The Knowledge Engineering Review, Vol. 33, e15, 1–19. © Cambridge University Press, 2018
doi:10.1017/S0269888918000218

Language independent recommender agent

OSMAN YUCEL and SANDIP SEN

Tandy School of Computer Science, The University of Tulsa, 800 S Tucker Dr, Tulsa, OK, USA;
e-mail: osman-yucel@utulsa.edu, sandip-sen@utulsa.edu

Abstract

This paper presents a new 'Language Independent Recommender Agent' (LIRA), using information distributed over any text-source pair on the Web about candidate items. While existing review-based recommendation systems learn the features of candidate items and users' preferences, they do not handle varying perspectives of users on those features. LIRA constructs agents for each user, which run regression algorithms on texts from different sources and builds trust relations. The key advantages of LIRA can be listed as: LIRA does not require reviews from target users, LIRA calculates trust values based on prediction accuracy instead of social connections or rating similarity, LIRA does not require the reviews to come from the same community or peer user group. Since ratings of the reviewers are not necessary for LIRA, we can collect and use reviews from different sources (web pages, professional critiques), as long as we know the corresponding item and source of that text. Since LIRA does not combine text from different sources, texts from different sources are not required to be in the same language. LIRA can utilize text from multiple languages, as long as sources are consistent with their language usage.

1 Introduction

The demand for accurate recommendation systems has dramatically increased with the growth of user engagement in varying e-Commerce portals and is facilitated by the rapid proliferation of personal and usage data on the Internet. An average user does not have enough time or resources to systematically explore the content on e-Commerce sites to identify items of interest. Hence, there is a pressing need for automated and personalized assistance for exploring content at online stores and service outlets.

The recommendation systems must be able to learn about both the preferences of the users as well as the properties of the contents to provide accurate personalized recommendations to the users. Both of these requirements present complex challenges.

The first challenge is collecting information about content properties. The information about the items used by current recommendation systems is generally provided by the creator of the content or by a few expert individuals. This source bias limits the usefulness of the available information for the purpose of recommendation. This is because effective recommendation systems need to leverage subjective information from the perspective of individual users of the system. To illustrate this problem, consider the following recommendation request by a user on the *GoodReads* website:

> *'Id Like to read a page-turner book with a strong female lead! i like YA, with humore and some supernatural aspects to it.' (sic)*

The second critical challenge is learning the preferences of the users: humans are not perfectly rational beings and will not necessarily be aware of or be able to effectively articulate their own preferences. There are a number of issues that prevent the recommendation systems from accurately modeling the users.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Studies have, for example, exposed paradoxical transitive preferences from users (such as $A \succ B \succ C \succ A$) when asked to compare a set of items (Davis, 1958). In real life when people try to make decisions about candidate items we might be interested in, they usually base their decision about the candidates on what other people say about those items. They do not only consider what has been said about the items but the source of that information is also important. For example, a movie critique (C) saying 'good' for a movie might not mean anything for target user (T) while a blogger (B) using the same word for a movie might be very important for decision-making process of T. Another example about real life is, in some cases, it is not important if the source of information has liked the item or not. For example, if T likes every movie which was defined as a 'World War II (WW2)' movie by B, T will probably like the next movie which was defined as 'WW2' by B, regardless of B's liking of that movie.

Existing systems interpret the reviews of users by relating them to their ratings. This introduces two problems: (i) since the rating system can differ among different web pages (different ranges, thumbs up/down, etc.), using reviews from different web pages is challenging and (ii) those systems can miss the fact that a feature of an item which causes one user to dislike an item, might cause another user to like the item. To illustrate this problem, we analyze a review by $User_A$ accompanied with a low rating:

'Romantic chick-flick; total waste of time!'

While it is obvious that $User_A$ did not like the item, the same review can suggest that $User_B$, who likes romantic movies, might like this movie. Another problem with existing review-based systems is that they extract the features of the items and subsequently discard the information about the reviewers. For example, suppose $User_A$ reviewed a hotel $Hotel_X$ as:

'The view from the hotel was excellent!'

Let another user, $User_B$, review a hotel $Hotel_Y$ using the exact same words. Existing review-based recommendation systems generally interpret these reviews as '$User_A$ and $User_B$ both care about the views from the hotels' and '$Hotel_X$ and $Hotel_Y$ both have nice views.' Therefore, these systems will recommend $Hotel_X$ to $User_B$ and $Hotel_Y$ to $User_A$. But if $Hotel_X$ has a nice city view, which $User_A$ likes, and $Hotel_Y$ has a nice nature view, which $User_B$ likes, these recommendations are misleading. Language Independent Recommender Agent (LIRA) avoids this problem by learning that $User_B$'s reviews are not predictive of $User_A$'s ratings.

Another problem that existing review-based recommendation systems face is that not every user is willing to write reviews for the items they evaluate. Rating/review ratios in *IMDB* and *GoodReads* are only 1/400 and 1/20, respectively. Because of this sparsity, *systems which require reviews from the target users will fail to produce recommendations for a large majority of the users.* LIRA does not require target users to write reviews and can recommend items to users as long as they rate items.

Finally, most of the existing review-based systems depend on topic modeling approaches or finding similarities between different users' reviews. This limits their applicability to situations where all the reviews are in the same language. They also fail to work when the writing style of the users change, such as using 'gr8' instead of 'great'. LIRA can use reviews from different languages for recommendation, and even different writing styles, as long as users are consistent about their language usage or writing style.

In this paper, we present an agent-based recommender system, LIRA, which creates agents for target users (A_T) and reviewers to create accurate recommendations. LIRA consists of targets user's agent, which processes messages from agents of other users, which sends the words sequences (N-Grams) in those users' reviews to target user's agent. LIRA agents, A_Ts, learn from those messages and then make predictions for unknown items based on the learnt knowledge.

We summarize the problems confounding other recommender systems but which LIRA can handle:

1. LIRA can make use of different perspectives of the users on different features of items, such as liking the different types of views in different hotels.
2. LIRA does not depend on other users' or sources' ratings, it can work on the ratings of a target user, given that reviews are available for items rated by the target user. This property also enables LIRA to utilize reviews from different sources, such as blogs, reviews from different websites, etc.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

3. LIRA does not depend on social connections, explicit statement, or rating similarity for calculating trust for a review source. It can calculate these trust values by measuring the capability of those sources to predict target user's rating in training set.
4. LIRA only needs ratings of target users and does not require them to write reviews to get recommendations.
5. LIRA is language independent: it can leverage reviews written in any language.

2 Background and related work

In this section, we provide the literature review on the methods we use and also about the state of recommendation systems research.

2.1 Term frequency-inverse document frequency

Term frequency-inverse document frequency (TF-IDF) is a measure of how important a word is to a document (Leskovec *et al*., 2014). It is widely used in the text mining approaches. TF-IDF value is calculated by multiplying two measures: term frequency (TF) and inverse document frequency (IDF).

TF value is calculated using the following equation:

$$\mathrm{TF}_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{1}$$

In the equation above, TF_{ij} is the term frequency value of word i in document j, f_{ij} is the frequency of i in j (number of occurrences), $\max_k f_{kj}$ is the frequency of the most frequent word k in document j. Thus, the most frequent term in document j gets a TF of 1 and other terms get fractions as their term frequency for this document.

IDF value is calculated using the following equation:

$$\mathrm{IDF}_i = \log_2 \frac{N}{n_i} \tag{2}$$

In the equation above IDF_i is the inverse document frequency of word i, n_i is the number of documents that word i appears in, and N is the total number of documents that we have in our collection.

Then TF-IDF value becomes:

$$\mathrm{TF{-}IDF}_{ij} = \mathrm{TF}_{ij} \times \mathrm{IDF}_i \tag{3}$$

The terms with the highest TF-IDF score are often the terms that best characterize the topic of the document. We use TF-IDF because it can eliminate too frequent words, as we do not want the system to be affected by words, which appears too frequently.

2.2 Regression algorithms

In this section, we are providing a list of regression algorithms which can be used with LIRA.

2.2.1 Gaussian process

A Gaussian process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions (Williams & Rasmussen, 2006). Every Gaussian process can be fully specified with its mean function $m(x)$ and co-variance function $k(x, x')$. Given N data points, X_n, t_n where the inputs x are vectors of some input dimension, and targets t are the output values, the Gaussian process tries to infer the underlying function, $f(x)$ from the given data (MacKay, 1998). When that function is inferred, a Gaussian process can use that function to predict t_{n+1} for a new data point x_{n+1}.

