Progress in Artificial Intelligence manuscript No.
(will be inserted by the editor)

Improving recommender systems by encoding items and user profiles considering the order in their consumption history

Pablo Pérez-Núñez · Oscar Luaces · Antonio Bahamonde · Jorge Díez

Received: date / Accepted: date

Abstract The aim of Recommender Systems is to suggest items (products) to satisfy each user's particular taste. Representation strategies play a very important role in these systems, as an adequate codification of users and items is expected to ease the induction of a model which synthesizes their tastes and make better recommendations. However, in addition to gathering information about users' tastes, there is an additional aspect that can be relevant for a proper codification strategy, namely the order in which the user interacted with the items. In this paper, several encoding strategies based on neural networks are analyzed and applied to solve two different recommendation tasks in the context of music playlists. The results show that the order in which the musical pieces were listened to is relevant for the codification of items (songs). We also find that the encoding of user profiles should use a different amount of historical data depending on the learning task to be solved. In other words, we do not always have to use all the available data; sometimes it is better to discard old information, as tastes change over time.

Keywords Recommender Systems · Profiles · Matrix Factorization · Collaborative Filtering.

This work was funded under grants TIN2015-65069-C2-2-R from the MINECO (Spanish Ministry of the Economy and Competitiveness) and IDI-2018-000176 from the Principado de Asturias Regional Government, partially supported with ERDF funds.

P. Pérez-Núñez, O. Luaces, A. Bahamonde and J. Díez
Artificial Intelligence Center, Universidad de Oviedo, Gijón, Asturias, Spain
E-mail: {pabloperez,oluaces,abahamonde,jdiez}@uniovi.es

1 Introduction

Most of the digital resources we use today make recommendations to their users using Recommender Systems; see [21, 22]. This happens, for example, when we enter the site of a digital newspaper. The order in which news is placed is already, in some way, a reading recommendation. The ordered lists of news that appear in the margins (*"most read news"* or *"most commented"*) also constitute a recommendation based, in this case, on the interaction of other users with the digital publication. Another example of recommendation can be found within the articles, in the form of links to other news, constituting recommendations of the type *"related news"*.

Recommender Systems have found a very important field of application in online stores, with the aim of increasing customer satisfaction and thereby promoting an increase in sales. In this case, products are recommended with suggestions like *"customers who saw this product also saw ..."* and *"frequently bought together"*. Customers are offered rankings of products based on the number of sales or buyer reviews. We can also find recommendations on platforms for streaming multimedia content, for example, Netflix or Spotify, on websites dedicated to organizing reviews of hotels and restaurants, such as TripAdvisor, Yelp, etc., as well as for movies [11], news items in digital publications [7] and web services [4], among others.

Some of the aforementioned examples are elementary, non-personalized recommendations, based simply on the behavior (or taste) of the majority of customers: if many people watch a movie which, in addition, has very good ratings, then it is quite likely to be recommended to users who have not yet watched it.

There are other systems that make their recommendations taking into account additional information present in the context. One of the most basic systems, already mentioned above, considers the item that a user is viewing and recommends other items similar to it, or products that were purchased together with it. These systems take into account the product that is being viewed to make a more focused recommendation.

Other recommenders store a profile for each user related to their history of consumption. This profile is intended to encode their tastes or preferences to some extent [7]. Subsequently, these profiles can be used to make personalized recommendations, which will hopefully achieve greater user satisfaction. The way in which this profile is computed may be more or less elaborate.

One way to take the consumption profile into account is to represent each user by a vector which encodes the information related to their interaction with the products involved. Throughout this article, we use the word *interaction* to refer to any form of product consumption: the purchasing of an item, listening to a song, viewing a product in an online store, reading a news item in a digital publication, and so on. This vector encoding can be achieved by, for example, the construction of *embeddings* using matrix factorization [15]. Similarly, products can also be represented by vectors that summarize the interaction in which they were involved.

In this paper, which is based on a previous publication [19], we will explore the benefits to be obtained by using two techniques based on neural networks, *word2vec* [16] and *doc2vec* [14] (also known as *Paragraph Vector*), for encoding users and products. These algorithms were originally designed to encode words and documents in processes that require representing texts for learning tasks. The codification obtained for each word is based on its context (surrounding words) in each document. The consumption of certain types of products is also related to the temporal context, for example, the reproduction of a sequence of songs (*playlist*), so we can use these techniques, assuming that a song is more likely to be played, depending on the songs played in the near past (i.e. the temporal surroundings).

In order to show the importance of this sequential order, in this paper we compare the codification obtained by these techniques with another one obtained by constructing an embedding from a *one-hot* encoding, in which the order of consumption is not taken into account in any way whatsoever.

