Natural Language Processing for Recommender Systems

Oren Sar Shalom, Haggai Roitman, and Pigi Kouki

1 Introduction

Recommender systems process all types of signals at their disposal in order to suggest the most relevant items to users. These signals can take diverse forms, from categorical to numerical values, from tabular data to unstructured data. Arguably, the most significant form of content information is textual data, which may hold detailed and invaluable information. Examples include user-generated reviews, which elaborate on experiences of users with items, and textual description on items which can detail an assortment of relevant properties. While the information residing within the textual signals is tremendous, we need to employ advanced techniques in order to extract meaningful insights from it. This is where Natural Language Processing (NLP) comes in as a useful tool to this goal. NLP [18] is a scientific field focusing on automatically processing and analyzing textual data, thus it can be of great help for recommender systems. In this chapter, we break down the various types of inputs that a recommender system can take and identify cases where NLP capabilities can potentially assist each input type. Then, for each such case, we present methods to incorporate the textual input into the recommender system, analyze it and indicate its relative advantages and limitations.

Processing efforts of recommender systems can be coarsely divided into an offline modeling phase and an online recommendation phase. The former aims to model users' preferences and items' traits while the latter considers the current

O. S. Shalom (✉)
Facebook, Tel Aviv-Yafo, Israel

H. Roitman
Buyer Experience Research Department, eBay Research, Netanya, Israel

P. Kouki
Data Science Department, Relational AI, Berkeley, CA, USA

F. Ricci et al. (eds.), *Recommender Systems Handbook*,
https://doi.org/10.1007/978-1-0716-2197-4_12

state and ephemeral needs of the user. The output contains the recommendations themselves, possibly alongside with some additional means to persuade the user. We now list the possible inputs and outputs while focusing on text:

1. Offline input: collaborative filtering algorithms collect past usage patterns in order to model users and items. This data may include *textual reviews*, which detail the multifaceted experiences of users with items.
   Additionally, content-based filtering utilizes side-information on items, where such information may include textual description of the items. This topic is covered in Chap. 7 and we refer the interested reader to that chapter.
2. Online input: *conversational recommenders* allow the users to detail their current needs in free-text.
3. Output: *explanations* can significantly increase the effectiveness of recommenders, and text generation is one of the prominent techniques to achieve this goal.

This chapter is organized as follows. In Sect. 2 we cover the most prominent approaches for review-based recommenders. Section 3 highlights conversational recommenders. In Sect. 4 we detail on methods to generate explanations.

This chapter discusses only scenarios pertaining to *applications* of NLP for recommender systems. We acknowledge there are *techniques* that were originally designed for NLP but can be useful for recommenders as well, however they are out of scope for this chapter. For instance, Transformers [72] were designated to process tokens and afterwards were adapted to recommenders to process user sessions.

Some recommenders allow users to write textual feedback on the *recommendations* rather than on items. That is, the users may specify the relevancy level of the suggested recommendations. However, this is a rare scenario (a notable example is [24]) and, therefore, is out of scope for this chapter.

2 User Generated Reviews

User-item experiences can be complex and multifaceted, where the user may have a different opinion on various aspects of the item. As an example from the movies domain, a user may not like the special effects because they are either too bombastic or too moderate; likewise, the user may have justified opinions on the acting, directing, plot, etc. To better model users and items, recommenders should capture the entire users' impressions of items. Namely, it is important to infer not only to what extent a user likes an item, but also *why* and *which* factors led that impression. User ratings, the most widely used type of explicit feedback, inevitably lose valuable data because they have to summarize that rich experience into a single scalar.

Compared to that, textual reviews are feasibly the most elaborate type of feedback and they allow to fully describe the multifaceted experience of users with items. Concretely, a textual review may unveil any of the following.

1. User preferences (e.g., "I prefer drama movies")
2. Traits of the discussed item ("starred by Tom Hanks")
3. The matching between them ("amazing soundtrack")
4. The context of the interaction ("I watched this movie *with my friends*")

The unstructured form of textual reviews allows them to detail the multifaceted experiences of users with items. But at the same time, it is non-trivial to integrate this input source alongside with traditional collaborative filtering data.

Broadly speaking, as discussed in Chap. 15 there are two major tasks for recommender systems: ranking and rating prediction. In this chapter we review the most prominent approaches to incorporate textual reviews for these two tasks, and explain how the special characteristics of reviews are exploited to improve performance. Furthermore, since each review holds relatively rich information, then few interactions are required to model users and items, which helps to mitigate the cold start problem. Unless otherwise stated, the input to any of the described algorithms is a list of (u, i, r_{ui}, t_{ui}) quartets, each of which indicates an event where user u interacted with item i, assigned it an explicit rating r_{ui} and complemented it a textual review t_{ui}.

2.1 Affinity to Sentiment Analysis

Sentiment analysis is an NLP task aiming to automatically quantify the emotions or opinions expressed within textual data [1]. In its most basic form, it converges to a binary classification problem, where the predicted sentiment is either positive or negative. Occasionally, polarity precision is not sufficient and more fine-grained prediction is required. That is, the sentiment score may span multiple ordinal categories e.g., on a 1–5 Likert scale.

Fine-grained sentiment analysis resembles the rating prediction task of review-based recommenders, and we would like to stress the differences between the two tasks. While both tasks have access to the textual data during training time, they differ at inference time. Sentiment analysis predictions are made with respect to a given text; while a recommender predicts the rating a user would give to an item *before* they have experienced with it. That basically means that the textual review written by the user for the item is *not available* at prediction time. Hence, the rating prediction task is much harder as predictions are made solely based on past behavior, without an accompanying text.

2.2 Traditional Methods

In recent years, with the advent of deep learning, state-of-the-art methods for most NLP tasks apply deep learning (DL) techniques, and review-based recommenders are no exception. However, for completeness, and, to better understand the intuition behind DL approaches, we briefly survey some of the most influential traditional approaches.

Several early works employ topic modeling to represent users and items. McAuley et al. [47] define as a document the set of all reviews written on a certain item and apply LDA [4] on the corpus induced by the entire catalog. The topic distribution of each item is considered as its latent features, while the users' latent features are estimated by optimizing rating prediction with gradient descent. A drawback of this approach stems from the fact that the item vectors are unmindful to the ratings. This makes them sub-optimal and, as a result, the user vectors are sub-optimal as well. This drawback was remedied in a later work [2], which considers the ratings and the reviews simultaneously.

The Ratings Meet Reviews (RMR) algorithm [43] is a probabilistic generative model that combines a topic model with a rating model. It applies topic modeling on item review text in order to find the distribution of each item over the latent topics. A rating r_{ui} given by user u to item i is assumed to be generated by a Gaussian mixture model: user u is associated with a Gaussian per latent topic and the topic distribution of item i serves as the mixture weights. However, this process neglects the expressed sentiments. For example, two items with the exact same topic distributions, but opposite sentiments will have the same predicted ratings for any user.

Diao et al. [20] proposed a probabilistic model based on collaborative filtering and topic modeling. It uncovers the relevant aspects in the given domain, finds the interest distribution of users and content distribution of item and finally infers per-aspect sentiment as expressed in the reviews. A limitation of this algorithm is its inability to incorporate explicit ratings, which omits a valuable signal.

Overall, the aforementioned approaches exhibit two shortcomings:

1. Overlook context. These approaches take a bag-of-words approach, and process individual tokens or n-grams while ignoring their order. In contrast, deep learning approaches are able to process each token with respect to its *context* tokens.
2. Operate solely based on lexical similarity. This is a pitfall because semantically similar reviews may have a low lexical overlap. On the other hand, deep learning approaches are based on distributed representations, which capture the *semantics* of the words.

2.3 Deep Learning: Preliminaries

Per the fundamental limitations of the traditional methods, it comes as no surprise that state-of-the-art review-based recommender are based on deep learning approaches. This subsection presents few definitions and notations required to understand these approaches. These include the definition of Text Processor and some remarks on machine learning techniques.

2.3.1 Text Processor

All notable work in recent years on recommender systems that leverage text (e.g., reviews) rely on a deep learning *Text Processor* unit. Given an input text, this unit represents it by a dense vector which captures the most meaningful aspects with respect to the task at hand.

There are various plausible architectures for the Text Processor, e.g., based on LSTM [25] or Transformers [72]. Most commonly used is the relatively simple Convolutional Neural Network (CNN) Text Processor [32]. This architecture first projects the texts to a latent space by using word embeddings. Then, it applies multiple filters of variable lengths over all sliding windows, each responding to different semantic meanings. Next, a max-pooling operation is performed, to obtain a fixed size vector and to make the algorithm position-invariant. Finally, a fully-connected layer transforms the output to the desired latent space. Jacovi et al. [30] and Kim [32] provide a detailed analysis for the user of convolutional neural networks using text in classification tasks.

We reiterate that different architectures (e.g., LSTM-based) can be used interchangeably.

2.3.2 Machine Learning Annotation

Fully Connected Layer: to ease the notation throughout this chapter, we denote a fully connected layer with output dimensionality of k by $FC^k(\cdot)$. This layer refers to the classic neural network component, in which all k' input neurons connect to all k neurons in the output layer. The input neurons are represented by a matrix X and the layer computes $\sigma(XW+b)$, where $W \in \mathbb{R}^{k' \times k}$ and $b \in \mathbb{R}^k$ are model parameters and σ is a nonlinear activation function, defaulted to ReLU [52] in this chapter.

Optimization: all described algorithms use conventional techniques for improving the speed, stability and generalization of the models. This includes dropout layers [69], regularization and adaptive learning rate optimization algorithms like Adam [33]. Additionally, other techniques which were not applied in the original papers such as batch normalization [29], could be utilized to further improve performance. Since these optimization techniques are not the essence of the models, we omit them from the descriptions of the algorithms. Also, modern models iterate through the training data in mini-batches. To facilitate the notation, we simply describe the process for a single instance.

2.4 Review-Based Recommenders for Rating Prediction

We now proceed to the descriptions of landmarks review-based recommenders. We iteratively explain each of these methods, explain how they work, analyze their strengths but also indicate possible weaknesses (which motivates continuous improvements).

2.4.1 *DeepCoNN*

The seminal algorithm *DeepCoNN* [84] was the first to apply deep learning capabilities in order to incorporate textual reviews for recommenders, and it has inspired a great amount of follow-up work. The basic intuition behind this algorithm is that generated textual reviews encapsulate essential information about both users and items. Therefore, two instances of text processors Γ_U and Γ_I are learned to extract meaningful information on the users and the items, respectively. Specifically, to represent user u, all reviews written by that user are concatenated to form text t_u, which holds all necessary information to model u. Then, the text is fed to Γ_U, which identifies the user preferences as expressed in their textual reviews and generates p_u, the distributed (vectorial) representation of that user. The representation q_i of item i is done similarly by concatenating all reviews written on that item and applying the item text processor:

$$p_u = \Gamma_U(t_u) \qquad p_i = \Gamma_I(t_i)$$

Given the user and item vectors p_u and q_i, most CF algorithms predict their associated rating using a dot product (plus possibly the global mean and the corresponding user and item bias terms). *DeepCoNN* takes a different approach to combine these vectors. It concatenates them as $z = p_u \parallel q_i$ and predicts the rating using a Factorization Machine (FM) [61]. FM can model all the second-order interactions in a given vector in linear time, which may add some expressiveness power to simple inner product. Formally, the predicted rating is given by:

$$\hat{r} = FM(z) = \mu + \sum_{j=1}^{|z|} w_j z_j + \sum_{j=1}^{|z|-1} \sum_{k=j+1}^{|z|} \langle v_j, v_k \rangle \, z_j z_k \qquad (1)$$

where μ is the global computed average rating, w models the strength of each component in z and v_i is a learned vector of some predefined size, corresponding to each component z_i, which models the second order interactions in z. Training is done end-to-end, where the objective is to minimize the absolute prediction error. Evaluation on several benchmark datasets asserted the viability of user-generated reviews to improve rating prediction of future interactions, as it outperformed all previous methods to incorporate reviews.

