Evaluating Recommender System Stability with Influence-Guided Fuzzing

David Shriver,[1] Sebastian Elbaum,[1] Matthew B. Dwyer,[1] David S. Rosenblum[2]
[1]Department of Computer Science, University of Virginia, Charlottesville, VA, USA
[2]Department of Computer Science, National University of Singapore, Singapore
{dlshriver, selbaum, matthewbdwyer}@virginia.edu, david@comp.nus.edu.sg

Abstract

Recommender systems help users to find products or services they may like when lacking personal experience or facing an overwhelming set of choices. Since unstable recommendations can lead to distrust, loss of profits, and a poor user experience, it is important to test recommender system stability. In this work, we present an approach based on inferred models of influence that underlie recommender systems to guide the generation of dataset modifications to assess a recommender's stability. We implement our approach and evaluate it on several recommender algorithms using the MovieLens dataset. We find that influence-guided fuzzing can effectively find small sets of modifications that cause significantly more instability than random approaches.

Introduction

Recommender systems filter information to help users make decisions when lacking personal experience or knowledge (Resnick and Varian 1997) or when the set of choices is overwhelmingly large (Herlocker et al. 2004). We see them everywhere, from e-commerce sites such as Amazon, news article recommendation at the New York Times, and movie recommendation by Netflix, to guides for venture capital opportunities (Zhao, Zhang, and Wang 2015), safe combinations of pharmaceutical drugs (Chiang et al. 2018), and collision avoidance actions for aircraft (Julian et al. 2016).

The *robustness* of these systems—their ability to make good recommendations in the presence of noise—can have significant impact on the end goals of both providers and users (O'Mahony et al. 2004), such as the profitability of an e-commerce site, or the safety and effectiveness of recommended drug combinations. In this work, we focus on a particular dimension of robustness called *stability*, which measures how recommendations change when the system is trained on modified data, regardless of the recommendation quality. Intuitively, a system lacks stability when there exist a number of changes to the dataset that can have an effect on the recommendations that is disproportionate to the amount of change. The lack of stability is problematic in that it causes recommendation inconsistencies that can lead to loss of user trust and limits system adoption (Adomavicius and Zhang 2012).

Determining the stability of a system, however, is extremely challenging, due in part to the size of the input space that needs to be explored. As an example, the dataset we use later in the paper has 943 users, 1682 items, and 100,000 ratings on a scale from 1 to 5. A naïve stability assessment approach that simply adds a single random rating to that dataset would likely overestimate the system stability since finding one addition among the 8 million possible (943 users $\times$ 1682 items $\times$ 5 ratings) that can cause instability is very unlikely. Similarly, an exhaustive search for instability-inducing modifications seems infeasible, especially when considering multiple modifications (i.e., adding k new ratings to our dataset means exploring a space of at least $10^{(6*k)}$ potential modifications) or when considering datasets that can contain hundreds of millions of data points. Furthermore, even if generating all modifications was feasible, the system must then be retrained on the modified dataset, which is often an even more expensive process.

Our approach to this problem is to exploit the assumptions underlying the design of recommender algorithms as to how the relationships among items and users *influence* recommendation behavior. For instance, a user-based recommender algorithm assumes that *influence between users* is important and therefore employs similarity measures between users in order to determine influence. Unfortunately, these influence relationships are encoded by the training process in complex, internal data structures that are typically difficult to access and interpret. Nevertheless, we can infer characteristics of the influence relationships learned by the system by analyzing the recommendations it produces and the ratings it was trained on. For instance, we can estimate the influence of a user based on the number of items rated by that user that are recommended to other users.

Based on this insight, our approach first *infers* a model of influence underlying a recommender system, and then it performs *fuzzing* on the training dataset in order to assess the system's stability. Fuzzing is a widely-used automated software testing technique that generates invalid, unexpected or other directed random inputs to cost-effectively test the stability a system (Miller, Fredriksen, and So 1990)[1].

As part of its output, the approach includes examples of changes and types of changes to the dataset that cause in-

[1]A recent survey on fuzzing can be found at (Liang et al. 2018).

stability. These examples can then be used by developers to improve the design of the recommender algorithm, adjust its parameters, or even discard and replace the algorithm entirely, in an effort to meet the stability requirements for a given dataset. Stability results also can inform model training tasks, such as when additional preprocessing of the data is needed, when regularization needs to be modified, or when to perform retraining during deployment.

Our overall contributions are: (1) We approximate influence relationships learned by recommender systems from the recommendations they produce and their training data. (2) We use these models for heuristic-based, influence-guided fuzzing of recommender system stability, to search for dataset modifications having a disproportionate effect on computed recommendations. (3) We evaluate our approach on several recommender algorithms with the MovieLens dataset. We find that our influence-guided fuzzing heuristics are much more effective than randomly generating modifications. For instance, for one recommender system tested, 10 rating modifications generated with our influence-guided heuristic caused 44% of users to have their top-ranked item removed from their recommendations after retraining, a 40000x increase over random.

