Explainable Product Search with a Dynamic Relation Embedding Model

QINGYAO AI*, School of Computing, University of Utah, USA
YONGFENG ZHANG, Department of Computer Sciences, Rutgers University, USA
KEPING BI and W. BRUCE CROFT, College of Information and Computer Sciences, University of Massachusetts Amherst, USA

Product search is one of the most popular methods for customers to discover products online. Most existing studies on product search focus on developing effective retrieval models that rank items by their likelihood to be purchased. However, they ignore the problem that there is a gap between how systems and customers perceive the relevance of items. Without explanations, users may not understand why product search engines retrieve certain items for them, which consequentially leads to imperfect user experience and suboptimal system performance in practice. In this work, we tackle this problem by constructing explainable retrieval models for product search. Specifically, we propose to model the "search and purchase" behavior as a dynamic relation between users and items, and create a dynamic knowledge graph based on both the multi-relational product data and the context of the search session. Ranking is conducted based on the relationship between users and items in the latent space, and explanations are generated with logic inferences and entity soft matching on the knowledge graph. Empirical experiments show that our model, which we refer to as the Dynamic Relation Embedding Model (DREM), significantly outperforms the state-of-the-art baselines and has the ability to produce reasonable explanations for search results.

CCS Concepts: • **Information systems** → *Document representation*; *Retrieval models and ranking*; *Retrieval tasks and goals*;

Additional Key Words and Phrases: Product search, explainable model, knowledge graph, relation embedding

1 INTRODUCTION

Product search represents a special retrieval problem where users submit queries to a search engine to find products. As the e-commerce market keeps growing[1] and millions of new products are introduced every day, search has become one of the most effective and popular methods for people to discover products. According to a recent marketing report,[2] the majority of sales on Amazon comes from its product search service. Thus, the quality of product search affects both user experience with online shopping and the profits of e-commerce companies.

Due to their applications in e-commerce, product search engines are usually optimized for user transactions. In a typical product search scenario, users submit a query to a search engine, and the system returns a list of products for users to explore. Items on the search engine result pages are often sorted by their likelihood to be purchased so the number of user transactions can be maximized in each search session [6, 17, 57].

Showing relevant items on the top, however, is not enough to guarantee the effectiveness of product search. Existing studies based on this paradigm often simplify the problem of product search by assuming that users will purchase an item as long as it is observed and relevant [6, 57]. They ignore the fact that there is a significant gap between the item relevance perceived by search engines and e-shopping users [75]. Because purchasing is expensive and highly personal [6], users often need a good reason to justify their purchases. As modern product search systems become increasingly sophisticated, it is difficult for normal users to understand why search engines retrieve certain items for them. A direct consequence is that users may not perceive a retrieved item as relevant even when it satisfies their search intents.

Therefore, to actually optimize user purchases, a good product search engine needs to retrieve relevant products as well as provide good explanations of why retrieved items should be interesting to users. Previous studies on product recommendation have shown that providing appropriate explanations significantly improves user acceptance for recommended items [26, 55]. It benefits recommendation systems in multiple ways, including user satisfaction, system transparency, debugging complexity, and so on [11, 15, 56, 74]. It is reasonable to assume that providing explanations for retrieval results will be equally beneficial for product search.

Despite its potential, the explainability of retrieval systems has not been well studied in product search. There are two problems that limit the development of explainable product search systems. First, purchasing is a complicated behavior, as it depends on multiple factors such as user preference, product presentation, and search context. To provide high-quality explanations, we need to consider the relationship between users and products from multiple angles (e.g., brands, categories, etc.). As far as we know, no existing retrieval model can directly incorporate different product knowledge for product search. Second, producing a readable explanation requires the system to have logical reasoning. For example, the system should be able to infer that "*Bob* likes *Apple* products" after seeing him search and purchase multiple products from *Apple*. An explanation is reasonable and effective only when it is formulated based on well-grounded logic, while how to construct such a product retrieval model with logical reasoning ability is still an open question for the IR community.

In this article, we present our initial attempt to tackle the problem of explainable product search. Inspired by the studies of relation prediction in knowledge base [12, 13], we propose to create a unified knowledge graph on multiple types of product data and conduct retrieval with it. Our motivation is to integrate multi-relational product information for search and generate explanations with logic inference on the knowledge graph. Previous studies on knowledge graphs and

---

embeddings mainly focus on modeling static data relationships that do not change. In product search, however, the relationships between users and items are not deterministic among different search sessions. For example, a camera lens is relevant to a photographer when she searches for "camera," but not when she searches for "toothbrush." To solve this problem, we propose a Dynamic Relation Embedding Model (DREM) that dynamically models the relationship between users and items based on the search context. Although inferring the relationship of an arbitrary user-item pair with observed data is often infeasible due to data sparsity [69], we show that reasonable explanation paths can be extracted with our proposed method through entity soft matching. Empirical experiments and analysis with Amazon benchmark datasets show that incorporating different product knowledge with DREM has significant potential for explainable product search.

Our main contributions can be summarized as follows:

- We propose a Dynamic Relation Embedding Model to construct a session-dependent knowledge graph for product retrieval.
- We propose a Soft Matching Algorithm to efficiently extract explainable paths with knowledge embeddings for search explanations.
- We conducted both retrieval experiments and case studies to verify the effectiveness of the proposed approach in product retrieval and explainable search.

The rest of this article is organized as follows: In Section 2, we discuss the related work. Then we introduce our approach and how to extract explanations for product search in Sections 3 and 4. We describe our experimental setup and results in Sections 5 and 6. We conclude our work and discuss future studies in Section 7.

2 RELATED WORK

There are four lines of studies that are related to our work: product search, explainable systems, knowledge embedding, and neural information retrieval.

2.1 Product Search

Product search refers to the problem of retrieving relevant products for customers to satisfy their purchase intents. Previous studies on product search mainly focus on retrieving products based on their structured aspects such as brand, category, and so on. For example, Lim et al. [35] propose a document profile model to suggest semantic tags for each item based on their structural aspects, so we can retrieve products by matching queries with multiple product aspects simultaneously. Despite their success, searching with structured data cannot satisfy the need of e-shopping users today, as their intents become more complicated. As shown by Duan et al. [17], writing structured search queries (e.g., SQL) is usually considered hard and inconvenient for search users. In most cases, queries submitted to product search engines are free-form text that is difficult to structure. However, there often exists a large vocabulary gap between the descriptions of products and user queries [6, 57]. Nurmi et al. [44] find that the words customers used to write their shopping lists are often different from those sellers used to describe their products. Because of these problems, IR researchers have proposed to construct semantic latent space for product search and conduct matching between queries and products with their latent representations. For instance, Yu et al. [65] construct a Latent Dirichlet Allocation model with product information and use it to diversify e-commerce search results. Van Gysel et al. [57] introduce a Latent Semantic Embedding model that maps and matches n-grams from queries and product descriptions into a hidden space. Ai et al. [3, 6] constructed hierarchical and attentive embedding models that model the generative process of user purchases and product reviews so items can be ranked by their likelihood to be purchased given the user and the query. Guo et al. [20] propose a TranSearch model that can

directly match text queries with product images. Bi et al. [10] use the embedding representations of clicked items as context information to refine the ranking of retrieved products. Zhang et al. [73] and Bi et al. [9] also extract product aspects from review data and build embedding networks that encode both items and their extracted aspects for conversational product search and recommendation. Besides latent embedding techniques, there is a variety of studies [7, 28, 30] on extracting different text and product features and feeding them into learning-to-rank models for the optimization of different product retrieval objectives. Wu et al. [61] manually extract multiple statistic features from product search logs and construct an ensemble tree model to predict user clicks. Wu et al. [62] developed a special ranking loss function that optimizes product search engines by maximizing the revenue generated from online transactions. While they are effective for product search on unstructured free text, it is difficult to extend these methods with structured metadata and their relationships. As far as we know, our work is the first study that jointly models structured and unstructured multi-relational data in a single latent space for product search.

2.2 Explainable System

While there have been a variety of studies on the transparency of complicated systems, such as the interpretability of a machine learning model [27, 31, 42] and the importance of different ranking features [25, 58], the concept of explainability in this article refers to a system's ability on generating human-readable explanations for its results, so users are more likely to conduct certain behavior (e.g., clicks, purchases, etc.) after reading them. Explainability is an important criterion to measure the quality of a production system and has been extensively studied for social and product recommendation. Strategies investigated for explainable systems include straightforward methods based on structured information (e.g., social tags, "people also bought") and complicated methods based on the generated topic aspects from text data [70]. For example, Sharma and Cosley [50] propose to incorporate social network information for music recommendation. They generate recommendation explanations based on the social relationship between users and items. Zhang et al. [74] conducted product recommendation with extracted product features and user opinions from phrase-level sentiment analysis to provide explanations for the recommendation results accordingly. He et al. [23] applied an aspect extraction algorithm [76] on product reviews and modeled the ternary relation between users, items, and aspects to conduct explainable recommendation. Tintarev and Masthoff [56] studied different factors that affect the quality of an explanation and tried to build an evaluation guideline for explainable recommendation. Despite the popularity of explainable systems in recommendation, there are few studies on explaining search results for retrieval systems. To the best of our knowledge, this article is the first work that tries to tackle the problem of explainable search for product retrieval. Also, different from previous studies that attempt to create explanations directly with the recommendation models, we propose a post hoc explanation algorithm that leverages the knowledge representations built by the retrieval model to generate search result explanations.

2.3 Knowledge Embedding

Knowledge embedding refers to a technique that models multi-relational data by constructing latent representations for entities and relationships. The goal is to extract local or global connectivity patterns between entities and use these patterns to understand and generalize the observed relationships between a specific entity and others [12]. Inspired by the success of collaborative filtering in product recommendation [49], most existing methods for multi-relational data are designed based on matrix factorization or related approaches. For instance, Harshman and Lundy [21] introduce a tensor factorization method for relation prediction. Singh and Gordon [52] propose a collective matrix factorization method to improve relationship predictive accuracy by exploiting

information from one relation while predicting another. Nickle et al. [43] propose a three-way model to conduct collective learning on multi-relational data. Liang et al. [34] propose to jointly decompose the user-item interaction matrix and the item-item co-occurrence matrix with shared item embeddings. There is also another line of studies trying to tackle the problem with non-parametric Bayesian frameworks. Miller et al. [41] experiment with a non-parametric Bayesian model for link prediction on social networks. Zhu et al. [77] further improve the framework by introducing a max-margin technology. More recently, the advance of deep learning techniques has led many researchers to explore the effectiveness of knowledge embedding with neural models. Sochet et al. [54] propose a neural tensor network to infer unobserved knowledge base entries based on the embedding of entities. Bordes et al. [13] propose to embed symbolic representations from knowledge bases into a flexible continuous vector space with neural network architectures. Furthermore, Bordes et al. [12] designed a translation-based embedding model (transE) to jointly model entities and relationships within a single latent space. He et al. [22] propose a translation-based recommendation system to predict users' personalized sequential behaviors. Yang et al. [63] use the intermediate information embedding built by existing recommendation systems to analyze user-item relationships and create explanations. Zhang et al. [72] and Ai et al. [1] construct knowledge embedding graphs to generate post hoc explanations for recommendation results. In this work, we follow a similar paradigm with previous work but focus specifically on adapting the existing knowledge embedding techniques for explainable product search.

2.4 Neural Information Retrieval

Neural technology has attracted a lot of attention in the IR community in recent years. There has been a variety of neural models proposed for different information retrieval tasks such as ad hoc retrieval [4], question answering [64], and recommendation [24]. For example, Huang et al. [29] propose to embed tri-grams with neural network and generate rank lists according to the cosine similarity between the aggregated tri-gram embeddings of queries and documents. Vulić and Moens [59] conduct multi-lingual retrieval by representing queries and documents with their averaged word embedding. Guo et al. [18] categorize existing neural retrieval models into two groups—the representation-based models and the interaction-based models. Based on this idea, they propose a deep relevance matching model that explicitly models the interaction patterns between queries and relevant documents. There are also many studies on applying deep learning technology on other complicated retrieval problems, such as user modeling [14, 36] and context-aware relevance modeling [2, 8]. Similar to Van Gysel et al. [57] and Ai et al. [6], we use neural networks, especially the embedding-based neural retrieval framework, as the main technique for the development of product retrieval models. However, as shown in Section 6, we also conduct a systematic comparison of our proposed models and the state-of-the-art non-neural baselines.

