Social Network Analysis and Mining (2021) 11:82
https://doi.org/10.1007/s13278-021-00784-6

ORIGINAL ARTICLE

Improving e-commerce product recommendation using semantic context and sequential historical purchases

Mahreen Nasir[1] · C. I. Ezeife[1] · Abdulrauf Gidado[1]

Received: 13 February 2021 / Revised: 27 July 2021 / Accepted: 4 August 2021 / Published online: 2 September 2021

Abstract
Collaborative Filtering (CF)-based recommendation methods suffer from (i) sparsity (have low user–item interactions) and (ii) cold start (an item cannot be recommended if no ratings exist). Systems using clustering and pattern mining (frequent and sequential) with similarity measures between clicks and purchases for next-item recommendation cannot perform well when the matrix is sparse, due to rapid increase in number of items. Additionally, they suffer from: (i) lack of personalization: patterns are not targeted for a specific customer and (ii) lack of semantics among recommended items: they can only recommend items that exist as a result of a matching rule generated from frequent sequential purchase pattern(s).
To better understand users' preferences and to infer the inherent meaning of items, this paper proposes a method to explore semantic associations between items obtained by utilizing item (products') metadata such as title, description and brand based on their semantic context (co-purchased and co-reviewed products). The semantics of these interactions will be obtained through distributional hypothesis, which learns an item's representation by analyzing the context (neighborhood) in which it is used. The idea is that items co-occurring in a context are likely to be semantically similar to each other (e.g., items in a user purchase sequence). The semantics are then integrated into different phases of recommendation process such as (i) preprocessing, to learn associations between items, (ii) candidate generation, while mining sequential patterns and in collaborative filtering to select top-N neighbors and (iii) output (recommendation). Experiments performed on publically available E-commerce data set show that the proposed model performed well and reflected user preferences by recommending semantically similar and sequential products.

Keywords Recommendation systems · Sequential pattern mining · Semantics · Sequential model · Clickstream data · Historical purchases · Collaborative filtering · TF-IDF · Vector space model · Cold start · Sparsity · E-commerce

1 Introduction

In this section, we will discuss about some preliminaries such as mining sequential patterns for recommendation (Sect. 1.1.1) and use of semantics in recommendations systems (Sect. 1.1.2) to lay the foundation of our work.

✉ Mahreen Nasir
nasir11d@uwindsor.ca

C. I. Ezeife
cezeife@uwindsor.ca

Abdulrauf Gidado
gidado@uwindsor.ca

[1] School of Computer Science, University of Windsor, Windsor, ON N9B 3P4, Canada

1.1 Background

Recommendation systems (RSs) facilitate customers' purchase decision by recommending products or services of interest (Aggarwal 2016; Bobadilla et al. 2013; Jannach et al. 2010). Designing a recommender system targeted towards an individual customer's need is crucial for retailers to increase revenue and retain customers' loyalty. Collaborative filtering (CF), a common recommendation technique, takes user–item interaction matrix as input, which represents interactions either explicitly (users' ratings) or implicitly (users' browsing or buying behavior), and outputs top-item recommendations for each target user, by finding similarities among users or items (Ekstrand et al. 2011; Ricci et al. 2011; Schafer et al. 2001, 2007; Su and Khoshgoftaar 2009). The input matrix suffers from (i) sparsity (has low user–item interactions) and (ii) cold start (an item cannot be recommended if

Table 1 Purchase sequence database

SID	Purchase sequences
1	< 21239, (21239, 20655, 21242), (21239, 21242), 21366, (21242, 22246) >
2	< (21239, 21366), 21242, (20655, 21242), (21239, 21377) >
3	< (21377, 22246), (21239, 20655), (21366, 22246), (21242, 20655) >
4	< 21377, 22198, (21239, 22246), 21242, 20655, 21242 >

no ratings exist). Content-based method, on the other hand, generates recommendations based on the content (features) of the item and suffers from content overspecialization (lack of diversity in recommended products) due to the use of specific features only (Adomavicius and Tuzhilin 2011).

Items with which a user interacts (e.g., clicked, rated or purchased) can provide a strong indication of her interests and facilitate in learning a good user profile leading to recommendations that match her interests (Bhatta et al. 2019). However, users' interests and preferences change with time. The timestamp of a user interaction (click or purchase event) is an important attribute, and learning the sequential patterns of user interactions based on the timestamps is useful to: (i) understand the long- and short-term preferences of user and (ii) predict the next items for purchase by users as the time interval between any such interactions provides useful insights about users' behavior. Sequential pattern mining mines frequent or high-utility sequential patterns from a sequential database comprising of historical purchase or click sequences. Systems (ChoiRec12, SuChen15, SainiRec17, HPCRec18, HSPRec19) using clustering and pattern mining (frequent and sequential) with similarity measures between clicks and purchases for next-item recommendation cannot perform well when the matrix is sparse, due to rapid increase in number of items. Additionally, models utilizing sequential pattern mining suffer from: (i) lack of personalization: patterns are not targeted for a specific customer, as they infer decisions based on a global view of sequences and (ii) lack of contextual similarities among recommended items: They can only recommend items that exist as a result of a matching rule generated from frequent sequential purchase pattern(s) based on a minimum support threshold.

1.1.1 Mining sequential patterns in recommendation systems

Sequential pattern mining (Agrawal and Srikant 1995; Ayres et al. 2002; Bhatta et al. 2019; Mabroukeh and Ezeife 2010) mines frequent or high-utility sequential patterns from a sequential database such as historical purchase or click sequence database of customers to learn associations between itemset sequences. The problem of sequential pattern mining can be stated as:

Given (i) a set of sequential records (called sequences) representing a sequential database SDB = $\{s_1, s_2, s_3, \ldots, s_n\}$ with sequence identifiers 1,2,3,….n, where each sequence can be a set of items, (ii) a minimum support threshold called min sup ξ and (iii) a set of k unique items or events I = $\{i_1, i_2, \ldots, i_k\}$, the goal of sequential pattern mining algorithm such as PrefixSpan (Pei et al. 2004) is to mine all frequent itemsets (frequent sequential patterns) from a sequence database having the support count greater than or equal to the user-defined threshold of minimum support. A sequence s is said to be a frequent sequence or a sequential pattern if and only if sup(s) > = minsup (minimum support). Table 1 shows an example of daily purchase sequential database. Each sequence in the database (Table 1) represents a tuple with format < SID, sequence > where SID represents the unique sequence identifier and sequence contains list of item sets with each item set consisting of one or more items. For example, SID 4 shows the customer first purchased item '21377' in a single purchase, then item '22198' was purchased followed by a collective purchase of items (21239, 22246) and then items 21242, 20655 and 21242 were bought in separate purchases.

With purchase sequential database (Table 1) and a minimum support (min_sup = 1), some of the frequent sequential patterns generated using PrefixSpan (Jian Pei et al. 2004) are: < 21239, 21242 >, < 21242, 22246 >, < 20655, 21242 >, < (21366, 22246) >, < 21242, 20655, 21242 >, < (21239, 29655, 21242) >.

So, in this paper, we exploit the effectiveness of sequential associations between users' various interactions by extracting complex sequential patterns of customer purchases and then integrating those learned patterns into the collaborative filtering's user–item rating matrix to address its limitations such as: (i) cold start—when no ratings are available for a user–item and (ii) sparsity—when there is less user-interaction data. The frequent sequential purchase patterns derived from customers' historical data can lead to better next-item recommendations for the target user by capturing users' short- and long-term preferences and can improve the accuracy and diversity of recommendations by finding the sequential relationship between frequently purchased items. Furthermore, we aim to enrich the process of generating frequent sequential patterns by incorporating the semantic knowledge of products extracted from products' metadata.

1.1.2 Using semantics in sequential recommendation

Semantics (meanings) are required to have a deep comprehension of the information conveyed by the textual content and to achieve the goal of improving the quality of user profiles and the effectiveness of intelligent information access platforms. According to de Gemmis et al. 2015, techniques in semantic recommenders can be categorized as: (i) top down and (ii) bottom up. To capture the semantics of target user's information needs, top-down approaches utilize external knowledge sources, through the use of concepts extracted from taxonomies (IS-A hierarchy), dictionaries or ontologies (formal representations of categories, properties and relations between concepts, data and entities) for creating user profiles and interpreting the meaning of items. ***Top-down approaches*** aim to facilitate recommender systems in interpreting documents written in natural language and provide meaningful reasoning by providing knowledge such as linguistic, common sense and cultural backgrounds. They introduce semantics by: (i) mapping the features describing the item with semantic concepts, such as word-sense disambiguation (Semeraro et al. 2007) and entity linking, or (ii) linking the item to a knowledge graph such as ontological knowledge (Middleton et al. 2004), structured or unstructured encyclopedic knowledge, like Wikipedia (Gabrilovich and Markovitch 2009; Semeraro et al. 2009).

Bottom-up approaches (distributional models), on the other hand, interpret the semantics by exploring the syntagmatic and paradigmatic relations between words in high-dimensional vector spaces. Syntagmatic relation is a type of semantic relation between words that co-occur in the same sentence or text (Asher, 1994). For example, "the lion chased the deer," representing syntagmatic relationship between lion and chase or "I liked my new iphone" representing relationship between "iphone" and "liked." Paradigmatic relation is a different type of semantic relation between words that can be substituted with another word in the same categories (Hjørland (2015)), for example, substituting 'cat' with 'dog' or 'coca-cola' with 'coca-cola cherry'. These methods work on the principle of vector space model (Turney and Pantel 2010) in which each term is represented. A term-context matrix is created to learn the vector-space representation of each term by encoding the context (for example, a situation) in which the term is used. A context can be set according to the granularity required by the representation of term. The granularity can be coarse-grained (a whole document) or fine-grained (a paragraph, sentence, a window of words). An example of coarse-grained context could be, a document of customers' review about the quality of beverages and the cutlery served at the restaurant they dined at. A fine-grained context can be the description of items purchased by a customers' over the weekend.

A vector-space representation (called **WordSpace**) is learnt according to terms' usage in contexts. Specifically, terms sharing a similar usage are very close in the word space. Given a WordSpace, a vector-space representation of documents (called DocSpace) is typically built as the centroid vector of word representations, that is, a document vector will be represented as the average vector of the words in the document. Once we obtain the vector representations, it is possible to perform similarity calculations between items according to their semantic representation. A common similarity measure is cosine similarity.

Content is required to extend and improve user modeling (preferences) and more importantly to address the limitations of collaborative filtering systems such as: (i) sparsity—when there is less user-interaction data, (ii) cold start—when no ratings are available for a user–item. Furthermore, while generating candidate item(s) by mining the frequent sequential patterns for next-item recommendation, sequential pattern mining techniques suffer from some drawbacks that include: (i) absence of contextual similarities among recommended products, that is, they recommend items based on a match with the sequential rules generated from sequential patterns and do not consider relationships between items according to their context. For example, if there are two sequential rules for chocolate purchase such as:

a) Ferrero Rocher, Ferrero Rondnoir Rafeallo and
b) Kinder Chocolate Bar, Kinder Surprise Kinder Beuno,

where rule (a) (indicating that a purchase of Ferrero Rocher, Ferrero Rondnoir will most likely lead to the purchase of Rafeallo), so for a new purchase sequence where a customer bought Nutella and Rafeallo, he can be recommended to purchase "Ferrero Collection" as they all belong to same brand and have "similar" characteristics such as their ingredients; however, if there is no rule for the product "Nutella," the customer will not be recommended with a product. Conventional sequential rules fail to capture such semantic relationships between products and will only recommend product that exist as a result of a matching rule with products "Nutella and Rafeallo." Therefore, it cannot recommend products that are similar in semantics as they are based only on the frequent sequential occurrence. i.e., frequency count based on a minimum support threshold. Additionally, they (ii) lack of personalization.

Therefore, *semantics (meanings)* are required to have a deep comprehension of the information conveyed by the textual content and to achieve the goal of improving the quality of user profiles. More specifically, semantics are needed to (i) better understand the associations between items and therefore recommend diverse items and (ii) model user preferences in an effective way.

To better reflect users' preferences and to interpret the meaning of the items, *in this paper*, we explored the effectiveness of utilizing semantic knowledge (meaningful relationships between items) learnt through the use of models based on distributional hypothesis (context in which the items are used such as their co-occurrence in the purchase sequences) such as Prod2vec (Grbovic et al. 2015), Glove (Pennington et al. 2014), Doc2vec (Le and Mikolov, n.d.) and TF-IDF (Salton 1988) methods to learn the semantic relationships between products. This semantic knowledge can then be integrated into different phases of recommendation process such as (i) pre-processing, to learn associations between items, (ii) candidate generation, while mining sequential patterns and in collaborative filtering to select top-N candidates that show semantic and sequential association between items, and (iii) output (recommendation). Thus, the inclusion of semantic knowledge into all phases of recommendation process can address the issues of sparsity and cold start and provide recommendations which are diverse, are similar in context (usage) and better reflect user's long- and short-term interests.

1.2 Contributions

The contributions of this study are to improve the performance of recommendation system in the context of sparse user–item interactions and are divided into feature and procedural contributions:

1.2.1 Feature contribution

The main features proposed in this study are to: (i) extract semantic knowledge of items from items' metadata (title, description and brand) and customers' purchase histories (co-purchased and co-reviewed products), to compute semantic similarities between items, (ii) enrich the process of mining frequent sequential patterns by mining semantic-rich frequent sequential patterns from purchase sequences of customers with semantically similar buying behaviors and incorporating items' semantic knowledge (semantic similarity), (iii) integrating items' semantic and sequential information to enhance the item–item matrix in CF without utilizing item ratings, thereby providing personalized recommendations.

