applied sciences

MDPI

Article

New Vector-Space Embeddings for Recommender Systems

Sandra Rizkallah *, Amir F. Atiya and Samir Shaheen

Department of Computer Engineering, Faculty of Engineering, Cairo University, Giza 12613, Egypt; amir@alumni.caltech.edu (A.F.A.); sshaheen@eng.cu.edu.eg (S.S.)

* Correspondence: sandrawahid@cu.edu.eg

Abstract: In this work, we propose a novel recommender system model based on a technology commonly used in natural language processing called word vector embedding. In this technology, a word is represented by a vector that is embedded in an n-dimensional space. The distance between two vectors expresses the level of similarity/dissimilarity of their underlying words. Since item similarities and user similarities are the basis of designing a successful collaborative filtering, vector embedding seems to be a good candidate. As opposed to words, we propose a vector embedding approach for learning vectors for items and users. There have been very few recent applications of vector embeddings in recommender systems, but they have limitations in the type of formulations that are applicable. We propose a novel vector embedding that is versatile, in the sense that it is applicable for the prediction of ratings and for the recommendation of top items that are likely to appeal to users. It could also possibly take into account content-based features and demographic information. The approach is a simple relaxation algorithm that optimizes an objective function, defined based on target users', items' or joint user–item's similarities in their respective vector spaces. The proposed approach is evaluated using real life datasets such as "MovieLens", "ModCloth", "Amazon: Magazine_Subscriptions" and "Online Retail". The obtained results are compared with some of the leading benchmark methods, and they show a competitive performance.

Keywords: collaborative filtering; item recommendations; item vectors; recommender systems; spherical embeddings; word embeddings

Citation: Rizkallah, S.; Atiya, A.F.; Shaheen, S. New Vector-Space Embeddings for Recommender Systems. *Appl. Sci.* **2021**, *11*, 6477. https://doi.org/10.3390/app11146477

Academic Editor: Ángel González-Prieto

Received: 6 June 2021
Accepted: 8 July 2021
Published: 13 July 2021

1. Introduction

The growing long term trend towards the increasing role of online services has accelerated even further since the start of the pandemic in 2020. These services include e-commerce, online advertisement, streaming services, booking channels and others. This poses a challenge since there are millions of users and millions of products or items and they need to "find" each other. Recommender systems [1,2] aim to provide this matching. They are basically intelligent systems that predict the user's preferences and recommend the items that would most likely interest him/her. The purchase behavior of a user generally follows a pattern that reveals his likings, and this purchase pattern plays an important role in recommending items to the user. For example, a user who may have a purchase pattern of buying or giving high ratings for high end electronic gadgets is more likely to initiate a purchase action if he is presented with a recommendation of similar gadgets.

Recommender Systems benefit both the consumer and the businesses. The consumer gets to easily find the products or services he seeks, and businesses will increase their sales as they will be targeting more willing users. Recommender Systems can be classified into six classes [3]:

- *Collaborative Filtering:* These are referred to as "people-to-people correlation". These systems recommend items for users based on other users with similar taste. Two types of collaborative filtering exist [4]:
  - User–user (user-based): items are recommended based on the ratings that users with similar tastes have given to these items.

  - Item–item (item-based): items are recommended based on the similarity between them. The similarity computation between items depends on users who have rated/purchased both of these items.

- *Content-Based:* Recommendations are suggested based on the user's history of preferences and profile where the features of the items are taken into consideration. For example a user who liked or purchased a particular item with certain features/specifications and a certain price range is more likely to show interest in another item possessing similar features.
- *Demographic:* These systems categorize users and provide recommendations based on the demographic profile of the user, such as age, gender, country, occupation, and so forth. Users in the same demographic group tend to have similarities in their interests, as opposed to users in different demographic groups.
- *Knowledge-Based:* These systems use knowledge to construct relations between items and users' needs and preferences. The recommendations are suggested based on how the item is beneficial to a certain user.
- *Community-Based:* These systems make use of the information available in social networks. Recommendations can also be considered as "people-to-people correlation", but this time relying on the user's network of friends.
- *Hybrid:* These systems combine different approaches from those mentioned above.

"Collaborative Filtering" (CF) is one of the most widely used Recommender Systems classes [5]. Because the user's past transactions and the item's past ratings and selling events may not be large enough to assess his preference for the item, other like-minded users' behavior and other resembling items will be used as proxies. In other words, users with similar taste are matched with the current user, their purchasing and ratings behavior is observed and, based on that, items are recommended to the user in question. In such models, novel items are also introduced to the user in order to discover new interests. It is well-known that less frequently sold items present the greatest challenge in Recommender Systems because of the dearth of purchasing precedence. This makes collaborative filtering a promising approach.

Recommendations generated by CF fall into two classes: prediction or recommendation. The first one attempts to predict the rating that the user is likely to give to a certain item. Recommendation is concerned with the top k items that the user is likely to be interested in or purchase [6]. Conventional approaches for CF include matrix factorization and neighborhood-based collaborative filtering. Matrix factorization [7] is based on building a user–item interaction matrix to which dimensionality reduction techniques such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) are applied. In neighborhood-based CF, user–item interactions are directly used (based on neighbors or similar items/users) to predict interest for new items [8].

Moreover, recent research in the field of recommender systems has adopted a number of techniques, including deep learning [9–12]. Different architectures of neural networks are proposed such as multilayer perceptron (MLP), convolutional neural network (CNN) and recurrent neural network (RNN). The non-linearity in data can be effectively modeled by deep neural networks through non-linear activations. This allows the capturing of complex and detailed user–item interaction patterns. Deep learning is also beneficial in learning useful representations from input data, specially where in real applications descriptive information about items and users is accessible. Interface optimization is also introduced in recommender systems where visual aspects are taken into consideration [13,14].

In this work, we propose the use of a novel technology called word vector embedding for the Recommender Systems problem. This technology was introduced and was further developed in the natural language processing field. In word vector embedding, words are represented as numerical vectors where the vectors' positions in an n-dimensional space convey the words' semantics. Words of similar meanings will have vectors near each other from the distance point of view. In other words, the vector encodes the corresponding

word's meaning together with all its semantic relations with similar and dissimilar words of the corpus.

Recommender systems are also based on similarities (items to items and users to users). They seem to be promising candidates for the application of vector embedding systems. Users can be represented as vectors that embody all their likings, interests and demographics, with similar-minded users having vectors that are nearby in the space. In other words, the vector encodes the type of person and his preferences. Similarly, an item or a product is a complex entity that is described by several features, for example purchase patterns, ratings, product description and specification, and common preferences among similar users. All this makes simple feature vector modeling not straightforward, and this calls for vector embeddings that are learned using intricate similarity modeling by careful placement of the vectors. In the embedding approach, the item or product is modeled as a vector that encodes the intrinsic features, values and preferences of this item, where these features are encoded over all of its components in a distributed fashion.