3 MODEL DESCRIPTION

We now provide detailed descriptions of the Dynamic Relation Embedding Model (DREM) for explainable product search. DREM jointly models different user/product knowledge and creates a knowledge graph with both static and dynamic relationships. In this section, we first provide an overview of DREM and describe how to conduct product search with it, then we discuss the modeling of static and dynamic entity relationships in detail.

3.1 Overview

As discussed previously, explainable product search requires retrieval systems to be capable of modeling and conducting logical inference with different product information. The relationships between product-related entities usually are complicated and not injective. For example, an item
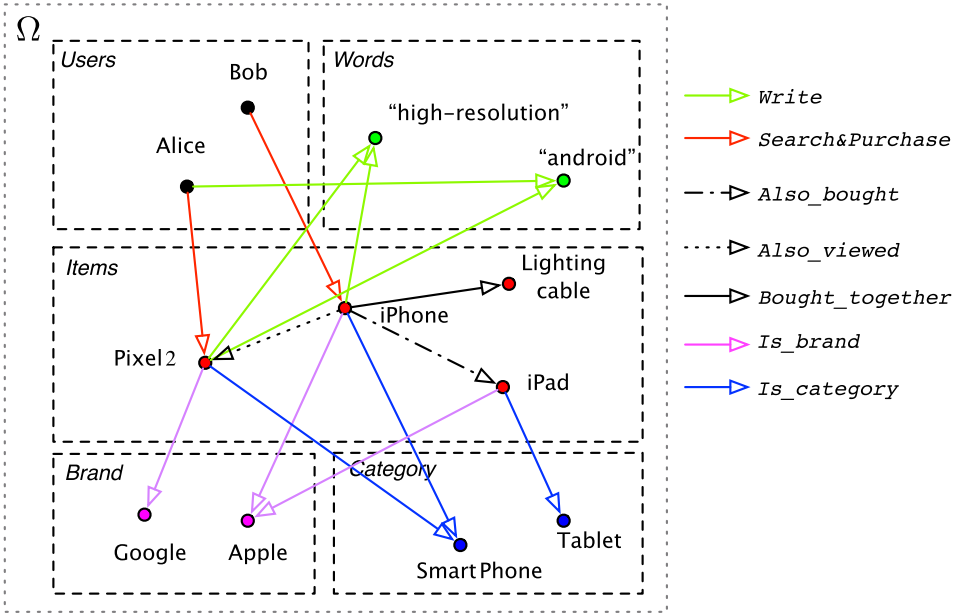
Fig. 1. An example of a knowledge graph created by DREM for product search.

could belong to multiple categories and a category could include multiple items. One of the most popular methods to model such multi-relational data is to construct a *knowledge graph*. In a knowledge graph, each node represents an entity and each edge represents the existence of a certain relationship between two entities. With this design, a knowledge graph satisfies the need of logical inference, because any relationship between an arbitrary pair of entities can be inferred by the path between them. In this article, we design DREM to construct a knowledge graph for product data and conduct explainable product search accordingly.

An example of a knowledge graph created by DREM is shown in Figure 1. In DREM, each user, product, and related entities are represented with vectors in a single latent space Ω. Two entities are linked with an edge if there is a relationship between them. Each edge is labeled with a unique symbol to denote the type of the relationship. To conduct product search with DREM, we create a special edge named as *Search&Purchase* to model the relationship between users and items. In a search session, a user will be *Search&Purchase* with an item if he or she purchases the item. Thus, the problem of product search is to find items that are likely to have the *Search&Purchase* relationship with the users.

Inspired by previous work on relationship prediction [12], we assume that all relationships can be viewed as translations from one entity to another. Suppose that there exists a relationship r between two entities x and y. Let x be the head entity and y be the tail entity; then, we can directly get y by translating x with r as

$$y = x + r.$$

Therefore, any relationship in Ω can be treated as a linear transformation between entities and represented with a latent vector that shares the same dimensionality with Ω. We refer to the latent vectors of entities and their relationships in Ω as *entity embeddings* and *relationship embeddings*, respectively.

The key of DREM is to effectively infer the embedding representations of entities and relationships. Although a variety of methods has been proposed for learning knowledge base

embeddings [12, 13], none of them is applicable to DREM, because the knowledge graph of DREM is not static. Usually, user intent varies in different search sessions. The relationship between users and items cannot be determined without the search context. Therefore, *Search&Purchase* is a dynamic relationship that must be computed for product search on the fly. In the next sections, we describe how to jointly model static and dynamic relationships in DREM.

3.2 Static Relation Modeling

Given the formulation of entities and relationships in the latent space, a generic solution to estimate the embedding parameters is to use an EM-like algorithm [38]. For example, we can iteratively minimize the empirical loss of (x, r) by computing r as the mean of $y - x$ for all entity pairs with r, and computing x as the mean of $y - r$ for all entity pairs with both x and r. Such a method, however, is not appropriate for search problems, because it does not explicitly differentiate entities with and without the relationship. A trivial solution that gives similar representations to all x and y in Ω could still have a low empirical loss of (x, r) in practice. Based on this consideration, we propose to learn DREM by maximizing the posterior probability of observed relationships and minimizing the unobserved ones.

Let r be a static relationship between head entity $x \in X_r$ and tail entity $y \in Y_r$. X_r and Y_r are the sets of all possible entities that are of the same types with x and y, respectively. We refer to r as static, because it holds for x and y universally, regardless of the search context. Inspired by the study of embedding-based generative framework [5, 6, 39], we define the probability of observing tail entity y given head entity x and relationship r as

$$P(y|x,r) = \frac{\exp((\boldsymbol{x} + \boldsymbol{r}) \cdot \boldsymbol{y})}{\sum_{y' \in Y_r} \exp((\boldsymbol{x} + \boldsymbol{r}) \cdot \boldsymbol{y}')}, \tag{1}$$

where $\boldsymbol{r} \in \mathbb{R}^\alpha$, $\boldsymbol{x} \in \mathbb{R}^\alpha$, and $\boldsymbol{y} \in \mathbb{R}^\alpha$ are the embedding representations of r, x, and y with α dimensions. We directly optimize DREM through maximizing the log likelihood of observed $(x, r, y) \in S_{(x,r,y)}$ triples for all static relationships as

$$\mathcal{L}(S_{(x,r,y)}) = \sum_{(x,r,y) \in S_{(x,r,y)}} \log P(y|x,r), \tag{2}$$

where $S_{(x,r,y)}$ is the set of all observed static (x, r, y) triples in the training data. As shown in Equation (1), $P(y|x,r)$ is a softmax function over y, which essentially assumes that $\sum_{y \in Y_r} P(y|x,r) = 1$. Therefore, the maximization of $\mathcal{L}(S_{(x,r,y)})$ will minimize the probability of unobserved (x, r, y).

Optimizing $\mathcal{L}(S_{(x,r,y)})$ directly, however, is prohibitive in practice. The computational complexity of $\mathcal{L}(S_{(x,r,y)})$ is $O(\alpha|S_{(x,r,y)}||Y_r|)$, and the size of $S_{(x,r,y)}$ and Y_r can be large (e.g., there are millions of items in Amazon product datasets). To efficiently train DREM on large-scale data, we adopt a negative sampling strategy to approximate $P(y|x,r)$ in $\mathcal{L}(S_{(x,r,y)})$. Negative sampling was first proposed by Mikolov et al. [40] and has been widely applied in machine learning and information retrieval tasks [4, 6, 32, 40, 71]. The idea of negative sampling is to approximate the denominator of softmax functions by randomly sampling some negative samples from the corpus. Specifically, we sample negative instance y' from Y_r and compute $\log P(y|x,r)$ as

$$\log P(y|x,r) = \log \sigma((\boldsymbol{x} + \boldsymbol{r}) \cdot \boldsymbol{y}) + k \cdot \mathbb{E}_{y' \sim P_r}[\log \sigma(-(\boldsymbol{x} + \boldsymbol{r}) \cdot \boldsymbol{y}')], \tag{3}$$

where k is the number of negative instances, $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and P_r is a noise distribution for $y \in Y_r$.

In fact, as shown by previous studies [33], the optimization of $\mathcal{L}(S_{(x,r,y)})$ with the negative sampling strategy is theoretically principled, as it essentially guides the model to factorizing the

matrix of mutual information between relations and entities. Let $\ell(x, r, y)$ be the expected loss on a specific relation triple $(x, r, y) \in S_{(x,r,y)}$ based on Equation (2) and Equation (3), then we have

$$\ell(x, r, y) = \#(x, r, y) \cdot \log \sigma(\boldsymbol{y} \cdot (\boldsymbol{x} + \boldsymbol{r})) + k \cdot \#(x, r) \cdot P_r(y) \cdot \log \sigma(-\boldsymbol{y} \cdot (\boldsymbol{x} + \boldsymbol{r})), \qquad (4)$$

where $\#(x, r, y)$ and $\#(x, r)$ are the numbers of observed relation triple (x, r, y) and pair (x, r) in $S_{(x,r,y)}$. If we derive the partial gradient of $\ell(x, r, y)$ with respect to $\boldsymbol{y} \cdot (\boldsymbol{x} + \boldsymbol{r})$ and let it be zero, then we can easily get the following result:

$$\boldsymbol{y} \cdot (\boldsymbol{x} + \boldsymbol{r}) = \log \left(\frac{\#(x, r, y)}{\#(x, r)} \cdot \frac{1}{P_r(y)} \right) - \log k. \qquad (5)$$

In this work, we follow the common practice of defining $P_r(y)$ as the normalized frequency of y in all observed (x, r, y) for r, so the right-hand side of Equation (5) is actually a shifted version of pointwise mutual information between (x, r) and y, and optimizing $\mathcal{L}(S_{(x,r,y)})$ with negative sampling is similar to factorizing the mutual information matrix of $(x, r, y) \in S_{(x,r,y)}$.

3.3 Dynamic Relation Modeling

In DREM, we create a relationship between users and products named as *Search&Purchase*. Due to the nature of search tasks, this relationship is dynamic and cannot be determined without the search context. For example, Canon cameras could be linked with users when the query is "digital camera," but not when it is "mobile phone." Therefore, the embeddings of *Search&Purchase* are session-dependent and have to be computed on the fly. For simplicity, we represent the context of a product search session with the query submitted by the user. Note that other session information such as previous queries and clicks [51] can also be incorporated into the framework if needed.

Let q be the query submitted by user u, $\{w_q\}$ be the words of the query, and $\boldsymbol{v}$ be the embedding representation of the relationship *Search&Purchase*. Then, we can compute $\boldsymbol{v}$ with a function of q as

$$\boldsymbol{v} = f(q) = f(\{w_q | w_q \in q\}). \qquad (6)$$

Previous studies have proposed several methods to model search intents with queries in latent space [6, 57, 59, 66]. Ai et al. [6] have explored and compared three options, including averaged word vectors [59], non-linear projections [57], and recurrent neural networks (RNN) [45]. In their experiments, the non-linear projection method usually produces the best and most robust results. Suppose that the latent space of DREM has α dimensions, then the non-linear projection method defines $f(q)$ as

$$f(q) = \tanh \left(W \cdot \frac{\sum_{w_q \in q} \boldsymbol{w_q}}{|q|} + b \right), \qquad (7)$$

where $|q|$ is the length of q, $\boldsymbol{w_q}$ is the embedding of w_q, and $W \in \mathbb{R}^{\alpha \times \alpha}$, $b \in \mathbb{R}^{\alpha}$ are parameters to be learned in the training process. In this work, we employ this non-linear projection function to compute $f(q)$. We tried other query embedding functions [45, 59] as well, but observed no significant performance improvement in our retrieval experiments.

