Engineering Applications of Artificial Intelligence 85 (2019) 347–356

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Collaborative filtering embeddings for memory-based recommender systems☆

Daniel Valcarce *, Alfonso Landin, Javier Parapar, Álvaro Barreiro

Information Retrieval Lab, Computer Science Department, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain

ARTICLE INFO

Keywords:
Embedding vector
User representation
Item representation
Collaborative filtering
Recommender systems

ABSTRACT

Word embeddings techniques have attracted a lot of attention recently due to their effectiveness in different tasks. Inspired by the continuous bag-of-words model, we present `prefs2vec`, a novel embedding representation of users and items for memory-based recommender systems that rely solely on user–item preferences such as ratings. To improve the performance and prevent overfitting, we use a variant of dropout as regularization, which can leverage existent `word2vec` implementations. Additionally, we propose a procedure for incremental learning of embeddings that boosts the applicability of our proposal to production scenarios. The experiments show that `prefs2vec` with a standard memory-based recommender system outperforms all the state-of-the-art baselines in terms of ranking accuracy, diversity, and novelty.

1. Introduction

The massive and growing amount of information available nowadays poses several challenges to information systems. As browsing and searching for relevant pieces of information becomes increasingly difficult, personalized information systems have gained popularity. In particular, recommender systems have proven to be effective tools in providing relevant information to users of a platform. Shen et al. (2016)

Among the different recommendation approaches, collaborative filtering provides high-quality recommendations when vast amounts of user information are available (Ning et al., 2015; Koren and Bell, 2015). Collaborative filtering approaches produce recommendations by exploiting the user–item feedback available to the system such as ratings, reviews, clicks, or purchases. These methods aim to find patterns in the feedback information and do not require extra data about either users or items. In contrast, content-based techniques rely on item metadata to provide recommendations.

We distinguish two main families of collaborative filtering techniques: model-based and memory-based approaches. Model-based systems learn a predictive model from the user–item feedback. The recommendation model is trained to produce tailored rankings of items to each user (Koren and Bell, 2015). Matrix factorization techniques are among the most successful model-based recommender systems. In contrast, memory-based (a.k.a. neighbourhood-based) approaches generate recommendations using the user–item feedback directly (Ning et al., 2015). We can distinguish between user-based and item-based recommenders in memory-based approaches. User-based systems find similarities between users (similar users are called user neighbours, and they form a user neighbourhood) to recommend items that like-minded users like. On the other hand, item-based systems estimate the preference of a user to an item based on similar items that the user liked. Likewise, we coin the term item neighbourhood to denote the set of similar items. In pure memory-based recommenders, similarities are computed using the user–item feedback solely.

Although model-based approaches usually perform better than memory-based techniques (Hu et al., 2008; Rendle et al., 2009; Cremonesi et al., 2010; Koren and Bell, 2015), they tend to be more complex since they involve training a model and tuning several hyperparameters. Additionally, those models are usually difficult to interpret. In fact, there have been some recent efforts to make model-based approaches explainable (Abdollahi and Nasraoui, 2016). In contrast, in item-based neighbourhood-based systems, we can easily justify the recommendation of a particular item by presenting the list of its neighbour items. Moreover, model-based systems should be retrained to incorporate new users, new items, or fresh user–item interactions. However, item-based recommenders can leverage precomputed item similarities to produce recommendations to new users and, likewise, user-based approaches can recommend new items with precomputed user similarities (Ning et al., 2015). To update the recommender, we only need to recompute the similarities involving new data (users, items, or

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.engappai.2019.06.020.

* Corresponding author.
E-mail addresses: daniel.valcarce@udc.es (D. Valcarce), alfonso.landin@udc.es (A. Landin), javierparapar@udc.es (J. Parapar), barreiro@udc.es (Á. Barreiro).

https://doi.org/10.1016/j.engappai.2019.06.020
Received 15 January 2019; Received in revised form 20 May 2019; Accepted 26 June 2019
Available online 5 July 2019
0952-1976/© 2019 Elsevier Ltd. All rights reserved.

user–item interactions). With most model-based techniques, we need to retrain the whole model if we are to produce good recommendations.

Collaborative filtering techniques usually represent users by the list of items they interacted with and items by the list of users who interacted with them. For example, when using explicit feedback in the form of ratings, users are represented by the ratings they issued and items by the ratings they received. Then, we can use pairwise metrics to compute similarities between pairs of users or items (Cremonesi et al., 2010; Ning et al., 2015; Valcarce et al., 2016).

Recent work in natural language processing has developed highly scalable models for learning word and document representations. After training in an extensive corpus, these models can produce dense embedding vectors of words and documents that outperform traditional sparse one-hot and bag-of-words representations in several tasks (Mikolov et al., 2013a,b; Le and Mikolov, 2014).

In this paper, we propose to leverage these successful models to construct novel representations of users and items to incorporate to the aforementioned memory-based methods. We call our proposal `prefs2vec` because it builds user or item collaborative filtering embeddings using the preferences registered in the system. We experiment with ratings datasets, but these preferences could also come from implicit feedback. We seek to produce user and item representations that can be plugged into existent memory-based recommender systems. The computation of these user and item embeddings involves training. Although these embeddings can be precomputed offline, we also propose a quick procedure for approximately updating the embeddings with a small computation cost. Moreover, the recommendation process is still very simple, explainable, and can leverage new information without retraining the whole model because we use well-known memory-based recommender algorithms.

We also propose to use a variant of dropout in the input layer to prevent overfitting and improve the performance of the model. The advantage of this type of regularization is that we can leverage existent `word2vec` continuous bag-of-words implementation to compute `prefs2vec` embeddings.

We conduct experiments on three datasets. We find that a competitive neighbourhood-based recommender provides better recommendations with our user and item representations than with the standard rating-based representation. Additionally, it also outperforms strong state-of-the-art model-based baselines. These findings support the effectiveness of our proposal. Moreover, we also propose how to update embedding representations without learning a new whole model from scratch – which requires substantially less time to train – and perform almost the same. This property is vital to provide fresh recommendations in a production environment.

2. Background and related work

In this section, we introduce the top-N recommendation task (which is the main task in recommender systems), and then we describe related work in memory-based recommender systems as well as in embedding representations.

2.1. Top-N recommendation

For each user u in the set of users $\mathcal{U}$, a recommender system produces recommendations from the set of items $\mathcal{I}$. Collaborative filtering exploits only the past user–item feedback such as ratings. We denote the rating that the user u gave to the item i by $r_{u,i}$ (this value is zero if u did not rate i). We use $\mathcal{I}_u$ to denote the set of items rated by the user u. Likewise, $\mathcal{U}_i$ denotes the set of users who rated the item i.

Recommender systems used to be evaluated as rating predictors using error-based metrics. However, it has been recognized that this task is a poor model of the recommendation process (Herlocker et al., 2004; Cremonesi et al., 2010; Bellogín et al., 2011; Gunawardana and Shani, 2015). Instead, top-N recommendation (Cremonesi et al., 2010) has emerged as a more realistic task. Top-N recommendation task consists in providing a good small ranking of N items for each user. In this way, the output of the recommendation model for user u and item i, $\hat{r}_{u,i}$, is not used as a predicted rating, but as a score for ranking purposes.

2.2. Memory-based recommender systems

Memory-based (or neighbourhood-based) collaborative filtering recommenders directly exploit the user–item feedback (Ning et al., 2015). This family of recommender systems can be classified in either user-based approaches or item-based approaches depending on whether they leverage patterns in the user-user or the item–item relationships, respectively. Item-based techniques usually work better than their user-based counterparts (Deshpande and Karypis, 2004; Cremonesi et al., 2010; Ning et al., 2015); however, they may generate less serendipitous recommendations because they tend to recommend similar items to those rated by the user (Ning et al., 2015). Also, depending on the dataset, user-based approaches can outperform item-based ones (Valcarce et al., 2016).

