The Imitation Game: Algorithm Selection by Exploiting Black-Box Recommenders

Georgios Damaskinos, Rachid Guerraoui, Erwan Le Merrer, Christoph Neumann

The Imitation Game: Algorithm Selection by Exploiting Black-Box Recommenders

Georgios Damaskinos[1*], Rachid Guerraoui[1], Erwan Le Merrer[2], and Christoph Neumann[3]

[1] Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{georgios.damaskinos,rachid.guerraoui}@epfl.ch
[2] Université de Rennes, Inria, CNRS, IRISA, France
erwan.le-merrer@inria.fr
[3] InterDigital, Rennes, France
christoph.neumann@interdigital.com

Abstract. Cross-validation is commonly used to select the recommendation algorithms that will generalize best on yet unknown data. Yet, in many situations the available dataset used for cross-validation is scarce and the selected algorithm might not be the best suited for the unknown data. In contrast, established companies have a large amount of data available to select and tune their recommender algorithms, which therefore should generalize better. These companies often make their recommender systems available as black-boxes, i.e., users query the recommender through an API or a browser. This paper proposes RECRANK, a technique that exploits a black-box recommender system, in addition to classic cross-validation. RECRANK employs graph similarity measures to compute a *distance* between the output recommendations of the black-box and of the considered algorithms. We empirically show that RECRANK provides a substantial improvement (33%) for the selection of algorithms for the MovieLens dataset, in comparison with standalone cross-validation.

Keywords: Recommender algorithm selection · Black-box exploitation · Cross-validation · Graph similarity · Spearman ranking.

1 Introduction

The availability of open source recommendation algorithms and engines is appealing for startups or institutions that bootstrap their online services. A plethora of approaches, from collaborative filtering techniques to neural network based approaches are now at disposal[4], along with the deluge of research results that are thoroughly described (but not open-sourced). The users of online services generate a huge volume of data thus

* Corresponding author.

[4] https://github.com/grahamjenson/list_of_recommender_systems

triggering the advantage shift from solely leveraging a good item recommendation algorithm, to having access to *both* a good algorithm and a considerable amount of data for training or parameterizing it. In that context, it is clear that the big industrial players, have a steady and decisive advantage over potential newcomers on the market since they have both significant engineering work-forces and a large audience to get data from. Those established companies propose recommendation services, that interact with users through queries from browser-interactions or standard APIs. The recommendation algorithm acts as a black-box from the perspective of the user, and for potential observers such as those newcomers.

We present RECRANK, a method to sort a list of available recommendation algorithms based on their ability to generalize on unknown data. In stark contrast with cross-validation, this method exploits the recommendations of an established black-box recommender, and captures how well each of the available recommendation algorithms *imitates* the black-box recommender. We evaluate RECRANK and depict its superiority against classic cross-validation and an alternative ranking method. Our code is available [1].

Problem setting. Let $\mathcal{D}$ be the corpus of data that the recommender interacts with. This data includes tuples of the form $\langle u, i, l \rangle$ where a user u gives feedback l (implicit or explicit) for a certain item i. We split $\mathcal{D}$ into three parts: $\mathcal{D} = \{\mathcal{D}_a \cup \mathcal{D}_b \cup \mathcal{D}_u\}$.

On the one hand, an entity (e.g., a startup) targets to bootstrap a recommender. This entity has access to an *available dataset* $\mathcal{D}_a$, typically limited in size. This entity has also access to a set of open-sourced or in-house *candidate* recommendation algorithms, and needs to select the ones that will generalize best (i.e., provide good recommendations) on an *unknown dataset* $\mathcal{D}_u$.

On the other hand, a well-established recommendation service (e.g., IMDB, Spotify) enables *queries* to its recommender, typically through an API. We assume that the well-established recommender was trained using a private (i.e., available only to itself) dataset $\mathcal{D}_b$ (typically significantly larger than $\mathcal{D}_a$) and a private algorithm. This algorithm is a black-box to a user, and we denote it as $f(\mathcal{D}_b)$. The inputs to the black-box recommender are queries of the form $\langle u, i, l \rangle$ and the output is a ranked list of the top-N recommendations for user u, typically based on knowledge of the black-box recommender regarding u and i. For example, users make queries such as $\langle$user, song, click$\rangle$ (Spotify), $\langle$user, video, like$\rangle$ (YouTube) or $\langle$user, movie, rating$\rangle$ (IMDB) and get a ranked list of recommendations such as "Made for Alice" (Spotify), "Up next" (YouTube) or "Recommended For You" (IMDB), respectively.