The application of word embeddings to recommender systems is a fairly new idea. Only a few works have simultaneously applied this idea recently. The goal of this research is to explore the vector embeddings as a potential addition to the arsenal of other existing technologies, The other goal is to introduce a novel vector embedding that makes this technology applicable to more Recommender Systems problem types. Conventional word vector embeddings, such as the word2vec method [15,16], are trained on context or co-occurrence data. This means that they operate on a basket of purchased items (for a specific user) and from that they infer the similarities between items and possibly provide a recommendation. Therefore, they are not applicable to the problem of learning user ratings, because this basket of elements does not exist and only ratings of items by users do. The method we propose here is applicable in a direct way to both the "recommendation" problem (recommending items to the user based on his past purchases) and the "prediction" problem (predicting a rating for a novel item based on the users' previous patterns of ratings). Moreover, it is very versatile in the sense that it could take into account content-based features, demographic features and social network connections. All these features could be encoded in the vector representations, with CF being the dominant component. Many other conventional approaches, such as matrix factorization, do not directly model the combined effect of CF, content-based, demographic-based and social network features.

In addition, our proposed model has several other beneficial features:

- For our "prediction" problem we have a novel modeling framework. We train an embedding whereby each user maps to a vector and each item maps to a vector in the same space. The users' and items' vectors train side by side, and user–item, user–user and item–item similarities can be taken into account;
- The design is much easier than existing embedding methods. Rather than the existing cumbersome (deep) neural network based approaches that are trained using context data [15,16], the proposed approach is a simple relaxation procedure that is computationally efficient;
- The proposed method possesses an explicit capability to model sentiment, such as likes/dislikes. This is because the vectors are placed in a spherical space. This allows it to directly tackle the polarity issue, that is, "like" (vectors will be near each other) and "dislike" (vectors are on opposite poles of the sphere). Sentiment feedback in the form of like/dislike is particularly prevalent in video channels and music subscription services;
- The algorithm has an incremental nature. This makes adding a user or an item require only a small computational load;
- Conventional methods, such as matrix factorization using SVD or otherwise, are computationally very expensive, especially since the number of users and the number of items can be several tens of thousands. Even though using gradient methods for cases of matrix sparsity can reduce the computational cost, this is still a challenge. At the very least, the storage of such a huge matrix is a problem;

- Rather than dealing with huge user vectors, such as in the neighborhood-based techniques, our proposed embedding method considers (user or item) vectors of dimensions 10, 20 or 50. All information is therefore efficiently compressed;
- The developed embedding model produces a performance competitive with the state-of-the-art conventional recommender systems.

2. Related Work

This section addresses the existing work in the literature on recommender systems, and specifically on collaborative filtering. The section is divided based on the major approaches that are adopted, with a focus on the proposed word embedding works in the field of Recommender Systems.

2.1. Matrix Factorization

The matrix factorization approach is based on building a user–item interaction matrix, for example user–item rating or purchase events matrix. The matrix is often organized such that one dimension represents users while the second dimension represents items. Usually, user–item matrices are sparse as a user is likely to rate or purchase a limited number of items. The approach further decomposes this matrix into the product of two matrices of lower dimensionality. This can be accomplished using dimensionality reduction techniques such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA). The goal of the approach is to discover the latent features underlying the interactions between users and items. The method can be briefly outlined as follows:

Consider R to be the ratings matrix, which includes the users' ratings for items. The goal is to find the two matrices P and Q such that $R \approx P \times Q^T$. This way, user–item interactions can be modeled as inner products. Each user is associated with a vector p while each item is associated with a vector q. The inner product between q_i and p_u captures user u's taste/interest for item i. The dimensionality reduction problem is often augmented by a regularization factor, in order to reduce overfitting. Even though the SVD method can obtain an exact solution, due to the computational complexity, stochastic gradient descent is more often used to solve the problem [17–19].

There have been some variants and improvements of the standard matrix factorization approach. Only a few examples are given below just as an illustration. In [20], the matrix factorization is enhanced by using social networks' information through incorporating social regularization. The social regularization is developed in such a way that a user's taste is close to the user's friends' average taste.

In [21], an Enhanced SVD (ESVD) matrix factorization model is proposed. The model combines classical matrix factorization techniques with active learning in order to perform ratings completion. Moreover, an iterative multi-layer ESVD model is proposed to improve the accuracy of prediction.

In [22], a new regularization method is proposed for matrix factorization. The method is called the elastic net, such that both ridge and lasso regularization methods are combined in a linear way. Moreover, stochastic gradient descent is applied to fit the model. The goal of the model is predicting the users' ratings for unseen movies.

2.2. Neighborhood-Based Collaborative Filtering

This is the other major methodology for tackling the Recommender Systems problem. In this approach, the user–item interactions are directly used to predict the potential interest in new items. Two strategies are typically adopted: user-based or item-based recommendations. In user-based methods, the prediction is based on the ratings of other users for a specific item. These users are the neighbors, that is, the users with similar taste to the given user. In item-based methods, the roles of items and users are reversed, and the prediction is based on the given user's ratings for similar or neighbor items.

In user-based models, the set of k neighbors for a given user are first extracted based on the used similarity measure. The rating prediction can then be performed using aggregation

approaches such as the average and the weighted sum. In item-based models, the k neighbors for each item in the system are first determined. The rating prediction is based on the ratings the given user has assigned to the neighboring items (by averaging or other aggregation methods). This approach is known for its simplicity and reasonably good results. However, it suffers from low scalability and sparsity of the Recommender Systems databases [23–25].

Some variants and modifications of the neighborhood approach have also been proposed. Again, only a few examples are given below. In [26], a neighborhood-based CF approach is proposed. The approach aims at enhancing the computed neighborhood such that fake neighbors are eliminated. Fake neighbors are users who have spurious similarities, but the resemblance is not genuine. In addition, semantic similarity is also deployed, in the sense that users do not need to have ratings for the same items in order to be compared.

In [27], an adaptive k-nearest neighbors collaborative filtering is proposed for recommender systems. A knowledge-based domain specific ontology is devised to obtain customized recommendations.

In [28], a user-based recommender system is proposed. A method is developed to select the optimal neighborhood for a given user. The selection is performed using two strategies: k-nearest neighbors and threshold-based neighbors. The threshold is computed as the distance between a given user and its neighbors, proving that using higher thresholds is more error prone.

In [29], traditional user-based collaborative filtering approaches are extended to achieve better accuracy and coverage levels. The proposed method relies on covering-based rough set theory where redundant users are removed. Since popular items appeal to many users, this can generate many non-genuinely similar users, which preferably have to be eliminated.

2.3. Word Embedding

Word embedding [30] applications to Recommender Systems have recently been proposed. In such models, items or users are represented by vectors, which are learned from the data. The main goal is to have a low dimensional vector space that contains beneficial information for recommending items. To our knowledge, most of the work done in this area is based on the word2vec model, introduced in [15,16].

In [31], the authors propose an approach that embeds products into the vector space. The products' vectors are learned using a skip-gram model [16] such that products with similar contexts (purchases) have close vectors. Then, clustering of the products' vectors is performed using the K-means clustering algorithm. Moreover, the transition probabilities between clusters are modeled. Products with the highest cosine similarities from the highest probability clusters are used for recommendations. Furthermore, vectors are learned for users such that these vectors are updated to predict the products from the users' purchases. It is noted that user to product recommendations are less persistent than product to product recommendations due to the need to continuously account for the new user's purchases. The proposed model relies on email receipts data.

