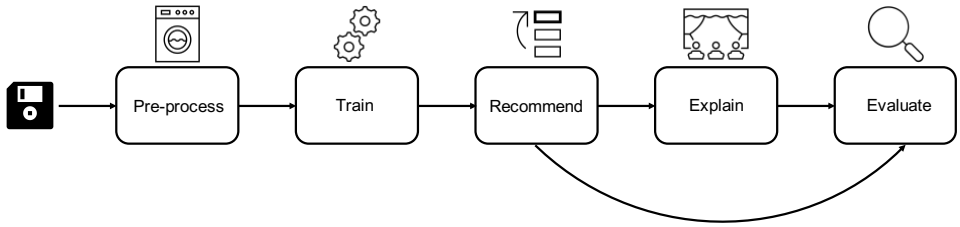


Fig. 1: recoXplainer supports the development of explainable recommendation systems through five development steps: data pre-processing, model training, recommending, explaining, and evaluating the recommendation and/or the explanations generated.

The paper is organised as follows. Section II surveys the field of explainable RSs. Section III gives a description of the library with its implemented black-box and explanation algorithms as well as the included standard evaluation procedures and metrics. Section IV exemplifies how to use the library through code snippets. Next, Section III-D presents a use-case, and compares the capability of post-hoc explanation algorithms to explain the recommendations generated by black-box algorithms using embeddings. Finally, Section VI concludes the paper and outlines some future works.

II. Explainable Recommender Systems

Research into explainable or interpretable AI systems has a long tradition [14], [15] and enjoys great popularity within AI [16]. Providing an exhaustive overview on past and present efforts goes beyond the scope of the present paper, however [17], [18], [19], [20], [21], for instance, offer comprehensive entry points into the relevant literature of explainable AI and its applications.

Current thinking holds that transparent explanations show the user how a model functions internally [2], either through deterministic descriptions of the functioning of algorithms, or by explaining single components [22]. In the context of RSs, this is the case for the type of explanations used by platforms like Facebook or Amazon, e.g., "Friends who liked item X also liked items $Y, Z, \ldots$" (referred to as neighbour-style explanations [23]); or by Netflix, e.g., "Because you watched/liked item $X, \ldots$, you are recommended item $Y, Z, \ldots$". These explanation formats follow multiple objectives, such as to inspire trust and satisfaction or to persuade them in making choices [2]. Another type of explanations are attribute-based, where the user is presented with the attributes (i.e., human comprehensible features, such as genre) that most influenced a given recommendation [24]. The latter explanation style is frequently applied for content-based RSs.

According to the taxonomy of explanation strategies provided by Friedrich and Zanker [24], and more recently by Mittelstad [22], there are three main algorithmic paradigms for building explanations. This categorisation depends on the type of reasoning used, namely white-box, model-based, and black-box. White-box approaches rely on transparent algorithms that generate recommendations that are explainable by design (e.g., content-based and knowledge-based). In model-based approaches, explanations are generated by combining a recommendation and an explanation in a single model (e.g., [25], [26]). Interpretability is then obtained by performing a further transformation into the latent space to induce linearity, or by constraining the loss function with an interpretable component, such that the model is no longer a black-box. In contrast, black-box paradigms build post-hoc explanations by interpreting the behaviour of the black-box model after it has been trained, rather than constraining it (e.g., [22], [27]). In this context, explanations are generated by choosing a proxy (i.e., a potential interpretable model) that approximates the behavior of the black-box. Peake et al. [27], for instance, explain recommendations generated by collaborative filtering by means of association rules that act as local explanations of the black-box. Hence, the explanations are generated on the output of the model, and there is no under-the-hood reworking of the black-box.

White-box, model-based, and black-box paradigms have different advantages and limitations. White-box approaches generate recommendations that are interpretable by design. However, they do not always scale well, and the explicit information needed to draw the recommendations is not always available or accessible due to privacy concerns. Post-hoc explanations commonly impose an accuracy-interpretability trade-off. They require an additional training step to train the proxy that is used to approximate the black-box. Besides, the proxy is usually only capable of generating a subset of the original decisions. Therefore, it cannot explain all the decisions made by the black-box. Model-based approaches have a big advantage w.r.t. post-hoc explanations since they do not require to train an interpretable proxy, or external data for computing the explanations. However, they come at a price, such as losing the flexibility of the model (e.g., Constrained Matrix Factorisation [25] has a strong popularity bias). Finally, post-hoc explanations do not affect the structural integrity of the black-box model as in the case of some model-based approaches.

A common problem of post-hoc explanations of black-box models based on collaborative filtering is the lack of information that can be exploited to generate comprehensible explanations. To bridge this gap, some approaches

proposed to discover similarity relations between user preferences (e.g., items in the past) and external sources of information related to content features, e.g., in the form of linked data or expert reviews. For instance, the authors of [28], [29] proposed algorithm-agnostic frameworks able to generate natural language explanations supporting the suggestions provided by generic recommendation algorithms.

III. RecoXplainer

recoXplainer is a unified, extendable and easy-to-use library for development and assessment of XRSs. recoXplainer includes several explainability techniques that are useful for various groups of stakeholders. The goal is to equally help applied data scientists and researchers focusing their efforts on improving this field. Hence, the library provides an interface similar to other popular machine learning and deep learning libraries, such as scikit-learn and pytorch.

recoXplainer supports an explainability pipeline that consists of five main steps (Fig. 1):

1) Pre-process: Data pre-processing is an essential step for the creation of a recommender engine. Current pre-processing features include making user and item ids consecutive (to avoid sparsity when creating latent factors), binarisation (to transform explicit feedback into implicit), and data splitting (to create training, development, and test sets).
2) Train: Once a dataset is pre-processed, it can be used to train one of the algorithms implemented in the library. Currently, the following state-of-the-art algorithms are supported: Alternating Least Squares (ALS) [30], Bayesian Personalised Ranking (BPR) [31], Generalised Matrix Factorisation (GMF) [32], and Multi-Layer Perceptron (MLP) [32].
3) Recommend: This step allows one to produce a ranked list of recommendations given a set of users. The implementation is flexible, and it is independent from the machine learning library used to create the trained model. For instance, a machine learning developer can train a model in tensorflow and still create a recommendation list using our library. The user can extend the library by adding new implementations through re-implementing the recommender method interface.
4) Explain: This step allows one to produce a list of explanations given a set of recommendations for a list of users. The library supports the generation of item-style and user-style explanation formats. These formats depend on the explanation algorithm used. The library includes both model-based and post-hoc explanation algorithms. Among model-based algorithms there are ALS-Explain [30], Explainable Matrix Factorisation [33], and Explainable Autoencoder [34]. Post-hoc explanations are drawn using Association Rules, K-Nearest Neighbor (kNN), and Gaussian Mixture Model (GMM).

TABLE I: Symbols and definitions used throughout the paper.