A decision that must be adopted to codify the consumption profiles of users is how far we will go back in the temporal sequence to build and encode such a profile. In this paper, we compare the results obtained using a codification of the recent consumption history with another one using a long-term consumption history. Notice that, in certain situations, it could be useful to keep both the long-term and the short-term profiles, especially when the sales of the products for which you wish to make recommendations are seasonal in behavior. For example, if we intend to make food recommendations, we could take into account the fact that, in some regions, certain dishes are usually consumed during winter, while others are more frequently consumed in the summer.

Recommender systems can be classified into two major groups. On the one hand, we can build *collaborative filters* [11], in which only the information we have about the interaction between users and products is used, and this behavior can give us an idea of their preferences. On the other hand, we have *content based* recommender systems [17], which use additional information on users, such as their location, gender, age, etc., and on the products, for example, genre and release date, if they are movies.

In this paper, we only use collaborative filters, as we wish to study the impact of the ordering of interactions between users and products. Including information based on content could distort the study, although it is likely that this may well improve the results in terms of precision in the recommendation, because we would have more information at our disposal.

The paper is organized as follows. In the next section, we present a brief review of previous papers related to the elaboration of profiles. Section 3 is devoted to detailing how profiles can be encoded using *word2vec* and *doc2vec*, which will be used to encode and compare two types of profiles for each user, one long-term (*consolidated profile*) and another short-term (*recent profile*). Then, in Section 4, we pose two recommendation tasks that will be used to test the performance of these profiles. In Section 5, we discuss the results obtained in the two recommendation tasks mentioned above, comparing the scores obtained using the different codification approaches. Finally, we present our conclusions and point to a future line of research.

2 Related Work

There are several papers that address the codification of users and items using matrix factorization.

In [24], the authors present an algorithm to project artists and songs in a vectorial space using matrix factorization. The learning tasks addressed in their study are related to guessing the artist who performs a given song, predicting the songs performed by a given artist,

and predicting similar artists (respectively, songs) with respect to a given one. These questions are, in fact, recommendation tasks and the projections of artists and songs can be considered as profiles.

Something similar occurs in [5], a paper in which the projections of users and songs are calculated in order to generate *playlists* to the user's liking.

The T-RECSYS system [8], a hybrid tune recommender that analyzes the user's history and the characteristics of songs, has been presented recently. The main difference with respect to our approach is that it uses a fixed representation for users and songs. The system describes songs by genre, artist type, artist era, mood, tempo, and release year, and users are described by the last nine songs they have heard. In contrast, our system learns the representation (in the form of embeddings) during the training.

Matrix factorization has also been used to condense people's tastes regarding food products. In [15], the authors obtained useful consumer profiles to ascertain their preferences regarding the consumption of meat with different maturation periods.

In [7], the authors present a news recommendation system applicable in the context of digital newspapers. Both the news and the readers were represented by one-hot vectors, from which an embedding is computed in order to obtain a better representation. In their paper, the best results were obtained when the readers are represented via the concatenation of the vectors representing the last two read news items, i.e. taking into account the order of reading. The cited study led to a slightly different approach, presented in [6], in which the authors show that it is possible to improve the novelty and diversity of recommendations with only a small penalization in accuracy.

Regarding encoding using *word2vec* / *doc2vec*, our study is not the first to use these approaches in a field other than text processing. For instance, *word2vec* has been successfully used in several algorithms for network embedding tasks. The first one was DeepWalk [18]. DeepWalk learns latent representations of the vertices of a network which encode social relations in a vector. Inspired by DeepWalk, LINE [23] and *node2vec* [9] were developed to address the same problem. An interesting theoretical study of the performance of all these algorithms can be found in [20]. In our paper, we also use *word2vec*, but we adapt it to the task of recommendation. Moreover, we go one step further, applying *doc2vec* to codify user profiles.

3 Building profiles from the interaction between users and items

Let us assume a set of users, $\mathcal{U}$, and a set of products, $\mathcal{P}$. In addition, we know the interactions each user has had with the products and in what order these interactions have occurred. Thus, for each user we have an ordered list of products (items) with which the user has interacted over time:

$$\mathcal{D} = \{(u, p_1^u, p_2^u, \ldots) : u \in \mathcal{U}, p_j^u \in \mathcal{P}\}. \quad (1)$$

Sets $\mathcal{U}$ and $\mathcal{P}$ contain only identifiers of users and products, respectively. A common representation for the elements of these sets is to use *one-hot* vectors. A one-hot vector representing the i-th element of the set has a 1 in its i-th component and the rest are all equal to 0. Therefore, the dimension of these one-hot vectors must be equal to the cardinality of the set of elements they represent. The *word2vec* and *doc2vec* will be in charge of computing the corresponding *embeddings* to project users and items (in one-hot codification) in a different space. The approach used by these algorithms, which we will briefly describe below, takes into account the order in the sequence of consumed products. Hence, we will compare the performance of a recommender system using this codification with the simpler one-hot representation, which does not consider the order in the sequence at all.