2.4.2 *TransNets*

If at test time, the target textual review t_{ui} written by user u for item i was available to the model, it could have improved accuracy (see Sect. 2.1). Although textual reviews are not available at test time, they are still available at training time and *TransNets* [8] enhances *DeepCoNN* by harnessing them.

To this end, another subnetwork is trained, dubbed as the Target Network, which aims to reconstruct t_{ui} from p_u and q_i. This empowers the recommender since it allows to model users and items such that their representations can reconstruct the target reviews, simulating a situation where the target review is available at test time.

Since the semantics of the target review is of key importance rather than the actual raw text, at training time this subnetwork *transforms* the target review using another instance of a Text Processor and its representation z_T is to be reconstructed. First, the representation of the target network z_T has to capture the meaning of the text reviews. This is achieved by generating representations with predictive capabilities of the ratings. Formally:

$$z_T = \Gamma_T(t_{ui})$$
$$r_T = FM_T(z_T)$$

In view of the fact that Z_T should guide p_u and q_i and not vice versa, training is carefully done. Per each training instance, the parameters of the model are divided into 3 groups, and they are updated sequentially:

1. The parameters of the target network, which comprises Γ_T and FM_T, are updated by minimizing the rating prediction error: $| r_T - r |$.
2. The parameters of the target network remain fixed and Γ_U and Γ_I are learned by minimizing the loss: $(z - z_T)^2$.
3. The parameters of FM are updated by minimizing $| \hat{r} - r |$.

We emphasize that these sequential sub-steps occur every instance, rather than training to convergence each subnetwork.

The Target Network functions only to improve the training process and does not participate at inference time, as the target review is not available at this point. Hence, predictions are computed exactly as in *DeepCoNN*. *TransNets* adds another optimization technique: it passes z through stacked fully-connected layers, to allow modeling of more complex interactions between the user and the item.

2.4.3 Extended TransNets

Thus far the discussed models ignore the *identities* of the users and the items and representations are based exclusively on the review texts. However, the identities may supply usable information. After all, pure collaborative filtering methods rely solely on this signal. For this reason, the Extended TransNets (*TransNet-Ext*) model introduces embedding matrices for users and items, Ω_U and Ω_I, respectively. The latent representation of a user-item pair is supplemented with the concatenation of the user and item embeddings:

$$\omega_u = \Omega_U(u) \qquad \omega_i = \Omega_I(i)$$
$$z_{ext} = \omega_u \parallel \omega_i \parallel z$$

Now z_{ext} is used to predict the rating as in Eq. 1. The embedding matrices do not stand on their own, but rather they are complementary to the textual representations. That is, vector p_u represents the preferences of user u as captured by the text processor. However, this vector does not perfectly reflect the true preferences of the user, whether because of limitation of the text processor or simply because the textual reviews do not contain all required information to accurately analyze user u. The role of vector ω_u is to supplement the preferences induced by the texts, very much like the deviation matrix presented in [26]. A similar analogy applies also for the item embedding matrix Ω_I.

2.5 *State-of-the-Art of Review-Based Recommendations*

2.5.1 Motivation

A common drawback to the aforementioned work and more recent work CF [9, 11, 12, 64, 73] is the required effort to process a single review during training. The feed-forward operation of a review given by user u to item i includes all reviews written by u and the reviews written on i. Let w be the average number of words per review and l_I and l_U be the average number of reviews associated with items and users, respectively. Hence the running time complexity to process a single review is $O(w \cdot (l_I + l_U))$.

If a typical user writes tens of reviews, an item could appear in thousands of reviews and a review comprises more than one hundred tokens, then processing a *single* review requires hundreds of thousands operations. This scalability issue might pose an insurmountable problem for real-life recommenders, as training on large datasets is infeasible and therefore a solution to reduce training time is required.

As we detail in this section, the Matching Distribution by Reviews (*MDR*) [65] takes a different perspective on reviews, which both eliminates the running time issue and improves accuracy.

2.5.2 Intuition

Previous algorithms assume that reviews describe item characteristics and user preferences and therefore apply a dedicated text processor for each of these two entity types. Comparing to that, *MDR* claims that reviews concentrate on the *matching* between them, i.e., explain the extent at which the user liked various aspects in the item. This observation is very beneficial since in some cases it is possible to infer the matching between the user and the item regarding a specific trait, even though the individual user preferences or item traits are not disclosed in the review. Consider for example the following review snippet from the movie domain: “I loved the soundtrack”. It is clear the soundtrack of the movie fits the preferences of the user, although no information is given on the soundtrack itself, like type or duration.

In each domain there could be an enormous amount of factors relevant to generating recommendations. In our running example from the movie domain, such factors include genre, shooting location, plot complexity, animation style, etc. A single review is unlikely to supply adequate information on all relevant factors. For example, by reading a review, it may be apparent to what extent a user likes the genre but less clear whether they enjoyed the animation style. *MDR* also models the inevitable uncertainty in processing individual reviews. This is done by predicting a *distribution* rather than just a point estimate, which does not allow to encode uncertainty.

As motivated above, both textual reviews and explicit ratings refer to the matching level between the user and the item, but they differ in the level of thoroughness. A textual review is an elaborate form of feedback, as it reasons and details the factors that made the user form their judgment on the interacted item. Therefore, it is only natural to view the reviews as augmented *labels* to train the collaborative filtering algorithm, in addition to explicit ratings.

2.5.3 Algorithm Overview

The algorithm consists of two phases: the first, learns how to model the textual reviews and, the second, utilizes the modeled reviews as augmented labels. These phases are learned sequentially, i.e., after the parameters of the first phase are learned, they are held fixed and then the second phase's parameters are learned.

This scheme alleviates the running time issues raised in Sect. 2.5.1, as the whole algorithm has linear time complexity. The first phase is linear in the number of tokens and the second phase is linear in the number of reviews.

In this chapter we detail the steps of the algorithm and explain the intuition behind them. We do not, however, give the theoretical justifications for them. The curious reader can see the proofs in the original paper [65].

2.5.4 Phase I: Distribution of Matching Vectors

As motivated above, a textual review can bring to light the matching between a user and an item across the latent features. The purpose of this phase is to automatically infer this information. To avoid clutter, subscripts u and i are omitted when their existence is clear from the context. To this end, *MDR* passes each review t through a text processor and obtains a matching vector m. Each component f in the matching vector reflects the matching between the user and the item with regards to the f-th latent feature. Therefore the sum of the components in the matching vector predicts the overall satisfaction of the user with the item, which is also reflected by the explicit rating given by the user to the item. That is $\hat{r} = \sum_f m^f$, where m^f stands for the f-th component in vector m.

The problem with this approach is it assumes all latent features can be adequately estimated by any review. However, a specific review t may be ambiguous or simply does not cover all latent features. To account for this, *MDR* finds the *distribution* of the matching vectors. The distribution of choice is multivariate normal distribution $\mathcal{N}(\mu(t), \Sigma(t))$ with a diagonal covariance matrix, which can be parameterized by two vectors: mean $\mu(t)$ and variance $\Sigma(t)$. When the model has high confidence in the predicted matching value of a specfic factor, then the associated variance of this factor would be low, and vice versa. Given text t, the distribution of matching vectors is simply inferred by computing e, the output of a text processor on the review. Then two different fully connected layers are applied, to yield the parameters of the mean and variance $\mu(t)$ and $\Sigma(t)$.

$$e = \Gamma(t) \tag{2}$$

$$\mu(t) = FC(e) \qquad\qquad \Sigma(t) = FC(e)$$

In stochastic gradient descent, a single training example represents the whole distribution of the training data. By the same token, a single sampled vector $z \sim \mathcal{N}(\mu(t), \Sigma(t))$ can represent the whole distribution of matching vectors. Sampling is a non-differentiable operation, thereby it would eliminate all gradients and hamper the process of back-propagation. Therefore, the reparameterization trick [34] is exerted. This means that the sampling operation is done on a newly introduced input layer, that samples a vector $\epsilon \sim \mathcal{N}(0, I)$. Then, the appropriate vector z is mapped by scaling and shifting $\mathcal{N}(0, I)$ to align with $\mathcal{N}(\mu(t), \Sigma(t))$. Finally, the matching vector m is obtained by feeding z to a fully connected layer.

$$\epsilon \sim \mathcal{N}(0, I) \qquad\qquad z = \mu(t) + \Sigma^{\frac{1}{2}}(t) \odot \epsilon$$

$$m = FC(z) \tag{3}$$

The predicted rating is the sum of the elements in the matching vector where conventionally, also bias terms and global mean are added: $\hat{r}_{ui} = \sum_f m_{ui}^f + b_i + b_u + \mu$. The loss function asks to minimize mean squared error.

2.5.5 Phase II: Collaborative Filtering with Augmented Labels

This phase applies a rating prediction algorithm, while benefiting from augmented labels that were derived in the previous phase.

Latent factor models represent users and items by some vectors of fixed size. Each component f in this space refers to a latent feature and the components in a user vector measure to what extent the user *prefers* these latent features.

Comparably, components in an item vector measure the extent to which the item *holds* these latent features.

While any CF algorithm could be used in this phase, the authors of *MDR* chose the simple *SVD* [58] as the underlying engine. This algorithm asks to minimize the squared error: $(r_{ui} - \hat{r}_{ui})^2$, where the predicted rating is computed as: $\hat{r}_{ui} = p_u \cdot q_i + b_u + b_i + \mu$. The inner product between user vector p_u and item vector q_i indicates the affinity between them. Inner product equals to the sum over the components in the element-wise product: $p_u \cdot q_i = \sum_f (p_u \odot q_i)^f$. As such, the element-wise product indicates the matching between the user and the item over the latent features. We dub the vector $p_u \odot q_i$ as the *collaborative matching vector*.

Using phase I of *MDR*, each review t_{ui} is represented by a distribution of matching vector. The mean of this distribution, dubbed as the *textual matching vector* represents the matching between the user and the item over the latent features and therefore can be utilized as an augmented label for the collaborative filtering algorithm. This is done by minimizing the distance between the collaborative matching vectors and the textual matching vectors. All in all, the loss function is defined as:

$$\mathcal{L}_{ui} = (\hat{r}_{ui} - r_{ui})^2 + \alpha \cdot \|p_u \odot q_i - m_{ui}\|_2^2 \tag{4}$$

Where hyperparameter α controls the relative importance of terms in the loss function. This loss function contains two labels: explicit rating and textual matching vector. While the former gives a coarse direction to the optimization process, the latter provides a direction per each of the f factors, which can significantly improve performance. Indeed evaluation on several benchmark dataset proves the superiority of this approach over all previous baselines.

