Knowledge-Based Systems 280 (2023) 111016

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Creating synthetic datasets for collaborative filtering recommender systems using generative adversarial networks

Jesús Bobadilla [a], Abraham Gutiérrez [a], Raciel Yera [b], Luis Martínez [b,*]

[a] *Departamento de Sistemas Informáticos, ETSI Sistemas Informáticos, Universidad Politécnica de Madrid, C/ Alan Turing s/n, 28031, Madrid, Spain*
[b] *Departamento de Informática, Universidad of Jaén, Jaén, Spain*

A R T I C L E   I N F O

Keywords:
Recommender systems
Generative adversarial networks
Deep learning
Collaborative filtering

A B S T R A C T

Research and education in machine learning requires diverse, representative, and open datasets that contain sufficient samples to handle the necessary training, validation, and testing tasks. Currently, the Recommender Systems area includes a large number of subfields in which accuracy and beyond-accuracy quality measures are continuously being improved. To feed this research variety, it is both necessary and convenient to reinforce the existing datasets with synthetic ones. This paper proposes a Generative Adversarial Network (GAN)-based method to generate collaborative filtering datasets in a parameterized way by selecting their preferred number of users, items, samples, and stochastic variability. This parameterization cannot be performed using regular GANs. Our GAN model is fed with dense, short, and continuous embedding representations of items and users, instead of sparse, large, and discrete vectors, to ensure fast and accurate learning, as compared to the traditional approach based on large and sparse input vectors. The proposed architecture includes a DeepMF model to extract the dense user and item embeddings and a clustering process to convert the dense GAN generated samples to the discrete and sparse samples necessary to create each required synthetic dataset. The results from three different source datasets show adequate distributions and expected quality values and evolutions in the generated datasets compared to the source datasets. Synthetic datasets and source codes are available to researchers.

1. Introduction

Recommender systems (RS) are a relevant area in artificial intelligence due to the growing popularity of social networks. The big companies that extensively use RSs are TripAdvisor, Netflix, Spotify, YouTube Music, TikTok, YouTube and Amazon [1]. These companies make use of the RS models to recommend to users similar items (music, videos, trips, news) to those that they have already consumed; some other companies, such as Facebook, work hard to collect customer activity to provide personalized advertising rather than personalized products or services. RSs are usually classified according to their filtering approach [2]; content-based RSs select the recommended items by looking for similar content [3]; since most item contents is text, natural language processing models are used. Reviews [4] and tweets [5] are two common types of content-based filtered data. Product images can also be processed to make recommendations; convolutional neural networks are the most commonly used models to perform this task [6]. Social filtering has been extensively used to improve social-based recommendations. This type of filtering uses data such as tags, followers, and being followed, and makes use of the concepts of reputation and trust [7]. Geographic information, such as GPS coordinates and POI, is mainly used to support context-aware filtering [8]. Demographic filtering (age, gender, country, etc.) is commonly combined with other types of filtering, implementing recommendation ensembles [9]. *Beyond the previous filtering strategies, collaborative filtering (CF) [10] is the most important approach for implementing RSs, since it provides superior accuracy, particularly when combined with some other types of filtering.* Effective RS research makes use of innovative models, adequate quality measures, and representative datasets.

The historical evolution of CF begins with the use of memory-based models, mainly the K-Nearest Neighbors algorithm [11]. Memory-based approaches were replaced by model-based machine learning approaches due to their overall performance: they are superior in accuracy of results, also in time to obtain predictions (once the model has learned); and their output is capable of being explained through post-hoc techniques [12]. Matrix Factorization (MF) [13] is the most widely used machine learning model to implement collaborative filtering; it performs a dimensional reduction of users and items, capturing the

* Corresponding author.
E-mail addresses: jesus.bobadilla@upm.es (J. Bobadilla), abraham.gutierrez@upm.es (A. Gutiérrez), ryera@ujaen.es (R. Yera), martin@ujaen.es (L. Martínez).

https://doi.org/10.1016/j.knosys.2023.111016
Received 17 December 2022; Received in revised form 12 September 2023; Accepted 15 September 2023
Available online 23 September 2023

main patterns that relate them to the votes cast. Additionally, by using Non-Negative Matrix Factorization (NFM) [14], semantic meanings can be assigned to latent factors. Bayesian NMF [15] allows clustering users and making predictions simultaneously, which opens the door to effective recommendations to user groups and social clustering applications [16]. *Nowadays, CF research is mainly developed by deep learning models, where DeepMF [17] is the basis for modern approaches*. DeepMF is the model that we use in this paper, in which users are coded in a latent space by means of an embedding layer, whereas items are coded in a different latent space by means of a second embedding layer; finally, predictions are made by making the dot product of both, item and user embeddings. DeepMF improves MF due to the inherent competence of neural networks to capture the non-linear relations hips between samples. Neural Collaborative Filtering (NCF) [18] is extensively used to implement CF; this model replaces the DeepMF dot layer with a Multi-Layer Perceptron (MLP) and outperforms DeepMF when applied to large and complex datasets. Beyond accuracy, deep learning models are emerging to perform some innovative tasks, such as improving fairness, where the DeepFair model [19] achieves a trade-off between equity and precision; green computing [20]; results explanation via latent space visualization [21] and efficient neighborhood identification [22]. The adversarial network-based recommendation has recently been introduced in the RS area [23] and we will focus on it in the 'Related work' section. Generative Adversarial Networks (GAN) [24] are responsible for the popular fake faces and fake videos that flood social networks. Their architecture has two separate neural networks that compete against each other ('adversarial'), such as an art forger competing against an art expert, ensuring that both improve their work. The GAN 'forger' is a generator model that creates fake samples from random noise vectors, while the GAN 'expert' is a discriminator model implemented as a simple binary classifier: fake, non-fake. *However, while RS research is mainly focused on proposing novel recommendation models, this paper tries to make progress in CF datasets*.