In [32], an item embedding approach is proposed, extending the Prod2Vec model introduced in [31]. The extension is based on incorporating items' metadata during the training phase. The approach is tested on a music dataset.

In [33], a co-factorization model is proposed. The building blocks of the proposed model are matrix factorization and item embedding. The item embeddings are learned based on the word2vec model introduced in [16]. The embeddings are learned using the user's consumed items in addition to co-occurrence counts, which are computed as the number of users who consumed both items. The model is tested using datasets for scientific papers, movies and songs.

In [34], recommendations for venues and check-ins are presented for users. The model is based on word2vec techniques: skip-gram and continuous bag of words (CBOW), introduced in [15,16]. Vectors for items as well as users are learned. Moreover, three

recommendation methods are proposed. The first method is based on recommending the most similar items to a given user by calculating the cosine similarity between the items' vectors and the user's vector. The second method finds the users similar (neighbors) to a given user and then the recommendation is made using these neighbors' preferred items. Finally, the third method combines the previous two methods. The neighbors are found, then the most similar items to both the given user and the neighbors are found and used for recommendation. The approach is tested using a Foursquare check-in dataset.

In [35], a recommender system is proposed based on CBOW word2vec [15] and GloVe [36]. The approach mimics these NLP models such that items are treated as words and users' sessions are treated as sentences. Vectors are learned for items where items' similarities are computed using the cosine distance. These similarities are used to perform recommendations based on the user's history. Moreover, at the recommendation phase the items' similarities are weighted with the user's ratings to obtain a ranked set of recommendations. The approach is tested on a movie dataset where a recall@20 measure is used for performance evaluation. The testing is performed by removing the last movie that the user has seen and uses the last k user's seen movies to recommend a set of 20 movies. If the recommended set includes the movie held out for testing, then a success is counted, otherwise a failure. Finally, the percentage of successes over all users is computed. The authors showed that the GloVe based model outperforms the word2vec based model.

In [37], an item-based collaborative filtering method is proposed. The method is based on the word2vec skip-gram model with negative sampling, introduced in [16]. The paper highlights the effectiveness of item-based similarities for recommender systems where the user information is not incorporated into the model. The authors make a number of justifications as follows. First, better item representations are generated since such methods optimize learning the items' relations directly. Second, the users' information may not be available, for example, a number of online stores allow transactions without requiring the explicit identification of users. Third, the model's computational complexity increases when the number of users notably exceeds the number of items. In the proposed model, vectors for items are learned where a sentence of words is represented as a basket of items. The model is applied on a music dataset as well as a product dataset.

Again in [38], an item-based collaborative filtering method is proposed. Vectors for items are learned based on the skip-gram model with negative sampling, introduced in [16]. Moreover, a new user preference model is designed such that short-term and long-term user preferences are defined.

In [39], an item-based collaborative filtering method is also proposed. The method uses the skip-gram model with negative sampling, introduced in [16] to learn items' vectors. These vectors are learned from transactions data. Moreover, user representations are also generated using average and max pooling for the user's transactions. These users' representations are used to obtain a ranked list of recommendations. The method is tested using an online retail dataset.

In [40], the change in a user's interest given new recommendations is addressed. An attentive item2vec model is proposed, extending the model introduced in [37]. The model learns attributes for user behavior given potential recommendations of items using a context–target attention mechanism. Finally, a neural representation for the user is generated.

3. Proposed Approach

The proposed approach assigns a vector for each item and/or user. The location of the vector is designed in such a way that vectors in the vicinity of each other correspond to similar items or users. The vectors are of relatively high-dimension, in order to encode all the diverse information in the items, and are typically selected somewhere between 10 and 50 in dimension. We assume that the vectors lie on a sphere (i.e., their length is fixed as $\|x\| = 1$). This endows it with a sentiment polarity feature. For example, a user who likes an item will have his vector close to the item's vector, but if he dislikes it, his vector will be

at the polar opposite of the item's vector. The training algorithm that designs the locations of the vectors is a simple relaxation algorithm that works by minimizing a simple error function. An analogous methodology has been developed for the natural language field in [41]. The next subsections elaborate each of the developed systems.

3.1. General Approach

Each user is represented by a vector, and each item is represented by a vector in the same space (i.e., they have the same dimension). Let x_i denote such vector representation. Assume $\|x_i\| = 1$,that is, the vectors lie on the unit sphere. In addition to the sentiment polarity issue, this assumption represents a normalization operation for the vectors, and therefore prevents run-away similarity or distance computations. The distance between any two vectors $\|x_i - x_u\|^2 = \|x_i\|^2 + \|x_u\|^2 - 2x_i^T x_u = 2 - 2x_i^T x_u$ is therefore negatively proportional to their dot product. Rather than using distance as dissimilarity measure, as in many word embedding methods, we use the dot product as a similarity measure.

The proposed vector embedding algorithm is supervised in nature, unlike many of the other word embedding methods that are unsupervised or semi-supervised. The algorithm seeks to determine the vectors pertaining to the users and items such that they satisfy given similarity numbers. Thus, it optimizes the locations of the x_is such that the achieved similarity $x_i^T x_u$ is as close as possible to the target similarity score s_{ui}. Towards this end, we define the following optimization problem:

$$E = \sum_u \sum_i w_{ui} \left[x_u^T x_i - s_{ui} \right]^2$$
$$\text{subject to} ||x|| = 1, \forall u,\ i, \tag{1}$$

where w_{ui} is a weighing coefficient that could potentially put a different emphasis on different terms. There are a number of different versions and similarity terms that could be utilized. A general form that takes into account different possible similarity measures is the following:

$$E = \sum_{u=1}^{U} \sum_{u'=1}^{U} w_1 \left[x_u^T x_{u'} - v_{uu'} \right]^2 + \sum_{u=1}^{U} \sum_{i=1}^{N} w_2 \left[x_u^T x_i - r_{ui} \right]^2 + \sum_{i=1}^{N} \sum_{i'=1}^{N} w_3 \left[x_i^T x_{i'} - o_{ii'} \right]^2$$
$$\text{subject to} ||x|| = 1, \forall u,\ u', i,\ i'. \tag{2}$$

We explain this general formulation as follows. Let x_u and $x_{u'}$ be vectors pertaining to the users and let x_i and $x_{i'}$ be vectors pertaining to the items. The first term corresponds to user–user similarity, where the dot product $x_u^T x_{u'}$ is the computed similarity of the vectors representing user u and user u', and $v_{uu'}$ is the target similarity to be learned. This term can be used to model the following:

- Demographic similarities: For example users who have similar demographic features, such as age, gender, country, profession, and so forth, will have assigned target similarities $v_{uu'}$ some number close to 1, while users with only a fraction of the demographic features will have a low target similarity;
- Community-based: Users who are in similar social networks, such as being friends, will have a high target similarity, otherwise it is low;
- Purchase pattern similarities: The target similarity $v_{uu'}$ can be proportional to the amount of items in common in both users' purchasing patterns.