Let $\mathcal{A}$ be the set of considered recommendation algorithms, along with their hyper-parameters. We define $P_{\boldsymbol{A}}(\mathcal{D}) \in \mathbb{R}$ as the performance measure of an algorithm $\boldsymbol{A} \in \mathcal{A}$ given a dataset $\mathcal{D}$ (e.g., the precision of $\boldsymbol{A}$ after splitting $\mathcal{D}$ into a training and validation set). Let $\mathcal{K}$ be *any* side knowledge. We consider $r(\mathcal{A}, \mathcal{D}, \mathcal{K})$ to be a ranking function producing a sorted array $[\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_{n-1}]$, such that $P_{\boldsymbol{A}_0}(\mathcal{D}) \geq P_{\boldsymbol{A}_1}(\mathcal{D}) \geq \cdots \geq P_{\boldsymbol{A}_{n-1}}(\mathcal{D})$. Each algorithm in $\mathcal{A}$ is solely trained and cross-validated using $\mathcal{D}_a$. We

define the optimal ranking as $r^* := r(\mathcal{A}, \mathcal{D}_a \cup \mathcal{D}_u, \emptyset)$, i.e., the perfect ranking for the available trained algorithms after taking into account the yet unknown data $\mathcal{D}_u$.

Black-box exploitation. We define the problem of exploiting a black-box recommender for algorithm selection as finding the ranking:

$$r := \arg\max_{r\prime} \rho(r^*, r\prime(\mathcal{A}, \mathcal{D}_a, f(\mathcal{D}_b))) \tag{1}$$

with ρ being the Spearman ranking correlation score (§3). The goal is to obtain side knowledge from the black-box in order to produce a ranking that gets closer to the optimal ranking. Our working hypothesis is that:

$$\rho(r^*, r(\mathcal{A}, \mathcal{D}_a, f(\mathcal{D}_b)) > \rho(r^*, r(\mathcal{A}, \mathcal{D}_a, \emptyset))) \tag{2}$$

i.e., there exists a gain (in comparison with cross-validation) from the information $f(\mathcal{D}_b)$ leaked from the black-box.

Noteworthy, the option of building a proxy recommender that always employs the black-box is not practical. Sending all the data to the black-box implies potential privacy violations as the user feedback (e.g., movie ratings) is forwarded to a third-party. From the system performance perspective, there are significant additional bandwidth costs and the service throughput is bounded by the query APIs limits (e.g., IMDB via RapidAPI has a limit of 1000 requests per day [2]). Therefore, the goal is to bootstrap the service and then only utilize the selected algorithm locally.

2 RECRANK

We introduce, RECRANK, a ranking function that exploits the outputs of a black-box (series of top-N recommendations) to compute a *distance* between each algorithm in $\mathcal{A}$ and a black-box recommender, under the assumption that the black-box generalizes better due to its larger dataset (we validate this assumption in §3). The final ranking of RECRANK follows the ascending order of this distance: the better an algorithm imitates the black-box, the higher its ranking.

RECRANK consists of two components as shown in Figure 1. REC2GRAPH transforms the output of a recommender into a graph data structure. The graph obtained from the outputs of the black-box is compared to the graph obtained from the outputs of each algorithm in $\mathcal{A}$, in order to compute a distance D with GRAPHDIST. The graph representation captures latent information about the recommender outputs (e.g., popularity of certain items among different subsets of users). This information is important for the performance of RECRANK, as we empirically show in §3 by using a baseline that directly compares the outputs of the two algorithms. RECRANK is shown in Algorithm 1, where $get\_rec(X, \mathcal{D}_q)$ returns the top-N recommendations of algorithm X given inputs in *query dataset* $\mathcal{D}_q$.