1.2.2 Procedural contribution

To address the above limitations and achieve the research goals, the procedural contributions according to the feature contributions are:

(i) Propose a model which first learns products' semantic representations in the form of multidimensional vectors (embeddings) through various distribution models such as Prod2Vec (Grbovic et al. 2015), Glove (Pennington et al. 2014), Doc2vec (Le and Mikolov 2014), TF-IDF (Salton 1988) and their hybrids by using product ID's and product metadata (title, description and brand) as explained in Sect. 3.2. Next, cosine similarity between the obtained product vectors through these models is used to compute semantic similarity between products as products with similar vectors will be similar in semantics and mapped closely in the vector space (Sect. 3.3.1). This semantic similarity is used to create item–item similarity matrix for CF.

(ii) Next, for each target customer, top-N customers are determined based on semantic similarities between products in their purchase sequences, where each purchase sequence will be represented as an aggregate vector of all products in the sequence (Sect. 3.3.2). This is analogous to representing a sentence as collection of words. In the next phase, a database of these semantically similar purchase sequences is created and frequent sequential patterns using PrefixSpan (Pei et al. 2004) are extracted. During the mining process, we integrate the semantic information of products from the item similarity matrix (obtained in step (i)) along with their support count to prune patterns of products that are below our specified similarity threshold and minimum support threshold (Sect. 3.3.3). This provides us the items that are similar in semantics and purchased in sequential order.

(iii) Then, the item–item matrix is enriched with the semantic and sequential information by the proposed computing score between a pair of items before applying collaborative filtering for recommending top-K personalized products as explained in Sect. 3.4.

Our approach provides a unified structure for generating product recommendations which are semantically rich, sequential and personalized without requiring the use of user–item ratings.

In summary, the main contributions of our work as an extension to the earlier proposed model SEMSRec (Nasir and Ezeife 2020) are:

1. We propose a comprehensive model to learn item (product) semantics by utilizing more product features (metadata) such as product title, description and brands in addition to product id's and then incorporating those obtained semantics during mining frequent sequential purchase patterns and to enrich the item-to-item similarity matrix in collaborative filtering for top-K semantic and sequential recommendation.

2. Extend the process for learning products' semantics by training various distributional models and obtained products' representations by:

a) Individually training Prod2vec (Grbovic et al. 2015) and Glove (Pennington et al. 2014) models using product IDs from customers' purchase sequences sorted according to the time stamp.
b) Utilizing embeddings obtained from Prod2vec (Grbovic et al. 2015) and Glove (Pennington et al. 2014) to create hybrid embeddings for product representation.
c) Individually training TF-IDF and Doc2vec (Le and Mikolov 2014) models using more product features such as title, description and brand from customers purchase sequences and products' metadata, where a document represents the collection of products' title, description and brand purchased by the customer sorted according to the time stamp.
d) Utilizing embeddings obtained from TF-IDF and Doc2vec (Le and Mikolov 2014) to create hybrid embeddings for product representation.

Incorporating the obtained semantics (from step 2) in the process of sequential recommendation (while mining frequent sequential patterns) and to enrich item-to-item similarity matrix of collaborative filtering.
Proposing a score measure based on semantic similarity, confidence and lift measures to determine the relationship between products.
Extensive and more detailed experiments by including more E-commerce datasets such as Online Retail and Amazon data set under various categories (electronics, beauty, books and movies).

1.3 Problem description

This paper emphasis on generating top-K recommendations. Formally stating, given a set of users $U = \{u_1, u_2, u_3, \dots, u_{n\}}$ and a set of items $I = \{i_1, i_2, i_3, \dots, i_n\}$. Each user u has a purchase sequence PS_u representing the association of the user with sequence of some items from I such that $PS_u = (PS_1, PS_2, PS_3, \dots, PS_n)$. For purchase sequences PS_u of all such users, the task of the system is to generate a list of top-K semantically similar and sequential products for each user from the candidate set obtained through finding semantic similarities and sequential relationships between products by: (i) extracting semantic knowledge of items from items' metadata (title, description and brand) and (ii) customers' purchase histories (co-purchased and co-reviewed products).

The rest of this paper is organized as follows: Section 2 summarizes related work about recommendation systems in particular about sequential recommendation and semantic-based recommendation. Section 3 introduces the proposed model in detail along with the architecture. Sections 4 and 5 present the experiments and results along with analysis of the model. Finally, Sect. 6 presents the conclusion and future work.

2 Related work

Recommendation systems have been widely studied in the literature. We have explored related work in three main categories. The first category includes general recommendation methods, which are based on user feedback without any sequential order of user actions. The second group includes sequential recommendation methods based on: (i) sequential pattern mining and first-order Markov chains, which consider the last visited item and (ii) sequential recommender systems based on deep learning, which include various or all previously visited items. The third group involves recommendations by learning semantics (meaningful relationship between items) by using distributional models. In this section, we will discuss them, respectively.

2.1 General recommendation

Traditional recommendation methods include collaborative filtering (Aggarwal 2016), matrix factorization (Rendle et al. 2010) and rule-based approaches (Agrawal and Srikant 1994; Xiao and Ezeife 2018). Recommendations are generated based on using either explicit feedback (ratings, reviews) or implicit feedback (clicks, purchases). Bayesian-personalized ranking (BPR) (Rendle et al. 2010) works on implicit feedback and optimizes the latent factor model by utilizing a pairwise ranking loss in a Bayesian framework. HPCRec18 system (Xiao and Ezeife 2018) enriched the quantity (finding the value for unknown ratings) and quality (finding precise value for already rated items) of user–item rating matrix. To improve the quality of ratings, they used normalized purchased frequency matrix and to predict the ratings for next possible purchase; consequential bond between clicks and purchases in each session was mined. The main limitation was that customers' historical sequential behavior patterns were not integrated into the item-rating matrix and during the mining process of consequential bond. Hence, methods under this category are unable to capture sequential patterns of customers' behavior as they do not take into account the order of actions.

2.2 Sequential recommendation

Given past item interactions (e.g., clicks, views, purchases) of a user, sequential recommendation predicts the future item(s) that are of interest to the user. Some previous works on sequential recommendation using sequential pattern mining (Choi et al. 2012) and explicit sequential association rules based on statistical co-occurrences (Bhatta et al. 2019) have limitations of a huge search space and suffer from suitable threshold settings, which may lead to large number of rules most of which are redundant. The use of sequential pattern mining to explore the sequential relationship between purchase sequences for items recommendation is discussed by Choi et al. (2012). These models are hybrid as they combine collaborative filtering with sequential pattern mining for providing recommendations. According to authors, finding neighbor users not just on the basis of ratings on similar items but also considering the sequential relationship between purchase sequences can lead toward more accurate recommendation.

Some of the recent works by Bhatta et al. (2019); Choi et al. (2012); Saini et al. (2017); Salehi and Nakhai Kamalabadi (2013); Yap et al. (2012) have utilized sequential pattern mining techniques in E-commerce recommendation systems for mining customers' purchase patterns and recommending next items for purchase. The authors (Choi et al. 2012) proposed a hybrid online product recommendation system (HOPE), which integrates collaborative filtering (CF)-based recommendations using implicit rating and sequential pattern mining-based recommendations. The systems use implicit rating information instead of explicit rating information by computing implicit rating information from the transaction dataset. They propose to obtain better recommendation quality by integrating collaborative filtering and sequential pattern analysis of customer purchases each of which considers the rating information of users on items and the associations among items. Yap et al. 2012 state that most of the existing sequential pattern mining methods are not user-specific and propose a novel competence score to overcome such disadvantages. They proposed to assign more weight to sequences that have similar items as the target user (e.g., user's purchasing or viewing the same books as the target user). Saini et al. (2017) proposed a framework to mine sequences in the purchase patterns of customer transactions. Also, they intend to find the time gap between products purchased to help improve the recommendation of next products in sequence. The authors used SPADE algorithm to mine sequence of all products, which are (i) bought regularly, e.g., every month, and (ii) bought one after another in a sequence, e.g., most users find mobile phones and mobile covers as common purchase sequence.

HSPRec19 by Bhatta et al. (2019) mined frequent sequential purchase patterns and augment the item-rating matrix with sequential purchase patterns of customers for next-item recommendation. It mined frequent sequential click and purchase behavior patterns using the consequential bond between click and purchase sequences and then used this quantitatively and qualitatively rich matrix for collaborative filtering to provide better recommendations. The results by Bhatta et al. (2019) have showed significant improvement over the systems without using sequential pattern mining methods such as Kim et al. (2005); Kim and Yum (2011); Su and Chen (2015); Xiao and Ezeife (2018). Therefore, mining sequential patterns from customers' historical transactions can be very beneficial for retailers and can increase revenue and customer satisfaction by recommending tailored products to customers.

Other line of work in sequential recommendation integrated matrix factorization and Markov chains, for next basket recommendation, and proposed factorized personalized Markov chains (FPMC) (Rendle et al. 2010; Bernhard et al., 2016; Brafman et al., 2003; Shani et al. 2005). Recently, great progress has been shown by deep learning, and many new techniques such as recurrent neural networks (RNNs) (Hidasi et al. 2016) and convolutional neural networks (CNNs) (Tang and Wang 2018) have been adapted to sequential recommendation. RNN-based methods embed users' historical interactions into a latent vector to represent their preferences. To improve sequential recommendation by using memory networks, RUM (Chen et al. 2018) is proposed, which explicitly captures sequential patterns at item and feature level. Convolutional sequence embeddings (Caser) (Tang and Wang 2018), a CNN-based method, uses embedding matrix of the L most recent items and applies convolutional operations to capture high-order Markov chains. An approach using attention mechanism is proposed in a self-attention-based sequential model (SASRec) (Kang and McAuley 2018) to capture long-term preferences of users.

2.3 Semantic-based recommendation

Research works have introduced to learn item associations for recommendation by learning their semantics (meanings). According to de Gemmis et al. 2015, techniques in semantic recommenders can be categorized as (i) top down and (ii) bottom up. Top-down semantic approaches (exogenous techniques) use external knowledge sources to incorporate semantics. Word-sense disambiguation algorithms process the textual description and replace keywords with semantic concepts (as synsets from WordNet (Miller 1995)), whereas entity linking algorithms focus on the identification of the entities from the text and then linking those entities to other encyclopedic sources such as Wikipedia to create enriched item profiles (Gabrilovich and Markovitch 2009; Middleton et al. 2004; Semeraro et al. 2007). ***Bottom-up approaches (distributional models),*** on the other hand, interpret the semantics by exploring the syntagmatic and paradigmatic relations between words in high-dimensional vector spaces. ***In summary***, we can utilize the (big) corpora of data to directly learn a semantic vector-space representation of the terms of a language. It facilitates to infer lightweight semantics, which are not formally defined such as those in Wikipedia. Furthermore, they are flexible as items, concepts, entities are represented as vector and similarities can easily be computed between terms or terms and documents. Researchers have introduced various distributional models (Word2Vec) (Mikolov et al. 2013), Prod2Vec (Grbovic et al. 2015), Glove (Pennington et al. 2014), Doc2vec (Le and Mikolov 2014) and count-based model such as TF-IDF.

One of the earliest methods Word2vec (Mikolov et al. 2013) creates word embeddings (dense numeric representations of words which store the contextual (semantic) information in a low-dimensional vector) by using a two-layer neural-network-based distributional model for learning semantic representation of words. By using word embeddings, words that are close in context (meaning) are grouped near to one another in the vector space. The input to Word2vec is a text corpus, and its output is a set of vectors known as feature vectors that represent words in that corpus. However, these features representing those words are not explicitly defined. Each word is represented by a vector of real numbers of dimension d. The context of a word ***w*** within a sentence is the set of ***x*** surrounding words. For example, for the sentence "My new iphone fell out of my pocket and broke its screen" we have the target word "iphone." With a sliding window of size = 1, moving along a sentence, the skipgram model of Word2vec (Mikolov et al. 2013) predicts the probabilities of a word being a context word given a target word "iphone," where context words are those on the left and right of the target word within the sliding window. In this case, with a context, x = 2, for the word "iphone" the context words are {"My", "new", "fell", "out"} that is, two words each toward the left and right of the target word "iphone."

A context–target pair is considered as a new observation (training sample) in the data. For example, the target word "iphone" in the above case produces four training samples as (my, iphone), (new, iphone), (fell, iphone) and (out, iphone). This process is iterated for each word in the sentence. Word2Vec (Mikolov et al. 2013) is a distributional model since it learns a representation such that couples (context, word) appearing together have similar vectors. The algorithm is fed with a corpus, and training examples are created through skip-gram. During the training phase, the model learns the weights (feature vectors) of words. The model aims at maximizing the probability of predicting a context ***C*** given a word *w*.