Symbol	Definition
u	some user
i	some item
r_{ui}	rating of user u on item i
c_{ui}	confidence about observing r_{ui}
I	domain of all items
U	domain of all users
R	interaction matrix
$\hat{R}$	approximated interaction matrix
I_u	set of items rated by user u
U_i	set of users that rated item i
L_u	recommendation list for user u
P	users embedding
Q	items embedding
p_u	embedding for user u
q_i	embedding for item i
K	set of clusters
$Expl_{ui}$	Explainability of item i for user u

5) Evaluate: The library provides a number of metrics to evaluate both recommendations and explanations. Precision-oriented metrics, such as hit ratio, and normalised Discounted Cumulative Gain (nDCG), are used to evaluate a trained model. Off-line evaluation metrics, such as Mean Explainability Precision [33], Model Fidelity [27], and Explainability Score are used to evaluate explanations.

The remainder of this section describes the explanation algorithms and techniques that have been included in the recoXplainer library. Table I summarises the notation used throughout the paper.

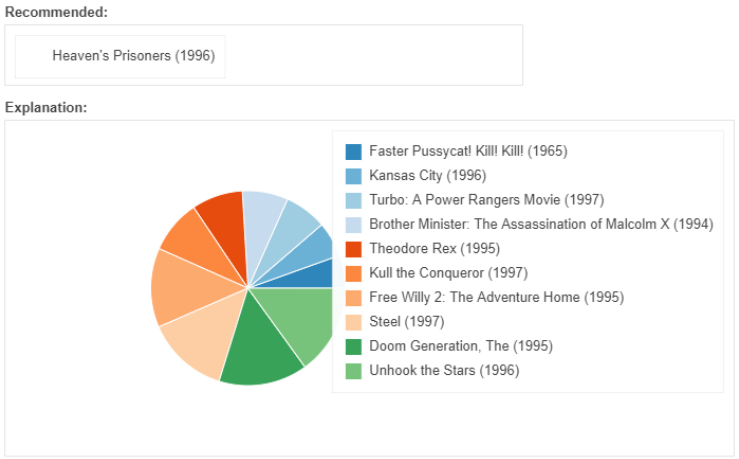
A. Implemented Black-box Algorithms

recoXplainer implements some frequently used collaborative filtering algorithms:

- Alternated Least Squares: ALS was introduced by Hu [30] for implicit feedback.
- Bayesian Personalised Ranking - Matrix Factorisation: BPR-MF is an algorithm introduced by [31] for implicit feedback.
- Generalised Matrix Factorisation: GMF was introduced by [32] in the context of neural collaborative filtering for recommender systems. It consists of an adaptation of Matrix Factorisation (MF) algorithms to a layered neural network.
- Multi-Layer Perceptron: MLP is a collaborative filtering algorithm introduced by [32]. This model learns to predict new items by first concatenating user and item embeddings and then by applying several hidden layers.

B. Model-based Explanation Algorithms

Model-based explanation algorithms generate explainable recommendations either by leveraging the linearity present in the recommendation algorithm, or by constraining the loss function with an interpretable component. Explanation algorithms belonging to this category do not



(a)

You were recommended ItemID-985 because similar users to you rated this item as follows:

Rating	Similar users' ratings
★	0
★★	0
★★★	0
★★★★	11
★★★★★	22
Average Rating:	4.5 ★★★★½

(b)

Fig. 2: Examples of (a) explaining a recommendation for a random user in ML-100K with ALS-Explain; (b) explanation generated with the Explainable Matrix Factorisation.

require additional computations (i.e., training an interpretable proxy) or external data to generate the explanations. However, in some cases they limit the flexibility of the model. For instance, Constrained or Explainable Matrix Factorisation is known to have a strong popularity bias. recoXplainer includes three model-based explanation algorithms: ALS-Explain [30], Explainable Matrix Factorisation [33], [35], and Explainable Autoencoder [34].

1) ALS-Explain: This explanation algorithm was proposed by Hu [30]. It is an explanation method that leverages the linearity present in matrix factorisation and the update rules to generate item-style explanations. Traditional ALS learns the user and item factors (embeddings) by minimising the following cost function:

$$\min \sum_{u,i \in R} c_{ui}(r_{ui} - p_u q_i^T)^2 + \beta(\|p_u\|^2 + \|q_i\|^2)$$

The optimisation process fixes, in alternating turns, first the user factors P, then the item factors Q, such that the cost function becomes quadratic. Therefore, minimisation can be achieved by solving the following least squares problems:

$$p_u = (Q^T C^u Q + \beta I)^{-1} Q^T C^u p(u), \quad (1)$$
$$q_i = (P^T C^i P + \beta I)^{-1} P^T C^i p(i), \quad (2)$$

where $p(.)$ is the binary encoding of the ratings for user u or item i; C^i and C^u are diagonal matrices defined as $C^i_{uu} = c_{ui}$ and $C^u_{ii} = c_{ui}$ respectively. To explain recommendations the user factors are replaced with the item factors in Eq. 1, such that a prediction depends only on the item representations (i.e., $\hat{r}_{ui} = q_i^T p_u = q_i^T (Q^T C^u Q + \beta I)^{-1} Q^T C^u p(u)$). By defining $W^u = (Q^T C^u Q + \beta I)^{-1}$, a sort-of weighted similarity is computed between two items as $s^u_{ik} = y_i^T W^u y_k$ to generate a prediction as:

$$\hat{r}_{ui} = \sum_{k:r_{uk}>0} s^u_{ik} c_{uk} \ .$$

Hence, the latent model is reduced to a linear model. Each prediction is generated by the linear contributions of past interactions. ALS-Explain generates item-style explanations (see Fig. 2a). It is worth mentioning that there is an interest to implement ALS-Explain as an additional feature in Apache Spark[3].

2) Explainable Matrix Factorisation: Explainable Matrix Factorisation (EMF) was introduced by Abdollahi and Nasraoui [33] and refined in [35] as a recommendation approach. Explainable recommendations are obtained by adding an extra soft-constraint to the traditional matrix factorisation formula as shown below:

$$\min \sum_{u,i \in R} (r_{ui} - p_u q_i^T)^2 + \frac{\beta}{2}(\|p_u\|^2 + \|q_i\|^2) + \lambda \|p_u - q_i\|^2 E_{ui},$$

where R is the set of ratings for user u on item i, $\|p_u\|^2, \|q_i\|^2$ are the L_2 regularisation terms and E_{ui} holds the information of how explainable item i is for user u. Note that $\|p_u - q_i\|$ constrains the representations of the user/item vectors in the latent space in such a way that they are close to each other (i.e., their difference is close to zero), in order to minimise the objective function (usually achieved via stochastic gradient descent).

Hence, a user-item 'explainability' matrix E is determined by computing for every user u and its identified neighbourhood (i.e., the k most similar user profiles) how frequently item i has been highly rated, i.e. E_{ui}. This matrix holds the explainability coefficients for any item i for a user u as follows:

$$E_{ui} = \sum_{\substack{\forall r \in R \\ r \geq P_\tau}} r * |NN^k(u)_{ir}|, \quad (3)$$

where $NN^k(u)$ is the set of nearest neighbours for user u, and $NN^k(u)_{ir}$ corresponds to the set of nearest neighbours of target user u who 'positively' rated i (with r greater than a threshold P_τ).

[3]https://issues.apache.org/jira/browse/SPARK-27447

Explainable Matrix Factorisation generates user-style explanations (see Fig. 2b for an example). By interchanging E_{ui} to the item-based neighbourhood, the model can be optimised to learn item-style explanations. To the best of our knowledge, recoXplainer is the first library implementing this recommendation algorithm in Python.