2.6 Empirical Evaluation

This section compares the performance of the strongest review-based models on several benchmark datasets. The first dataset is `Yelp17`, which contains restaurant reviews, introduced in the Yelp Challenge.[1] Each of the other three datasets is a different domain taken from the latest release of Amazon reviews[2] [48], which contains product reviews from Amazon website. These datasets vary in size and sparsity, where complete statistics can be found at [65].

Table 1 summarizes the empirical results. It shows the viability of incorporating textual reviews, as all review-based algorithms improve SVD, which does not incorporate reviews. Furthermore, *MDR* outperforms all other baselines, in addition to improved running time complexity.

[1] https://www.yelp.com/dataset-challenge.

[2] http://jmcauley.ucsd.edu/data/amazon.

Table 1 MSE comparison with baselines. Best results are indicated in bold

	SVD	*DeepCoNN*	TransNetTransNet	TransNetTransNet-Ext	TransNetMDR	Improvement
Yelp	1.8661	1.7045	1.6387	1.5913	**1.4257**	10.4%
A-Electronics	1.8898	2.0774	1.8380	1.7781	**1.5329**	13.8%
A-Clothes	1.5212	1.7044	1.4487	1.4780	**1.2837**	13.1%
A-Movies	1.4324	1.5276	1.3599	1.2691	**1.1782**	7.2%

2.7 Review-Based Recommenders for Ranking

In this subsection we explain how to incorporate user generated reviews for the ranking problem. Naturally, to achieve optimal performance on this problem, the training procedure should be optimized directly for *ranking*. However, all hitherto mentioned work optimize *rating prediction* while neglecting the ranking problem.

As a side note, we would like to stress that any existing method for the rating prediction problem can be adjusted for ranking. This can be simply done by adding a ranking loss (e.g., the pairwise BPR loss [62]) as an additional term of the loss function, or by more advanced techniques like [26, 67]. However, this direction has not been significantly investigated in the context of review-based recommenders and is therefore omitted from this chapter.

Recently Chuang et al. [15] suggested the Text-aware Preference Ranking (TPR). This method simultaneously optimizes two main objectives: user-item ranking and item-word ranking. The former aims to rank items in the history of the user higher than missing items. The latter seeks to rank words that appear in an item's review higher than other words. However, this method takes a bag-of-words approach, which may not capture the semantics of the entire review, and is not clear how more elaborate methods can be integrated instead. Furthermore, TPR cannot incorporate explicit ratings. Since usually textual reviews are accompanied with explicit ratings, leaving this signal out might lead to sub-optimal performance.

We now turn to the description of the main algorithm of this subsection. Zhang et al. [81] presented a Joint Representation Learning (JRL) framework to incorporate heterogeneous input signals. In the context of this chapter, the considered input sources are reviews (denoted as V_1) and ratings (V_2).

The algorithm is learned end-to-end and comprises the following three steps:

1. Create input-source specific representations for users and items.
2. Integrate representations to have a single representation.
3. Apply ranking optimization on the integrated representations.

We now elaborate on each of the aforementioned steps.

2.7.1 Input Source Modeling

In each input source k (either reviews or ratings), any user u and item i is represented by a dedicated vector p_u^k and q_i^k, respectively. These vectors aim to optimize loss function $\mathcal{L}_k$.

Modeling of Textual Reviews

Each review t_{ui} is represented in an unsupervised manner by vector $\boldsymbol{t}_{ui}$ (denoted in bold).

The authors of JRL adopted the PV-DBOW model [41]. This is a generative model, where review's representation should maximize the likelihood of its comprising words. To this end, each word w in the vocabulary is represented by a learned vector $\boldsymbol{w}$. Then, the log probability of having word w in a review is approximated by the negative sampling (NEG) procedure: $\log P(w|t_{ui}) = \log\sigma(\boldsymbol{w}^T \boldsymbol{t}_{ui}) + t \cdot \mathbb{E}_{w_N \sim P_V}[\log\sigma(-\boldsymbol{w}_N^T \boldsymbol{t}_{ui})]$, where t is the number of negative samples and P_V is the noise distribution. In this paper t was set to 5 and P_V is the unigram distribution raised to the 3/4rd power. Hence, the objective for a given review is:

$$\mathcal{L}_1(u,i) = \sum_{w \in t_{ui}} f_{w,t_{ui}} \log P(w|t_{ui}) \tag{5}$$

where $f_{w,t_{ui}}$ counts the number of occurrences of w in the review.

Then user u is represented by an average of their reviews: $p_u^1 = \frac{1}{|R_u^1|} \sum_{i \in R_u^1} \boldsymbol{t}_{ui}$, where R_u^1 is the set of items reviewed by user u. In a similar fashion item vectors are computed.

Modeling of Numerical Ratings

Modeling users and items according their explicit ratings is rather straightforward. The predicted rating given by user u to item i is calculated by a two-layer fully connected network: $\hat{r}_{ui} = FC^1(FC^t(p_u^2 \odot q_i^2))$, with $\odot$ denoting the element-wise product, ELU [17] as the activation function and t is a hyperparameter that determines the dimensionality of the hidden layer. The objective in this input source is to minimize the squared prediction error:

$$\mathcal{L}_2(u,i) = (\hat{r}_{ui} - r_{ui})^2 \tag{6}$$

2.7.2 Integrated Representation

The integrated representation of each user and item is obtained by a simple concatenation of the representations across the input sources: $p_u = p_u^1 \parallel p_u^2$ and $q_i = q_i^1 \parallel q_i^2$. We address to a limitation of this approach in Sect. 2.7.4.

2.7.3 Ranking Optimization

Until this point, the embeddings are not optimized for ranking, but to model textual data ($\mathcal{L}_1$) and to predict ratings ($\mathcal{L}_2$). Therefore, JRL incorporates the widely adopted BPR loss [62] as the pairwise learning-to-rank method. For each observed user-item interaction $(u, i^+) \in R$, a random item i^- such that $(u, i^-) \notin R$ is sampled as a negative sample. The objective is to distinguish between positive and negative pairs:

$$\mathcal{L}_{RANK}(u, i) = \log \sigma(p_u \cdot q_{i^+} - p_u \cdot q_{i^-}) \tag{7}$$

For more details on the BPR loss, we refer the reader to Chap. 3. Personalized recommendations are generated by ordering all items according to their predicted scores. Combining the input-source dependant loss functions in Eqs. 5 and 6 together with the ranking objective presented in Eq. 7, gives us the objective function of JRL:

$$\mathcal{L} = \sum_{(u,i)\in R} \mathcal{L}_{RANK}(u, i) + \mathcal{L}_1(u, i) + \mathcal{L}_2(u, i) \tag{8}$$

2.7.4 In-Depth Analysis

Decoupling Input Sources

Albeit not mentioned in the original paper, while the algorithm assumes the existence of several modalities, they do not need to be aligned. For instance, a user may leave explicit ratings for some items, to write textual reviews on other items and the final user representation will consider all of these interactions. This is in contrast to all previously described algorithms in this chapter, that require each interaction to comprise both explicit ratings and textual reviews.

Fusion of Representations

The personalized score is computed as: $s = p_u \cdot q_i = (p_u^1 \parallel p_u^2) \cdot (q_i^1 \parallel q_i^2) = p_u^1 \cdot q_i^1 + p_u^2 \cdot q_i^2$. Hence, it is the sum of the scores given by two pairs of representations, and at prediction time there is no interaction between these two representations. Perhaps the model could be improved if the integrated representation was done by feeding the input-source representations to fully connected layers, which allow to capture non-linear correlations between the representations.

Generalized Modeling of Textual Reviews

Modeling of textual reviews is done by PV-DBOW, which does not obtain state-of-the-art results on document representation. Fortunately, JRL is agnostic to the choice of text modeling, and any other unsupervised could be seamlessly used. For instance, reviews can be modeled using AutoEncoders [42] or contextual embedding models (e.g., GPT-3 [6]).

2.7.5 Empirical Evaluation

We now compare relevant baselines to asses the vitality of JRL. All experiments were conducted on various domains of the Amazon review dataset presented in Sect. 2.6. For each dataset, 70% of the interactions of each user were randomly selected for training and the remaining were kept for test. All users are presented with $K = 10$ recommendations and a recommended item is marked as "correct" if it resides within the test items of that user. We report two prevalent top-K evaluation measures:

- HT: Hit-ratio, which is the percentage of users with at least a single correct recommendations.
- NDCG: considers also the position of the correct recommendations.

Additional details on these evaluation measures can be found in Chap. 15. The first baseline is BPR [62], a seminal ranking algorithm that does not leverage reviews. Another baseline, which was presented in Sect. 2.4.1, is *DeepCoNN* [84]. This baseline does incorporate reviews, but is optimized for the rating prediction task.

Table 2 shows the performance of each algorithm. First, it gives another evidence for the contribution of review to recommender systems, as *DeepCoNN*

Table 2 Ranking comparison. Best results are indicated in bold

	BPR		*DeepCoNN*		JRL	
Measure	HT	NDCG	HT	NDCG	HT	NDCG
A-Movies	4.421%	1.267%	10.522%	3.800%	**13.245%**	**4.334%**
A-CDs	8.554%	2.009%	13.857%	4.218%	**16.774%**	**5.378%**
A-Clothes	1.767%	0.601%	3.286%	1.310%	**4.634%**	**1.735%**
A-Cellular	5.273%	1.998%	9.913%	3.636%	**10.940%**	**4.364%**
A-Beauty	8.241%	2.753%	9.807%	3.359%	**12.776%**	**4.396%**

consistently outperforms BPR across all datasets. Most notably is the benefit of having a designated algorithm for ranking, as JRL gains a substantial improvement comparing to other algorithms.

2.8 Discussion and Future Outlook

This section showed the tremendous value of user-generated reviews in modeling users and items for both rating prediction and ranking. Over the past years, the recommender systems community has unceasingly improved the means to incorporate textual reviews, which results in increasing performance and reduction in running time complexity. Currently, reviews are an integral part of the modeling process of state-of-the-art production systems and we anticipate a plethora of work in this direction, towards more accurate recommenders.

We should note that most existing review-based algorithms assume the availability of explicit ratings. At first, it might seem like a hard assumption that limits applicability. However, platforms that collect textual user reviews usually also collect explicit numeric ratings, and hence we are not restricted by the additional requirement of having explicit ratings.

So far, existing work in this field relies on datasets such that *all* interactions are associated with reviews. However, this leads to a *selection bias* because only a subset of the users tend to write reviews, while the rest of the population is ignored. These users may differ in their tastes and usage patterns from the general population and therefore the system may suffer from poor performance on the general population. Furthermore, a user may be more likely to devote time for writing a review if the experience with the item was extremely positive or negative. This is another form of a potential selection bias, where extreme cases are overemphasized. Designers of real-life recommenders should consider these aspects and integrate also data that represents the general population.