3.1 Item encoding

Word2vec [16] is an algorithm designed to encode the words of a corpus by means of vectors, in such a way that they are arranged in space according to how they are related in the texts of the corpus. Words with close representations usually have a strong semantic relationship. For example, the authors explain in [16] that we can expect to obtain a vector close to the one representing the word "Paris" if we subtract the vector of "Spain" from the one of "Madrid" and then we add the vector of "France".

Word2vec encoding is achieved by processing a given text, defining a window which is moved along the entire sequence of (ordered) words. Using this window of words, the authors propose two possible approaches, which yield the following learning tasks:

- *Continuous Bag of Words (CBOW)*: the learning task consists in predicting the central word of the window from its context.
- *Skip-gram*: the words of the context are predicted using the central word as input (Figure 1).

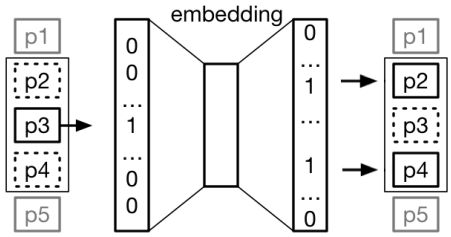

Figure 1 Architecture of *word2vec* using skip-gram with a window size of 1 (number of words before and after the input word). The input is a word and the output is its context, represented as the sum of the one-hot vectors corresponding to the words in the surroundings of the input word

In both cases, the context is defined by a window size which determines the number of words before and after a given one, and it is represented as the sum of their corresponding one-hot vectors. Figure 1 shows a schema of *word2vec* using skip-gram, in which the word $\boldsymbol{p}_3$ is used as input, and its context as the desired output. In this illustration, the context are the words $\boldsymbol{p}_2$ and $\boldsymbol{p}_4$ or, more specifically, the sum of the vectors that represent these words. There is a hidden layer between the input and the output layers in which we will obtain the representation for each word after the training process.

In their paper, the authors claim that the skip-gram strategy offers better results and therefore we will use this strategy to codify the items in our recommender system. Thus, to learn the codification of each item, we will process the ordered lists of interactions with items collected in (1) as if they were sequences of words in a text. That is, for a given window size, we will learn to predict the context of items with which there has been interaction before and after the interaction with a given item.

3.2 User encoding

Once we have codified the items, we can build a profile for each user taking into account each one's specific interaction sequence. To do so, we propose to use the *doc2vec* algorithm [14], which has been designed to encode documents or paragraphs from words previously encoded by *word2vec*. In their paper, the authors claim that the codification of documents obtained by this algorithm improves the performance in information retrieval tasks with respect to the popular *bag of words*. To obtain the codification of documents, the authors also suggest two possible frameworks:

– A *Distributed Memory version of Paragraph Vector (PV-DM)*: the codification of documents is given by an embedding computed as follows. The input is the concatenation of the vector representing the document with a sequence (sliding window) of words (previously encoded using *word2vec*) in said document; the desired output is the word following the sequence of input words. The embedding used to encode the words remains fixed during the learning process, modifying only the embedding of documents to minimize the prediction error.
– A *Distributed Bag of Words version of Paragraph Vector (PV-DBOW)*: a document is taken and a few words are chosen randomly from its content. Starting from the vector that represents the document, its embedding is learned in order to predict the presence of the previously chosen words, regardless of their order.

In both cases, the final goal is to modify the embedding of documents during the training stage in order to obtain a codification that minimizes the prediction error. In [14], the authors show the good performance of their proposals and recommend using the concatenation of both, PV-DM and PV-DBOW. Note that PV-DM takes into account the order of the words, while PV-DBOW does not.

In our study we will consider the list of items with which a user has interacted, (1), as the equivalent of a document; there will hence be one document for each user. We can then use *doc2vec* to encode each user. In other words, we will identify a user's profile by the sequence of products with which they interacted.

3.3 Long-term and short-term profiles

In order to make some kinds of recommendations, information regarding consumption a long time ago may be useless, or even detrimental to the prediction. Consider, for example, the evolution of a person's musical tastes over the years. It is not unusual to find people who enjoy musical genres which they did not use to listen to when they were younger. The very same applies to tastes related to food consumption.