3) Explainable Autoencoder: Explainable Autoencoder is a single layer autoencoder [34]. The encoder takes as input a user's ratings vector (R_u) and a user's explainability score vector (E_u) that is pre-computed following Eq. 3. The vectors are mapped to a latent space as follows:

$$h = \sigma(W_1(R_u + E_u) + b),$$

where σ is the sigmoid activation function, W_1 is the weight matrix, and b is a bias vector. The decoding maps the latent representation into the predicted ratings $\hat{R}_u$ as follows:

$$\hat{R}_u = \sigma(W_2(h + b)).$$

The autoencoder is optimised by stochastic gradient descent using the mean square error loss function.

C. Post-hoc Explanation Algorithms

Post-hoc explanations are obtained in two steps. First, an interpretable proxy is trained using the recommendations generated by a black-box algorithm. Then, the trained proxy is used to compute explainable recommendations. Approaches under this category usually deal with an interpretabilty-accuracy trade-off. The recommendations generated by the proxy are typically more interpretable but less accurate than those generated by the black-box. recoXplainer implements three proxies: Association Rules (AR), k-Nearest Neighbours (kNN), and clustering based on Gaussian Mixture Models (GMM).

1) Association Rules: Association rule mining is a data mining approach that discovers the relationship between two categorical items X and Y of the form "IF X THEN Y" [37]. Given a catalogue of items I, a transaction $T \subseteq I$ is defined as the set of items that are jointly liked or interacted with (i.e., the set of co-interactions). Given an antecedent $X \subseteq I$, and a consequent $Y \subseteq I$, such that $X \cap Y = \emptyset$, an association rule between X and Y indicates that if X is in T then there is a strong likelihood that Y is also in T. The quality of the mined association rules is measured in terms of metrics such as support, confidence and lift. An association rule is usually denoted as $X \Rightarrow Y$.

Association rules can be used to provide item-style explanations (see Fig. 3a). This kind of explanations falls within one of the archetypes of explanation styles [38]. They are human interpretable[4], and they are able to approximate Matrix Factorisation [27].

Our implementation reproduces the work of [27] and extends it beyond Matrix Factorisation, providing a model-agnostic explanation approach. The explanation algorithm works as follows:

1) Association rules are mined using the predictions generated by a black-box RS (including observed interactions).
2) For each user, the learned transactions are filtered such that antecedents are in the training set, and consequents are unseen or non-interacted items respectively.
3) The resulting subset is ordered by support/confidence/lift.
4) The top-D consequents of the rules are kept as the explainable item set.

The implementation of association rule mining uses the mlxtend library.[5]

2) K-Nearest Neighbors: Similarly to AR, K-Nearest Neighbor (kNN) uses the community-similar interactions patterns to generate a prediction of the format "You are recommended y because it is similar to $x, z \ldots$" (see Fig. 3b for an example). Formally, given a target user and the associated positively-rated items, the algorithm identifies the k-most similar items for each target a and ranks them according to aggregated similarities with the different targets (see e.g., [37]).

The similarity between two items i and j is computed by means of the cosine similarity, after considering the items i and j as rating vectors $\vec{i}$ and $\vec{j}$ in the user space. Cosine similarity measures the cosine angle between those vectors as: $sim(\vec{i}, \vec{j}) = cos(\vec{i}, \vec{j}) = (\vec{i} \cdot \vec{j})/(|\vec{i}| * |\vec{j}|)$. Once similarities among all items are computed, a neighbourhood of an item i is formed by choosing the items with the highest similarity value ($NN(i)$). Predictions over a target user u and item i are calculated as the weighted sum [37]:

$$p_{u,i} = \left(\sum_{j \in NN(i)} sim(\vec{i}, \vec{j}) * R_{u,j} \right) / \left(\sum_{j \in NN(i)} sim(\vec{i}, \vec{j}) \right) .$$

By switching the items' space to the users' space, kNN can be used to generate user-based explanations. Similarity-based explanations are interpretable by design [33], [25], [23], [38] and are informative for the users [40].

The explanation algorithm based on kNN works as follows:

1) For each user and recommendation from the black-box model the kNN items are computed.
2) Neighbours are filtered so that they are in the training set of the user.
3) Only unseen interactions are filtered, and the similarity score is used to rank items (in analogy to the confidence score used to evaluate association rules).
4) The top-D predictions and their corresponding explanations are drawn.

The implementation of kNN uses the scikit-learn library.[6]

3) Gaussian Mixture Model: Gaussian Mixture Model (GMM) is a probabilistic model-based clustering approach [41]. This approach assumes that the data is generated by a mixture of underlying probability distributions.

[4]Several studies (see e.g., [39], [25], [38]) have shown how item-style explanations are understandable by users as well as informative [40].

[5]http://rasbt.github.io/mlxtend/

[6]https://scikit-learn.org/stable/

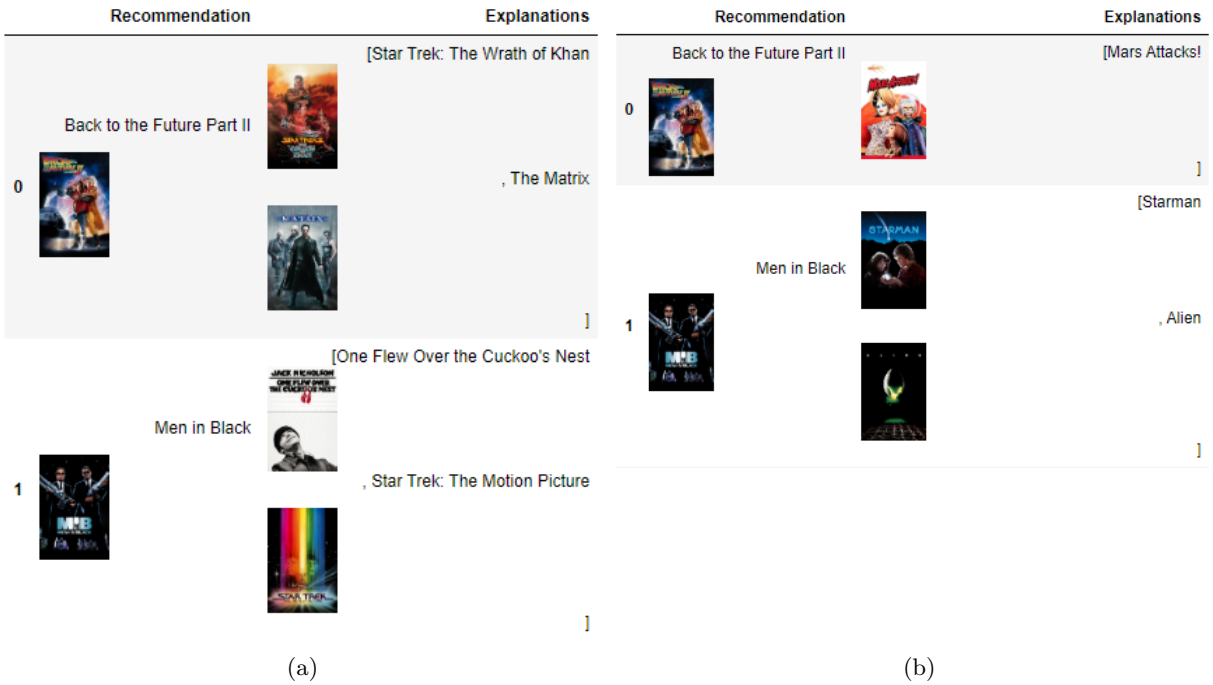


Fig. 3: Example of post-hoc (item-style) explanation with (a) association rule and (b) kNN explanations on a top-3 list generated with MLP for a random user in ML-1M [36].