In this respect, it is essential to identify quality measures as a key element to carry out adequate research, since they allow the baselines of the state of the art to be compared with the proposed algorithms, methods and models. Beyond the usual prediction and recommendation quality measures (MAE, MSD, precision, recall, F1, NDCG, etc.), some other measures, such as novelty and diversity [25], have recently acquired growing importance. Of these, diversity is currently the main focus of researchers' attention, due to the risks of inappropriate recommendations in social networks, such as those that exhibit a lack of variability and promote prefixed ideas and behaviors. Diversity and reliability in RS have been improved by introducing diversity-enhancing constraints in the MF model [26]; additionally, a deep learning classification model [27] is proposed to obtain the recommendation reliability values from the softmax output layer of the neural network. *Quality values are obtained when a model or method is tested on balanced CF datasets.* To obtain balanced training and testing sets, with respect to their user and item distributions, deterministic strategies are proposed in [28]. Most of the RS research makes use of popular CF datasets such as MovieLens, FilmTrust, MyAnimeList or CiteSeer; CF datasets include different domains such as music, movies, POIs, tourism, news, research papers, tagged data, etc. Some of these datasets have been filled with explicit votes from users, while others contain implicit interactions between users and systems. There are also datasets filled with crawled Web pages or academic PDFs [29] and some others are enriched with social tags that researchers add to the articles [30]. A selection of relevant social CF datasets is provided in [31] and related to some articles using them. Recently, an educational news dataset [32] was released; which included contextualized information: time and location. Finally, an RS dataset has also been provided that contains artificial intelligence research data [33] to obtain segmented information, clustering, and geographical locations. *Beyond these works, it is particularly relevant that parameterized synthetic datasets have not yet been used, so consequently the CF research does not benefit from the flexibility that parameterization provides in the experiment design:* different dataset sizes, number of users and items, and so on. This paper aims to fill the gap by proposing a procedure, coined as GANRS, which focuses on the use of GANs to generate collaborative filtering recommender systems datasets in a parameterized way. Please note that current RS GAN-based models cannot simultaneously set the number of generated users, items, and rating distributions.

Regarding our contribution, two main overall approaches can be identified in the state-of-art: statistical and generative. The main advantage of the statistical approach is that several relevant parameters can be simultaneously set: number of users, number of items, dataset size, etc. The main drawback of this approach is its poor accuracy. On the other hand, current model-based generative approaches improve accuracy compared to statistical frameworks, but they lack flexibility, since parameterization is very limited. In fact, current GAN designs are focused on user profiles, and they can generate as many new fake users as required, but other relevant parameters cannot be set, such as the number of items that is fixed in the source set of user vectors and then, also, in the fake generated set of user vectors. This can be explained with an example: when we run a regular GAN to generate fake images, the synthetic images have the same shape (resolution and number of channels) as the source images. In the RS field, the synthetic user vectors contain the same number of items as the real user vectors. Following the example, there are some specific GAN designs that return 'super-resolution' images (they can increase resolution), but to our knowledge there are no RS GANs designed to generate fake users containing more items (or fewer items). Our proposed method is designed to simultaneously set some CF relevant parameters, such as the number of users and items.

The rest of the paper has been structured as follows: related work is introduced in Section 2, focusing on the most recent uses of the GAN models applied to RS. Section 3 explains the proposed model and its formalization. Section 4 presents the design, result, and discussion of the experiments. Finally, Section 5 contains the main conclusions of the article and discusses future work.

2. Background

2.1. Basics on generative adversarial networks

GANs are designed to generate data from scratch [24]. They have been commonly used to create fake images, although their use has been spreading to many other domains: music, medicine, financial data, etc. The GAN architecture composes of two deep network models: generator and discriminator. The generator model learns to create samples as similar as possible to those in a dataset (e.g., a dataset with human faces images), whereas the discriminator model learns to detect fake samples (those samples created by the generator). To better understand the GANs we can consider the example of a painting forger and a forgery expert: the more imitations the forger paints, the better their results, and the better the expert's ability to detect fake paintings. Both people successively improve their abilities. When the learning begins, the GAN discriminator (the forgery expert, in our example) has an easy job, since the generator does not have the painting patterns. After thousands of learning epochs, the generator has learnt the patterns well enough to confuse the discriminator, who is forced to tune their weights. If the learning loop iterates enough times, both the generator and the discriminator models are well designed and the painting in the dataset contains suitable patterns, the generator will be able to create synthetic (fake) samples that are difficult to distinguish from the originals.

Fig. 1 shows the GAN architecture [24]; the discriminator model makes a binary classification between fake and real samples. The generator model updates its weights (learns) when the discriminator correctly classifies a fake sample. The discriminator model updates its weights when it incorrectly classifies a sample. Note that the generator takes a random noise distribution as input to generate samples; then,

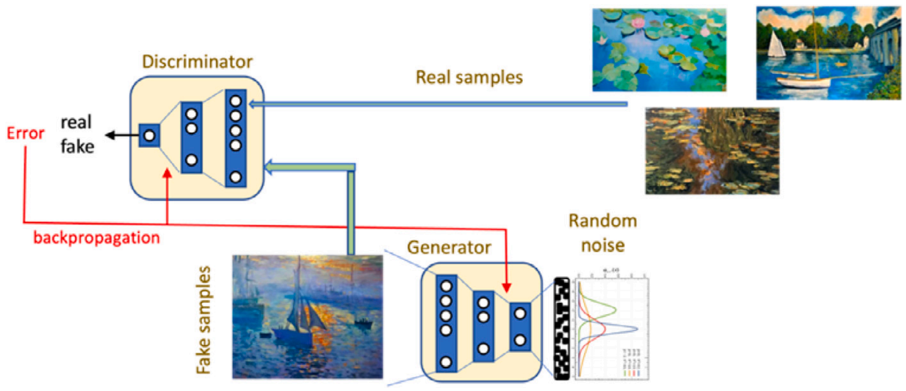

Fig. 1. Generative Adversarial Networks architecture.

once it has learnt, for each input random noise vector that feeds the generator a sample is created with the patterns of the dataset samples. For this reason, by providing random noise vectors we can create as many samples as required, which means that, in our context, we can create fake CF datasets of any size by creating fake profiles.