Looking to the future, with the great predictive power of reviews, comes a potential threat by malevolent stakeholders. As textual reviews add invaluable information on top of explicit ratings, an attack that generates reviews might be considerably striking. Hence, there is an inherent future need for designing systems that can effectively detect adversarial reviews.

Another line of research could be to utilize reviews in order to break down the items and to identify fine-grained recommendable units. Bauman et al. [3] for example suggested a bag-of-words approach to identify aspects of consumption. Then for example, beyond recommending a restaurant the system could suggest specific dishes. This is an interesting problem that has not been widely studied yet, with a variety of potential applications (e.g., suggest a hotel as well as a room with a specific view or to recommend a specific chapter in a handbook).

3 Conversational Preference Elicitation

The main purpose of the preference elicitation reference elicitation process is to allow end-users to reveal their preferences by interacting with the recommender system. Such interaction may be implemented through a conversational interface using natural language.

While the preference elicitation task may be viewed as a kind of *intent detection* task, there are two fundamental differences between the two. First, intent detection may be performed also in ad-hoc systems that have no interaction with the user, e.g., predicting query intent [5]. Second, in the context of conversational systems, intent detection is usually treated as a classification task, which is bounded by a predefined set of intents (classes) [44]. This in comparison to the preference elicitation task which may require to reason over a large set of possible user preferences.

Overall, we describe three main types of preference elicitation approaches that utilize NLP methods: *critiquing*, *facet-based* and *question-based*. We start by shortly discussing conversational recommender systems as the environment in which such preference elicitation process is implemented.

3.1 Conversational Recommender Systems

Conversation recommender systems (CRS) have become quite common nowadays and are implemented by natural language interfaces (e.g., chat-bots) or speech (e.g., intelligent assistants). Existing conversational systems can be roughly classified as *chit-chat*, *informational* or *task-oriented*. In a sense, a conversation recommender system is both informational and task-oriented. For example, a CRS may be utilized for recommending news to a user, hence satisfying some information need. On the other hand, a CRS may be utilized to fulfill some user goal such as reserving a restaurant, visiting a location or purchasing some item.

Utilizing a CRS is specially effective whenever user preferences are either ill-defined or unclear (e.g., in a cold-start scenario). In a conversational setting, users may express their preferences towards items using a natural language interface. At the same time, the recommender system can further utilize the same natural language interface to interact with its end-users to clarify their needs and provide better personalization.

Different from "traditional" recommender systems, which are usually passive in the sense that they operate in one-shot interaction paradigm, a CRS allows an *interactive*, *mixed-initiative*, dialogue with its end-users. This in turn, allows for varying the recommendation until enough evidence on user's preferences is gathered. Alternatively, a CRS may adapt its previous recommendations to the change in user's preferences as reflected in the ongoing conversation.

A typical CRS has two main components: *Preference-Elicitation* and *Recommendation*. The role of the preference elicitation component is to gather enough information about user preferences for maximizing the predicted user utility. The recommendation component is responsible to provide recommendations based on preference information that was actively curated during the conversation with the user. This component may be also tightly integrated with the preference elicitation component, driving the latter's decisions on which user preferences should be clarified (e.g., present options for item filtering, ask for feedback, etc).

In the rest of this section, we mainly focus on the preference elicitation process using natural language understanding and the representation of derived user preferences for recommendation during a conversational recommendation setting. For broader details on the general scope of conversational recommender systems, the interested reader is referred to recent surveys [31, 59].

3.2 Critiquing

In the critiquing setting, the user is allowed to define constraints on items immediately after those were recommended. For example, in a restaurant reservation setting, the user may refine recommended restaurants by their location, price or rating. Using a critique-based conversational recommendation strategy, users incrementally define their preferences towards recommendable items. In a sense, critiquing allows users to modify their learned "static" preferences and adjust the recommender system to their current "dynamic" tastes [77].

A CRS usually presents to the user several options for refinement based on the properties of the pool of recommendable items (e.g., size, color, price, etc). A simple approach to implement a critiquing strategy is to show user options for selection or fill a manual form. User input is usually assumed to define her critiques to be treated as negative feedback on recommended items.

While it is more intuitive for users to interact with CRS in a natural language, such communication form has been less common in earlier critiquing-based recommender systems [10]. Natural language-based interaction is more complex to model, as the set of specified user critiques may be still subjective, ambiguous or unbounded. Hence, most previous works have been focused on system-suggested (predefined and bounded) critiques or user-initiated critiques [10].

Critiques may be automatically derived from past user interactions [63]. Such *experience-based* methods are implemented by analyzing the properties of items that were successfully recommended in previous similar conversational sessions. Yet, most critique methods strongly rely on the availability of a item metadata (e.g., catalog) or assume a fixed set of critiques.

An alternative and more flexible way to automatically obtain potential critiques is to curate them from textual sources associated with recommendable items. A popular approach is to extract *keyphrases* from item reviews [46, 77]. Keyphrases may be extracted using statistical language-modeling techniques (e.g., identifying terms that are salient in the text), using random-walk methods, etc. For a review on state-of-the-art keyphrase extraction methods, the reader is referred to [49].

We next describe two recent deep-learning methods that exploit keyphrases extracted from item reviews for implementing a critiquing-based conversational preference elicitation process. Using deep-learning techniques allow to better represent both user preferences and feedback (user critiques) in the same latent embedded space, and as a result, improve recommendation accuracy.

3.2.1 CE-NCF

As a first method, we describe the *Critiquable and Explainable Neural Collaborative Filtering* (CE-NCF) method [77]. This method has two versions, *deterministic* and *variational*. Here we only describe the deterministic version.

A key assumption which is now made is that, both the observable user i and item j (binary) ratings r_{ij} and (binary) explanation (keyphrases) vector s_{ij} are generated from the same latent representation z_{ij} which is jointly encoded from the latent user u_i and item v_j representations. Initial user and item embeddings can be obtained by any basic method (e.g., [77] used a randomized SVD method that was fined-tuned during end-to-end training). The above assumption is formulated into a deep-learning framework by first encoding each user-item pair into an initial latent representation $\hat{z}_{ij} = f_e(u_i, v_j)$.

Critiquing augments the latent representation which modifies item ratings to better suite user's current preferences. This is achieved in several steps, as follows. First, a prediction function $\hat{s}_{ij} = f_s(\hat{z}_{ij})$ is applied to map the latent representation into explanation predictions for each recommended item to a particular user. The user then takes a critique action which indicates explanations she disagrees with, "zeroing out" the corresponding keyphrases in $\hat{s}_{ij}$. Next, the inverse function $\tilde{z}_{ij} = f_s^{-1}(\tilde{s}_{ij})$ is applied to project back the critiqued explanation to the latent representation. Finally, the model updates both item ratings $\tilde{r}_{ij} = f_r(\tilde{z}_{ij})$ and explanations $\tilde{s}_{ij} = f_s(\tilde{z}_{ij})$. In addition, to more flexibly control the effect of user critiques, whenever a user makes a critiquing action, the latent representation is updated according to the following linear combination: $\tilde{z}_{ij} = \rho\hat{z}_{ij} + (1 - \rho)\tilde{z}_{ij}$, where $\rho \in [0, 1]$ is a hyperparameter.

To train the model end-to-end (together with item recommendation), the following objective is minimized:

$$\begin{aligned}\min \mathcal{L} = \min \sum_{ij} \mathcal{L}_0(r_{ij}, f_r \circ f_e(u_i, v_j)) \\ +\lambda_1 \sum_{ij} \mathcal{L}_1(s_{ij}, f_s \circ f_e(u_i, v_j)) \\ +\lambda_2 \sum_{ij} \mathcal{L}_2(f_e(u_i, v_j), \tilde{f}_s^{-1} \circ f_s \circ f_e(u_i, v_j)) \\ +\lambda_3 \|\theta\|_2^2,\end{aligned}$$

where the three loss functions $\mathcal{L}_0$,$\mathcal{L}_1$ and $\mathcal{L}_2$ are further taken as Mean Squared Error (MSE) and λ_1, λ_2 and λ_3 are corresponding hyperparameters.

3.2.2 Latent Linear Conversational Critiquing

In a sense, each single critiquing step can be viewed as a series of functional transformations that produce a modified prediction $\tilde{r_i} = f_m(r_i, \tilde{s_i})$ of item preferences for user i given critiqued keyphrases s_i [46]. Having multiple critiquing steps, a user is iteratively provided with the item recommendations $\tilde{r}_i^t$ and, based on those, the user makes a critique action c_i which updates the representation of critiqued keyphrases $\tilde{s}_i^t$ through a cumulative critiquing function: $\tilde{s}_i^t = \psi(s_i, \tilde{s}_i^{t-1}, c_i)$. Alternatively, the user may accept the item recommendation and the current conversation ends.

To derive $\tilde{r_i}$, both user preferences r_i and the critiqued keyphrases are co-embedded to derive a uniform representation $\hat{z}_i$, as follows. First, user ratings r_i are represented by a projected linear embedding $z_i = r_i V$. Next, given user's critique action c_i, the cumulative critiquing representation is derived at step t as: $\tilde{s}_i^t = \tilde{s}_i^{t-1} - \max(s_i, 1) \odot c_i^t$ (with $\tilde{s}_i^0 = 0$) and its latent representation is obtained: $\tilde{Z}_i^t = diag(\tilde{s}_i^t)W^T + B$. Each row $\tilde{z}_i^k$ of the matrix $\tilde{Z}_i$ captures the latent representation of the kth critiqued keyphrase, and each row of B has an identical bias term b. The unified representation $\hat{z}_i$ is then obtained by merging z_i with $\tilde{Z}_i$ as follows:

$$\hat{z}_i = \phi_\lambda(z_i, \tilde{Z}_i) = \lambda_0 z_i + \lambda_1 \tilde{z}_i^1 + \ldots + \lambda_{|K|}\tilde{z}_i^{|K|}.$$

Given $\hat{z}_i$, the updated item ratings are simply given by: $\hat{r}_{ij} = \langle \hat{z}_i, w_j \rangle$ (where $\langle \cdot, \cdot \rangle$ denotes inner product and w_j is the latent representation of item j).

λ weights can be manually set (e.g., having uniform weight assuming that all critiques have the same importance). Yet a better alternative is to learn λ using Linear-Programming (LP) optimization [46].

3.3 Facets-Based Preference Elicitation

In a facet-based preference elicitation process, the conversation context defined by user utterances is used to predict which item facets represent the user's preferences. At each step of the conversation, the CRS may decide to either recommend items to the user based on such derived facets or try to clarify user preferences (and hence better learn the item facets that are most relevant).

The facet-based approach allows a more incremental preference elicitation towards targeted item recommendation. Yet, the facet-based approach still requires some memorization effort by users, as users may not be familiar in advanced with all item facets [59] (e.g., what is the meaning of some unit).

In the context of preference elicitation, there are two main challenges that need to be addressed. The first is *mapping user utterances into facets* and the second is *generating clarification questions based on such facets* to be presented to the user whenever more feedback is required.