In practice, this means that each cluster can be mathematically represented by a parametric probability distribution. Under this view, clustering a set of points into k clusters amounts to infer the mixing probabilities (weights) and parameters of the mixture distribution of k Gaussian probability distributions, one for each cluster [41, Ch. 3].

A common problem of mixture models is that the number of component k is not given a priori, and it needs to be estimated. The search for an optimal k is usually addressed by taking into account model selection criteria, such as the Bayesian information criterion (BIC) or the Akaike information criteria (AIC). These criteria penalise models that have more parameters (e.g., clusters) to learn, and reward models that fit the data well. Models selected by the BIC tends to be simpler then the one selected by the AIC [41, Ch. 3].

The main idea of using GMM to generate explanations is that trained embeddings, learned by a black-box algorithm, reflect intrinsic similarities of users in terms of their past interactions w.r.t. certain items. Thus, it is possible to generate item-style explanations by taking into account the embeddings belonging to the same cluster. Table II shows several examples of item-style explanations that are generated using GMM as clustering algorithm.

The explanation algorithm based on GMM works as follows:

1) User and item embeddings learned by a black-box algorithm are extracted.
2) Embeddings' dimension is reduced using a dimensionality reduction technique.
3) Embeddings are clustered using GMM and BIC as a criterion to find an optimal number of clusters.
4) Item-style explanations of each user recommendation are extracted by taking into account the most similar item embeddings in the cluster.

The dimensionality reduction step is required because, since user and item embeddings belong to a high dimensional space, it is commonly difficult to find clustered structures [42]. To reduce such complexity, we adopted a dimensionality reduction technique, namely *Uniform Manifold Approximation* (UMAP) [43]. Another approach is t-SNE [44]. Whilst these approach reduce the dimensions, they are also typically able to preserve (part of) the original distances of the high-dimensional embeddings' space. The implementation of GMM and UMAP uses the scikit-learn and umap-learn library respectively.[7]

D. Evaluating Explanations

In offline evaluation of explanations, a formal definition of interpretability is used as a proxy for quantifying the explanation quality [45]. Our library implements three metrics belonging to two categories of offline evaluation metrics for explainability. The first category of metrics evaluates whether a model behaves as expected. The second category evaluates explainability via the proxy. recoXplainer implements one metric for the first category and two metrics for the second one.

It is important to notice that offline evaluation is not the only way explanations should be evaluated. In recommender systems, there is often the need to evaluate the generated explanations with users. However, online evaluation is out of the scope of this paper. The explanations

[7]https://pypi.org/project/umap-learn/

TABLE II: Examples of item-style explanations. Recommendations and item-style explanations for a random user in ML-1M are generated from item embeddings trained with MLP and clustered according to GMM.

Rank	Recommendation		Because you liked:
1	While You Were Sleeping	⇒	[Free Willy, Sleepless in Seattle, Welcome to the Dollhouse]
2	The Doom Generation	⇒	[The Man Who Knew Too Little, My Own Private Idah, Another Stakeout]
3	Persuasion	⇒	[I.Q., Everyone Says I Love You, True Romance]

generated by our library are indeed archetypal explanation styles, namely item- and user-style explanations, for which a number of studies, which measure their human-understandability, already exist (e.g., [39], [25], [38], [40]).

1) Mean Explainability Precision: This metric is particularly pertinent to the case of model-based explanations (see Section III-B), when additional components are added to the loss function. We implemented Mean Explainability Precision (MEP) as proposed by Abdollahi and Nasraoui [33]. According to this metric, the explainability of a recommendation list L_u for a given user u is measured as the ratio between the number of explainable items in L_u and $|L_u|$, as shown as follows:

$$\mathsf{MEP} = \frac{1}{|U|} \times \sum_{u \in U} \frac{|\{i : i \in L_u, E_{ui} > 0\}|}{|L_u|},$$

where U is the set of users, and E_{ui} is a formalisation of the definition of interpretability as defined by the algorithm (see Eq. 3). MEP varies from 0 to 1, with 1 being the highest achievable score. This metric is very sensible to the definition of interpretability, therefore, an ill-defined formulation of E_{ui} can yield miss-leading results. As a case in point, we set $P_\tau = 0$ in Eq. 3, and run it on ML-100K. In this scenario, MEP would always be equal to one, thus, any recommendation will be considered fully explainable.

2) Model Fidelity: Model Fidelity belongs to the second category of metrics. It aims at evaluating explainability via the proxy. More specifically, it measures how accurate the interpretable proxy is in generating an 'explainable version' of the predictions of the black-box. Model fidelity is a typical measure adopted in post-hoc explanation approaches of black-box models [27]. Model fidelity can be defined as:

$$\mathsf{Fidelity} = \frac{|L \cap ProxPred|}{|L|},$$

where L is the set of recommendations generated by the black-box model, and $ProxPred$ is the set of the proxy predictions. Model Fidelity varies from 0 to 1, with 1 being a perfect match between the black-box model and the interpretable model. Fig. 4 shows the results of the fidelity of a proxy trained using the recommendations generated by several black-box algorithms that previously went through a precision-oriented tuning process (using the ML-1M dataset).

3) Explainability Score: The fidelity metric considers the number of interpretable recommendations that can be retrieved w.r.t. the original recommendations. However, a different metric definition is required to measure to what extent recommendations generated by a black-box algorithm can be explained using latent factors. To this end, we define a metric that measures the explainability of a recommended item in terms of the number of user interactions that can support its explanation.

More formally, given a user u and a recommendation i, we define the *Explainability Score* of u w.r.t. i as follows:

$$\mathsf{Expl}_{ui} = \frac{|I_u \cap K_l|}{|I_u|}, \tag{4}$$

where I_u is the set of interactions of user u, and K_l is the cluster containing i (with $\{K_1, K_2, \ldots, K_n\}$ representing clusters). Then, the explainability score of a recommendation list L_u is defined as follows:

$$\mathsf{Expl}_{L_u} = \frac{1}{|L_u|} \sum_{\forall i \in L_u} \mathsf{Expl}_{ui} \ . \tag{5}$$

Notice that the set of learned embeddings K_l used in Eq. 4 can be replaced by a set of item embeddings computed using a different learning method than clustering. For instance, one can be interested in considering the most similar item embeddings w.r.t. a given item recommendation, such as in the case of nearest neighbours.

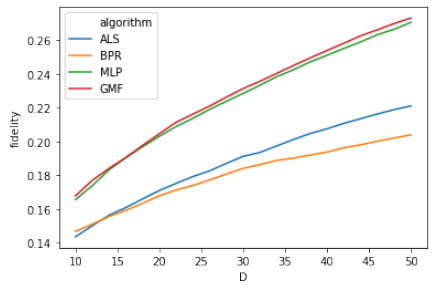
4) Other Metrics: recoXplainer also implements precision-oriented metrics, namely Hit Rate and nDCG, to assess the performance of black-box recommenders.