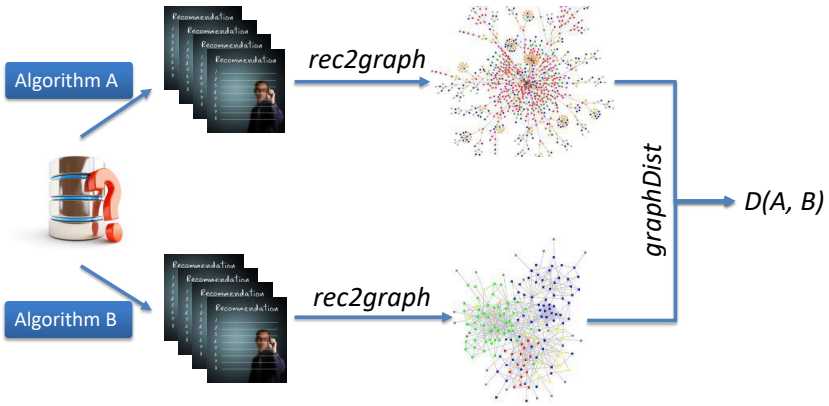

Fig. 1: Core components of RECRANK. Algorithm A is each algorithm in set $\mathcal{A}$ using the available dataset $\mathcal{D}_a$, while B is the black-box recommender. RECRANK builds a graph for each algorithm using REC2GRAPH and computes a distance between graphs using GRAPHDIST.

Algorithm 1: RECRANK

Input: Candidate algorithm set $\mathcal{A}$, black-box B, query set $\mathcal{D}_q$

G_b = REC2GRAPH ($get\_rec(B, \mathcal{D}_q)$)
for A ***in*** $\mathcal{A}$ **do**
  G_a = REC2GRAPH ($get\_rec(A, \mathcal{D}_q)$)
  $D(A, B)$ = GRAPHDIST (G_a, G_b)
  $distances.append(D(A, B))$
end
return $sort(distances)$

REC2GRAPH. This method transforms the output of the queried recommender into a graph. Building a graph from recommendations was recently shown to be interesting for several applications [12,7]. For each query $\langle u, i, l \rangle$ in the query dataset, we denote as $\mathcal{D}_q$, the recommender outputs a list of the top-N recommendations. REC2GRAPH constructs the graph according to the following rules.

- *Vertex* i: There exists a recommendation for item i and/or there exists a query for item i in $\mathcal{D}_q$ (e.g., a movie rating).
- *Edge* e_{ij}: Item j is at least in one of the top-N recommendation lists given as an output to a query for item i.
- *Weight* $w_{ij} = \frac{\sum_{e_{ij}} ranking\_score}{\sum_{e \in E} \sum_{e} ranking\_score}$

where $\sum_{e_{ij}} ranking\_score$ is the summation of the recommender output over all recommendations for item j triggered by a query for item i.[5]

The edge weight captures the fact that there are typically multiple recommendations between the same items. For example, a user might receive the same item in multiple top-N recommendations before she either clicks it or the recommender lowers its ranking and removes it from the top-N list. The denominator normalizes each weight in order for the graphs of different algorithms to be comparable, given that the scores of each recommender have different scales.

GRAPHDIST. In order to compare the two graphs, GRAPHDIST extracts a set of features for each graph (denoted as $\mathbf{X}_A$ and $\mathbf{X}_B$) and computes the distance between the two algorithms:

$$D(A, B) := \|\mathbf{X}_A - \mathbf{X}_B\| \tag{3}$$

GRAPHDIST extracts features that capture the state of the algorithm recommendations. For the features that involve distributions, we employ statistical values depending on whether we assume a Gaussian prior or not. We list a subset of these features below, and note that $\mathbf{X} \in \mathbb{R}^{31}$; the full set of extracted features is available in our code [1].

- *Number vertices and number of edges.* These illustrate the number of distinct recommendations.
- *Vertex in-degree.* This shows how polarized the recommendations are, i.e., how many popular items are recommended and how much is their popularity.
- *PageRank.* This indicates the PageRank centrality in the graph of the recommended items.
- *Eigenvector and betweeness centrality.* These centrality measures show how many items are the most central, i.e., most popular among the recommendations.
- *Closeness centrality.* This also captures the topological proximity of a given item to the others in the graph.
- *Assortativity.* This shows the connectivity between nodes of similar degree, i.e., how much popular items are connected to other popular items.
- *Shortest distances.* For each vertex, we compute the mean value of its shortest distances with each other vertex. We then average these mean values across all vertexes. This feature captures how close each item node is to the others.