To measure GAN loss, we use cross-entropy. The discriminator (D) loss can be expressed as the sum of the expectations:

$$max_D V(D) = E_{x\sim p_{data(x)}}[log D(x)] + E_{z\sim p_{z(z)}}[log(1 - D(G(z)))] \quad (1)$$

Where the first term of the equation is used to recognize real images, and the second term recognizes generated images. Z represents the noisy vector, G is the generator, and $G(Z)$ is the generated sample. $D(G(z))$ is the classification result of the discriminator when its input is a fake sample. $D(x)$ is the classification result of the discriminator when its input is a real sample.

The generator loss is designed to learn when the discriminator correctly classifies (the true label is 1, and the fake label is 0). Its equation is:

$$min_G V(G) = E_{x\sim p_{data(x)}}[log(1 - D(G(z)))] \quad (2)$$

The GAN is a minimax in which G wants to minimize V while D wants to maximize it:

$$min_G max_D V(D, G) = E_{x\sim p_{data(x)}}[log D(x)] + E_{z\sim p_{z(z)}}[log(1 - D(G(z)))] \quad (3)$$

As in the previous example, GAN models act on non sparse values (e.g., pixels in a picture), but they are not designed to work with sparse vectors or matrices. Our problem here is that CF datasets contain extraordinarily sparse matrices of ratings (users only vote or consume a very limited number of the available items). Using the regular GAN architecture is not an adequate approach to addressing CF-based RS. This paper proposes an extended GAN architecture where embeddings are introduced to code the sparse and discrete vectors of votes to dense and continuous vectors. This innovation makes it possible to use a regular GAN to generate dense and continuous vectors efficiently and accurately. This compression stage forces us to design the corresponding stage to decompress the generated dense vectors. The adopted solution makes it possible to set both the number of users and items in the generated dataset, which is a relevant innovation in the state-of-art.

2.2. Related works

Generative deep learning is an innovative field in the CF RS area. Although some variational autoencoder approaches have been published [34,35], current research is mainly focused on GAN models [36]. A CF subfield where GANs are used is the attack/defense strategies [37], where these models can reinforce security in RS. Nevertheless, the most extended uses of CF GANs are: (a) to solve the issue of noisy data, and (b) to tackle the data sparsity problem, and implement a data augmentation framework by capturing the distribution of real data. CFGAN [38] is a model that generates purchase vectors rather than the IDs of items and then uses the generated fake purchase vectors to augment the real vectors. The Wasserstein version of CFGAN is the unified GAN (UGAN) [39] and reports improvements compared to CFGAN. To prioritize long and short-term RS information (interactions between users and items that change quickly or slowly), the PLASTIC [40] model trains a generator and uses it as a reinforcement learning agent. The recurrent GAN: RecGAN [41], learns temporal patterns in ratings; combining GAN and recurrent neural networks (RNNs) models. To capture negative sampling information in the CF datasets, IPGAN [42] implements two different generative models: one for positive instances and another for negative instances. IPGAN considers the relations between the positive ratings sampled and the negative ones selected.

Currently, the DCGAN model [43] combines GAN and reinforcement learning models to catch the information of the RS sessions, rather than the traditional historical matrices of votes from users to items. Session information includes the responses of users to current recommendations. The user's immediate feedback is managed by the reinforcement learning model combined with the GAN. The NCGAN [44] incorporates a neural network to extract nonlinear features from users, and a GAN to guide the recommendation training; the generator model makes user recommendations, whereas the discriminator model measures distances between real and generated distributions. An innovative method to improve the information flow from generator to discriminator [45] reduces the discrepancies between both models in the CF GAN. A regularization Wasserstein GAN model is used in [46], combined with an autoencoder acting as a generator, reporting accuracy improvement when applied to high-dimensional and sparse CF matrices. A CGAN (Conditional GAN) is used [47] to improve CF recommendations, and the sizes of the rating vectors can be set, simplifying the generator and discriminator tasks. Additionally, it allows conditional rating generation to be established. For datasets that do not follow standard Gaussian distributions, a missing data imputation based on GAN [48] is proposed; results show improved quality in several representative classification data sets. Trust information is used in [49] to make effective recommendations. They propose a GAN where the discriminator is an MLP model, and the generator is a long-short term memory network (LSTM) model [50]. Finally, CF datasets are usually imbalanced due to their social data collection (e.g: more young people than old people). To address this limitation, [51] proposes a Wasserstein GAN model in the generator, and the PacGAN concept in the discriminator [52], to minimize the mode collapse problem.