IV. recoXplainer in Practice

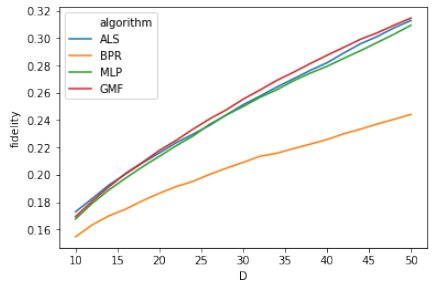
Fig. 5 shows the UML diagram of the main classes implemented in recoXplainer. The classes can be organised in five main categories according to the functionalities available in the library: data handling, model training, recommendation, explanation, and evaluation.

A. Data Handling

The `DataReader` class provides functionalities to manipulate a given dataset, and to convert it to a tensor representation (for GPU computation). ML libraries such as *pytorch* and *tensorflow* already implement pre-processing utilities, e.g., the `DataSet` class in *tensorflow*, or the `DataLoader` in *pytorch*. Nonetheless, they do not provide utilities for recommender systems. The `DataReader` class implements several functionalities: making user and item ids consecutive (`make_consecutive_ids_in_dataset`) to avoid sparsity when user and item ids are converted into embeddings; converting explicit feedback to implicit feedback (`binarize`), (sometimes) required by black-box algorithms; and splitting the dataset into train



(a) Association Rules (ARs)



(b) Nearest neighbours (kNN)

Fig. 4: Model Fidelity when increasing the number D of the predictions of the proxy model for the ML-1M dataset. The fidelity of the ARs and kNN proxies is low when only a few predictions are considered. The proxies become more 'faithful' to the black-box when more predictions are taken into account. However, this gain in fidelity penalises the interpretability of the proxies since their complexity increases as well (e.g., number of rules or neighbours).

and test sets by leaving out the n latest interactions (`split_leave_n_out`), a typical procedure implemented to train and evaluate recommendation algorithms [46]. These functionalities are shown in the following code's snippet.[8]

```
#Import the ml-1m dataset
data = DataReader(**cfg.ml1m)
#Re-arrange users' and items' Ids
data.make_consecutive_ids_in_dataset()
#binarize for implicit feedback
data.binarize(binary_threshold=1)
#Prepare train and test set
sp = Splitter()
train, test = sp.split_leave_n_out(data, frac=0.1)
```

B. Training a Model

Once the dataset has been loaded and prepared, training a recommender algorithm in recoXplainer is easy. The developer can instantiate one of the available models in recoXplainer (or first add a new one as we will see later), and train it. recoXplainer supports two categories of recommender algorithms: model-based (`ALSExplain`, `EMFModel`, and `ExplAutoEncoderModel`), and black-box (`ALS`, `BPR`, `GMFModel`, and `MLPModel`). Training one of these algorithms amounts to instantiate the corresponding class and call the `fit` method. The training time is of course dependent on the platform, but recoXplainer supports fast computation using GPU (if available). The following snippet shows an example of how to instantiate and train a GMF and an Extended Matrix Factorisation model respectively.

```
#training GMF
gmf = GMF(**cfg.model.als)
gmf.fit(train)

#training EMF
emf = EMFModel(**cfg.model.emf)
emf.fit(train)
```

C. Making Recommendations

Making recommendations to a set of users is achieved by instantiating the `Recommender` class. This class takes as input a trained model and the training set used. The class provides the method `recommend_all` to compute the top-N recommendations for each user. Alternatively, when one knows a specific user id, the `recommend_user` method can be used. To evaluate the recommendations, the `Evaluator` class provides a common interface to compute precision oriented metrics such as hit ratio (`cal_hit_ratio`) and nDCG (`cal_ndcg`). The evaluation is achieved by taking into account the set of generated recommendations and the test set obtained by the `Splitter` during the pre-processing.

```
#recommend using EMF
recommender = Recommender(train, emf)
recommendations = recommender.recommend_all()

#evaluate
eval = Evaluator(test)
eval.cal_hit_ratio(recommendations)
eval.cal_ndcg(recommendations)
```

D. Explanations

The `Explainer` class provides a common interface to explain the recommendations generated by any of the implemented recommendation algorithms (`explain_recommendations`). Explaining recommendations is achieved according to different explanation algorithms, e.g., through a model-based approach or a proxy (see Sections III-B and III-C). To this end, each algorithm re-implements the `explain_recommendation_to_user` method according to the algorithm specifics. For instance, `EMFExplainer` extracts the explanations intrinsically computed by `EMFModel`, whereas `ARPostHocExplainer` computes explanations by first mining the association rules and then extracting the explanations.

```
#generate explainable recommendations
expl = EMFExplainer(emf, recs, train)
expls = expl.explain_recommendations()

#explain GMF recommendations using association rules
ar_expl = ARPostHocExplainer(gmf, recs, train)
ar_expls = ar_expl.explain_recommendations()
```

[8] Classes are configured through a config file. Class parameters are available through `**cfg`.

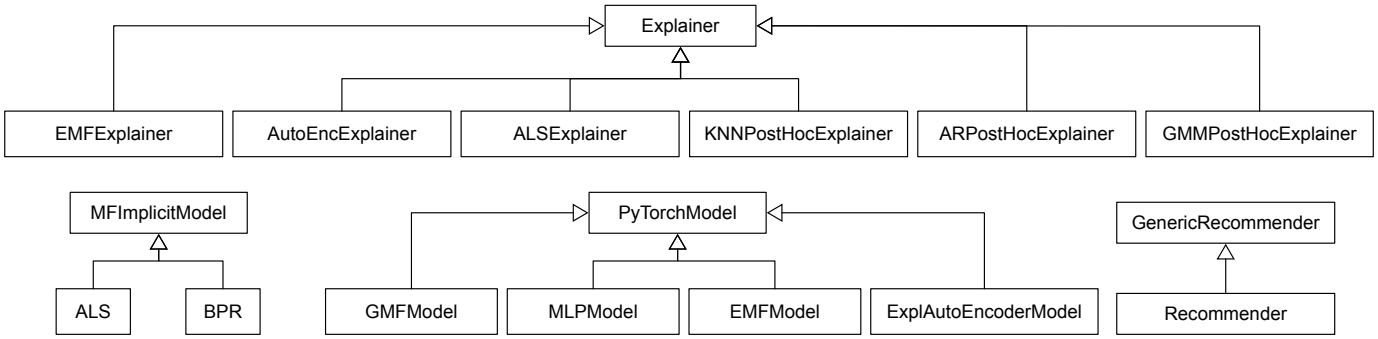


Fig. 5: UML diagram of the (main) classes implemented in recoXplainer.