Mapping user utterances to facets requires to "identify" mentions in the utterance text related to facet types and/or values. As an example, an utterance such as , "*I need a restaurant **near my home***" identifies a location facet with a range of possible values anchored by the user's own location. As another example, a utterance such as "*I would prefer **drama** to **comedy***" defines two facets with a preference order between the user's preferred movie genres.

Facet extraction from query-related text (e.g., search results) has been previously studied in the context of interactive information retrieval [37, 38]. A common facet curation approach is to first apply textual clustering (e.g., K-Means or Hierarchical) [55, 79]. Each cluster then represents a facet and a clustering labeling technique may be applied to extract a short textual description of the facet [7]. Alternative facet extraction techniques include semantic class extraction [57, 66] (using distributional similarity or pattern mining), topic models [74, 76] mention detection [22], named entity recognition [40] and entity linking [36] .

We next shortly describe a more recent approach that is better tailored for CRS. Using deep-learning methods, a give user utterance may be mapped into a set of potential facet type and values [71], as follows. Let (f, v) denote a specific facet-value pair, e.g., (*color, red*), (*size, small*), etc. Given a user utterance at time step t, e_t, an n-gram vector z_t is first extracted from e_t's text. Next, the sequence of n-grams up to the current time is encoded using an LSTM network into a vector $h_t = LSTM(z_1, z_2, \ldots, z_t)$. To predict facet values probabilities of a given facet f_i, a softmax activation layer is then applied.

The set of facet-values probabilities are then used for representing the dialogue state s_t. This state is used for belief tracking to decide whether to recommend items based on existing facets or further clarify user preferences by asking questions about a specific facet [71].

The decision whether to recommend an item or clarify user preference towards any of the facets may be implemented using a reinforcement learning approach [71]. To this end, at each step t of the conversation, the CRS agent has $l + 1$ actions it can take. The first l actions capture the decision of the agent on whether to clarify the user's preference regarding the values of one the l possible facets. For example, a question such as "*What **color** do you prefer*?" can be asked given that the color facet has the highest belief in the model.

Alternatively, the CRS agent may decide to provide a recommendation based on existing facet-based belief s_t. In [71], such recommendation is implemented using Factorization Machines [61] considering s_t as an additional feature-set. The decision policy of the CRS agent is further learned using the REINFORCE [75] algorithm.

3.4 *Question-Based Preference Elicitation*

In a question-based preference elicitation setting, the CRS aims to unveil user preferences through a series of one or more clarification questions. In a sense, such an approach is similar to the facet-based approach, with two additional options. First, the clarification questions can be topical rather than just asking about existing item facets (properties) [13]. Second, compared to the facet-based approach which asks questions about single facets at a time in an absolute manner, using a relative approach may further allow to identify more clear user preference patterns [14].

We next discuss several recent question-based preference elicitation methods.

3.4.1 "System Ask, User Respond" (SAUR)

In the "System Ask, User Respond" (SAUR) setting [82], each conversation is assumed to be initiated by some user information need (e.g., "*Can you find me a **mobile phone** on Amazon?*") and then the CRS agent may either provide a recommendation or start with a series of clarification questions about aspects that are relevant to the user information need (e.g., "*What **operating system** do you prefer?*", "*Do you have requirements on **storage capacity**?*", etc.). Each conversation used for training the model is assumed to end with a successful item recommendation (e.g., judged by analyzing the user's feedback).

Formally, given a conversation session, initiated with user query related to product category c and succeeded with k questions that have been asked so far by the agent (p_l) and answered by the user (q_l): $Q = p_1, q_1, p_2, q_2, \ldots, p_k, q_k$, the task is to predict the next question to ask p_{k+1}. Here, each question is assumed to be about a single product aspect (e.g., operating system) curated from item reviews and each answer has a value specified by the user to that aspect.

While the SAUR model addresses both item recommendation and question generation, we next focus only on the latter sub-task and the interested reader is referred to [82] for the full details.

A given conversation context Q is represented at step t by first concatenating all the text until that step and encoding it with a Gated Recurrent Unit [16] (GRU). Let c_k denote the encoding at current step k. Each item v_j is further represented by a textual summary (denoted s^r_j) obtained by concatenating its own description with review text associated with it. The representation is then obtained using a GRU with an attention mechanism that allows to make the representation of that item sensitive to a specific conversation context Q. Two memory units are further utilized. The first $m^1_j = GRU(s^r_j, c_k)$ memorizes the relevance of item v_j to Q and used for item recommendation. The second $m^2_j = GRU(s^r_j, m^1_j)$ serves as question memory for item v_j.

The question sub-task now aims to train a model which will maximize the likelihood of the next aspect in the conversation p_{k+1}. Such a likelihood is estimated by first concatenating its own representation with c_k and the average memory $\bar{m}^2$

over all items and then applying a two-layer feed forward neural network followed by a softmax layer. Since applying the softmax may be computationally costly due to the large number of aspects to consider, an alternative is to apply a sigmoid with negative sampling proportional to aspect popularity [82].

3.4.2 Topic-Based Questions

Most conversational recommender systems ask questions related to properties of end-recommended items. Yet, such an approach may not scale well when the item pool is too large and constantly updated [13] (e.g., videos on YouTube).

An alternative, which might scale better, is to ask users question on topics so user feedback can be propagated among items sharing the same topic [13]. The key idea in such an approach is to present to users a top-N list with topics which they can provide feedback on. User feedback is further captured via user clicks.

Formally, for a given user event history (e.g., video watching) $E = e_1, e_2, \ldots, e_T$, the goal is to estimate the topical distribution of user's interest at time $T + 1$, i.e.: $p(q|E)$. For a large enough data, such distribution can be easily trained using a sequential learning model (e.g., GRU or LSTM). Hence, at questioning time, for a given user event sequence E', questions to be asked can be sampled from the topics with highest likelihood.

Using user's feedback on the displayed top-N topic list, a more personalized recommendation can be made by estimating the likelihood of the next response (items being recommended) $p(r|E', topic = q)$. Such a likelihood can be estimated, for example, by restricting the items being considered for recommendation to those that belong to topic q.

3.4.3 Asking Absolute vs. Relative Questions

The two methods we discussed so far present questions to users in an absolute fashion one by one in a sequential order [82] or as a list of topics to pick [13] as the conversation progress. Yet, such questioning strategy does not fully capture the relative user preferences. An alternative, therefore, is to present clarification questions to users in a relative form [14]. The key idea here is to pick items (or aspects to ask) which would allow to reveal as much information as possible about user's preferences. For example, a relative question may be given about two items that are relatively far from each other in the latent space.

We next discuss the questioning framework of [14]. This framework has two versions, *Absolute* (recommendation) and *Pairwise*. For our discussion purposes we only focus on the Absolute model and refer the reader to [14] for the second one.

As a first assumption, a latent factor recommendation setting is utilized to implement the underlying Absolute model. Specifically, a simplified version of the *Matchbox Recommender* model [70] is implemented. In a nutshell, in this model each user i is modeled by a bias variable $\alpha_i \sim \mathcal{N}(0, \sigma_2^2)$ and a trait vector $u_i \sim \mathcal{N}(0, \sigma_1^2\mathbf{I})$. In a similar manner, each item j has a bias variable $\beta_j \sim \mathcal{N}(0, \sigma_2^2)$ and a trait vector $v_j \sim \mathcal{N}(0, \sigma_1^2\mathbf{I})$. Then, the unobserved affinity between user i and item j is given by $y_{ij} = \alpha_i + \beta_j + u_i^T v_j$. Observations are made $\hat{y}_{ij} \sim \mathcal{N}(y_{ij}, \epsilon_{ij})$, where ϵ_{ij} models the affinity variance, accounting for noise in user preferences. User (dis)like observation is then given by $\hat{r}_{ij} = \mathbf{1}[\hat{y}_{ij} > 1]$. The hyper-parameters σ_1,σ_2 model the variance in traits and biases. The model variables are learned by maximizing the log-posterior over the item and user variables with fixed hyper-parameters, given the training observations [14]. To "bootstrap" the model, item embeddings are learned offline from logged observations, while user parameters are initialized to the mean user values assuming questions are to be asked in a new user setting [14].

Having defined the underlying recommendation model, we next describe two main options for asking questions about user preferences towards items, either *Absolute* or *Relative*. The key idea is to ask a few questions so both user's preferences and question quality can be learned. Such task is implemented in [14] using ideas that are borrowed from *active learning* (query for labels that provide the highest amount of new information) and *bandit learning* (balance between model exploitation and exploration). In the context of conversational recommenders, such a balance may help focus questions on the most relevant part of the latent space, while still considering that highly preferred items may lie in as of yet unexplored areas of the space [14]. At a given time, the model confidence on user preference towards items j is captured by the current variances of the posterior of the noisy affinities y_j^{cold}. As the system asks about an item j^* and observe the user's feedback, the variance of the inferred noisy affinity of this item and of the nearby items in the learned embedding is reduced [14].

The general preference elicitation algorithm is implemented as follows. For a new user i, the noisy affinities y_{ij} are inferred. Then, while more questions are allowed to be asked, an item j^* is picked for absolute (relative) question. Several question (item pick) selection strategies are explored in [14], showing preferable results to an approach based on *Thompson Sampling*: $j^* = \arg\max_j \hat{y}_{ij}$. The user feedback $(i, j^*, 0/1)$ is then incorporated into the model to update the noisy affinities y_i according to [70].

To extend to relative preference elicitation, after an item A is selected, we first assume the user *did not like* this item and incorporate a virtual feedback $(i, A, 0)$ into the model. Then, a second item B is selected based on the inferred updated posterior model when it is used as the prior model. Here, the intuition behind such relative question is that the two items the user is asked to give a relative preference on should be relatively far apart in the latent space. This allows the system to both learn user preferences more efficiently while the user is not forced to choose among very similar items [14], hence introducing diversity.

3.5 Discussion and Future Outlook

Recent advances in NLP now allow to better understand user preferences through interactive dialogue with the recommendation system. Traditional preference elicitation methods have been implemented with a predefined set of preferences or UI-aided tools (e.g., forms). Using textual sources such as item descriptions or user reviews, now allows to extract and process potential preference-related data (e.g., facets) more easily and integrate such data within existing recommendation methods. Moreover, using the mixed initiative nature of conversation, clarification questions about user preferences can be utilized to allow the recommendation system better personalization to its users.

Preference elicitation using NLP methods can be improved in two main ways. First, most existing works utilize user reviews as a source for preference-related metadata. Other sources may be further utilized and should be explored, including, for example, text curated from user discussions and forums, social media or news.

Second, existing preference elicitation NLP-based methods still utilize very simplistic language models learned by traditional deep learning methods such as CNNs and RNNs. New pre-trained language models such as BERT [19] and GPT-3 [6] should be further explored in the context of this task, allowing to improve textual representation and automatic generation of facets and clarification questions.