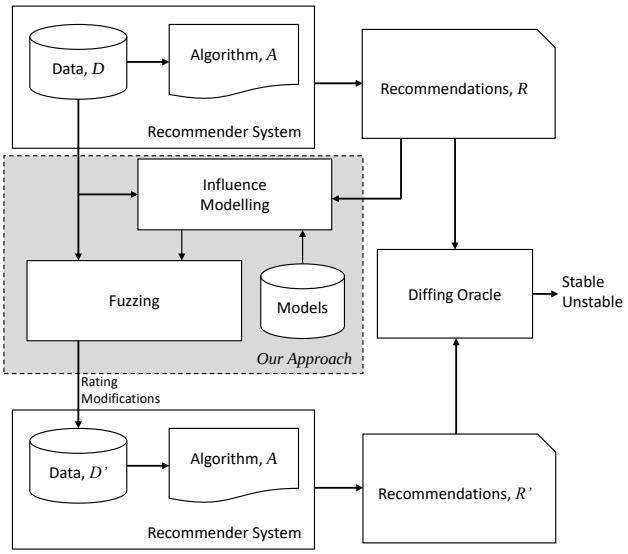

Figure 1: Diagram of our approach for generating modified datasets based on inferred influence models.

Related Work

O'Mahony et al. describe two aspects to robustness, *accuracy* and *stability* (O'Mahony et al. 2004). *Accuracy*, in regard to robustness, is a measure of the recommendation quality after changes are made to the dataset, while *stability* is a measure of how different the recommendations are after a change is made. Gunawardana and Shani consider robustness to be "the stability of the recommendation in the presence of fake information" (Gunawardana and Shani 2015). Adomavicius and Zhang take a slightly different view of robustness and stability, defining stability as the consistency of recommendations over some period of time under the assumption that any new ratings added to the dataset completely agree with the prior recommendations (Adomavicius and Zhang 2010; 2012). Using this definition of stability, they find that the stability of a recommender system does not necessarily correlate with the accuracy of the system. In this work, we define stability as the consistency of recommendations, given any small set of rating modifications, and we relax the assumption of Adomavicius and Zhang that new ratings must align with the past.

In prior work, recommender system robustness was commonly evaluated in the context of attacking recommender systems. Shilling attacks, or profile-injection attacks, add user profiles to a recommender system with crafted sets of item ratings in order to increase or decrease the position of some item in the recommendations of all users (O'Mahony et al. 2004; Lam and Riedl 2004). Gunes et al. provide a comprehensive survey on shilling attacks against recommender systems (Gunes et al. 2014). Our approach differs from adversarial and shilling attacks in a couple significant ways. First, our fuzzing approach automatically infers influence models from the execution of the recommender system. These models aim to approximate portions of the training dataset that are driving most recommendations. Second, our fuzzing approach uses those models to generate rating modifications, which are much smaller than adding whole new profiles, users, or items to a dataset. Whereas shilling attacks add new user profiles to promote or demote targeted items, our approach generates small sets of rating modifications for existing users and items to provide a general assessment of the stability of a recommender system. Additionally, because shilling attacks add new users to a system, they require adding many ratings for each of those new users. For example Lam and Riedl introduce between 25 and 100 users with 3404 ratings each to a dataset of almost 1,000,000 ratings (a percent change of between 8% and 34%) (Lam and Riedl 2004). In contrast, our goal is to identify small sets of modifications (under 1% change) that cause significant change to the recommendations produced by the system.

Approach

In this section we present our approach to assessing the stability of recommender systems by fuzzing the dataset used to train the recommender. A diagram of our approach is shown in Figure 1. A recommender system consists of an algorithm, A and a dataset, D. The algorithm uses the dataset to compute a model of user preferences. The recommender system employs the learned model to compute recommendations for users of the system. Our approach infers an influence model from a dataset D and a set of recommendations R. The inferred influence model is then used for fuzzing D to produce a new dataset D'. Using the modified dataset we train a new recommender system and generate recommendations R'. The recommendations R and R' are then compared using a differential oracle to assess the instability of the original recommender system.

To simplify our presentation, we begin by defining recommender systems in terms of users, items, ratings, and rankings. Then we describe how to approximate recommender

Table 1: Influence Model Functions

Name	Definition
`User`	$I^{\mathcal{U}}(u) = \lvert\{u' \mid rank(i,u') \neq \bot \wedge rating(u,i)\}\rvert$
`Item`	$I^{\mathcal{I}}(i) = \lvert\{u \mid rank(i',u) \neq \bot \wedge rating(u,i)\}\rvert$
`Attribute`	$I^{\mathcal{A}_\mathcal{I}}(a) = \lvert\{(u,i) \mid rank(i,u) \neq \bot \wedge a \in attr(i)\}\rvert$
`Rating`	$I^{\mathcal{R}}(r) = \lvert\{i \mid rank(i,u) \neq \bot \wedge \lvert r - avg\{rating(u',i)\}\rvert \leq \epsilon\}\rvert$

systems with inferred models of influence. Next, we explain how we can use these models for influence-guided fuzzing of modifications to a dataset. Finally, we define differential oracles for recommender stability, and discuss the assumptions made by our approach.

Defining Recommender Systems

Let $\mathcal{U}$ be a finite set of users, $\mathcal{I}$ a finite set of Items, and $\mathcal{R}$ a possibly infinite, ordered set of rating values. Without loss of generality, we assume that $\mathcal{R} \subset \mathbb{Z}_{>0}$ (within some interval $[l, h]$).

This framework extends to recommender systems in general. For instance, in a system that recommends pharmaceutical drugs that are safe to take with a given prescription (Chiang et al. 2018), the items to recommend are drugs, the user is a prescription, and the ratings encode whether the prescription included a drug.