E. Evaluating Explanations

The last step in the development pipeline of recoXplainer is the evaluation of the generated explanations using the metrics of mean explainable precision, fidelity, and explainability score. These metrics are implemented in the `ExplanationEvaluator` class.

```
#evaluating explanations using mean explainable
#precision (mep)
ex_eval = ExplanationEvaluator(train.num_user)
mep = ex_eval.mep(res, emf.explainability_matrix)
```

F. Extending recoXplainer

Another advantages of recoXplainer is that it can be extended in an straightforward way. The only requirement for adding a new recommendation algorithm it that it has to implement the `predict` method. This method takes as input a user and item id and compute a recommendation, e.g., using a `forward` step.

```
class MFModel(torch.nn.Module):

    def forward(self, user_indices, item_indices):
        #implement prediction
        ...

    def fit(self, dataset_metadata):
        #implement optimisation
        ...
        return;

    def predict(self, user_id, item_id):
        #return a recommendation
        pred = self.forward(user_id, item_id)
        return pred
```

V. Use-case: Explaining Black-box Recommendations using Latent Factors

Explaining recommendations generated by matrix factorisation algorithms is challenging. The latent factors used by Matrix Factorisation are indeed not associated with any semantics [47]. Thus, they are not directly applicable to draw any explanations. recoXplainer implements post-hoc explanation algorithms that can explain 'black-box' recommendations by taking into account the embeddings learned by a black-box algorithm. The approach followed amounts to cluster item embeddings by exploiting their similarity, and then to use them to draw item-style explanations. We ran an evaluation to compare the post-hoc explanation algorithms implemented in recoXplainer w.r.t. their capability to explain recommendations using item embeddings.

To perform the comparison, we first trained and optimised the black-box algorithms (see Section III-A) on a set of publicly available datasets. Then, we sought soft clusters of the learned items' embeddings. Finally, we generated item-style explanations over the entire set of predictions for each algorithm. In the following, we describe the datasets used, the experimental settings, and the results.

A. Datasets

In our experiments we used three benchmark datasets that are publicly accessible. The datasets vary in terms of domain, size, and sparsity:

- ML-100K [48] is a dataset containing 100K ratings in a liker scale (1-5) from 943 users on 1,682 movies. Each user rated at least 20 items.
- ML-1M [48] is a dataset containing 1 million ratings in likert scale (1-5), from 6,040 users on 4,000 movies. Each user in the dataset rated at least 20 items.
- Ciao-DVD [49] is a dataset collected from the Ciao website in December, 2013. The dataset was cleaned from duplicates by keeping only the latest rating in case of repeated scores. Besides, we kept only users that had at least 5 ratings. The resulting dataset contains 2,615 users, 14,313 items and a total of 49,599 ratings.

The recommendation algorithms implemented in recoXplainer are designed to perform on implicit feedback. Therefore, the above datasets were transformed in implicit signals by converting every rating to 1, while 0 would indicate the absence of interaction.

B. Experimental Settings

1) Evaluation Metrics: We tuned the algorithms to maximise the Hit Rate (HR) and the normalised Discounted Cumulative Gain (nDCG) on all the datasets [46]. We built the users' test set by extracting one relevant random item (R_u) from the entire set of rated items. Then, similar to [50], [32], we extracted 99 random items with

TABLE III: Algorithms' hyper-parameters maximising HR and nDCG found by the grid search.

Dataset	Alg.	LD	BS	LR	RT	Ep	HR	nDCG
ML-1M	ALS	64	N/A	N/A	0.005	90	81.30	58.20
	BPR	64	N/A	0.005	0	70	74.01	50.40
	GMF	16	512	0.0005	10^{-7}	20	78.91	53.58
	MLP	8	1024	0.005	10^{-7}	30	76.27	49.17
ML-100K	ALS	16	N/A	N/A	0.005	30	81.50	55.80
	BPR	16	N/A	0.005	0	100	78.70	52.50
	GMF	8	1024	0.005	10^{-7}	30	81.55	54.43
	MLP	16	512	0.005	10^{-7}	30	79.53	51.69
Ciao-DVD	ALS	16	N/A	N/A	0.005	40	54.80	37.64
	BPR	16	N/A	0.01	0.01	100	56.10	37.30
	GMF	8	512	0.005	10^{-7}	30	53.17	34.05
	MLP	64	512	0.005	10^{-7}	10	60.04	36.84

Note: LD - Latent Dimensions; BS - Batch Size; LR - Learning Rate; RT - regularisation term; Ep - Epochs; HR - Hit Rate; nDCG - normalised Discounted Cumulative Gain.

unknown relevance (NR_u) for each user u, where u did not have any previous interaction with these items. Finally, for each item i in R_u, we ranked the top-N items from the set $\{i\} \cup NR_u$, on which the evaluation is performed. The evaluation metrics were averaged over all the items in R_u and over all the users. All experiments were conducted according to this protocol.

2) Implementation and Hyper-parameters Tuning: We ran a grid search to seek the best hyper-parameters on all the algorithms (where applicable, see Section III-A) varying in: regularisation term = [0, .01, .001, .005, .0001, .0005, .0000001], learning rate = [.01, .001, .005, .0001, .0005], latent dimensions = [8, 16, 32, 64], number of epochs = [10-100], and batch size = [512, 1024]. The best hyper-parameters found are shown in Table III. Running times for training the models are available in the code repository of the project.[9]

3) Procedure: We trained ALS, BPR, GMF, and MLP on three datasets, namely ML-100K, ML-1M, and Ciao-DVD. After saving the embeddings of the trained models with highest HR and nDCG, we performed a dimensionality reduction transformation using UMAP [43].[10] Then, we clustered the embeddings using GMM, and we chose the one minimising the BIC as the optimal number of clusters. We ran GMM for each algorithm and dataset, and we varied the number of clusters from 1 to 500. The optimal number of clusters for all datasets and algorithms is shown in Table IV. Each experiment consisted of 20 runs. Computing the explanations for a given user takes less than 0.01 sec.

4) Evaluating Post-hoc Explanation Algorithms: We compared the three post-hoc explanation algorithms w.r.t. their capabilities of explaining recommendations using item embeddings. To this end, we used the explainability score metric defined in Eq. 4. In the case of GMM-based clustering, for each recommended item i in a recommendation list, we clustered the embeddings using GMM, and we used the clusters as K_l in Eq. 4. In the case of nearest neighbours, for each recommended item i in a recommendation list, we searched for the top k most similar item embeddings w.r.t. i, where the similarity is computed via cosine similarity. This set of similar item embeddings is used as items for K_l in Eq. 4. In the case of association rules, we first trained the association rules on the recommended items generated by the black-box algorithms. Then, for each user u, we kept any association rule $e \Rightarrow i$ such that e is an item with which u previously interacted, and i is an unseen or non-interacted item. We ordered the resulting subset by confidence. Finally, we kept the top k consequents of the rules as items for K_l.

[9]https://github.com/ludovikcoba/recoxplainer

[10]As mentioned previously, another technique for dimensionality reduction is t-SNE [44]. UMAP, however, is known to preserve not only the internal but also the external distances among the dimensions of the high-dimensional embeddings' space.

TABLE IV: Detailed view of explainability scores for GMM, Nearest Neighbours (NN) and Association Rules (AR) respectively. Taking into account users' item interactions according to the clusters obtained by GMM leads to higher explainability scores. All values for $\mathsf{Expl}_{\mathsf{GMM}}$, $\mathsf{Expl}_{\mathsf{NN}}$ and $\mathsf{Expl}_{\mathsf{AR}}$ are statistically significant. k is the highest average number of interactions per user given the clusters.