2.2.2 Support vector regression

The support vector machine, originally proposed by Vapnik (2013), is a learning algorithm (Cristianini & Shawe-Taylor, 2000; Palaniswami & Shilton, 2002) with the ability to provide function estimation. By

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

using a mapping, $\Phi{:}X \rightarrow F$, where X is the domain and F is usually a high-dimensional feature space, support vector regression operates in feature space to approximate unknown functions in an output space Y, thereby using nonlinear functions to linearly estimate an unknown regression.

2.2.3 *Neural networks*

Artificial neural network is a widely used regression method (Wang, 2003). It aims to simulate the working method of human brain to learn and make predictions about data. It is implemented by creating rather simple nodes, called perceptrons, and creates fully connected layers to form a network. Perceptrons are linear separators, which gets the weighted sum of inputs and checks if that some is over or under the threshold value for that perceptron.

Actual learning in neural networks is done by learning the correct weight values between perceptrons. At every iteration, weights between every pair of nodes are updated using

$$W_{i,j} \leftarrow W_{i,j} + \alpha \times a_i \times \Delta_j, \tag{4}$$

where Δ_j is the error associated with node j, α is the learning rate, and a_i is the output of node i. When a neural network is learning, the only correct value which is known is the output value, therefore the error of the system can only be measured for the output layer. So the error associated with output node, o, is

$$\Delta_o = \mathrm{Err}_o \times g(\mathrm{input}_o). \tag{5}$$

The error made by output nodes is back-propagated to the hidden nodes to find their contribution on the error. So error for a hidden layer node h becomes:

$$\Delta_h = g(\mathrm{input}_o) \times \sum_i W_{h,i}\Delta_i \tag{6}$$

2.3 *Recommendation systems*

Recommendation techniques can be broadly divided into two categories: *non-personalized* and *personalized* recommendation techniques. *Non-personalized recommendation* techniques compute the average statistics of the recommended items and recommend the same item set to all customers, that is, they do not consider individual user preferences. For instance, a bookstore might recommend a bestseller book to all of its customers. *Non-personalized* systems have the advantage of being fast. Moreover, they do not suffer from the *cold start* problem[1], since they do not need an initial data (ratings, etc.) from each customer. Nevertheless, the quality of the results is low because of the lack of personalization.

Personalized recommendation techniques provide better results by considering individual user preferences. Two well-known approaches are *content-based* and *collaborative filtering*. Moreover, some hybrid approaches have emerged to overcome the problems of these approaches. Adomavicius and Tuzhilin (2005) provide a detailed survey about recommender systems.

Content-based filtering algorithms recommend items that are similar to the ones that the user has liked. *Content-based* systems usually depend on textual content, for example, keywords. Information retrieval (IR) techniques, such as *TF-IDF* (Manning *et al*., 2008), can be applied on the keywords. The similarity among items can be determined by some scoring heuristics, such as *cosine similarity*, after the *TF-IDF* vectors are calculated. Apart from *IR* techniques, *Bayesian classifiers* and different machine learning techniques like *clustering decision trees* and *artificial neural networks* can be used for the similarity calculation (Adomavicius & Tuzhilin, 2005). Content-based recommendation systems have been used by the researchers on various topics (Lops *et al*., 2011), such as music (Soleymani *et al*., 2015), books (Mooney & Roy, 2000), movies (Debnath *et al*., 2008), etc. However, these systems need the items to be marked with features, which is usually objective, and cannot make use of user similarities.

Collaborative filtering algorithms identify people with overlapping interests. These algorithms rely on the assumption that people who have exhibited similar interests in the past will continue to have similar interests in the future. Instead of considering correlation of items as in *content-based* algorithms, correlation between users' preferences is examined. This is achieved by analyzing user ratings of items: if two

[1] The problem of not being able to provide recommendations because of the lack of preference information about the user.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

users have provided similar ratings for many items, then it is concluded that these users have similar preferences. Another way of finding users with similar interests is to match demographic characteristics of users, for example, age, education, geographic locations, gender, etc. After determining like-minded users, the transitive relationship between users and items is utilized. For instance, if user c_1's preferences are similar to that of user c_2's and user c_2 likes item s_1, s_1 is recommended to c_1 if c_1 has not viewed item s_1 yet.

As in the case of *content-based* approaches, *IR* techniques, like *cosine measure*, can be used to calculate similarities between users. However, in *collaborative filtering*, similarity between vectors of the actual user-specified ratings is measured (Adomavicius & Tuzhilin, 2005). Moreover, different machine learning techniques are also applicable for identifying similarities between users (Mitchell, 1997; Adomavicius & Tuzhilin, 2005; Alpaydin, 2014).

RegSVD, proposed by Paterek (2007), improves on SVD by learning factor vectors directly on known ratings through a suitable objective function which minimizes prediction error. The proposed objective function is regularized in order to avoid over-fitting. Gradient descent is applied to minimize the objective function.

Collaborative filtering techniques have their own limitations. They work reliably only when there are sufficient number of users in the system with overlapping characteristics. Another concern is that when a new item is added to the system, it cannot be recommended to others until a number of people have rated it. Furthermore, these techniques are computationally expensive, and the recommendation process becomes cumbersome for millions of users and items. Collaborative filtering has also been tried on various domains such as e-commerce (Linden *et al*., 2003), movies (Ungar & Foster, 1998), etc. Another shortcoming is that these systems cannot identify when two users rate an item with the same rating with different reasons.

2.3.1 Recommendation with topic models

A topic model is a statistical model that generates documents (strings of words) from some set of topic clusters. Parametric estimation techniques and unsupervised clustering algorithms such as the expectation-maximization algorithm (Dempster *et al*., 1977) can fit models to observed documents. With the model, it is possible to estimate a likely distribution over topics on a document level and a distribution over words on the topic level. Topic models are an effective tool for characterizing documents since they are not necessarily dependent on a language's underlying grammar, and yet can identify significant topic trends using the rate of each word's appearance. Some of the more popular existing topic model frameworks are latent semantic analysis (Deerwester *et al*., 1990) and latent Dirichlet allocation (LDA) (Blei *et al*., 2003).

Topic models are often used in recommender systems in a variety of creative ways (Jakob *et al.*, 2009; Hariri *et al.*, 2011; Chen & Wang, 2013; Liu *et al.*, 2013; Bao *et al.*, 2014). Generally, such recommender systems use topic models to generate latent features in their documents, which can then be. supplied to some supervised learning algorithm (Paterek, 2007; Jakob *et al*., 2009; Chen & Wang, 2013).

The fLDA topic model from Agarwal and Chen (2010) extended the LDA model to include ratings as well. The paper approaches the problem where a system has some users and items, and the goal of the recommender system is to make recommendations based on the user's rating history and a textual description of each item. The fLDA model simultaneously learns: (a) the correlation between items' documents and features using LDA, and (b) the users' affinity towards each topic, to produce (c) a rating distribution for each user i and item j. The rating distribution inside the model is defined as a Normal (or binomial) distribution: $y_{ij} \sim \text{Normal}\,(\alpha_i + \beta_j + s_i^T \overline{z}_j, \sigma^2)$ where y_{ij} is the rating, α_i is the user's rating bias, β_j is the item's rating bias, s_i is the user's affinity towards each topic as a vector, and $\overline{z}$ is a latent variable measuring how each word in the description of item j correlates with a topic.

Similarly, the TopicMF model by Bao *et al*. (2014) incorporates a topic model based on matrix factorization; this model considers both ratings and reviews of items in its recommendation. Their model combines this topic model using matrix factorization with another recommender system using matrix factorization to produce a combined model. Given I users, J items, $D = I \times J$ reviews available, W total words and an observed user-to-item rating matrix $\mathbf{r}$ and a review-to-word frequency matrix $\mathbf{F}$, the algorithm attempts to find the best factorization for K hidden topics: $\mathbf{r} = \mathbf{u}_{I\times K}\mathbf{v}_{J\times K}^T, \mathbf{F} = \Theta_{D\times K}\Phi_{W\times K}^T$.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

The hidden matrices are learned by minimizing the least squares equation: $\min_{\mathbf{u},\mathbf{v},\Theta,\Phi} || \mathbf{r}-\mathbf{u}\mathbf{v}^T || + || \mathbf{F}-\Theta\Phi^T || + \gamma$ where γ is an additional term that includes bias values and measures to prevent learned parameters from growing too large.