A platform for multi-agent RS simulation is the probabilistic-based RecSim [53], which generates synthetic profiles of users and items, and uses Markov chains and recurrent neural networks. The Virtual-Taobao [54] is a multiagent reinforcement learning system designed to improve search in the social Taobao website; it makes use of a

GAN to simulate internal distributions. A simple matrix factorization is used [55] to inject topic diversification into the recommendation process. The DataGenCars [56] is a Java-based generator of RS synthetic data; it contains a statistical basement that provides flexibility, but it returns low accuracy compared to deep learning generative models. Finally, the SynEvaRec framework [57] provides the generation of synthetic RS datasets using the Synthetic Data Vault (SVD) library. This library models multivariate distributions using copula functions; its CTGAN sub-library includes GAN models. The main advantage of SynEvaRec, compared to previous frameworks, is that it can use different RSs as a source; its main drawbacks are the poor quality of the results in most of the cases, and the excessive time it takes to perform the training stage.

Previous research mainly focuses on improving different objectives such as noise reduction, recommendation quality, prediction values, defense against attacks, or balancing data. To make this happen, many different approaches and information sources have been combined: the use of GAN, CGAN, Wasserstein GAN, etc. GAN models have been combined with Recurrent Neural Networks [58] and LSTM networks [50], and reinforcement learning has been introduced in the GAN-based architectures. Long and short data have been introduced to the proposed models, in addition to trust information, session logs, including responses of the users to previous recommendations, and inferred negative votes. The pure generation of synthetic datasets does not seem to be a goal in this novel field of GAN applied to CF RS, which is currently focused on improving prediction and recommendation quality results by means of data augmentation based on the inherent ability of the GAN model to capture the complex nonlinear patterns of high-dimensional and sparse CF datasets. The innovation of our proposal is to generate representative and useful CF synthetic datasets, rather than to improve the existing results that are of varying quality. Additionally, it allows representative parameters to be set and a whole 'family' of synthetic datasets to be obtained, taking real datasets as a source, such as Movielens, Netflix, or MyAnimeList. These parameters are the number of users, the number of items, the number of samples and the variability of the generated data. By varying the parameter values, we can generate different versions of the same CF pattern, such as a Movielens-based dataset containing 8000 users and 3000 items, or another that contains 2000 users and 1000 items, among others. In this way, we can test the accuracy and performance impact of the dataset size, its sparsity, its number of users and items, as well as check the improvement of the MAE when the number of users increases. As far as we know, there are no published methods or models for creating, in a parameterized deep learning model, accurate and scalable synthetic datasets from diverse sources.

3. The generative adversarial networks-based approach for datasets building in collaborative filtering

As previously mentioned in the Introduction section, our research problem is defined as obtaining a larger, scalable synthetic dataset from an original RS dataset that synthesizes similar user behavior and valuation patterns in relation to the original dataset. In addition, it is desirable that such generation be parameterized, allowing the number of users, items, samples and variability of the distribution to be controlled.

Next, this section proposes the GANRS method, which uses a GAN network to generate synthetic CF datasets; the GAN is fed with a real CF dataset and the model learns its internal patterns. The most innovative contribution is to feed the GAN with dense and small embedding representations of users and items, instead of the traditional approach where the GAN inputs are large and comprise sparse vectors containing the votes cast for each user. The main advantage of the GANRS method is that it greatly reduces the complexity of the GAN architecture, its convergence speed, and its performance.

The traditional and sparse-based GAN architectures deal with very large input vectors: as large as the number of items in the dataset, which can be in the tens of thousands, and require a very large dense layer in the model to hold this huge amount of data. What is more, between 97% to 99% of the data is usually missing, since users only vote for or consume a tiny proportion of the available products or services, hence the extraordinary sparsity in the CF datasets. Following the huge dense layer, in classical GAN architectures, it is necessary to stack a large multilayer perceptron to reduce dimensionality. By comparison, the proposed model replaces the large dense layer with two embeddings, one to code users and the other to code items (borrowed from the DeepMF model in the first stage of the proposed method). Embedding layers are specifically designed to deal with sparse data; they receive integer values (user and item IDs, in our case), and they provide small embedding representations (typically 5 to 15 float values in the CF scenarios). Related users or items share similar embedding representations, and this feature allows for extraordinarily simplification of the model. Overall, the proposed architecture is much smaller than traditional architectures, it contains far fewer parameters, and consequently, learns faster. Additionally, it better captures the complex nonlinear relations between items and users, in the same way that non-GAN RS models do to improve predictions.

The formalization of the GANRS method is presented and structured according to the following seven stages, also illustrated in Fig. 2:

- Stage 0. CF definitions

  1 Let U be the set of users who make use of a CF RS.
  2 Let I be the set of items available for voting in the CF RS.
  3 Let V be the range of allowed votes; usually $V = \{1, 2, 3, 4, 5\}$.
  4 Let S be the set of samples contained in the CF dataset; in which $N = |S| = the\ total\ number\ of\ votes\ cast$.
  5 $S = \{\langle u, i, v\rangle_1, \langle u, i, v\rangle_2, \ldots, \langle u, i, v\rangle_N\}$; where each $u \in \{1, \ldots, |U|\}$, each $i \in \{1, \ldots, |I|\}$, and each $v \in \{1, \ldots, |V|\}$.