The construction of GRAPHDIST makes RECRANK *interpretable*. Given the output of RECRANK, one can determine the contributing factor of each feature to this output. For example, if the *Vertex in-degree* feature has a very similar value for the candidate algorithm and the black box (i.e., contribution to the distance is minimal) comparing to the other features, then one can conclude that recommending popular items is an important factor for the final rank (output of RECRANK) of this candidate algorithm.

[5] If the recommender only outputs a top-N list, the output for each item is the rank (e.g., value $\in [1, 5]$ for top-5 outputs).

3 Evaluation

We study the performance of RECRANK on the *MovieLens* dataset[6] that consists of 100,000 ratings from 943 users on 1682 movies.

Table 1: Candidate recommendation algorithms. Information regarding the hyperparameters is available in our open-source repository [1].

Library	*Model-based*	*Memory-based*	*Baselines*
Librec	AOBPR, BIASEDMFlib, BPMFlib, EALSlib, LDAlib, LLORMAlib, NMFlib, SVDPPlib, PMF2lib PMFlib, RBMlib	KNNlib	MPOPlib, RANDlib
Surpriselib	NMF, PMF, SVD, SVDpp	KNNWithMeans	

Recommendation algorithms. We collect recommendation algorithms from open-source libraries: 14 algorithms from Librec[7] and 5 algorithms from SurpriseLib[8], as summarized in Table 1. We consider that a different implementation of the same algorithm constitutes a new candidate recommendation algorithm (e.g., KNNlib and KNNWithMeans): the output recommendations depend on various factors that differ among the two libraries (e.g., the formula for calculating the rating prediction). Additionally, we include two versions of the PMF algorithm (denoted as PMFlib and PMF2lib) with a different hyper-parameter tuning setup. Therefore, we illustrate that a different tuning (that can be the result of a difference in the resources for the A/B testing phase) leads to a different recommendation behavior (Table 2), thus to a different candidate recommendation algorithm.

Evaluation metrics. We now describe the metrics used for reflecting the performance of the candidate recommender and for demonstrating the efficacy of RECRANK.
Precision. We adopt this metric to test the accuracy of the recommendations. Given that $\mathcal{H}_u$ is the set of recommended items that were clicked by a user u (hits), and $\mathcal{R}_u$ is the set of items recommended to u, we denote the precision for u by $Precision_u$ and define it as follows.

$$Precision_u = \frac{|\mathcal{H}_u|}{|\mathcal{R}_u|} \tag{4}$$

[6] http://grouplens.org/datasets/movielens/
[7] https://www.librec.net/
[8] https://www.surpriselib.com/

The overall precision over the whole test set is the average over the precision values for all users in the test set. Note that a recommended item is considered as a *hit*, if the user rates that item anytime later than the time of the recommendation with a rating score larger than 50% of the maximum score [5].

Recall. We use this metric to capture the sensitivity of a recommender to the frequency of updates. Given that $\mathcal{C}_u$ is the set of items clicked by a user u, we denote the recall for u by $Recall_u$ and define it as follows.

$$Recall_u = \frac{|\mathcal{H}_u|}{|\mathcal{C}_u|} \tag{5}$$

The overall recall is the average over the recall values for all the users in the test set.

F1-score. We employ this standard metric to measure the recommendation accuracy in order to combine the precision and recall into a single score.

$$F1_u = 2 * \frac{Precision_u * Recall_u}{Precision_u + Recall_u}$$

Spearman correlation. We use this metric to evaluate the ranking quality of RECRANK. Moreover, we compute the Spearman rank-order correlation coefficient between the output ranking and the optimal ranking r^*, i.e., the ranking after evaluating the candidates on the dataset $\mathcal{D}_u$. A value of 0 indicates no correlation and a value of 1 an exact monotonic relationship; thus the higher the value of this metric, the better the performance. We compute this metric as follows:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}, \;\; 1 \leq i \leq C \tag{6}$$

where $d_i = rank(A_i) - optimal\_rank(A_i)$, is the difference between the two ranks for each candidate algorithm and n is the number of candidates.