McAuley and Leskovec (2013) have proposed the hidden factors as topics (HFT) algorithm, which creates topics to identify the hidden topics in the reviews to describe the items of interest, and also the users' interest in those items. They are also using their algorithm to identify the categories of those items automatically.

These systems work on reviews provided by the users, but in the process of combining them into topics they lose the information about the reviewers. For example, if two people both mention that they like the 'view' of two different hotels, those items and users will be considered similar by these systems. However, the types of the views can be different and the users opinion of a 'good view' might also differ between users.

2.3.2 *Hybrid recommendation systems*

Sparsity of available data makes the personalization of rating prediction difficult, as it makes it hard to link users and products to each other. To overcome this problem, some researchers focused on generating hybrid recommendation systems to combine the strengths of different approaches (Kazienko & Musiał, 2006). Mostly, the base of these hybrids is a combination of content-based (Schein *et al*., 2002; Kazienko & Musiał, 2006; Levi *et al*., 2012) and collaborative recommendation systems (Sarwar *et al*., 2001). Some researchers tried to incorporate the information from social networks to this hybrid recommendation systems either as social tags (Middleton *et al*., 2004) or social trust (Sugiyama *et al*., 2004). Some of the more recent studies focused on creating hybrid recommendation systems which also make use of sentiment analysis approaches (Homoceanu *et al*., 2011; Yang *et al*., 2013). Leung *et al*. (2008) proposed a method to extract association rules from data and use it as a supplement to enhance the recommendation system.

2.3.3 *Recommendation using trust*

The area of trust-based recommender systems has been the object of extensive study for the past years. Indeed, trust has been shown to provide significant improvements to classical collaborative filtering techniques. The main difference between most of these trust-enhanced methods is the acquisition of trust values between actor pairs.

Golbeck (2005) introduced *TidalTrust* algorithm, which uses a modified breadth-first search, to estimate the trust by using transitive rules. In this study, the trust of user u for user v is calculated using u's trusted users' trust on v. Massa and Avesani (2007) and Massa and Bhattacharjee (2004) proposed a similar method where instead of only looking at one level trust transitivity, they remove cycles from the trust network. Using a *trust propagation horizon* phase, they allow the trust to diffuse along the network.

While the studies above require an explicit statement about direct trust between users, it is very difficult to get that information from user explicitly. Therefore, some other studies focused on defining their own implicit trust values based on implicit information. O'Donovan and Smyth (2005) proposed a trust value based on the similarity of two user's ratings, while Wang *et al*. (2011) generated a trust metric based on the similarity among users' tastes.

3 Methodology

In this section, we define how LIRA works. We first describe the preprocessing steps, that is, how the data are prepared, and then explain how LIRA uses that data. While the dataset includes detailed meta-data about users, items, and reviews, our recommender system will only examine the textual content and numeric rating associated with the records. We do not need the rating to accompany the review so we create two sets of triples: (*Reviewer*, *Item*, *Review*) and (*User*, *Item*, *Rating*). This property shows the reviews and ratings are only related through the items. Note that sets of *Users* and *Reviewers* are not necessarily disjoint. If a person rates items but does not write reviews, they will only appear in the set of

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Users. Likewise, if a person writes reviews but do not provide ratings, such as bloggers or critiques, they only appear in the set of *Reviewers*. However, if a person both writes reviews and provides ratings, they appear both in *Reviewers* and *Users* sets.

3.1 Preparing data

For this study, we have used 'Amazon Reviews' dataset in five categories (McAuley *et al*., 2015a,b). In addition to these, we have also used 'GoodReads' dataset that consists of book ratings and reviews from many different languages.

In all datasets we used, we run the following pre-processing steps:

1. Remove products which do not have any reviews, along with ratings for those products.
2. Remove users who have less than 10 ratings, along with ratings by those users.
3. Remove reviews for products which do not have any ratings.

Information about the resulting dataset is shown in Table 1.

Instead of following the traditional method of randomly splitting the data into two sets (training and testing), we use a slightly different approach. Because our algorithm works on every user separately, we split the data at the user level. We chose a random split of 80–20% of ratings for each individual user and use the former as the training and trust calculation set, and the latter for testing our algorithm. All of our datasets are large enough to validate with hold-out method.

3.2 LIRA details

LIRA uses a framework that utilizes separate learners for every available source (see Figure 1). It learns the writing behavior and predictiveness of reviews from each source separately. For this example, the rating scale used is [1, 5].

Table 1 Properties of datasets

	Raters	Reviewers	Items	Ratings	Reviews
Apps	20 104	87 271	13 105	334 927	752 325
Automotive	360	2 928	1 579	4 971	18 822
Baby	4 382	19 445	6 883	67 060	159 780
Beauty	5 123	22 359	11 654	91 824	195 763
Music	1 763	5 541	3 539	41 152	64 546
GoodReads	5 094	3 011	106 213	518 855	163 549

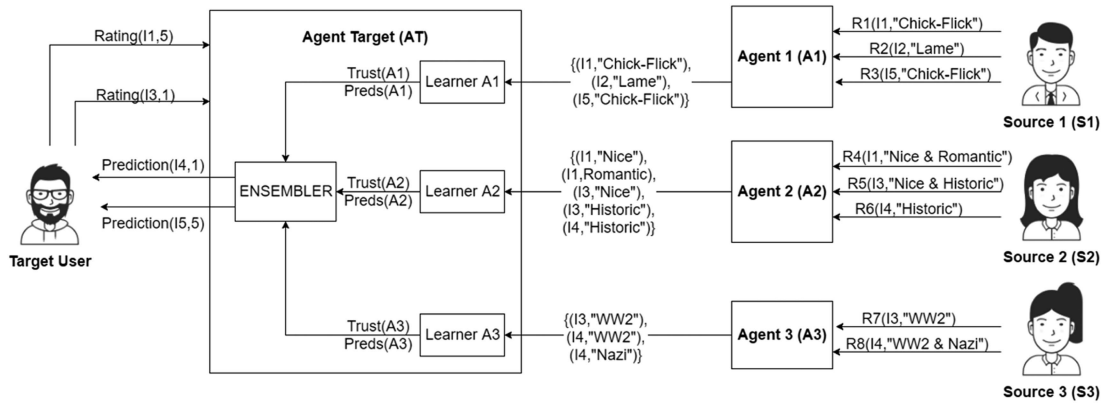


Figure 1 Overview of LIRA

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

We start the initial processing by creating data points that consist of the reviews by every source separately and marking them with target user's rating for the corresponding item. That data point is then converted to a feature vector where the features are the words in the review and their values are their respective TF-IDF (Leskovec *et al.*, 2014) values. Note that LIRA is a general framework, within which any text to feature vector conversion approach can be used at this step.

3.2.1 Text regression

We choose a machine-learning regression algorithm, such as neural network (Wang, 2003), Gaussian process (Williams & Rasmussen, 2006), etc. to learn to predict the target user's ratings for unseen items. Any regression algorithm can be used at this step of LIRA. In Figure 1, *Learner A1* will only find the common item *I*1 among target user's ratings and source *A*1's reviews. *Learner A1* will learn that the word sequence **'Chick-Flick'** is likely to result in the target user rating that item 5. Similarly, *Learner A2* will find two common items, *I*1 and *I*3, getting the reviews from *A*2 for those items. *Learner A2* will learn that the word **'Romantic'** by *A*2 means high ratings by target user and **'Historic'** by *A*2 means low ratings by target user. *Learner A2* will also realize that word **'Nice'** by *A*2 is not useful for predicting target user's ratings.

3.2.2 Training

In the training phase, LIRA creates learners for every target user-source pair, to learn how the source's reviews are predictive of the target user's ratings. Hence, for every target user, $User_T$, and source, *S*, pair, LIRA creates $Learner_{TS}$ and feeds ($Review_{SI}$, $Rating_{TI}$) pairs to this learner. Here, $Review_{SI}$ is source *S*'s review of $Item_I$ and $Rating_{TI}$ is $User_T$'s rating of $Item_I$.