The second term corresponds to user–item similarity, where the dot product $x_u^T x_i$ is the predicted similarity of user u and item i, and r_{ui} is the target similarity to be learned. This term can be used to model the following:

- Rating similarities: For example, the user u gives a rating for an item i and this rating would be set as the target similarity r_{ui}. (A high rating will produce a large similarity and vice versa.) This way, we can predict how any user would rate an item, by simply computing the similarity as the dot product after training is complete. This is what we call "Prediction";
- Purchase pattern based: The target similarity r_{ui} can be proportional to the amount of times the user u purchased this item i in previous purchases.

The third term corresponds to item–item similarity, where the dot product $x_i^T x_{i'}$ is the predicted similarity of items i and i', and $o_{ii'}$ is the target similarity to be learned. This term can be used to model the following:

- Content-based: For example, items i and i' that are similar or related product-types and have similar features and/or have a similar price range would have a high target similarity $o_{ii'}$;
- Purchase pattern based: The target similarity $o_{ii'}$ is proportional to the number of times the items i and i' are purchased together by some user. This is what we call "Recommendation".

Please note that the weighting functions w_{ij} should be set to be proportional to the importance of the term considered. For example, CF-related terms, such as Prediction, Recommendation, or Term 2's purchase pattern, should be weighted higher than the other terms.

Let us for now consider Equation (1), as this is the general form that encompasses Equation (2) and other forms. The objective of the approach is to find the vectors x_i that minimize the error defined in (1). The goal is to have these vectors express the desired similarities, that is, the dot products would be as close as possible to the given similarity values s_{ij}. The norm bound ($x^T x = 1$) can be incorporated as a Lagrange multiplier.

Solving this optimization problem all at once is hard, as this is a fourth order multi-dimensional polynomial equation. However, if we consider a specific item or user's vector x_j and fix the others, it becomes a quadratic equation (with a Lagrange multiplier term) and therefore has an analytic solution. We exploit this fact and develop a relaxation algorithm such that one vector is learned at a time while fixing the others. After that, another vector is learned (again while fixing the others), and so on until all the vectors are learned. The algorithm does not stop at this point but a few more cycles are performed, going through all vectors one-by-one, until convergence. The convergence is marked such that the decrease in error is no longer significant from one cycle to the other. Here is a short description of the steps. For more details refer to [41], including a proof of convergence. The steps of the training algorithm are summarized as follows:

1. Generate the initial vectors x_i randomly, normalizing them as unit length.
2. For $i = 1$ to N perform the following:
3. Fix all x_j except x_i and update x_i as follows:

$$x_i = A^{-1} b, \tag{3}$$

where

- A is a matrix defined as:

$$A = \sum_{j \neq i} w_{ij} x_j x_j^T - \lambda I, \tag{4}$$

- b is a vector defined as:

$$b = \sum_{j \neq i} w_{ij} x_j s_{ij}, \tag{5}$$

- λ is the scalar Lagrange multiplier, evaluated using a simple one-dimensional bisection search so that: $x_i^T x_i = 1$.

4. Repeat Steps 2–3 until the vectors converge.

Since a vector may be involved in more than one transaction, that is, user u rates a number of items or item i is rated by a number of users, there is a competition between vectors such that the vectors are placed optimally with respect to each other, satisfying the similarities of the collective set of transactions. After training is complete, the similarity between any two vectors x_i and x_j can be simply obtained using their dot product.

In order to have some focus in the work and avoid straying into too many possible avenues of application of this vector embedding, we will concentrate in this work on the "Prediction" and the "Recommendation" formulation (Terms 2 and 3 in (2)). These are described next.

3.2. Prediction

In this formulation, we seek to predict the similarity scores between a user u and item i. An example is the prediction of how user u would rate item i. This would be accomplished by observing the pattern of ratings of the concerned user and other similar users and items. This problem corresponds to Term 2 in Equation (2), which tackles user–item similarity terms, where vectors x_i correspond to the items, and vectors x_u correspond to the users.

The system uses a corpus of users' transactions where a transaction specifies that user u rated item i with the rating r_{ui}. The ratings are typically available as a number on a scale of 1 to 5 or 1 to 10 (which are translated in a normalized fashion to be in the range from 0 to 1), or they are "like" and "dislike" (for which r_{ui} is taken as 1 and -1 respectively). Available ratings are used in the error function, and missing ratings (which is the great majority) will be estimated after training, using the dot product between the corresponding vectors. The error function is given by:

$$E = \sum_{u=1}^{U} \sum_{i=1}^{N} w_{ui} \left[x_u^T x_i - r_{ui} \right]^2 \quad \text{subject to} ||x|| = 1, \forall u, i, \tag{6}$$

where:

- x_u is the user's vector.
- x_i is the item's vector.
- r_{ui} is the rating user u assigned to item i, fed to the system as the rating scaled value where the maximum value is 1.
- w_{ui} is a weighing coefficient that constraints the system, which is designed to put different emphases on different ratings terms. In most practical cases we take w_{ui} as constant or one, unless certain ratings are deemed more influential.
- U is the number of users.
- N is the number of items.

The algorithm to minimize this error function (6) is similar to that described in the last subsection. The proposed approach will therefore have the vectors for the users and the vectors for the items side-by-side in the same space. A vector pertaining to a user will be close to one pertaining to an item if the user likes the item. Like-minded users will also be drawn together, even though not explicitly trained for that. This is because common item preferences will act as the glue that pulls the users' vectors together. The same applies for vectors of similar items.

3.3. Recommendation

Recommendation is the problem of recommending items to the user based on the purchasing pattern of the user and other similar users. This corresponds to Term 3 in the formulation (2). The system uses a corpus of users' transactions where a transaction

specifies the items purchased by a user. Representative vectors are to be learned for items such that items occurring together in transactions are placed close to each other in the vector space. The more transactions that include an item pair, the closer the vectors of these items are placed with respect to each other. This conveys the fact that these items are more likely to be purchased together following a purchase pattern. For example, items such as "keyboard" and "mouse" are more likely to be purchased together than items such as "keyboard" and "cap". After training is complete, the similarity between vectors is determined by computing the dot product.

The input to the model is a corpus of users' transactions. The method begins by parsing transactions such that item pairs are extracted. Each item pair is assigned a score proportional to the number of transactions where this item pair occurs. The error function is given by:

$$E = \sum_{i=1}^{N} \sum_{i'=1}^{N} w_{ii'} \left[x_i^T x_{i'} - o_{ii'} \right]^2 \quad \text{subject to} ||x_i|| = 1, i = 1, \ldots, N, \tag{7}$$

where:

- x is the item's vector.
- $o_{ii'}$ is a number proportional to the number of transactions where the items i and i' occur together, fed to the system as a scaled value where the maximum value is 1.
- $w_{ii'}$ is a weighing coefficient whose purpose is to put different emphases on different terms. Typically we take them to equal one.
- N is the number of items.

Vectors are learned for the items such that the dot products between such vectors are as close as possible to the given similarity values $o_{ii'}$. The algorithm for training such a model is similar to the one described in the previous subsection.

To recommend items for a given user, we use "Average pooling". A vector is computed representing the user's taste. This vector depends on the user's transactions' history. In our model, this user representative vector is computed using average pooling on vectors of the items that are present in this user's transactions' history, as in (8). The user's vector adequately represents the user's taste as it is placed in a middle ground between the different items that this user purchased.

$$Vec_{user} = \frac{1}{T} \sum_{t=1}^{T} itemvec_t, \tag{8}$$

where

- $itemvec$ is the vector of the item.
- T is the number of items in the user's history.