Similar to the modeling of static relationships, we use a softmax function to compute the conditional probability of item i given user u with the dynamic relationship v as:

$$P(i|u, \boldsymbol{v}) = \frac{\exp((\boldsymbol{u} + \boldsymbol{v}) \cdot \boldsymbol{i})}{\sum_{i' \in I} \exp((\boldsymbol{u} + \boldsymbol{v}) \cdot \boldsymbol{i'})}, \qquad (8)$$

where I is the set of all items. Again, we employ the negative sampling strategy to approximate the log likelihood of observed (u, v, i) triples as

$$\log P(i|u, \boldsymbol{v}) = \log \sigma((\boldsymbol{u} + \boldsymbol{v}) \cdot \boldsymbol{i}) + k \cdot \mathbb{E}_{i' \sim P_i}[\log \sigma(-(\boldsymbol{u} + \boldsymbol{v}) \cdot \boldsymbol{i'})], \qquad (9)$$

where P_i is an uniform distribution for $i \in I$. Let $D_{(u,v,i)}$ be the set of observed (u,v,i) triples in the training data, then the final optimization goal of DREM is to maximize the log likelihood of $D_{(u,v,i)}$ and $S_{(x,r,y)}$ as

$$
\begin{aligned}
\mathcal{L} &= \sum_{(u,v,i) \in D_{(u,v,i)}} \log P(i|u,v) + \sum_{(x,r,y) \in S_{(x,r,y)}} \log P(y|x,r) \\
&= \sum_{(u,v,i) \in D_{(u,v,i)}} \log \sigma((\boldsymbol{u}+\boldsymbol{v})\cdot\boldsymbol{i}) + k\cdot\mathbb{E}_{i'\sim P_i}[\log \sigma(-(\boldsymbol{u}+\boldsymbol{v})\cdot\boldsymbol{i'})] \\
&\quad + \sum_{(x,r,y) \in S_{(x,r,y)}} \log \sigma((\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y}) + k\cdot\mathbb{E}_{y'\sim P_r}[\log \sigma(-(\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y'})] \\
&= \sum_{(u,q,i) \in D_{(u,q,i)}} \log \sigma((\boldsymbol{u}+f(q))\cdot\boldsymbol{i}) + k\cdot\mathbb{E}_{i'\sim P_i}[\log \sigma(-(\boldsymbol{u}+f(q))\cdot\boldsymbol{i'})] \\
&\quad + \sum_{(x,r,y) \in S_{(x,r,y)}} \log \sigma((\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y}) + k\cdot\mathbb{E}_{y'\sim P_r}[\log \sigma(-(\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y'})],
\end{aligned}
\tag{10}
$$

where *Search&Purchase* (u,q,i) is the only dynamic relation in $D_{(u,v,i)}$. In DREM, $\boldsymbol{u}$, $\boldsymbol{i}$, and embeddings for all other entities are jointly learned with the parameters W and b in Equation (7). To conduct product search for a specific user u with query q, we simply rank products $i \in I$ with their estimated purchase probability $P(i|u, \boldsymbol{v})$.

Empirically, the weight of static and dynamic relationships do not need to be equal in the model optimization. To explicitly control their relative importance in the final entity representations, we add a hyper-parameter λ in Equation (10) as

$$
\begin{aligned}
\mathcal{L} &= \lambda \sum_{(u,q,i) \in D_{(u,q,i)}} \log \sigma((\boldsymbol{u}+f(q))\cdot\boldsymbol{i}) + k\cdot\mathbb{E}_{i'\sim P_i}[\log \sigma(-(\boldsymbol{u}+f(q))\cdot\boldsymbol{i'})] \\
&\quad + (1-\lambda) \sum_{(x,r,y) \in S_{(x,r,y)}} \log \sigma((\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y}) + k\cdot\mathbb{E}_{y'\sim P_r}[\log \sigma(-(\boldsymbol{x}+\boldsymbol{r})\cdot\boldsymbol{y'})].
\end{aligned}
\tag{11}
$$

For simplicity, we assign equal weights for all relationships in most cases ($\lambda = 0.5$), but we discuss the results of DREM with respect to different λ in Section 6.1.3. Also, in this article, we assume that all users and items have appeared in $D_{(u,v,i)}$ or $S_{(x,r,y)}$ at least once. We leave the exploration of cold-start product search for future studies.

3.4 Time Complexity

The construction of DREM includes two phases: the offline training of entity/relation embeddings and the online testing on unobserved user-query pairs. The time complexity in the training phase mainly depends on the dimensionality of embedding vectors and the size of training data. For each static relationship triple, the computation of local loss (Equation (3)) is $O(k\alpha)$, where k is the number of negative samples and α is the size of each embedding vector. For each dynamic relationship triple, the computation of the relation embedding $\boldsymbol{v}$ (Equation (7)) is $O((|q|+\alpha)\alpha)$, and the computation of local loss (Equation (9)) is $O((|q|+\alpha+k)\alpha)$. Thus, the time complexity of training DREM in one epoch is $O((\mathbb{E}_q[|q|]+\alpha+k)\alpha|D_{(u,v,i)}| + k\alpha|S_{(x,r,y)}|)$, where $\mathbb{E}_q[|q|]$ is the average number of words in each query, and $|D_{(u,v,i)}|$ and $|S_{(x,r,y)}|$ is the number of observed dynamic and static relation triples, respectively. Because k and α are hyper-parameters, the computation cost of DREM is linear to the size of the training data, which is considered to be efficient in general.

For online testing, each item must be assigned with a score to generate the ranked list for a given user-query pair. As discussed in Section 3.3, we rank items according to the estimated purchase
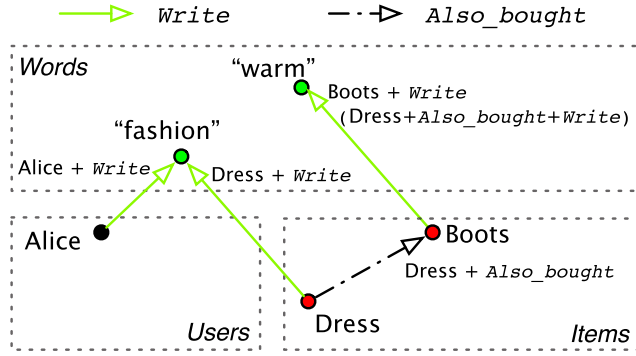
Fig. 2. An example explanation path from user *Alice* to item *Dress* through word "fashion" in DREM.

probability $P(i|u, \boldsymbol{v})$ in Equation (8). Because $\exp(x)$ is a monotone increasing function and the denominator of the softmax function is equal for all items, we can directly rank items based on the dot product between $\boldsymbol{i}$ and $\boldsymbol{u} + \boldsymbol{v}$, which has $O((|q| + \alpha)\alpha)$ complexity. Because we only need to compute $\boldsymbol{v}$ once per query, the computation cost for each testing user-query pair is $O((|q| + \alpha + |I|)\alpha)$, where $|I|$ is the total number of items in the product collection. Since $|I|$ is much larger than $|q| + \alpha$, the overall complexity is approximately $O(|I|\alpha)$. To further improve the efficiency, one can reduce $|I|$ by adding additional retrieval phases to filter out irrelevant documents before applying DREM. We leave these for future studies.

4 EXPLANATION EXTRACTION

An important advantage of DREM is its support for explainable product search. With the knowledge graph, we can directly infer entity relationships and provide explanations of why retrieved items should be interesting to the users. In this section, we discuss how to construct explanation paths in DREM and extract possible explanations for search results in product search.

4.1 Explanation Path

We formulate the problem of explaining why item i is retrieved for user u as finding an explanation path between i and u in the knowledge graph. Figure 2 shows an example search session where we retrieve a dress for user *Alice*. As shown in the figure, both the dress and *Alice* are linked with the word "fashion" by the relationship *Write* in the knowledge graph. Based on this observation, we can say that "we retrieve this dress for *Alice* because she often writes about *fashion* in her reviews and *fashion* is frequently used to describe the dress by other users."

Formally, let Ω_x^r and Ω_y^r be the subspaces of Ω for the head and tail entities of relationship r, respectively. Then, we define an explanation path from u to i as two lists of relationships $\{r_u^k\}$ (size n) and $\{r_i^j\}$ (size m) that connect u and i:

$$\boldsymbol{u} + \sum_{k=1}^{n} \boldsymbol{r}_u^k = \boldsymbol{e} = \boldsymbol{i} + \sum_{j=1}^{m} \boldsymbol{r}_i^j, \tag{12}$$

where the head entity space of r_u^1 is same to the entity space of user u ($\Omega_x^{r_u^1} = \Omega_u$), the head entity space of r_i^1 is same to the entity space of item i ($\Omega_x^{r_i^1} = \Omega_i$), the tail entity space of r_u^n is same to the tail entity space of r_i^m ($\Omega_y^{r_u^n} = \Omega_y^{r_i^m}$), and $\Omega_y^{r_u^{k-1}} = \Omega_x^{r_u^k}$ ($k \in [2, n]$), $\Omega_y^{r_i^{j-1}} = \Omega_x^{r_i^j}$ ($j \in [2, m]$). The relationship r_u^k and r_i^j can either be an identity relationship Φ (which projects an entity to itself)

or any relationship in the observed data $S_{(x, r, y)}$ and $D_{(u, v, i)}$. Here, e is an entity in $\Omega_y^{r_i^m}$ ($\Omega_y^{r_u^n}$) that links u and i with $\{r_u^k\}$ and $\{r_i^j\}$. Given this explanation path, we can generate a search explanation as "we retrieve item i for user u, because u has relationships $\{r_u^k\}$ with e, and i is also linked with e through $\{r_i^j\}$." Therefore, the key of explainable product search is the finding of $\{r_u^k\}$ and $\{r_i^j\}$ given the user u and the retrieved item i.

4.2 Extraction Algorithm

Finding an explanation path, however, is difficult for an arbitrary (u, i) pair. Because we only observe a limited number of relationship triples in the training data, the knowledge graph built on product data usually is sparse [6, 69, 71] . In most cases, it is impossible to find two sets of relationships $\{r_u^k\}$ and $\{r_i^j\}$ that directly link the user u to the item i. To tackle this problem, we propose a Soft Matching Algorithm (SMA) to extract explanation paths in DREM.

Let Ω_e be a subspace of Ω that contains all entities with the type of e, and e_u, e_i be the projections of u and i in Ω_e given particular relation paths, then we define the soft matching score for u and i through $e \in \Omega_e$ as

$$S(e|u, i) = \log(P(e|e_u)P(e|e_i)) = \log P(e|e_u) + \log P(e|e_i), \tag{13}$$

where $P(e|e_u)$ and $P(e|e_i)$ are the probability of observing e given e_u and e_i. Intuitively, $P(e|e_u)$ and $P(e|e_i)$ can be modeled with any functions that measure the similarity between e, e_u, and e_i. In DREM, a straightforward method to compute $P(e|e_u)$ and $P(e|e_i)$ is to adopt the embedding-based generative framework as described in Equation (1). This, however, ignores the length of the path from u to i, and could potentially favor long and less meaningful search explanations in practice. To explicitly encourage short explanation paths, we add a decay factor β and define $P(e|e_u)$ and $P(e|e_i)$ as

$$P(e|e_u) = \frac{\exp(e_u \cdot e - \beta n)}{\sum_{e' \in \Omega_e} \exp(e_u \cdot e')}, P(e|e_i) = \frac{\exp(e_i \cdot e - \beta m)}{\sum_{e' \in \Omega_e} \exp(e_i \cdot e')}, \tag{14}$$

where β is a hyper-parameter that controls the effect of probability decay, and n, m are the length of path $p_u = \{r_u^k\}$, $p_i = \{r_i^j\}$ that translate u, i to e_u, e_i. In this work, we set β as 1.

A summary of SMA for explanation extraction is shown in Algorithm 1. Let $G = \{\Omega_e, r\}$ be a graph where each node Ω_e denotes a subspace of entity type e, and each edge r denotes a relationship that connects two nodes with weight 1. First, given an arbitrary pair (u, i), we find the shortest path from Ω_u to Ω_e and Ω_i to Ω_e as $p_u[e]$ and $p_i[e]$ with the Dijkstra algorithm [16]. Second, we compute their translations e_u, e_i in Ω_e with $p_u[e]$, $p_i[e]$ and their soft matching scores with $e \in \Omega_e$ using Equations (13) and (14). Finally, we sort entity e from each $\Omega_e \subset \Omega$ with their matching probabilities in descending order, and select the best path $\{p_u[e], e, p_i[e]\}$ to generate search explanations. We manually ignore the path that only contains *Search&Purchase*, because it does not provide any information for search explanation.

