Latent Linear Critiquing for Conversational Recommender Systems

Paper ID 9446

Abstract

Critiquing is a method for conversational recommendation that iteratively adapts recommendations in response to user preference feedback. In this setting, a user is iteratively provided with an item recommendation and attribute description for that item; a user may either accept the recommendation, or critique the attributes in the item description to generate a new recommendation. Historical critiquing methods were largely based on explicit constraint- and utility-based methods for modifying recommendations w.r.t. critiqued item attributes. In this paper, we revisit the critiquing approach in the era of recommendation methods based on latent embeddings with subjective item descriptions (i.e., keyphrases from user reviews). Two critical research problems arise: (1) how to co-embed keyphrase critiques with user preference embeddings to update recommendations, and (2) how to modulate the strength of multi-step critiquing feedback, where critiques are not necessarily independent, nor of equal importance. To address (1), we build on an existing state-of-the-art linear embedding recommendation algorithm to align review-based keyphrase attributes with user preference embeddings. To address (2), we exploit the linear structure of the embeddings and recommendation prediction to formulate a linear program (LP) based optimization problem to determine optimal weights for incorporating critique feedback. We evaluate the proposed framework on two recommendation datasets containing user reviews. Empirical results compared to a standard approach of averaging critique feedback show that our approach reduces the number of interactions required to find a satisfactory item and increases the overall success rate.

Introduction

Critiquing is a method for conversational (a.k.a. sequential interactive) recommendation that adapts recommendations in response to user preference feedback regarding item attributes. For example, in unit critiquing (Burke, Hammond, and Young 1996), a user might critique a digital camera recommendation by requesting an item with higher resolution and in compound critiquing (Reilly et al. 2004a; 2005), a user might further explore items that have longer battery life *and* lower price than an initial recommendation. Further extensions such as incremental critiquing consider the cumulative effect of iterated critiquing interactions (Reilly et al. 2004b) while experience-based methods attempt to collaboratively leverage critiquing interactions from multiple users (McCarthy, Salem, and Smyth 2010).

These historical critiquing methods were largely based on constraint- and utility-based methods for modifying recommendations w.r.t. critiques of explicitly known item attributes. In this paper, we revisit the critiquing approach in the era of state-of-the-art recommendation methods based on latent embeddings (Sedhain et al. 2015; Wu et al. 2016; Sedhain et al. 2016a; He et al. 2017; Liang et al. 2018). In addition, we assume that item attributes are *not explicitly known*, but rather represented as keyphrases sourced from subjective user reviews. Though some work has focused on explanations in critiquing (Reilly et al. 2005) and other work (Grasch, Felfernig, and Reinfrank 2013; Thompson, Goker, and Langley 2004) has respectively explored speech- and dialog-based interfaces for critiquing-style frameworks, these architectures have nonetheless assumed that item attributes are explicitly known *a priori*.

In our novel setting of *latent critiquing*, it is not immediately clear how critiques of subjective keyphrases should be incorporated into the latent user preference representation to modulate future recommendations. To this end, we formalize and address two critical research problems in this paper:

(1) How can we co-embed item critiques with general user preference information to properly take the critiques into account for subsequent recommendations?

(2) How can we appropriately modulate the strength of each critique in multi-step critiquing feedback, where critiques are not necessarily independent, nor of equal importance?

To address problem (1), we extend an existing state-of-the-art linear embedding recommendation algorithm that we term Projected Linear Recommendation (PLRec) (Sedhain et al. 2016a) to embed user preferences and align review-based keyphrase attributes with those embeddings. To address problem (2), we exploit the linear structure of the embeddings and recommendation prediction to formulate a linear program (LP) based optimization problem to determine optimal weights for incorporating critique feedback. To this

end, we refer to our overall framework as *latent linear critiquing* (LLC) since it exploits linear embedding structure to perform critiquing in a latent recommendation setting.

We evaluate our proposed methods for LLC on two recommendation datasets containing user reviews. Empirical results compared to standard approaches for averaging critique feedback show that our LP LLC approaches reduce the number of interactions required to find a satisfactory item and increases the overall percentage of successfully retrieved items in our experimental setting. We further investigate variants of our LP LLC formulation and also investigate performance for popular and long-tail items. In summary, this paper provides a modern update of the critiquing framework to combine latent embedding based recommendation with critiques of subjective keyphrase item descriptions in what we hope to be a rich space for future research.

Preliminaries

Notation

Before proceeding, we define the following notation:

- $R \in \mathbb{R}^{|I| \times |J|}$. This is a user preference matrix. Entries $r_{i,j}$ are either 1 (preference observed) or 0 (preference not observed). $\mathbf{r}_i$ represents all feedback from user i, and $\mathbf{r}_{:,j}$ represents all user feedback for item j.
- $S \in \mathbb{R}^{|I| \times |K|}$. This is the user-keyphrase matrix. Given user reviews from a corpus, we extract keyphrases that describe item attributes from all reviews as shown in Table 1. This matrix contains users and term frequencies of keyphrases. We use $\mathbf{s}_i$ to represent ith user's keyphrase frequencies, and $\mathbf{s}_{:,k}$ to represent kth keyphrase's frequency across all users.
- $S' \in \mathbb{R}^{|J| \times |K|}$. This is the item-keyphrase matrix. Given item reviews written by all users from a corpus, we extract keyphrases that describe item attributes as shown in Table 1. This matrix contains items and term frequencies of keyphrases. We use $\mathbf{s}'_j$ to represent jth item's keyphrase frequencies, and $\mathbf{s}'_{:,k}$ to represent kth keyphrase's frequency across all items.
- $j^{-k} \in \{j | S'_{j,k} = 0, \forall j\}$. This item set represents items that do not contain the critiqued keyphrase k.
- $j^{+k} \in \{j | S'_{j,k} > 0, \forall j\}$. This item set represents items that contain the critiqued keyphrase k.