Next, recommendations for items are performed by obtaining the items that are close to the user's vector. This can be obtained by computing the dot product between the user's vector and the item's vector. Then, items that have the largest dot products are returned as the system's recommendation. Figure 1 summarizes this method.

Algorithm: Average Pooling RS

Inputs: Item Vectors (I), Items in user's history (I_{user}), Number of items to recommend (k)

Outputs: Items recommended for *user* (I_{rec})

Compute the user vector Vec_{user} as in Equation (8)
For each *itemvec* in I:
    Compute dot product between Vec_{user} and *itemvec*
SortedDP = Sort the obtained dot products descendingly
I_{rec} = first k items in *SortedDP*

Figure 1. Recommender Systems Average Pooling Method.

4. Evaluation and Results

4.1. Prediction

The proposed models are tested on some of the most popular datasets in the Recommender Systems field: "MovieLens" [42]: "MovieLens 100 K" and "MovieLens 1 M". Moreover, we have also tested the "ModCloth" [43] dataset, which is a sparser dataset. Sparsity here refers to the fact that each user rated only a small number of items. Furthermore, we evaluated the performance on "Amazon: Magazine_Subscriptions" (Amazon MS) [44]. This dataset has a relatively small number of reviews/ratings, thus the results give us insight into how the approaches perform in such cases. Table 1 outlines the different aspects for each dataset used. For all these datasets, the ratings provided are on a scale from 1 to 5.

Table 1. Datasets Aspects.

	Dataset			
Aspect	**MovieLens 100 K**	**MovieLens 1 M**	**ModCloth**	**Amazon MS**
Total number of ratings	100,000	1,000,209	99,893	2375
Number of users	943	6040	44,784	348
Number of items	1682	3706	1020	157
Maximum number of ratings by one user	737	2314	250	30
Minimum number of ratings by one user	20	20	1	3
Mean number of ratings by one user	106	165.6	2.2	6.8
Median number of ratings by one user	65	96	1	6

For each dataset, we adopted the following experimental procedure:

- Applied 5-fold cross-validation.
- Five experiments were performed such that four folds were used as the training set and the remaining fifth fold was used as the test set, where in each experiment the folds were rotated as in the standard K-fold validation procedure.
- The training set consisted of the transactions fed to the vectors' learning algorithm where users' and items' vectors of dimension 50 were learned for all the used datasets except the "Amazon MS" dataset where the vector's dimension is only 10. The ratings in the given transactions were first normalized by dividing by five before being fed to the algorithm (so the maximum rating value became 1 and the minimum rating value became 0.2).
- The test set was the set used for performance evaluation by predicting the rating value for each transaction in the set. The prediction was performed by calculating the dot product between the user's vector and the item's vector involved in the transaction. Then, the obtained predicted rating was restored to the original ratings' scale (for example by multiplying by 5).

- For each test set, two evaluation metrics were computed: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) [45].
  $RMSE = \sqrt{(\frac{1}{|Q|}) \sum_{(u,i)\epsilon Q} (r_{ui} - \hat{r}_{ui})^2}$
  $MAE = \frac{1}{|Q|} \sum_{(u,i)\epsilon Q} |r_{ui} - \hat{r}_{ui}|$
  , where Q is the test set, r_{ui} is user u's true rating given to item i and $\hat{r}_{ui}$ is the predicted rating of the Recommender System.
- The mean value for each metric RMSE and MAE was computed for all the five folds. This is the value recorded in the results.

Furthermore, we compared the obtained results with the benchmark methods introduced in [46,47]. These methods are described as follows:

- Singular Value Decomposition (SVD) [48]:
  SVD is the matrix factorization technique where the $m \times n$ ratings matrix R (that includes users' ratings for the items) is factored into three matrices as follows:
  $R = U.S.V^T$, where U and V are orthogonal matrices of sizes $m \times r$ and $n \times r$ respectively.
  r is the rank of the rating matrix R.
  S is an $r \times r$ diagonal matrix that includes all singular values of matrix R stored in descending order.
  The SVD approach is a kind of dimensionality reduction method and it provides the best lower rank approximations for the original matrix R. The matrix S can be reduced to S_k by including only the largest k diagonal values. Consequently, U and V are reduced, obtaining $R_k = U_k \cdot S_k \cdot V_k'$. For user u and item i, the rating prediction is computed as the dot product between the u-th row of $U_k S_k^{1/2}$ and i-th column of $S_k^{1/2} V_k'$ followed by adding the user average back.
- SVD++ [49]:
  This is an extension of the SVD approach, where implicit feedback is incorporated. An implicit feedback refers to the fact that a user u rated an item i without regard to the value of the given rating.
- Non-negative Matrix Factorization (NMF) [50]:
  This approach is similar to SVD; however, stochastic gradient descent (SGD) is used for the optimization procedure. SGD is carried out with a specific choice of step size ensuring the non-negativity of factors given that the factors' initial values are positive.
- Slope One [51]:
  The technique relies on the SlopeOne algorithm, where predictors are used to compute the average difference between the ratings of one item and another for users who rated both. The rating prediction is defined as:
  $\hat{r_{ui}} = \mu_u + \frac{1}{card(R_i(u))} \sum_{j\epsilon R_i(u)} dev(i,j)$
  , where μ_u is the average of ratings given by user u.
  $R_i(u)$ is the set of relevant items: the set of items j rated by user u that also have at least one common user with i.
  $card$ is the number of elements in a set.
  $dev(i,j)$ is the average deviation of ratings of items i with respect to items j, given by:
  $dev(i,j) = \frac{1}{card(U_{ij})} \sum_{u\epsilon U_{ij}} r_{ui} - r_{uj}$
- K-Nearest Neighbors (k-NN) [24]:
  The predictions of ratings are performed based on the ratings given by the user's nearest-neighbors. In such models, known as memory-based models, the stored ratings are directly used in the prediction process.
- Centered k-NN [24]:
  This is a k-NN technique that takes into account the average ratings of each user to provide a corrective action to the prediction.

- k-NN Baseline [52]:
  This is a k-NN technique that incorporates baseline estimates. These estimates account for data tendencies such as the likelihood that some users give higher ratings than other users or that some items receive higher ratings than other items.
- Co-Clustering [53]:
  Clusters are built simultaneously for users and items. The predictions are based on the average ratings of the co-clusters (user–item neighborhoods). Formally, rating prediction is given as:
  $\hat{r_{ui}} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i})$
  , where $\overline{C_{ui}}$ is the average rating of co-cluster C_{ui} for user u and item i.
  $\overline{C_u}$ is the average rating of user u's cluster.
  $\overline{C_i}$ is the average rating of item i's cluster.
- Baseline [52]:
  In this technique, a simple baseline estimate is computed for the rating predictions, taking into account the aforementioned data tendencies. Formally, rating prediction is defined as:
  $\hat{r_{ui}} = b_{ui} = \mu + b_u + b_i$
  , where μ is the overall average rating.
  Parameters b_u and b_i are determined by solving a least squares problem.