4 Generating Textual Explanations

As recommendations have become central to shaping decisions, users increasingly demand convincing explanations to help them understand why particular recommendations are made. To this end, the research community has lately focused on studying both how to generate explanations from recommender systems as well as the effect of explanations on influencing user behavior [23, 54]. Chapter 19 provides a detailed overview of explanations in recommender systems. This section complements Chap. 19 by focusing on describing *how* explanations can be generated from text. In more detail, in what follows, we will describe the most representative methods of generating explanations by extracting snippets from text that accompanies an item that is recommended by the system. The input is the user-item matrix as well as the reviews available for the items that are in abundance in most recommender systems. The explanation text can be presented as a description of the item or a comprehensive review (or a set of reviews). The input for most of the algorithms described is the user-item rating matrix as well as the reviews that are written about items. The structure of the Section is as follows: we first discuss deep learning methods that find and show to the user the most useful review (in some cases reviews can be personalized) and then we describe works that extract explicit product features and user opinions from review text.

4.1 Review-Level Explanations

User-generated reviews can be viewed, at least in part, as explanations of the ratings given by users. As discussed previously in Sect. 2, recommendation accuracy can be greatly improved when using review text that is available for the items. At the same time, reviews can help towards an additional goal: explaining why an item is recommended to a user. Here, we will discuss literature that has demonstrated the effective application of reviews in the area of explanations ranging from finding and showing to the user the most useful review to more complex methods where the models can automatically generate personalized reviews to show to each user.

Early works that proposed to combine latent dimensions in rating data with review text had a two-fold contribution: first, the approaches could use the text reviews with ratings to provide accurate recommendations and, second, the approaches could use the most useful reviews to show to the users as an explanation for each recommendation (review level explanations). Most of these representative works are the following: McAuley et al. [47] combine latent rating dimensions with latent topics in reviews learned by LDA topic models. The proposed method is able to generate interpretable topics, that can be used to suggest informative reviews. Along the same lines, Diao et al. [20] also propose to combine ratings with review text to identify aspects (relative to topics). The key difference with [47] is that in this work, the authors focus on both learning the aspects as well as learning the sentiments of these aspects (negative vs. positive). From the deep learning community, the neural attentional regression model with review-level explanations model (`NARRE`) [9] uses the text of the reviews to both predict item ratings and learn the usefulness of the reviews. In this approach, only the highly-useful reviews are provided as explanations to the users. The explanations provided in all the above studies are not personalized, i.e., when the recommender system comes up with a specific item recommendation, all users will be shown the exact same explanation which is the most useful reviews. In what follows we describe works that generate personalized explanations in the form of reviews.

4.1.1 Multi-Task Learning for Recommendation and Personalized Explanation

Recently, the research community has shown increased interest in personalized review generation that can serve as a form of explanation to the users. Lu et al. [45] jointly learn recommendations and explanations by introducing a multi-task learning framework. Instead of finding the most useful review, the proposed model generates new reviews that are personalized to the taste of each user. More specifically, the review text is used for learning the user preferences and item properties, and implicit and explicit feedback (e.g., clicks or ratings), is used to learn the level of interest of a user in the attributes of an item. The approach employs a matrix factorization model (see Chap. 3 for more details) that generates the user-item ratings combined with a sequence-to-sequence model that generates a personalized review (that serves as an explanation) for each user-item recommendation.

In order to generate an unbiased personalized review (i.e., explanation), Lu et al. [45] use an adversarial approach over a sequence-to-sequence model, where a review generator and discriminator network are trained simultaneously. For the generator network, which eventually produces the explanations, the set d_i of all reviews written by user i is mapped onto a textual feature vector $\widetilde{U}$ and subsequently fed into a bi-directional Gated Recurrent Unit (GRU) [16] which concatenates the last-step hidden forward and backward activations into a vector-based representation h_T for the review. Overall, given the vector $\widetilde{U}$, the model generates reviews that attempt to maximize the probability of a T'-lengthed review $y_{i,1}, \ldots, y_{i,T'}$ given the user's textual vector:

$$p(y_{i,1}, \ldots, y_{i,T'}|\widetilde{U}) = \prod_{t=1}^{T'} p(y_{i,t}|\widetilde{U}_i, y_{i,1}, \ldots, y_{i,t-1})$$

Once the reviews are generated, they are next fed into the discriminator that attempts to discern real from artificially-generated (from the generator) reviews. In this adversarial setting, the discriminator is implemented as a convolutional neural net, typical in text classification tasks. More specifically, the review words are mapped into vectors which are fed into a convolutional, max-pooling and fully-connected projection layers (in this order) while the final output is adjusted using a sigmoid function. Overall, the discriminator network attempts, through sampling of reviews and using policy gradient descent [75], to maximize the function:

$$max_{\phi} E_{Y \sim p_{data}}[log D_{\phi}(Y)] + E_{Y' \sim G_{\theta}}[log(1 - D_{\phi}(Y'))]$$

where $Y \sim p_{data}$ are the ground-truth (i.e., real) sampled reviews and $Y' \sim G_{\theta}$ are the generated ones. In the process, the learned textual features and the matrix-factorization-based textual features are regularized in order to allow the sequence-to-sequence model to leverage the user preferences identified from the collaborative filtering step. Finally, personalized recommendation explanations are generated by employing a review decoder that combines the vectors $\widetilde{U}_i$ (i.e., the reviews of user i) and $\widetilde{V}_j$ (i.e., the reviews of item j) to generate reviews for a given user-item pair.

Another popular way of providing useful explanations to recommendations is by generating (oftentimes personalized) synthetic reviews. Ouyang et al. [56] combine three generative models that provide natural language explanations while leveraging the available helpfulness votes of existing reviews. The approach also takes into account the set of item attributes that may be of interest to the user while generating the reviews. At a high level, the goal is to maximize the likelihood of an explanation $e = (y_1, \ldots, y_l)$ of length l for a given set of input attributes for item i $a_i = (a_i, \ldots, a_{|a|})$:

$$p(e|a) = \prod_{t=1}^{l} p(y_t|y_{i,1}, \ldots, y_{i,t-1}, a)$$

To this end, Ouyang et al. [56], propose three textual models that produce text as both character and word sequences using the user id, item id, item rating and review helpfulness score as input. More specifically, the approach starts in the encoding phase by identifying all word tokens with their positions in the review corpus that will be used both during the encoding (training step) and decoding (generating step). A GCN (generative concatenative) model is employed at the beginning of the process. The goal of the GCN is to learn the relations of attributes and text by considering them as one (concatenated) input. More specifically, the model accepts input of the form $X_t' = [x_t^r : x_t^a]$ where x_t^r is the encoded review text and x_t^a are the one-hot encoded attributes at time step t. After the first GCN model, a context and attention model are utilized that aim to transform the attributes to fixed-length [21] embeddings. The goal of these two models is to learn how attributes and text aligns and to provide good initialization weights for the decoder. At the end of the encoding phase, encoded attributes are reshaped to the proper decoder shape: $A = tanh(H[x_i^a, \dots, x_{|a|}^a] + b_a)$ and are provided as initialization weights for the decoder.

The decoding phase utilizes Recurrent Neural Networks (RNNs) with long short-term memory (LSTM) [28]. Given, at a time t, inputs x_t, the LSTM cell state C_{t-1}, the previous output H_t, W the weights, b the bias and $\hat{C}$ the candidate state, the computation of the decoding step proceeds as follows with $\odot$ denoting an element-wise product operation:

$$\begin{aligned}
\hat{C}_t &= tanh(W_x^c x_t + W_h^c H_{t-1} + b_c)\\
f_t &= \sigma(W_x^f x_t + W_h^f H_{t-1} + b_f)\\
i_t &= \sigma(W_x^i x_t + W_h^i H_{t-1} + b_i)\\
C_t &= f_t \odot C_{t-1} + i_t \odot C_t'\\
o_t &= \sigma(W_x^o x_t + W_h^o H_{t-1} + b_o)\\
H_t &= o_t \odot tanh(C_t)
\end{aligned}$$

As a final step, for text generation, the decoder output H_t is provided as input to a softmax function that essentially attempts to maximize the probability $p(y_t | y_{<t}, a)$ by greedily inferring the characters and words. As a result, the final explanation review text is produced by inferring the index Y_t of the generated character/word through the following process:

$$\begin{aligned}
p(y_t | y_{i,1}, \dots, y_{i,t-1}, a) &= softmax(WH_t + b)\\
Y_t &= argmaxp(y_t | y_{i,1}, \dots, y_{i,t-1}, a)
\end{aligned}$$

4.1.2 Providing Explanations for Recommendations in Reciprocal Environments

Reciprocal environments involve applications such as job searching or online dating, where a recommendation should be mutually beneficial to two or more users. Kleinerman et al. [35] study how to generate useful reciprocal explanations for such user-matching (i.e., reciprocal) recommendations on top of two-sided collaborative filtering approaches [53, 78]. In general, the reciprocal setting assumes that, each user x, has, for each predefined attribute a in the system, provided their personal values for these attributes $A_x = \{v_a\}$, together with the (implicit or explicit) user's preference $p_{x,a}$ for each of the attributes. In Kleinerman et al. [35], preferences are identified implicitly based on messages sent among users. More specifically, $p_{x,a}$ is the number of messages sent by x to users that have v_a as a value for attribute a.

Kleinerman et al. [35] approach the problem of reciprocal explanations for two users x and y by generating two one-sided explanations $e_{x,y}$ and $e_{y,x}$. To this end, they propose two single-sided explanation generation approaches named *Transparent* and *Correlation-based*.

In the *Transparent* approach, in order for the system to explain recommending user y to user x, it returns the top-k attributes of y that are considered the most important based on x's preferences $p_{x,a}$.

In the *Correlation-based* approach, the system measures the correlation between a given attribute value v_a in user y's profile and the likelihood of user x indicating that specific attribute as a preference (specifically, sending a message in [35]). For each user x, all users $I = i$ that x has interacted with are first identified and two binary metrics are computed: $M_x(i)$ which captures whether x has indicated a preference (sent a message) to i, and $S_{x,v_a}(i)$ which captures whether user i has v_a in their profile. Then, given the two binary vectors M and S, we compute the Pearson's correlation metric between M and S for each attribute value v_a. The top-k attributes with the highest correlation are considered to be the single-sided explanations $e_{x,y}$.

4.2 Feature-Level Explanations

In what follows, we describe works that extract explicit product features and user opinions from review text. To infer user opinions, sentiment analysis is used that in some cases also leverages the sentiments of the social connections of the users.

4.2.1 Explicit Factor Models for Explainable Recommendation Based on Phrase-Level Sentiment Analysis

Zhang et al. [83] present an approach called Explicit Factor Model. The approach starts by identifying product aspects (features) that may be of interest to the users through sentiment analysis on the reviews and uses learned and latent features to generate both recommendations and aspect-level explanations.