For this reason, we will codify two different user profiles: a long-term and a short-term profile. The long-term profile is expected to collect the tastes of a user over a long period of time, while the short-term profile is devised to gather the tastes expressed more recently by the user.

Essentially, the procedure to obtain these profiles will be the same for both, the difference residing in the data used to train the *doc2vec* algorithm. Thus, in order to obtain a long-term profile, we will use all available information about a user's interactions with the products, regardless of the moment each interaction

occurred. On the other hand, the short-term profile will be obtained using only the most recent interactions. In general, the threshold to determine which interactions are considered recent will depend on the type of items we are dealing with.

4 Recommendation tasks

With the aim of testing the relevance of the order in the interaction with items, we will pose two recommendation tasks that will be approached using different coding strategies for the items and for the users. Some of these strategies take into account the order (*word2vec* and the codification of user profiles with *doc2vec*) and they will be compared with the embeddings obtained during the learning task from one-hot vectors, which is a codification that does not consider the order in the interactions. The results obtained in these two tasks with these variations in the codification of users and items are discussed in Section 5.

4.1 Task 1: Will the user interact with a specific product?

To solve this task, we have a training set whose examples are triplets that contain a user, a product and a label to indicate whether that user has interacted or not with said product. In the rest of the paper, we will use $\boldsymbol{u}$ and $\boldsymbol{p}$ to denote the vectors which encode a user and an item, respectively. Thus,

$$\mathcal{D}_1 = \{(\boldsymbol{u}, \boldsymbol{p}, z) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{p} \in \mathcal{P}, z \in \{+1, -1\}\}. \tag{2}$$

We will solve this learning task by estimating the following probability:

$$\begin{aligned} \Pr(z|\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V}) &= \sigma(z \cdot g(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V})), \\ \text{where } \sigma(x) &= \frac{1}{1+e^{-x}}, \end{aligned} \tag{3}$$

where $\boldsymbol{W}$ and $\boldsymbol{V}$ are parameters that must be found using the *Maximum a Posteriori Probability (MAP)* estimate, and g is a compatibility function between users and products, defined as the following scalar product:

$$\mathrm{g}(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V}) = \langle \boldsymbol{W}\boldsymbol{u}, \boldsymbol{V}\boldsymbol{p} \rangle. \tag{4}$$

To enhance the expressiveness of the learned model, we will add a bias.

The loss function to be minimized, in this case, is

$$-\log \prod_{(\boldsymbol{u},\boldsymbol{p},z)} \Pr(z|\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{W}, \boldsymbol{V})) = \log\left(1+e^{-z\langle \boldsymbol{W}\boldsymbol{u}, \boldsymbol{V}\boldsymbol{p}\rangle}\right), \tag{5}$$

also known as *softplus*.

4.2 Task 2: What will be the next product?

The second task to solve is to try to anticipate the user's immediate next interaction. In this case, the training set will consist of triplets containing the user, $\boldsymbol{u}$, and two products with which the user has interacted consecutively, first with $\boldsymbol{p}_i$ and then with $\boldsymbol{p}_j$,

$$\mathcal{D}_2 = \{(\boldsymbol{u}, \boldsymbol{p}_i, \boldsymbol{p}_j) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{P}\}. \tag{6}$$

To solve this task, we will learn the following function:

$$\Pr(y = j|\boldsymbol{u}, \boldsymbol{p}_i, \theta) = \frac{e^{v_j}}{\sum_k e^{v_k}} = \mathrm{softmax}(v)_j \tag{7}$$

where j is the identifier of product $\boldsymbol{p}_j$, k is the identifier of any product, θ is the set of parameters (weights of a neural network, for example) that must be learned, and v is the output value of the network whose input is the concatenation of the vectors representing the user and the product.

The usual loss function to be minimized in learning tasks of this kind is *cross entropy*. However, when the number of classes is high, which is common in recommendation tasks, the calculation of $\mathrm{softmax}(v)$ is very expensive. Therefore, it is common to resort to estimation strategies, such as *noise-contrastive estimation (NCE)* [10], which we have used in this paper.

5 Experimental results

In this section, we describe and analyze the experimental results we have obtained using different codification schemas for users and items. More specifically, we analyze the performance of 7 codifications to represent the user profiles and 2 for the items.