A dataset D defines the partial function $rating\colon \mathcal{U} \times \mathcal{I} \to \mathcal{R}$ that captures how users rate items, with $rating(u,i) = \bot$ if u has not rated i. In addition to defining $rating$, D characterizes each $u \in \mathcal{U}$ by attributes drawn from the set $\mathcal{A}_\mathcal{U}$, and each $i \in \mathcal{I}$ by attributes drawn from the set $\mathcal{A}_\mathcal{I}$ through functions $attr\colon \mathcal{U} \to 2^{\mathcal{A}_\mathcal{U}}$ and $attr\colon \mathcal{I} \to 2^{\mathcal{A}_\mathcal{I}}$, respectively. A user or item may be characterized by multiple attributes.

Given D and a parameter k for computing top-k rankings, a recommender algorithm Q computes a partial function $rank_{k,Q}\colon \mathcal{I} \times \mathcal{U} \to [1,k]$ for $k \leq |\mathcal{I}|$, where $rank_{k,Q}(i,u) = \bot$ if item i is not ranked for user u.

For every $u \in \mathcal{U}$ there are a number of ranked items, $k_u \leq k$, and the projection of rank onto users, $rank_{k,Q}(u)\colon \mathcal{I}_{k_u} \to [1,k_u]$, is a bijection, where $\mathcal{I}_{k_u} \subseteq \mathcal{I}$, $|\mathcal{I}_{k_u}| = k_u$.

In what follows, we drop the subscript Q from $rank_{k,Q}$ since Q is typically apparent from the context, and we drop the subscript k since k is typically a parameter to Q.

Inferring Influence

Recommender systems generate recommendations based on some notion of influence between aspects of the dataset, such as users ($\mathcal{U}$), items ($\mathcal{I}$), or attributes ($\mathcal{A}_\mathcal{I}$ or $\mathcal{A}_\mathcal{U}$). For instance a user-based algorithm computes users that are influential to a given user, based on the similarity of ratings between pairs of users. Users with high similarity in their ratings are considered more influential to each other than users with low similarity in their ratings. In an item-based recommender algorithm, influence occurs between items, and is computed based on the similarity of ratings between items.

Because influence is used to produce recommendations, we conjecture that we can approximate the influence of various aspects (users, items, ratings, or attributes) of a dataset for a recommender system, based on the data used to train the recommender and its recommendations. By observing the relationship between some aspect in the dataset and some aspect in the recommendations, such as whether user u rated an item that is recommended to user u', we can build an approximate model of the overall influence. In general, we approximate influence scores with a function:

$$I^{\mathcal{A}} : \mathcal{A}^n \to \mathbb{R}, \text{ where } \mathcal{A} \in \{\mathcal{U}, \mathcal{I}, \mathcal{R}, \mathcal{A}_\mathcal{I}, \mathcal{A}_\mathcal{U}\} \quad (1)$$

This function maps aspects of the dataset to a real value representation of the influence of that aspect. In this work, we focus on influence functions over single types of aspect ($n = 1$), however richer forms of influence are possible. Using the computed influence scores, we can approximate the internal influence model of a recommender system as a list of aspects and their associated inferred influence score.

In this work we present four models of influence, which are shown in Table 1. These models of influence capture the four major aspects of a recommender system: users, items, ratings, and attributes. The first column of Table 1 gives a short descriptive name to the influence model, and the second provides a definition in terms of the recommender system elements defined earlier. The superscript of the influence function signifies the type of aspect over which this influence applies. The influence function computes a score for an aspect based on the dataset and the rankings produced by the recommender system of interest. We describe each of these models in more detail below.

User influence measures the impact of a user u on all other users. A user, u is considered to have impacted another user u' if user u has provided a rating for an item appearing in the top-k items for user u'. This model of influence captures the intuition that if a user affects the recommendations of many users, then that user likely has high influence.

Item influence measures the impact that an item i has on recommendations, by counting the number of users who are recommended at least one item, and have rated item i. This model of influence captures the intuition that items rated by many users are likely to be more influential.

Attribute influence measures the impact of an item attribute on the recommendation of items with that attribute. An attribute, a is considered to be influential if many recommended items have attribute a. This model captures the intuition that if many recommended items have a similar attribute, then that attribute must be significant.

Rating influence measures the impact of rating values on recommendation. A rating value r is considered to be more impactful if more recommended items have an average rating within some value ϵ of r. This model captures the intuition that if many recommended items have similar average rating values, then that rating value is more influential. In other words, items with average ratings near r are more likely to be recommended if r has high influence. While, the

notion of influence for users, items, and attributes is intuitive, the notion of rating influence is less intuitive. However, we see in Section that it can be unusually effective.

Influence-Guided Fuzzing

Using influence models, such as those defined above, we can define fuzzing heuristics which produce a set of modifications to the original dataset that are likely to cause a recommender system to exhibit unstable behavior.

We allow three basic types of dataset modifications: add, remove, and change. *Add* inserts a new rating to the dataset for a user u, item i, and value r if no rating value existed for u and i in the original dataset. *Remove* deletes an existing rating for a user u and item i from the data set. *Change* deletes an existing rating for a user u and item i and inserts a new rating with value r for user u and item i.

We define a modification fuzzing heuristic as a function:

$$M : I \times D \times \mathbb{Z} \rightarrow 2^{\mathcal{M}} \tag{2}$$

$\mathcal{M} \in \{A, R, C\}$, where A is a set of additions, R a set of removals, and C a set of changes. As per Equation 2, a heuristic function takes an influence function, a dataset, and a number of modifications to be made, and outputs a set of additions, a set of removals, and a set of changes, or some subset of these.