- Stage 1. DeepMF training

  6 Let E be the size of two neural layer embeddings used to vectorize each user and each item belonging to U and I, respectively.
  7 Let $f^{eu}(u) = [e^u_0, e^u_1, \ldots, e^u_E]$, where f^{eu} is the embedding layer output of the users, where $u \in \{1, \ldots, |U|\}$.
  8 Let $f^{ei}(i) = [e^i_0, e^i_1, \ldots, e^i_E]$, where f^{ei} is the embedding layer output of the items, where $i \in \{1, \ldots, |I|\}$. By combining both dense vectors of user and item embeddings: $([e^u_0, e^u_1, \ldots, e^u_E]$ and $[e^i_0, e^i_1, \ldots, e^i_E])$, we can make rating predictions in the DeepMF training stage. The dot product of the user embedding and the item embedding in each $\langle u, i, v\rangle_j \in S$ provides its rating prediction:
  9 $\hat{y}_j = f^{eu}(u) \cdot f^{ei}(i) = [e^u_0, e^u_1, \ldots, e^u_E] \cdot [e^i_0, e^i_1, \ldots, e^i_E]$
  10 $\frac{1}{2}(y_j - \hat{y}_j)^2$ is the output error used in the DeepMF neural network to start the backpropagation algorithm, where the neural weights are iteratively improved from the δ_j values: $\triangle\ w_{ji} = \alpha y_j f'(Net_i) \sum_k w_{ik}\delta_k$, when k is a hidden layer, and $\triangle\ w_{ji} = \alpha y_i f'(Net_i)\frac{1}{2}(y_k - \hat{y}_k)^2$, if k is the output layer. i, j, and k are successive sequential layers. Net_i represents the cumulative input received for an artificial neuron, $Net_i = \sum_j y_j * w_j$, where j is the index of the neurons in the layer preceding the current neuron.

- Stage 2. DeepFM feedforward
  Once the DeepMF has learned, we can collect the embedding representation of each user and each item in the CF RS.

  11 Let $E^* = \{\langle u, [e^u_0, e^u_1, \ldots, e^u_E]\rangle, \forall u \in U\}$, be the set of embeddings for all the RS users. ($u \in [1\ldots\#U]$, one to u)
  12 Let $E^*(u) = [e^u_0, e^u_1, \ldots, e^u_E]$

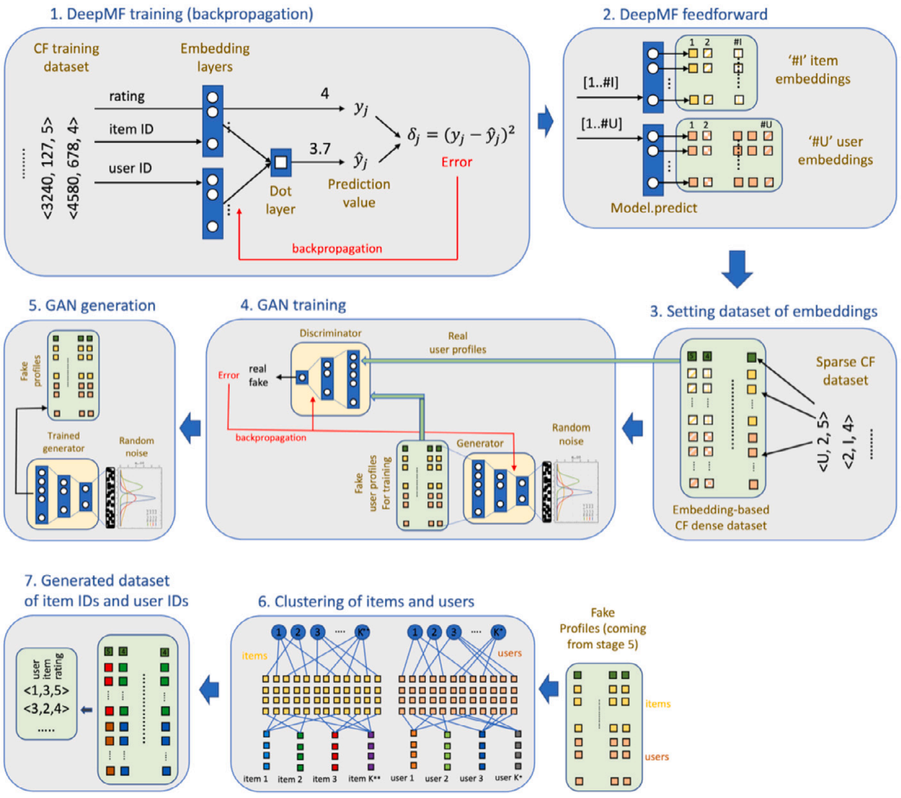

Fig. 2. The stages of the proposed GANRS method.

13 Let $E^{**} = \{\langle i, [e^i_0, e^i_1, \dots, e^i_E]\rangle, \forall i \in I\}$, be the set of embeddings for all the RS items. ($i \in [1...\#I]$, one to i)

14 Let $E^{**}(i) = [e^i_0, e^i_1, \dots, e^i_E]$

- Stage 3. Setting the dataset of embeddings

15 Let $R = [\langle E^*(u), E^{**}(i), v\rangle], \forall\langle u, i, v\rangle_j \in S$ be the embedding-based dataset of real samples.

- Stage 4. GAN training

16 Let f^D be the discriminator D model belonging to a GAN model.

17 Let f^G be the generator G model belonging to a GAN model.

18 Let f^{GD} be the optimization function of the GAN model; $f^{GD} = Min_G Max_D f(D, G) = E_R[log(D(R))] + E_z[log(1 - D(G(z)))]$, where E_R is the expected value for real samples, z is the random noise that feeds the generator G, and E_z is the expected value for the generated fake profiles $G(z)$. Note that R refers to [15].

- Stage 5. GAN generation

19 Let $F = f^G(z)$ be the generated dataset of fake samples from different random noise vectors z.

- Stage 6. Clustering of items and users.