The impact of an ordering mismatch does not depend on the rank of the mismatch. For example, the Spearman correlation between $\{1,2,3,4\}$ and $\{1,2,4,3\}$ is the same as $\{1,2,3,4\}$ and $\{2,1,4,3\}$. This ensures an equal weight for all the ranked candidates based on the fact that the entity that employs RECRANK can have access to any subset of these candidates.

Evaluation scheme. We replay the dataset, ordered by the timestamp, to capture the original temporal behavior. We split the dataset into $\mathcal{D}_a, \mathcal{D}_b, \mathcal{D}_u, \mathcal{D}_q$, according to Figure 2, and derive $\mathcal{D}_q$ by randomly sampling 1000 ratings from $\mathcal{D}_u$. We then train all the available recommendation algorithms on $\mathcal{D}_b$ and evaluate them on $\mathcal{D}_u$. Given our assumption regarding a black-box recommender with (a) significantly more data available for training and (b) superior algorithm (§1), we (a) make $\mathcal{D}_b$ significantly larger than $\mathcal{D}_a$ and (b) select the recommendation algorithm with the highest F1-score as the black-box. The remaining recommendation algorithms constitute our *candidate recommendation algorithms*. Finally, we re-train each candidate recommendation algorithm on the training set (first split of $\mathcal{D}_a$) and tune on the validation set (second split of

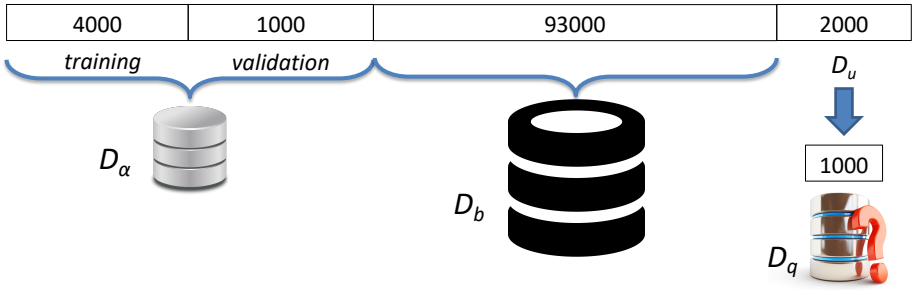

Fig. 2: Chronological data split for MovieLens. The first part is used for $\mathcal{D}_a$, the largest part for the black-box $\mathcal{D}_b$, and the last part is the yet unknown data $\mathcal{D}_u$.

$\mathcal{D}_a$, i.e., most recent 1000 ratings, based on the benchmark for evaluating stream-based recommenders [9]). Further information regarding our training setup (e.g., choice of hyperparameters) is available in our open-source repository [1].

Baselines. We compare RECRANK with the traditional ranking approach (i.e., cross-validation) along with a baseline algorithm, namely SETDISTRANK. SETDISTRANK computes the distance between algorithms directly from their outputs (i.e., without the REC2GRAPH and GRAPHDIST methods). The comparison with SETDISTRANK illustrates the importance of these two methods for the performance of RECRANK. SETDISTRANK computes the distance as follows:

$$D(A,B) = \frac{|\bigcap_{u\in\mathcal{U}} Recommended_u| + |\bigcap_{i\in\mathcal{I}} Recommended_i|}{2} \tag{7}$$

where $\mathcal{U}$ is the set of users and $\mathcal{I}$ is the set of items. $Recommended_u$ is the per-user recommendation set, i.e., the set of all the items recommended after a query from user u. $\bigcap_{u\in\mathcal{U}} Recommended_u$ denotes the intersection among all the per-user recommendation sets of the algorithms A, B. $Recommended_i$ is defined respectively.