Because the time complexity of the Dijkstra algorithm [16] is $O(|R| + |E| \log |E|)$ where $|E|$ is the number of entities (or nodes) in the knowledge graph and $|R|$ is the number of relations (or edges) between the entities, the computational cost of SMA could be large if we want to explore all possible paths in the explanation generation process. Thanks to the decay factor in Equation (14), however, we could ignore the paths with more than a certain number of hoops (e.g., 4 in our experiments) and limit the Dijkstra algorithm [16] to a relatively small graph for efficiency. Also, because search explanations can be extracted separately with the retrieval process, one could apply an asynchronous web service that shows the search results first while waiting for the generation

---

ALGORITHM 1: Soft Matching Algorithm (SMA)

Input: u, i, β, $G = \{\Omega_e, r\}$
Output: $path\_set$, $score\_set$
Procedure `Main()`
1   Initialize $p_u \leftarrow \{\}, p_i \leftarrow \{\}, path\_set \leftarrow \{\}, score\_set \leftarrow \{\}$
2   **for** $\Omega_e \in G$ **do**
3         $p_u[e] = Dijkstra(\Omega_u, \Omega_e, G)$
4         $p_i[e] = Dijkstra(\Omega_i, \Omega_e, G)$
    end
5   **for** $\Omega_e \in G$ **do**
6   **for** $e \in \Omega_e$ **do**
7             $score\_set[e] = S(e|u, i)$ // Equations (13) and (14).
8             $path\_set[e] = \{p_u[e], e, p_i[e]\}$.
        end
    end
9   **return** $path\_set$, $score\_set$
Function `Dijkstra($\Omega_x, \Omega_y, G$)`
10    $p_{xy} \leftarrow$ the shortest path from Ω_x to Ω_y in G;
11   **return** p_{xy};

---

of explanations to avoid hurting user experience. We leave the exploration of how to efficiently generate search result explanations for future studies.

5 EXPERIMENTAL SETUP

In this section, we describe the details of our experiment settings. We conduct experiments with Amazon product datasets and compare our method with state-of-the-art product search systems including both text-based models [46] and latent space models [6, 57].

5.1 Datasets

The Amazon product dataset[3] is a well-known benchmark for product search and recommendation [6, 57, 71]. It contains information for millions of customers, products, and associated metadata including descriptions, reviews, brands, and categories. In our experiments, we used four subsets of the Amazon product data, which are *Electronics*, *Kindle Store*, *CDs & Vinyl*, and *Cell Phones & Accessories*. We use the 5-core data provided by McAuley et al. [37] where each user and product has at least 5 purchases and 5 reviews.

Query Extraction. To the best of our knowledge, no large-scale query log is available on the Amazon dataset. A common paradigm used by previous studies is to extract queries from the category information of each product. Similar to Van Gysel et al. [57], we adopt a two-step process to extract search queries for each user. First, given an arbitrary user and his purchase history, we extract the hierarchical category information of each item with more than two levels. Second, we remove duplicated words and stopwords from a single hierarchy of categories and concatenate the terms to form a topic string. The topic string is then treated as a query submitted by the user that leads to a purchase of the corresponding item. Because users often search for "a producer's name, a brand, or a set of terms which describe the category of the product" in e-shopping [48],

---

Table 1. Statistics for the 5-core Datasets for *Electronics*, *Kindle Store*, *CDs & Vinyl*, and *Cell Phones & Accessories* [37]

	Electronics	Kindle Store	CDs & Vinyl	Cell Phones & Accessories
Corpus				
Vocabulary size	142,922	95,729	202,959	22,493
Number of reviews	1,689,188	982,618	1,097,591	194,439
Number of users	192,403	68,223	75,258	27,879
Number of items	63,001	61,934	64,443	10,429
Number of brands	3,525	1	1,414	955
Number of categories	983	2,523	770	206
Relationships				
Write per user	$777.23_{\pm1748.6}$	$1174.23_{\pm3682.39}$	$1846.88_{\pm7667.51}$	$500.01_{\pm979.78}$
Write per item	$2373.62_{\pm5860.33}$	$1293.47_{\pm1916.72}$	$2156.83_{\pm4024.15}$	$1336.64_{\pm2698.30}$
Also_bought per item	$36.70_{\pm38.56}$	$82.56_{\pm29.92}$	$57.28_{\pm39.22}$	$56.53_{\pm35.82}$
Also_viewed per item	$4.36_{\pm9.44}$	$0.16_{\pm1.66}$	$0.27_{\pm1.86}$	$1.24_{\pm4.29}$
Bought_together per item	$0.59_{\pm0.72}$	$0.00_{\pm0.04}$	$0.68_{\pm0.80}$	$0.81_{\pm0.77}$
Is_brand per item	$0.47_{\pm0.50}$	$0.00_{\pm0.00}$	$0.21_{\pm0.41}$	$0.52_{\pm0.50}$
Is_category per item	$4.39_{\pm0.95}$	$9.85_{\pm2.61}$	$7.25_{\pm3.13}$	$3.49_{\pm1.08}$
Train/Test				
Number of reviews	1,275,432/413,756	720,006/262,612	804,090/293,501	150,048/44,391
Number of queries	904/85	3313/1290	534/160	134/31
Number of user-query pairs	1,204,928/5,505	1,490,349/232,668	1,287,214/45,490	114,177/665
Relevant items per pair	$1.12_{\pm0.48}/1.01_{\pm0.09}$	$1.87_{\pm3.30}/1.48_{\pm1.94}$	$2.57_{\pm6.59}/1.30_{\pm1.19}$	$1.52_{\pm1.13}/1.00_{\pm0.05}$

queries extracted with this paradigm are usually sufficient to simulate real-world product search queries [6, 57].

Entities and Relationships. In this work, we consider five types of entities and their relationships in product search. The entities we used are *user*, *item*, *word*, *brand*, and *category*. We ignore words that have appeared less than five times in the corresponding corpus. Also, we split hierarchical category information of each product into multiple distinct categories and replace each category as a unique symbol in the training data. For example, a two-level category hierarchy *Camera, Photo → Digital Camera Lenses* will be considered as two separate entities and anonymized as foo_1 and foo_2. An item that belongs to this category hierarchy will be connected to both foo_1 and foo_2.

The relationships used in our experiments include

- *Write*: Word w was written by user u in their reviews ($u \rightarrow w$) or written for item i in the item's reviews ($i \rightarrow w$).
- *Also_bought*: Users who purchased item i_1 previously also purchased item i_2 ($i_1 \rightarrow i_2$).
- *Also_viewed*: Users who viewed item i_1 previously also viewed item i_2 ($i_1 \rightarrow i_2$).
- *Bought_together*: Item i_1 was purchased with item i_2 in a single transaction ($i_1 \rightarrow i_2$).
- *Is_brand*: Item i belongs to brand b ($i \rightarrow b$).
- *Is_category*: Item i belongs to category c ($i \rightarrow c$).

The statistics of entities and relationships in the Amazon product datasets are summarized in Table 1. Similar to previous studies [6, 69, 71], the observed relation triples in our data are highly sparse.

5.2 Baselines

To demonstrate the effectiveness of the DREM as a product search model, we incorporate five baselines in our experiments: the language modeling approach for IR [46], a probabilistic retrieval model (BM25) [47], an ensemble learning-to-rank model (LambdaMART) [61], the Latent Semantic Entity model [57], and the Hierarchical Embedding Model [6].

5.2.1 QL. The language modeling approach for IR, which is often referred to as the Query Likelihood model (QL), is first proposed by Ponte and Croft [46]. It is a unigram model that ranks documents based on the posterior probability of observing the query words given a document's language model estimated with maximum likelihood estimation. In this article, we concatenate the title, description, and reviews of an item in the training data to form a document for it and compute its ranking scores with respect to a query q based on the language modeling approach with Dirichlet smoothing [67] as:

$$\log(P(q|d)) = \sum_{w \in q} \#(w, q) \log \frac{\#(w, d) + \mu \frac{\#(w, C)}{|C|}}{|d| + \mu},$$

where $\#(w, q)$, $\#(w, d)$, and $\#(w, C)$ are the frequencies of word w in the query q, document d, and the corpus C, respectively; $|d|$ and $|C|$ are the lengths of d and C; and μ is a hyper-parameter that controls the effect of Dirichlet smoothing.

5.2.2 BM25. Built on the bag-of-words representations of queries and documents, BM25 is a classic probabilistic retrieval model proposed by Robertson and Walker [47]. It assumes a 2-Poisson distribution for observed words in the corpus and ranks documents with a statistical scoring function as

$$BM25(q, d) = \sum_{w \in q} IDF(w, C) \cdot \frac{\#(w, q) \cdot (k_1 + 1)}{\#(w, q) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avg(|d|)}\right)},$$

where $IDF(w, C)$ is the inverse document frequency of word w in the corpus C, $avg(|d|)$ is the average document length, and k_1 and b are two hyper-parameters for the ranking function. Similar to QL, we concatenate the title, description, and reviews in the training data to form a document for each product.

5.2.3 LambdaMART. As a representative study on applying learning-to-rank techniques to product search, Wu et al. [61] construct a LambdaMART model for product search by manually extracting a variety of ranking features for each item with their text data and user behavior logs. In this article, we construct a learning-to-rank baseline with LambdaMART following the same pipeline used by Wu et al. [61]. Due to the limits of Amazon review datasets, we cannot compute certain features such as session features and time features, but we manage to reproduce most global statistic features and query-item features proposed by Wu et al. [61]. Detailed feature descriptions are listed in Table 2.

5.2.4 LSE. The Latent Semantic Entity model (LSE) is the first latent space model proposed for product search by Van Gysel et al. [57]. It encodes queries and n-grams with a non-linear projection function similar to Equation (7). It also learns the embedding representations of items by maximizing the similarity between an item and the encoded n-grams extracted from the corresponding item reviews. Specifically, for each n-gram s in the product review of an item i, the similarity between s and i in LSE is computed as

$$P(i|s) = \sigma(\boldsymbol{i} \cdot f(s)),$$

Table 2. A Summary of the Ranking Features Extracted for Constructing a Learning-to-Rank
Model on the Amazon Product Search Dataset

Global Statistic Features	
Length	The length of product title, descriptions, reviews.
Purchase	The total number of purchases on each item in the training set.
Distinct Purchase	The distinct number of users who have purchased a certain item in the training set.
Query-item Features	
TF	The average term frequency of query terms in product title, descriptions, reviews, and the whole document (title+description+reviews).
IDF	The average inverse document frequency of query terms in product title, descriptions, reviews, and the whole document (title+description+reviews).
TF-IDF	The average value of $tf \cdot idf$ of query terms in product title, descriptions, reviews, and the whole document (title+description+reviews).
BM25	The scores of BM25 [47] on product title, descriptions, reviews, and the whole document (title+description+reviews).
LMABS	The scores of Language Model (LM) [46] with absolute discounting [68] on product titles, descriptions, reviews, and the whole document (title+description+reviews).
LMDIR (QL)	The scores of LM with Dirichlet smoothing [68] (which is same with QL) on product titles, descriptions, reviews, and the whole document (title+description+reviews).
LMJM	The scores of LM with Jelinek-Mercer [68] on product titles, descriptions, reviews, and the whole document (title+description+reviews).

where i is the representation of i in the latent space, $f(x)$ is the non-linear projection function in Equation (7), and $\sigma(x)$ is a sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Products are retrieved based on their similarity with the query in the latent space.

5.2.5 HEM. The Hierarchical Embedding Model (HEM) proposed by Ai et al. [6] is a state-of-the-art retrieval model for personalized product search. It is constructed based on a generative framework that assumes reviews are generated by the language model of users/items and purchases are generated by the joint model of users and queries. Similar to DREM, HEM learns the embeddings of users, items, and queries by maximizing the likelihood of observed review data and ranks items based on their posterior probability given the user and the query. However, HEM only considers the information of users, items, and product reviews, and does not differentiate the relations between different types of entities in the optimization process.

5.3 Evaluation Methodology

To train and test different product search models, we partitioned each dataset into a training set and a test set. Following the methodology used by Ai et al. [6], we randomly hide 30% of the user reviews from the training data and use their corresponding purchase information as the ground truth for testing. We randomly select 30% queries as test queries, and if all queries for an item were selected as test queries, we randomly pick one from the test query set and put it back to the training data. After that, we match the queries with user-item pairs in the test set to construct the final test data. An item is relevant to a user-query pair if and only if it is relevant to the query

and has been purchased by the user. In this setting, all query-user-item triples in the test set are unobserved in the training process. More statistics about our data partitions are shown in Table 1.

To evaluate retrieval performance in our experiments, we adopt three metrics, which include the mean average precision (MAP), the mean reciprocal rank (MRR), and the normalized discounted cumulative gain (NDCG). For each user-query pair, we only retrieve 100 items to generate the rank list. Both MAP and MRR are computed based on the whole rank list, while NDCG is computed based only on the top 10 items. Significant differences are measured by the Fisher randomization test [53] with $p < 0.01$.