Projected Linear Recommendation

Linear Recommendation methods learn a matrix representing either user-user or item-item similarity by casting the problem in a linear regression framework (Ning and Karypis 2011; Sedhain et al. 2016b). An unconstrained linear recommender uses an objective of the following form:

$$\underset{W}{\operatorname{argmin}} \sum_i ||\mathbf{r}_i - \mathbf{r}_i W^T||_2^2 + \Omega(W), \tag{1}$$

where W represents the similarity matrix to train, and Ω is a regularization term. Unfortunately, in modern recommendation problems that have large numbers of users and items, such naïve linear recommendation approaches are infeasible due to the space requirements of explicitly storing the similarity matrix W.

Table 1: The two datasets we use in this paper along with example keyphrases extracted from reviews in the dataset.

Dataset	Reason Type	Keyphrases
Beer	Head	white, tan, offwhite, brown
	Malt	roasted, caramel, pale, wheat, rye
	Color	golden, copper, orange, black, yellow
	Taste	citrus, fruit, chocolate, cherry, plum
CDs&Vinyl	Genre	rock, pop, jazz, rap, hip hop, R&B
	Instrument	orchestra, drum
	Style	concert, opera
	Religious	chorus, christian, gospel

Projected Linear Recommendation (PLRec) is a term that we use for the state-of-the-art recommendation method of (Sedhain et al. 2016a) that mitigates the scalability problem described above by projecting preferences from R into a reduced-dimension embedded space prior to linear regression. Formally, the PLRec objective can be represented as

$$\underset{W}{\operatorname{argmin}} \sum_i ||\mathbf{r}_i - \mathbf{r}_i V W^T||_2^2 + \Omega(W), \tag{2}$$

where parameters W are learned and V is a fixed embedding projection matrix. It is critical to note here that because V is fixed, the above objective still leads to a convex linear regression problem. PLRec obtains V by taking a low-rank SVD approximation of the observation matrix R such that $R = U \Sigma V^T$, and the rank $|L|$ of V is far smaller than the observation dimensions $|I|$ and $|J|$. We denote the projected, embedded representation of user i as $\mathbf{z}_i = \mathbf{r}_i V$.

Conversational Critiquing

In the conversational critiquing setting of this paper, a user is iteratively provided with item recommendations and keyphrase descriptions for that item; a user may either critique the keyphrases in the item description or accept the item recommendation, at which point the iteration terminates. Two examples of such a conversational critiquing interaction from our experimentation are provided in Table 5.

Formally, critiquing at a single time step can be viewed as a series of functional transformations that produce a modified prediction $\hat{\mathbf{r}}_i$ of item preferences for user i given critiqued item keyphrases $\mathbf{s}_i$ as follows:

$$\hat{\mathbf{r}}_i = f_m(\mathbf{r}_i, \tilde{\mathbf{s}}_i), \quad \textit{given} \quad \tilde{\mathbf{s}}_i = \psi(\mathbf{s}_i, \mathbf{c}_i), \tag{3}$$

where the critique-modified recommendation function f_m takes user preferences $\mathbf{r}_i$ and critiqued keyphrases $\tilde{\mathbf{s}_i}$ as input and produces a recommendation $\hat{\mathbf{r}}_i$ as output. The function ψ applies a user critiquing action $\mathbf{c}_i$ to user keyphrases $\mathbf{s}_i$.

In the case of critiquing over multiple time steps as demonstrated in Figure 1, a user is iteratively provided with the item recommendations $\tilde{\mathbf{r}}_i^t$ for each time step t, and, based on the recommendations, the user may make a new critique $\mathbf{c}_i^t$ and update the representation of critiqued keyphrases

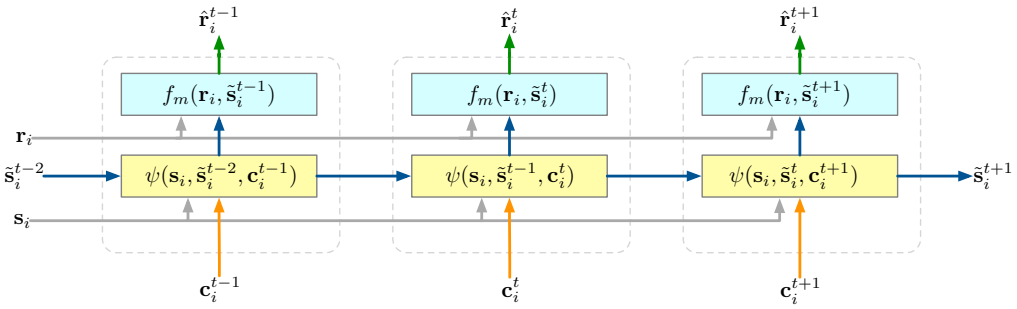


Figure 1: The flow of conversational critiquing over three time steps. Previously critiqued keyphrases $\tilde{\mathbf{s}}_i^{t-2}$ at time $t-2$ are combined by ψ with the newly critiqued keyphrase $\mathbf{c}_i^{t-1}$ at time $t-1$ to yield $\tilde{\mathbf{s}}_i^{t-1}$. The recommendation produced by f_m at time $t-1$ for user i is produced from the user's historical preferences $\mathbf{r}_i$ and cumulative critiques $\tilde{\mathbf{s}}_i^{t-1}$ up to time $t-1$. This process repeats for all t until the user accepts the recommendation (or terminates) and ceases to provide additional critiques.