5.1 Dataset description

For the experimentation, we used a publicly available dataset downloaded from the *www.last.fm* website, which has been previously published in Chapter 3 of the book [3] [1]. The set contains a log of music reproductions of 992 users over approximately 5 years. The log contains more than 19 million reproductions on a set of one and a half million songs. It also includes the exact date and time of each reproduction, so it is straightforward to extract an ordered list for each user, as defined in (1).

[1] http://www.dtic.upf.edu/∼ocelma/MusicRecommendationDataset/lastfm-1K.html

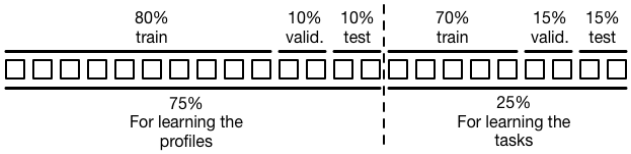

Figure 2 Method used for splitting each user's playlist into training, validation and test sets

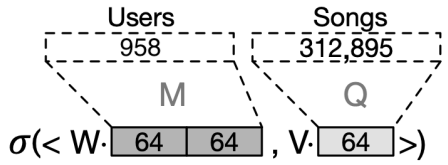

Figure 3 Network designed for learning task 1 (Section 5.4). The vectors that encode the user profile (dark gray) and the song (light gray) can be precalculated by *doc2vec* and *word2vec*, respectively, or they can be learned ad-hoc to solve the task, starting from the one-hot representation and optimizing the parameters $\boldsymbol{M}$ and $\boldsymbol{Q}$

5.2 Experimental setting

We filtered the data, identifying songs logged with different names, removing those that were heard less than 10 times, and users who listened to less than 100 songs. This filtering reduced the original dataset down to 958 users and 312,895 songs.

We proceeded as follows to build the corresponding training, validation and test sets:

- Each playlist associated with a user was split into two parts, one part was used to obtain the codification of the user profile (75%) while the other to train the recommender system in the tasks described in 4.1 and 4.2 (25%), see Figure 2. It is important to separate the data used for testing the performance of the profiles from the data used to build them. Otherwise, the results in the learning tasks could be biased, as we will be testing with data already seen in the construction of the profiles, and the results would be optimistic.
- Each of these two parts was in turn split into training/validation/test sets, using the training and validation parts to search for hyper-parameters with better performance in the corresponding optimization tasks (the encoding of users/songs, and the learning tasks).

The results shown in Tables 1 and 2 were obtained with the models trained with the most promising hyper-parameters, evaluating their performance on the test sets.

All the algorithms used in this article were implemented in Python using the TensorFlow [1] library, and the SGD-Adam [13] optimizer with *early stopping* [2].

5.3 Encoding songs and users

As explained in Section 3.1, we used *word2vec* to encode the songs. We used a window size of 2 to obtain vectors in a 64-dimensional space; i.e. each one-hot vector representing a song in a space with 312895 dimensions (the cardinality of the set of songs) is encoded in a new vector with 64 dimensions. These values for the window size and the encoding dimensionality showed the best results on the validation set.

Once the songs were encoded, we obtained the long-term and short-term user profiles by means of the two *doc2vec* frameworks, PV-DM and PV-DBOW. This yielded 3 different encodings for the long-term profile of each user: using PV-DM ($\mathbb{L}_{\text{dm}}$), using PV-DBOW ($\mathbb{L}_{\text{dbow}}$), and using both codifications concatenated ($\mathbb{L}$), as suggested by the authors of the *doc2vec* algorithm. The same encodings were obtained for the short-term profiles, i.e. $\mathbb{S}_{\text{dm}}$, $\mathbb{S}_{\text{dbow}}$ and $\mathbb{S}$.

We used a 64-dimensional space for both frameworks, so the encoding of $\mathbb{L}$ and $\mathbb{S}$ projects users into a space with 128 dimensions. We used a window size of 2 songs when using PV-DM. For the PV-DBOW encodings, we used the whole list of songs listened to by each user (the original approach uses a sample of words belonging to a document, but in our case, the list of songs is short enough to be used without sampling).

Long-term profiles were obtained using all the available data, i.e. the whole list of songs for each user in the training data set, while short-term profiles were encoded with the songs included in the training set during the last month logged for each user.

Finally, we also obtained an encoding in which the vectors of the users and products, in one-hot format, are projected in a space of 64 dimensions for songs and 128 dimensions for users. We used the same space dimensionalities as those of the profiles obtained with *word2vec* and *doc2vec* to guarantee that the spaces have the same encoding capacity in order to ensure a fair comparison. The projection is made via an embedding resulting from the learning process of each recommendation task, i.e. the computation of two matrices, $\boldsymbol{M}$ and $\boldsymbol{Q}$ (see Figures 3 and 4).