20 Let K^* be the number of clusters used to group the embeddings of the users.

21 Let K^{**} be the number of clusters used to group the embeddings of the items.

22 Let $h^*(u) = c | c \in \{1, \dots, K^*\}$, be the clustering operation that assigns a centroid to each user.

23 Let $h^{**}(i) = c | c \in \{1, \dots, K^{**}\}$, be the clustering operation that assigns a centroid to each item.

- Stage 7. Setting dataset of item IDs and user IDs

24 Let H be the item IDs and users IDs discrete dataset obtained from the embedding-based dataset F of fake samples. $H = \{\langle h^*(u), h^{**}(i), v\rangle | \forall\langle E^*(u), E^{**}(i), v\rangle \in F\}$

25 Let $S = \{H\}$ be the synthetic generated dataset version of H where duplicated samples are removed.

26 Let $G' = \{\langle h^*(u), h^{**}(i), v\rangle \in H | \nexists\langle h^*(u'), h^{**}(i'), v'\rangle \in H$ where $h^*(u) = h^*(u') \wedge h^*(i) = h^{**}(i) \wedge v \neq v'\}$

Fig. 2 shows the seven designed stages to generate different synthetic datasets from real datasets (Movielens, Netflix, etc.). Stage 1 (top left graph in Fig. 2) shows the training of a DeepMF model used to set both the embedding layer of users and the embedding layer of items. Basically, embedding layers in a neural network efficiently convert an input from a sparse representation into an output dense representation. For each input sample $\langle user, item, rating\rangle$ in the training set, the output dot layer combines the embedding layer values to predict the rating value and to obtain the output error "(rating - prediction)" that will we backpropagated to update the learning parameters. Steps 6 to 10 formalize these concepts. Once the DeepMF model has learned, Stage 2 (top right graph in Fig. 2) shows the DeepMF feedforward process where each item ID (from one to the number of items in the dataset, range $[1...\#I]$) feeds the item embedding, which outputs the item ID dense representation; usually, CF embedding vectors have a size from 5 to 10. The same applies to user IDs as input and their output dense representations. Please note that the number of items in the dataset will be different from the number of users. Steps 11 to 14 explain this second stage.

The purpose of the third stage is to convert the source sparse CF dataset into its dense representation. To accomplish the task, for each source $\langle user, item, rating\rangle$ sample in the dataset (e.g.: $\langle 8920, 345, 4\rangle$) we replace the user ID (8920 in this example) with its related dense representation; the same applies for the item ID. Using embeddings of size 5, the result in the example could be such as:

Table 1
Example of samples representation.

Sparse	Dense
$< \mathbf{890}, 47, 5 >$	$< [\mathbf{0.03}, \mathbf{0.94}, \mathbf{1.02}, \mathbf{0.87}, \mathbf{-0.78}], [-1.23, 0.99, 1.02, 0.65, -0.48], 5 >$
$< \mathbf{890}, \mathbf{31}, 4 >$	$< [\mathbf{0.02}, \mathbf{0.95}, \mathbf{0.99}, \mathbf{0.81}, \mathbf{-0.69}], [\mathbf{0.45}, \mathbf{-0.78}, \mathbf{0.83}, \mathbf{-0.15}, \mathbf{0.09}], 4 >$
$< 968, \mathbf{31}, 4 >$	$< [-1.04, 0.04, 0.66, -0.67, 0.11], [\mathbf{0.42}, \mathbf{-0.71}, \mathbf{0.80}, \mathbf{-0.10}, \mathbf{0.14}], 4 >$
$< 123, \mathbf{31}, 2 >$	$< [1.56, -1.12, 0.33, 1.22, -0.87], [\mathbf{0.43}, \mathbf{-0.75}, \mathbf{0.80}, \mathbf{-0.11}, \mathbf{0.06}], 2 >$

Table 2
Main parameter values of the tested datasets.

Dataset	#users	#items	#ratings	Scores	Sparsity
Movielens 100K	943	1682	99,831	1 to 5	93.71
Netflix*	23,012	1,750	535,421	1 to 5	98.68
MyAnimeList	19,179	2,692	548,967	1 to 10	98.94

$\langle[0.03, 0.94, 1.02, 0.87, -0.78], [-1.23, 0.99, 1.02, 0.65, -0.48], 4\rangle$.

Stage 3 in Fig. 2 shows an illustrative example. Step 15 formalizes the operation. The dense dataset obtained will be used in Stage 4 to train a GAN capable of generating fake user and item profiles, as well as their associated rating values, which will be, even at this stage, the ratings that will be in the dataset generated at the end of the proposal. Our GAN will use the Stage 3 dense dataset to train the discriminator by providing it with the necessary real samples. The GAN generator takes Gaussian random noise as input and iteratively learns how to generate increasingly good fake profiles capable of cheating the discriminator model. Once the generator and the discriminator have learnt, the generator can convert input noise vectors into dense samples that mimic the patterns of the real dataset provided in stage 3. Stage 4 is formalized in steps 16 to 18.

The last stage in Fig. 2 (bottom left graph) uses the trained GAN generator model (Stage 4) to generate as many fake samples as desired. We feed the generator with successive vectors of random noise values following a Gaussian distribution, and the generator outputs successive fake dense samples following the patterns of the real dataset (obtained in Stage 3). The higher the standard deviation of the Gaussian distribution, the higher the variety of individual values in the generated dense fake samples. As an example, a low standard deviation value in the random noise Gaussian distribution leads to a higher proportion of votes '3' (ranging from 1 to 5), while choosing a high standard deviation value will produce a higher density of votes '5' and '1'. Ratings are generated in the same way as items and users: they are coded in the dense embedding generated by the GAN. Synthetic ratings are continuous values, whereas real ratings are discrete, usually in the range $\{1, \dots, 5\}$. To make this conversion, a function assigns the maximum value in the range (usually '5') to the synthetic continuous values greater than it; analogously the function assigns the minimum value (usually '1') to the continuous values lower than it. Finally, a round function is performed to ensure discrete values. Step 19 formalizes the generation of fake samples.