Recently, Neural Language models (NL) such as Word2vec (Mikolov et al. 2013) have been extended to recommender systems. Prod2vec (Grbovic et al. 2015) uses the terminology from Word2vec (Mikolov et al. 2013) and learns vector representations of products to find similarities between products by considering a purchase sequence as a sentence and the products in the sequence as words. In principle, the products occurring in the same context (neighborhood) are similar in semantics (meaning). The model aims to maximize the objective function over the entire set S of purchase sequences, where products from the same purchase sequence are in random order. Prod2vec models the context of purchase sequences, where products with similar contexts, that is with similar neighboring purchases, will have similar vector representations and are closer to each other. According to the analogy of Word2vec (Mikolov et al. 2013), each *purchase sequence* is a sentence and *product id's* are words, that will be fed to the model for training.

GloVe (Pennington et al. 2014) is an unsupervised learning algorithm for obtaining vector representations for words from their co-occurrence information, i.e., how frequently they appear together in a large text corpora. The idea is that a certain word generally co-occurs more often with one word than another. For example, the word "butter" is more likely to occur with shared subset of words (context) such as "margarine." Given some word w_i occurring in the document and considering the *context window* surrounding w_i, if the window size is n, then these are the subsequent words in that document, i.e., words $w_{i+1} \ldots w_{i+n}$. The model builds a *co-occurrence matrix* X, which is a symmetric word-by-word matrix in which X_{ij} is the number of times word w_j appears inside $w_{i's}$ window among all documents, and then, from that matrix, the model learns the vector representations of words. The main intuition behind the model is that ratios of word–word co-occurrence probabilities encode some form of meaning. For example, consider the co-occurrence probabilities for target words butter and honey with various probe words from the vocabulary. The training objective of GloVe (Pennington et al. 2014) is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.

TF-IDF approach uses keyword matching or vector space model (VSM) for representing items (documents, products, videos, music). More specifically, each item (for example, a document) is represented as an n-dimensional vector of term weights, where each weight shows an association between the document and the term. If $d = \{d_1, d_2, \ldots\ldots, d_N\}$ denote a set of documents and $T = \{t_1, t_2, \ldots\ldots, t_N\}$ be the vocabulary, that is the set of words in the documents where T is obtained by applying some standard natural language processing operations such as tokenization, stop words removal and stemming, then each document d_j is represented as a vector in a n-dimensional vector space, such that $d_j = < w_{1j}, w_{2j}, \ldots, w_{nj} >$ where w_{kj} is the weight for term t_k in document d_j. The next task is then to represent each item (document in this case) in the VSM by weighting the terms and then measuring the similarities between feature vectors of items in order to find items similar to the target. TF-IDF weighting (term frequency-inverse document frequency) is a term

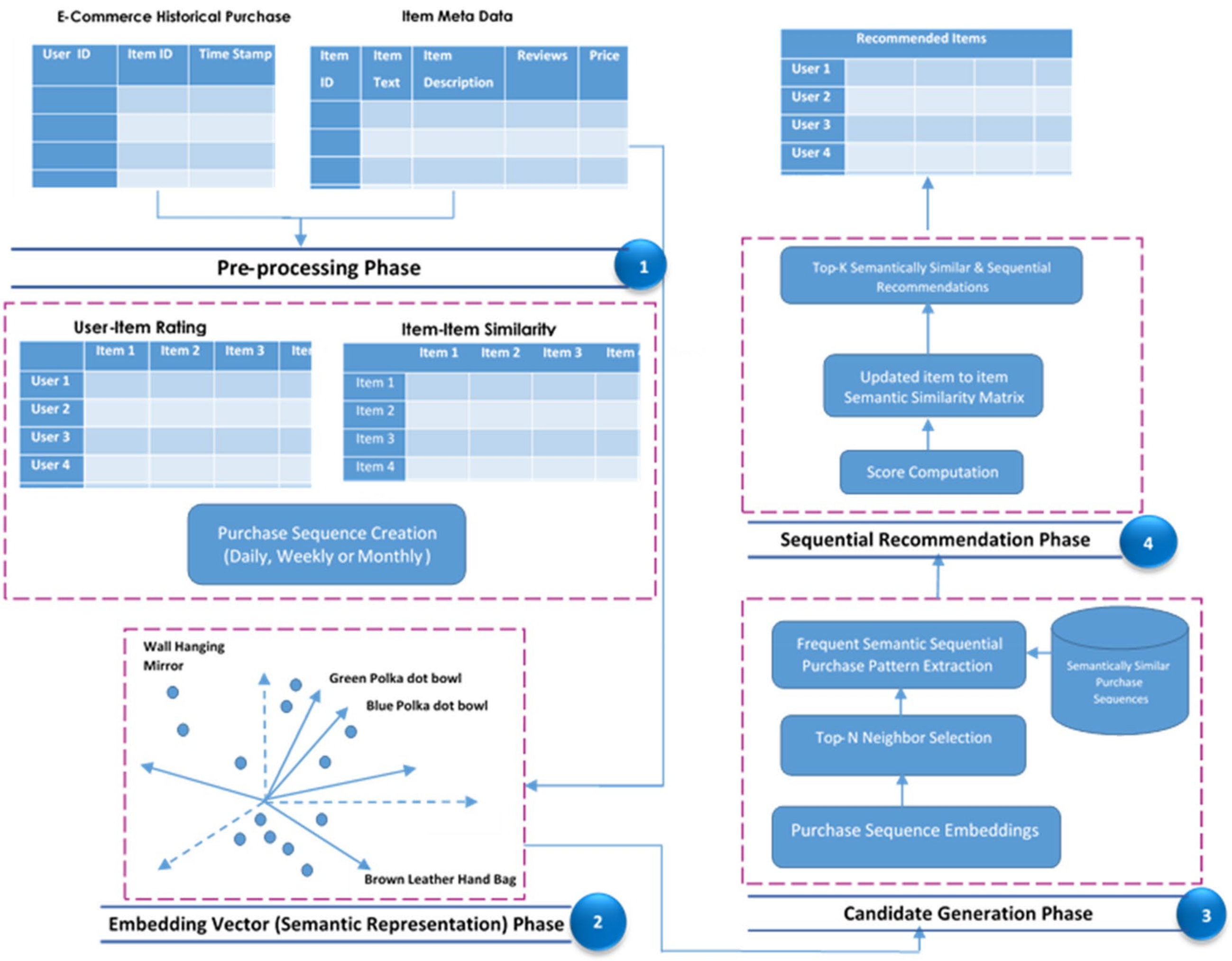
E-Commerce Historical Purchase
User ID
Item ID
Time Stamp
Item Meta Data
Item ID
Item Text
Item Description
Reviews
Price
Pre-processing Phase
1
User-Item Rating
Item 1
Item 2
Item 3
User 1
User 2
User 3
User 4
Item-Item Similarity
Item 1
Item 2
Item 3
Item 4
Purchase Sequence Creation (Daily, Weekly or Monthly)
Wall Hanging Mirror
Green Polka dot bowl
Blue Polka dot bowl
Brown Leather Hand Bag
Embedding Vector (Semantic Representation) Phase
2
Recommended Items
User 1
User 2
User 3
User 4
Top-K Semantically Similar & Sequential Recommendations
Updated item to item Semantic Similarity Matrix
Score Computation
Sequential Recommendation Phase
4
Frequent Semantic Sequential Purchase Pattern Extraction
Semantically Similar Purchase Sequences
Top-N Neighbor Selection
Purchase Sequence Embeddings
Candidate Generation Phase
3

Fig. 1 Architecture of the proposed model (SSHRec)

weighting scheme (Salton 1988) which computes weight of a term as product of TF weight and IDF weight.

SEMSRec (Nasir and Ezeife 2020) was proposed to integrate e-commerce products' semantic and sequential relationships extracted from customers' purchase histories into CF's item similarity matrix to provide semantically similar and sequential recommendations. However, the limitation is that they did not include any item metadata for learning semantics of products.

3 Architecture of the proposed model based on semantic context and sequential historical purchases (SSHRec)

We propose a component-based architecture for our proposed system. Figure 1 presents the diagram depicting the architecture of the system. The system consists of four main phases.

3.1 Data pre-processing

This module is responsible to pre-process the users' (customers') historical purchase data and products' metadata (title, description and brands) for input to the system. The customers' historical data depend on the purchases made by the customers' over a period of time. The historical data need to be cleaned and transformed to obtain it in the correct form. The transformations involve filling missing values, removing duplicate records, sorting and grouping customers' purchase sequences according to the timestamp. Other preprocessing tasks for the product metadata involve natural language processing operations such as: a) tokenization (the process of segmenting text into words, clauses or sentences such as separating words and removing punctuations), b) stop words' removal (removal of commonly used words unlikely to be useful for learning such as a, the, of) and c) stemming which involves reducing related words to a common stem such as reducing the words loved, loving and lovely to the word love obtained from customer's review about a product expressing her likeliness towards the product. These pre-processing operations need to be performed before the data can be input to the models for learning products semantic representations. This is where the feature extraction, construction and selection take place too to get the data model.

3.2 Embedding vector (semantic representation phase)

This phase involves learning the product representations, which are then used in the later phases for computing product similarities, semantically similar purchase sequences, extracting semantically frequent sequential patterns and then incorporating this semantic and sequential information into the item–item similarity matrix in collaborative filtering for generating Top-K recommendations.

The semantics are learned based on:

(a) Product IDs

In this setting, the corpus consists of product ids from customers' purchase sequences sorted according to the time stamp and is used to individually train the Prod2vec (Grbovic et al. 2015) and GloVe (Pennington et al. 2014) models to obtain products' vector representations. Here, for the model training, a corpus of sentences in case of Prod2Vec (where a sentence represents sequence of products purchased by customers sorted according to the time stamp) and documents in case of GloVe (where a document represents collection of sequences representing products purchased by the customer) are used.

In a different setting, vectors obtained after training both models are combined (averaged) to obtain a unified feature vector representation of each product in the corpus.

(b) Product's metadata

To explore the impact of obtaining product semantics from other product features (textual data), a corpus of documents and tokens in case of TF-IDF (where a document represents collection of products' title, description, brand and tokens comprise of unique words present in the textual data) and documents for Doc2vec (Le and Mikolov 2014) are used (where a document represents collection of product descriptions, title and brand in a list of list format and each list element represents description, title and brand of a product purchased, and a document ID for each document).

In a different setting, vectors obtained after training both models are combined (averaged) to obtain a unified feature vector representation of each product in the corpus.

Prod2vec (Grbovic et al. 2015) is based on Word2vec, which is a highly scalable predictive model for learning word embeddings from text. It is based on the distributional hypothesis, which states that words that appear in the same contexts are close to each other in meanings. A similar hypothesis can be applied in larger contexts such as online shopping where we can treat products as word tokens and use user sequences (analogical to sentences) to learn product embeddings. Word2Vec encodes semantics of the words, which is exactly what we need for similar products; therefore, Prod2vec (Grbovic et al. 2015) is used to generate product embeddings. However, the product embeddings generated by Prod2vec (Grbovic et al. 2015) only take into account the information of the user purchase sequence, that is, only the local co-occurrence information. Glove model (Pennington et al. 2014) on the other hand is a word vector representation method where training is performed on aggregated global word–word co-occurrence statistics from the corpus (set of all purchase sequences). Therefore to capture information from the purchase sequences at the local and the global level, product embeddings were obtained by training both models (Prod2vec and Glove) individually and then unifying the learnt embeddings for better representation of product vectors. Experiments showed that unified embeddings gave better results in terms of finding similar products.

We further enhanced the products' representations by including other types of item information, which is the items' metadata (e.g., product titles, descriptions and brand), and used Doc2vec (Le and Mikolov 2014) and TF-IDF because Prod2vec (Grbovic et al. 2015) and (Pennington et al. 2014) do not take into account these types of information such as the items' metadata. In Doc2vec model (Le and Mikolov 2014), the words (products) and the paragraph (products' metadata in a purchase sequence) are trained jointly to learn product vector representations (embeddings).

The rationale to obtain product embeddings through aggregating two models (e.g., Prod2vec and Glove) on product sequences was to capture information about the products (embeddings) from the purchase sequences at the local and the global level.

Next, we present the details of obtaining product representations based on product Ids and the metadata using the Prod2vec (Grbovic et al. 2015), Glove (Pennington et al. 2014), Doc2vec (Le and Mikolov, n.d.) and TF-IDF (Salton 1988) methods. To explain the input format and the working of the models Prod2Vec (Grbovic et al. 2015), Glove (Pennington et al. 2014),TF-IDF(Salton, 1988) and Doc2vec (Le and Mikolov 2014), we will use data from Table 2 and Table 3 representing sample historical product purchase records of customers and products' metadata. For each of these models, we will obtain a semantic representation of items across d = 100 dimensions, where an item can

Table 2 Sample historical product purchase records

Invoice no.	Stock code	Invoice date	Customer ID
536365	20674	12/1/10 8:26	17850.0
536365	21242	12/1/10 8:26	17850.0
536365	20675	12/1/10 8:26	17850.0
536365	21245	12/1/10 8:28	17850.0
536365	20677	12/1/10 8:28	17850.0
536365	20655	12/1/10 8:30	17850.0
536365	20677	12/1/10 8:30	17850.0

represent documents comprising of products, articles, books, customer reviews or product descriptions.

Example 3.1. *Product's Semantic Representation Using Prod2Vec.*

Step 1: Creating purchase sequences

From Table 2, we create purchase sequences (Table 4) for each customer sorted according to the timestamp. Table 5 then represents the sequences in the form of lists for learning products' vector representations using Prod2Vec.