Memory-based techniques rely on a set of similar users/items called neighbourhood to estimate the relevance of a given item for a particular user. Although several clustering techniques can be used, the most common technique for computing neighbourhoods is k Nearest Neighbours (k-NN): find the top k most similar users/items to the target user/item according to a pairwise similarity metric (Ning et al., 2015). Once the set of candidate neighbours has been computed, the memory-based recommender usually employs the user–item feedback from the neighbourhood and the pairwise similarities to estimate the given user–item relevance (Deshpande and Karypis, 2004; Cremonesi et al., 2010; Valcarce et al., 2016).

Standard similarity metrics are based on user or item representations described by the set of user–item interactions (ratings, clicks, purchases, etc.). Pearson's correlation coefficient, cosine, or adjusted cosine are common similarity metrics for this purpose (Ning et al., 2015); however, recent work has shown that cosine similarity outperforms other metrics for top-N recommendation (Cremonesi et al., 2010; Valcarce et al., 2016).

Regarding memory-based recommenders, non-normalized neighbourhood formulations have proven to perform better for top-N recommendation (Deshpande and Karypis, 2004; Cremonesi et al., 2010). In particular, in this work, we use WSR, a neighbourhood-based recommendation technique with state-of-the-art performance (Valcarce et al., 2016). The user-based and item-based versions of WSR are presented below:

$$\hat{r}_{u,i} = \sum_{v \in V_u} s_{u,v}\, r_{v,i} \tag{1}$$

$$\hat{r}_{u,i} = \sum_{j \in J_i} s_{i,j}\, r_{u,j} \tag{2}$$

where $s_{\cdot,\cdot}$ represents the cosine similarity metric between users or items. We refer to the neighbourhood of user u by V_u. Likewise, J_i refers to the neighbourhood of item i.

2.3. Embedding representations

The most common word and text representations rely on sparse high-dimensional vectors based on one-hot and bags-of-words (BOW (Harris, 1954)) models, respectively. Nevertheless, recent neural embedding methods have provided dramatic advances in several natural language processing tasks by learning a fixed-length dense vector representation of words (Mikolov et al., 2013a,b; Pennington et al., 2014), and texts (Le and Mikolov, 2014). These models are fed with large chunks of text and compute continuous vector representations of words and texts. Mikolov et al. proposed the continuous bag-of-words (CBOW) and the skip-gram (SG) models for computing word embeddings and

developed word2vec, a tool for their efficient computation (Mikolov et al., 2013a). While the SG model aims to predict the surrounding words within a fixed window, the CBOW model predicts the actual word given the surrounding words (Mikolov et al., 2013a). GloVe is another successful model for learning word embeddings based on global matrix factorization and local context window methods (Pennington et al., 2014). On the other hand, Le and Mikolov also introduced two models for learning representations of any piece of text (Le and Mikolov, 2014). Training of these neural models used to be infeasible since the vocabulary size may be huge in real applications. However, recent advances such as hierarchical softmax and negative sampling have provided highly scalable training algorithms (Mikolov et al., 2013a,b).

2.4. Embeddings in collaborative filtering

Recently, the word2vec skip-gram model (Mikolov et al., 2013a,b) has been used to build item representations in collaborative filtering scenarios (Grbovic et al., 2015; Vasile et al., 2016; Barkan and Koenigstein, 2016). Grbovic et al. proposed prod2vec, a model that learns item embeddings for product recommendation (Grbovic et al., 2015). This algorithm takes as input email receipts and builds an embedding vector for each product considering the order of the purchases. This item embedding representation was studied in item-to-item and user-to-item recommendation tasks. Moreover, this model was launched on Yahoo! Mail in the form of product ads. This last fact supports the scalability and effectiveness of skip-gram models for large-scale product recommendation. Later, Vasile et al. introduced meta-prod2vec, an extension which shifts from a pure collaborative filtering scenario to a hybrid by enriching the previous model with item metadata (Vasile et al., 2016). On the other hand, Barkan and Koenigstein proposed item2vec, a model based on skip-grams that learns item embeddings from implicit feedback (music play events and purchases) (Barkan and Koenigstein, 2016). This proposal disregards temporal information by setting the window size to the length of the user profile. The embeddings were used for genre classification of items without any metadata.

Embedding vectors have also been introduced in matrix factorization approaches, which are the most common model-based recommenders (Koren and Bell, 2015). Guardia-Sebaoun et al. proposed to embed temporal information into user and item representations to build time-aware recommender systems (Guàrdia-Sebaoun et al., 2015). They modelled user actions as ordered sequences of items. They used these sequences to learn item embeddings, and they represented each user as a series of movements in the item space. Then, they applied a matrix factorization algorithm to these time-aware representations. Liang et al. on the other hand, propose CoFactor, a matrix factorization approach that jointly decomposes the user–item matrix and the shifted pointwise mutual information matrix (Liang et al., 2016). The factorization of the later matrix is equivalent to the word2vec skip-gram model trained with negative sampling (Levy and Goldberg, 2014).

Our model, prefs2vec, is designed to compute user or item embeddings that can be exploited by any neighbourhood-based recommendation approach. This model can learn embeddings from either explicit or implicit feedback and does not require temporal information. In contrast, prod2vec and meta-prod2vec are conceived for product recommendation by exploiting email receipts in order. On the other hand, the item2vec model, based on skip-grams, is designed to classify items using implicit data. Finally, the other embedding approaches used for collaborative filtering recommendation (Guàrdia-Sebaoun et al., 2015; Liang et al., 2016) are used with matrix factorization techniques rather than memory-based recommenders.

3. User and item embeddings

Word embedding models are based on the idea that words close to each other, within a fixed-length window, are similar. In this way, relations such as $man - woman \approx king - queen$ hold. Inspired by these recent advances in word embeddings, we propose an algorithm to build user or item embeddings using collaborative filtering information with explicit feedback in the form of ratings. With item embeddings, we would be able to obtain representations that capture similar relations; for instance, when producing item embeddings in the movie domain, we may find $The\ Return\ of\ the\ King - The\ Fellowship\ of\ the\ Ring \approx Return\ of\ the\ Jedi - Star\ Wars$.

We posit that co-occurring items in the same user are similar—likewise, the users who rate the same item share some commonalities. In our proposal, we do not assume that we have temporal information and, thus, the items in a user profile do not follow a particular order—we leave this avenue of research for future work. In this section, we describe how to compute these user and item embeddings and how to use them in conjunction with memory-based recommender systems to produce valuable recommendations.

We can distinguish between user-based and item-based approaches in the family of memory-based recommender systems. Given a user-based approach, we can easily derive the item-based one and vice versa: we only need to swap user and items in the formulation. For this reason and for the sake of brevity, we will only present the item-based formulation of the collaborative filtering embeddings. The user-based counterpart is analogous.

To compute collaborative filtering embeddings, we propose to adapt the word2vec continuous bag-of-words (CBOW) model to collaborative filtering top-N recommendation in contrast to previous work on the skip-gram model for different recommendation tasks (Grbovic et al., 2015; Vasile et al., 2016; Barkan and Koenigstein, 2016; Liang et al., 2016). The CBOW model seeks to predict a particular word given its context which is defined by a fixed-length window centred on the occurrence of that word in the document (Mikolov et al., 2013a). In the item-based collaborative filtering scenario, items play the role of words, and user profiles are analogous to documents. Therefore, we aim to predict a particular item given items in its context.