Since the reviews are in free-form text, they are first converted to feature vectors using TF-IDF. It should be noted that text-to-feature conversion is done separately for every source in LIRA. Therefore, as long as a given source is consistent with its individual writing style and language usage, stop-word elimination will eliminate the words that are not informative only for that particular source's reviews. For example, if the source starts every review with a '**Merhaba**[2]', this word will be identified as a stop-word, and will be eliminated.

These feature vectors are used to train, $Learner_{TS}$, using a plug-and-play regression algorithm. The algorithm for training LIRA is presented in Algorithm 1. Note that if a source has not reviewed any items rated by the target user, it cannot be used as a predictor for this user's rating, and is eliminated from consideration (see line 4).

Algorithm 1: LIRA Training For a Target User $User_T$.

Input: $Ratings_T$, set of ratings provided by T
Input: $Sources$, set of sources who provides reviews
Output: $LearnerSet_T$, set of trained regression models for every source

```
1 LearnerSet_T = {};
2 foreach S ∈ Sources do
3     Reviews_TS ← getReviewsBySourceForItemsRatedByT(Ratings_T, S);
4     if Reviews_TS = ∅ then
5         skip source;
6     EliminateStopWords(Reviews_TS, Ratings_T);
7     FeatureVectors ← TextToFeatureConversion(Reviews_TS);
8     Learner_TS.train(FeatureVectors, Ratings_TS);
9     LearnerSet_T.add(Learner_TS)
```

3.2.3 Trust calculation

We have mentioned in Section 2.3.3 that existing recommendation systems assign a trust value between users either by using rating similarities between users or by analyzing social connections between them.

[2] *Hello* in Turkish.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

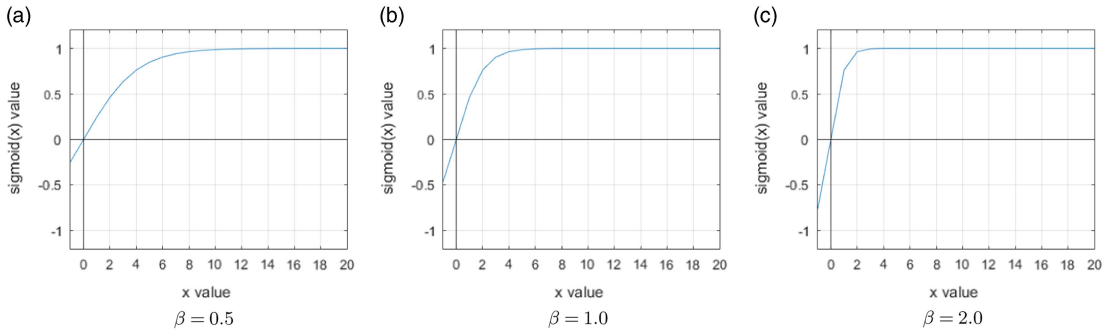


Figure 2 Behavior of adjusted sigmoid function with different β values

Unlike those systems, we decided to assign trust values to represent their capability of making predictions for the target user.

As mentioned in Section 3.1, the available dataset is split into training and trust calculation sets. First, all the learners are trained using the training set. Then we make all the learners predict every item in the trust calculation set. This approach is similar to the use of validation set in common supervised machine learning approaches. Root mean square error (RMSE) for learner L is calculated as:

$$\mathrm{RMSE}(L) = \sqrt{\frac{\sum_{I \in P_L} (\mathrm{Prediction}(L, I) - r_I)^2}{\mid P_L \mid}}. \tag{7}$$

In Equation (7), P_L is the set of items which L can make predictions on, r_I is the actual rating, and Prediction(L, I) is the rating prediction of L for item I.

This error measure is useful but not sufficient for a complete evaluation of our system. If we only consider the RMSE, a $Learner_A$ which made only one, albeit perfectly accurate, prediction will have the same trust value as a $Learner_B$ who was able to perfectly predict a large number of items. To address this issue, we include the number of predictions in our trust calculation. We use a sigmoid function to ensure that the magnitude of this count does not suppress the error value. As the number of predictions is non-negative, we choose the sigmoid function given in Equation (8) such that $\mathrm{sigmoid}(0) = 0$ and $\mathrm{sigmoid}(\infty) = 1$. We introduce the β parameter to adjust how quickly our sigmoid function reaches ~ 1 (see Figure 2).

$$\mathrm{sigmoid}(x) = \frac{2}{1 + \mathrm{e}^{-\beta x}} - 1. \tag{8}$$

We need to combine the error value and number of common items for every learner, to get the final trust value. Trust should be positively correlated with the number of common items. So we choose the sigmoid function, with the number of common items as argument, as the numerator of the trust expression. The error measure calculated must be negatively correlated with the trust value. As the best case has 0 errors, to ensure the trust value lies between [0, 1] we use (1 + errors) as the denominator term. For tuning the relative importance of the number of common items and error, we use an α parameter as the exponent of the denominator. The resultant trust value is then calculated as

$$\mathrm{trust}(L) = \frac{\mathrm{sigmoid}(\mid P_L \mid)}{(1 + \mathrm{RMSE}(L))^{\alpha}}. \tag{9}$$

Figure 3 shows how the importance of RMSE and the number of predictions change according to the values assigned to α and β parameters.

In some cases, a prediction may be made by a new learner, where they do not have any previous predictions to calculate a trust value. To avoid losing the information from those learners, we assign them a default trust value. Since we do not know about the capabilities of this learner yet, we conservatively

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

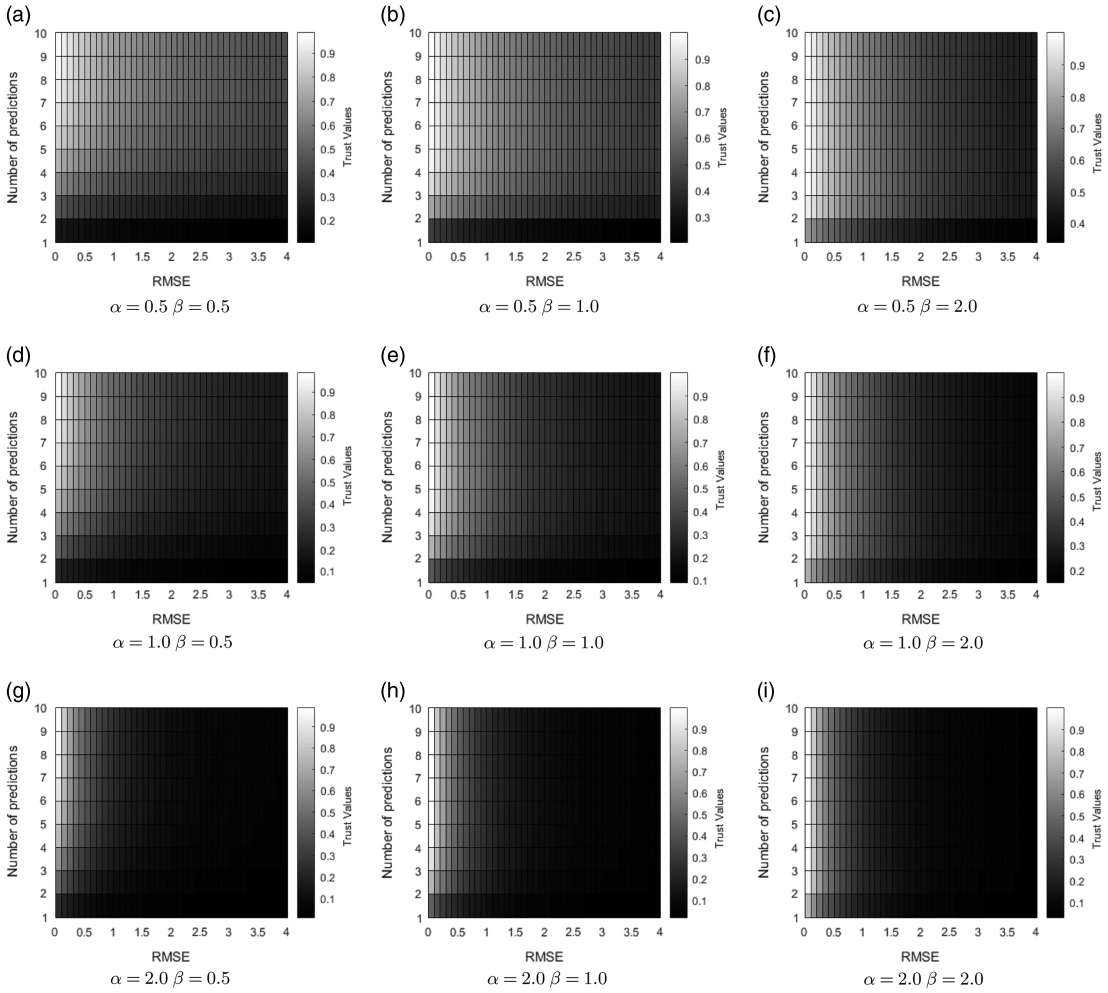


Figure 3 Behavior of trust function with different α and β values

assign them the minimum possible trust value, which is calculated by assuming they have made a single prediction with maximal prediction error possible. Since the maximum value of error in a prediction can be the difference between maximum and minimum values in the rating scale we end up with the following