5.4 Implementation Details

For QL and BM25, we used galago[4] to index and retrieve items. For LambdaMART, we manually extract features from raw data and build the model with an open-source learning-to-rank package ranklib.[5] And for LSE and HEM, we used the implementation provided by Ai et al. [6].[6] QL, BM25, LSE, and HEM conduct product search based on the text information of items, which is the same as DREM built with the *Write* relationship only. To further analyze the usefulness of other relationships, we tested DREM built on *Write* together with other relationships. We refer to DREM with *Also_bought*, *Also_viewed*, *Bought_together*, *Is_brand*, and *Is_category* as $DREM_{AB}$, $DREM_{AV}$, $DREM_{BT}$, $DREM_{Bnd}$, and $DREM_{Cat}$, respectively. DREM with *Write* only and the DREM with all relationships are referred to as $DREM_{NoMeta}$ and $DREM_{All}$.

The latent space models (LSE, HEM, and DREM) are trained with stochastic gradient descent with batch size 64. We manually clip the norm of batch gradients with 5 to avoid unstable parameter updates. We train each model with 20 epochs and gradually decrease the learning rate from 0.5 to 0 in the training process. For baselines, we tuned the Dirichlet smoothing parameter μ of QL from 1K to 3K, and the BM25 scoring parameter k_1 and b from 0.5 to 4 and 0.25 to 1, respectively. The number of trees and leaf nodes in LambdaMART are set as 1,000 and 10, respectively, and we tuned the personalization weight η of HEM from 0 to 1. We also tuned the embedding size α for LSE, HEM, and DREM from 100 to 500. To better illustrate the importance of different product relationships in different datasets, we fix the dynamic relation weight λ in Equation (11) as 0.5 for most experiments, but we will discuss its effect in Section 6.1.3. We will release our data and code upon the publication of this manuscript.

6 RESULTS AND DISCUSSIONS

In this section, we report the results of our experiments. We first present and discuss the retrieval performance of DREM and baseline models. Then, we provide a case study to analyze the effectiveness of DREM for explainable product search.

6.1 Retrieval Performance

Table 3 summarizes the results of our product search experiments on the four subsets of Amazon product data. We group the models into three groups—the baseline models (QL, BM25, LambdaMART, LSE, HEM); DREM with *Write* and another relationship among *Also_bought*, *Also_viewed*, *Bought_together*, *Is_brand*, and *Is_category* ($DREM_{AB}$, $DREM_{AV}$, $DREM_{BT}$, $DREM_{Bnd}$, $DREM_{Cat}$); and the DREM with *Write* only or with all the relationships ($DREM_{NoMeta}$, $DREM_{All}$)

6.1.1 Overall Results. As we can see from the table, the relative performance of bag-of-words models (QL, BM25) and latent space models without personalization (LSE) varies across different

---

[4]https://sourceforge.net/p/lemur/wiki/Galago/.
[5]https://sourceforge.net/p/lemur/wiki/RankLib/.
[6]https://github.com/QingyaoAi/Hierarchical-Embedding-Model-for-Personalized-Product-Search.

Table 3. Comparison of Baselines and DREM on the Amazon Product Search Datasets

Model	Electronics			Kindle Store		
	MAP	MRR	NDCG	MAP	MRR	NDCG
QL	0.289	0.289	0.316	0.011	0.012	0.013
BM25	0.283	0.280	0.304	0.021	0.013	0.014
LambdaMART	0.180	0.181	0.237	0.028	0.029	0.018
LSE	0.233	0.234	0.239	0.006	0.007	0.007
HEM	0.308^{*+}	0.309^{*+}	0.329^{*+}	0.029	0.035^{*}	0.033^{*}
DREM_{NoMeta}	0.291	0.291	0.319	0.036^{*}	0.044^{*}	0.042^{*}
DREM_{AB}	0.283	0.283	0.312	0.043^{*+}	0.052^{*+}	0.050^{*+}
DREM_{AV}	0.318^{*+}	0.319^{*+}	0.349^{*+}	0.035^{*}	0.043^{*}	0.041^{*}
DREM_{BT}	0.320^{*+}	0.321^{*+}	0.346^{*+}	0.037^{*}	0.045^{*}	0.042^{*}
DREM_{Bnd}	0.314^{*+}	0.315^{*+}	0.340^{*+}	0.037^{*}	0.044^{*}	0.043^{*}
DREM_{Cat}	0.299^{+}	0.300^{+}	0.360^{*+}	0.048^{*+}	0.056^{*+}	0.056^{*+}
DREM_{All}	$\mathbf{0.366^{*+\dagger}}$	$\mathbf{0.367^{*+\dagger}}$	$\mathbf{0.408^{*+\dagger}}$	$\mathbf{0.057^{*+\dagger}}$	$\mathbf{0.067^{*+\dagger}}$	$\mathbf{0.067^{*+\dagger}}$
Model	CDs & Vinyl			Cell Phones & Accessories		
	MAP	MRR	NDCG	MAP	MRR	NDCG
QL	0.009	0.011	0.010	0.081	0.081	0.092
BM25	0.027	0.018	0.016	0.083	0.081	0.115
LambdaMART	0.054^{*+}	0.057^{*+}	0.051^{*+}	0.121	0.121	0.148
LSE	0.018	0.022	0.020	0.098	0.098	0.084
HEM	0.034	0.040	0.040	0.124^{*+}	0.124^{*+}	0.153^{*+}
DREM_{NoMeta}	0.034	0.041	0.040	0.107	0.107	0.127
DREM_{AB}	0.046^{+}	0.054^{+}	0.054^{+}	0.098	0.098	0.120
DREM_{AV}	0.034	0.041	0.040	0.095	0.096	0.096
DREM_{BT}	0.037^{+}	0.044^{+}	0.042^{+}	0.089	0.089	0.096
DREM_{Bnd}	0.035	0.041	0.040	0.134^{*+}	0.134^{*+}	0.152^{+}
DREM_{Cat}	0.059^{*+}	0.068^{*+}	0.070^{*+}	0.193^{*+}	0.193^{*+}	0.229^{*+}
DREM_{All}	$\mathbf{0.074^{*+\dagger}}$	$\mathbf{0.084^{*+\dagger}}$	$\mathbf{0.086^{*+\dagger}}$	$\mathbf{0.249^{*+\dagger}}$	$\mathbf{0.249^{*+\dagger}}$	$\mathbf{0.282^{*+\dagger}}$

$*$, $+$, and $\dagger$ denote significant differences to all baselines (QL, BM25, LambdaMART, LSE, and HEM), DREM_{NoMeta}, and all tested models, respectively, in Fisher randomization test [53] with $p \leq 0.01$. The best performance is highlighted in boldface.

datasets. While QL and BM25 have comparable performance on all datasets, LSE outperformed them on *CDs & Vinyl* but performed worse than them on *Electronics* and *Kindle Store*. As shown by previous studies [18, 19], the main difference between unigram models and latent space models is their ability to conduct semantic matching. The latter performs well when vocabulary mismatch between queries and documents is severe, while the former works better in other cases. Our results indicate that the severity of vocabulary mismatch is low on *Electronics* or *Kindle Store*, but high on *CDs & Vinyl*. By incorporating personalization, HEM consistently outperformed QL and LSE on all the datasets tested in our experiments. Because purchasing is a highly personalized behavior, incorporating user information can help HEM better understand the search intents of each user and retrieve items that suit different individuals. The results for LambdaMART are more complicated. While it achieved superior performance on *CDs & Vinyl*, it also produced bad results on *Electronics*. Further analysis on ranking features is needed to understand why learning-to-rank models perform differently on different product categories, which is beyond the scope of this article, and we will leave it for future studies.

After incorporating other product knowledge information discussed in Section 5.1, DREM$_{All}$ significantly outperformed all baseline models on all datasets. It obtained 19%, 97%, 118%, and 101% improvements with respect to MAP over HEM on *Electronics*, *Kindle Store*, *CDs & Vinyl,* and *Cell Phones & Accessories*, respectively. This demonstrates the usefulness of multi-relational product data and the effectiveness of DREM as a product retrieval model.

In Table 3, the performance of HEM and DREM$_{NoMeta}$ is competitive in most cases. HEM and DREM$_{NoMeta}$ are both constructed based on users, items, and their associated reviews. The only difference between them is the method they used to model entity relationships. HEM directly uses user embeddings to predict both review words and purchased items, while DREM$_{NoMeta}$ uses relationship embeddings to project users into the space of words and items separately. According to our results, the two paradigms are equally effective for product search and neither of them is consistently better than the other. However, DREM is more powerful in terms of extendability, because it creates a knowledge graph that can integrate different kinds of product information for retrieval tasks.

6.1.2 Usefulness of Different Relationships. In our experiments, we analyze the importance of different relationships by training DREMs with each of the relationships separately. As shown in Table 3, the importance of relationships varies considerably on different datasets. On *Electronics*, nearly all types of product knowledge brought benefits to DREM except DREM$_{AB}$, which is built on the *Write* and *Also_bought* relationships. As shown by Ai et al. [6], the importance of personalization for product search is less significant on *Electronics* than on other datasets. Two co-purchased items in *Electronics* are less likely to satisfy the same type of user preference or search intent. For example, users may not intend to buy a keyboard when they search for "mouse," despite the fact that they often buy keyboards before or after the purchase of a mouse. Therefore, the relationship *Also_bought* introduces less information but more noise for DREM on *Electronics*.

In contrast to *Electronics*, the incorporation of *Also_bought* significantly improved the retrieval performance of DREM on *Kindle Store* and *CDs & Vinyl*. DREM$_{AB}$ outperformed DREM$_{NoMeta}$ by 19% and 35% with respect to MAP on *Kindle Store* and *CDs & Vinyl*, respectively. This indicates that co-purchased items often fit the same need of users in *Kindle Store* and *CDs & Vinyl*. This is reasonable, because *Kindle Store* and *CDs & Vinyl* consist of books and music, of which people usually have consistent tastes. If a CD is relevant to a query, then other frequently co-purchased CDs are also likely to be relevant.

As shown in Table 3, we observed that *Is_brand* is more useful for product search on *Cell Phones & Accessories* than on other datasets. On *Cell Phones & Accessories*, DREM$_{Bnd}$ significantly outperformed DREM$_{NoMeta}$ with a 25% improvement on MAP. According to a recent report,[7] most people have high loyalty to the manufacturer of their phones and 56% of people who currently possess a smartphone used to own a phone from the same manufacturer. Thus, it is not surprising to see that *Is_brand* exhibits high correlations with user purchases in *Cell Phones & Accessories*.

Although we have split each hierarchical category into distinct categories and anonymized them in model construction, there might be a concern that incorporating category entities in DREM may hurt the fairness of the evaluation, since the test queries are generated based on the hierarchy of categories. In our experiment, we indeed observed that DREM with *Is_category* (DREM$_{Cat}$) performed better than the DREM with other relationships on *Kindle Store*, *CDs & Vinyl,* and *Cell Phones & Accessories*. However, it is worth noticing that DREMs without *Is_category* also significantly outperformed the state-of-the-art baseline methods. In Table 3, DREM$_{BT}$ on *Electronics*, DREM$_{AB}$ on *Kindle Store*, DREM$_{AB}$ on *CDs & Vinyl,* and DREM$_{Bnd}$ on *Cell Phones & Accessories* obtained 4%,

---

(a) *Electronics*

(b) *Kindle Store*
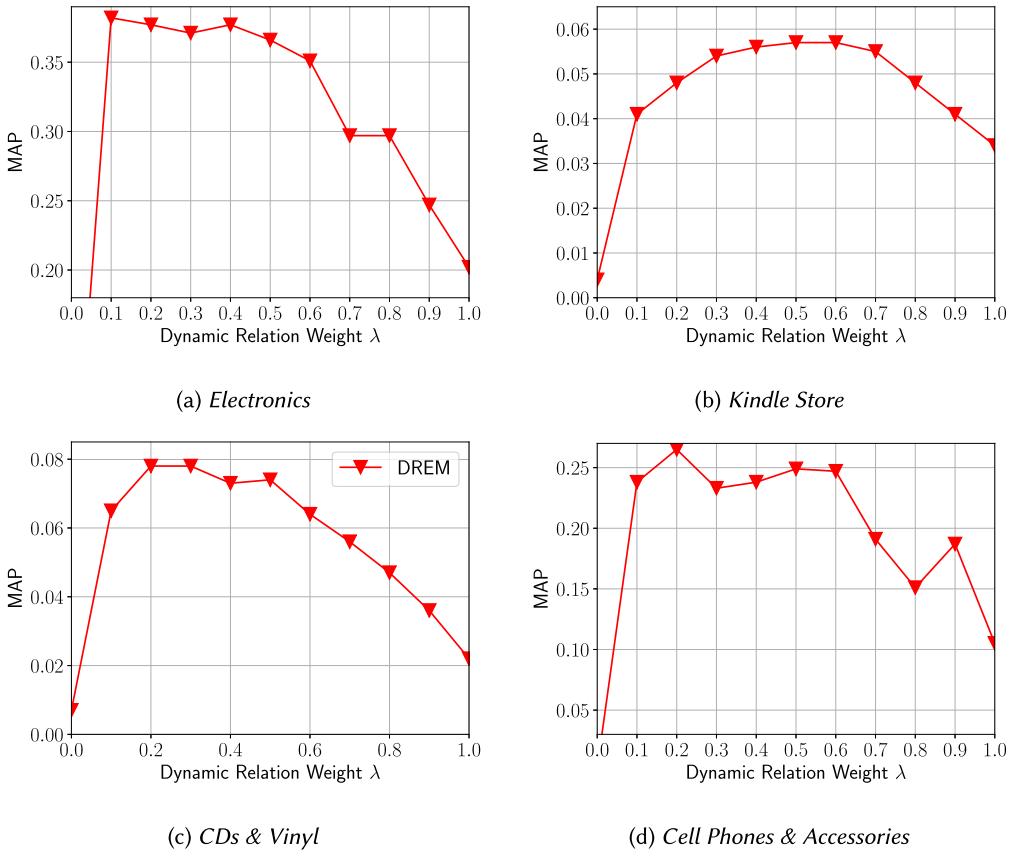
(c) *CDs & Vinyl*

(d) *Cell Phones & Accessories*

Fig. 3. The performance of DREM with different dynamic relation weight λ.