We chose to use the CBOW model for several reasons. First, its superior efficiency compared to the skip-gram model (Mikolov et al., 2013a). Additionally, we think that the CBOW model is more suitable for recommending than the skip-gram model because we want to predict which item is useful given a context which is a user profile. Instead, the skip-gram model would try to predict a suitable context for a particular item which is a more unnatural task in recommender systems. Moreover, recommender systems usually have to deal with sparsity and cold start problems. For this reason, the CBOW model is more adequate because it averages the input context producing a smoothed probability estimate that works better than the skip-gram model when there is not enough information.

Fig. 1 presents the diagram of the architecture of the prefs2vec, inspired in the CBOW neural model and adapted to item-based collaborative filtering. The network with one hidden layer learns from a set of training samples that consists of a context and a target item. The context is just the set of items rated by a user who also rated the target item. Therefore, for each user, we can create $|I_u|$ training samples, i.e., one sample for each item the user rated. Following this notation, for a user u and an item i, the input layer consists of the input context vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{|I_u|}\}$ corresponding to all the items that the user u rated with the exception of i. Each vector is a one-hot encoded representation of the context items. The one-hot representation of an item j is a vector $\{x_{j1}, \ldots, x_{j|I|}\}$ where all components are zero except jth component which is one. Note that $|I|$ refers to the number of items in the dataset.

The hidden layer is composed of d units with a linear activation function. The number of hidden units, d, is a hyperparameter of the

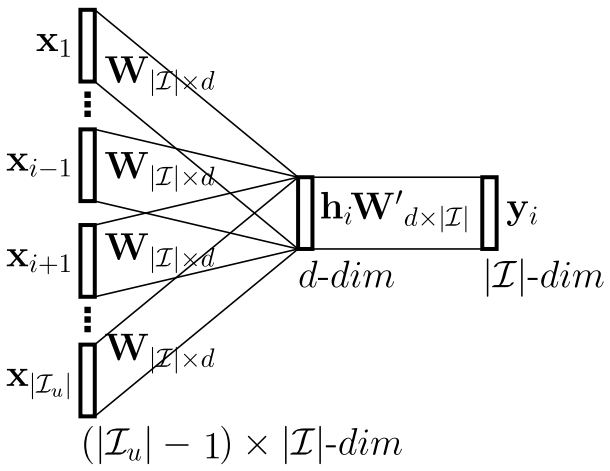

Fig. 1. Architecture of the `prefs2vec` embedding model. The input layer consists of the aggregation of $|\mathcal{I}|_u - 1$ one-hot encoded vectors of dimension $|\mathcal{I}|$, the hidden layer has d units and the output layer has $|\mathcal{I}|$ units.

embedding model. If we use the matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{I}|\times d}$ to denote the weights of the connections between the input layer and the hidden layer, each row $\mathbf{v}_j$ represents the d-dimensional input embedding vector of item j. The output of the hidden layer for item i, $\mathbf{h}_i$, is the average of the embedding vectors corresponding to the input context items weighted by the rating that the user u gave to each context item:

$$\mathbf{h}_i = \frac{\mathbf{W}}{\sum_{j\in\mathcal{I}_u\setminus\{i\}} r_{u,j}} \sum_{j\in\mathcal{I}_u\setminus\{i\}} r_{u,j}\,\mathbf{x}_j = \frac{\sum_{j\in\mathcal{I}_u\setminus\{i\}} r_{u,j}\,\mathbf{v}_j}{\sum_{j\in\mathcal{I}_u\setminus\{i\}} r_{u,j}} \tag{3}$$

Although this work is evaluated with explicit feedback datasets, we can easily adapt our proposal to implicit feedback by merely changing the rating in the previous equations with the particular implicit feedback such as clicks or purchases.

Finally, the output layer is composed of $|\mathcal{I}|$ units with a softmax activation function. We can gather the weights between the hidden layer and the output layer in matrix $\mathbf{W}' \in \mathbb{R}^{d\times|\mathcal{I}|}$ where each column $\mathbf{v}'_i$ is the d-dimensional output embedding vector of item i. The input of the output layer is given by $\mathbf{v}'^T_i\mathbf{h}$. The output of the network $\mathbf{y}$ is the posterior distribution of items given the context items, i.e., all the items in the user profile excluding the target item. We compute these probabilities using the softmax activation function, which is a log-linear classification model:

$$p\left(i \mid \mathcal{I}_u \setminus \{i\}\right) = \mathbf{y}_i = \frac{\exp\left(\mathbf{v}'^T_i\mathbf{h}_i\right)}{\sum_{j\in\mathcal{I}}\exp\left(\mathbf{v}'^T_j\mathbf{h}_i\right)} \tag{4}$$

If the context of a target item i is the rest of items in each of the user profiles where i occurs, the proposed neural embedding model is trained to minimize the following loss function:

$$\mathcal{L} = -\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u}\log p\left(i \mid \mathcal{I}_u \setminus \{i\}\right) \tag{5}$$

The training process of this model involves learning matrices $\mathbf{W}$ and $\mathbf{W}'$ by backpropagation. However, learning $\mathbf{W}'$ is impractical in large-scale scenarios because, for each training sample, the softmax function requires to iterate over all items (see Eq. (4)). To solve this problem, Mikolov et al. used two approximations of the softmax function: hierarchical softmax and negative sampling (Mikolov et al., 2013a,b). In this work, we use negative sampling since it provides faster training than hierarchical softmax and similar effectiveness.

The idea of negative sampling is to update only a few item vectors: the one which corresponds to the output word i as well as a small random sample of other items vectors. We refer to this sample, known as the negative sample, as $\mathcal{N}$ and its size n is a hyperparameter of the model. Using negative sampling, instead of Eq. (4), the loss function is computed as follows:

$$p_{ns}\left(i \mid \mathcal{I}_u \setminus \{i\}\right) = \mathbf{y}_i = \sigma\left(\mathbf{v}'^T_i\mathbf{h}_i\right)\prod_{j\in\mathcal{N}}\sigma\left(-\mathbf{v}'^T_j\mathbf{h}_i\right) \tag{6}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic function. In this way, the time complexity of the training will become independent of the number of items.

We experimented with input and output embeddings obtaining better results with the first ones. For the sake of brevity, hereinafter we will only work with the input embeddings (matrix $\mathbf{W}$).

3.1. Dropout regularization

When we feed the previously described network with ratings from collaborative filtering datasets, there is a point when adding more iterations to the training process degrades the effectiveness. This is a symptom of overfitting: the model is adapting too much to the training data and loses its generalization capacity. Early stopping, ℓ_2 regularization and dropout are some options to prevent overfitting. Early stopping consists in stop training the network after the performance in the validation set degrades. However, we decide to use dropout with a fixed number of iterations because it provided better effectiveness and a notable improvement in the training time in our experiments. Moreover, we did not use ℓ_2 regularization because of two reasons. On the one hand, Phaisangittisagul (2016) showed that dropout provides better performance than ℓ_2 regularization on one hidden layer networks with a high number of hidden units. On the other hand, we can leverage existent `word2vec` CBOW implementations when using dropout as we will see later on.

Dropout is a regularization technique that addresses overfitting by randomly dropping units and their connections during training (Srivastava et al., 2014). Dropout forces the neural model to learn more robust features and boosts generalization by reducing the co-adaptation among units. We apply dropout to the input layer because the behaviour of users is noisy and, thus, some of the items in a user profile may not be relevant to a particular item. This noise harms the embedding model because the training samples could have some noisy context items. By dropping some of the inputs randomly, we make our model more robust since we are taking random samples of the input contexts. In the experimental section, we present empirical results that show an increase in effectiveness due to dropout and also an important boost in efficiency. Additionally, our experiments show that dropout also prevents the performance from degrading after a certain number of iterations. The improvement in computation time is due to the deactivation of a large number of units during training.