5.4 Results in Task 1

For the task of learning a model to predict whether a user will listen to a song in the future, we solve the optimization problem posed in Section 4.1. The set $\mathcal{D}_1$, defined in (2), is constructed using 25% of the data (see Figure 2) as follows:

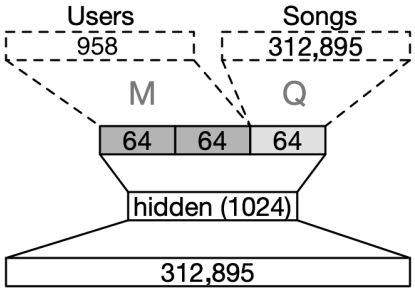

Figure 4 Network designed for learning task 2 (Section 5.5). The different encodings for users and songs are the same as for task 1

Table 1 Scores achieved with different encoding strategies in learning task 1. $\mathbb{L}$ denotes the codification of users by concatenating the PV-DM and PV-DBOW encodings; i.e. $\mathbb{L} = \mathbb{L}_{\text{dm}} \oplus \mathbb{L}_{\text{dbow}}$ (respectively, $\mathbb{S} = \mathbb{S}_{\text{dm}} \oplus \mathbb{S}_{\text{dbow}}$)

Encoding				
User	Song	Precision	Recall	F_1
one-hot	one-hot	77.5	73.8	75.6
one-hot	*word2vec*	**83.1**	**80.2**	**81.6**
$\mathbb{L}$	one-hot	81.5	76.1	78.7
$\mathbb{L}$	*word2vec*	80.5	79.1	79.8
$\mathbb{L}_{\text{dm}}$	*word2vec*	77.9	76.7	77.3
$\mathbb{L}_{\text{dbow}}$	*word2vec*	79.9	78.5	79.2
$\mathbb{S}$	one-hot	79.4	73.3	76.3
$\mathbb{S}$	*word2vec*	77.5	74.5	76.0
$\mathbb{S}_{\text{dm}}$	*word2vec*	70.4	68.1	69.2
$\mathbb{S}_{\text{dbow}}$	*word2vec*	75.4	73.7	74.6

- We include a positive example (labeled with +1) for each user-song pair, provided the song was listened to by the user.
- We create a negative example (labeled with −1) for each positive example, replacing the song with another, randomly sampled from the data, and not listened to by the user.

Table 1 shows the results, measured in terms of *Precision*, *Recall*, and their harmonic mean, F_1. Note that the best scores in all measures (in bold typeface) are achieved when songs are encoded with *word2vec* and users are encoded with an ad-hoc embedding.

The encoding of user profiles depending on the history of their interactions with songs is not especially helpful in solving this task. Neither the long- nor the short-term variations achieved the scores obtained by the ad-hoc encoding. However, all the long-term profile variations ($\mathbb{L}$, $\mathbb{L}_{\text{dm}}$, $\mathbb{L}_{\text{dbow}}$) perform better than their short-term counterparts ($\mathbb{S}$, $\mathbb{S}_{\text{dm}}$, $\mathbb{S}_{\text{dbow}}$). This indicates that predicting whether a song will be listened to in the future is related more to the user's tastes from a global point of view, than to the tastes implicitly expressed in the songs heard recently (in the last month).

In addition, it was found that taking into account the order for the encoding of user profiles is not relevant for this task, as both $\mathbb{L}_{\text{dbow}}$ and $\mathbb{S}_{\text{dbow}}$ perform better than their corresponding PV-DM versions, $\mathbb{L}_{\text{dm}}$ and $\mathbb{S}_{\text{dm}}$ (recall that the PV-DBOW framework does not consider any order, while PV-DM does).

5.5 Results in Task 2

To solve the optimization problem in Section 4.2 (predict the next song to play), we will use a network with the structure shown in Figure 4. This network will be trained with the set $\mathcal{D}_2$, defined in (6), in which the consecutive pairs of reproduced songs are taken from the part of the data reserved for learning tasks (25%), while the rest is used to obtain the encodings with *word2vec* and *doc2vec*, as we did in task 1.

To evaluate the performance of the different combinations of profiles, the *precision at x* ($P@x$) measure was used, where $x \in \{5, 10, 20, 50, 100, 1000\}$. This measure indicate the percentage of times that the song that we want to predict is among the x songs with the highest probability predicted by our learned model.

The results in Table 2 once again show that the use of a precomputed encoding for songs (using *word2vec*) improves performance considerably. For user profiles, the best results are obtained with the embedding resulting from the one-hot input representation, calculated ad-hoc during the learning of the task. If we compare short-term vs. long-term profiles, we can see that the short-term profile performs better. This makes sense, given the nature of the task: predicting the next song to be listened to is related more to recently heard songs than to songs heard months ago.