Data	Alg.	#cls.	$\mathsf{Expl}_{\mathsf{GMM}}$	k	$\mathsf{Expl}_{\mathsf{NN}}$	$\mathsf{Expl}_{\mathsf{AR}}$
ML-1M	ALS	72	**0.113**	33.15	0.081	0.014
	BPR	55	**0.139**	42.00	0.098	0.019
	GMF	53	**0.157**	44.85	0.121	0.021
	MLP	15	**0.330**	154.00	0.218	0.051
ML-100K	ALS	36	**0.214**	32.60	0.152	0.024
	BPR	26	**0.221**	33.30	0.133	0.025
	GMF	6	**0.542**	123.10	0.430	0.057
	MLP	22	**0.332**	35.30	0.160	0.023
Ciao-DVD	ALS	432	**0.069**	21.65	0.052	0.040
	BPR	169	**0.128**	24.40	0.064	0.029
	GMF	52	**0.284**	63.05	0.103	0.025
	MLP	70	**0.257**	23.95	0.051	0.032

C. Results and Discussion

Fig. 6 shows the explainability score (Eq. 5) mean values of users' recommendations computed by taking into account users' item interactions clustered using GMM, most similar users' item interactions (NN), and association rules (AR) respectively. As it can be observed, the explainability score calculated using the optimal clusters' number obtained by GMM ($\mathsf{Expl}_{\mathsf{GMM}}$) is always greater than the explainability scores computed using NN ($\mathsf{Expl}_{\mathsf{NN}}$) and association rules ($\mathsf{Expl}_{\mathsf{AR}}$) respectively. One possible reason behind this is that GMM is able to identify users' latent behaviours, and, consequently, it groups users' interactions in a more consistent way.

Table IV offers a more detailed view and comparison of the three explainability scores. In the table, k represents the number of most similar item embeddings per user used to calculate the explainability scores for the NN and AR proxy respectively. We determined the value of k as the highest average number of interactions per user given the clusters. We found that this value offered a good trade-off between how many items were clustered together and how many items a user could have interacted within the catalogue. This value is on average bigger than the

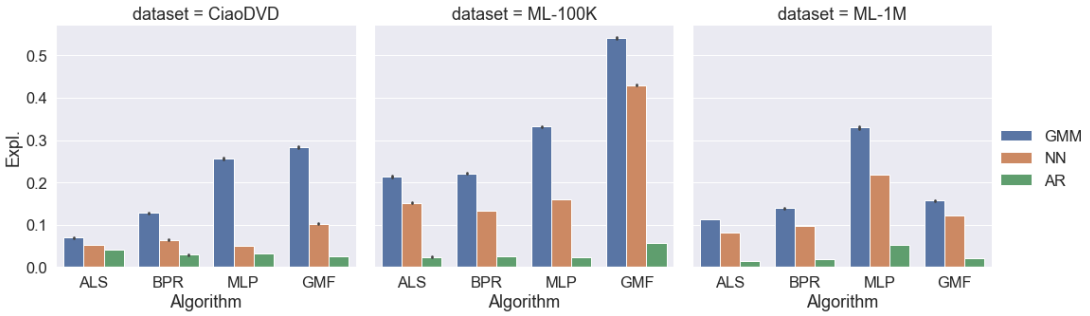


Fig. 6: Mean of explainability score for users' recommendations computed by taking into account users' item interactions clustered using GMM, AR and the NN most similar users' item interactions respectively (confidence intervals at 95%.)

number of elements (or past interactions) of a given user in any cluster, as we do not expect all the items of a user to be in the same cluster because (s)he might have performed unexpected and exploratory actions [51]. As a case of point, when training GMF on ML-100K, the learned embeddings were clustered in 6 distinct clusters. In this dataset we had an average of 59.5 ratings per user, and users had on average 23.4 interactions per cluster. However, we set k to 123.1 which was the highest average number of interactions per user given the clusters.

VI. Conclusion and Future Works

Although in the past few years RS research has been increasingly focused on the generation of explanations, to the best of our knowledge, there is no comprehensive library supporting and accelerating this type of work. Following the footsteps of other researchers in XAI [52], we implemented recoXplainer, a library supporting the development and assessment of explainable recommendation algorithms.

recoXplainer is an easy-to-use, unified, and extendable library that supports the development of eXplainable Recommender Systems. It does so by providing a development pipeline consisting of several steps, namely data preprocessing, training, recommendation, explanation and evaluation. It implements data manipulation techniques useful for recommender systems. It implements state-of-art collaborative filtering recommendation algorithms based on Matrix Factorisation, such as ALS, BPR, MLP, and GMF. It supports the generation of explainable recommendations by means of model-based and post-hoc explanation algorithms. The library is able to draw explanations for black-box recommendations by clustering learned item embeddings. recoXplainer also implements a standardised evaluation protocol for both precision- and explanation-oriented evaluation. The recoXplainer library will be beneficial for a wide range of practitioners in the field of eXplainable Recommender Systems.

Our library currently implements traditional explanation algorithms such as model-based and black-box techniques. We plan to integrate further approaches as well. For instance, enriching explanations with content-based information as proposed in [28], [29] will be an interesting addition to current functionalities. Crucially, one of the distinctive features of our library is its extensibility. This enables machine learning developers, and in particular, the recommender system community, to add more explanation techniques and algorithms to it.

Acknowledgment

A part of the work has been carried out at Alpha Health, Telefónica Innovación Alpha, Barcelona, Spain. The authors thank the reviewers for their valuable comments.

References

[1] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, and A. Holzinger, "Explainable ai: the new 42?" in *Int. Cross-Domain Conf. for Machine Learning and Knowledge Extraction*. Springer, 2018, pp. 295–303.

[2] N. Tintarev and J. Masthof, "Explaining recommendations: design and evaluation," in *Recommender Systems Handbook*. Boston, MA: Springer US, 2015, pp. 217–253.

[3] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *Foundations and Trends in Information Retrieval*, vol. 14, no. 1, pp. 1–101, 2020.

[4] M. D. Ekstrand, M. Ludwig, J. Kolb, and J. T. Riedl, "LensKit: a modular recommender framework," *Proc. of the fifth ACM conference on Recommender systems - RecSys '11*, p. 349, 2011.

[5] M. F. Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? A worrying analysis of recent neural recommendation approaches," in *RecSys 2019 - 13th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, Inc, 9 2019, pp. 101–109.

[6] L. Çoba and M. Zanker, "Replication and reproduction in recommender systems research - evidence from a case-study with the rrecsys library," in *Advances in Artificial Intelligence: From Theory to Practice*. Springer, 2017, pp. 305–314.

[7] S. Maksymiuk, A. Gosiewska, and P. Biecek, "Landscape of R packages for explainable artificial intelligence," *CoRR*, vol. abs/2009.13248, 2020.

[8] D. Rothman, *Hands-On Explainable AI (XAI) with Python*. Packt, 2020.

[9] L. Gianfagna and A. D. Cecco, *Explainable AI with Python*. Springer International Publishing, 2021.

[10] R. Confalonieri, T. Weyde, T. R. Besold, and F. M. del Prado Martín, "Trepan Reloaded: A Knowledge-driven Approach to Explaining Black-box Models," in *Proceedings of the 24th European Conference on Artificial Intelligence*, vol. 325. IOS press, 2020, pp. 2457–2464.