Experimental results. We train the candidate recommendation algorithms (Table 1) by using the data scheme in Figure 2 and present the results in Table 2. First, we observe that the ranking derived from cross-validation on $\mathcal{D}_u$ (2nd column) is different than the optimal ranking (3rd column). Therefore, there is room for RECRANK to get a better ranking. We train all the algorithms with the black-box dataset $\mathcal{D}_b$ in order to select the black-box recommender. According to the results shown in the 4th column, this algorithm is LLORMAlib. The 5th column contains the results when training each algorithm on the training set and evaluating on the query set $\mathcal{D}_q$, which constitutes the comparison case for all presented competitors.

We compare the performance of the ranking algorithms by comparing the Spearman correlation with respect to the third column of Table 2. Table 3 depicts the results.

Table 2: F1 @ top-20 recommendations (and rank) of candidate algorithms (black-box algorithm is in bold).

	Standard cross-validation	Optimal ranking	Black-box ranking	Cross-validation on query set
Training set Evaluation set	$\mathcal{D}_a^{training}$ $\mathcal{D}_a^{validation}$	$\mathcal{D}_a^{training}$ $\mathcal{D}_u$	$\mathcal{D}_b$ $\mathcal{D}_u$	$\mathcal{D}_a^{training}$ $\mathcal{D}_q$
AOBPRlib	0.213 (4)	0.087 (5)	0.156 (5)	0.046 (4)
BIASEDMFlib	0.136 (9)	0.054 (11)	0.057 (11)	0.028 (13)
BPMFlib	0.618 (1)	0.419 (2)	0.408 (2)	0.374 (2)
EALSlib	0.197 (7)	0.084 (7)	0.138 (6)	0.046 (4)
KNNlib	0.206 (6)	0.085 (6)	0.126 (7)	0.046 (4)
KNNWithMeans	0.039 (19)	0.024 (18)	0.027 (15)	0.016 (18)
LDAlib	0.210 (5)	0.095 (4)	0.164 (4)	0.046 (4)
LLORMAlib	0.611 (2)	0.420 (1)	**0.412** (1)	0.377 (1)
MPOPlib	0.188 (8)	0.074 (9)	0.115 (8)	0.042 (8)
NMF	0.115 (13)	0.048 (15)	0.023 (16)	0.032 (11)
NMFlib	0.045 (18)	0.016 (19)	0.002 (19)	0.012 (19)
PMF	0.083 (16)	0.043 (16)	0.016 (18)	0.022 (17)
PMF2lib	0.133 (10)	0.076 (8)	0.058 (10)	0.037 (9)
PMFlib	0.106 (15)	0.050 (13)	0.030 (14)	0.027 (14)
RANDlib	0.083 (16)	0.039 (17)	0.023 (16)	0.024 (16)
RBMlib	0.565 (3)	0.398 (3)	0.403 (3)	0.358 (3)
SVD	0.123 (11)	0.049 (14)	0.059 (9)	0.033 (10)
SVDpp	0.120 (12)	0.052 (12)	0.051 (12)	0.031 (12)
SVDpplib	0.109 (14)	0.055 (10)	0.031 (13)	0.027 (14)

Table 3: Candidate algorithms ranking evaluation.

Ranking method	Spearman correlation
Standard cross-validation	0.53
Cross-validation on query set	-0.44
SETDISTRANK	0.68
RECRANK	**0.79**

The poor performance of the cross-validation on the query set $\mathcal{D}_q$ can be attributed to the chronological sorting of the ratings; the cross-validation becomes accurate if the validation and training set are closer in time and thus have significant overlap in the user sets. We then observe the substantial improvement (33%) that RECRANK provides in comparison with the standard cross-validation. The fact that SETDISTRANK also outperforms the cross-validation approach confirms that the information provided by the black-box recommender is valuable even without REC2GRAPH and GRAPHDIST.

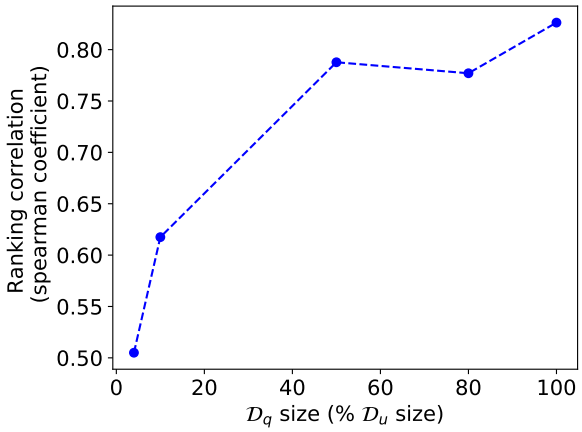

Fig. 3: Effect of the amount of queries on the performance of RECRANK.