Initially, the approach identifies the feature descriptions from the reviews $\mathcal{F}$, the opinion descriptions $(\mathcal{O})$ and the feature sentiments, i.e., $(\mathcal{F}, \mathcal{O})$ pairs. The set $\mathcal{L} = (\mathcal{F}, \mathcal{O}, \mathcal{S})$ is then used to generate feature, review-sentiment $(\mathcal{F}, \mathcal{S}')$ pairs. Next, a user-feature attention matrix X with $\mathcal{F} = \{F_1, F_2, \ldots, F_p\}$ columns (i.e., features), and $\mathcal{U} = \{u_1, \ldots, u_m\}$ rows (i.e., users) is generated with each element being either 0 if user u_i did not mention feature F_j, or the value of a sigmoid function $\sigma(t_{ij})$ otherwise. Similarly, an item-feature quality matrix Y is generated with $\mathcal{P} = \{p_1, \ldots, p_n\}$ rows and $\mathcal{F} = \{F_1, F_2, \ldots, F_p\}$ columns. Again, each element is either 0 if item p_i does review feature F_j, or the value of a sigmoid function $\sigma(t_{ij})$. The final factorization model is augmented with additional r' latent factors $H_1 \in \mathbb{R}_+^{m\times r'}$, $H2 \in \mathbb{R}_+^{n\times r'}$ attempting to capture hidden factors affecting the user decision $P = [U_1 \; H_1]$ and $Q = [U_2 \; H_2]$. It also uses the user-item ratings A to capture both explicit and implicit features both for recommendations and explanations:

$$\min_{U_1,U_2,V,H1,H2} \{||PQ^T - A||_F^2 + ||Y_1V^T - X||_F^2 + ||U_2V^T - Y||_F^2$$

$$+ (||U_1||_F^2 + ||U_2||_F^2) + (||H_1||_F^2 + ||H_2||_F^2) + ||V||_F^2\}$$

$$s.t.\; U_1 \in \mathbb{R}_+^{m\times r}, U_2 \in \mathbb{R}_+^{n\times r}, V \in \mathbb{R}_+^{p\times r}, H_1 \in \mathbb{R}_+^{m\times r'}, H_2 \in \mathbb{R}_+^{n\times r'}$$

Given the solution of the factorization, a recommendation of item j to user i can be computed as: $R_{ij} = \alpha \frac{\sum_{c\in C_i} \hat{X}_{ic}\cdot\hat{Y}_{jc}}{kN} + (1-\alpha)\hat{A}_{ij}$, where $\hat{X} = U_1V^T$, $\hat{Y} = U_2V^T$, $A = U_1U_2^T + H_1H2^T$, N is the maximum scale of ratings (e.g. 5), α is a scaling factor, and C_i are the columns of $\hat{X}$ with the k largest values. Explanations are generated by identifying the best and worst-performing features $F_{c_{best}}$ and $F_{c_{worst}}$ of a product p_j for each user u_i: $c_{best} = argmax_{c\in C_i}\hat{Y}_{jc}$ and $c_{worst} = argmin_{c\in C_i}\hat{Y}_{jc}$.

4.2.2 Social Collaborative Viewpoint Regression with Explainable Recommendations

In addition to concepts, topics, reviews and their sentiments, Ren et al. [60], also utilize the sentiments of users' social connections in order to create a notion of viewpoints, i.e., tuples of concepts, topics, review sentiments and social-connection sentiments. Ren et al. [60] consider the typical setting of users $\mathcal{U} = \{u_1, \ldots, u_U\}$, items $\mathcal{I} = \{i_1, \ldots, i_I\}$, pairs $\mathcal{Q} = \{(u, i)\}$ of user-ratings $r_{u,i}$ and reviews $\mathcal{D} = \{d_1, \ldots, d_{|Q|}\}$ with each review being a set of words $d = \{w_1, \ldots, w_{|d|}\}$ and additionally consider a set of trusted social relations $\mathcal{T}_{u_i,u_j}$ where user u_i trusts user u_j.

Topics z are defined as a probability distribution over words, while concepts e is a feature in close proximity to a topic. Sentiments l over words w are considered to depend on topics. A viewpoint is a finite mixture over $v =< e, z, l >$ tuples. The user's u rating values θ^u are then defined as an $R \times V$ matrix with each element θ^u_{i,v_j} corresponding to the probability of rating r, for a given pair of user-viewpoint (u, v). For discovering the concepts, Ren et al., employ a word2vec [50] model while they use a recursive deep model for sentiment analysis [68].

In order to utilize the social connections, a latent factor model is proposed where, in addition to viewpoints, topics, concepts and sentiments, the trusted relations of a user u are modeled by considering the viewpoint distribution of u's social relations $\{\theta_v^{u1}, \ldots, \theta_v^{uF_u}\}$ and a base distribution $\theta^0_{u,v}$. Inference is performed using a Gibbs EM sampler. The E-step approximates the distribution $p(\mathcal{V}, \mathcal{Z}, \mathcal{L}|\mathcal{W}, \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{F})$, while the M-step maximizes each user's u viewpoint distribution θ_u. Once the Gibbs EM sampling is completed, for each user u we compute a recommendation based on the maximum value of predicted rating probabilities: $P(r_{u_i} = r|u, i) = \sum_{v \in \mathcal{V}} \theta^u_{r,v} \cdot \pi_{i,v}$, with $P(v|i) = \pi_{v,i}$ being the viewpoint distribution of item i. Explanations are finally derived by presenting to user u the topics of the recommended items.

4.2.3 Review-Aware Explainable Recommendation by Modeling Aspects

Aspects (or features) of recommended items can be very useful in providing good explanations. He et al. [27] utilize textual reviews to identify product aspects which are modeled as a tripartite graph of users, items and aspects. The problem of recommendation and explanation is hence viewed as node ranking over this tripartite graph. He et al. [27], identify the aspects of the items [80] and extract feature, opinion, sentiment tuples (F, O, S). Next, a tripartite graph $G = (U \cup P \cup A, E_{UP} \cup E_{UA} \cup E_{PA})$ is defined, where U, P and A are the vertices corresponding to users, items and aspects, while E_{UP}, E_{UA} and E_{PA} are the user-to-items, user-to-aspects and items-to-aspects edges respectively. A user u_i, rating an item p_j with a review containing aspect a_k is a triangle with edges e_{ij}, e_{jk}, e_{ik} in the graph. The edges are weighted, capturing the strength of relationship among the nodes. Matrices R, Y and X represent the weights for E_{UP}, E_{UA} and E_{PA} respectively. The goal is to identify a ranking function f that will produce the predicted preference of user u for p.

The approach of He et al. [27] (named TriRank) learns f with two useful properties encoded as regularizers: (a) smoothness, i.e., nearby nodes should have similar scores, and, (b) fitting, i.e., the learned function should not cause significant deviation from the observed data. The overall function that the approach aims to optimize through alternating least squares (ALS) is:

$$Q(f) = \alpha \sum_{i,j} r_{ij} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(p_j)}{\sqrt{d_j^p}})^2 + \beta \sum_{j,k} x_{jk} (\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2$$
$$+ \gamma \sum_{i,k} y_{ik} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2 + \eta_U \sum_i (f(u_i) - u_i^0)^2$$
$$+ \eta_P \sum_j (f(p_j) - p_j^0)^2 + \eta_A \sum_k (f(a_k) - a_k^0)^2$$

The parameters α, β, γ capture the desired weight of the smoothness constraint on the user-item, item-aspect and user-aspect edges, while η_U, η_P and η_A capture the desired weight of the fitting constraint on users, items and aspects. Explanations are provided as recommended items, i.e., the aspects of a recommended item are presented to the user together with how well the item captures a specific aspect.

4.3 Discussion and Future Outlook

All methods described above generate explanations from the review text. As a result, the question here is how one can deal with items that have a very small number of reviews. One possible suggestion is to use additional features that are available for the items (e.g., description, title).

Most of the studies described above are using offline metrics (such as perplexity) that focus on evaluating the goodness of a language model is in evaluating the quality of the explanations. However, a real recommender system cannot rely exclusively on the results of the offline evaluation metrics since there is a large number of other factors that play a significant role in the successful application of explanations. For example, recent studies [39, 51] have shown that users with different personality traits have different preferences in explanations. As a result, more user studies are needed to better understand how the explanations provided by the aforementioned methods are perceived by real end-users of a recommendation engine.

Another very promising direction is for the explanations to account for privacy. Consider, for example, an explanation of the form “We recommend bar Crudo because it is one of the most popular gay bars in the area”. It is obvious that such an explanation involves several potential issues regarding the privacy of the user. Document understanding techniques (e.g., GPT-3 [6]) can be deployed to protect users’ privacy that may be compromised through explanations.

Another future work direction is to design and implement interactive systems that can foster exploration from the users. The majority of work described above provided natural language explanations in a static context where the user could not interact with the explainable recommender system and give feedback. In addition, most of the work did not allow for further exploration of the provided explanations by the interested user. Scrutability and control are two essential characteristics that an explainable recommender system should offer, thus it is worth for the natural language community to study this approach.

A final promising direction is to study whether explanations should participate in the process of ranking the recommendations. All methods described above first rank the recommendations and then generate the explanations for the top ranked items, i.e., the process of generating explanations does not affect the ranking process. A promising future direction would be to incorporate explanations in the prediction process of the recommender system in order to improve the accuracy and transparency of recommendations.

References

1. R.K. Bakshi, N. Kaur, R. Kaur, G. Kaur, Opinion mining and sentiment analysis, in *3rd International Conference on Computing for Sustainable Global Development*, INDIACom'16, pp. 452–455 (2016)
2. Y. Bao, H. Fang, J. Zhang, Topicmf: Simultaneously exploiting ratings and reviews for recommendation, in *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, AAAI'14, pp. 2–8 (2014)
3. K. Bauman, B. Liu, A. Tuzhilin, Aspect based recommendations: Recommending items with the most valuable aspects based on user reviews, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'17, pp. 717–725 (2017)
4. D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
5. D.J. Brenes, D. Gayo-Avello, K. Pérez-González, Survey and evaluation of query intent detection methods, in *Proceedings of the WSDM 2009 Workshop on Web Search Click Data*, WSCD'09, pp. 1–7 (2009)
6. T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners. Preprint (2020). arXiv:2005.14165
7. D. Carmel, H. Roitman, N. Zwerdling, Enhancing cluster labeling using wikipedia, in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pp. 139–146 (2009)
8. R. Catherine, W. Cohen, Transnets: Learning to transform for recommendation, in *Proceedings of the 11th ACM Conference on Recommender Systems*, RecSys'17, pp. 288–296 (2017)
9. C. Chen, M. Zhang, Y. Liu, S. Ma, Neural attentional rating regression with review-level explanations, in *Proceedings of the 2018 World Wide Web Conference*, WWW'18, pp. 1583–1592 (2018)
10. L. Chen, P. Pu, Critiquing-based recommenders: survey and emerging trends. User Model. User Adap. Inter. **22**(1-2), 125–150 (2012)
11. Z. Cheng, Y. Ding, X. He, L. Zhu, X. Song, M.S. Kankanhalli, Aˆ 3ncf: An adaptive aspect attention model for rating prediction, in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 3748–3754 (2018)
12. Z. Cheng, Y. Ding, L. Zhu, M. Kankanhalli, Aspect-aware latent factor model: Rating prediction with ratings and reviews, in *Proceedings of the 2018 World Wide Web Conference*, WWW'18, pp. 639–648 (2018)
13. K. Christakopoulou, A. Beutel, R. Li, S. Jain, E.H. Chi, Q&R: A two-stage approach toward interactive recommendation, in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD'18, pp. 139–148 (2018)