Table 3 Sample of product metadata

Stock code	Title	Description	Brand
20674	Green polka Dot bowl	Earthenware, largest measures 5.5 inch h × 12 inch l × 11.25 inch hand wash	Tag limited
21242	Red retrospot Plate	These beautiful plates are composed of high-rated heavyweight plastic materials rendering the plates leak-free, soak resistant, cut proof and unbreakable	Silver Spoons
20675	Blue Polka Dot bowl	This polka dot bowl is fun and festive and perfect for that bowl of cereal in the morning or bowl of ice cream in the evening. It is finished in a blue celadon glaze with a sprinkling of matte black polkadots. Dish washer safe	Creative innovations
21245	Green polka Dot plate	Add a splash of color with this bright party detail! Green and White Dots Dessert Plates (8), 7"	Party2u
20677	Pink polka Dot bowl	Earthenware,largest measures 5.5 inch h × 12 inch l × 11.25 inch. Hand wash	Tag limited
20655	Queen of skies Luggage Tag	Suitcase tag made PU material, in front with a protective film. waterproof. Fully Bendable / Flexible Material to Prevent Breaking or Losing Your leather luggage tags	PAGSRAH

Table 4 A purchase sequence database (purchase sequences)

SID	Purchase sequences
1	< 21239, (21239, 20655, 21242), (21239, 21242), (21366), (21242, 22246) >
2	< (21239, 21366), 21242, (20655, 21242), (21239, 21377) >
3	< (21377, 22246), (21239, 20655), (21366, 22246), (21242, 20655) >
4	< 21377, 22198, (21239, 22246), 21242, 20655, 21242 >
5	< (20674), (20674, 21245, 21239), 21242 >
6	< 21238, 21239, 21245 >
7	< (20655, 20674, 20675, 20675), (21242, 21245) >
8	< 20675, 21238, 21245 >
9	< 21366, 21239, 21242, 21245 >

Table 5 Lists of purchase sequences

SID	Sequence(s) as lists
S1	[21239, 21239, 20655, 21242, 21239, 21242, 21366, 21242, 22246]
S2	[21239, 21366, 21242, 20655, 21242, 21239, 21377]
S3	[21377, 22246, 21239, 20655, 21366, 22246, 21242, 20655]
S4	[21377, 22198, 21239, 22246, 21242, 20655, 21242]
S5	[20674, 20674, 21245, 21239, 21242]
S6	[21238, 21239, 21245]
S7	[20655, 20674, 20675, 20675, 21242, 21245]
S8	[20675, 21238, 21245]
S9	[21366, 21239, 21242, 21245]

$$PV_{p2vec} = \begin{bmatrix} & F0 & F1 & F2 & \dots & \dots & \dots & F97 & F98 & F99 \\ 20674 & 0.66072 & 0.176294 & -0.577179 & \vdots & \vdots & \vdots & 0.985326 & -0.113320 & 0.120507 \\ 21242 & 0.135658 & 0.208497 & -0.108157 & \vdots & \vdots & \vdots & 0.728440 & 0.023664 & 0.430904 \\ 20675 & 0.344165 & -0.012509 & -0.293663 & \vdots & \vdots & \vdots & 0.786582 & 0.182488 & 0.388695 \\ 21245 & 0.016946 & -0.051390 & -0.172432 & \vdots & \vdots & \vdots & 0.728440 & 0.023664 & 0.430904 \\ 20677 & 0.363226 & 0.076637 & -0.299294 & \vdots & \vdots & \vdots & 0.908400 & 0.081133 & 0.245601 \\ 20655 & -0.110575 & 0.226335 & -0.199941 & \vdots & \vdots & \vdots & 0.183031 & -0.334734 & 0.558633 \end{bmatrix}$$

Fig. 2 Product vectors by Prod2Vec model

Problem 1: Given the input purchase sequence ['20674', '21242', '20675', '21245', '20677', '20655'] represented as a sentence with words (products) as p1 = 20674, p2 = 21242, p3 = 20675, p4 = 21245, p5 = 20677, p6 = 20655. Consider the product '20675' as center product (word), the goal is to train the model to predict the neighboring (context) products which are ['20674', '21242', '21245', '20677', '20655'] by learning vector representations. The model works as explained below.

Input: Sentences (purchase sequences), input layer size–[1x V], input hidden weight matrix X–[V x N], dimension of embedding vector (hidden layer)–N, hidden-output weight matrix Y–[N x V], output layer size–C [1 x V].

Output: Vector representation of all products in V across N dimensions. The created sentences (purchased sequences) will be fed to the model for learning product vector representations.

Prod2Vec Algorithm Summary: Prod2Vec accepts product purchase sequences (e.g., as shown in Table 5) with some other input data listed above. It creates a product (word) vocabulary in the format of {index: unique product id}. Then, each product is represented in a one-hot vector format with dimension V where V is the vocabulary size and the position of the product is represented by a '1' in the vector. Next, a product of one-hot vector is taken with the input weight matrix X to get an embedded vector for the product, which is then multiplied with the output weight matrix Y to get embedded vectors for the context products (e.g., as shown in Fig. 2). For more step-by-step details on obtaining product vectors using Prod2Vec, the reader is encouraged to read conference version of paper (Nasir, M. & Ezeife, C. I., 2020). As a result, we obtain the product vector (PV) matrix where each product is represented across d = 100 dimensions (features). Figure 2 shows matrix PV for some sample products using Prod2Vec.

Example 3.2. *Learning Product Vector Representations Using Global Vectors (GloVe).*

To use ***Glove model for our task*****,** we represent a collection of purchase sequences consisting of product IDs sorted according to timestamp as documents and each product ID in the document representing a word. Glove will model the context of purchase sequences, where products with similar contexts that is with similar neighboring purchases will have similar vector representations and are closer to each other.

The steps to learn product vector representations (semantics) by creating a co-occurrence matrix and then the vector representations using Glove are explained below:

Problem 2: Given an input of purchase sequences (document) in a list of list format as [['20674', '21242', '20675', '21245', '20677', '20655'], ['20675', '21245', '20655'], ['21245', '20674', '20675']] where a document consists of a collection of customers' purchase sequences sorted according to the timestamp, each sequence consists of one or more products (words) and a window size of n, and the goal is to train the model to obtain the product co-occurrence matrix and then learn products' vector representations (semantics). Here in our example, we have three purchase sequences where each purchase sequence has a number of products represented by product ID's. The model works as explained below.

Input: Corpus of documents (purchase sequences in list of list format), window size—c.

Intermediates: Co-occurrence matrix M of size (n x n) where n is the number of unique products in the sequences

Output: Vector representation of all unique products in the vocabulary V across N dimensions. The created document (purchased sequences) will be fed to the model for creating co-occurrence matrix and then learning product vector representations.

Glove Algorithm Summary: Glove accepts product purchase sequences (e.g., shown in Table 5) with some other input data listed above. It creates a product (word) vocabulary in the format of {index: unique product id}. Then, it

$$\left[\begin{array}{lllllll} & 20674 & 21242 & 20675 & 21245 & 20677 & 20655 \\ 20674 & 0.0 & 1.0 & 1.5 & 1.33 & 0.25 & 0.20 \\ 21242 & 0.0 & 0.0 & 1.0 & 0.5 & 0.33 & 0.25 \\ 20675 & 0.0 & 0.0 & 0.0 & 2.5 & 0.50 & 0.83 \\ 21245 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.50 \\ 20677 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 20655 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right]$$

Fig. 3 Glove co-occurrence product matrix

collects word (product) co-occurrence statistics to get a co-occurrence matrix (Fig. 3) so that it can compute the product co-occurrence score for each pair of products.

Steps

1) Create a vocabulary of size (V) in the format {index: product id} consisting of all unique products. For example, V = {1: 20674; 2: 21242; 3: 20675; 4: 21245; 5: 20677; 6: 20655}
2) Collect word (product) co-occurrence statistics in a form of word (product) co-occurrence matrix X. Each element X_{ij} of such matrix represents how often product *i* appears in the context of product *j*. This is done by scanning the corpus as: for each product, look for context products within the defined *window_size* -c (here c = 1) after the term. Less weight is given for context products which are more distant from the target product, by using Eq. 1:

$$decay = \frac{1}{offset} \tag{1}$$

For example, in the above co-occurrence matrix, we can see that product "20674" (target) has co-occurred with product "21242" (context) as 1.0 times. This is computed by scanning the corpus and finding the positions where both these products occur together. Here, according to the corpus, they both occur together only in sequence 1 and the product "21242" is in the context window size (offset) of 1, so its co-occurrence score as per Eq. 4. is = 1/1/ = 1.0. In the same way, the co-occurrence between products "20674" and "20677" (context) is computed. As product "20677" co-occurs with product "20674" in only one sequence (first sequence) and at an offset 4, so its co-occurrence score will be = ¼ = 0.25. The co-occurrence score between other products is computed in the same way.

Produce vector values in continuous space for each word in the corpus, which represents how every pair of words i and j co-occur. This is done by using a soft constraint for each word pair of word i and word j, which states that word vectors are learnt such that their dot product (inner product) equals the logarithm of the words' probability of co-occurrence as shown in Eq. 2.

$$w_i^T w_j + b_i + b_j = \log(X_{ij}) \tag{2}$$

Here w_i represents vector for the target product, w_j is vector for the context product, b_i, b_j are scalar biases for the target and the context products. This is achieved by minimizing an objective function J, which evaluates the sum of all squared errors based on the above equation, weighted with a function f as given in Eq.3.

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f\left(X_{ij}\right)\left(w_i^T w_j + b_i + b_j - \log X_{ij}\right)2 \tag{3}$$

where V is the size of the vocabulary. Here f is a weighting function, which helps to prevent learning only from extremely common word pairs (product pairs). The following function (Eq. 4) is used in Glove (Pennington et al. 2014):

$$f\left(X_{ij}\right) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)\alpha, \ ifX_{ij} < XMAX \\ 1, \ otherwise \end{cases} \tag{4}$$

By using the above formulas and the co-occurrence matrix obtained in step 3, we learn the feature vector representations of products as shown in Fig. 4.

Where each row represents feature vector representation of a product across d dimensions. Here, the feature vectors (dimensions) have a size of 100, as given in Fig. 4. This is a model parameter and can be set as required. For example, vector representation of product with id "20674" is the first row in the matrix and represented as [−0.077740 -0.075161 0.075892 …… −0.020273 0.133662 0.034656].

Example 3.3 *Learning Product Vector Representations Using TF-IDF.*

Problem 3: Given a collection of documents d (e.g., books, articles, product descriptions, user reviews) and a set of features (terms) t, learn the document representation (semantics) by finding the association (weight) between the document and the features.

***For our task, the problem is reformulated* as**: Given a collection of documents (where a document represents collection of product descriptions in a list format as ['medium ceramic top storage jar', 'black candelabra t-light holder', 'woodland charlotte bag', 'airline bag vintage jet set brown'] where each element represents description of a product

$$PV_{glove}=\left[\begin{array}{cccccccccc} & F0 & F1 & F2 & \dots & \dots & \dots & F97 & F98 & F99 \\ 20674 & -0.077740 & 0.075161 & 0.075892 & \vdots & \vdots & \vdots & -0.020273 & 0.133662 & 0.034656 \\ 21242 & -0.114833 & 0.061225 & 0.109533 & \vdots & \vdots & \vdots & -0.025963 & 0.173161 & 0.065046 \\ 20675 & -0.123344 & 0.098004 & 0.120447 & \vdots & \vdots & \vdots & -0.022385 & 0.189432 & 0.054446 \\ 21245 & -0.065362 & 0.047590 & 0.059333 & \vdots & \vdots & \vdots & -0.021115 & 0.114979 & 0.041389 \\ 20677 & -0.102480 & 0.087887 & 0.101183 & \vdots & \vdots & \vdots & -0.019029 & 0.167862 & 0.033685 \\ 20655 & -0.000398 & -0.002544 & 0.003422 & \vdots & \vdots & \vdots & -0.001759 & 0.001247 & 0.003912 \end{array}\right]$$

Fig. 4 Product vectors by glove model

$$PV_{tfidf}=\left[\begin{array}{cccccccccc} & F0 & F1 & F2 & \dots & \dots & \dots & F97 & F98 & F99 \\ 20674 & 0.075502 & -0.026424 & -0.091291 & \vdots & \vdots & \vdots & 0.000088 & -0.073289 & -0.055986 \\ 21242 & 0.202582 & -0.226119 & -0.354920 & \vdots & \vdots & \vdots & 0.019935 & -0.028236 & 0.037867 \\ 20675 & 0.080445 & -0.041201 & -0.134280 & \vdots & \vdots & \vdots & 0.014696 & -0.030099 & -0.067652 \\ 21245 & 0.074386 & -0.036098 & -0.061312 & \vdots & \vdots & \vdots & 0.002169 & -0.076671 & 0.041561 \\ 20677 & 0.114951 & -0.034461 & -0.145326 & \vdots & \vdots & \vdots & -0.010915 & -0.047242 & -0.070353 \\ 20655 & 0.000589 & -0.001437 & -0.001532 & \vdots & \vdots & \vdots & -0.002298 & 0.004696 & 0.003122 \end{array}\right]$$

Fig. 5 Product vectors obtained from TF-IDF model

Table 6 Frequency count of unique tokens occurring in the product descriptions

Prod Id	Products	Tokens												
		blue	bowl	dot	green	luggage	pink	plate	polka	queen	red	retrospot	skies	tag
20674	Green polka dot bowl	0	1	1	1	0	0	0	1	0	0	0	0	0
21242	Red retrospot plate	0	0	0	0	0	0	1	0	0	1	1	0	0
20675	Blue polka dot bowl	1	1	1	0	0	0	0	1	0	0	0	0	0
21245	Green polka dot plate	0	0	1	1	0	0	1	1	0	0	0	0	0
20677	Pink polka dot bowl	0	1	1	0	0	1	0	1	0	0	0	0	0
20655	Queen of skies luggage tag	0	0	0	0	1	0	0	0	1	0	0	1	1

purchased and a set of features (where a feature will consist of unique tokens extracted from the product descriptions), find the association (weight) between the products and the features.