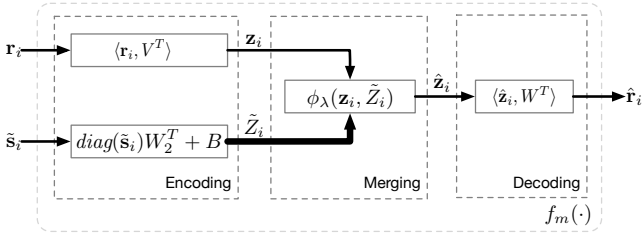


Figure 2: The structure of critiqued recommendation $f_m(\cdot)$. Historical user preferences $\mathbf{r}_i$ and critiqued keyphrases $\tilde{\mathbf{s}}_i$ from user i flow in at the left and are embedded into respective latent spaces (i.e., a single vector embedding $\mathbf{z}_i$ for preferences and a matrix $\tilde{Z}_i$ containing embeddings for each critique). These latent embeddings are then combined in the merging stage (a critical contribution of the paper) and finally decoded into new item recommendations $\hat{\mathbf{r}}_i$ for user i on the right that take into account their critiques.

$\tilde{\mathbf{s}}_i^{t-1}$ though a cumulative critiquing function ψ:

$$\tilde{\mathbf{s}}_i^t = \psi(\mathbf{s}_i, \tilde{\mathbf{s}}_i^{t-1}, \mathbf{c}_i^t) \tag{4}$$

The user may alternately accept the item recommendation, at which point the iteration terminates.

At this point we have defined the generic functional structure of the overall critiquing process, but we still need to provide formal definitions of f_m and ψ to address the critical problem of how keyphrase critiques modulate recommendations. We now proceed to do this in the specific setting of latent linear conversational critiquing that extends and leverages the linearity of PLRec.

Latent Linear Conversational Critiquing

In this section, we begin by extending the PLRec framework to embed language-based feedback mined from user reviews that will be used for co-embedding critiques with user preferences from PLRec. We then show how to combine critiques in a linear framework where we propose methods for both simple critique averaging and a more sophisticated framework for deriving critique weights by leveraging a novel linear programming optimization formalization.

Co-embedding of Language-based Feedback

First we augment the PLRec framework with the ability to embed language-based critiquing feedback in the same space as user preferences. To achieve this, we leverage recommendation datasets with both preference and review feedback from users.

PLRec naturally embeds user preferences over items into a latent space as described in Equation (2). Once this latent embedding for a user is obtained, we learn to embed keyphrase-based critiques for a user by training to recover a user's latent preference embedding *from* the preference content implicitly revealed through their reviews.

To make this concrete, for each user i, PLRec encodes their latent preference representation $\mathbf{z}_i$ as in Equation (2). In addition, we have review content for each user represented as a term frequency vector (user keyphrases) $\mathbf{s}_i$. With this, we can now cast the co-embedding task as the following linear regression problem:

$$\underset{W_2, \mathbf{b}}{\operatorname{argmin}} \sum_i ||\mathbf{z}_i - \tilde{\mathbf{z}}_i||_2^2 + \Omega(W_2), \tag{5}$$

where

$$\tilde{\mathbf{z}}_i = \mathbf{s}_i W_2^T + \mathbf{b} \tag{6}$$

and $W_2 \in \mathbb{R}^{|K| \times |L|}$ projects users' review text to into their latent representation and $\mathbf{b}$ is a bias term. We will show how to use the learned regression model to conduct critiquing in next section.

Latent Linear Critiquing

In this section, we aim to specify how the critiquing-based recommendation system can make new recommendations after a user i has provided critiques $\mathbf{c}_i^1 \cdots \mathbf{c}_i^t$ over t iterations (as demonstrated in Figure 1 and 2), where critiques $\mathbf{c}_i^t$ are encoded as one-hot keyphrase indicators that represent a user i's dislike of a keyphrase description at time step t.

We start off by recalling that for a fixed user i and item j, the prediction of an item preference $\hat{r}_{i,j}$ can be written as an inner product $\langle \cdot, \cdot \rangle$ of the user and item embedding:

$$\hat{r}_{i,j} = \langle \mathbf{z}_i, \mathbf{w}_j \rangle, \tag{7}$$

where user representation $\mathbf{z}_i$ comes from the embedding of a user's historical preferences and $\mathbf{w}_j$ is the row of W corresponding to item j's latent embedding.

As mentioned previously, to transform the user's critiques $\mathbf{c}_i$ into an embeddable term frequency representation that can be co-embedded with user preference embeddings, we use the cumulative critiquing function defined as follows:

$$\tilde{\mathbf{s}}_i^t = \psi(\mathbf{s}_i, \tilde{\mathbf{s}}_i^{t-1}, \mathbf{c}_i^t) = \tilde{\mathbf{s}}_i^{t-1} - \mathbf{s}_i \odot \mathbf{c}_i^t, \tag{8}$$

where $\odot$ represents element-wise multiplication of two vectors and the initial $\tilde{\mathbf{s}}_i^0$ is a zero vector of length $|K|$.