Figure 3 depicts the effect that the size of the query set has to the performance of RECRANK. As the portion of $\mathcal{D}_q$ used to query recommenders increases, RECRANK exploits more information to compute better distance values, resulting in a better final ranking. The results of Table 3 have been obtained with a query set $\mathcal{D}_q$ by sampling 50% of $\mathcal{D}_u$ (1000 out of 2000 ratings shown in Figure 2).

4 Related Work

RECRANK proceeds by comparing the outputs of recommenders similar to benchmarking frameworks [11,16]. These frameworks enable the ranking of a set of recommendation algorithms according to some metric (e.g., F1-score) - similar to what RECRANK does based on $\mathcal{D}_a$ and the output of a black-box service. These frameworks do not allow to compare against the output recommendations of a black-box service that has been trained and tuned on an unknown set of data.

Evaluation of recommenders is very challenging when using offline datasets such as MovieLens. We plan to evaluate RECRANK with additional metrics such as propensity-weighting techniques [3], as well as employ alternative online methodologies and user studies [4]. We also plan to include additional baselines to the standard (i.e., leave-one-out) cross-validation, e.g., based on k-fold validation [10]. Nevertheless, the functionality of RECRANK is independent of the evaluation methodology.

Black-box analysis of recommendation algorithms has been also studied for the goal of algorithmic transparency. Xray [13] infers which user data (e.g. e-mails, searches) a recommender is using. Le Merrer et al. [12] propose a framework for detecting bias in recommended items. Hou et al. [7] proposed to operate random walks on the graph extracted from the recommendations to a user by the Amazon book platform. While

these related works try to understand how the remote recommender system works, they do not try build their own recommendation system (i.e. they do not try to benefit from the gained insights to tune a recommender) .

Imitation learning [8] and knowledge distillation [15,6] apply the broad idea of learning a policy, a reward function or prediction function by observing an expert system. In that sense, RECRANK is the first attempt to improve the selection of a recommender algorithm by imitating an expert recommender system, typically in production.

RECRANK targets to boost the recommendation quality given a limited amount of data, a problem also known as cold-start. Techniques that boost the quality of a specific recommender, e.g., transfer learning [14], or meta-learing [17] can be used for creating better candidate recommendation algorithms as input to RECRANK.

5 Discussion and Limitations

It is important to notice that, while RECRANK is the first tool to exploit black-box recommender systems for algorithm selection, we do not claim it to be a silver bullet. We discuss the limitations of our work in the following.

Black-box recommender bias. The recommendations of the black-box during the operation of RECRANK may be biased, i.e., not solely targeting the relevance to the users. For example, a commercial music recommender may promote songs from certain premium producers with the goal of direct financial gain. As a second example, the black-box may be a relatively new service that undergoes an A/B testing phase or some "exploration" phase (e.g., with random recommendations). In such cases, we expect RECRANK to output a biased ranking. Given that it is impossible for the user of RECRANK to determine whether the black-box is biased at a given time, we propose multiple deployments of RECRANK across different times. The similarities between the outputs of these deployments could help indicate the deployments that are biased; if the bias is not transient, multiple deployments will not be effective.

Advancing the state-of-the-art. The goal of RECRANK is not to directly create new recommendation algorithms but to select the most promising ones among a list of candidates. Nevertheless, this selection could be instrumental in developing a new state-of-the-art recommender that is an ensemble of multiple recommendation algorithms. For example, the cold-start component could be intentionally designed to mimic the cold-start behaviour of a well-established service; in that case RECRANK would be of great interest.

Local VS black-box data. A question that may arise in our problem setup is whether there are any constraints for the relation between the training data of the candidate algorithms (local) and the training data of the black-box. In our evaluation (§3), this relation is that the data has no overlap but comes from the same dataset, i.e., the user rating

behaviour follows the preference and behavioural drifts of the MovieLens dataset [5]. We plan to evaluate the performance of RECRANK under alternative relation scenarios (e.g., the data comes from different datasets) as part of our future work.