[11] J. M. Alonso, C. Castiello, L. Magdalena, and C. Mencar, *Explainable Fuzzy Systems: Paving the Way from Interpretable Fuzzy Systems to Explainable AI Systems*. Springer International Publishing, 2021.

[12] E. Mariotti, J. M. Alonso, and R. Confalonieri, "A framework for analyzing fairness, accountability, transparency and ethics: A use-case in banking services," in *IEEE International Conference on Fuzzy Systems 2021 (Fuzz-IEEE 2021)*, 2021, to appear.

[13] L. Coba, R. Confalonieri, and M. Zanker, "Recoxplainer: An extendible tookit for the developoment of explainable recommendation systems," in *Tutorials of the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021. [Online]. Available: http://www.inf.unibz.it/~rconfalonieri/aaai21/

[14] M. G. Core, H. C. Lane, M. Van Lent, D. Gomboc, S. Solomon, and M. Rosenberg, "Building explainable artificial intelligence systems," in *AAAI*, 2006, pp. 1766–1773.

[15] W. Swartout, C. Paris, and J. Moore, "Explanations in knowledge systems: Design for explainable expert systems," *IEEE Expert*, vol. 6, no. 3, pp. 58–64, 1991.

[16] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[17] R. Confalonieri, L. Coba, B. Wagner, and T. R. Besold, "A historical perspective of explainable artificial intelligence," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 1, 2021.

[18] R. Confalonieri, T. Weyde, T. R. Besold, and F. Moscoso del Prado Martín, "Using ontologies to enhance human understandability of global post-hoc explanations of black-box models," *Artificial Intelligence*, vol. 296, 2021.

[19] J. M. Alonso and G. Casalino, "Explainable artificial intelligence for human-centric data analysis in virtual learning environments," in *First International Workshop, HELMeTO 2019*, ser. Communications in Computer and Information Science, vol. 1091. Springer, 2019, pp. 125–138.

[20] A. Anguita-Ruiz, A. Segura-Delgado, R. Alcalá, C. M. Aguilera, and J. Alcalá-Fdez, "eXplainable Artificial Intelligence (XAI) for the identification of biologically relevant gene expression patterns in longitudinal human studies, insights from obesity research," *PLOS Computational Biology*, vol. 16, pp. 1–34, 2020.

[21] S. H. A. El-Sappagh, J. M. Alonso, F. Ali, A. Ali, J. Jang, and K. S. Kwak, "An Ontology-Based Interpretable Fuzzy Decision Support System for Diabetes Diagnosis," *IEEE Access*, vol. 6, pp. 37 371–37 394, 2018.

[22] B. Mittelstadt, C. Russell, and S. Wachter, "Explaining Explanations in AI," in *Proceedings of the Conference on Fairness, Accountability, and Transparency - FAT* '19*. New York, New York, USA: ACM Press, 2019, pp. 279–288.

[23] M. Bilgic and R. J. Mooney, "Explaining Recommendations: Satisfaction vs. Promotion," *Proc. of Workshop on the Next Stage of Recommender Systems Research at IUI'05*, pp. 13–18, 2005.

[24] G. Friedrich and M. Zanker, "A Taxonomy for Generating Explanations in Recommender Systems," *AI Magazine*, vol. 32, no. 3, p. 90, 2011.

[25] B. Abdollahi and O. Nasraoui, "Explainable Matrix Factorization for Collaborative Filtering," in *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*, 2016, pp. 5–6.

[26] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 426–434.

[27] G. Peake and J. Wang, "Explanation mining: Post hoc interpretability of latent factor models for recommendation systems," in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 7 2018, pp. 2060–2069.

[28] C. Musto, F. Narducci, P. Lops, M. de Gemmis, and G. Semeraro, "Linked open data-based explanations for transparent recommender systems," *International Journal of Human-Computer Studies*, vol. 121, pp. 93–107, 2019.

[29] C. Musto, M. de Gemmis, P. Lops, and G. Semeraro, "Generating post hoc review-based natural language justifications for recommender systems," *User Modeling and User-Adapted Interaction*, pp. 1–45, 2020.

[30] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 12 2008, pp. 263–272.

[31] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-thieme, "BPR : Bayesian Personalized Ranking from Implicit Feedback," *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, vol. cs.LG, p. 452–461, 2009.

[32] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering," in *Proc. of the 26th International Conference on World Wide Web - WWW '17*, 2017, pp. 173–182.

[33] B. Abdollahi and O. Nasraoui, "Using Explainability for Constrained Matrix Factorization," in *Proc. of the 11th ACM Conference on Recommender Systems - RecSys '17*, 2017, pp. 79–83.

[34] P. S. Haghighi, O. Seton, and O. Nasraoui, "An explainable autoencoder for collaborative filtering recommendation," *CoRR*, vol. abs/2001.04344, 2020.

[35] L. Coba, P. Symeonidis, and M. Zanker, "Personalised novel and explainable matrix factorisation," *Data and Knowledge Engineering*, vol. 122, pp. 142–158, 2019.

[36] F. M. Harper, F. Xu, H. Kaur, K. Condiff, S. Chang, and L. Terveen, "Putting Users in Control of their Recommendations," *the 2015 ACM conference on Recommender systems, RecSys 2015*, pp. 3–10, 2015.

[37] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Proc. of the 2nd ACM Conference on Electronic Commerce*, 2000, pp. 158—167.

[38] A. Papadimitriou, P. Symeonidis, and Y. Manolopoulos, "A generalized taxonomy of explanations styles for traditional and social recommender systems," *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 555–583, 5 2012.

[39] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work - CSCW '00*, 2000, pp. 241–250.

[40] C. Trattner and D. Jannach, "Learning to recommend similar items from human judgments," *User Modeling and User-Adapted Interaction*, vol. 30, no. 1, 2020.

[41] C. C. Aggarwal and C. K. Reddy, *Data Clustering: Algorithms and Applications*, 1st ed. Chapman & Hall/CRC, 2013.

[42] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 1, pp. 1:1–1:58, Mar. 2009.

[43] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," *CoRR*, vol. abs/1802.03426, 2018.

[44] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[45] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *CoRR*, vol. abs/1702.08608, 2017.

[46] G. Shani and A. Gunawardana, "Evaluating recommendation systems," *Recommender systems handbook*, pp. 257–298, 2011.

[47] O. Levy and Y. Goldberg, "Dependency-based word embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 2014, pp. 302–308.

[48] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015.

[49] G. Guo, J. Zhang, D. Thalmann, and N. Yorke-Smith, "Etaf: An extended trust antecedents framework for trust prediction," in *Proc. of the 2014 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining*, 2014, pp. 540–547.

[50] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proc. of the fourth ACM conference on Recommender systems - RecSys '10*. New York, New York, USA: ACM Press, 2010, p. 39.

[51] S. Vargas and P. Castells, "Rank and relevance in novelty and diversity metrics for recommender systems," in *Proceedings of the 5th ACM Vonference on Recommender Systems*. ACM Press, 2011, pp. 109–116.

[52] V. Arya, R. K. E. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, S. Mourad, P. Pedemonte, R. Raghavendra, J. Richards, P. Sattigeri, K. Shanmugam, M. Singh, K. R. Varshney, D. Wei, and Y. Zhang, "One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques," *CoRR*, vol. abs/1909.03012, 2019.