Table 2 shows the errors of the proposed models as well as the competing approaches, using the five fold validation test. Missing entries mean that the system crashed and was not able to produce any results.

Table 2. Prediction Recommender Systems Results.

	MovieLens 100 K		**MovieLens 1 M**		**ModCloth**		**Amazon MS**	
Approach	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**
Sphere (our proposed method)	0.923	0.738	**0.857**	0.684	1.165	**0.783**	1.03	**0.605**
SVD	0.934	0.737	0.873	0.686	1.056	0.828	0.971	0.724
SVD++	**0.92**	**0.722**	0.862	**0.673**	1.06	0.829	**0.941**	0.689
NMF	0.963	0.758	0.916	0.724	1.2	0.948	1.035	0.759
Slope One	0.946	0.743	0.907	0.715	1.150	0.87	0.996	0.637
k-NN	0.98	0.774	0.923	0.727	-	-	1.156	0.771
Centered k-NN	0.951	0.749	0.929	0.738	-	-	0.958	0.611
k-NN Baseline	0.931	0.733	0.895	0.706	-	-	1.038	0.699
Co-Clustering	0.963	0.753	0.915	0.717	1.137	0.856	0.986	0.619
Baseline	0.944	0.748	0.909	0.719	**1.041**	0.82	1.002	0.766
Random	1.514	1.215	1.504	1.206	1.398	1.06	1.437	1.059

From Table 2, one can observe that among a total of 11 methods, our approach obtained the best RMSE for the MovieLens 1 M dataset, the second best RMSE (0.923) for the MovieLens 100 K dataset (being very close to the best RMSE (0.92) obtained by the SVD++ method) and the best MAE for ModCloth and Amazon MS datasets. Overall, the best two models are our proposed Sphere model and SVD++. We suggest these models be the candidates to be tested when designing a recommender system and possibly using these approaches in a hybrid model.

One can observe that all the approaches produced higher errors for the ModCloth dataset. This is due to the fact that ModCloth is a sparser dataset. It has many fewer reviews per user than the other datasets. Another point worth mentioning is that for all KNN-based algorithms, the RAM (12 GB) crashed while running for ModCloth dataset, that is, such algorithms consume huge memory sizes and do not scale well when increasing the number of users as in this dataset. Furthermore, our algorithm converges in a relatively low number of cycles. The average number of cycles are 10, 23, 11 and 7 for MovieLens 100 K, MovieLens 1 M, ModCloth and Amazon MS datasets, respectively. The algorithm's computational complexity is detailed in Appendix A.

Our algorithm possesses other benefits as well. It does not suffer as much as SVD++ and k-NN from memory problems. In the latter two approaches, a huge user–item ratings

matrix has to be stored, even though it is sparse. In our approach, in contrast, only the user–item review pairs have to be stored and used in the algorithm, which provides memory savings. Another advantage is that our algorithm is incremental in nature. So it can add any extra user or item without having to retrain the entire vectors' space. For example, it can begin the retraining process from the previous state of the vectors instead of starting with random vectors and retraining entirely from scratch. This incremental feature may be lacking in the SVD-type methods and it is not clear how one can start from a previous solution if extra data arrive.

However, the major advantage of our approach is its versatility. One can add user–user and item–item similarities to the existing user–item similarities. This makes the approach more general and allows us to add content-based, demographic and other similarities to create one large model that takes into account all available information. On the other hand, a model such as SVD is designed to handle only user–item similarities.

4.2. Recommendation

For the recommendation problem we used the "Online Retail" dataset [54] as a case study. This dataset includes customers' transactions for an online retailer that focuses on selling all-occasion gifts. These transactions occurred between 1 December 2010 and 9 December 2011. The dataset contains 541,909 instances where each instance has the following attributes:

- InvoiceNo: invoice number, which is a unique number identifying each transaction.
- StockCode: item/product code, which is a unique number identifying each product/item.
- Description: item/product descriptive name.
- Quantity: quantity of item/product per transaction.
- InvoiceDate: day and time of transaction.
- UnitPrice: price of item/product.
- CustomerID: customer number, which is a unique number identifying each customer.
- Country: country where customer resides.

We parsed the data to obtain the unique customers, there was a total of 4372 customers. Next, we divided the data into training and test sets where 80% and 20% of the customers constituted the training and test sets, respectively. The training set was used to provide input to the proposed item embedding algorithm where the items' vectors of dimension 20 were learned. After obtaining vectors for items, we recommended items for customers. For testing, the most recent transaction(s) of each customer were excluded from any computations in order to be used for assessment. These transactions were not regarded as part of the customer's history but were regarded as the customer's future transactions to be compared against the recommended items by the system. The number of such transactions was taken as one (i.e., the most recent customer's transaction) in one experiment, and two (i.e., the most recent two customer's transactions) in another experiment. These numbers were chosen as indicated in this dataset. The number of transactions per customer had an average of seven and a median of four. Finally, to assess the performance, the items in the excluded transactions were compared with the recommended items to determine the hit rate, which is defined as:

$$HitRate = \frac{\sum_{c \epsilon C} HitRate_c}{C}, \tag{9}$$

where

- $HitRate_c$ equals 1 if there is a hit for customer c and 0 otherwise. A hit is determined when at least one item in the set of recommended items matches one of the items in the true most recent transaction(s) of customer c.
- C is the total number of customers in the test set.

Moreover, we built another model that learns vectors (also of dimension 20) for items using the word2vec embedding approach [16]. This model was trained such that the

customer's transaction was regarded as the sentence while the item was regarded as the word. Therefore, the customers' transactions were fed to the model the same way sentences were fed during training word2vec on text corpora. The aim of building such a model was to compare the performance of the two models: using our sphere embedding approach and using the word2vec embedding approach. We performed a number of experiments changing k, which represents the number of items to recommend. Table 3 shows the performance (hit rate) of both models (larger numbers mean better recommendation).

Table 3. The Hit Rate of the Recommendation Results for the Proposed Sphere Model and Word2vec.

	k = 1	k = 5	k = 10	k = 15	k = 20
			Sphere (our approach)		
Future Transactions = 1	0.186	0.435	0.524	0.602	0.665
Future Transactions = 2	0.268	0.544	0.639	0.704	0.756
			word2vec		
Future Transactions = 1	0.098	0.306	0.447	0.516	0.573
Future Transactions = 2	0.14	0.404	0.549	0.626	0.685

From Table 3, the following observations can be stated:

- As k increases, the "HitRate" increases, because there would be more chance that the user's purchase is included in the recommended basket of items.
- Our approach outperforms word2vec for all k values.

A point that is worth mentioning is that, when using the word2vec approach, the user's positive taste can be represented, but not the user's negative taste. This is because vectors are learned for items in the users' history transactions. The question here is how such model can represent the user's negative taste, that is, the items that the user marks as uninteresting or marks as a dislike. This negative taste also plays a great role in the items to be recommended. The word2vec model has no way to link the user's positive and negative tastes in one comprehensive model. On the other hand, our model can represent this negative taste since the learned vectors for items are embedded on the sphere. The polar nature of the sphere allows the placement of the uninteresting or disliked items at opposite poles of the sphere. Therefore, our model can comprehensively represent positive as well as negative tastes.