It should be noted that the values of $P@5$ (below 25% in all cases) are due to the fact that the recommendation task is very difficult: with only 5 recommendations, the system is asked to guess the next song among 312,895 possible alternatives. A random guess would yield a score of $P@5 = 1.6^{-3}$, much lower than the worst of our results.

The last column in Table 2 shows the median of the position of the correct song that we have to predict ($\boldsymbol{p}_j$ in Equation 6), considering the ranking of all songs ordered by the probability given by our corresponding models. The lower this value, the better the predictions. Ideally, we would like to predict the correct song with the highest probability, so that it would be in position 1. The best median result obtained is 366, quite good considering that the full ranking exceeds three hundred thousand songs.

If we compare the user profiles encoded with the approaches that take into account the order ($\mathbb{L}_{\text{dm}}$ and $\mathbb{S}_{\text{dm}}$) versus those that do not ($\mathbb{L}_{\text{dbow}}$ and $\mathbb{S}_{\text{dbow}}$), it can

Table 2 Scores achieved using different encoding strategies in learning task 2. $P@x$ is the percentage of times the right song is among the x most probable songs predicted. Median indicates the median position for the true song when ranking all songs (312895) by their predicted probability (the lower the better)

Representation		Precision						
User	Song	$P@5$	$P@10$	$P@20$	$P@50$	$P@100$	$P@1000$	Median
one-hot	one-hot	10.7	13.1	16.1	21.4	26.7	53.1	787
one-hot	*word2vec*	**24.0**	**28.3**	**32.5**	**37.5**	**41.4**	58.3	**366**
$\mathbb{L}$	one-hot	1.1	2.2	3.9	7.9	12.8	44.8	1371
$\mathbb{L}$	*word2vec*	11.8	15.8	20.3	26.9	32.5	56.8	575
$\mathbb{L}_{\text{dm}}$	*word2vec*	13.5	17.6	22.1	28.4	33.9	57.7	522
$\mathbb{L}_{\text{dbow}}$	*word2vec*	13.8	18.0	22.3	28.4	33.7	56.7	566
$\mathbb{S}$	one-hot	5.1	7.1	10.0	15.4	21.2	50.0	1001
$\mathbb{S}$	*word2vec*	12.9	17.2	21.8	28.3	33.5	56.5	561
$\mathbb{S}_{\text{dm}}$	*word2vec*	17.3	21.6	26.0	31.9	37.0	**58.5**	447
$\mathbb{S}_{\text{dbow}}$	*word2vec*	11.9	16.1	20.9	27.6	33.1	56.3	577

be seen that the order in the creation of the user profile seems to be relevant: $\mathbb{S}_{\text{dm}}$ clearly outperforms $\mathbb{S}_{\text{dbow}}$ in all the measures, and $\mathbb{L}_{\text{dm}}$ yields a similar precision, but with a better median. This makes sense, as this learning task (predicting the next song) clearly depends on the sequence of songs previously heard. Moreover, short-term profiles outperform their corresponding long-term versions in all the measures, indicating that recently heard songs are more informative when predicting the next one. The exception is for the PV-DBOW-based profiles, which perform slightly better using long-term profiles. This phenomenon is also reasonable and may be interpreted as: "if order is not going to be considered, then it is better to know as much as possible from the user's tastes".

5.6 Discussion

The scores show that the performance of our approach is improved when songs are encoded using *word2vec*. This is an expected result, considering that *word2vec* yields an encoding based on sequences, i.e. taking into account (to some extent) the order in which songs are heard.

We usually listen to lists of songs that have some relation between them: they are either of the same genre, or by the same artist, or they are included in the same playlist (which, by the way, is usually made up of songs with a similar style). Thus, a sequence of songs heard by a user can be seen as a sequence of words in a sentence in the sense that a given word appears in a text depending on the surrounding words in order to form a sentence with a given semantic meaning.

Word2vec gathers a kind of latent meaning of the songs, which helps in describing (encoding) them. In turn, this latent information helps in the prediction tasks.

On the other hand, *doc2vec* did not perform very well. It was not able to beat the scores obtained with the one-hot encoding of users. We think that this is because there was not enough data to model each user using these techniques. *Doc2vec* needs a very high number of examples in order to properly learn good user profiles.

6 Conclusions

Recommender Systems have taken on an important role in our highly digitized society. Their success resides mainly in the personalization that is being achieved in their recommendations. This is why the creation of product and user profiles is of paramount importance.