Input: d = {d_1, d_2, ……., d_N}.

d= ['medium ceramic top storage jar', 'black candelabra t-light holder', 'woodland charlotte bag', 'airline bag vintage jet set brown']

t = {t_1, t_2,……, t_N}.

t= ['blue', 'bowl', 'dot', 'green', 'luggage', 'pink', 'plate', 'polka', 'queen', 'red', 'retrospot', 'skies', 'tag']

Output: $d_j = <w_{1j}, w_{2j},\dots,w_{nj}>$

TF-IDF Summary: TF-IDF aims to learn the product feature vectors by first creating set of all unique tokens (words) in product descriptions and then computing the term frequency count from the given product descriptions (Table 6 and Table 7). It then computes inverse document frequencies (as shown in Table 8), and finally, the product of term frequency (TF) and inverse document frequency (IDF) is computed to obtain a term-weighted vector representation, i.e., representing each product as an n-dimensional feature vector (Fig. 5, Table 9).

The steps to learn the product feature vectors using TF-IDF are given below:

Table 6 shows frequency count of thirteen terms occurring in the product descriptions. For example, the term "bowl" appears three times among the product descriptions (as indicated by a 1 in each corresponding row where the term appears in a product).

Step 1: Term Frequency Computation (TF)

Table 7 Term frequencies (TF-computation)

Product Id	Products	Tokens												
		blue	bowl	dot	green	luggage	pink	plate	polka	queen	red	retrospot	skies	tag
20674	Green polka dot bowl	0.00	0.25	0.25	0.25	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00
21242	Red retrospot plate	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.33	0.33	0.00	0.00
20675	Blue polka dot bowl	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00
21245	Green polka dot plate	0.00	0.00	0.25	0.25	0.00	0.00	0.25	0.25	0.00	0.00	0.00	0.00	0.00
20677	Pink polka dot bowl	0.00	0.25	0.25	0.00	0.00	0.25	0.00	0.25	0.00	0.00	0.00	0.00	0.00
20655	Queen of skies luggage tag	0.00	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.25	0.00	0.00	0.25	0.25

Table 8 IDF computation

Terms	blue	bowl	dot	green	luggage	pink	plate	polka	Queen	red	retrospot	skies	Tag
IDF	0.78	0.30	0.18	0.48	0.78	0.78	0.48	0.18	0.78	0.78	0.78	0.78	0.78

Table 9 TF-IDF of tokens in product descriptions (product vectors using TF-IDF)

Product Id	Products	Tokens											
		blue	bowl	dot	green	luggage	pink	plate	queen	red	retrospot	skies	tag
20674	Green polka dot bowl	0.00	0.08	0.04	0.12	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00
21242	Red retrospot plate	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.26	0.26	0.00
20675	Blue polka dot bowl	0.19	0.08	0.04	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00
21245	Green polka dot plate	0.00	0.00	0.04	0.12	0.00	0.00	0.12	0.04	0.00	0.00	0.00	0.00
20677	Pink polka dot bowl	0.00	0.08	0.04	0.00	0.00	0.19	0.00	0.04	0.00	0.00	0.00	0.00
20655	Queen of skies luggage tag	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.19	0.00	0.00	0.19

Term frequency (TF) will be computed by using Eq. 9. For example, term frequency for "blue" in product description "green polka dot bowl" is computed as: TF (blue, green polka dot bowl) = 0/3 = 0.
Similarly, the term frequencies of all tokens are computed and shown in Table 7:
Step 2: Inverse document frequency computation (IDF)
IDF for all the terms is computed using the formula (log N/n_k) where N is the total number of product descriptions and n_k represents the number of product descriptions in which the term appears at least once. Table 8 shows the IDF of all terms.
Step 3: TF-IDF computation
The TF-IDF is computed using Eq. 8, that is, taking the product of term frequency of each token in the product description with inverse document frequency of the token in that product description. For example, the TF-IDF of the term blue will be:

TF-IDF (blue, green polka dot bowl) = TF (blue) * IDF (blue, green polka dot bowl) = 0. * 0.78 = 0.
TF-IDF (bowl, green polka dot bowl) = TF (bowl) * IDF (bowl, green polka dot bowl) = 0.25 * 0.30 = 0.08.

After the computation of TF-IDF, we can represent each product as an n-dimensional feature vector, where each dimension represents a feature (token). For our example, we have thirteen features (tokens) and the product "green polka dot bowl" can be represented as term-weighted vector as:
Green polka dot bowl =

[0.00 0.08 0.04 0.12 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.00]

After training the model on all products in the purchase sequence, we reduce the dimension of the features by using singular value decomposition (SVD) and obtain final product vector matrix as shown in Fig. 5.

Example 3.4 Learning product vector representations Using Doc2vec (Le & Mikolov, 2014)

Problem 4: Given a collection of documents d, learn the document vector representation.

For our task, the problem is reformulated as: Given a collection of documents (where a document represents collection of product descriptions, product title and product brand in a list of list format as where each element represents description, title and brand of a product purchased, and a

$$PV_{doc2vec}=\left[\begin{array}{cccccccccc} & F0 & F1 & F2 & \ldots & \ldots & \ldots & F97 & F98 & F99 \\ 20674 & -0.729392 & -0.694623 & 0.360440 & \vdots & \vdots & \vdots & -0.059213 & 0.784198 & -0.709698 \\ 21242 & 0.478108 & 0.810839 & -0.67386 & \vdots & \vdots & \vdots & 0.876485 & 0.128663 & -0.818151 \\ 20675 & 1.560531 & 0.965368 & -1.731812 & \vdots & \vdots & \vdots & 0.182436 & -0.794381 & -0.585287 \\ 21245 & -0.309573 & 0.418730 & 0.247207 & \vdots & \vdots & \vdots & -0.468052 & 0.743325 & 0.427864 \\ 20677 & -0.100401 & 1.024751 & 0.034455 & \vdots & \vdots & \vdots & 0.346302 & -0.044198 & 0.284042 \\ 20655 & 0.685829 & 0.646765 & -0.305982 & \vdots & \vdots & \vdots & 0.438488 & 1.036193 & 1.161657 \end{array}\right]$$

Fig. 6 Product vectors by Doc2vec model

$$PV_{p2vec\&glove}=\left[\begin{array}{cccccccccc} & F0 & F1 & F2 & \ldots & \ldots & \ldots & F97 & F98 & F99 \\ 20674 & 0.005834 & 0.125728 & -0.250644 & \vdots & \vdots & \vdots & 0.482526 & 0.010171 & 0.077582 \\ 21242 & 0.010412 & 0.134861 & 0.000688 & \vdots & \vdots & \vdots & 0.351238 & 0.098413 & 0.247975 \\ 20675 & 0.110410 & 0.042747 & -0.086608 & \vdots & \vdots & \vdots & 0.382098 & 0.185960 & 0.221571 \\ 21245 & 0.024208 & -0.001900 & -0.056549 & \vdots & \vdots & \vdots & 0.483533 & 0.153193 & 0.265428 \\ 20677 & 0.130373 & 0.082262 & -0.099056 & \vdots & \vdots & \vdots & 0.444685 & 0.124498 & 0.139643 \\ 20655 & -0.055486 & 0.111896 & -0.098260 & \vdots & \vdots & \vdots & 0.090636 & -0.166744 & 0.281272 \end{array}\right]$$

Fig. 7 Hybrid product vectors by Prod2vec and Glove models

$$PV_{d2vec\&tfidf}=\left[\begin{array}{cccccccccc} & F0 & F1 & F2 & \ldots & \ldots & \ldots & F97 & F98 & F99 \\ 20674 & -0.034295 & 0.053539 & -0.088612 & \vdots & \vdots & \vdots & -0.024504 & -0.121241 & 0.018026 \\ 21242 & 0.042981 & -0.004462 & -0.270291 & \vdots & \vdots & \vdots & -0.024960 & -0.061913 & 0.104531 \\ 20675 & -0.022603 & 0.041644 & -0.109378 & \vdots & \vdots & \vdots & 0.004292 & -0.068960 & -0.068960 \\ 21245 & -0.021472 & 0.072360 & -0.129097 & \vdots & \vdots & \vdots & -0.064445 & -0.128279 & 0.069797 \\ 20677 & -0.007805 & 0.065704 & -0.117965 & \vdots & \vdots & \vdots & -0.014165 & -0.082256 & 0.027480 \\ 20655 & -0.038842 & 0.030226 & -0.069437 & \vdots & \vdots & \vdots & 0.097315 & 0.031924 & 0.204722 \end{array}\right]$$

Fig. 8 Hybrid product vectors by Doc2vec and TF-IDF models

document ID for each document, learn document representation (product representation).

Input: A collection of documents in a list of list format [[green polka dot bowl earthenware largest measures hand wash tag limited], [red retrospot plate beautiful plates composed high rated heavy weight plastic material render plate leak free soak resistant cut proof unbreakable, silver spoon], [green polka dot plate add a splash of color bright party detail green white dots dessert plates party2u]

Intermediates: word vectors and document Id vectors. The dimensions of the word vector are 1xV (on-hot vector) and 1xC where C represents the total number of documents. Weight matrix W of hidden layer has a dimension of VxN, whereas the weight matrix D of the hidden layer has a dimension of CxN.

Output: Vector representation of each document across N dimensions.

Doc2vec summary and steps: The method for learning document representation is similar to that of Word2vec with the difference that along with the generation of word vector **W** for each word, a document vector **D** is also generated for each document during the training phase. For example, TaggedDocument (words = [‘green’, ‘polka’, ‘dot’, ‘bowl’, ‘earthenware’, ‘largest’, ‘measures’, ‘hand’, ‘wash’, ‘tag’, ‘limited’] tags = ['0']) and in the end of training, a numeric representation of the document (products) is represented as shown in Fig. 6.

We then created two hybrid matrices of product vector representations as (i) hybrid of $PV_{p2vec} and PV_{glove}$) (Fig. 7)

$$M=\left[\begin{array}{cccccccccccc}
 & 20674 & 21242 & 21238 & 21245 & 21239 & 20655 & 20675 & 21366 & 22246 & 21377 & 22198 \\
20674 & 1.00 & 0.84 & 0.88 & 0.80 & 0.81 & 0.93 & 0.98 & 0.86 & 0.97 & 0.87 & 0.33 \\
21242 & 0.84 & 1.00 & 0.91 & 0.86 & 0.78 & 0.02 & 0.87 & 0.21 & 0.21 & 0.11 & 0.12 \\
21238 & 0.88 & 0.91 & 1.00 & 0.96 & 0.98 & 0.95 & 0.94 & 0.94 & 0.90 & 0.94 & 0.98 \\
21245 & 0.80 & 0.86 & 0.96 & 1.00 & 0.78 & 0.35 & 0.86 & 0.21 & 0.18 & 0.21 & 0.34 \\
21239 & 0.81 & 0.78 & 0.98 & 0.77 & 1.00 & 0.10 & 0.89 & 0.23 & 0.11 & 0.18 & 0.37 \\
20655 & 0.93 & 0.02 & 0.95 & 0.35 & 0.10 & 1.00 & 0.45 & 0.15 & 0.43 & 0.23 & 0.23 \\
20675 & 0.98 & 0.87 & 0.94 & 0.86 & 0.89 & 0.45 & 1.00 & 0.47 & 0.16 & 0.31 & 0.44 \\
21366 & 0.86 & 0.21 & 0.94 & 0.21 & 0.23 & 0.15 & 0.47 & 1.00 & 0.16 & 0.23 & 0.42 \\
22246 & 0.97 & 0.21 & 0.90 & 0.18 & 0.11 & 0.43 & 0.16 & 0.16 & 1.00 & 0.23 & 0.32 \\
21377 & 0.87 & 0.11 & 0.94 & 0.21 & 0.18 & 0.23 & 0.31 & 0.23 & 0.23 & 1.00 & 0.28 \\
22198 & 0.33 & 0.12 & 0.98 & 0.34 & 0.37 & 0.23 & 0.44 & 0.42 & 0.32 & 0.28 & 1.00
\end{array}\right]$$

Fig. 9 Item-to-item semantic similarity matrix

and (ii) hybrid of $PV_{tfidf} and PV_{doc2vec}$) (Fig. 8) to better learn product semantics. This is achieved by averaging the embeddings of each product in the respective matrices.

3.3 Candidate Generation Phase

This phase involves generating potential candidates for recommendation and comprises of three steps which are (i) computing products' semantic similarity, (ii) extracting top-N semantically similar neighbors and (iii) mining semantic embedded sequential patterns and rules. Next, we will discuss each of these steps in detail.