We define a small sample of possible fuzzing heuristics, shown in the top section of Table 2. These heuristics were chosen by keeping the modification type constant (as the Add modification type) and varying the influence type, and by keeping the influence type constant (as user influence) and varying the type of modification. This is not a complete listing of possible heuristics but is designed to cover a variety of influence models, given the limited space available. We discuss the intuition for each of these heuristics below. Each row of Table 2 is a fuzzing heuristic. The first column of each row provides a short name which we use as an identifier, and the second column provides a description of the heuristic where the **bolded** and *italicized* words specify the modification and influence types, respectively.

The influence-guided fuzzing heuristics defined here are given three letter names based on how they operate. The first letter is based on the type of modification they produce, where A is for add, C is for change, and R is for remove modifications. The second letter is the area of the influence model they select aspects from. An M means that the heuristic chooses the most influential aspect and an L means it chooses the least influential. The third letter specifies the type of influence used by the heuristic. User influence is specified by a U, item influence is an I, attribute influence is an A, and rating influence is an R.

AMU. The AMU fuzzing heuristic adds random ratings to the user with the highest influence score and is defined as:

$$\texttt{AMU} : I^{\mathcal{U}} \times D \times \mathbb{Z} \rightarrow 2^{A} \tag{3}$$

Items are selected from the set of items not yet rated by that user and both items and rating values are selected uniformly at random. The intuition behind this heuristic is that by adding ratings to the highest influence user, we may be able to cause a new item to be recommended to many other users, or cause a previously recommended item to stop being recommended for many users.

Table 2: Fuzzing Heuristics

Name	Description
AMU	**Add** a rating with random value to a random item for the most influential *user*.
RMU	**Remove** a rating from a random item for the most influential *user*.
CMU	**Change** the rating of a random item to the low value for the most influential *user*.
ALI	**Add** a rating with a random value to the least influential *item*.
AMR	**Add** a random rating value to an item with an average rating near the most influential average *rating*.
AMA	**Add** a rating with low value to a random user for an item with the most *attribute* influence.
ARAND	**Add** ratings to random users and items.
RRAND	**Remove** random ratings.
CRAND	**Change** random ratings to the low value.

RMU. The RMU fuzzing heuristic removes random ratings from the most influential user and is defined as:

$$\texttt{RMU} : I^{\mathcal{U}} \times D \times \mathbb{Z} \rightarrow 2^{R} \tag{4}$$

Items are selected uniformly at random from the set of items rated by that user. The intuition behind this heuristic is that by removing ratings from the highest influence user, that user can lose influence, causing the recommendations of other users to change.

CMU. The CMU fuzzing heuristic changes random ratings by the user with the highest influence score to have the lowest possible rating value. This heuristic is defined as:

$$\texttt{CMU} : I^{\mathcal{U}} \times D \times \mathbb{Z} \rightarrow 2^{C} \tag{5}$$

Items are selected uniformly at random from the set of items rated by that user. The intuition behind this heuristic is that by changing ratings of the highest influence user, we may cause a recommended item to no longer be recommended.

ALI. The ALI fuzzing heuristic adds random ratings to the item with the lowest influence score. We define the function for this heuristic as:

$$\texttt{ALI} : I^{\mathcal{I}} \times D \times \mathbb{Z} \rightarrow 2^{A} \tag{6}$$

Users are selected from the set of users that have not rated the least influential item. Both users and rating values are selected uniformly at random. The intuition is that adding ratings to the least influential item may cause it to become influential, and cause recommendations to change.

AMR. The AMR fuzzing heuristic adds new random ratings to items with average rating values near the most influential rating value. We define the function for this heuristic as:

$$\texttt{AMR} : I^{\mathcal{R}} \times D \times \mathbb{Z} \rightarrow 2^{A} \tag{7}$$

Items are selected uniformly at random from the set of items with an average rating within 0.05 of the most influential rating. This selects items very close to the influential rating. We experimented with several values of ϵ, and we chose 0.05 as the value that generally produced the most instability. Users are selected uniformly at random from the set of users that have not rated the selected item. The rating value to add is selected uniformly at random. The intuition behind this heuristic is that by adding random ratings to items with an influential average rating, we can move the average away from the influential rating value to reduce the likelihood that the item will be recommended.

AMA. The AMA heuristic adds low valued ratings to items with the highest aggregate attribute influence and is defined as:

$$\texttt{AMA} : I^{\mathcal{A}_\mathcal{I}} \times D \times \mathbb{Z} \rightarrow 2^A \tag{8}$$

Because items can have multiple attributes, this heuristic aggregates the influence scores of all attributes of an item by summing all of their influence scores. We use a slightly modified attribute influence function in order to negatively weight low influence attributes:

$$I_2^{\mathcal{A}_\mathcal{I}}(a) = 2 * |\{a' | I^{\mathcal{A}_\mathcal{I}}(a) \geq I^{\mathcal{A}_\mathcal{I}}(a')\}| - |\mathcal{A}_\mathcal{I}| \tag{9}$$

The item with the highest aggregate attribute influence is selected for modification. Users are selected uniformly at random from the users who have not rated the selected item. The intuition behind this heuristic is that by adding low ratings to items with many highly influential attributes, we can decrease the influence of those attributes, causing items with those attributes to not be recommended.