48%, 35%, and 8% improvements on MAP over the best baseline (HEM), respectively. Again, these results indicate the effectiveness of DREM and the usefulness of multi-relational product data for product search.

6.1.3 Parameter Sensitivity. There are two hyper-parameters used in the training of DREM—the dynamic relation weight λ in Equation (11) and the embedding size α. To analyze the parameter sensitivity of DREM, we plot the MAP of DREM$_{All}$ with different parameter settings in Figure 3 and Figure 4.

Figure 3 shows the performance of DREM$_{All}$ on different product categories with respect to the dynamic relation weight λ ranged from 0 to 1. When $\lambda = 0$, DREM$_{All}$ learns nothing on the dynamic relationships, and search queries would have no influence on the final search results, which means that the model will be degraded from a search model to recommendation model. As expected, the retrieval performance of DREM$_{All}$ with $\lambda = 0$ is significantly worse than other models. When $\lambda = 1$, DREM$_{All}$ does not incorporate any information from static relationships. While it performs reasonably well compared to the text-based baseline models such as QL and LSE, it produces inferior performance compared to DREM$_{All}$ with smaller λ. As shown in Figure 3, DREM$_{All}$ usually achieves the best performance when λ is larger than 0.1 but less than 0.7. This demonstrates that both dynamic and static relationship information are valuable for product search.
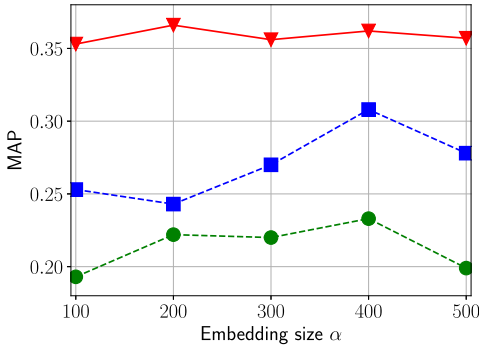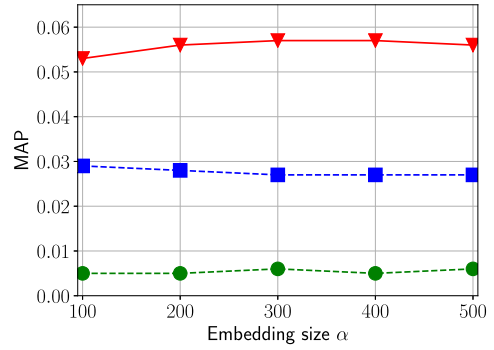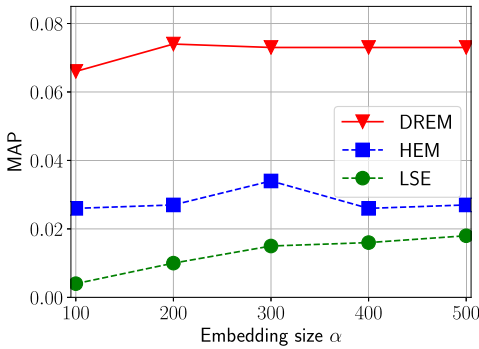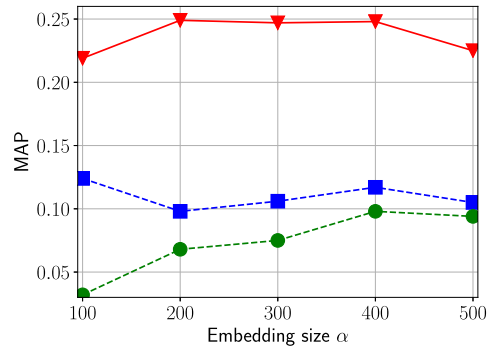
(a) *Electronics*

(b) *Kindle Store*

(c) *CDs & Vinyl*

(d) *Cell Phones & Accessories*

Fig. 4. The performance of DREM and baselines with different embedding size α. The red solid line with triangles represents the numbers for $DREM_{All}$; the green and blue dashed lines with circles and squares are results for LSE and HEM, respectively.

Figure 4 plots the retrieval performance of both baseline methods (LSE and HEM) and $DREM_{All}$. As we can see, the size of embeddings has minor effect on the performance of DREM. $DREM_{All}$ obtained similar results with different α and outperformed LSE and HEM with large margins. Therefore, in practice, we advise to start with a small α and increase it when necessary.

6.2 Case Study

To show the effectiveness of DREM as an explainable product search model, we show an example knowledge graph created by DREM in the experiments and conduct a laboratory study to analyze the performance of its search result explanations.

6.2.1 Example Knowledge Graph and Result Explanation Generation. Figure 5 depicts the knowledge graph created by $DREM_{All}$ on *Cell Phones & Accessories* for query "sports outdoors accessory electronics gadget fitness track." Here, we show the nodes and translations of user "A17V9XL4CWTQ6G," item "B00GOGV314" (*Up 24 Activity Tracker* by *Jawbone*), and item "B00BKEQBI0" (*Pebble Smartwatch* by *Pebble Technology*) in different entity subspace. Each edge in the graph represents a particular type of relationship. Entities are connected with their translations through edges with solid arrows. For clarity, we hide the item-item relationships (*Also_bought*,
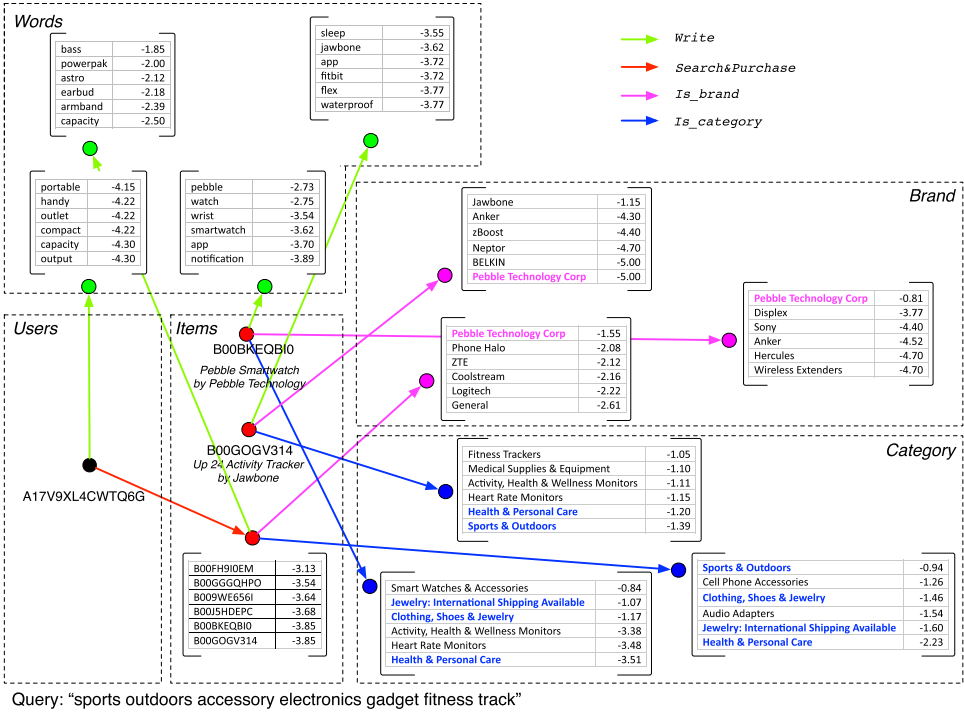
Fig. 5. The knowledge graph created by DREM$_{All}$ on *Cell Phones & Accessories* for query "sports outdoors accessory electronics gadget fitness track." Six top-retrieved entities and the corresponding probabilities (Equation (14)) are shown for each node.

Also_viewed, and *Bought_together*) in the graph. On each node, we show a list of six results selected from the top-retrieved entities with soft matching (Equations (13) and (14)). Entities shared by multiple lists in the same subspace are highlighted with colors. We use u, i_j, i_p to denote the node of the user, *Up 24 Activity Tracker*, and *Pebble Smartwatch*, and use $\vec{SP}$, $\vec{B}$, $\vec{C}$ to denote the relationships of *Search&Purchase*, *Is_brand,* and *Is_category*.

As shown in Figure 5, given the Soft Matching Algorithm, we can find the following explanation paths from user "A17V9XL4CWTQ6G" to *Pebble Smartwatch* "B00BKEQBI0":

- $u + \vec{SP} + \vec{B} \rightarrow$ **Pebble Technology** $\leftarrow i_p + \vec{B}$ with $S(e|u, i) = -2.36$.
- $u + \vec{SP} + \vec{C} \rightarrow$ **Clothing, Shoes, Jewelry** $\leftarrow i_p + \vec{C}$ with $S(e|u, i) = -2.63$.
- $u + \vec{SP} + \vec{C} \rightarrow$ **Jewelry:International Ship** $\leftarrow i_p + \vec{C}$ with $S(e|u, i) = -2.67$.
- $u + \vec{SP} + \vec{C} \rightarrow$ **Health&Personal Care** $\leftarrow i_p + \vec{C}$ with $S(e|u, i) = -5.84$.

With simple templates, we can create four explanations for why the user should be interested in *Pebble Smartwatch*:

- "Based on your profile and query, you may like to see something by *Pebble Technology*, and *Pebble Smartwatch* is a top product of this brand." ($S(e|u, i) = -2.36$)
- "Based on your profile and query, you may like to see something in *Clothing, Shoes, Jewelry*, and *Pebble Smartwatch* is a top product in this category." ($S(e|u, i) = -2.63$)

- "Based on your profile and query, you may like to see something in *Jewelry:International Ship*, and *Pebble Smartwatch by Pebble Technology* is a top product in this category." ($S(e|u,i) = -2.67$)
- "Based on your profile and query, you may like to see something in *Health&Personal Care*, and *Pebble Smartwatch* is a top product in this category." ($S(e|u,i) = -5.84$)

Similarly, for *Up 24 Activity Tracker* "B00GOGV314," we have the following explanation paths that connect it to the search user "A17V9XL4CWTQ6G":

- $u + \vec{SP} + \vec{C} \rightarrow$ **Sports&Outdoors** $\leftarrow i_j + \vec{C}$ with $S(e|u,i) = -2.33$:
  "Based on your profile and query, you may like to see something in *Sports&Outdoors*, and *Up 24 Activity Tracker* is a top product in this category."
- $u + \vec{SP} + \vec{C} \rightarrow$ **Health&PersonalCare** $\leftarrow i_j + \vec{C}$ with $S(e|u,i) = -3.43$:
  "Based on your profile and query, you may like to see something in *Health&Personal Care*, and *Up 24 Activity Tracker* is a top product in this category."
- $u + \vec{SP} + \vec{B} \rightarrow$ **PebbleTechnology** $\leftarrow i_j + \vec{B}$ with $S(e|u,i) = -5.81$
  "Based on your profile and query, you may like to see something by *Pebble Technology*, which is a top brand related to *Up 24 Activity Tracker by Jawbone*."