3.3.1 Computing Products' Semantic Similarity

Next step is to create the item-to-item semantic similarity matrix (M) to compute products' semantic similarity by applying cosine similarity using Eq.5 on product vectors in the joint PV matrix.

$$\text{Cosine Similarity}(x,y)=\frac{\sum_{i=1}^{n}x_iy_i}{\sqrt{\sum_{i=1}^{n}x_i^2}\sqrt{\sum_{i=1}^{n}y_i^2}} \tag{5}$$

where x_i and y_i represent components of vectors for products x and y, respectively. For example, to compute the similarity of product "20674" (Green Polka Dot Bowl) with product "21239" (Pink Polka Dot Cup) and product "20675" (Blue Polka Dot Bowl), we will take their corresponding product vectors (column) from the hybrid PV matrix and 21239, 20675 compute cosine similarity between them. So, Cosine similarity (20674, 21239) is 0.81 and cosine similarity (20674, 20675) is 0.98, which shows that product 20674 is more close to product 20675 in the vector space than product 21239. Similarity between other products is computed in the same way. Next, populate the item–item similarity matrix M using Eq. 6. Each entry $R_{x,y}$ in the matrix M represents semantic similarity between products x and y in the vector space. Figure 9 shows a sample of matrix M.

$$M_{x,y}=\begin{cases}1, & if x=y\\ \text{Cosine Similarity}(x,y), & otherwise\end{cases} \tag{6}$$

3.3.2 Extracting Top-N Semantically Similar Neighbors

This step involves computing vector representation (semantic information) of purchase sequences by aggregating vectors of each product in the purchase sequence. This is obtained by computing mean of vectors (across N dimensions) of all products in the purchase sequence by utilizing the hybrid matrices created in Sect. 3.1.2. For example, to compute cosine similarity between the target customers' purchase sequence in the test data and other purchase sequences in the train data, consider a sample purchase sequence in test data as ['23077', '23078', '2307 6', '22437'] with vector representation $\overrightarrow{PS_{ut}}=[-0.044143$ 0.044143 0.001968 $\ldots.-0.001035$ 0.018966 $0.02645]$ and a purchase sequence in training data [21238, 21239, 21245] with vector representation as $\overrightarrow{PS_u}=[0.032593$ $0.066846-0.172250\ldots\ldots\ \ldots-0.025199-0.043636$ $0.08122]$, so the cosine similarity between vectors of these two purchase sequences using Eq. 5 is 0.81.

Similarly, cosine similarity between this purchase sequence vector $\overrightarrow{PS_{ut}}$ in test data is computed with other purchase sequence vectors $\overrightarrow{PS_u}$ in the training data. The results of cosine similarity are then sorted in decreasing order to select top-N (where N = 5, 10, or 15) customers with similar purchase behaviors, i.e., having products that

Table 10 Purchase sequence database of top-N customers

SID	Sequence
1	< 21239, (21239, 20655, 21242), (21239, 21242), (21366), (21242, 22246) >
2	< (21239, 21366), 21242, (20655, 21242), (21239, 21377) >
3	< (21377, 22246), (21239, 20655), (21366, 22246), (21242, 20655) >
4	< 21377, 22198, (21239, 22246), 21242, 20655, 21242 >
5	< (20674), (20674, 21245, 21239), 21242 >
6	< 21238, 21239, 21245 >
7	< (20655, 20674, 20675, 20675), (21242, 21245) >
8	< 20675, 21238, 21245 >
9	< 21366, 21239, 21242, 21245 >

are semantically similar to products in the target sequence. This process is repeated for all the purchase sequences in the test data (i.e., for each target user).

So, here we obtain semantically similar purchase sequences by computing similarities between the test and the train purchase sequences using the hybrid similarity matrices ($PV_{p2vec\&glove}$ and $PV_{doc2vec\&tfidf}$).

3.3.3 Mining Semantic Embedded Sequential Patterns and Rules

Purchase sequences of top-N semantically similar customers were extracted and database of those purchase sequences was created as shown in Table 7. Using Sequential Historical Database (SHOD) (Bhatta, Ezeife and Butt, 2019) format, these semantically similar purchase sequences were formatted to extract frequent semantic sequential patterns. For example, using SHOD format, a sequence will be represented as < 21239 21366–1 21242–1 20655 21242–1 21239 21377–2 > where a −1 indicates the end of item and −2 indicates the end of sequence. For example, in our running example, Table 10 shows purchase sequences with products that are semantically similar to our target purchase sequence [‘23077’, ‘23078’, ‘23076’, ‘22437’].

Next, frequent semantic sequential purchase patterns from these semantically similar purchase sequences (Table 10) are extracted using Prefix Span (Pei et al. 2004). As these frequent sequences were generated from purchase sequences with products, which are semantically similar, so the sequences obtained represents products that are similar in semantics and frequently purchased in sequential order. A detailed step-by-step example of extracting sequential patterns using PrefixSpan (Pei et al. 2004) is explained in Sect. 1.1.1. At this step, we enrich the process of mining sequential patterns by integrating semantic information of products from matrix M. The goal is to generate semantic-rich frequent sequential patterns by pruning product sequences that have a semantic similarity score less than the specified similarity threshold in addition to the traditional method of removing sequences based on the support count. A similarity threshold of 0.5 was used. This gives us frequent sequential purchase patterns of products which are similar in semantics.

For example, first, some of the frequent sequential length-2 (L_2) itemsets with their support count are: L_2 = {(21239, 20655):4, (21239, 21242):5, (21239, 22246):3, (20655, 21242):5, (21242, 20655):3, (21366, 21242):4}. We will then check the semantic similarity between these products to reduce the search space and further prune the itemsets that are not semantically similar by looking at item-to-item semantic similarity matrix (Fig. 9). In this case, the patterns (21239, 20655):4 and (21239, 22246):3 will be pruned out as semantic similarity between (21239, 20655) is 0.10 and between (21239, 22246) is 0.11, which are less than our specified threshold of 0.5. So, final semantic sequential purchase patterns are:

L = {21239, 20655, 21242, 21366, 22246, 21377, 21245, (21239, 21242), (21239, 21245), (21242, 21239), (21242, 21245), (21239, 21242, 21239), (21239, 21242, 21245), (21242, 21239, 21242), (21242, 21239, 21245)}.

From these semantic sequential purchase patterns, one of the semantic sequential rule can be:

21239, 2124 $\rightarrow$ 221245
Pink Polka Dot Cup, Red Retrospot Plate Green $\rightarrow$ Polka Dot Plate

Here we can see that the recommended product 21245 (Green Polka Dot Plate) is similar in semantics to the products 21239 (Pink Polka Dot Cup) and 21242 (Red Retrospot Plate) which the user has already liked or purchased and is more close to the interest of user.

3.4 Sequential recommendation phase

This last phase involves incorporating the semantic and sequential associations between products into the item–item similarity matrix in order to enrich the collaborative filtering item–item matrix for recommending products to customers, which are similar in semantics and are purchased in sequential order as well. The steps involved in this phase are: (i) score computation for products, (ii) semantic and sequentially rich item-to-item similarity matrix and (iii) semantically rich and sequential Top-K recommendation.

3.4.1 Score computation for products

After extracting semantically rich frequent sequential purchase patterns, this sequential information about products is populated into the semantically rich item-to-item similarity matrix M. For each entry $R_{x,y}$ in the matrix M, we update the matrix entries by computing a score using Eq. 7,

$$\text{Score}(x, y) = \alpha(\text{CosineSimilarity}(x, y)) + \beta(\text{Confidence}(x, y)) + \gamma(\text{lift}(x, y)) \tag{7}$$

where CosineSimilarity(x, y) is already computed using Eq.5 and Confidence (x,y) and lift(x,y) are computed using Eq. 8 and Eq. 9 as:

$$\text{Confidence}(x, y) = \frac{\text{Support}(x, y)}{\text{Support}(x)} \tag{8}$$

$$\text{lift}(x, y) = \frac{\text{Support}(x, y)}{\text{Support}(x)*\text{Support}(y)} \tag{9}$$

where support(x,y) measures how frequently products x and y occur sequentially in all available sequences.

and Confidence(x,y) determines the sequential co-occurrence of products x and y given all sequences in which x occurs. The lift score lift (x,y) indicates whether there is a relationship between items x and y, or whether the two items are occuring together in the same order simply by chance (i.e., at random). Unlike the confidence metric whose value may vary depending on direction (e.g., confidence{x → y} may be different from confidence{y → x}), lift has no direction. This means that the lift(x,y) is always equal to the lift (y,x).

For example, consider the products "21242" and "21245," their cosine similarity with product "21239" is 0.58 and 0.88, respectively. Confidence(x,y) and lift (x,y) based on their frequent sequential support count can be calculated as:

$$Confidence(21239, 21242) = \frac{Support(21239, 21242)}{Support(21239)} = \frac{6/9}{7/9} = 0.85$$

$$lift(21239, 21242) = \frac{Support(21239, 21242)}{Support(21239) * Support(21242)} = \frac{6/9}{7/9 * 7/9} = 1.10$$

$$Confidence(21239, 21245) = \frac{Support(21239, 21245)}{Support(21239)} = \frac{2/9}{7/9} = 0.28$$

$$lift(21239, 21245) = \frac{Support(21239, 21245)}{Support(21239) * Support(21245)} = \frac{2/9}{7/9 * 5/9} = 0.51$$

A lift score of 1 implies that there is no relationship between x and y and they occur together by chance). A lift score of greater than 1 shows a positive relationship, indicating that x and y occur together more often, whereas a lift score of less than 1 shows that both x and y occur together less often than random. Based on the above computed values, the Score(x,y) between products using Eq. 7 can be computed as:

$Score(21239, 21245) = \alpha(\text{Cosine Similarity}(x, y)) + \beta$ $(\text{Confidence}(x, y)) + \gamma(lift(x, y) = 0.5 *0.58 + 0.3*0.85 + 0.2 *1.10 = 0.76$.

Similarly,

$Score(21239, 21242) = 0.5 *0.88 + 0.3*0.28 + 0.2*0.51 = 0.62$.where $\alpha + \beta + \gamma = 1$. Similarly, score for other products is computed.

3.4.2 Semantic and sequentially rich item-to-item similarity matrix

The entries in the matrix M are updated with this score value showing the semantic and sequential relationship between products. Figure 10 shows semantic and sequentially rich updated item-to-item matrix M1 populated using Eq. 10 after score calculations for sample products. Matrix M1 can now be used by CF to recommend Top-K personalized items to users.

$$\text{M1}_{x,y} = \begin{cases} 1, \, if \; x = y \\ Score(x, y), \; otherwise \end{cases} \tag{10}$$

3.4.3 Semantically rich and sequential top-k recommendation

Finally, next item(s) for a user is predicted by taking the purchase sequence of each user in train data (user profile) and then generating recommendations for every item in the user profile by looking at its score with other available products from the matrix $M1$. Items having the highest score are retrieved and sorted in decreasing order. This process is repeated for all items in the user profile, and then, lists of

$$M1 = \left[\begin{array}{cccccccccccc} & 20674 & 21242 & 21238 & 21245 & 21239 & 20655 & 20675 & 21366 & 22246 & 21377 & 22198 \\ 20674 & 1.00 & 0.16 & 0.39 & 0.40 & 0.24 & 0.17 & 0.47 & 0.24 & 0.97 & 0.05 & 0.57 \\ 21242 & 0.55 & 1.00 & 0.87 & 0.98 & 0.71 & 0.25 & 0.35 & 0.21 & 0.39 & 0.31 & 0.12 \\ 21238 & 0.56 & 0.87 & 1.00 & 0.01 & 0.59 & 0.35 & 0.61 & 0.91 & 0.76 & 0.39 & 0.98 \\ 21245 & 0.40 & 0.02 & 0.01 & 1.00 & 0.17 & 0.38 & 0.65 & 0.02 & 0.34 & 0.58 & 0.34 \\ 21239 & 0.24 & 0.62 & 0.59 & 0.76 & 1.00 & 0.13 & 0.57 & 0.42 & 0.41 & 0.15 & 0.37 \\ 20655 & 0.17 & 0.21 & 0.35 & 0.38 & 0.13 & 1.00 & 0.07 & 0.69 & 0.59 & 0.61 & 0.23 \\ 20675 & 0.98 & 0.87 & 0.94 & 0.65 & 0.86 & 0.07 & 1.00 & 0.42 & 0.13 & 0.28 & 0.44 \\ 21366 & 0.86 & 0.21 & 0.94 & 0.02 & 0.67 & 0.62 & 0.42 & 1.00 & 0.47 & 0.59 & 0.42 \\ 22246 & 0.97 & 0.21 & 0.90 & 0.34 & 0.41 & 0.04 & 0.90 & 0.47 & 1.00 & 0.15 & 0.32 \\ 21377 & 0.87 & 0.11 & 0.94 & 0.58 & 0.15 & 0.96 & 0.28 & 0.59 & 0.15 & 1.00 & 0.28 \\ 22198 & 0.33 & 0.12 & 0.98 & 0.39 & 0.67 & 0.57 & 0.76 & 0.90 & 0.73 & 0.66 & 1.00 \end{array}\right]$$

Fig. 10 Semantic and sequentially rich updated item-to-item matrix M1

Table 11 Product recommendation by our proposed method

User's purchase products	Top-3 products (semantically similar and sequential)		
21242	21245	21238	21239
20655	21242	20655	20675
20675	20674	21239	21238

top K items are generated which are semantically similar and purchased in sequential order.