Differential Stability Oracles

To test the stability of recommender systems, we define oracles as Boolean predicates of the form:

$$f\{d(rank(u), rank'(u)) \mid u \in \mathcal{U}\} < \delta \tag{10}$$

This predicate ensures that a function f applied to the set of distances between users' rankings using the original and modified datasets is below a specified threshold δ. In this work, we consider f to be a function that computes the average distance. We can compute the distance between rankings using a variety of metrics, depending on which types of change we wish to be sensitive to.

Many possible metrics may be used to compute the distance between rankings, which cover a range of important aspects of change, including the order of the top-k rankings, the inclusion of items in the top-k recommendations, and the exclusion of important items in a user's top-k.

In this work, we introduce the *TopOut* measure to quantify significant ranking changes. This measure checks whether the top item in the unmodified ranked list has dropped out of the top-k rankings when using the modified dataset. We assume that the top ranked item is likely to be one of the most difficult to change. Therefore, if this item is not in the ranked list after modifications are added to the dataset, then the recommendations should be considered to have significantly changed. We define this measure as:

$$TopOut(R_u, R'_u) = \begin{cases} 1 & r_1 \notin R'_u \\ 0 & otherwise \end{cases} \tag{11}$$

R_u is the set of items recommended to user u using the recommender trained on the original dataset and R'_u is the set of items recommended to user u after modifications are made to the dataset. The item r_1 is the top-ranked item ($rank(r_1, u) = 1$) in the ranked list of u. This metric is sensitive only to the top ranked item for a user, which generally has the highest likelihood of being preferred by the user. To change the value of this metric, the top item in the original ranking must not be included in the new top-k ranked list.

Practical Considerations and Usage

For this approach to be applicable, certain preconditions must be met. First, the developer must have read and write access to the full dataset that was used to train the recommender system under test. This is a reasonable as testing will generally be performed by a developer or a dedicated tester of the system, and will thus have access to this data. Second, we assume that additions, removals, or changes are realistic modifications that can occur to a dataset which is the case for most recommendation systems that evolve over time. Third, in defining differential oracles for recommender stability, we assume that identifying a threshold of acceptable instability δ is possible either by using standard or historical measures. While choosing a threshold value is domain-, application-, and even dataset-dependent, an empirical process can be used to find appropriate values. For instance, a space of threshold values for the initial system can be explored to help developers set expectations, and then that baseline threshold can be used as the system or the dataset evolves.

When those preconditions are met, the approach can provide not only better stability estimates than existing approaches but also concrete dataset changes that may cause significant instability. Historical stability estimates can then be used by developers to assess the evolution of their recommender from a robustness perspective and to pinpoint departures from established trends. Developers can also use the concrete dataset changes to determine how best to adjust the existing algorithm underlying the recommendation system to make it more robust to variations in the dataset. Last, the stability estimates and the dataset changes can guide data cleansing procedures (by for example increasing or decreasing the impact of certain records or aspects) and assist in the definition of data retraining policies after deployment.

Study

We carried out a study to explore the cost-effectiveness of our fuzzing heuristics, and we also explore how the type of influence and type of modifications used by a fuzzing heuristic affects its ability to find instability-inducing modifications. More specifically, we answer the following questions:

RQ1: How effective and efficient are the different influence models in guiding generation of instability-inducing modifications?

RQ2: How can we fuzz a recommender system if the algorithm is a black box and the type of underlying influence is unknown?

Study Design

We evaluated our approach to fuzzing recommender systems by applying the selected influence-guided fuzzing heuristics to several recommender systems using a variety of recommender algorithms and a movie ratings dataset. We discuss these choices in more detail below.

We evaluated the fuzzing heuristics for 3 sizes of modification set: 1, 10, and 100. These sizes correspond to changes of 0.001, 0.01, and 0.1 percent of the dataset respectively and were chosen to be much smaller than the size of the dataset. With less than 0.1% of the ratings being modified, we would expect the recommendations to exhibit proportionally small amounts of change. For each heuristic, size, and recommender system (described next), we generated 100 modification sets. We then trained each recommender on the modified dataset and generate Top-10 recommendations for every user. The log of the average $TopOut$ metric over the 100 generated modification sets are plotted in Figure 2.

Recommender Algorithms For this study we selected four recommendation algorithms. The algorithms were chosen to represent a variety of recommender techniques ranging from memory-based to model-based, and from content-based to collaborative filtering. An additional criteria that guided our algorithm selection was that a candidate algorithm had to work with the MovieLens dataset (Harper and Konstan 2015), either because it was a part of the Lenskit framework or because its adaptation to that framework required only minor data wrangling. We describe each of the four chosen algorithms below.

User-User. The User-User algorithm is a memory-based collaborative filtering algorithm introduced by the GroupLens project (Resnick et al. 1994). The algorithm computes user similarity scores based on user rating vectors. In this work, we use the implementation of User-User provided by the Lenskit framework (Ekstrand et al. 2011), which computes user similarity using vector cosine similarity (Breese, Heckerman, and Kadie 1998).

Item-Item. The Item-Item algorithm is a model-based collaborative filtering algorithm. Similarity scores are precomputed for all pairs of item rating vectors. The predicted value of an item is estimated by aggregating the ratings of the most similar items (Sarwar et al. 2001; Deshpande and Karypis 2004). We again use the Item-Item implementation provided by the Lenskit framework (Ekstrand et al. 2011).