With the critiqued keyphrases $\tilde{\mathbf{s}}_i^t$ and the mapping between keyphrase and user latent representation learned in Equation (5), we are able to provide a latent representation of *all* critiques in matrix form

$$\tilde{Z}_i^t = diag(\tilde{\mathbf{s}}_i^t) W_2^T + B, \tag{9}$$

where each row $\tilde{\mathbf{z}}_i^k$ of the matrix $\tilde{Z}_i$ represents latent representation of the kth critiqued keyphrase, and each row of B is the identical bias term $\mathbf{b}$. While the matrix $\tilde{Z}_i$ appears large, it is sparse and simply used here for notational convenience to represent *stacking* of all critiqued keyphrase embeddings in matrix form.

Then under an assumption that critiques at each time step should be weighted equally (with the user preference), we can perform simple **Uniform Average Critiquing** by defining the merging function as

$$\phi_\lambda(\mathbf{z}_i, \tilde{Z}_i^t) = \lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}_i^1 \cdots \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|} \tag{10}$$

where $\lambda_0, \lambda_1, \ldots, \lambda_{|K|}$ are identical and sum to 1 (hence all λ's equal $\frac{1}{|K|+1}$). Note that we omit the identical time step t superscripts to reduce notational clutter. Clearly, this merging function averages the user preferences and critiqued keyphrase embeddings uniformly. We can then obtain the *refined* recommendation score $\hat{r}_{i,j}^t$ of each user i for item j at time step t by decoding the latent representation and rearranging terms as follows:

$$\begin{aligned}
\hat{r}_{i,j}^t &= \langle \phi_\lambda(\mathbf{z}_i, \tilde{Z}_i), \mathbf{w}_j \rangle \\
&= \langle (\lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}_i^1 + \cdots + \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|}), \mathbf{w}_j \rangle \\
&= \frac{1}{|K|+1} \left(\langle \mathbf{z}_i, \mathbf{w}_j \rangle + \langle \tilde{\mathbf{z}}_i^1, \mathbf{w}_j \rangle + \cdots + \langle \tilde{\mathbf{z}}_i^{|K|}, \mathbf{w}_j \rangle \right),
\end{aligned} \tag{11}$$

where the second line substitutes the definition of $\phi_\lambda(\mathbf{z}_i, \tilde{Z}_i)$ from Equation (10) and the final line distributes the inner product over the summation but factors out the common coefficient.

An alternative average critiquing approach, we call **Balanced Average Critiquing**, is that all the critique embeddings $\tilde{Z}_i$ are averaged together and then averaged again with the embedding of user preferences. Formally, the balanced average critiquing is defined as:

$$\begin{aligned}
\phi_\lambda(\mathbf{z}_i, \tilde{Z}_i^t) &= \lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}_i^1 + \cdots \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|} \\
&= \frac{1}{2} (\mathbf{z}_i + \frac{1}{|K|} (\tilde{\mathbf{z}}_i^1 + \cdots \tilde{\mathbf{z}}_i^{|K|})),
\end{aligned} \tag{12}$$

where $\lambda_0 = \frac{1}{2}$ and the rest of the λs equals to $\frac{1}{2 \times |K|}$. Clearly, this always weights the embeddings from a user's historical preferences the same as the total embedding of the critiques (no matter how many critiques there are). This user-balanced average reflects the notion that user preferences after critiques should not deviate too drastically from the base user preferences to begin with.

Unfortunately, the simple average critique weighting assumption described above can be violated in a few ways in practice:

- Not all critiques are equal. More specifically, later critiques may be more subtle refinements of earlier critiques.
- Not all critiques are independent. For instance, two critiques may substantially overlap (and thus should split their effective impact) while another critique may be more orthogonal to the first two and deserve more weight.

Therefore, we look for better solutions by seeking a way to optimize the weights used to merge critiques.

Linear Programming Critiquing Optimization

Based on the above criticisms, we revisit the merging function in its basic form

$$\phi_\lambda(\mathbf{z}_i, \tilde{Z}_i^t) = \lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}_i^1 \cdots \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|} \tag{13}$$

and consider that each critique could be weighted with λ_n individually as follows:

$$\begin{aligned}
r_{i,j}^t &= \langle \lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}_i^1 \cdots \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|}, \mathbf{w}_j \rangle \\
&= \langle \lambda_0 \mathbf{z}_i, \mathbf{w}_j \rangle + \langle \lambda_1 \tilde{\mathbf{z}}_i^1, \mathbf{w}_j \rangle + \cdots + \langle \lambda_{|K|} \tilde{\mathbf{z}}_i^{|K|}, \mathbf{w}_j \rangle,
\end{aligned} \tag{14}$$

which gives maximum flexibility to merging function. Now, the question is how can we automatically find a better non-uniform weighting, i.e., optimal λ values?

In short, we can view this question as an optimization task of the λ's in Equation (14) according to an objective and constraints that we think reflect what an optimum weighting of λ's should be. While the design of such an optimization problem is inherently subjective, we believe there are a few key desiderata of such optimal λ's:

- The rating of items known to have the critiqued keyphrase description (j^{+k}) should be pushed as low as possible. To achieve this, we choose to explicitly penalize the top 100 ranked items from the PLRec *with* the critiqued keyphrase in the optimization objective. This is under assumption that low ranked items should not affect the intended target item's rank.
- The rating of items not containing the critiqued keyphrase (j^{-k}) should be pushed as high as possible. To achieve this, we choose to explicitly promote the top 100 ranked items from PLRec *without* the critiqued keyphrase in the optimization objective.
- Because we are ranking and λ weights could be infinite if unbounded, we need to ensure the λ remain within a reasonable range.