We expect the performance of RECRANK to degrade as the difference between the characteristics (e.g., how frequent are popular items rated and with what scores) of the local and black-box training data grows. Nevertheless, we highlight that the operation of RECRANK does not have any constraints regarding the training data as the query data is the same both for the candidate algorithms and the black-box. The smaller the relevance between the query data and the training data, the less the performance degradation of RECRANK due to differences in the training data. For example, if the query data only contains new users and new items, then RECRANK is essentially imitating the cold-start behaviour of the black-box and we thus do not expect differences in the training data to significantly degrade RECRANK performance.

The only requirement is for the input and output format of the query dataset to be compliant with the black-box. This requirement is easily satisfied given our generic form of input (tuples of the form $\langle u, i, l \rangle$) and output (ranking list of top-N recommendations) as mentioned in §1.

6 Concluding remarks

We present RECRANK, an algorithm that facilitates recommender algorithm selection, traditionally made solely via cross-validation. Our initial results show a promising potential for this tool. Nevertheless, these results do not constitute an in-depth experimental validation and there is work towards measuring the true potential of RECRANK. We also plan to compare REC2GRAPH with alternative methods for transforming the recommender outputs into graphs [12,7]. Finally, We propose RECRANK as one instance of an algorithm that exploits a black-box recommender; we believe this proposal will motivate related works for finding other good performing alternatives with a similar goal.

References

1. RECRANK source-code. https://github.com/gdamaskinos/RecRank
2. IMDB via RAPIDAPI query limit. https://rapidapi.com/blog/how-to-use-imdb-api/
3. Agarwal, A., Wang, X., Li, C., Bendersky, M., Najork, M.: Offline comparison of ranking functions using randomized data. In: REVEAL (2018)
4. Beel, J., Langer, S.: A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In: TPDL. pp. 153–168. Springer (2015)
5. Damaskinos, G., Guerraoui, R., Patra, R.: Capturing the moment: Lightweight similarity computations. In: ICDE. pp. 747–758. Ieee (2017)
6. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
7. Hou, L., Liu, K., Liu, J.: Navigated random walks on amazon book recommendation network. In: Complex Networks (2018)
8. Hussein, A., Gaber, M.M., Elyan, E., Jayne, C.: Imitation learning: A survey of learning methods. CSUR **50**(2), 21 (2017)
9. Kille, B., Lommatzsch, A., Turrin, R., Serény, A., Larson, M., Brodt, T., Seiler, J., Hopfgartner, F.: Overview of clef newsreel 2015: News recommendation evaluation lab (2015)
10. Košir, A., Odić, A., Tkalčič, M.: How to improve the statistical power of the 10-fold cross validation scheme in recommender systems. In: RepSys. pp. 3–6. ACM (2013)
11. Kowald, D., Kopeinik, S., Lex, E.: The tagrec framework as a toolkit for the development of tag-based recommender systems. In: UMAP. pp. 23–28. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3099023.3099069, http://doi.acm.org/10.1145/3099023.3099069
12. Le Merrer, E., Trédan, G.: The topological face of recommendation: models and application to bias detection. In: Complex Networks (2017)
13. Lécuyer, M., Ducoffe, G., Lan, F., Papancea, A., Petsios, T., Spahn, R., Chaintreau, A., Geambasu, R.: Xray: Enhancing the web's transparency with differential correlation. In: USENIX Security Symposium. pp. 49–64 (2014)
14. Pan, W., Xiang, E.W., Liu, N.N., Yang, Q.: Transfer learning in collaborative filtering for sparsity reduction. In: AAAI (2010)
15. Polino, A., Pascanu, R., Alistarh, D.: Model compression via distillation and quantization. In: ICLR (2018)
16. Said, A., Bellogín, A.: Rival: a toolkit to foster reproducibility in recommender system evaluation. In: RecSys. pp. 371–372. ACM (2014)
17. Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., Larochelle, H.: A meta-learning perspective on cold-start recommendations for items. In: NIPS. pp. 6904–6914 (2017)