FunkSVD. FunkSVD is a model-based collaborative filtering algorithm that uses stochastic gradient descent to learn a matrix factorization (Funk 2006). In this work, we use the implementation of FunkSVD provided by the Lenskit framework (Ekstrand et al. 2011), which learns 25 latent features.

LightFM. LightFM is a hybrid recommender algorithm that uses both ratings and item attributes to build a recommender model (Kula 2015). In our study, we use the Python implementation of the algorithm provided by its author[2]. It is the only algorithm of the four that explicitly uses item attributes when building a recommender model. Using the MovieLens dataset, we provide the genres as item attributes.

MovieLens 100k Dataset We use the dataset released in 1998 from the MovieLens recommendation system. The dataset is available as a group of tab separated files, containing 100 thousand integer ratings (from 1 to 5) collected from 943 users over a period of 8 months on 1682 movies. Each user has rated at least 20 items. More details about the data collection process and the dataset itself are available at https://grouplens.org/datasets/movielens/100k/ (Harper and Konstan 2015).

Treatments To study the effectiveness of influence-guided fuzzing, we will compare the defined heuristics from Table 2 against three random baselines: `ARAND`, `RRAND`, and `CRAND`. The `ARAND` heuristic adds new ratings to the dataset by randomly selecting users, items, and rating values. The `RRAND` heuristic removes ratings from the dataset by randomly selecting users, items, and rating values. The `CRAND` heuristic randomly changes ratings in the dataset to have the lowest rating value by randomly selecting users and items. These baselines were chosen to control for the effect of the influence model on the user or item choice. By comparing the influence-based heuristics to the random baseline of the same modification type (e.g., `CRAND` and `CMU`), we can evaluate the effectiveness of using influence models to guide fuzzing of recommender systems.

Differential Oracle Our evaluation reports the results from using the $TopOut$ distance metric ($d = TopOut$), and we use a differential oracle that computes the average distance over all users ($f = avg$).

Methodology To evaluate efficiency, we measure how long it takes each heuristic to generate a modification. We generate 100 modification sets of size 1, 10, and 100 for each heuristic and report the average time to generate a modification set of each size.

To compare the effectiveness of influence-guided fuzzing heuristics to random fuzzing, we compute the average $TopOut$ instability across 100 generated modification sets for each configuration of recommender and modification set size. We explored modification sets of size 1, 10, and 100, which are small compared to the original dataset. Changing 100 ratings would change only 0.1% of the dataset used in this study. We arbitrarily chose to stop with a maximum modification set size of 100. We then compute the ratio of the $TopOut$ instability induced by each heuristic to the instability induced by the random fuzzing heuristic of the same modification type. A ratio greater than 1.0 indicates that a heuristic outperforms random. A value of 10 means that influence-guided fuzzing caused 10 times more instability than random.

Threats to Validity The recommendations produced by a system are dependent on both the recommender algorithm as well as the dataset used to train the system. Because we only evaluate our approach with a single dataset, our results may not generalize to other datasets. That said, the approach is general, and this dataset is commonly used and includes user and item attributes for content-based recommendation.

[2]https://github.com/lyst/lightfm

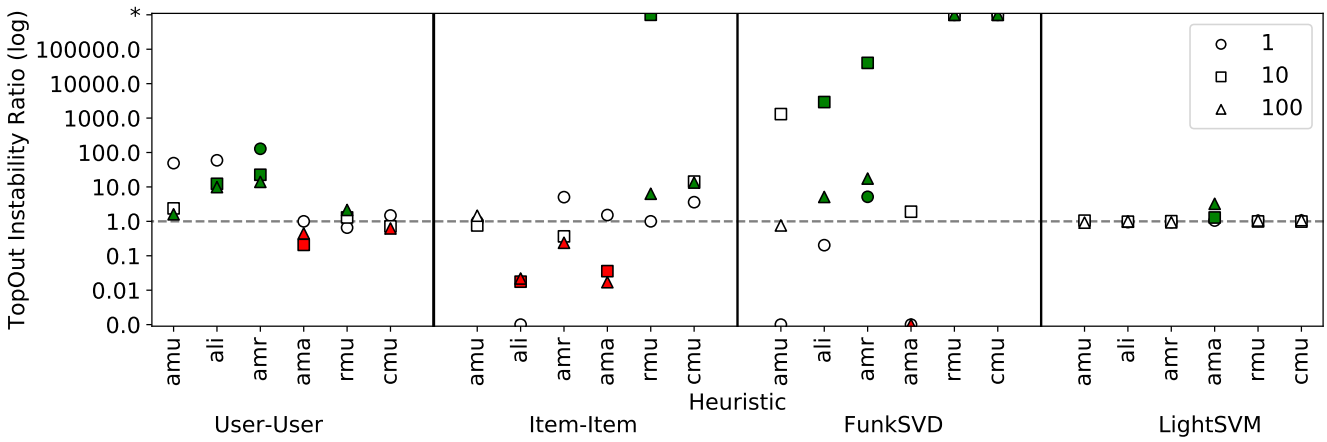

Figure 2: Ratio of mean $TopOut$ instability for each influence-guided fuzzing heuristic compared to the mean $TopOut$ instability for ARAND using sets of 1, 10, and 100 Add-type modifications. Heuristics that perform significantly better or worse than the baseline (with $p < 0.05$), are presented in color.