In our running example, for the sequence where the user purchased products as < 21242, 20655, 20675 >, the recommended products will be as shown in Table 11.

Eliminating the common products and those already purchased by the user, the final set of recommended items for the user will be {21245, 20674, 21238, 21239}.

4 Experimental evaluation

In this section, we present our experimental setup and then results and analysis.

4.1 Datasets and implementation details

- Online Retail[1]: This dataset contains purchases made during an eight-month period between 01/12/2010 and 09/12/2011 for a UK-based retail company that sells unique all-occasion gifts.
- Amazon[2]: This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs). To test our model, we selected the review-K core (which is a subset of the data set where all items have at least K reviews, where K = 5) and product metadata for categories includ-

Table 12 Data set statistics (product reviews and metadata)

		Reviews (Amazon) and Purchases (Online Retail) Data							Metadata
Data set /Statistics		Total no. of transactions	No. of unique users	No. of unique items	Average no. of reviews/ purchases per item	Max. sequence length	Average sequence length	Minimum Sequence Length	No. of unique items
Amazon	Fashion	883,636	743,216	186,054	4.66	40	1.17	1	186,637
	Movies and TV	8,290,109	3,755,907	181,996	45.55	4036	2.21	1	203,970
	Beauty	353,956	317,982	32,586	10.86	23	1.11	1	32,992
Online Retail		240,007	2974	3282	58.10	294	16.64	1	4497

[1] https://archive.ics.uci.edu/ml/datasets/online+retail.

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

ing fashion, beauty, movies and TV. Details of data set statistics are provided in Table 12.

We implemented the proposed model using Python. For mining sequential patterns using PrefixSpan (Jian Pei et al., 2004), we used open-source data mining library SPMF.[3] To compare our proposed model with the baselines, we used the code provided from their authors and the github repository.[4]

4.2 Preprocessing and Hyper-parameter Tuning

For data set partitioning, we adopted commonly used strategies of: (i) leave one out (the most recent, i.e., last sequence of each user is used for testing and all remaining sequences for training) and (ii) temporal user splitting (where a percentage of the last interactions of each user is reserved for testing rather than just one). Availability of a rating or review (Amazon) and purchase (Online Retail) is considered as user–item interaction, and we used timestamps to determine the sequential order of actions. Purchases made by each customer were grouped into sequences according to the timestamp. Data were preprocessed to create train and test data. For leave one out, the training data were created from those purchase sequences and the last purchase sequence of each customer was used to create the test set for evaluating model's performance. In the temporal user splitting, we used train and test splits of (a)70%, 30% and (b) 80%, 20%. Users with at least five purchasing records are selected.

All the models have some parameters to tune. We follow the reported optimal parameter settings for the baseline methods. For our model, the embedding dimension **d** is determined by grid search in the range {10,20,30,40,50,100}, number of top users with similar behavior **N** as {5,10,15}, number of top recommendation items **K** in {1,5,10,20,50,100}, minimum support **s** (%) for mining sequential patterns as {1,2,3}and semantic similarity **m_sem = 0.5**. The values of coefficients (alpha, beta and gamma) while computing score measure for products were explored through grid search, and the best results were with $\alpha = 0.5, \beta = 0.3$ *and* $\gamma = 0.2$. For other parameters, optimal performance was with d = 100, N = 15 and s = 1, m_sem = 0.5.

4.3 Evaluation metrics

The model was evaluated on many metrics including Precision@K, Recall@K, Mean Reciprocal Rank (MRR), Hitrate@K and NDCG@K. The different evaluation metrics are defined as:

- **Precision@K:** It is defined as the proportion of recommended items in the top-K set that are relevant.
- **Recall@K:** Recall is defined as the proportion of relevant items found in the top-K recommendations.
- **Mean Reciprocal Rank (MRR):** The reciprocal rank of items recommended is the multiplicative inverse of the rank of the first correct recommended item among the top-K recommendations. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of ground truths R, where $rank_i$ refers to the rank position of the first relevant item for the i-th ground truth.
- **Hit rate@K:** Percentage of users that can receive at least one correct recommendation.
- **Normalized Discounted Cumulative Gain (NDCG@K):** Evaluates ranking performance by taking the positions of correct items into consideration.

For each user in a test sequence, we predict lists of top-K personalized items where K is in {1,5,10,20,50,100}.We first compute the per-user score for each K and then report the global average score for all users for each K .

4.4 Complexity analysis

The dominant term in the computational complexity of our proposed model is O(n^2) mainly due to computing similarity at item level and sequence level. Next, we discuss the complexity according to the models, data processing and mining of sequential patters.

4.4.1 Complexity of models

Prod2vec: The complexity is proportional to the vocabulary size (unique products), which is computationally expensive in practical tasks as it can easily reach millions of products. As an alternative, negative sampling is used which significantly reduces the computational complexity. Depending on the size of the corpus, the complexity grows linearly with the size of the corpus. Therefore, if N represents the size of the corpus and V represents the unique products in the vocabulary, the complexity will be O(N*log(V)).
Glove: The computational complexity of the model depends on the number of nonzero elements in the co-occurrence matrix X. As this number is always less than the total number of entries of the matrix, the model scales no worse than O($|V^2|$) where V represents the size of the vocabulary.
(c)Doc2vec: The runtime of the model is linear in the number of input documents, i.e., purchase sequences

[3] https://www.philippe-fournier-viger.com/spmf/index.php.

[4] https://github.com/mquad/sars tutorial.

with products and metadata. Keeping all other parameters equal, increasing the number of input documents, the runtime will be increased
(d) TF-IDF: If N represents the number of documents and TN represents the total number of terms, then the worst-case time complexity of this will be O(TN * N). In practice, the number of documents in which a particular term appears is very less, and hence, the time taken will be much lower than that.

4.4.2 Complexity of dataset pre-processing

The various Natural Language Processing (NLP operations were performed using the Natural Language ToolKit (NLTK) library from python consists of a set of text processing libraries for faster data preprocessing.

4.4.3 Complexity of mining the sequential patterns

As we used Prefix Span (Pei et al. 2004), in which no candidate sequence needs to be generated so major complexity is involved in constructing projected databases with respect to the sequential pattern(s). However, as the projected databases keep shrinking so it also lowers the computation process. The reasons are because in a sequence database, the number of sequential patterns that grow quite long is usually small, and therefore, in a projected database, when prefix grows, the number of sequences reduces substantially. In the worst case, in PrefixSpan, a projected database is constructed for every sequential pattern. If there are N sequential patterns, then the complexity for constructing projected databases will be O(N). The cost is non-trivial, if a good number of sequential patterns exists.

4.5 Baseline methods for comparison

To show the effectiveness of our model, we considered recommendation baselines under three groups.

The *first group* includes general recommendation methods based on user feedback without any sequential order of user actions.

1. **Popularity-Based (POP).** All items are ranked by their popularity in all users' sequences, where popularity is determined by the number of interactions.
2. **Bayesian-Personalized Ranking (BPR)** (Rendle et al. 2010)**.** A state-of-the-art method for non-sequential item recommendation on implicit feedback, utilizing matrix factorization model.

The *second group* includes sequential recommendation methods based on sequential pattern mining and first-order Markov chains, which consider the last visited item.

3. **Historical Purchase Click (HPCRec)** (Xiao and Ezeife 2018) **and Historical Sequential Purchase (HSPRec)** (Bhatta et al. 2019)**.** It mines frequent sequential click and purchase behavior patterns using the consequential bond between click and purchase sequences and then using this quantitatively and qualitatively rich matrix for collaborative filtering to provide better recommendations.
4. **Factorized Personalized Markov Chain (FPMC)** (Rendle et al. 2010). A hybrid approach that combines matrix factorization (MF), which factorizes the matrix on user–item preferences for learning users' general taste and Markov chains (MCs) that model sequential behavior through a transition graph built over items which predict users' next action based on the recent actions.

The *third group* includes sequential recommender systems based on deep learning, which include various or all previously visited items.

5. **GRU4Rec** (Hidasi et al. 2016).To model sequential dependencies and making predictions in session-based recommendation systems, we proposed this method based on recurrent neural networks (RNNs).
6. **Convolutional Sequence Embeddings Caser** (Tang and Wang 2018)**.** A convolutional neural network (CNN)-based method, which takes the embedding matrix of the L most recent items and applies convolution operations on it to achieve sequential recommendation.
7. **Self-Attentive Sequential (SASRec)** (Kang and McAuley 2018)**.** It captures long-term user preferences by using attention mechanism and makes its predictions based on relatively few actions.

5 Results and analysis

Our proposed model SSHRec gave improved performance after incorporating products' metadata to learn product semantics and using semantic similarity, confidence and lift measures to compute relationship between products in comparison with the SEMSRec model and other baselines on all K tested. High precision and recall measures of sequential recommenders such as SASRec (Kang and McAuley 2018), Caser (Tang and Wang 2018)), HSPRec19 (Bhatta et al. 2019) and RNN (Hidasi et al. 2016) indicate that learning sequential information about customers' behavior is important to capture user's long- and short-term preferences and to improve quality of recommendations. Factorized personalized Markov chain (FPMC) (Rendle et al. 2010) model, on the other hand, showed least performance with the lowest precision and recall score, indicating that it could not learn the semantics of items and sequential purchase patterns of

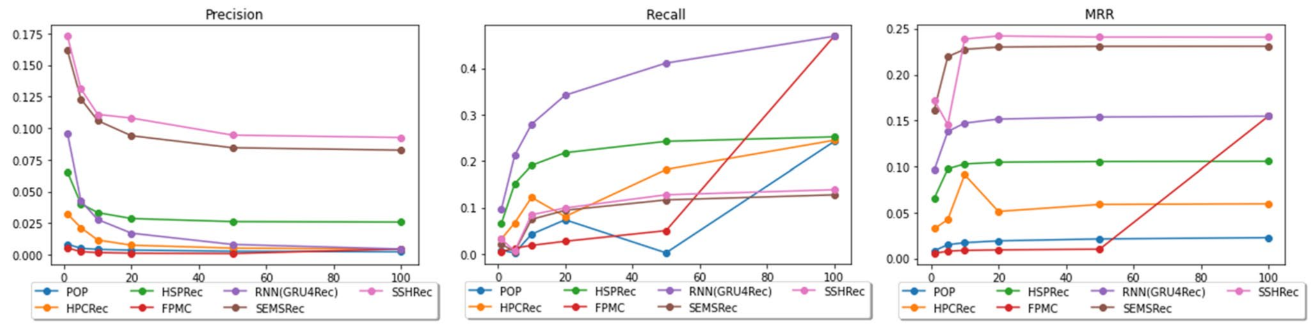

Fig. 11 Performance comparison of proposed SSHRec with other models

Table 13 Results of proposed system (SSHRec) on different datasets with K = 10

Evaluation metrics/data set	Amazon			Online Retail
	Fashion	Movies and TV	Beauty	
Precision@K	0.0075	0.0165	0.0909	0.1006
Recall@K	0.0194	0.0360	0.4856	0.1145
MRR	0.0126	0.0276	0.1705	0.1509
Hitrate@K	0.0559	0.1326	0.6705	0.4544
Mean Average Precision	0.0075	0.0154	0.0855	0.0887
NDCG@K	0.0166	0.0319	0.3696	0.1450

customers effectively. SEMSRec (Nasir and Ezeife 2020), on the other hand, gave slightly high performance as compared to other sequential recommender baselines, indicating that integrating items' semantic information to compute item similarities can improve the quality of recommendations. However, the proposed extension SSHRec has outperformed SEMSRec significantly showing the importance of using metadata for learning product semantics and using semantic similarity, confidence and lift measures to compute relationship between products. Figure 11 shows the performance of SSHRec in comparison with other systems on various evaluation metrics.

Furthermore, the results of the proposed model (SSHRec) on different datasets including Online Retail and various categories in Amazon data (Fashion, Beauty, Movies) are presented in Table 13. Here, we report results on all evaluation metrics at a cutoff of K = 10. We notice that SSHRec performed considerably well Amazon's beauty dataset with a hitrate of 67% and 45%, respectively.

Next, we will discuss the impact of the parameters, minimum support, s and number of customers with similar purchase behavior (N) one at a time by holding the remaining parameters at their optimal settings.

5.1 Influence of Top-N customers (N)

We vary the number of customers (N) with similar purchase behaviors generated after finding semantic similarity between training sequences and the target sequence, to explore how it effects the quality of overall recommendations. We used different values of N as 5, 10 and 15, respectively. By increasing the value of N, gradual decrease in model performance was noticed in terms of Top-K recommendations. This is because when the number of customers is increased, number of similar purchase sequences also increases, so we do get more similar products to recommend from, however, those similar products are not necessarily purchased in sequence (which is important to capture users' long- and short-term behaviors), which lowers the recommender's performance. Optimal performance was when N = 5. Furthermore, choosing N less than five yielded purchase sequences, which were very short in length and did not contribute much in yielding useful semantic sequential purchase patterns and rules; therefore, we did not consider that.

5.2 Influence of embedding size

We vary the size of embeddings d and trained multiple models based on embedding size {10,20,30,50,100} and found that d = 100 gave optimal results. The detailed results are shown in Table 14.