The word2vec possesses another serious limitation. It is applicable to the "Recommendation" problem only. This is because it is designed to be trained on context data. This means that it is given a number of items in a basket of purchases (or a number of words in a sentence) and it is trained to predict an item in the group (or a word in the midst of the sentence). This makes it difficult to use for the "Prediction" problem, analyzed in the previous subsection.

5. Conclusions

This work tackles a new field of items/users embeddings where vectors are learned for items/users as opposed to words. We develop new models for recommender systems that are based on embedding items/users as vectors on a sphere. The method is a simple relaxation algorithm where representative vectors are learned. These vectors encode the underlying features for items/users. The developed approach is applicable to different formulations of Recommender Systems, and in particular is tested on the "prediction" and the "recommendation" problems. Moreover, the approach can model a number of interactions including item to item, user to item and user to user. The proposed approach can also include content-based and demographic similarities, which are typically hard to incorporate in a model. The generality of using any target similarity can open the door for other types, including using other input domains such as textual reviews, browsing patterns, and so forth. These topics have experienced little application in recommender systems and could

be promising to consider. The proposed models are successfully applied to real life datasets including "MovieLens", "ModCloth", "Amazon: Magazine_Subscriptions" and "Online Retail" as case studies, and they show competitive results. Moreover, the diversity of the used datasets (dense vs sparse and large number of ratings vs small number of ratings) provides an analytical insight into the performance of different recommender systems' approaches.

Author Contributions: Conceptualization, S.R. and A.F.A.; methodology, S.R., A.F.A. and S.S.; software, S.R.; validation, S.R; formal analysis, S.R., A.F.A., and S.S.; investigation, S.R., A.F.A., and S.S; resources, S.R.; data curation, S.R.; writing—original draft preparation, S.R.; writing—review and editing, S.R. and A.F.A.; visualization, S.R.; supervision, A.F.A. and S.S.; project administration, A.F.A. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Complexity Analysis

Let N be the number of vectors (number of users plus the number of items). Also, let M be the dimension of the vector and let R be the average number of ratings per user (or average number of ratings per item if we are dealing with items). The computations involve the following for the basic method:

(a) Computing the matrix A according to Equation (4): The complexity is $O(M^2R)$. Note that the summation is over existing ratings for user i, rather than over all items, so it is not a large summation.
(b) Computation of b according to Equation (5) is $O(MR)$.
(c) Computation of Equation (3) is $O(M^3)$.
(d) Computation of λ through the bisection search is $O(M^3B)$, where B is the number of bisection search steps. Typically B is around 20.
(e) Each of these computations is repeated N times (the total number of vectors or users and items).
(f) Typically N is very large (several thousands), while M, the dimension of each vector is usually small (we take it as 10, 20 or 50). Using this fact and retaining only the largest terms, we get the amount of computation per cycle as approximately equal to $O(NM^2R + NM^3B)$.
(g) The term $N \times R$ (number of users times number of ratings per user plus number of items times number of ratings per item) is essentially twice the total number of ratings (twice because we count that from the users' side and also from the items' side).
(h) So the total complexity per cycle is $O(KM^2 + NM^3B)$ where K is the total number of ratings.
(i) We have also implemented an accelerated version of the part that searches and computes λ using eigenvalue decomposition. This relieves us from solving the linear equations B times. The complexity of the λ search using bisection becomes $O(NM^3)$ instead of $O(NM^3B)$ per cycle, i.e., better than the previous basic algorithm. So the total complexity per cycle becomes $O(KM^2 + NM^3)$. The details are not given here to avoid distraction to algorithmic issues.
(j) For the off-line algorithm, i.e., applying the algorithm from scratch to train all vectors the complexity equals the preceding times the number of cycles C, i.e., $O(CKM^2 + CNM^3)$. Typically C is around 10 to 20. For the update phase, i.e receiving an extra rating or user and hence implementing only one cycle, the complexity becomes $O(M^2R + M^3)$ (for the eigenvalue λ search method).
(k) The terms C, M, and B in the complexity formula are usually small, e.g., 10 to 50. The only impactful terms are K (the total number of ratings) and N (the total number of users plus items). These can for example be 100,000 or a million. What

is important in the complexity analysis is that these terms (i.e., the K and the N) turned out to appear as linear terms, not more.

References

1. Ortega, F.; González-Prieto, Á. Recommender Systems and Collaborative Filtering. *Appl. Sci.* **2020**, *10*, 7050. [CrossRef]
2. Fayyaz, Z.; Ebrahimian, M.; Nawara, D.; Ibrahim, A.; Kashef, R. Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities. *Appl. Sci.* **2020**, *10*, 7748. [CrossRef]
3. Ricci, F.; Rokach, L.; Shapira, B. Recommender systems: Introduction and challenges. In *Recommender Systems Handbook*; Springer: Boston, MA, USA, 2015; pp. 1–34.
4. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web, Hong Kong, China, 1–5 May 2001; pp. 285–295.
5. Sohail, S.S.; Siddiqui, J.; Ali, R. Classifications of Recommender Systems: A review. *J. Eng. Sci. Technol. Rev.* **2017**, *10, 132–15* . [CrossRef]
6. Isinkaye, F.O.; Folajimi, Y.; Ojokoh, B.A. Recommendation systems: Principles, methods and evaluation. *Egypt. Inform. J.* **2015**, *16*, 261–273. [CrossRef]
7. Gómez-Pulido, J.A.; Durán-Domínguez, A.; Pajuelo-Holguera, F. Optimizing Latent Factors and Collaborative Filtering for Students' Performance Prediction. *Appl. Sci.* **2020**, *10*, 5601. [CrossRef]
8. Yang, Z.; Wu, B.; Zheng, K.; Wang, X.; Lei, L. A survey of collaborative filtering-based recommender systems for mobile internet applications. *IEEE Access* **2016**, *4*, 3273–3287. [CrossRef]
9. Zhang, S.; Yao, L.; Sun, A.; Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.* **2019**, *52*, 1–38. [CrossRef]
10. Bobadilla, J.; Alonso, S.; Hernando, A. Deep Learning Architecture for Collaborative Filtering Recommender Systems. *Appl. Sci.* **2020**, *10*, 2441. [CrossRef]
11. Shafqat, W.; Byun, Y.C.; Park, N. Effectiveness of Machine Learning Approaches Towards Credibility Assessment of Crowdfunding Projects for Reliable Recommendations. *Appl. Sci.* **2020**, *10*, 9062. [CrossRef]
12. Shafqat, W.; Byun, Y.C. Incorporating Similarity Measures to Optimize Graph Convolutional Neural Networks for Product Recommendation. *Appl. Sci.* **2021**, *11*, 1366. [CrossRef]
13. Sulikowski, P.; Zdziebko, T. Horizontal vs. Vertical Recommendation Zones Evaluation Using Behavior Tracking. *Appl. Sci.* **2021**, *11*, 56. [CrossRef]
14. sulikowski, P.; Zdziebko, T.; Coussement, K.; Dyczkowski, K.; Kluza, K.; Sachpazidu-Wójcicka, K. Gaze and Event Tracking for Evaluation of Recommendation-Driven Purchase. *Sensors* **2021**, *21*, 1381. [CrossRef]
15. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
16. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems—Volume 2 (NIPS'13), Lake Tahoe, NV, USA, 5–8 December 2013; Curran Associates Inc.: Red Hook, NY, USA, 2013; pp. 3111–3119.
17. Koren, Y.; Bell, R.; Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
18. Yang, X.; Guo, Y.; Liu, Y.; Steck, H. A survey of collaborative filtering based social recommender systems. *Comput. Commun.* **2014**, *41*, 1–10. [CrossRef]
19. Girase, S.; Mukhopadhyay, D. Role of matrix factorization model in collaborative filtering algorithm: A survey. *arXiv* **2015**, arXiv:1503.07475.
20. Ma, H.; Zhou, D.; Liu, C.; Lyu, M.R.; King, I. Recommender systems with social regularization. In Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, Hong Kong, China, 9–12 February 2011; pp. 287–296.
21. Guan, X.; Li, C.T.; Guan, Y. Matrix factorization with rating completion: An enhanced SVD model for collaborative filtering recommender systems. *IEEE Access* **2017**, *5*, 27668–27678. [CrossRef]
22. Mitroi, B.; Frasincar, F. An elastic net regularized matrix factorization technique for recommender systems. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 May–3 April 2020; pp. 2184–2192.
23. Melville, P.; Sindhwani, V. Recommender systems. *Encycl. Mach. Learn.* **2010**, *1*, 829–838.
24. Desrosiers, C.; Karypis, G. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*; Springer: Boston, MA, USA, 2011; pp. 107–144.
25. Bobadilla, J.; Ortega, F.; Hernando, A.; Gutiérrez, A. Recommender systems survey. *Knowl.-Based Syst.* **2013**, *46*, 109–132. [CrossRef]
26. Martín-Vicente, M.I.; Gil-Solla, A.; Ramos-Cabrer, M.; Pazos-Arias, J.J.; Blanco-Fernández, Y.; López-Nores, M. A semantic approach to improve neighborhood formation in collaborative recommender systems. *Expert Syst. Appl.* **2014**, *41*, 7776–7788. [CrossRef]
27. Subramaniyaswamy, V.; Logesh, R. Adaptive KNN based recommender system through mining of user preferences. *Wirel. Pers. Commun.* **2017**, *97*, 2229–2247. [CrossRef]