Algorithm 2: LIRA Trust Calculation For a Target User $User_t$

Input: $Sources$, set of sources who provides reviews
Input: $Predictors_t$, set of regression models previously trained for t separately for every source
Input: $Ratings_t$, set of ratings provided by t
Output: $TrustVector_t$, a vector which holds t's trust values for the sources

foreach $s \in Sources$ **do**
  $Learner_s \leftarrow Predictors_t.get(s)$;
  $Count_s = 0$;
  foreach $i \in Ratings_t$ **do**
    if *s has a review for i* **then**
      $Count_s \leftarrow Count_s + 1$;
      $Text \leftarrow getReview(s, i)$;
      $FeatureVector \leftarrow TextToFeatureConversion(Text)$;
      $Prediction \leftarrow Learner_s.predict(FeatureVector)$;
      $Error_s.add(Rating_{ti}, Prediction)$;
  $Trust_{ts} \leftarrow CalculateTrust(Error_s, Count_s)$;

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

expression (Equation (10)) for the trust value of the new learner (l_{new}):

$$\text{trust}(l_{\text{new}}) = \frac{\text{sigmoid}(1)}{(1+\text{Rating}_{\max}-\text{Rating}_{\min})^{\alpha}}. \tag{10}$$

3.2.4 Predicting target user's ratings for candidate items

In the prediction step of LIRA, we predict the rating for the unseen items by the target user. To achieve this, we will use the regression model trained in Section 3.2.2 and trust values calculated in Section 3.2.3.

For a candidate item, *i*, we first retrieve every source that has written a review for *i* and also has a learner trained for target user $User_t$. Every source's review is separately fed into their respective learner for *t*, which calculates predictions by individual sources. These predictions are then combined using a weighted average approach, where the weights correspond to trust values for each source. To avoid losing any predictive value in reviews from new sources, which did not have any reviews that can be used in the trust calculation step, we set their trust level to a default value explained in Equation (10).

$$\text{Prediction}(t,i) = \frac{\sum_{s\in(\text{Sources}_i\cap\text{Trained Sources}_t)} \text{Trust}_{ts}\times\text{Prediction}_s(t,i)}{\sum_{s\in(\text{Sources}_i\cap\text{Trained Sources}_t)} \text{Trust}_{ts}} \tag{11}$$

The prediction LIRA makes for a $Target\ User_t - Item_i$ pair is calculated using the formula in Equation (11). In this equation, Trust_{ts} is the trust value calculated for $Source_s$ for $Target\ User_t$'s ratings, as described in Section 3.2.3. $\text{Prediction}_s(t,i)$ is the prediction made by $Source_s$'s review for $Target\ User_t - Item_i$. $Source_i$ is the set of sources who have provided reviews for $Item_i$ and $Trained\ Sources_t$ is the set of sources with a trained learner for $Target\ User_t$. Note that, for some $Target\ User_t - Item_i$ pairs, $\text{Sources}_i \cap \text{Trained Sources}_t = \emptyset$. Therefore LIRA will not be able to make predictions for those pairs.

Learners which do not have a review written for the item, we are trying to predict the rating for cannot make prediction and we exclude those sources. For example, *Learner A*2 will not be able to make a prediction for *I*5.

The final prediction for an item I_X, $\hat{rI_X}$ is calculated as:

Algorithm 3: LIRA Rating Prediction For a Target User, $User_t$ and Candidate Item, $Item_i$

Input: $Sources_i$, set of sources who have provided reviews for i
Input: $Predictors_i$, set of regression models previously trained for t separately for every source
Input: $TrustVector_t$, a vector which holds t's trust values for the sources
Output: $\hat{r}_{ti}$, predicted rating of t for i

```
TotalTrust ← 0;
RawPrediction ← 0;
foreach s ∈ Sources_i do
    if s ∈ Predictors_t then
        if s ∈ TrustVector_t then
            SourceTrust ← TrustVector_t.get(s);
        else
            SourceTrust ← defaultTrust;
        Learner_s ← Predictors_T.get(s);
        Text ← getReview(s, i);
        FeatureVector ← TextToFeatureConversion(Text);
        Prediction ← Learner_s.predict(FeatureVector);
        TotalTrust ← TotalTrust + SourceTrust;
        RawPrediction ← RawPrediction + SourcePrediction × SourceTrust;
r̂_ti = RawPrediction/TotalTrust;
```

$$\hat{rI_X} = \frac{\sum_{u\in\text{Rev}(I_X)} \text{Trust}(L_u)\times\text{Prediction}(L_u, I_X)}{\sum_{u\in\text{Rev}(I_X)} \text{Trust}(L_u)}, \tag{12}$$

where $\text{Rev}(I_X)$ is the set of reviewers who have provided review for item I_X, $\text{Trust}(u)$ is the trust value for that reviewer, and $\text{Prediction}(L_u, I_X)$ is the prediction by *Learner u*, L_u, for item I_X.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

3.3 Scalability

Given the description of the approach, it may appear that this approach is computationally expensive because of the need for separately training a learner for every user-source pair, where every reviewer is used as a source for every other user. For a system with N users, where each user is also a reviewer, the number of learned models will be $N \times (N-1)$. However, some properties of the algorithm can be used to redress this problem as follows:

1. It is immediately obvious that creating the classifiers for separate users are independent processes. Therefore, this approach is easily parallelizable.
2. More importantly, LIRA is designed to be an agent-based system. So, every target user can run their own agent, and therefore their own learners, on their local computer, allowing them to get recommendations from the system without the burden of running LIRA for any other user[3]. This way they will be training only one classifier which contains N sub-learners, where N is the number of sources, which will not be computationally prohibitive.

Another approach, to make the system even more efficient and part of our planned future work is to train only a fixed number of most trustworthy sources for each target user. Assuming the top K sources per user is identified, the number of total models in the system will be reduced to $N \times K$. Periodically, the pool of trusted sources can be updated by a process of exploration. This problem can be represented as an instance of multi-arm bandit problem (Gittins *et al*., 2011) where reviewers correspond to the available options and their accuracy on predicting the target user's ratings is the pay-off for the corresponding option.

4 Experimental setup

In this section, we explain the experiments we run, and the motivation behind them.

4.1 Selection of methods to use

We evaluate our algorithm on three datasets to demonstrate the general applicability and domain independence of our approach. We selected 'Automotive', 'Baby', and 'Digital Music' as the experimental domains because of their distinct characteristics. We are assuming that the opinions are mostly objective and based on the quality of the item for the 'Automotive' domain, that the opinions are mostly subjective and based on the preference of users for the 'Music' domain, and that the opinions will be a combination of objective quality and subjective personal preference in the 'Baby' domain.

For selecting appropriate method for regression and the tokenizer used in text to feature conversion, we run LIRA using the candidate methods on 100 randomly selected users and test which method yields the best results.

4.1.1 Regression method selection

To choose which regression method to use, we have tested LIRA using following regression learning algorithms: Neural network, Gaussian process, and SMOReg. For these algorithms, we used the default settings from WEKA. For the number of hidden nodes is (attributes + classes)/2, learning rate is 0.3, and sigmoid function has been used as the activation function. Kernel function used for both Gaussian Process and SMOReg is *polynomial kernel function* (Fan *et al*., 1995).