Optimization Objective

Based on the above desiderata, we arrive at the following objective of optimal weighting for each time step t:

$$\min \sum_{j^{+k}} \sum_{j^{-k}} \hat{r}^t_{i,j^{+k}} - \hat{r}^t_{i,j^{-k}} \tag{15}$$

Critically, we remark that since all of the embeddings are fixed, we see that the objective in Equation (15) is linear in the optimized variables and this is in fact a simple linear programming (LP) formulation of our desiderate. This is a fortuitous observation since LPs can be solved efficiently at scale (millions of variable and constraints) that can facilitate practical, scalable deployment of this optimal weighting approach.

In addition to the primary objective function, we propose three different candidates for constraints on this objective:

LP Option 1 Here, we assume the weighting of the original user preferences remain unchanged. For the critiques though, we allow for the fact that some critiques may partially cancel out others (i.e., having a negative weight) in the event of non-independence. Thus, we restrict λ's according to the following constraints:

$$\lambda_0 = 1, \lambda_{1 \cdots |K|} \in [-1, +1] \tag{16}$$

LP Option 2 Here, we again assume the weighting of the original user preferences remain unchanged, but now we allow for a much larger range for the λ's that may allow for more extreme weighting of critiques relative to the user preferences:

$$\lambda_0 = 1, \lambda_{1 \cdots |K|} \in [-r, +r], \quad \textit{and} \quad r \in \mathbb{Z}_{\gg 0} \tag{17}$$

LP Option 3 In this setting, we require all λ's to be positive, but further constraint that they all sum to one, thus making any increase in a single λ occur only if another is decreased. Here, we also allow the LP to zero out the impact of the user preferences if it believes this is optimal:

$$\sum_{n=0}^{|K|} \lambda_n = 1, \quad \textit{and} \quad \lambda_{0 \cdots |K|} \in [0, 1] \tag{18}$$

Experiments

Dataset

We evaluate the proposed latent linear critiquing framework on two publicly available datasets: BeerAdvocate (McAuley, Leskovec, and Jurafsky 2012) and Amazon CDs&Vinyl (McAuley et al. 2015; He and McAuley 2016). Each of the datasets contains more than 100,000 reviews and product rating records. For the purpose of Top-N recommendation, we binarize the rating column of both datasets with a rating threshold ϑ. In CDs&Vinyl, the threshold is $\vartheta > 3$ out of 5. In BeerAdvocate, we define the rating threshold $\vartheta > 4$ out of 5. Table 2 shows overall dataset statistics for our experiments.

Table 2: Summary of datasets. We selected 40 keyphrases for CDs&Vinyl and 75 keyphrases for BeerAdvocate. Keyphrase coverage shows the percentage of reviews/comments that have at least one selected keyphrase.

Dataset	# Users	# Items	$\lvert r_{i,j} > \vartheta \rvert$	Rating Sparsity	Keyphrase Coverage	Keyphrase Average Counts (per User)
CDs&Vinyl (Amazon)	6,056	4,395	152,670	0.5736%	75.48%	13.9969
Beer (BeerAdvocate)	6,370	3,668	263,278	1.1268%	99.29%	55.1088

Automated Keyphrase Extraction

The datasets do not contain pre-selected keyphrases. Hence, we used the following generic processing steps to extract candidate keyphrases from the reviews to be used for explanation and critiquing for each dataset:

1. Extract separate unigram and bigram lists of high frequency noun and adjective phrases from reviews of the entire dataset.
2. Prune the bigram keyphrase list using a Pointwise Mutual Information (PMI) threshold to ensure bigrams are statistically unlikely to have occurred at random.
3. Represent each review as a sparse 0-1 vector indicating whether each keyphrase occurred in the review.

Recall that keyphrase examples were provided in Table 1.

Critiquing User Simulation Evaluation

In order to perform an evaluation of each model's performance in a multi-step conversational recommendation scenario using offline data provided in our datasets, we conduct an evaluation by user simulation. Concretely, as described in Algorithm 1, we track the conversational interaction session of simulated users by randomly selecting a target item from their test set, having the user critique keyphrases inconsistent with the target item, and repeating until an iteration limit or the target item appears within the top-N recommendations on that iteration. For each user, we collect results from three simulated sessions and estimate the average success rate as well as the average session length for model comparison.