Although the GANRS method could be considered complete, this is not the case because our goal is to generate fake datasets of sparse samples (such as Movielens or Netflix); it is then necessary to convert from the obtained dense representation in Stage 5 to the usual sparse representation seen in Stage 1. The process is not straightforward, since all the dense representations of the fake samples are different from each other; this will be better explained using the example in Table 1: it can be observed that user 890 (two first rows) has very similar dense embedding values, but there are not identical, since the GAN generator is not able to create the same exact values from the noise input vectors. The same situation occurs in Table 1 for the item with ID 31. Consequently, the GANRS method provides a way to 'group' similar dense embeddings into a unique ID; that is, to convert the dense bold vectors of the user in Table 1 into a unique user ID (need not be 890), and the dense bold vectors of the item into a unique item ID (need not be 31, either).

To group similar dense embeddings into a unique ID, a K-Means clustering [59] has been chosen. This algorithm has the relevant feature that a number K of clusters must be chosen a priori, and it is very convenient in this context, since, in this way, we will have the opportunity to establish the number of users and the number of items in the GANRS synthetic generated dataset. Stage 6 of Fig. 2 shows this concept, where K^* has been selected as number of users and K^{**} has been selected as number of items. Two separate K-Means processes are run: one to group user embeddings, and the other to group item embeddings. Steps 20 to 23 formalize these two clustering processes. To better understand this stage, we can consider an example where one million fake samples have been generated and we want to create a synthetic dataset containing two thousand fake users and one thousand fake items. To accomplish this task, we should obtain two thousand groups collected from the one million user vectors (the same for the one thousand item groups). On average, five hundred user vectors could be assigned to each user group (and, analogously, one thousand item vectors to each item group), but we know that this depends on the user and item vector patterns. To adequately accomplish the grouping task, machine learning provides us with clustering algorithms, of which the k-means allow us to set the number of desired groups (two thousand for users and one thousand for items, in our example). Running both clustering processes (one for users and the other for items) we can assign a fake user ID to all the fake user vectors in each cluster. Please note that the ID number can be assigned at random to each of the two thousand clusters (the same for the one thousand item IDs).

Fig. 3 illustrates the concept where graphs at the top show the two k-means clustering processing performed in the proposed model: one to group item vectors (yellow circles), and the other one to group user vectors (orange circles). Gray ellipses represent the k-means clustering groups. All the fake user vectors in each cluster collapse into the same user vector, which codes a sample representative of its group, and is different from the samples in the rest of the clusters (same for items). In this way, we obtain the selected representative K* users and K** items. The graphs at the bottom in Fig. 3 show the final stages of the proposed method; Stage 6 draws the K* clusters of users and the K** clusters of items, from the previous clustering with blue circles. Each of the K* clusters (of users) groups a set of user vectors (columns of orange squares), and each of the K** clusters (of items) groups a set of item vectors (columns of yellow squares). Each cluster of user vectors collapses into a representative user: at the bottom of Stage 6 graph (the same for items). Once the representative users and items are set, we can generate the fake dataset of embeddings by translating each generated embedding sample (bottom-right graph) to its equivalent representative concatenated embedding of representative (collapsed) users and items (at the bottom of the Stage 6 graph). Previously, we illustrated a case where a generated sample collapses its item vector in the "item 3" representative code (vector of red squares), and it collapses its user vector in the "user 1" representative code (vector of brown squares). In stage 7 the complete embedding, and the translation to the $\langle 1, 3, 5\rangle$ sparse tuple codification can be seem. This is also true for the following fake embedding, which collapses in the "item 1" (green) and "user 3" (blue), generating the sparse tuple < 3,2,4>. Note that the GAN-generated profiles (bottom-right in Fig. 3) are not limited to a fixed number of users and items, whereas their Stage 7 version (bottom-left in Fig. 3) are limited to the ranges $\{1, \dots, K*\}$, and $\{1, \dots, K**\}$, making it possible to preset the number of users and items of the synthetic dataset.

The seventh stage in Fig. 2 converts dense fake samples (coming from Stage 5) into sparse samples $\langle user, item, rating\rangle$. To accomplish this task, for each sample in the dense representation we replace its user vector with its centroid number (from 1 to K^*) and its item vector with its centroid number (from 1 to K^{**}); the rating value remains the same as that already generated by the framework in Stages 4–5.

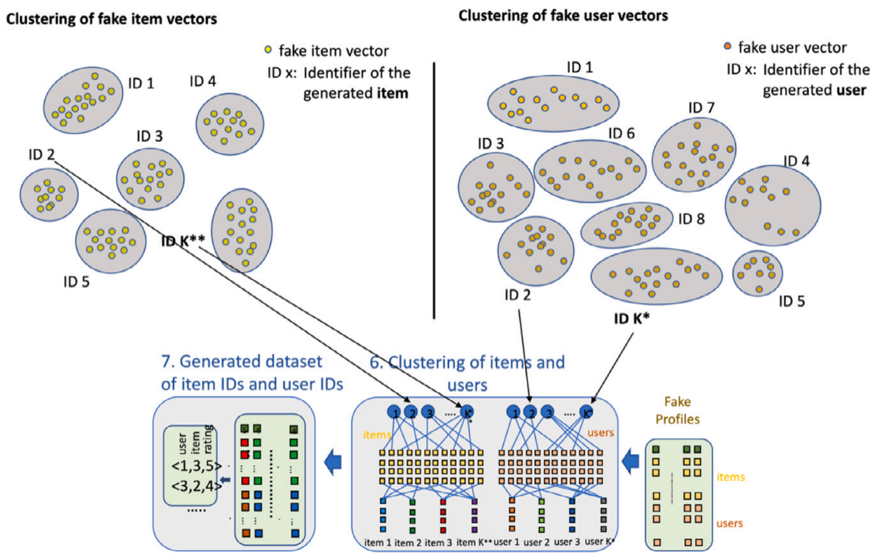
Clustering of fake item vectors
fake item vector
ID x: Identifier of the generated item
ID 1
ID 2
ID 3
ID 4
ID 5
ID K**
Clustering of fake user vectors
fake user vector
ID x: Identifier of the generated user
ID 1
ID 2
ID 3
ID 4
ID 5
ID 6
ID 7
ID 8
ID K*
7. Generated dataset of item IDs and user IDs
6. Clustering of items and users
user item rating
<1,3,5>
<3,2,4>
items
users
item 1 item 2 item 3 item K** user 1 user 2 user 3 user K*
Fake Profiles
items
users