Note that dropout is applied in the training phase to switch off randomly certain units for each training sample with probability p. Afterward, in the testing phase, all the units are used and, to account for the missing activations during training, their connections are reduced by a factor p. However, in our case, this is not necessary because we only care about the weights of matrix $\mathbf{W}$ and not about the final output of the network. Moreover, scaling $\mathbf{W}$ by a constant would only change the scale of the embeddings without any effect in the recommendation process.

3.2. Implementation details

One of the advantages of `prefs2vec` being based on `word2vec` CBOW model is the number of `word2vec` implementations available in different frameworks. Here, we describe how to leverage these existent implementations to build the `prefs2vec` model in practice. We also show how to emulate dropout in the input layer without modifying the `word2vec` implementation. In this work, we used the implementation available in the `gensim` library (Ř.ehůřek and Sojka, 2010).

The original `word2vec` CBOW model expects a text corpus and outputs word vectors. The corpus is composed of ordered sequences

of words which we will call documents, but bear in mind that they can also be paragraphs, sentences, or another group of words. The window size hyperparameter w of `word2vec` controls how many surrounding words we take as context. For example, if $w = 1$, the context of a word is the preceding and the following words in the document. Although the input context words are averaged in the original CBOW model, the order of the words in the documents still matters because this order defines the context.

To produce collaborative filtering item embeddings, we build a document representing each user in the dataset. Each document contains the identifiers of the items that form the user profile. To consider the whole user profile as the context, we set the window to the maximum user profile size, $w = w_{max}$ to cover the entire user profile. The order of the items in the profile does not matter because we take as input the average of the whole user profile. To introduce preference values into the embedding model, we can repeat items in the user profile as many times as the rating value. In this way, when we compute a simple average of the input, the result would be equivalent to the weighted averaged of Eq. (3).

To introduce the dropout effect in the input layer, we can modify the hyperparameter w. If we set w to a value lesser than the maximum length of the user profile, we would only take some of the items as context *dropping out* the rest of the input units. Additionally, to achieve a random deactivation of input context vectors, we can shuffle the user profiles at each iteration. If we combine shuffling user profiles at each iteration with setting w to an appropriate value, we will obtain a similar effect to dropout. This is a variant of the original dropout technique which drops units randomly with probability p (Srivastava et al., 2014). We found that our variant provides good results in this task (as we will see in the experimentation) and allows us to reuse existent `word2vec` CBOW implementations.

The training complexity of each training step of `prefs2vec` using d dimensions, window size w and n negative samples is given by $\mathcal{O}(d \times (w + n))$. At each training step, we only have w inputs which are multiplied by the same matrix $\mathbf{W}$ to obtain a d-dimensional vector. Note that the effect of w is upper bounded by the maximum length of the user profile. We can see that using dropout in the form of a window produces a notable improvement in the training complexity of the embedding technique. Thanks to the use of negative sampling, the softmax output is approximated using only n negative samples instead of the whole user or item set. Additionally, the number of training samples scales linearly with the number of ratings in the collection in both cases. These facts confirm the scalability of the `prefs2vec` model for producing user and item collaborative filtering embeddings in large-scale scenarios.

3.3. Recommendation with embeddings

Once we have computed the collaborative filtering embeddings with `prefs2vec`, we use these representations instead of the rating vectors to compute similarities for memory-based recommender systems. In the case of item-based recommendation techniques, the rating vector of item i has dimension $|\mathcal{U}|$. This vector is very sparse: all components are zero except the components corresponding to the users that rated the item i. In other words, only $|\mathcal{U}_i|$ components are nonzero. In general, $|\mathcal{U}_i| \ll |\mathcal{U}|$ due to the high sparsity scenarios recommender systems have to deal with. In contrast, our item-based embeddings are small dense vectors of dimension d. Computing similarities using small dense vector provides substantial performance improvements because we can leverage SIMD instructions of modern GPUs and multi-core CPUs.

We compute item neighbourhoods using k-NN with cosine similarity over the embedding representation of items:

$$s_{i,j} = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\| \|\mathbf{v}_j\|} \tag{7}$$

If we have a large-scale collection, we can make some optimizations to the neighbourhood computation phase. We can normalize the embedding vectors and, thus, eliminate the need for the denominator of Eq. (7). In this way, cosine similarity becomes a simple dot product. Moreover, if we are willing to accept a small decrease in accuracy in exchange for a huge speed-up, we can employ approximate k-NN algorithms such as NN-descent (Dong et al., 2011).

We employ the WSR algorithm (see Eqs. (1) and (2)) for computing recommendations which is a very competitive neighbourhood-based recommendation technique (Valcarce et al., 2016). We also compute the similarity between users or items using the cosine similarity over the `prefs2vec` embeddings.

Table 1
Datasets statistics.

Dataset	Users	Items	Ratings	Density
MovieLens 20M	138,493	26,744	20,000,263	0.540%
R3-Yahoo	15,400	1,000	365,703	2.375%
LibraryThing	7,279	37,232	749,401	0.277%

4. Experimental evaluation

In this section, we describe the experimental settings and present the results of the experimentation. For the sake of reproducibility, we provide the source code of our proposal.[1]

4.1. Datasets

We conduct our experiments on rating-based collections: MovieLens 20M.[2] film dataset, the R3-Yahoo![3] music dataset and the LibraryThing book dataset. We present the details of each collection in Table 1 We randomly partitioned the datasets by taking 80% of ratings of each user for training and the rest for test purposes.

4.2. Evaluation protocol

We follow the TestItems evaluation approach (Bellogín et al., 2011). For each user u, we rank all the items that have a rating by any user in the test set and were not rated by u in the training set. In contrast to others, this protocol provides a reliable assessment of the quality of the recommendations because it measures how well a recommender system discerns relevant items in the collection (Bellogín et al., 2011).

We measure the accuracy of the recommendations using Normalized Discounted Cumulative Gain (nDCG) as recommended in Valcarce et al. (2018). We use the *standard formulation* as described in Wang et al. (2013) with the ratings in the test set as graded relevance judgements. We measured diversity using the complement of the Gini index (Fleder and Hosanagar, 2009). This metric ranges from zero when a single item is recommended for every user to 1 when all the items are equally recommended among the users. Last, we assess the novelty of the recommendations using the mean self-information (MSI) (Zhou et al., 2010). We evaluate all the metrics at a rank cut-off of 100 because it has been shown to have better robustness to sparsity and popularity biases and greater discriminative power than using shallower cut-offs (Valcarce et al., 2018). All the metrics are computed for each user ranking and averaged over the whole set of users. If a recommendation model is unable to provide recommendations to certain users, this is penalized by giving a score of zero in all metrics for those users.

[1] The source code is located at https://gitlab.irlab.org/alfonso.landin/prefs2vec.

[2] https://grouplens.org/datasets/movielens/20m.

[3] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r.

Table 2
Values of the hyperparameters for user-based and item-based WSR (WSR-UB and WSR-IB), PureSVD, BPRMF, WRMF, CoFactor, NeuMF and user-based and item-based WSR with `prefs2vec` embeddings (`prefs2vec`-WSR-UB and `prefs2vec`-WSR-IB).