In this work we look at only a small subset of the possible influence functions and modification fuzzing heuristics. There are many other heuristics that can generate instability-inducing modifications or better approximation functions for computing influence. We do show that heuristics can be used to generate instability-inducing modifications more effectively than random methods.

There are many possible metrics for computing the distance between recommendations. In this work, we only report the results of $TopOut$ since the results of using the $Jaccard$ and AOD measures were similar.

RQ1:Effectiveness and Efficiency of Fuzzing

Effectiveness of Heuristics *Overall, influence-guided fuzzing is effective at generating smaller sets of modifications that induce higher amounts of instability than random, succeeding in 3 of 4 recommendation algorithms, often providing order of magnitude improvements.*

Figure 2 shows the ratio of the $TopOut$ instability for each influence-guided fuzzing heuristic, compared to its corresponding random baseline. The x-axis contains the heuristic and recommender pairs, and the y-axis represents the ratio on a log scale. In cases where the baseline caused a $TopOut$ instability of 0.0, the ratio could not be computed, so we plot these points at the value $*$ on the highest value of the y-axis. We also perform a statistical pairwise comparison of the instability induced by each influence-guided fuzzing heuristic to the instability induced by ARAND using Welch's t-test. Heuristics that perform significantly better or worse than the baseline (with $p < 0.05$), are presented in color in Figure 2.

For the User-User recommender system, 3 of the influence-guided fuzzing heuristics perform significantly better than random, for at least one size of modification set. In fact, for modification sets of size 1, the AMR heuristic induced instability 128x better than random. Unlike User-User, single modifications were not as effective on FunkSVD, and only performed better than random for AMR. However, sets of 10 and 100 modifications performed significantly better than random when using ALI and AMR. When generating sets of 10 modifications, AMR induced instability over 40000x greater than ARAND. With LightFM, 3 of our influence-guided heuristics (AMU, ALI, and AMR) performed similarly to random. However, AMA performed significantly better than the random baseline for sets of 10 and 100 modifications, causing 1.3x and 3.2x greater instability, respectively. While, Add-type modifications tended to be ineffective for Item-Item, the RMU and CMU heuristics outperformed their corresponding baselines by 6.3x and 13.4x greater instability, respectively.

Fitting of Influence Model *The effectiveness of a given influence-guided fuzzing heuristic depends on how closely the inferred influence model used by the heuristic approximates the actual influences used by the recommender algorithm.* For instance, in Figure 2, we see that the AMA heuristic is not effective at fuzzing instability-inducing modifications. This is likely because AMA uses an inferred attribute influence model, while the three algorithms for which it does not perform well do not rely on any attribute data. However, for a recommender that does rely on item attribute information, such as LightFM, the AMA heuristic is effective at generating instability-inducing modifications.

Similarly, the rating influence based AMR heuristic is unusually effective at causing significant changes to recommendations for the User-User and FunkSVD recommenders. We believe this is because influential rating values for these recommenders are on the extreme ends of the rating scale. The most influential rating value for the User-User recommender is 1.0, while the influential rating value for the FunkSVD recommender is 5.0. For the other two recommenders, for which AMR performed poorly, the most influential rating value was closer to the middle of the scale.

Effectiveness by Modification Type For three of the recommender systems explored in this work, *Add-type mod-*

Table 3: Time (in seconds) to generate a modification set.

Heuristic / Set Size	1	10	100
ARAND	0.539	0.608	2.020
AMU	0.600	0.673	1.692
ALI	0.607	0.801	1.809
AMR	0.458	0.580	1.582
AMA	3.126	3.053	3.753

ifications were the most effective at influencing change. For these recommenders, at least one heuristic performed significantly better than random for modification sets of size 10 and 100. For example, the AMA heuristic is effective at fuzzing instability-inducing modification sets for the LightFM recommender. For both the User-User and FunkSVD algorithm, the AMR heuristic is effective at generating single modifications that induce significant change in the recommendations. However, Add-type modifications were not effective at inducing instability for the Item-Item recommender.

Remove-type modifications were more effective than random for sets of 100 modifications for User-User, Item-Item, and FunkSVD. This modification type was also effective for the Item-Item recommender when fuzzing sets of 10 modifications.

Overall, Change-type modifications affected the fewest recommender systems, but when they were effective, they caused large amounts of change. Change-type modifications were only effective on two algorithms, Item-Item and FunkSVD, and only for sets of 100 modifications. However, they were able to produce higher levels of instability in the Item-Item recommender than any other modification type. We conjecture that this is because the value of ratings in these systems is more influential than the relationships between users or items.

Efficiency of Heuristics We briefly evaluate the efficiency of each heuristic to generate a set of modifications. The mean time (in seconds) over the course of 100 trials for each Add-type heuristic and set size are reported in Table 3. The cost of fuzzing heuristics for Change and Remove modifications were similar.

The time to generate modification sets for random and influence-based fuzzing heuristics are comparable for AMU, ALI, and AMR, and within a factor of 6 in the worst case for AMA[3]. But even in the worse case, the timing are practically the same when considering the time it takes to train the recommender system on the new data (over 2 minutes when training on the small MovieLens 100k dataset).

RQ2: How effective is fuzzing in the absence of algorithm information?

We explore whether influence-guided fuzzing is an effective technique for testing stability when the recommender algorithm under test is a black box, and the sorts of influence used by the system are unknown.

[3]AMA takes longer due to the repeated aggregation of the influence of multiple attributes for every item (see Equation 9).