28. Ayyaz, S.; Qamar, U. Improving collaborative filtering by selecting an effective user neighborhood for recommender systems. In Proceedings of the 2017 IEEE International Conference on Industrial Technology (ICIT), Toronto, ON, Canada, 22–25 March 2017; pp. 1244–1249.
29. Akama, S.; Kudo, Y.; Murai, T. Neighbor Selection for User-Based Collaborative Filtering Using Covering-Based Rough Sets. In *Topics in Rough Set Theory. Intelligent Systems Reference Library*; Springer: Cham, Switzerland, 2020; Volume 168, pp. 141–159.
30. Gutiérrez, L.; Keith, B. A systematic literature review on word embeddings. In *International Conference on Software Process Improvement*; Springer: Cham, Switzerland, 2018; pp. 132–141.
31. Grbovic, M.; Radosavljevic, V.; Djuric, N.; Bhamidipati, N.; Savla, J.; Bhagwan, V.; Sharp, D. E-commerce in your inbox: Product recommendations at scale. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, 10–13 August, 2015; pp. 1809–1818.
32. Vasile, F.; Smirnova, E.; Conneau, A. Meta-prod2vec: Product embeddings using side-information for recommendation. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 225–232.
33. Liang, D.; Altosaar, J.; Charlin, L.; Blei, D.M. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 59–66.
34. Ozsoy, M.G. From word embeddings to item recommendation. *arXiv* **2016**, arXiv:1601.01356.
35. Krishnamurthy, B.; Puri, N.; Goel, R. Learning vector-space representations of items for recommendations using word embedding models. *Procedia Comput. Sci.* **2016**, *80*, 2205–2210. [CrossRef]
36. Pennington, J.; Socher, R.; Manning, C. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October, 2014; pp. 1532–1543.
37. Barkan, O.; Koenigstein, N. Item2vec: Neural item embedding for collaborative filtering. In Proceedings of the 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), Vietri sul Mare, Italy, 13–16 September 2016; pp. 1–6.
38. Yang, Z.; He, J.; He, S. A collaborative filtering method based on forgetting theory and neural item embedding. In Proceedings of the 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 24–26 May 2019; pp. 1606–1610.
39. Lu, N.; Ohsawa, Y.; Hayashi, T. Learning Sequential Behavior for Next-Item Prediction. In Proceedings of the Annual Conference of JSAI 33rd Annual Conference, Niigata, Japan, 4–7 June 2019; p. 3B4E203._3B4E203. [CrossRef]
40. Barkan, O.; Caciularu, A.; Katz, O.; Koenigstein, N. Attentive Item2vec: Neural attentive user representations. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 3377–3381.
41. Rizkallah, S.; Atiya, A.F.; Shaheen, S. A Polarity Capturing Sphere for Word to Vector Representation. *Appl. Sci.* **2020**, *10*, 4386. [CrossRef]
42. Harper, F.M.; Konstan, J.A. The movielens datasets: History and context. *Acm Trans. Interact. Intell. Syst.* **2015**, *5*, 1–19. [CrossRef]
43. Wan, M.; Ni, J.; Misra, R.; McAuley, J. Addressing marketing bias in product recommendations. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February, 2020; pp. 618–626.
44. Ni, J.; Li, J.; McAuley, J. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November, 2019; pp. 188–197.
45. Chen, M.; Liu, P. Performance Evaluation of Recommender Systems. *Int. J. Perform. Eng.* **2017**, *13*. [CrossRef]
46. Hug, N. Surprise: A python library for recommender systems. *J. Open Source Softw.* **2020**, *5*, 2174. [CrossRef]
47. Surprise. Available online: https://github.com/NicolasHug/Surprise (accessed on 29 January 2021).
48. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. *Application of Dimensionality Reduction in Recommender System A Case study*; Technical Report; Minnesota Univ Minneapolis Dept of Computer Science: Minneapolis, MN, USA, 2000.
49. Kumar, R.; Verma, B.; Rastogi, S.S. Social popularity based SVD++ recommender system. *Int. J. Comput. Appl.* **2014**, *87*, 33–37 [CrossRef]
50. Luo, X.; Zhou, M.; Xia, Y.; Zhu, Q. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1273–1284.
51. Lemire, D.; Maclachlan, A. Slope one predictors for online rating-based collaborative filtering. In Proceedings of the 2005 SIAM International Conference on Data Mining, Newport Beach, CA, USA, 21–23 April 2005; pp. 471–475.
52. Koren, Y. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. Knowl. Discov. Data* **2010**, *4*, 1–24. [CrossRef]
53. George, T.; Merugu, S. A scalable collaborative filtering framework based on co-clustering. In Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05), Washington, DC, USA, 27–30 November 2005; p. 4.
54. Chen, D.D. Online Retail Dataset. 2015. Available online: https://archive.ics.uci.edu/ml/datasets/online+retail (accessed on 27 August 2020).