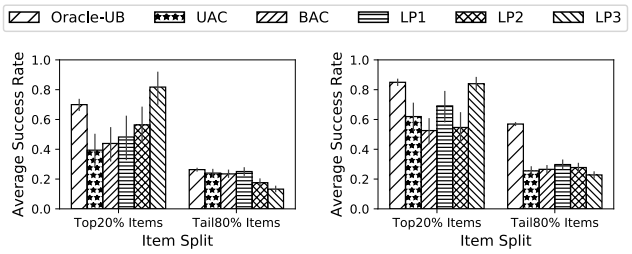
Algorithm 1 User Simulation Evaluation

procedure EVAL(R^{te} for test)
  for each user i **do**
    for each target item j, where $r^{te}_{i,j} = 1$ **do**
      for time step $t \in range(1, MAX)$ **do**
        user act critique $\mathbf{c}^t$
        optimize $\lambda_0 ... \lambda_t$
        $\hat{r}^t_{i,j} \leftarrow \langle \lambda_0 \mathbf{z}_i + \lambda_1 \tilde{\mathbf{z}}^1_i + \cdots \lambda_{|K|} \tilde{\mathbf{z}}^{|K|}_i, \mathbf{w}_j \rangle$
        if j in Top-N recommendation list **then**
          break session with success
      length $\leftarrow min(t, MAX)$
  return average success rate & length

In the experiments, we report average length of sessions and average success rate. The Top-N ranking threshold in

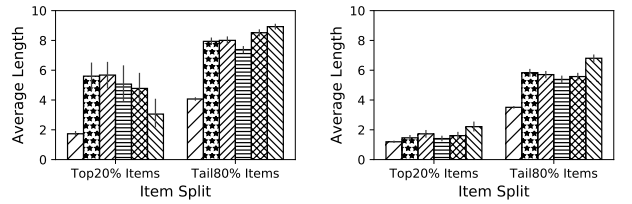
Table 3: Percentage of target items that are successfully retrieved within Top1, Top5 and Top10 on Amazon CDs&Vinyl and Beer datasets. We omit the error bars since the confidence interval is in 4th digit.

Model	CDs&Vinyl			Beer		
	Success Rate@1	Success Rate@5	Success Rate@10	Success Rate@1	Success Rate@5	Success Rate@10
Oracle-UB	0.1245	0.2714	0.3577	0.3722	0.5837	0.6712
UAC	0.1217	0.2452	0.3109	0.1026	0.2827	0.3974
BAC	0.1083	0.2465	0.3257	0.1135	0.2825	0.3892
LP1	**0.1236**	**0.2577**	**0.3312**	**0.1357**	**0.3217**	**0.4268**
LP2	0.0841	0.1871	0.2516	0.1267	0.2967	0.4105
LP3	0.0584	0.1455	0.2044	0.1084	0.2672	0.361



(a) CDs&Vinyl (b) Beer

Figure 3: Long tail item analysis average success rate at target rank 5 comparison for different critiquing algorithms on CDs&Vinyl and Beer with 95% confidence interval. All figures share the legend.



(a) CDs&Vinyl (b) Beer

Figure 4: Long tail item analysis average length at target rank 5 comparison for different critiquing algorithms on CDs&Vinyl and Beer with 95% confidence interval. All figures share the legend.

this experiment is selected from $\{1, 5, 10\}$, which varies the difficulty of finding the target item. The maximum allowed critiquing iterations in the Algorithm is set to 10.

Critiquing Algorithms Comparison

Before introducing our evaluation models, we first introduce a model that has omniscient knowledge of target items and the correctness of keyphrase descriptions that we term "Oracular Upper Bound".

In general we cannot assume keyphrase labels are accurate descriptions nor can we assume that all relevant keyphrase labels occur for an item (they are derived from sparse user review content) and hence in practice we cannot simply prune all items that contain a critiqued keyphrase.

However, if we "cheat" and assume knowledge of the user simulation and the perfect nature of keyphrase labels that are provided to the user in the simulation, we can define the "Oracular Upper Bound" that perfectly prunes the recommendation list after each critique. Because this assumes knowledge of the user simulation that our proposed methods do not assume, this method is useful as deriving an upper bound on performance of a system with perfect keyphrase descriptions.

Now, we list the following critiquing models used in our experiments:

- **Oracle-UB:** Oracular upper bound. As defined previously, a simple model that acts as an "oracle" and knows a target item will not have a critiqued keyphrase by design and that all keyphrase labels are accurate. It accumulates all critiqued keyphrases and explicitly removes all items reviewed with any critiques.
- **UAC:** Uniform Average Critiquing, which we have described in Equation (10).
- **BAC:** Balanced Average Critiquing, which we have described in Equation (12).
- **LP 1-3** Respectively, see Equations (16)–(18).

Critiquing Empirical Results

Average Success Rate Table 3 shows the percentage of target items that reach target ranks before the session terminates. The results illustrate that Oracle-UB behaves as an upper bound with knowing additional keyphrase information and it is expected that our proposed models cannot beat it. It is obvious that the proposed LP1 outperforms two baselines and other LP variants in both datasets in terms of all target ranks. This is because LP1 not only takes into consideration the base user preferences but also leverages critiquing feedback with appropriate weighting. On the Beer dataset, LP2 retrieves more target items than both baselines in terms of all target ranks.

From Table 2, we know that user interaction and keyphrase information in Beer dataset are much denser than CDs&Vinyl dataset. This helps all models achieve higher performance on Beer than CDs&Vinyl dataset especially for Oracle-UB. Given the experiment setting that it always critiques the most popular keyphrase and removes items as many as possible, there would not be many unaffected items left before session terminates if keyphrase information is dense enough.

Table 4: Average length of sessions given target ranks as Top1, Top5 and Top10 on Amazon CDs&Vinyl and Beer datasets with 95% confidence interval.