4.1.2 Tokenizer selection

To choose which tokenizer method to use, we have tested LIRA using following tokenizers: Word Tokenizer which only extracts words from a given text, and N-Gram Tokenizer, which tries to extract

[3] This means that LIRA implementations should be distributed and not susceptible to the bottlenecks and single-point-of-failure issues plaguing centralized approaches.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

N-Grams, including unigrams (words) (Mayfield & McNamee, 2003). We limited the N-Grams to unigrams, bigrams, and trigrams, to reduce to complexity and prevent the data from getting too sparse.

4.2 All ratings experiment

For this set of experiments, we have used all the ratings and reviews in the dataset. LIRA works separately for every user, so for every user LIRA goes over every other users' review to predict the rating of the target user.

As LIRA requires a separate trust calculation set, it splits the given training set 75–25%[4], and then uses the larger subset for training and the smaller one for trust calculation. Similar to the split explained in Section 3.1, this split is also done at the user level. This process is also used for the next set of experiments.

We have chosen to compare our algorithm against state of the art algorithms in both rating-based and review-based categories. We have chosen RegSVD (Paterek, 2007) as the rating-based competitor, and HFT (McAuley & Leskovec, 2013) as the review-based competitor.

4.3 100 user experiment

This set of experiments are done to test LIRA's capability of making use of the reviews from other sources. We randomly choose 100 users from the dataset as the group of target users. In our case, other sources then correspond to reviews from other users not in the chosen set. In general, however, these others sources can be taken from completely different platforms, such as websites, blogs, critique reviews, etc.

For the 100 users chosen, we only use their own ratings and not their own reviews, but use the reviews from all the users in the dataset, to predict their ratings. Hence, we consider all the users, except the selected 100, as outside sources for reviews. Note that the reviews from users, except the target user, are also used in the prediction, since those 99 users are considered on-site reviewers.

Since the competing algorithms are only capable of working with on-site data, we test them using ratings and reviews of the 100 selected users.

This approach is motivated by the following real-life scenario. Consider a newly built movie recommendation website which has few (in our example 100) users. As this system will only have the ratings and reviews of these few users, the only rating data that can be used for the recommendation systems is that. However, LIRA can make use of outside sources for the recommendation, such as critique reviews, fan blogs, etc.

Recommendation services with only such small number of users (100 users in this case), corresponds to very sparse datasets. LIRA is not adversely affected by this sparsity, as it is able to make use of reviews from other outside sources.

We made sure that the same set of 100 randomly selected users and exactly the same training and test sets are used to test competing algorithms and LIRA.

4.4 Error measurement

To calculate the error made by our algorithms and competing algorithms, we used the two most commonly used error measurement approaches in recommendation systems: mean absolute error (MAE) (Willmott & Matsuura, 2005) and RMSE (Levinson, 1946).

4.4.1 Mean absolute error

MAE is a very intuitive error measurement approach. As the name suggests MAE is an average of the absolute errors made while predicting each instance (see Equation (13)). MAE is primarily used when the

[4] This 75–25% split is made on the training set. Since the whole dataset was split as 80% training set and 20% test set, the ratios on the whole split becomes 60% model building set, 20% weight calculation set, and 20% test set.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

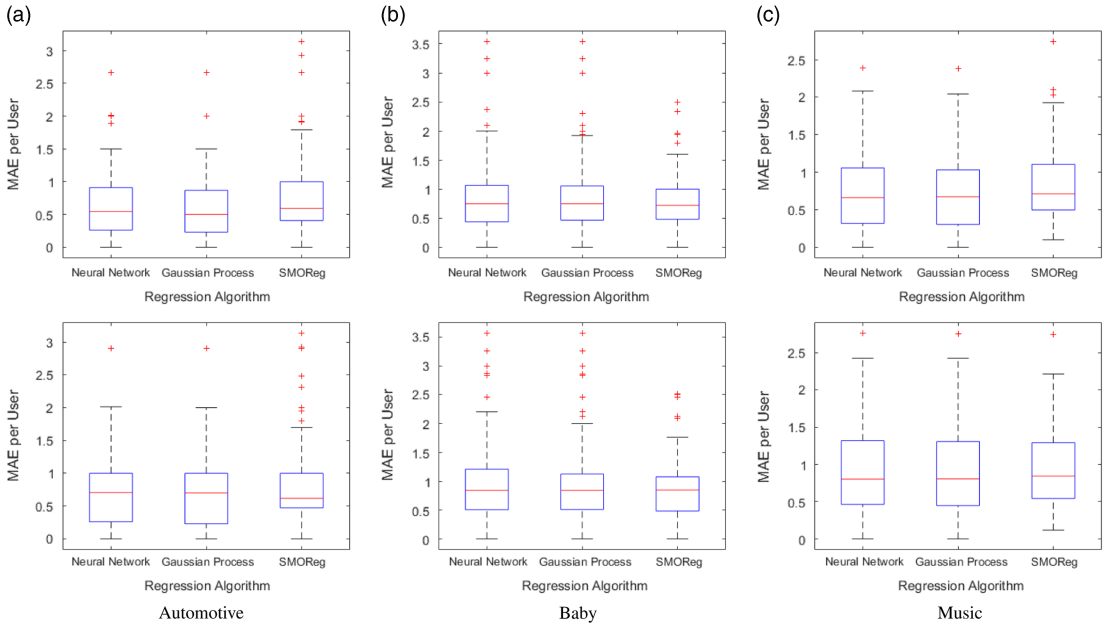


Figure 4 Box-plot of errors per user made by LIRA using each candidate regression algorithm

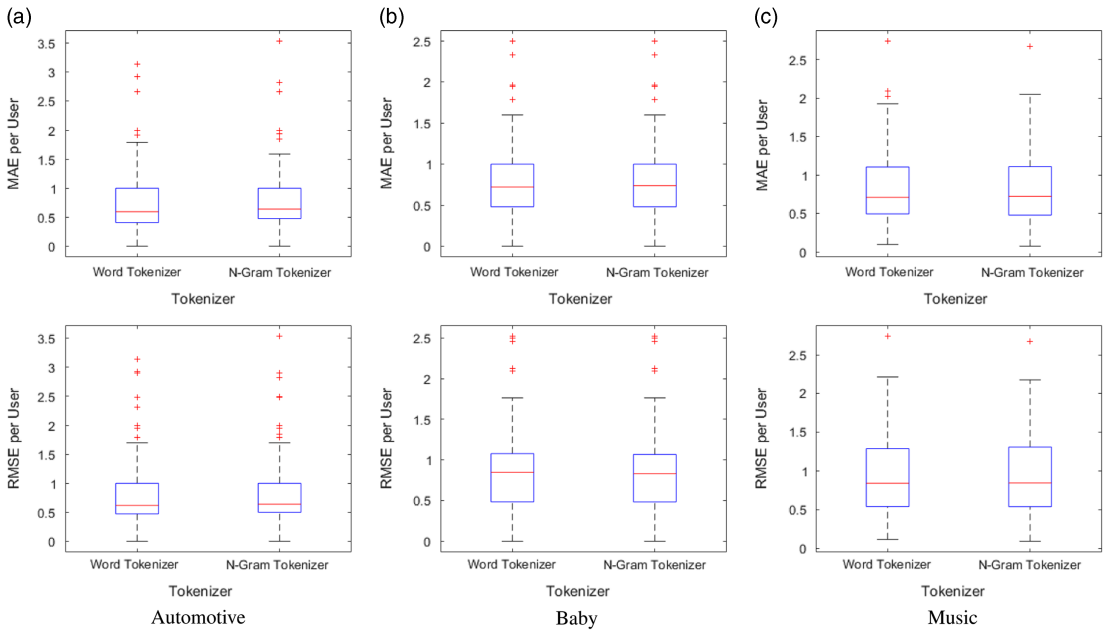


Figure 5 Box-plot of errors per user made by LIRA using each tokenizer

values to be predicted are in the same scale and every error made by the predictor is equally important.

$$\text{MAE} = \frac{\sum_{(u,i)\,|\,R_{u,i}} \mid r_{u,i} - \hat{r}_{u,i} \mid}{\mid \{(u,i) \mid R_{u,i}\} \mid} \tag{13}$$

In this equation, $R_{u,i}$ is a Boolean value signaling whether user u has rated item i in our dataset, as we can only measure the error if we know the actual rating. Variables $r_{u,i}$ and $\hat{r}_{u,i}$ denote the actual and predicted ratings, respectively. Finally, the sum of the absolute differences between actual and predicted ratings are divided by total number of predictions to obtain the average.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Table 2 Average error of LIRA using two tokenizers

	Word Tokenizer		N-Gram Tokenizer	
	MAE	RMSE	MAE	RMSE
Automotive	0.7945	0.8804	0.8181	0.9011
Baby	0.7965	0.8831	0.7913	0.8752
Music	0.8293	0.9683	0.8289	0.9676

MAE = mean absolute error; RMSE = root mean square error.