Table 4: Mean $TopOut$ instability for APORTFOLIO

Configuration Recommender,M	ARAND	APORTFOLIO
User-User,10	0.002969	**0.035599**
User-User,100	0.031294	**0.257794**
Item-Item,10	0.002375	0.000244
Item-Item,100	0.026681	0.008653
FunkSVD,10	0.000011	**0.130721**
FunkSVD,100	0.053446	**0.768982**
LightFM,10	0.011474	0.012450
LightFM,100	0.010933	**0.017678**

To accomplish that, we introduce a portfolio approach to generating modifications. We assume a budget of n modifications and a user specified portfolio of m influence-guided fuzzing heuristics. For each modification in the modification set, we use a round robin approach to select one of the m fuzzing heuristics to generate the modification.

We evaluate this approach with a portfolio of the four influence-guided fuzzing heuristics studied previously. We will call this hybrid heuristic APORTFOLIO. Using budgets of $n = 10$ and $n = 100$, we compare the effectiveness of APORTFOLIO to ARAND using each of the four recommenders studied above.[4] The average $TopOut$ instability for APORTFOLIO is reported in Table 4. Values in this table are **bold** if they are significantly better than random.

For the User-User, FunkSVD, and LightFM recommenders, the portfolio approach is able to find modification sets that cause more instability than ARAND. APORTFOLIO was not effective for the Item-Item recommender system. This is likely because none of the Add-type heuristics performed well on the Item-Item recommender. *Overall, using a portfolio of influence-guided fuzzing heuristics is effective at generating modification sets that cause significant changes in recommendations.*

Conclusions and Future Work

We present an approach that uses influence-guided fuzzing to test the stability of recommender systems. We build on the insight that influence models can be inferred from the recommendations produced by a recommender system and the dataset used to train that system. We define a sample of fuzzing heuristics that use inferred influence models to generate modifications to the original dataset that induce instability in the recommendations. To test instability we define a test oracle that uses a threshold of acceptable instability, measured as the distance between users' recommendations. Our study shows that influence-guided fuzzing is effective at finding small sets of modifications that cause significantly more instability than random approaches. In future work,

[4]We do not use $n = 1$ for this approach because it is equivalent to selecting a single heuristic.

we will perform a more exhaustive search of this space to identify what features of heuristics and influence models are most effective. For example, using hybrid forms of influence over multiple types of aspects in the dataset, and fuzzing heuristics that take advantage of multiple types of influence.

Acknowledgments

This work has been supported in part by National Science Foundation awards #1526652 and #1617916, and by A*STAR SERC PSF grant 152120008.

References

Adomavicius, G., and Zhang, J. 2010. On the stability of recommendation algorithms. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, 47–54. New York, NY, USA: ACM.

Adomavicius, G., and Zhang, J. 2012. Stability of recommendation algorithms. *ACM Trans. Inf. Syst.* 30(4):23:1–23:31.

Breese, J. S.; Heckerman, D.; and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 43–52. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Chiang, W.; Shen, L.; Li, L.; and Ning, X. 2018. Drug recommendation toward safe polypharmacy. *CoRR* abs/1803.03185.

Deshpande, M., and Karypis, G. 2004. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.* 22(1):143–177.

Ekstrand, M. D.; Ludwig, M.; Konstan, J. A.; and Riedl, J. T. 2011. Rethinking the recommender research ecosystem: Reproducibility, openness, and lenskit. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, 133–140. New York, NY, USA: ACM.

Funk, S. 2006. Netflix update: Try this at home.

Gunawardana, A., and Shani, G. 2015. *Evaluating Recommender Systems*. Boston, MA: Springer US. 265–308.

Gunes, I.; Kaleli, C.; Bilge, A.; and Polat, H. 2014. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review* 42(4):767–799.

Harper, F. M., and Konstan, J. A. 2015. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* 5(4):19:1–19:19.

Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; and Riedl, J. T. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* 22(1):5–53.

Julian, K. D.; Lopez, J.; Brush, J. S.; Owen, M. P.; and Kochenderfer, M. J. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 1–10.

Kula, M. 2015. Metadata embeddings for user and item cold-start recommendations. In Bogers, T., and Koolen, M., eds., *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems*, volume 1448, 14–21. CEUR-WS.org.

Lam, S. K., and Riedl, J. 2004. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, 393–402. New York, NY, USA: ACM.

Liang, H.; Pei, X.; Jia, X.; Shen, W.; and Zhang, J. 2018. Fuzzing: State of the art. *IEEE Transactions on Reliability* 67(3):1199–1218.

Miller, B. P.; Fredriksen, L.; and So, B. 1990. An empirical study of the reliability of unix utilities. *Commun. ACM* 33(12):32–44.

O'Mahony, M.; Hurley, N.; Kushmerick, N.; and Silvestre, G. 2004. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.* 4(4):344–377.

Resnick, P., and Varian, H. R. 1997. Recommender systems. *Commun. ACM* 40(3):56–58.

Resnick, P.; Iacovou, N.; Suchak, M.; Bergstrom, P.; and Riedl, J. 1994. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186. New York, NY, USA: ACM.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International Conference on World Wide Web*, 285–295. New York, NY, USA: ACM.

Zhao, X.; Zhang, W.; and Wang, J. 2015. Risk-hedged venture capital investment recommendation. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, 75–82. New York, NY, USA: ACM.