Model	CDs&Vinyl			Beer		
	Average Length@1	Average Length@5	Average Length@10	Average Length@1	Average Length@5	Average Length@10
Oracle-UB	4.6211± 0.0894	4.005± 0.072	3.6344± 0.0627	4.5044± 0.0647	3.3933± 0.0478	2.9333± 0.0414
UAC	8.9841± 0.1451	7.8703± 0.2026	7.3012± 0.2167	6.4514± 0.217	5.597± 0.2276	5.0562± 0.2301
BAC	9.1174± 0.1411	7.8898± 0.2107	7.1864± 0.2293	6.2641± 0.2007	5.4797± 0.2085	4.9446± 0.2099
LP1	**8.4132± 0.1393**	**7.327± 0.2067**	**6.7956± 0.2192**	**5.9248± 0.2141**	**5.1876± 0.227**	**4.7397± 0.2294**
LP2	9.2646± 0.1448	8.3975± 0.1998	7.8449± 0.2221	6.0956± 0.2105	5.3104± 0.2246	4.8067± 0.2183
LP3	9.5285± 0.0951	8.8102± 0.1552	8.3335± 0.1831	6.5575± 0.2115	6.5326± 0.2126	6.4371± 0.2157

Table 5: User Case Study for target rank 1 on the CDs&Vinyl and Beer datasets.

Dataset	Time Step t	Recommended Item	Top Keyphrases Describing the Item	Critiqued Keyphrase	Sucessfully Retrieved?
CDs&Vinyl	0	No Code	Rock, Metal, Rap	Rap	-
	1	Jar Of Flies	Rock, Pop, Ballad	Pop	-
	2	Pearl Jam	Rock, Metal, Instrument	-	✓
Beer	0	Sierra Nevada Southern Hemisphere Harvest Fresh Hop Ale	Citrus, Sweet, Caramel	Citrus	-
	1	Maharaja	Sweet, Orange, Grapefruit	Grapefruit	-
	2	Fuller's ESB	Caramel, Fruit, Copper	-	✓

Compared to LP2 and LP3's average success rate on CDs&Vinyl dataset, their performance are much more competitive on the Beer dataset. This shows that relying more on the original user preference achieves better performance with lack of keyphrase information. When the keyphrase information is noisy, extreme weighting on critiquing feedback (LP2) or having option to weight less on the base user preferences (LP3) are not necessary.

Average Session Length Table 4 shows the average length of sessions given target ranks as Top1, 5 and 10. Similar to the average success rate of retrieving target items, Oracle-UB acts as an upper bound due to its definition. LP1 consistently outperforms two baselines and other LP on both datasets in terms of all three target ranks. On Beer dataset, LP2 has fewer average session length than both baselines in terms of all target ranks. For the same reason caused by dataset density, all models have fewer average session length on Beer than CDs&Vinyl dataset. Compared to LP2 and LP3, LP1 always performs better. This indicates that weighting extremely on critiquing feedback or having option to potentially ignore the base user preferences are not necessary.

Long-tail Item Analysis Figures 3 and 4 illustrate long tail item analysis on both of Amazon CDs&Vinyl and beer datasets. In this analysis, we first split items based on their popularity into top 20% items and tail 80% items. Then we report average success rate and average session length in terms of those two item pools and all target ranks.

For both metrics, it is expected that all models perform better in terms of top 20% items. LP3 provides competitive performance in terms of top 20% items. This shows that having option to ignore the base user preference and solely relying on popularity bias with critique feedback help in finding popular items.

For tail 80% items, LP1 outperforms LP2 and LP3, which shows that base user preferences help to retrieve more tail 80% items.

Case Study To qualitatively evaluate the performance of the critiquing in a real environment, we simulated two use-cases of the proposed model (LP1) on the two review datasets as we have described in evaluation. Table 5 shows two representative examples we encountered during our investigation. Both cases start with a test user, a target rank, a target item and its Top-3 most popular described keyphrases. Each use case ends until the target item is successfully retrieved.

Conclusion

In this paper, we proposed a novel method for combining multi-step critiquing style conversational recommendation with modern recommender systems based on latent embeddings. We proposed an extension to PLRec to allow it to co-embed keyphrase critiques with embeddings of user preferences. We further defined a linear latent critiquing framework that exploited the linearity of PLRec to produce both standard averaging approaches to combining user preferences and critiques as well an optimization setting based on linear programming (LP). We evaluated the proposed framework on two recommendation datasets containing user reviews. Empirical results compared to a standard approach of averaging critique feedback showed that our best LP approach reduces the number of interactions required to find a satisfactory item and increases the overall success rate over all competing methods.

References

Burke, R. D.; Hammond, K. J.; and Young, B. C. 1996. Knowledge-based navigation of complex information spaces. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, 462–468. AAAI Press.

Grasch, P.; Felfernig, A.; and Reinfrank, F. 2013. Recomment: Towards critiquing-based recommendation

with speech interaction. In *Proceedings of the 7th ACM Conference on Recommender Systems (RECSYS)-13*, 157–164.

He, R., and McAuley, J. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, 507–517. International World Wide Web Conferences Steering Committee.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, 173–182. International World Wide Web Conferences Steering Committee.

Liang, D.; Krishnan, R. G.; Hoffman, M. D.; and Jebara, T. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, 689–698.

McAuley, J.; Targett, C.; Shi, Q.; and Van Den Hengel, A. 2015. 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.

McAuley, J.; Leskovec, J.; and Jurafsky, D. 2012. Learning attitudes and attributes from multi-aspect reviews. In *2012 IEEE 12th International Conference on Data Mining*, 1020–1025. IEEE.