In this paper, we have evaluated the performance of profiles encoded with the *word2vec* and *doc2vec* algorithms, as well as with the embeddings obtained ad-hoc for solving a specific learning task. The comparison was made in the context of two different learning tasks: to predict whether a song will be listen to in the future, and to predict the next song to be heard.

The results show that the performance of a recommender system is substantially improved when songs are encoded using *word2vec*. However, the profile obtained for users using *doc2vec* does not improve the performance obtained with the ad-hoc embeddings obtained from one-hot vectors. Note also that short-term profiles perform better than long-term profiles when the nature of the learning task is related to short-term tastes (predicting the very next song). Conversely, long-term profiles perform better in a learning task where the outcome does not critically depend on recent interactions.

Furthermore, we may also conclude that it is important to take into account the order in the sequence of interactions with items for user profile encoding. This conclusion is supported by the fact that order-based

encodings ($\mathbb{L}_{\text{dm}}$ and $\mathbb{S}_{\text{dm}}$, both based on PV-DM) obtained better results than other user profile encodings (except for the ad-hoc embeddings, although this result can be explained by the fact that ad-hoc embeddings are especially learned to solve the corresponding task).

Our results suggest that it is worthwhile to continue exploring the possibilities of these profiles. Moreover, we belive LSTM [12] networks can constitute a promising alternative to *doc2vec* encoding due to their ability to deal with sequences.

Acknowledgments

We are grateful to NVIDIA Corporation for the donation of the Titan Xp GPU used in this research.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016)
2. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: Advances in neural information processing systems, pp. 402–408 (2001)
3. Celma, O.: Music Recommendation and Discovery in the Long Tail. Springer (2010)
4. Chan, N.N., Gaaloul, W., Tata, S.: A recommender system based on historical usage data for web service discovery. Service Oriented Computing and Applications **6**(1), 51–63 (2012)
5. Chen, S., Moore, J.L., Turnbull, D., Joachims, T.: Playlist prediction via metric embedding. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining, pp. 714–722. ACM (2012)
6. Díez, J., Martínez-Rego, D., Alonso-Betanzos, A., Luaces, O., Bahamonde, A.: Optimizing novelty and diversity in recommendations. Progress in Artificial Intelligence pp. 1–9 (2018)
7. Díez, J., Martínez-Rego, D., Alonso-Betanzos, A., Luaces, O., Bahamonde, A.: Metrical representation of readers and articles in a digital newspaper. In: 10th ACM Conference on Recommender Systems (RecSys 2016) Workshop on Profiling User Preferences for Dynamic Online and Real-Time Recommendations (RecProfile 2016). ACM (2016)
8. Fessahaye, F., Perez, L., Zhan, T., Zhang, R., Fossier, C., Markarian, R., Chiu, C., Zhan, J., Gewali, L., Oh, P.: T-recsys: A novel music recommendation system using deep learning. In: 2019 IEEE International Conference on Consumer Electronics (ICCE), pp. 1–6. IEEE (2019)
9. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 855–864. ACM (2016)
10. Gutmann, M., Hyvärinen, A.: Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. JMLR **13**, 307–361 (2012)
11. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. In: Proceedings of ACM Conference on Computer Supported Cooperative Work, pp. 241–250. ACM (2000)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997). DOI 10.1162/neco.1997.9.8.1735
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2014)
14. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. CoRR **abs/1405.4053** (2014)
15. Luaces, O., Díez, J., Joachims, T., Bahamonde, A.: Mapping preferences into euclidean space. Expert Systems with Applications **42**(22), 8588 – 8596 (2015)
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
17. Pazzani, M.J., Billsus, D.: The adaptive web. chap. Content-based Recommendation Systems, pp. 325–341. Springer-Verlag, Berlin (2007)
18. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 701–710. ACM (2014)
19. Pérez-Núñez, P., Luaces, O., Bahamonde, A., Díez, J.: Representaciones basadas en redes neuronales para tareas de recomendación. In: Proceedings of CAEPIA 2018, pp. 1179—1184. AEPIA (2018)
20. Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., Tang, J.: Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pp. 459–467. ACM (2018)
21. Resnick, P., Varian, H.R.: Recommender systems. Commun. ACM **40**(3), 56–58 (1997). DOI 10.1145/245108.245121
22. Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. In: Recommender systems handbook, pp. 1–35. Springer (2011)
23. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, pp. 1067–1077. International World Wide Web Conferences Steering Committee (2015)
24. Weston, J., Bengio, S., Hamel, P.: Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval. Journal of New Music Research **40**(4), 337–348 (2011)