14. K. Christakopoulou, F. Radlinski, K. Hofmann, Towards conversational recommender systems, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD'16, pp. 815–824 (2016)
15. Y.-N. Chuang, C.-M. Chen, C.-J. Wang, M.-F. Tsai, Y. Fang, E.-P. Lim, TPR: Text-aware preference ranking for recommender systems, in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM'20, pp. 215–224 (2020)
16. J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, in *NIPS 2014 Workshop on Deep Learning* (2014)
17. D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), in *4th International Conference on Learning Representations (Poster)*, ICLR'16 (2016)
18. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P.P. Kuksa, Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
19. J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL'19, pp. 4171–4186 (2019)
20. Q. Diao, M. Qiu, C.-Y. Wu, A.J. Smola, J. Jiang, C. Wang, Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars), in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'14, pp. 193–202 (2014)
21. L. Dong, S. Huang, F. Wei, M. Lapata, M. Zhou, K. Xu, Learning to generate product reviews from attributes, in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, EACL'17, pp. 623–632 (2017)
22. P. Ferragina, U. Scaiella, Tagme: on-the-fly annotation of short text fragments (by wikipedia entities), in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM'10, pp. 1625–1628 (2010)
23. G. Friedrich, M. Zanker, A taxonomy for generating explanations in recommender systems. AI Magazine **32**(3), 90–98 (2011)
24. S. Frumerman, G. Shani, B. Shapira, O.S. Shalom, Are all rejected recommendations equally bad? towards analysing rejected recommendations, in *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization*, UMAP'19, pp. 157–165 (2019)
25. F.A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with LSTM. Neural Computation **12**(10), 2451–2471 (1999)
26. G. Hadash, O.S. Shalom, R. Osadchy, Rank and rate: multi-task learning for recommender systems, in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys'18, pp. 451–454 (2018)
27. X. He, T. Chen, M.-Y. Kan, X. Chen, Trirank: Review-aware explainable recommendation by modeling aspects, in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM'15, pp. 1661–1670 (2015)
28. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
29. S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *ICML'15*, pp. 448–456 (2015)
30. A. Jacovi, O.S. Shalom, Y. Goldberg, Understanding convolutional neural networks for text classification, in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, BlackboxNLP'18, pp. 56–65 (2018)
31. D. Jannach, A. Manzoor, W. Cai, L. Chen, A survey on conversational recommender systems. Preprint (2020). arXiv:2004.00646
32. Y. Kim, Convolutional neural networks for sentence classification, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP'14, pp. 1746–1751 (2014)

33. D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in *3rd International Conference on Learning Representations*, ICLR'15 (2015)
34. D.P. Kingma, M. Welling, Auto-encoding variational Bayes, in *2nd International Conference on Learning Representations*, ICLR'14 (2014)
35. A. Kleinerman, A. Rosenfeld, S. Kraus, Providing explanations for recommendations in reciprocal environments, in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys'18, pp. 22–30 (2018)
36. N. Kolitsas, O.-E. Ganea, T. Hofmann, End-to-end neural entity linking, in *Proceedings of the 22nd Conference on Computational Natural Language Learning*, CoNLL'18, pp. 519–529 (2018)
37. W. Kong, J. Allan, Extracting query facets from search results, in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'13, pp. 93–102 (2013)
38. W. Kong, J. Allan, Precision-oriented query facet extraction, in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM'16, pp. 1433–1442 (2016)
39. P. Kouki, J. Schaffer, J. Pujara, J. O'Donovan, L. Getoor, Personalized explanations for hybrid recommender systems, in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI'19, pp. 379–390 (2019)
40. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer, Neural architectures for named entity recognition, in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL'16, pp. 260–270 (2016)
41. Q. Le, T. Mikolov, Distributed representations of sentences and documents, in *The 31st International Conference on Machine Learning*, ICML'14, pp. 1188–1196 (2014)
42. J. Li, T. Luong, D. Jurafsky, A hierarchical neural autoencoder for paragraphs and documents, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, ACL'15, pp. 1106–1115 (2015)
43. G. Ling, M.R. Lyu, I. King, Ratings meet reviews, a combined approach to recommend, in *Proceedings of the 8th ACM Conference on Recommender systems*, RecSys'14, pp. 105–112 (2014)
44. B. Liu, I. Lane, Attention-based recurrent neural network models for joint intent detection and slot filling, in *17th Annual Conference of the International Speech Communication Association*, Interspeech'16, ed. by N. Morgan, pp. 685–689 (2016)
45. Y. Lu, R. Dong, B. Smyth, Why i like it: Multi-task learning for recommendation and explanation, in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys'18, pp. 4–12 (2018)
46. K. Luo, S. Sanner, G. Wu, H. Li, H. Yang, Latent linear critiquing for conversational recommender systems, in *Proceedings of The Web Conference 2020*, WWW'20, pp. 2535–2541 (2020)
47. J. McAuley, J. Leskovec, Hidden factors and hidden topics: understanding rating dimensions with review text, in *Proceedings of the 7th ACM conference on Recommender systems*, RecSys'13, pp. 165–172 (2013)
48. J. McAuley, R. Pandey, J. Leskovec, Inferring networks of substitutable and complementary products, in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'15, pp. 785–794 (2015)
49. Z.A. Merrouni, B. Frikh, B. Ouhbi, Automatic keyphrase extraction: a survey and trends. J. Intell. Inf. Syst. **54**, 1–34 (2019)
50. T. Mikolov, K. Chen, G.S. Corrado, J. Dean, Efficient estimation of word representations in vector space, in *1st International Conference on Learning Representations*, ICLR'13 (2013)
51. M. Millecamp, K. Verbert, S. Naveed, J. Ziegler, To explain or not to explain: the effects of personal characteristics when explaining feature-based recommendations in different domains, in *Proceedings of the 6th Joint Workshop on Interfaces and Human Decision Making for Recommender Systems*, IntRS'19, pp. 10–18 (2019)

52. V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814 (2010)
53. J. Neve, I. Palomares, Latent factor models and aggregation operators for collaborative filtering in reciprocal recommender systems, in *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys'19, pp. 219–227 (2019)
54. I. Nunes, D. Jannach, A systematic review and taxonomy of explanations in decision support and recommender systems. User Model. User Adap. Inter. **27**(3–5), 393–444 (2017)
55. S. Osiński, J. Stefanowski, D. Weiss, Lingo: Search results clustering algorithm based on singular value decomposition. Intell. Inf. Syst., 359–368 (2004)
56. S. Ouyang, A. Lawlor, F. Costa, P. Dolog, Improving explainable recommendations with synthetic reviews, in *RecSys Workshop on Deep Learning for Recommender Systems*, DSLR'18 (2018)
57. M. Paşca, E. Alfonseca, Web-derived resources for web information retrieval: From conceptual hierarchies to attribute hierarchies, in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'09, pp. 596–603 (2009)
58. A. Paterek, Improving regularized singular value decomposition for collaborative filtering, in *Proceedings of KDD Cup*, pp. 5–8 (2007)
59. F. Radlinski, N. Craswell, A theoretical framework for conversational search, in *Proceedings of the 2017 Conference on Human Information Interaction and Retrieval*, CHIIR'17, pp. 117–126 (2017)
60. Z. Ren, S. Liang, P. Li, S. Wang, M. de Rijke, Social collaborative viewpoint regression with explainable recommendations, in *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, WSDM'17, pp. 485–494 (2017)
61. S. Rendle, Factorization machines, in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM'10, pp. 995–1000 (2010)
62. S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, BPR: bayesian personalized ranking from implicit feedback, in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, UAI'09, pp. 452–461 (2009)
63. Y. Salem, J. Hong, History-aware critiquing-based conversational recommendation, in *Proceedings of the 22nd International Conference on World Wide Web*, WWW'13 Companion, pp. 63–64 (2013)
64. O.S. Shalom, G. Uziel, A. Karatzoglou, A. Kantor, A word is worth a thousand ratings: Augmenting ratings using reviews for collaborative filtering, in *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, SIGIR'18, pp. 11–18 (2018)
65. O.S. Shalom, G. Uziel, A. Kantor, A generative model for review-based recommendations, in *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 353–357 (2019)
66. S. Shi, H. Zhang, X. Yuan, J.-R. Wen, Corpus-based semantic class mining: distributional vs. pattern-based approaches, in *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING'10, pp. 993–1001 (2010)
67. Y. Shi, M. Larson, A. Hanjalic, Unifying rating-oriented and ranking-oriented collaborative filtering for improved recommendation. Information Sciences **229**, 29–39 (2013)
68. R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, pp. 1631–1642 (2013)
69. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
70. D.H. Stern, R. Herbrich, T. Graepel, Matchbox: large scale online bayesian recommendations, in *Proceedings of the 18th International Conference on World Wide Web*, WWW'09, pp. 111–120 (2009)

71. Y. Sun, Y. Zhang, Conversational recommender system, in *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR'18, pp. 235–244 (2018)
72. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems*, NIPS'17, pp. 5998–6008 (2017)
73. Q. Wang, H. Yin, H. Wang, Q. Viet Hung Nguyen, Z. Huang, L. Cui, Enhancing collaborative filtering with generative augmentation, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD'19, pp. 548–556 (2019)
74. X. Wang, D. Chakrabarti, K. Punera, Mining broad latent query aspects from search sessions, in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'09, pp. 867–876 (2009)
75. R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3–4), 229–256 (1992)
76. F. Wu, J. Madhavan, A. Halevy, Identifying aspects for web-search queries. J. Artif. Intell. Res. **40**, 677–700 (2011)
77. G. Wu, K. Luo, S. Sanner, H. Soh, Deep language-based critiquing for recommender systems, in *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys'19, pp. 137–145 (2019)
78. P. Xia, B. Liu, Y. Sun, C. Chen, Reciprocal recommendation system for online dating, in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM'15, pp. 234–241 (2015)
79. O. Zamir, O. Etzioni, Grouper: A dynamic clustering interface to web search results. Computer Networks **31**(11-16), 1361–1374 (1999)
80. L. Zhang, B. Liu, S.H. Lim, E. O'Brien-Strain, Extracting and ranking product features in opinion documents, in *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING'10, pp. 1462–1470 (2010)
81. Y. Zhang, Q. Ai, X. Chen, W.B. Croft, Joint representation learning for top-n recommendation with heterogeneous information sources, in *Proceedings of the 26th ACM International Conference on Information and Knowledge Management*, CIKM'17, pp. 1449–1458 (2017)
82. Y. Zhang, X. Chen, Q. Ai, L. Yang, W.B. Croft, Towards conversational search and recommendation: System ask, user respond, in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM'18, pp. 177–186 (2018)
83. Y. Zhang, G. Lai, M. Zhang, Y. Zhang, Y. Liu, S. Ma, Explicit factor models for explainable recommendation based on phrase-level sentiment analysis, in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR'14, pp. 83–92 (2014)
84. L. Zheng, V. Noroozi, P.S. Yu, Joint deep modeling of users and items using reviews for recommendation, in *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, WSDM'17, pp. 425–434 (2017)