Model	MovieLens 20M	R3-Yahoo!	LibraryThing
WSR-UB	$k = 75$	$k = 125$	$k = 50$
WSR-IB	$k = 75$	$k = 75$	$k = 25$
PureSVD	$d = 30, \kappa = 10^{-6}$	$d = 15, \kappa = 10^{-6}$	$d = 700, \kappa = 10^{-6}$
BPRMF	$d = 50,\ \lambda = 0.01,\ \alpha = 0.01,\ i = 10^5$	$d = 175,\ \lambda = 0.01,\ \alpha = 0.01,\ i = 10^5$	$d = 600,\ \lambda = 0.001,\ \alpha = 0.01,\ i = 10^6$
WRMF	$d = 100,\ \lambda = 0.1,\ \alpha = 2,\ i = 50$	$d = 20,\ \lambda = 0.001,\ \alpha = 2,\ i = 50$	$d = 500,\ \lambda = 0.001,\ \alpha = 2,\ i = 50$
CoFactor	$d = 100,\ c_0 = 0.3,\ c_1 = 3,\ k = 1,$ $\lambda_\theta = \lambda_\beta = \lambda_\gamma = 10^{-5}$	$d = 30,\ c_0 = 1,\ c_1 = 10,\ k = 1,$ $\lambda_\theta = \lambda_\beta = \lambda_\gamma = 10^{-5}$	$d = 500,\ c_0 = 1,\ c_1 = 10,\ k = 1,$ $\lambda_\theta = \lambda_\beta = \lambda_\gamma = 10^{-5}$
NeuMF	$d = 64,\ i = 20,\ n = 5$	$d = 12,\ i = 5,\ n = 5$	$d = 1024,\ i = 20,\ n = 5$
`prefs2vec`-WSR-UB	$k = 150,\ d = 300,\ w = 50,\ i = 500,$ $n = 10$	$k = 100,\ d = 300,\ w = 5,\ i = 200,$ $n = 10$	$k = 75,\ d = 300,\ w = 10,\ i = 50,\ n = 10$
`prefs2vec`-WSR-IB	$k = 75,\ d = 300,\ w = 50,\ i = 100,$ $n = 10$	$k = 150,\ d = 300,\ w = 10,\ i = 200,$ $n = 10$	$k = 10,\ d = 300,\ w = 20,\ i = 50,\ n = 10$

Table 3
Values of nDGC@100, Gini@100, MSI@100 on MovieLens 20M, R3-Yahoo! and LibraryThing datasets for different recommendation algorithms. Statistical significant improvements (permutation test with $p < 0.01$) in nDCG@100 and MSI@100 with respect to user-based and item-based WSR (WSR-UB and WSR-IB), PureSVD, BPRMF, WRMF, CoFactor, NeuMF and user-based and item-based WSR with `prefs2vec` embeddings (`prefs2vec`-WSR-UB and `prefs2vec`-WSR-IB) are superscripted with a, b, c, d, e, f, g, h and i, respectively.

Model	Metric	MovieLens 20M	R3-Yahoo!	LibraryThing
WSR-UB	nDCG@100	0.4449^{bcdfgi}	0.0704^{c}	0.2255
	Gini@100	0.0310	0.3107	0.0417
	MSI@100	210.0885^{dh}	305.2605	333.4988
WSR-IB	nDCG@100	0.3842^{d}	0.0727^{acfg}	$0.3085^{acdefgh}$
	Gini@100	0.0310	0.3356	0.2768
	MSI@100	213.6453^{adh}	315.0507^{ad}	$461.7353^{acdefghi}$
PureSVD	nDCG@100	0.4125^{bd}	0.0659	0.2475^{agh}
	Gini@100	0.0281	0.2942	0.0991
	MSI@100	214.4369^{abdh}	321.8010^{abd}	427.8504^{adghi}
BPRMF	nDCG@100	0.3552	0.0721^{acg}	0.2997^{acefgh}
	Gini@100	0.0259	0.3683	0.0907
	MSI@100	202.4853	312.9293^{a}	377.6941^{ah}
WRMF	nDCG@100	$0.4466^{abcdfgi}$	0.0713^{c}	0.2938^{acgh}
	Gini@100	0.0481	0.4240	0.1524
	MSI@100	248.0235^{abcdfh}	340.0716^{abcdgh}	$443.4348^{acdfghi}$
CoFactor	nDCG@100	0.4437^{bcdgi}	0.0701^{c}	0.2959^{acegh}
	Gini@100	0.0478	0.4530	0.1477
	MSI@100	247.5458^{abcdh}	$344.0558^{abcdegh}$	439.5213^{acdghi}
NeuMF	nDCG@100	0.4202^{bcdi}	0.0700^{c}	0.2262
	Gini@100	0.0565	0.3926	0.1572
	MSI@100	$250.9000^{abcdefh}$	321.7298^{abd}	422.3690^{adh}
`prefs2vec`-UB	nDCG@100	$\mathbf{0.4504}^{abcdefgi}$	$\mathbf{0.0735}^{acefg}$	0.2242^{ag}
	Gini@100	0.0317	0.4441	0.0422
	MSI@100	205.9805^{d}	329.1373^{abcdg}	345.3843^{a}
`prefs2vec`-IB	nDCG@100	0.4172^{bcd}	0.0719^{cg}	$\mathbf{0.3107}^{acdefgh}$
	Gini@100	**0.0666**	**0.4767**	**0.4321**
	MSI@100	$\mathbf{257.8653}^{abcdefgh}$	$\mathbf{348.4311}^{abcdefgh}$	$\mathbf{499.1308}^{abcdefgh}$

4.3. Effectiveness

We compare the effectiveness of user-based and item-based WSR using `prefs2vec` embeddings against several state-of-the-art baselines of different nature. The most natural baselines are the user-based and item-based WSR algorithms using the traditional rating-based representation (Valcarce et al., 2016). We also employ PureSVD (Cremonesi et al., 2010), BPRMF (Rendle et al., 2009) and WRMF (Hu et al., 2008), three state-of-the-art matrix factorization techniques. Additionally, we use CoFactor (Liang et al., 2016), a variant of WRMF that jointly factorizes the user–item matrix and an embedding model. Finally, we used NeuMF, a state-of-the-art neural collaborative filtering approach (He et al., 2017). We tune the hyperparameters of all the baselines using a grid search to maximize nDCG@100. Since `prefs2vec` has several hyperparameters, we decided to keep it simple and fix the number of dimensions d to 300 and the number of negative samples n to 10. Additionally, we used 300 iterations on the MovieLens and 100 iterations on the other datasets. We only fine-tune the hyperparameter w which is key to the performance of `prefs2vec` embeddings due to its regularization properties (more discussion about this in Section 4.4). For the sake of reproducibility, we report the values of hyperparameters for all techniques in Table 2 (we employed the notation used in the original papers). The best values for each recommender system are presented in Table 3.

We use the permutation test ($p < 0.01$) to study the statistical significance of the improvements regarding nDCG@100 and MSI@100 (Smucker et al., 2007). We cannot apply this procedure to the Gini index because we are using a paired test, and Gini is a global metric.

Table 3 shows that WSR with `prefs2vec` embeddings provides the best results in terms of nDCG@100 in all datasets. While the user-based approach produced the best results on the MovieLens and Yahoo! datasets, the item-based approach outperformed the user-based method on the LibraryThing dataset. In general, the choice between user-based or item-based approaches depends on the characteristics of the datasets and should be discovered empirically (Ning et al., 2015; Valcarce et al., 2016). Nevertheless, the item-based embedding representation presents high figures of diversity and novelty.

We observe that the prefs2vec embeddings always outperform the WSR-UB and WSR-IB. In fact, the user-based embeddings improve the user-based rating representation in the same way that the item-based embeddings are superior to the item-based rating representation in all metrics: ranking accuracy, diversity, and novelty. However, the difference in nDCG@100 is not statistically significant in the item-based case on the R3-Yahoo! and LibraryThing, where the value with the embeddings is slightly greater. In contrast, prefs2vec produces important improvements in diversity and novelty. Overall, these results support the use of prefs2vec embeddings instead of the traditional rating representation with WSR, a state-of-the-art memory-based recommender.