McCarthy, K.; Salem, Y.; and Smyth, B. 2010. Experience-based critiquing: Reusing critiquing experiences to improve conversational recommendation. In *International Conference on Case-Based Reasoning*, 480–494. Springer.

Ning, X., and Karypis, G. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, 497–506. IEEE.

Reilly, J.; McCarthy, K.; Mcginty, L.; and Smyth, B. 2004a. Dynamic critiquing. In *Advances in Case-Based Reasoning, 7th European Conference (ECCBR) 2004*, 37–50.

Reilly, J.; McCarthy, K.; McGinty, L.; and Smyth, B. 2004b. Incremental critiquing. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 101–114. Springer.

Reilly, J.; McCarthy, K.; McGinty, L.; and Smyth, B. 2005. Explaining compound critiques. *Artif. Intell. Rev.* 24(2):199–220.

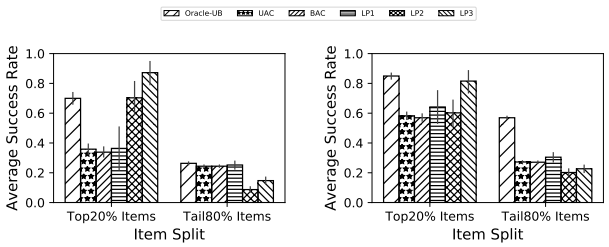
Sedhain, S.; Menon, A.; Sanner, S.; and Xie, L. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on the World Wide Web (WWW-15)*.

Sedhain, S.; Bui, H.; Kawale, J.; Vlassis, N.; Kveton, B.; Menon, A. K.; Bui, T.; and Sanner, S. 2016a. Practical linear models for large-scale one-class collaborative filtering. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 3854–3860. AAAI Press.

Sedhain, S.; Menon, A. K.; Sanner, S.; and Braziunas, D. 2016b. On the effectiveness of linear models for one-class collaborative filtering. In *Thirtieth AAAI Conference on Artificial Intelligence*.

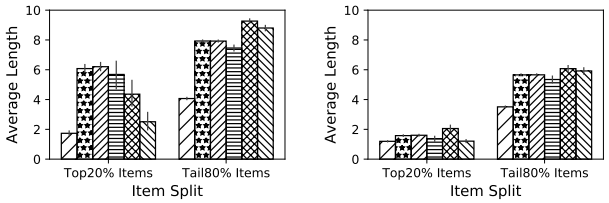
Thompson, C. A.; Goker, M. H.; and Langley, P. 2004. A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research* 21:393–428.

Wu, Y.; DuBois, C.; Zheng, A. X.; and Ester, M. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 153–162. ACM.



(a) CDs&Vinyl (b) Beer

Figure 5: Long tail item analysis average success rate at target rank 5 comparison for different critiquing algorithms on CDs&Vinyl and Beer with 95% confidence interval. All figures share the legend.



(a) CDs&Vinyl (b) Beer

Figure 6: Long tail item analysis average length at target rank 5 comparison for different critiquing algorithms on CDs&Vinyl and Beer with 95% confidence interval. All figures share the legend.

Appendix

New Experiment Results with $\min(-1, -k)$

LK: I will update this part for reference.

Table 6: Percentage of target items that are successfully retrieved within Top1, Top5 and Top10 on Amazon CDs&Vinyl and Beer datasets. We omit the error bars since the confidence interval is in 4th digit.

Model	CDs&Vinyl			Beer		
	Success Rate@1	Success Rate@5	Success Rate@10	Success Rate@1	Success Rate@5	Success Rate@10
Oracle-UB	0.1245	0.2714	0.3577	0.3722	0.5837	0.6712
UAC	0.1202	0.2492	0.3211	0.1185	0.2953	0.3974
BAC	0.1207	0.2486	0.3208	0.117	0.2929	0.3891
LP1	**0.1233**	**0.2563**	**0.3381**	**0.1374**	**0.3263**	**0.4447**
LP2	0.0425	0.1071	0.1508	0.0932	0.2281	0.3173
LP3	0.0618	0.1684	0.229	0.1037	0.2612	0.3481

Table 7: Average length of sessions given target ranks as Top1, Top5 and Top10 on Amazon CDs&Vinyl and Beer datasets with 95% confidence interval.

Model	CDs&Vinyl			Beer		
	Average Length@1	Average Length@5	Average Length@10	Average Length@1	Average Length@5	Average Length@10
Oracle-UB	4.6211± 0.0894	4.005± 0.072	3.6344± 0.0627	4.5044± 0.0647	3.3933± 0.0478	2.9333± 0.0414
UAC	9.0071± 0.0431	7.8641± 0.0613	7.218± 0.0665	6.2321± 0.0597	5.4099± 0.0619	4.8805± 0.062
BAC	9.003± 0.0433	7.8644± 0.0614	7.2163± 0.0666	6.2381± 0.0597	5.4154± 0.0619	4.9051± 0.0623
LP1	**8.481± 0.1402**	**7.4045± 0.2059**	**6.7401± 0.2307**	**5.9063± 0.2199**	**5.1282± 0.2321**	**4.597± 0.2297**
LP2	9.6359± 0.0947	9.1125± 0.154	8.7375± 0.1734	6.4481± 0.2118	5.8262± 0.2144	5.4581± 0.2128
LP3	9.4826± 0.1019	8.6179± 0.1811	8.1414± 0.2028	6.4189± 0.2047	5.6716± 0.2178	5.2823± 0.2178