5.3 Influence of train and test split

For data set partitioning, we adopted the strategies of (i) leave one out (the most recent, i.e., last sequence of each

Table 14 Prediction performance of SSHRec with different embedding dimension d

Evaluation Metrics	K	Embedding size					
		10	20	30	40	50	100
Precision@K	1	0.1298	0.1725	0.2392	0.2918	0.3009	**0.3284**
	5	0.0427	0.0740	0.1080	0.1328	0.1395	**0.1528**
	10	0.0307	0.0538	0.0764	0.0922	0.0977	**0.1065**
	20	0.0236	0.0411	0.0560	0.0660	0.0701	**0.0768**
	50	0.0185	0.0301	0.0388	0.0442	0.0468	**0.0510**
	100	0.0160	0.0242	0.0294	0.0336	0.0349	**0.0374**
Recall@K	1	0.0968	0.1094	0.1255	0.1364	0.1387	**0.1431**
	5	0.1107	0.1504	0.1834	0.2034	0.2064	**0.2167**
	10	0.1224	0.1728	0.2091	0.2302	0.2338	**0.2452**
	20	0.1395	0.2037	0.2410	0.2637	0.2698	**0.2830**
	50	0.1845	0.2584	0.3002	0.3246	0.3332	**0.3500**
	100	0.2348	0.3173	0.3609	0.3922	0.3990	**0.4181**
MRR		0.1298	0.1725	0.2392	0.2918	0.3009	**0.3284**
Hitrate@K	1	0.1298	0.1725	0.2392	0.2918	0.3009	**0.3284**
	5	0.2006	0.3189	0.4256	0.4907	0.5090	**0.5347**
	10	0.2752	0.4338	0.5486	0.6045	0.6249	**0.6510**
	20	0.3894	0.5825	0.6896	0.7442	0.7628	**0.7899**
	50	0.6130	0.8068	0.8695	0.8966	0.9048	**0.9173**
	100	0.7923	0.9143	0.9356	0.9482	0.9482	**0.9512**
NDCG@K	1	0.1337	0.1777	0.2464	0.3005	0.3099	**0.3382**
	5	0.1353	0.1836	0.2377	0.2752	0.2829	**0.3030**
	10	0.1398	0.1931	0.2461	0.2808	0.2891	**0.3081**
	20	0.1468	0.2072	0.2607	0.2951	0.3047	**0.3243**
	50	0.1650	0.2324	0.2892	0.3245	0.3357	**0.3570**
	100	0.1850	0.2580	0.3165	0.3554	0.3660	**0.3886**

Table 15 Prediction performance of SSHRec with different train and test split strategies

		Train test split		
Evaluation metric	K	Train = 80% test = 20%	Train = 70% test = 30%	Leave one out
Precision@K	1	**0.4846**	0.3805	0.3284
	5	**0.2279**	0.1906	0.1528
	10	**0.1450**	0.1290	0.1065
	20	**0.0932**	0.0868	0.0768
	50	**0.0524**	0.0537	0.0510
	100	**0.0347**	0.0375	0.0374
Recall@K	1	**0.1967**	0.1323	0.1431
	5	**0.3484**	0.2466	0.2167
	10	**0.3918**	0.2911	0.2452
	20	**0.4425**	0.3386	0.2830
	50	**0.5175**	0.4176	0.3500
	100	**0.5847**	0.4892	0.4181
MRR		**0.4846**	0.3805	0.3284
Hitrate@K	1	**0.4846**	0.3805	0.3284
	5	**0.6835**	0.6100	0.5347
	10	**0.7780**	0.7099	0.6510
	20	**0.8611**	0.8163	0.7899
	50	**0.9400**	0.9275	0.9173
	100	**0.9607**	0.9648	0.9512
NDCG@K	1	**0.4993**	0.3897	0.3382
	5	**0.4512**	0.3444	0.3030
	10	**0.4667**	0.3601	0.3081
	20	**0.4908**	0.3820	0.3243
	50	**0.5243**	0.4189	0.3570
	100	**0.5515**	0.4498	0.3886

user is used for testing and all remaining sequences for training) and (ii) temporal user splitting (where a percentage of the last interactions of each user is reserved for testing rather than just one). Availability of a rating (Amazon) and purchase (Online Retail) is considered as user–item interaction, and we used timestamps to determine the sequential order of actions. Data were preprocessed to create train and test data. Purchases made by each customer were grouped into sequences according to the timestamp. For leave one out, the training data were created from those purchase sequences and the last purchase sequence of each customer was used to create the test set for evaluating model's performance. In the temporal user splitting, we used train and test splits of (a) 70%, 30% and (b) 80%, 20%. Users with at least 5 purchasing records are selected. The experiments showed that the proposed model SSHRec performed well when the data set was split using temporal user setting with training as 80% and test as 20%, which indicates that including more historical user interactions better captures users' interest and provides relevant recommendations. Results of the model while using different train and test split strategies are shown in Table 15.

6 Conclusion and future work

We propose a model to improve e-commerce product recommendation using semantic context and sequential historical purchases (SSHRec) by exploring the effectiveness of semantic associations between items obtained from item (products') metadata such as title, description and brand based on their semantic context (co-purchased and co-reviewed products). The semantics of these interactions were obtained through distributional hypothesis and then integrated into different phases of recommendation process such as (i) pre-processing, to learn associations between items, (ii) candidate generation, while mining sequential patterns and in collaborative filtering to select top-N and (iii) output (recommendation). Experiments performed on publically available e-commerce data sets (Online Retail and Amazon (Fashion, Beauty, movies and TV) showed that the proposed model performed well in terms of recommending products that are semantically similar to user preferences and are sequential to better reflect user's long- and short-term preferences. For future work, we intend to enhance the item matrix by extracting semantic and sequential information from other data sources such as customer's wish list and their social networks. We also aim to explore the effect of incorporating users' side information such as their demographics (age, gender, location) in addition to the items' metadata to capture user's interests.

Funding This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

References

Adomavicius, G., & Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook* (pp. 217–253). Springer.

Aggarwal CC (2016) An Introduction to Recommender Systems. In: Aggarwal CC (ed) Recommender Systems: The Textbook. Springer, Berlin

Agrawal R, Srikant R, Road H, Jose S (1994) Fast Algorithms for Mining Association Rules. 13.

Agrawal R, Srikant R (1995) Mining sequential patterns. Proceedings of the Eleventh International Conference on Data Engineering. https://doi.org/10.1109/ICDE.1995.380415

Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential PAttern mining using a bitmap representation. Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/775047.775109

Bernhard SD, Leung CK, Reimer VJ, Westlake J (2016) Clickstream prediction using sequential stream mining techniques with markov chains. *Proceedings of the 20th International Database Engineering & Applications Symposium*, 24–33. https://doi.org/10.1145/2938503.2938535

Bhatta R, Ezeife CI, Butt MN (2019) Mining sequential patterns of historical purchases for e-commerce recommendation. In: Ordonez C, Song IY, Anderst-Kotsis G, Tjoa AM, Khalil I (eds) Big data analytics and knowledge discovery. Springer, Berlin

Bobadilla J, Ortega F, Hernando A, Gutiérrez A (2013) Recommender systems survey. Knowl-Based Syst 46:109–132. https://doi.org/10.1016/j.knosys.2013.03.012

Brafman RI, Heckerman D, Shani G (2003) Recommendation as a stochastic sequential decision problem. In: ICAPS (pp 164–173)

Chen X, Xu H, Zhang Y, Tang J, Cao Y, Qin Z, Zha H (2018) Sequential recommendation with user memory networks. Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. https://doi.org/10.1145/3159652.3159668

Choi K, Yoo D, Kim G, Suh Y (2012) A hybrid online-product recommendation system: combining implicit rating-based collaborative filtering and sequential pattern analysis. Electron Commer Res Appl 11(4):309–317. https://doi.org/10.1016/j.elerap.2012.02.004

de Gemmis M, Lops P, Musto C, Narducci F, Semeraro G (2015) Semantics-aware content-based recommender systems. In: Ricci F, Rokach L, Shapira B (eds) Recommender systems handbook. Springer, US

Ekstrand MD, Riedl JT, Konstan JA (2011) Collaborative Filtering Recommender Systems. Now Publishers Inc.

Gabrilovich E, Markovitch S (2009) Wikipedia-based semantic interpretation for natural language processing. J Arti Intell Res 34:443–498. https://doi.org/10.1613/jair.2669

Grbovic M, Radosavljevic V, Djuric N, Bhamidipati N, Savla J, Bhagwan V, Sharp D (2015) E-Commerce in Your Inbox: product Recommendations at Scale. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1809–1818. https://doi.org/10.1145/2783258.2788627

Hidasi B, Karatzoglou A, Baltrunas L, Tikk D (2016) Session-based Recommendations with Recurrent Neural Networks. arXiv:1511.06939 *[Cs]*.

Jannach D, Zanker M, Felfernig A, Friedrich G (2010) Recommender systems: an introduction. Cambridge University Press, UK

Kang, W.-C., & McAuley, J. (2018). Self-Attentive Sequential Recommendation. ArXiv:1808.09781 *[Cs]*.

Kim YS, Yum B-J (2011) Recommender system based on click stream data using association rule mining. Expert Syst Appl 38(10):13320–13327. https://doi.org/10.1016/j.eswa.2011.04.154

Kim YS, Yum B-J, Song J, Kim SM (2005) Development of a recommender system based on navigational and behavioral patterns of customers in e-commerce sites. Expert Syst Appl 28(2):381–393. https://doi.org/10.1016/j.eswa.2004.10.017

Le Q, Mikolov T (2014) Distributed representations of sentences and documents. PMLR 32(2):1188–1196

Mabroukeh NR, Ezeife CI (2010) A taxonomy of sequential pattern mining algorithms. ACM Comput Surv. https://doi.org/10.1145/1824795.1824798

Middleton SE, Shadbolt NR, De Roure DC (2004) Ontological user profiling in recommender systems. ACM Trans Infor Sys 22(1):54–88. https://doi.org/10.1145/963770.963773

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed Representations of Words and Phrases and their Compositionality. In:CJC Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26* (pp. 3111–3119). Curran Associates, Inc. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Miller GA (1995) WordNet: a lexical database for English. Commun ACM 38(11):39–41. https://doi.org/10.1145/219717.219748

Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M-C (2004) Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440. https://doi.org/10.1109/TKDE.2004.77

Pennington J, Socher R, Manning C (2014) Glove: global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Rendle S, Freudenthaler C, Schmidt-Thieme L (2010) Factorizing personalized Markov chains for next-basket recommendation. *Proceedings of the 19th International Conference on World Wide Web*, 811–820. https://doi.org/10.1145/1772690.1772773

Ricci F, Rokach L, Shapira B (2011) Introduction to recommender systems handbook. In: Ricci F, Rokach L, Shapira B, Kantor PB (eds) Recommender systems handbook. Springer, US

Saini S, Saumya S, Singh JP (2017) Sequential purchase recommendation system for e-commerce sites. In: Saeed K, Homenda W, Chaki R (eds) Computer information systems and industrial management. Springer, Berlin

Salehi M, Nakhai Kamalabadi I (2013) Hybrid recommendation approach for learning material based on sequential pattern of the accessed material and the learner's preference tree. Knowl-Based Syst 48:57–69. https://doi.org/10.1016/j.knosys.2013.04.012

Salton G (1988) Automatic text processing: The transformation, analysis, and retrieval of information by computer. Addison-Wesley

Schafer JB, Konstan JA, Riedl J (2001) E-Commerce recommendation applications. Data Min Knowl Disc 5(1):115–153. https://doi.org/10.1023/A:1009804230409

Schafer JB, Frankowski D, Herlocker J, Sen S (2007) Collaborative filtering recommender systems. In: Brusilovsky P, Kobsa A, Nejdl W (eds) The adaptive web: methods and strategies of web personalization. Springer, Berlin

Semeraro G, Lops P, Basile P, de Gemmis M (2009) Knowledge infusion into content-based recommender systems. Proceedings of the Third ACM Conference on Recommender Systems. https://doi.org/10.1145/1639714.1639773

Semeraro G, Degemmis M, Lops P, Basile P (2007) Combining learning and word sense disambiguation for intelligent user profiling. Proceedings of the 20th International Joint Conference on Artifical Intelligence, 2856–2861

Shani G, Heckerman D, Brafman RI (2005) An MDP-based recommender system. J Machine Learn Res 6:1265–1295

Su Q, Chen L (2015) A method for discovering clusters of e-commerce interest patterns using click-stream data. Electron Commer Res Appl 14(1):1–13. https://doi.org/10.1016/j.elerap.2014.10.002

Su X, Khoshgoftaar TM (2009) A survey of collaborative filtering techniques. Adv Arti Intell 2009:1–19. https://doi.org/10.1155/2009/421425

Tang J, Wang K (2018) Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. arXiv:1809.07426 *[Cs]*.

Turney PD, Pantel P (2010) From frequency to meaning: vector space models of semantics. J Artif Intell Res 37:141–188. https://doi.org/10.1613/jair.2934

Xiao Y, Ezeife CI (2018) E-Commerce product recommendation using historical purchases and clickstream data. In: Ordonez C, Bellatreche L (eds) Big data analytics and knowledge discovery. Springer, Berlin, pp 70–82

Yap G-E, Li X-L, Yu PS (2012) Effective next-items recommendation via personalized sequential pattern mining. In: Lee S, Peng Z, Zhou X, Moon Y-S, Unland R, Yoo J (eds) Database systems for advanced applications. Springer, Berlin, pp 48–64