Regarding the model-based baselines, the best factorization technique in terms of nDCG@100 depends on the dataset. While WRMF is the best matrix factorization approach on MovieLens 20M, BPRMF outperforms the rest on R3-Yahoo! and LibraryThing. Nevertheless, WSR with prefs2vec embeddings surpass these factorization methods in all metrics. Note that matrix factorization techniques are considered to be among the most effective collaborative filtering methods (Koren and Bell, 2015). Concerning NeuMF, it provides more diverse and novel recommendations than prefs2vec on MovieLens 20M but at the expense of an inferior figure in nDCG@100, when compared with the user-based embeddings. When compared with the item-based embeddings the inverse applies, with NeuMF producing more accurate recommendations at the expense of diversity and novelty. On the rest of datasets, prefs2vec outperforms NeuMF in all metrics.

In general, Table 3 shows that the factorization approaches outperform WSR with the traditional rating representation in novelty and diversity. However, the use of prefs2vec embeddings can increase the diversity and novelty figures of WSR surpassing PureSVD, BPRMF, WRMF, CoFactor, and NeuMF in most cases.

4.4. Regularization effect

To introduce regularization into the prefs2vec model, we proposed to use a variant of dropout in the input layer controlled by the window hyperparameter w. The smaller the window, the greater the regularizing effect. Moreover, the training complexity of each training step grows with the window size—recall that the complexity is $\mathcal{O}(d \times (w+n))$. Therefore, a small window yields also important efficiency gains. In Table 4, we present the time per iteration (averaged over 10 executions) with different window sizes using 300 dimensions and 10 negative samples. Note that the iteration times on the MovieLens 20M are larger than on other datasets because of its size. We observe that the use of regularization ($w < w_{max}$) greatly reduces the training time. However, how does the hyperparameter w affect the quality of the recommendations?

In Fig. 2, we plot the values of nDCG@100 generated by WSR with prefs2vec embeddings. We use user embeddings on the MovieLens 20M and R3-Yahoo! datasets and item embeddings on the LibraryThing collection because those are the settings that yield the highest values of nDCG. We employ the values of the hyperparameters used in the previous experiment, and we explore different values for the number of iterations and the window size. On the MovieLens 20M dataset, without using dropout ($w = w_{max}$), we were not able to compute more than 50 iterations within one day; however, we can see a decreasing trend after two iterations.

Fig. 2 confirms that we can increase the regularization effect by reducing the window size. With a correct value of w, we can counter overfitting because the recommendation quality becomes stable after a certain number of iterations. Although we need more iterations to converge when the window size decreases, each iteration is also cheaper (recall Table 4).

Table 4
Time per iteration (measured in seconds) on MovieLens 20M, R3-Yahoo! and LibraryThing datasets.

w	MovieLens 20M	R3-Yahoo!	LibraryThing
5	262.575	4.449	12.648
10	276.702	4.884	15.374
20	301.351	5.726	17.444
50	382.010	8.162	22.463
100	559.590	13.126	32.973
200	948.531	22.989	50.683
500	2115.945	47.988	88.682
w_{max}	32342.160	411.594	241.487

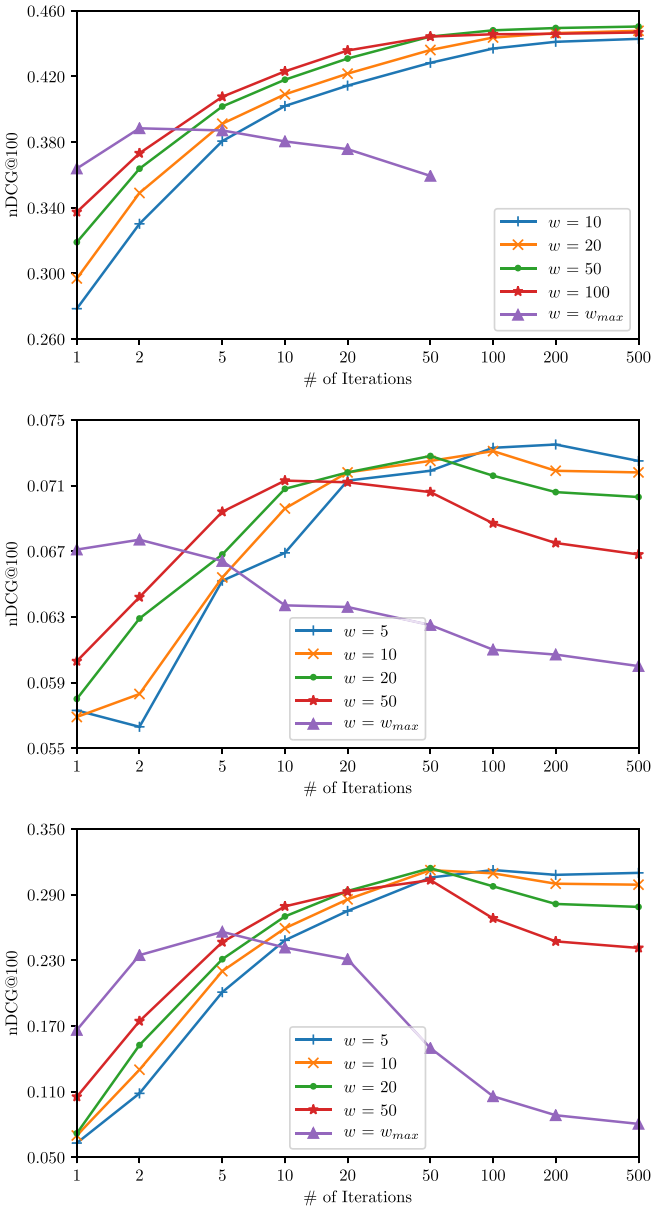

Fig. 2. Values of nDCG@100 of WSR with prefs2vec user embeddings on MovieLens 20M (top) and item embeddings on R3-Yahoo! (middle) and LibraryThing (bottom) varying the number of iterations and the window size w.

5. Incremental recommendation

Updating recommendations when new feedback is available is known as incremental recommendation. Also, other problems such as cold start arise: how can we deal with new users or new items