Given more information on the query and corresponding products, we find that most explanations above are actually reasonable. According to the query, the user is looking for electronic fitness trackers. *Pebble Technology* is a company famous for its fitness tracking devices, while *Pebble Smartwatch* is one of its bestsellers. Also, when the user is searching for fitness trackers within the domain of *Cell Phones & Accessories*, it is likely that he or she is interested in wearable devices with health tracking functions. *Pebble Smartwatch* is a wearable device well known for its multi-functionality and stylish design, while *Up 24 Activity Tracker* is one of its competitors that focuses on health tracking functions and has a cheaper price. It is reasonable to recommend the former based on its popularity in *Clothing, Shoes, Jewelry*, while recommend the latter based on its relationship with *Health&Personal Care*. In fact, the query word "fitness" is more related to *Health&Personal Care*, and the user purchased the *Up 24 Activity Tracker* in the end.

6.2.2 Laboratory User Study. Due to the limit of our experimental environment, we cannot access the original users of the Amazon product dataset to evaluate DREM in terms of explanation generation. Instead, we conduct a laboratory study and recruit graduate students to analyze the performance of result explanations created by DREM. For each of our experimental datasets (i.e., *Electronics, Kindle Store, CDs & Vinyl,* and *Cell Phones & Accessories*), we randomly sampled 50 test search sessions and the top-three explanations generated by the DREM$_{All}$ for the first item retrieved in each session. Because our annotators are not the original users who conducted the search, it is neither reasonable nor reliable to let them judge whether the sampled result explanations could satisfy the personal need of each user. Thus, in our experiments, we focus on evaluating whether the generated result explanations can (1) provide more information for the users and (2) attract people to purchase the item in general. Specifically, we follow the methodology proposed by Wang et al. [60] and create three questions for the annotations of search result explanations in our laboratory study:

- Q1: *Item Information.* After reading the search result explanations, have you learned more about the item we retrieved?
- Q2: *Query Information.* After reading the search result explanations, have you learned more about the query and what the relevant items may look like?

Table 4. The Performance of Search Result Explanations Generated
by DREM in the Laboratory User Study

Dataset	Q1: Item information	Q2: Query information	Q3: Usefulness
Electronics	$1.08_{\pm 0.58}$	$0.65_{\pm 0.68}$	$0.48_{\pm 0.69}$
Kindle Store	$0.88_{\pm 0.60}$	$0.26_{\pm 0.46}$	$0.40_{\pm 0.54}$
CDs & Vinyl	$0.93_{\pm 0.55}$	$0.42_{\pm 0.53}$	$0.44_{\pm 0.57}$
Cell Phones & Accessories	$0.85_{\pm 0.62}$	$0.27_{\pm 0.47}$	$0.19_{\pm 0.46}$

Q1, *Q2*, and *Q3* refer to the questions of whether the explanation provides more information about the item, whether the explanation provides more information about the query, and whether the explanation is useful in persuading the user to purchase the item. Each question is answered with three-level annotations: 0 (*irrelevant*), 1 (*fair*), and 2 (*excellent*). For each question and each dataset, we show both the mean and the variance of their evaluation scores.

- Q3: *Usefulness*. Overall, do you think people would be more likely to purchase the item after they read the search result explanations?

Annotators are asked to do a 3-level judgment for each question: 0 for *irrelevant*, 1 for *fair*, and 2 for *excellent*. In total, we have 200 annotated sessions in which each session has been judged by at least two graduate students with master's or Ph.D. degrees in Computer Science. To the best of our knowledge, DREM is the first model that can generate search result explanations for product search, so we only conduct the laboratory study on DREM in this article.

Table 4 shows the overall results of our laboratory study on DREM. As we can see, the average scores of DREM in Q1 (*item information*) are approximately 1 on all datasets, which means that the result explanations provided by DREM usually provide relevant information about the retrieved items. The average scores of DREM in Q2 (*query information*), however, are relatively low comparing to the scores of Q1. In DREM, we create result explanations by finding paths between users and items, which are modeled as entities, on a knowledge graph where queries are simply treated as a dynamic relationship. This approach may impose an emphasis on item modeling and make the model more likely to retrieve relevant information about the item rather than the query. As for Q3 (*usefulness*), DREM achieved a score of 0.48 on *Electronics*, 0.40 on *Kindle Store*, 0.44 on *CDs & Vinyl*, and 0.19 on *Cell Phones & Accessories*. While these scores are far from perfect, they indicate that the result explanations provided by DREM are useful in persuading people to purchase the corresponding items in general.

To understand the relationship between different evaluation metrics, we further analyze the distribution of the DREM's scores in Q1, Q2, and Q3. Figure 6 shows the average Q1 and Q2 scores of DREM in sessions with different explanation usefulness (i.e., the Q3 scores). Specifically, we discretize the Q3 scores by labeling the sessions with $[0, 0.5)$ as "0" (*irrelevant*), $[0.5, 1.5)$ as "1" (*fair*), and $[1.5, 2]$ as "2" (*excellent*). The columns of "2" in *Electronics*, *Kindle Store,* and *CDs & Vinyl* are missing, because there is no session with *excellent* result explanations in these datasets. As we can see, there is a correlation between both the scores of Q1, Q3 and the scores of Q2 and Q3. Useful result explanations (i.e., Q3 scores "1" or "2") usually provide more information on the retrieved item (i.e., higher Q1 scores) and the search query (i.e., higher Q2 scores) than unuseful explanations (i.e., Q3 scores "0"). Figure 7 depicts the percentage of annotated sessions with different <Q1, Q2> pairs with respect to Q3 scores. Here, we apply the same discretization strategy to Q1/Q2 scores and treat *fair* or *excellent* explanations as positive instances (i.e., $Q1^+$ and $Q2^+$) and *irrelevant* explanations as negative instances (i.e., $Q1^-$ and $Q2^-$). As shown in Figure 7, useful result explanations (i.e., Q3 scores equal or greater than 1) almost always provide relevant item information (i.e., $Q1^+$) or query information (i.e., $Q2^+$). This supports the hypothesis that result explanations can be useful only when additional information is provided that helps the

(a) *Electronics*

(b) *Kindle Store*

(c) *CDs & Vinyl*
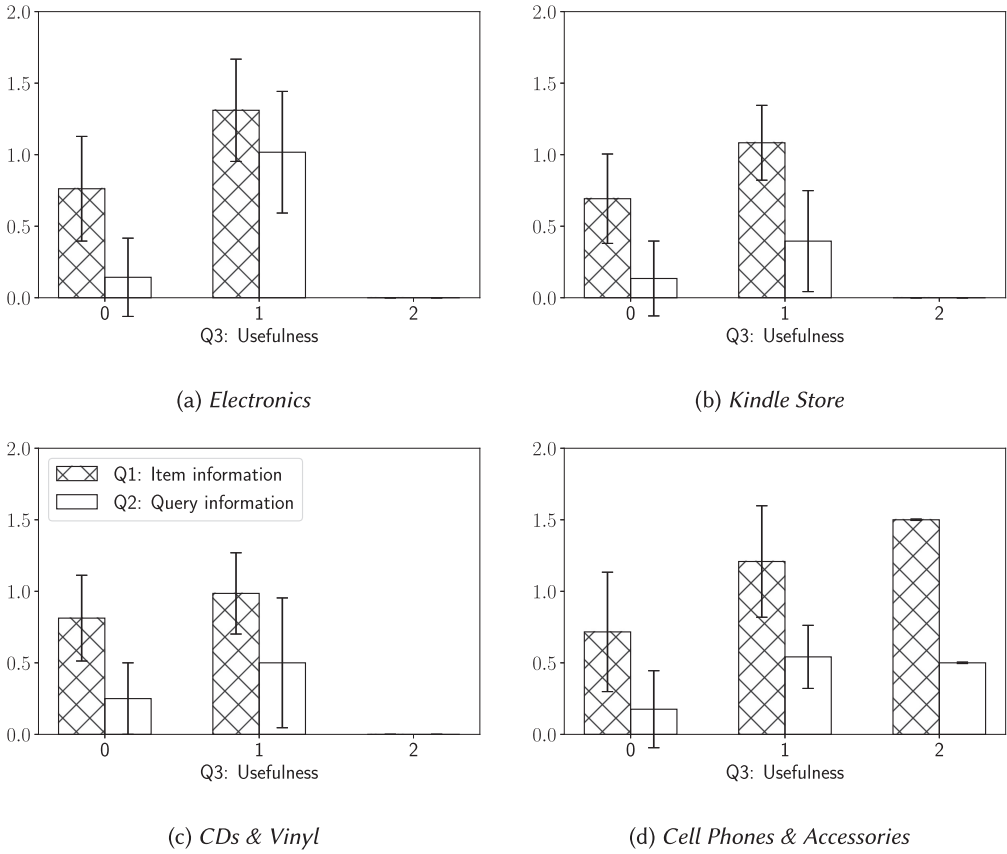
(d) *Cell Phones & Accessories*

Fig. 6. The average Q1/Q2 scores with respect to different usefulness (i.e., Q3 scores) for the result explanations created by DREM in the laboratory user study. The usefulness of the explanations in each session are discretized as "0" (*irrelevant*), "1" (*fair*), and "2" (*excellent*).

queries or the retrieved items. Also, even when their usefulness in persuading people to purchase the items is not good (i.e., Q3 scores are 0), the percentage of result explanations with <Q1⁻, Q2⁻> is lower than 16% in all datasets. This indicates that the result explanations created by DREM can provide relevant information about the items or the queries in most cases.

In our experiments, we have not observed significant correlation between the retrieval performance and the result explanation usefulness on different datasets, but there are some interesting comments provided by the annotators. First, the annotators find that different entity relationships could have different value for the generation of result explanations in each dataset. For example, the relationships of *Also_viewed* and *Also_bought* tend to be less useful in *Kindle Store* and *CDs & Vinyl,* because, when the annotators are not familiar with the topic of the retrieved item, they are also not familiar with the items that are viewed or bought together with the retrieved item. Also, the annotators note that the relevance of retrieved items could affect their judgment process. In our laboratory study, we only sample the result explanations for the first retrieved items in each sampled search session. In many cases, however, the first retrieved item of DREM is irrelevant to the query or not purchased by the user. The annotators find it particularly annoying when the result explanations are relevant while the actual retrieved items are not. This raises an interesting question of whether search result explanations should be created according to how

(a) *Electronics*

(b) *Kindle Store*

(c) *CDs & Vinyl*
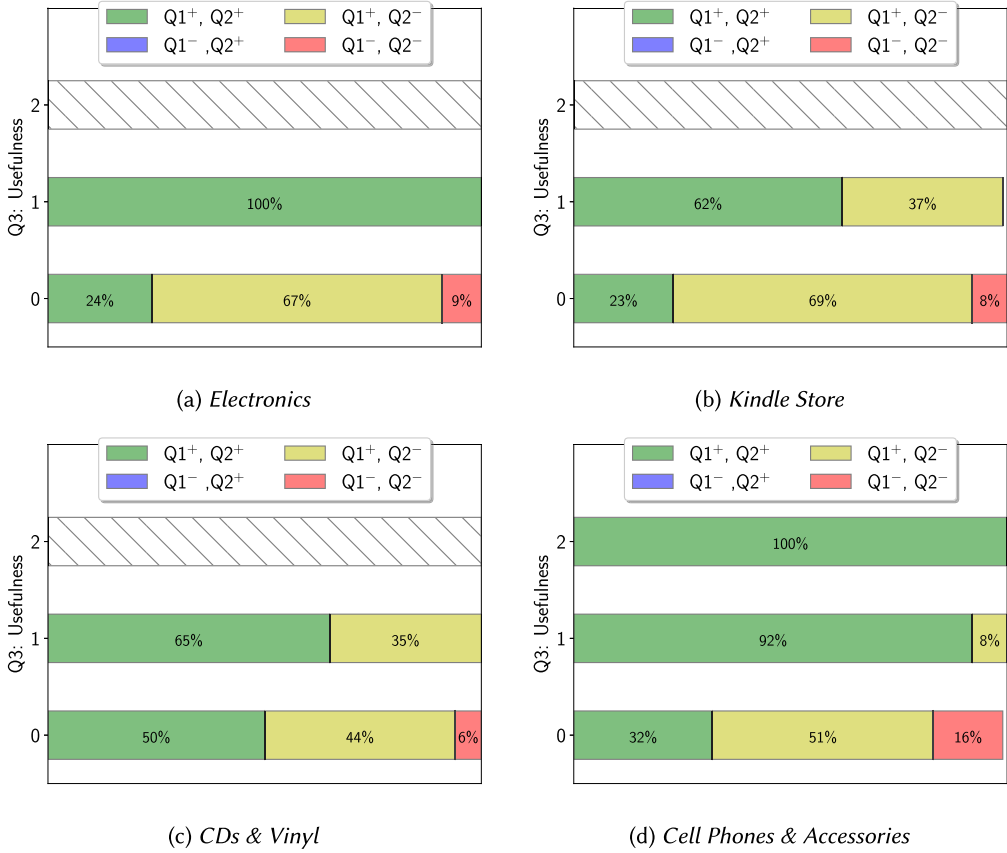
(d) *Cell Phones & Accessories*

Fig. 7. The distribution of sessions with different Q1 and Q2 scores with respect to explanation usefulness (i.e., Q3 scores) in the laboratory user study. The cases where Q1/Q2 scores are larger than 0.5 are treated as positive (i.e., $Q1^+/Q2^+$), while the cases where Q1/Q2 scores are less than 0.5 are treated as negative (i.e., $Q1^-/Q2^-$).

much we believe that the retrieved items are relevant to the users. We leave these topics for future studies.