Table 3 Results of LIRA against competing algorithms in 'All ratings experiment'

	RegSVD		HFT		LIRA		
	MAE	RMSE	MAE	RMSE	MAE	RMSE	Coverage
Apps	1.032	1.351	1.312	1.813	**0.919**	**1.152**	(0.74)
Automotive	1.702	2.081	1.262	1.804	**0.632**	**0.944**	(0.78)
Baby	1.037	1.340	1.243	1.710	**0.867**	**1.111**	(0.82)
Beauty	1.079	1.420	1.091	1.520	**0.843**	**1.120**	(0.81)
Music	0.757	1.022	0.864	1.231	**0.655**	**0.958**	(0.98)
Goodreads	1.015	1.396	0.818	1.093	**0.748**	**0.963**	(0.86)

HFT = hidden factors as topics; MAE = mean absolute error; RMSE = root mean square error; LIRA = Language Independent Recommender Agent.

Table 4 *t*-values for *t*-test comparison of LIRA against competing algorithms

	LIRA vs. RegSVD	LIRA vs. HFT
Apps	29.808	74.779
Automotive	19.546	10.613
Baby	14.915	24.037
Beauty	23.527	21.869
Music	7.681	16.005
Goodreads	67.618	20.865

HFT = hidden factors as topics; LIRA = Language Independent Recommender Agent.

4.4.2 Root mean squared error

RMSE calculates the standard deviation of the errors on the instances by the predictor. RMSE is primarily used when the greater errors are to be penalized harsher. Since error values are squared, RMSE of uniformly distributed errors will be lower than randomly distributed errors, given that MAE remains constant.

The formulation for MAE is given in Equation (14).

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i)\,|\,R_{u,i}} \left(r_{u,i} - \hat{r}_{u,i}\right)^2}{|\,\{(u,i)\,|\,R_{u,i}\}\,|}} \tag{14}$$

In this equation, $R_{u,i}$ is a Boolean showing whether user u has a rating on item i in our dataset, as we can only measure the error if we know the actual rating. Variables $r_{u,i}$ and $\hat{r}_{u,i}$ denote the actual and predicted ratings, respectively. Finally, the sum of the squared differences between actual and predicted ratings are divided by total number of predictions and square-rooted to get RMSE value.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Table 5 Results of LIRA against competing algorithms in '100 user experiment'

	RegSVD		HFT		LIRA		
	MAE	RMSE	MAE	RMSE	MAE	RMSE	Coverage
Apps	2.223	2.604	1.594	2.940	**1.034**	**1.384**	(0.76)
Automotive	2.323	2.682	1.559	2.070	**0.677**	**0.998**	(0.80)
Baby	2.842	3.100	1.190	1.550	**0.814**	**1.177**	(0.87)
Beauty	2.604	2.972	1.481	1.820	**0.939**	**1.172**	(0.83)
Music	2.086	2.472	1.136	1.670	**0.601**	**0.886**	(0.98)
Goodreads	2.128	2.472	1.207	1.687	**0.790**	**1.024**	(0.84)

MAE = mean absolute error; RMSE = root mean square error; LIRA = Language Independent Recommender Agent.

5 Results

In this section, we will present and analyze the results from the above-mentioned set of experiments.

5.1 Selection of methods to use

5.1.1 Regression method selection

We compare error per user by means of MAE and RMSE for three candidate regression algorithms: neural network, Gaussian process, and SMOReg.

Box-plots in Figure 4 show that Gaussian process algorithm outperforms other two algorithms, and hence we choose Gaussian process algorithm to conduct our further experiments.

5.1.2 Tokenizer selection

We compare error per user by means of MAE and RMSE for two tokenizer, Word Tokenizer which only extracts words, and N-Gram tokenizer which also extracts N-Grams.

Box-plots in Figure 5 and data in Table 2 show that the average errors with or without N-Grams are approximately same. Given that including N-Grams do not increase the accuracy of the system, we follow the *Occam's Razor* principle and conduct further experiments with only the Word Tokenizer.

5.2 All ratings experiment

The results of this experiment compare LIRA against the competing algorithms, in the condition where all the data available come from the same environment, therefore all three algorithms are able to make use of all the data. The resulting MAE and RMSE values are presented in Table 3, and bold values mark the minimum MAE and RMSE in every domain. These results show that LIRA outperforms both competing algorithms on all six domains. Since LIRA is language independent, it is able to generate better predictions than competing algorithms, especially in the *GoodReads* domain. It can also be seen that while both competing algorithms suffer from limited data in small-sized domains, such as *Automotive*, LIRA is still able to make accurate predictions.

As explained in Section 3.2.4, for $Target\ User_t - Item_i$ pairs where t does not have learners trained for any of the sources who reviewed i, LIRA will fail to make any prediction. Therefore, the coverage of LIRA is not 100%.

We applied paired t-tests to verify if the improvements LIRA provides are significant. The results show that all the improvements are significant at $p < 0.001$ level (see Table 4).

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

5.3 100 user experiment

The results of this experiment compare LIRA against the competing algorithms, in the condition where data from external sources are available. Since only LIRA is able to make use of external sources, competing algorithms will only be able to make use of data from the selected 100 users.

The resulting MAE and RMSE values are presented in Table 5, and bold values mark the minimum MAE and RMSE in every domain. These results show that LIRA outperforms both competing algorithms on all six domains. These results show that LIRA continues to produce low error rates and is not significantly affected by small size of the dataset. On the other hand, error rate of the competing rating-based algorithm, RegSVD, drastically increases when the number of ratings is low. Error rate of competing review-based system, HFT, also increases, as it is only capable of making use of reviews which are accompanied with ratings[5].

6 Conclusions

While most of the work on recommendation systems are based on ratings and rating similarities, limited representation power of ratings highlights the need for new recommendation systems which can make of use of the other information available to improve recommendation accuracy. With the introduction of Web 2.0 and burgeoning of online platforms, which rely on user-generated content, a rapidly increasing volume of on online and off-site sources make available item reviews that can provide valuable insights about user preferences and can be used to improve the coverage and accuracy of existing recommendation systems. Reviews written by users are more representative and detailed representation of a user's preference for an item. Descriptive reviews also express users' opinions about the features of candidate items and help us develop a deeper understanding of their preferences.

In this paper, we have proposed an agent-based approach, using the trust aspect of agent societies, that uses regression algorithms to predict the ratings of a target user for a candidate item. Our approach creates agents for both target users and reviewers to create accurate recommendations. LIRA consists of targets user's agent, which uses messages, in the form of words or sequence of words in their reviews, from the other agents. The agent of the target user learns predictive knowledge from those messages, which are subsequently used for producing recommendations. LIRA assigns trust to each source's agent using their accuracy on a set of unknown items. Final prediction of LIRA is calculated using a trust-weighted average of sources' individual prediction. Among other benefits, this approach allows the recommendation system to differentiate two users, who use the same word for different reasons, by inferring from their past reviews where identical words have different predictive values for the target user when used by different sources.

Another key feature of LIRA is that it does not use the ratings of the sources and uses only target user's ratings. This results in the LIRA approach having two key advantages. First, it can use reviews from different sources, such as different websites or fan blogs, without needing those texts to be accompanied with a rating value. This makes LIRA useful for addressing the 'cold start' problem of new recommender systems which do not have many users to write reviews for them. Second, by not taking the sources ratings into account, LIRA treats reviews as descriptive texts. For example, if one or more sources complain about a movie, because they found it 'too romantic', LIRA can infer that the movie might be recommended to a target user who does enjoy romantic movies.