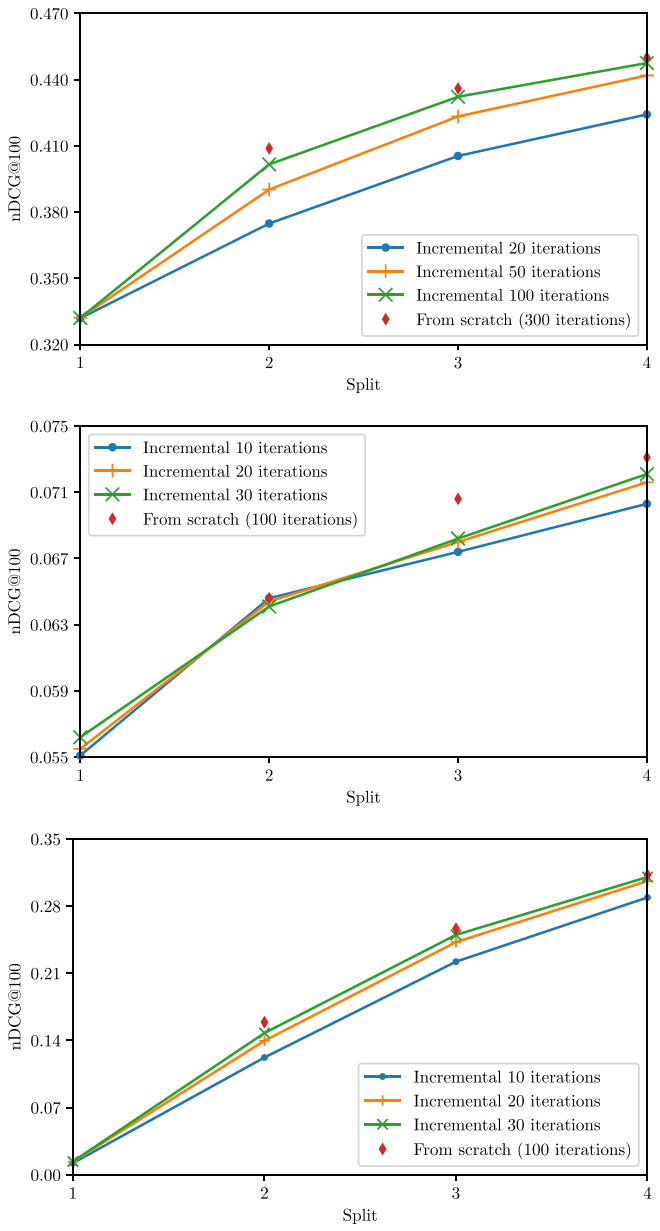

Fig. 3. Values of nDCG@100 of `prefs2vec` on different splits of the training set on MovieLens 20M (top), R3 Yahoo (middle) and LibraryThing (bottom) datasets using regular and incremental embeddings with different number of iterations.

that appear in the system? The incorporation of new information into model-based recommenders can be difficult. Some authors have proposed recommendation algorithms that can be updated (Takács et al., 2008), but many models have to be rebuilt from scratch. In contrast, memory-based recommender systems can exploit new feedback as soon as it is available. For example, we can incorporate preferences to WSR without any adaptation. However, we also need to update the neighbourhoods. This process can be done offline periodically. If we want to compute fresh recommendations immediately, we can use the old neighbourhoods with the new feedback. This will not produce the best recommendations with the actual information, but it would provide a good trade-off between efficiency and freshness.

When using `prefs2vec` embeddings, we can employ old embeddings with the new feedback to get approximately fresh recommendations. Also, we could have an offline process in charge of updating the embeddings periodically. However, computing embeddings from scratch periodically may be too expensive. Therefore, below, we present an alternative for updating `prefs2vec` embeddings efficiently.

5.1. Incremental `prefs2vec` embeddings

The embedding model starts from a random initialization of matrices $\mathbf{W}$ and $\mathbf{W}'$ and, using backpropagation, we learn the parameters of these matrices. This is a non-convex optimization problem and, therefore, we will only find a local optimum with gradient descent techniques. Moreover, in a production environment, we are receiving new data constantly: we may register new user–item ratings (or another kind of preferences) or include new items or users in the system. For this reason, we need to update the embeddings frequently in an efficient manner.

When we have new ratings, we can use them directly in WSR or other memory-based recommenders. However, we need to recompute the embeddings and the neighbourhoods. Instead of starting the learning process with a random initialization, we propose to start from the previous embeddings. This approach reduces the number of iterations needed to reach convergence. When we are computing the new embedding model, we should employ all the data, not only the new ratings, to reach a solution to the global problem. Additionally, in the item-based approach, when new items appear, we need to add new rows to $\mathbf{W}$ and new columns to $\mathbf{W}'$ corresponding the input and output item embeddings, respectively. Likewise, in the user-based approach, for each new user, we append a row and a column to $\mathbf{W}$ and $\mathbf{W}'$, respectively. We initialize the new rows or columns randomly; however, the convergence is quite fast because the rest of the embedding vectors are close to a local solution of a similar problem. Next, we present empirical results that support the validity of this incremental approach.

5.2. Evaluating the incremental embeddings

We conduct experiments on the three datasets to test the validity of the presented approach to build incremental embeddings. We split each training set into four parts of equal size randomly. We train item embeddings on LibraryThing and user embeddings on the other datasets. First, we compute `prefs2vec` embeddings from scratch using only the first split. We generate recommendations with those embeddings and use the test set to evaluate them. Since we are neglecting 75% of the training set, we expect a huge drop in performance. Next, we train the embeddings using the first two splits in two different ways. On the one hand, we learn new embeddings from scratch as a baseline. On the other hand, we start from the embeddings from the previous split, and we incrementally build updated embeddings following the proposed approach. We repeat this procedure with the third and fourth splits.

Fig. 3 compares the incremental embeddings against the embeddings trained from scratch in terms of nDCG@100. We used 300 iterations to train the embeddings from scratch on MovieLens and 100 iterations on the other datasets, matching the value for this parameter on the reported best results for each dataset. We use fewer iterations to train the incremental embeddings, using 20, 50 and 100 iterations on MovieLens and 10, 20 and 30 iterations on the other datasets. We can observe that we obtain `prefs2vec` embeddings that achieve similar effectiveness than the baseline with fewer iterations. Even with a low number of iterations, we can obtain good performance. This effect is less pronounced on R3-Yahoo!, probably due to its high sparsity, suggesting the need to train the embeddings from scratch from time to time.

6. Conclusions and future work

In this work, we presented `prefs2vec`, a neural embedding model for collaborative filtering inspired in `word2vec` continuous bag-of-words (CBOW) model. This model exploits collaborative filtering preferences to compute user and item embeddings. These embeddings can be easily plugged into standard memory-based recommender systems. The experiments showed that our proposal not only outperforms the traditional rating-based representation of users and items but also provided significant improvements compared to a representative set of state-of-the-art recommendation techniques in terms of ranking accuracy, diversity, and novelty.

We proposed a procedure to learn `prefs2vec` embeddings incrementally, allowing a quick update of the embeddings, which is vital in a production environment to provide fresh recommendations. This procedure, in combination with the ability of memory-based algorithms for providing incremental recommendations without the need for recomputing the whole neighbourhood, makes our proposal computationally affordable to production environments.

We proposed to regularize our model using a modified version of dropout, which not only improves the effectiveness of the embeddings but also the efficiency of the training process. The use of negative sampling and the variant of dropout supports the scalability of `prefs2vec` to large-scale collections. Moreover, we presented how to adapt `word2vec` CBOW implementations to compute `prefs2vec` embeddings. In this way, we can leverage any existent implementations, including parallel or distributed ones.

As future work, we contemplate extending the `prefs2vec` model to deal not only with sparsity and cold start problems but also with side information and temporal factors. For example, recent work has explored time-aware user and item representations for recommendation as trajectories in a latent space (Guàrdia-Sebaoun et al., 2015). Finally, we plan to study which benefits other embedding models such as GloVe (Pennington et al., 2014) could provide for collaborative filtering.

Acknowledgements

This work was supported by accreditation ED431G/01 (Xunta de Galicia & ERDF), project ED431B 2019/03 (Xunta de Galicia & ERDF) and project RTI2018-093336-B-C22 (MCIU & ERDF). The first and second authors also acknowledge the support of grants FPU014/01724 and FPU17/03210 (MCIU), respectively.

References

Abdollahi, B., Nasraoui, O., 2016. Explainable matrix factorization for Collaborative filtering. In: Proceedings of the 25th International Conference Companion on World Wide Web. In: WWW '16 Companion, International World Wide Web Conferences Steering Committee, Geneva, Switzerland, pp. 5–6. http://dx.doi.org/10.1145/2872518.2889405.

Barkan, O., Koenigstein, N., 2016. ITEM2VEC: Neural item embedding for collaborative filtering. In: 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP). IEEE, pp. 1–6. http://dx.doi.org/10.1109/MLSP.2016.7738886.