7 CONCLUSION AND FUTURE WORK

In this article, we present our initial attempt to tackle the problem of explainable product search. We propose a Dynamic Relation Embedding Model that jointly learns embedding representations for entities/relationships and creates session-dependent knowledge graphs. Empirical experiments show that our approach significantly outperforms the state-of-the-art product retrieval methods and has the ability to produce reasonable explanations for search results. This indicates that the construction of dynamic knowledge graph with multi-relational product data is beneficial for the effectiveness and explainability of product retrieval models.

We believe that the study of explainable retrieval models is in an early stage and could be fruitful for product search. In DREM, we generate search explanations based on each explanation path separately. As shown in our laboratory user study, these result explanations are far from perfect. In practice, it may be preferable to organize and combine multiple explanation paths to create a single but more persuasive explanation. Also, in our framework, the explanation path extracted by

the Soft Matching Algorithm is scored by heuristic functions (Equations (13) and (14)), which are empirically effective but not theoretically principled. How to unify product retrieval and search explanation in terms of the model design is still an open question, and we believe that one promising approach is to combine the power of neural embedding models with rule-based knowledge systems. Last but not least, although it seems intuitive that good result explanations could improve the transaction rate of e-commerce search, their actual effects on online systems are mostly unknown. Understanding the real impact of explainable search systems and developing effective evaluation metrics for result explanations are important research problems for the future of e-commerce search engines. We will explore these topics in future studies.

REFERENCES

[1] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms* 11, 9 (2018), 137.

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 135–144.

[3] Qingyao Ai, Daniel N. Hill, S. V. N. Vishwanathan, and W. Bruce Croft. 2019. A zero attention model for personalized product search. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM.

[4] Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. 2016. Analysis of the paragraph vector model for information retrieval. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR'16)*. ACM, 133–142.

[5] Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. 2016. Improving language estimation with the paragraph vector model for ad hoc retrieval. In *Proceedings of the 39th International ACM SIGIR Conference*. ACM, 869–872.

[6] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W. Bruce Croft. 2017. Learning a hierarchical embedding model for personalized product search. In *Proceedings of the 40th International ACM SIGIR Conference*. ACM, 645–654.

[7] Kamelia Aryafar, Devin Guillory, and Liangjie Hong. 2017. An ensemble-based approach to click-through rate prediction for promoted listings at Etsy. In *Proceedings of the ADKDD'17*. ACM, 10.

[8] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2018. Seq2Slate: Re-ranking and slate optimization with RNNs. Retrieved from *arXiv preprint arXiv:1810.02019* (2018).

[9] Keping Bi, Qingyao Ai, Yongfeng Zhang, and W. Bruce Croft. 2019. Conversational product search based on negative feedback. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM.

[10] Keping Bi, Choon Hui Teo, Yesh Dattatreya, Vijai Mohan, and W. Bruce Croft. 2019. Leverage implicit feedback for context-aware product search. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM.

[11] Mustafa Bilgic and Raymond J. Mooney. 2005. Explaining recommendations: Satisfaction vs. promotion. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'05) Beyond Personalization Workshop*, Vol. 5. 153.

[12] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 2787–2795.

[13] Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'11)*, Vol. 6. 6.

[14] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 531–541.

[15] Henriette Cramer, Vanessa Evers, Satyan Ramlal, Maarten Van Someren, Lloyd Rutledge, Natalia Stash, Lora Aroyo, and Bob Wielinga. 2008. The effects of transparency on trust in and acceptance of a content-based art recommender. *User Mod. User-Adapt. Interact.* 18, 5 (2008), 455.

[16] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Mathe.* 1, 1 (1959), 269–271.

[17] Huizhong Duan, ChengXiang Zhai, Jinxing Cheng, and Abhishek Gattani. 2013. Supporting keyword search in product database: A probabilistic approach. *Proc. VLDB Endow.* 6, 14 (2013), 1786–1797.

[18] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A deep relevance matching model for ad hoc retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 55–64.

[19] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. Semantic matching by non-linear word transportation for information retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 701–710.

[20] Yangyang Guo, Zhiyong Cheng, Liqiang Nie, Xin-Shun Xu, and Mohan Kankanhalli. 2018. Multi-modal preference modeling for product search. In *Proceedings of the ACM Multimedia Conference on Multimedia Conference*. ACM, 1865–1873.

[21] Richard A. Harshman and Margaret E. Lundy. 1994. PARAFAC: Parallel factor analysis. *Comput. Stat. Data Anal.* 18, 1 (1994), 39–72.

[22] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2018. Translation-based recommendation: A scalable method for modeling sequential behavior. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'18)*. 5264–5268.

[23] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*. ACM, 1661–1670.

[24] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[25] Yotam Hechtlinger. 2016. Interpretation of prediction models using the input gradient. Retrieved from *arXiv preprint arXiv:1611.07634* (2016).

[26] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. 2000. Explaining collaborative filtering recommendations. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. ACM, 241–250.

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. Retrieved from *arXiv preprint arXiv:1503.02531* (2015).

[28] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. ACM, New York, NY, 368–377. DOI:https://doi.org/10.1145/3219819.3219846

[29] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management*. ACM, 2333–2338.

[30] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 475–484.

[31] Viktoriya Krakovna and Finale Doshi-Velez. 2016. Increasing the interpretability of recurrent neural networks using hidden Markov models. Retrieved from *arXiv preprint arXiv:1606.05320* (2016).

[32] Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning (ICML'14)*, Vol. 14. 1188–1196.

[33] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 2177–2185.

[34] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M. Blei. 2016. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 59–66.

[35] Soon Chong Johnson Lim, Ying Liu, and Wing Bun Lee. 2010. Multi-facet product information search and retrieval using semantically annotated product family ontology. *Inf. Proc. Manag.* 46, 4 (2010), 479–493.

[36] Pablo Loyola, Chen Liu, and Yu Hirate. 2017. Modeling user session and intent with an attention-based encoder-decoder architecture. In *Proceedings of the 11th ACM Conference on Recommender Systems*. ACM, 147–151.

[37] Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring networks of substitutable and complementary products. In *Proceedings of the 21st ACM SIGKDD*. ACM, 785–794.

[38] Geoffrey McLachlan and Thriyambakam Krishnan. 2007. *The EM Algorithm and Extensions*. Vol. 382. John Wiley & Sons.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. Retrieved from *arXiv preprint arXiv:1301.3781* (2013).

[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 3111–3119.

[41] Kurt Miller, Michael I. Jordan, and Thomas L. Griffiths. 2009. Nonparametric latent feature models for link prediction. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 1276–1284.

[42] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recog.* 65 (2017), 211–222.

[43] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the International Conference on Machine Learning (ICML'11)*, Vol. 11. 809–816.

[44] Petteri Nurmi, Eemil Lagerspetz, Wray Buntine, Patrik Floréen, and Joonas Kukkonen. 2008. Product retrieval for grocery stores. In *Proceedings of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. ACM, 781–782.

[45] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech, Lang. Proc.* 24, 4 (2016), 694–707.

[46] Jay M. Ponte and W. Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*. ACM, 275–281.

[47] Stephen E. Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Springer-Verlag New York, Inc., 232–241.

[48] Jennifer Rowley. 2000. Product search in e-shopping: A review and research propositions. *J. Cons. Mark.* 17, 1 (2000), 20–35.

[49] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*. ACM, 285–295.

[50] Amit Sharma and Dan Cosley. 2013. Do social explanations work?: Studying and modeling the effects of social explanations in recommender systems. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1133–1144.

[51] Xuehua Shen, Bin Tan, and ChengXiang Zhai. 2005. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*. ACM, 43–50.

[52] Ajit P. Singh and Geoffrey J. Gordon. 2008. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 650–658.

[53] Mark D. Smucker, James Allan, and Ben Carterette. 2007. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the 16th ACM International Conference on Information and Knowledge Management*. ACM, 623–632.

[54] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 926–934.

[55] Nava Tintarev and Judith Masthoff. 2007. A survey of explanations in recommender systems. In *Proceedings of the IEEE 23rd International Conference on Data Engineering Workshop*. IEEE, 801–810.

[56] Nava Tintarev and Judith Masthoff. 2011. Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*. Springer, 479–510.

[57] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. 2016. Learning latent vector spaces for product search. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 165–174.

[58] Marina M.-C. Vidovic, Nico Görnitz, Klaus-Robert Müller, and Marius Kloft. 2016. Feature importance measure for non-linear learning algorithms. Retrieved from *arXiv preprint arXiv:1611.07567* (2016).

[59] Ivan Vulić and Marie-Francine Moens. 2015. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'15)*. ACM, 363–372.

[60] Nan Wang, Hongning Wang, Yiling Jia, and Yue Yin. 2018. Explainable recommendation via multi-task learning in opinionated text data. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 165–174.

[61] Chen Wu, Ming Yan, and Luo Si. 2017. Ensemble methods for personalized e-commerce search challenge at CIKM cup 2016. Retrieved from *arXiv preprint arXiv:1708.04479* (2017).

[62] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In *Proceedings of The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 365–374.

[63] Fan Yang, Ninghao Liu, Suhang Wang, and Xia Hu. 2018. Towards interpretation of recommender systems with sorted explanation paths. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'18)*. IEEE, 667–676.

[64] Liu Yang, Qingyao Ai, Jiafeng Guo, and W. Bruce Croft. 2016. aNMM: Ranking short answer texts with attention-based neural matching model. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 287–296.

[65] Jun Yu, Sunil Mohan, Duangmanee Pew Putthividhya, and Weng-Keen Wong. 2014. Latent Dirichlet allocation based diversified retrieval for e-commerce search. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, 463–472.

[66] Hamed Zamani and W. Bruce Croft. 2016. Estimating embedding vectors for queries. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR'16)*. ACM, 123–132.

[67] Chengxiang Zhai and John Lafferty. 2001. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*. ACM, 334–342.

[68] Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.* 22, 2 (2004), 179–214.

[69] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 353–362.

[70] Yongfeng Zhang. 2017. Explainable recommendation: Theory and applications. Retrieved from *arXiv preprint arXiv: 1708.06409* (2017).

[71] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W. Bruce Croft. 2017. Joint representation learning for top-n recommendation with heterogeneous information sources. In *Proceedings of the ACM Conference on Information and Knowledge Management*. ACM, 1449–1458.

[72] Yongfeng Zhang, Qingyao Ai, Xu Chen, and Pengfei Wang. 2018. Learning over knowledge-base embeddings for recommendation. Retrieved from *arXiv preprint arXiv:1803.06540*.

[73] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W. Bruce Croft. 2018. Towards conversational search and recommendation: System ask, user respond. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 177–186.

[74] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 83–92.

[75] Yongfeng Zhang, Jiaxin Mao, and Qingyao Ai. 2019. SIGIR 2019 tutorial on explainable recommendation and search. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1417–1418.

[76] Yongfeng Zhang, Haochen Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. 2014. Do users rate or review?: Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14)*. ACM, 1027–1030.

[77] Jun Zhu, Jiaming Song, and Bei Chen. 2016. Max-margin nonparametric latent feature models for link prediction. Retrieved from *arXiv preprint arXiv:1602.07428* (2016).