Since LIRA does not deal with the meaning of the words, and only use them as signals between agents, the language in which the review was written in becomes irrelevant. As long as sources are consistent with their choice of language or writing style, LIRA will be able to effectively leverage their reviews. The language-independent property of LIRA, whereby reviews written in different languages can be gainfully utilized, is a distinctive advantage compared to other review-based recommender systems that combine reviews from different sources for purposes such as topic modeling (Agarwal & Chen, 2010; McAuley & Leskovec, 2013).

[5] It should be noted that *100 user experiment* is conducted to show that accuracy of LIRA does not get affected by fewer number of users. Since available data are a subset of data used in *All ratings experiment*, which shows the difference is significant, significance values are not necessary for *100 user experiment*.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Last, but not the least, a key advantage of LIRA is that it does not require the target users to write reviews. It is well known that people write reviews much more infrequently than they rate items online. Hence LIRA can be used in many scenarios where review-based recommender systems that require target users to write reviews can be used.

We presented experimental results from two sets of experiments. In the first set, there was a rating accompanying every review. This experiment showed that analyzing reviews from different sources separately, allows LIRA to outperform state-of-art algorithms. Domains where LIRA was more successful than its competitors are likely those where people are more capable and likely to write more descriptive reviews. The second set of experiments show that when there is very little user and rating data available, only LIRA can make use of other sources of information to make accurate recommendations and thus outperform its competitors.

One of the properties of LIRA, which might look like a disadvantage, is that it needs to train regression algorithms for every user-source pair. However, it should be mentioned that the agents work independently from each other, which makes this approach highly parallelizable. We are currently also working on an approach which provides the same advantages that LIRA provides, without having to create separate learners for every source.

References

Adomavicius, G. & Tuzhilin, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749.

Agarwal, D. & Chen, B.-C. 2010. fLDA: matrix factorization through latent Dirichlet allocation. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 91–100. ACM.

Alpaydin, E. 2014. *Introduction to Machine Learning*. MIT press.

Bao, Y., Fang, H. & Zhang, J. 2014. . TopicMF: simultaneously exploiting ratings and reviews for recommendation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2–8.

Blei, D. M., Ng, A. Y. & Jordan, M. I. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* **3**, 993–1022.

Chen, L. & Wang, F. 2013. Preference-based clustering reviews for augmenting e-commerce recommendation. *Knowledge-Based Systems* **50**, 44–59.

Cristianini, N. & Shawe-Taylor, J. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press.

Davis, J. M. 1958. The transitivity of preferences. *Behavioral Science* **3**(1), 26–33.

Debnath, S., Ganguly, N. & Mitra, P. 2008. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th International Conference on World Wide Web*, 1041–1042. ACM.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W. & Harshman, R. A. 1990. Indexing by latent semantic analysis. *JAsIs* **41**(6), 391–407.

Dempster, A. P., Laird, N. M. & Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* **39**, 1–38.

Fan, J., Heckman, N. E. & Wand, M. P. 1995. Local polynomial kernel regression for generalized linear models and quasi-likelihood functions. *Journal of the American Statistical Association* **90**(429), 141–150.

Gittins, J., Glazebrook, K. & Weber, R. 2011. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons.

Golbeck, J. A. 2005. *Computing and Applying Trust in Web-Based Social Networks*. PhD thesis, University of Maryland at College Park, College Park, MD, USA. AAI3178583.

Hariri, N., Zheng, Y., Mobasher, B. & Burke, R. 2011. Context-aware recommendation based on review mining. *General Co-Chairs*, 27.

Homoceanu, S., Loster, M., Lofi, C. & Balke, W.-T. 2011. Will i like it? Providing product overviews based on opinion excerpts. In *IEEE 13th Conference on Commerce and Enterprise Computing (CEC)*, 26–33. IEEE.

Jakob, N., Weber, S. H., Müller, M. C. & Gurevych, I. 2009. Beyond the stars: exploiting free-text user reviews to improve the accuracy of movie recommendations. In *Proceedings of the 1st International CIKM Workshop on Topic-Sentiment Analysis for Mass Opinion*, 57–64. ACM.

Kazienko, P. & Musiał, K. 2006. *Recommendation Framework for Online Social Networks*. Springer.

Leskovec, J., Rajaraman, A. & Ullman, J. D. 2014. *Mining of Massive Datasets*. Cambridge University Press.

Leung, C. W.-K., Chan, S. C.-F. & Chung, F.-L. 2008. An empirical study of a cross-level association rule mining approach to cold-start recommendations. *Knowledge-Based Systems* **21**(7), 515–529.

Levi, A., Mokryn, O., Diot, C. & Taft, N. 2012. Finding a needle in a haystack of reviews: cold start context-based hotel recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, 115–122. ACM.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press

Levinson, N. 1946. The Wiener (root mean square) error criterion in filter design and prediction. *Journal of Mathematics and Physics* **25**(1), 261–278.

Linden, G., Smith, B. & York, J. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE* **7**(1), 76–80.

Liu, H., He, J., Wang, T., Song, W. & Du, X. 2013. Combining user preferences and user opinions for accurate recommendation. *Electronic Commerce Research and Applications* **12**(1), 14–23.

Lops, P., De Gemmis, M. & Semeraro, G. 2011. Content-based recommender systems: state of the art and trends. In *Recommender Systems Handbook*. Springer, 73–105.

MacKay, D. J. C. 1998. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences* **168**, 133–166.

Manning, C. D., Raghavan, P. & Schütze, H., *et al*. 2008. *Introduction to Information Retrieval*, **1**. Cambridge University Press.

Massa, P. & Avesani, P. 2007. Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and Information Systems* **3**(1), 39–64.

Massa, P. & Bhattacharjee, B. 2004. Using trust in recommender systems: an experimental analysis. In *Trust Management*. Springer, 221–235.

Mayfield, J. & McNamee, P. 2003. Single N-Gram stemming. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 415–416. ACM.

McAuley, J. & Leskovec, J. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, 165–172. ACM.

McAuley, J., Pandey, R. & Leskovec, J. 2015a. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. ACM.

McAuley, J., Targett, C., Shi, Q. & van den Hengel, A. 2015b. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52. ACM.

Middleton, S. E., Shadbolt, N. R. & De Roure, D. C. 2004. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)* **22**(1), 54–88.

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill.

Mooney, R. J. & Roy, L. 2000. Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, 195–204. ACM.

O'Donovan, J. & Smyth, B. 2005. Trust in recommender systems. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, 167–174. ACM.

Palaniswami, M. & Shilton, A. 2002. Adaptive support vector machines for regression. In *Proceedings of the 9th International Conference on Neural Information Processing*, *ICONIP'02*, 1043–1049. IEEE.

Paterek, A. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, 5–8.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, 285–295. ACM.

Schein, A. I., Popescul, A., Ungar, L. H. & Pennock, D. M. 2002. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 253–260. ACM.

Soleymani, M., Aljanaki, A., Wiering, F. & Veltkamp, R. C. 2015. Content-based music recommendation using underlying music preference structure. In *IEEE International Conference on Multimedia and Expo (ICME)*, 1–6. IEEE.

Sugiyama, K., Hatano, K. & Yoshikawa, M. 2004. Adaptive web search based on user profile constructed without any effort from users.In *Proceedings of the 13th International Conference on World Wide Web*, 675–684. ACM.

Ungar, L. H. & Foster, D. P. 1998. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, volume 1, 114–129.

Vapnik, V. 2013. *The Nature of Statistical Learning Theory*. Springer Science & Business Media.

Wang, J., Yin, J., Liu, Y. & Huang, C. 2011. Trust-based collaborative filtering. In *Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD),* 2650–2654. IEEE.

Wang, S.-C. 2003. Artificial neural network. In *Interdisciplinary Computing in Java Programming*. Springer, 81–100.

Williams, C. K. I. & Rasmussen, C. E. 2006. Gaussian processes for machine learning. *The MIT Press* **2**(3), 4.

Willmott, C. J. & Matsuura, K. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research* **30**(1), 79–82.

Yang, D., Zhang, D., Yu, Z. & Wang, Z. 2013. A sentiment-enhanced personalized location recommendation system. In *Proceedings of the 24th ACM Conference on Hypertext and Social Media*, 119–128. ACM.

https://doi.org/10.1017/S0269888918000218 Published online by Cambridge University Press