Bellogín, A., Castells, P., Cantador, I., 2011. Precision-Oriented evaluation of recommender systems. In: Proceedings of the 5th ACM Conference on Recommender Systems. In: RecSys '11, ACM, New York, NY, USA, pp. 333–336. http://dx.doi.org/10.1145/2043932.2043996.

Cremonesi, P., Koren, Y., Turrin, R., 2010. Performance of recommender algorithms on Top–N recommendation tasks. In: Proceedings of the 4th ACM Conference on Recommender Systems. In: RecSys '10, ACM, New York, NY, USA, pp. 39–46. http://dx.doi.org/10.1145/1864708.1864721.

Deshpande, M., Karypis, G., 2004. Item-based top-N recommendation algorithms. ACM Trans. Inf. Syst. 22 (1), 143–177. http://dx.doi.org/10.1145/963770.963776.

Dong, W., Moses, C., Li, K., 2011. Efficient K-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web. In: WWW '11, ACM, New York, NY, USA, pp. 577–586. http://dx.doi.org/10.1145/1963405.1963487.

Fleder, D., Hosanagar, K., 2009. BlockbusterCulture's next rise or fall: The impact of recommender systems on sales diversity. Manage. Sci. 55 (5), 697–712. http://dx.doi.org/10.1287/mnsc.1080.0974.

Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., Sharp, D., 2015. E-commerce in your inbox: Product recommendations at scale. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. In: KDD '15, ACM, New York, NY, USA, pp. 1809–1818. http://dx.doi.org/10.1145/2783258.2788627.

Guàrdia-Sebaoun, E., Guigue, V., Gallinari, P., 2015. Latent trajectory modeling: A light and efficient way to introduce time in recommender systems. In: Proceedings of the 9th ACM Conference on Recommender Systems. In: RecSys '15, ACM, New York, NY, USA, pp. 281–284. http://dx.doi.org/10.1145/2792838.2799676.

Gunawardana, A., Shani, G., 2015. Evaluating recommender systems. In: Ricci, F., Rokach, L., Shapira, B. (Eds.), Recommender Systems Handbook, second ed. Springer US, Boston, MA, USA, pp. 265–308. http://dx.doi.org/10.1007/978-1-4899-7637-6_8.

Harris, Z.S., 1954. Distributional structure. Word 10 (2–3), 146–162. http://dx.doi.org/10.1080/00437956.1954.11659520.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.-S., 2017. Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web. In: WWW '17, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, pp. 173–182. http://dx.doi.org/10.1145/3038912.3052569.

Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T., 2004. Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. 22 (1), 5–53. http://dx.doi.org/10.1145/963770.963772.

Hu, Y., Koren, Y., Volinsky, C., 2008. Collaborative filtering for implicit feedback datasets. In: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. In: ICDM '08, IEEE, Washington, DC, USA, pp. 263–272. http://dx.doi.org/10.1109/ICDM.2008.22.

Koren, Y., Bell, R., 2015. Advances in Collaborative filtering. In: Ricci, F., Rokach, L., Shapira, B. (Eds.), Recommender Systems Handbook, second ed. Springer US, Boston, MA, USA, pp. 77–118. http://dx.doi.org/10.1007/978-1-4899-7637-6_3.

Le, Q.V., Mikolov, T., 2014. Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning. In: ICML'14, JMLR.org, pp. 1188–1196. http://dx.doi.org/10.1145/2740908.2742760.

Levy, O., Goldberg, Y., 2014. Neural word embedding as implicit matrix factorization. In: Proceedings of the 27th International Conference on Neural Information Processing Systems. In: NIPS '14, MIT Press, Cambridge, MA, USA, pp. 2177–2185. http://dx.doi.org/10.1162/153244303322533223.

Liang, D., Altosaar, J., Charlin, L., Blei, D.M., 2016. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In: Proceedings of the 10th ACM Conference on Recommender Systems. In: RecSys '16, ACM, New York, NY, USA, pp. 59–66. http://dx.doi.org/10.1145/2959100.2959182.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013a. Efficient estimation of word representations in Vector Space. Clin. Orthop. Relat. Res. abs/1301.3, 1–12.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., 2013b. Distributed representations of words and phrases and their Compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems. In: NIPS'13, Curran Associates Inc., USA, pp. 3111–3119.

Ning, X., Desrosiers, C., Karypis, G., 2015. A comprehensive survey of Neighborhood-based recommendation methods. In: Ricci, F., Rokach, L., Shapira, B. (Eds.), Recommender Systems Handbook, second ed. Springer US, Boston, MA, USA, pp. 37–76. http://dx.doi.org/10.1007/978-1-4899-7637-6_2.

Pennington, J., Socher, R., Manning, C., 2014. Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. In: EMNLP '14, ACL, Stroudsburg, PA, USA, pp. 1532–1543. http://dx.doi.org/10.3115/v1/D14-1162.

Phaisangittisagul, E., 2016. An analysis of the regularization between L2 and dropout in single hidden layer neural network. In: Proceedings of the 7th International Conference on Intelligent Systems, Modelling and Simulation. In: ISMS '16, IEEE, pp. 174–179. http://dx.doi.org/10.1109/ISMS.2016.14.

Ř.ehůřek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA, Valletta, Malta, pp. 45–50.

Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L., 2009. BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. In: UAI '09, AUAI Press, Arlington, Virginia, USA, pp. 452–461.

Shen, J., Deng, C., Gao, X., 2016. Attraction recommendation: Towards personalized tourism via collective intelligence. Neurocomputing 173 (P3), 789–798. http://dx.doi.org/10.1016/j.neucom.2015.08.030.

Smucker, M.D., Allan, J., Carterette, B., 2007. A comparison of statistical significance tests for information retrieval evaluation. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. In: CIKM '07, ACM, New York, NY, USA, p. 623. http://dx.doi.org/10.1145/1321440.1321528.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958. http://dx.doi.org/10.1214/12-AOS1000.

Takács, G., Pilászy, I., Németh, B., Tikk, D., 2008. Investigation of various matrix factorization methods for large recommender systems. In: Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition - NETFLIX '08. ACM Press, New York, NY, USA, pp. 1–8. http://dx.doi.org/10.1145/1722149.1722155.

Valcarce, D., Bellogín, A., Parapar, J., Castells, P., 2018. On the robustness and discriminative power of information retrieval metrics for top-N recommendation. In: Proceedings of the 12th ACM Conference on Recommender Systems. In: RecSys '18, ACM, New York, NY, USA, pp. 260–268. http://dx.doi.org/10.1145/3240323.3240347.

Valcarce, D., Parapar, J., Barreiro, Á., 2016. Language models for Collaborative filtering neighbourhoods. In: Proceedings of the 38th European Conference on Information Retrieval. In: ECIR '16, Springer, Berlin, Heidelberg, pp. 614–625. http://dx.doi.org/10.1007/978-3-319-30671-1_45.

Vasile, F., Smirnova, E., Conneau, A., 2016. Meta-Prod2Vec: Product embeddings using side-information for recommendation. In: Proceedings of the 10th ACM Conference on Recommender Systems. In: RecSys '16, ACM, New York, NY, USA, pp. 225–232. http://dx.doi.org/10.1145/2959100.2959160.

Wang, Y., Wang, L., Li, Y., He, D., Chen, W., Liu, T.-Y., 2013. A theoretical analysis of NDCG ranking measures. In: Proceedings of the 26th Annual Conference on Learning Theory. In: COLT '13, JMLR.org, pp. 1–30.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J.R., Zhang, Y.-C., 2010. Solving the apparent diversity-accuracy dilemma of recommender systems. Proc. Natl. Acad. Sci. 107 (10), 4511–4515. http://dx.doi.org/10.1073/pnas.1000488107.