Fig. 3. Top graphs: clustering process to collapse user and item similar vectors into their representative user and item representations. Bottom graphs: translation from unlimited fake limited profiles to profiles in the range $\langle\{1\ldots K^{*}\},\{1..K^{**}\},rating\rangle$.

Fig. 2 shows an example of this operation, formalized in Step 24. Please note that repeated samples will appear in the previous discretization process, since the GAN generator can create very similar dense samples that will be converted to the same discrete encoding. There are several factors that modulate the number of repeated samples, such as the number of generated samples, the embedding size, the size of the noise vector and the standard deviation of the Gaussian distribution, but the most relevant factor is the number of chosen users or items (K^{*} and K^{**}): the lower the K, the higher the number of repeated samples. When the number of users or items is low, the average number of samples grouped in each cluster is high. Step 25 formalizes the process of removing repeated discrete samples. Finally, the GANRS method can generate a small proportion of samples in which different votes are cast from the same user to the same item; e.g.: $\langle 879,56,4\rangle$, $\langle 879,56,5\rangle$. This could be considered as a convenient behavior: code a higher range of votes (4.5 in the example) or express a change in the user's opinion. These cases can be unchanged, changed, or removed. Step 26 formalizes their removal operation.

Overall, it is important to keep in mind that new rating values are initially calculated in the context of Stage 4 of the proposal, where GAN is used for generating the fake samples of pairs $\langle user,item,rating\rangle$, using the user and item embeddings obtained in the previous stages as a base. Afterward, our methodology refines the obtained data to assure consistency (Stages 5–7).

Appendix A (Table 4) shows the main parameter and hyperparameter values used to design both the models DeepMF and GAN involved in the proposed GANRS method.

4. Experiments and results

Evaluating the quality of the generated datasets and comparing them with state-of-art synthetic datasets is not straightforward, since the traditional measures only cover distribution probabilities. This is the case of the Kullback–Leibler (KL) divergence D_{KL} $(P \parallel Q)$, where P and Q are two probability distributions. In our context, we face two main drawbacks to applying the KL divergence or any similar divergence measure: 1) P and Q are not distribution probabilities; they are datasets, and (2) a low D_{KL} value does not mean that Q (the generated dataset) is a good synthetic dataset obtained from P (the source dataset). In fact, if $D_{KL}=0$, usually $P=Q$, which is not a useful result. Of course, each CF dataset contains a reduced number of representative distribution probabilities, including: rating, user, and item distributions (Q_u, Q_i, Q_r); but comparing each distribution of the generated dataset with the corresponding distribution of the source dataset has the same intrinsic problem as explained above: $D_{KL}(P_u \parallel Q_u)=0$, $D_{KL}(P_i \parallel Q_i)=0$, $D_{KL}(P_r \parallel Q_r)=0$, does not mean that Q is a suitable synthetic dataset with regard to P, indeed Q must have a certain degree of variability regard to P. A common alternative approach that is used to deal with these situations is testing the quality results in the specific domain; in our case MAE, precision, recall, etc. Results should be interpreted according to graph trends rather than absolute values, since better results just mean that Q patterns are less complicated than P ones, and worse results tell us that Q patterns are more complicated than P ones. Which scenario is better? It depends on the objectives of the scientist that generates the synthetic datasets. Addressing the concerns explained, we provide a complete set of comparative graphs between the source (P) and generated datasets (Q), including probability distributions of the user, item, rating, and precision and recall trends. Designing specific quality measures that maximize each scientist's objectives (required distribution variability, required complexity in the resulting patterns, etc.) is challenging research, and would help compare state-of-art generative approaches, but this is out of the scope of this paper.

In this paper we evaluate the suitability of the presented procedure focused on building synthetic datasets. First, the traditional data sets to be used as a starting point for the present procedure are presented, as well as a description of the experiments to be performed. Subsequently, the obtained results are presented and discussed.

4.1. Experiments

To test the behavior of the proposed GANRS method, we will use three representative and open datasets in the CF field: Movielens [60], Netflix and MyAnimeList. We have chosen the 100K version of Movielens and a reduced version of the complete Netflix dataset: Netflix*, available in [61]. Table 2 shows the main parameter values for these datasets. A complete set of experiments has been run using Netflix*, whereas only a subset of these experiments is shown for Movielens and MyAnimeList, to reduce the size of the paper. Results from the Movielens and MyAnimeList tests are summarized at the end of this section. Each of the three source datasets is used to generate its corresponding synthetic version: setting different numbers of users, items, and samples, and changing the standard deviation of the Gaussian random noise.

Table 3
Parameter values of the synthetic datasets generated by GAN.
Source: Netflix*.

#	std	#users	#items	#	std	#users	#items	#	std	#users	#items	#samples
1	2.0	100	4000	6	2.5	100	4000	11	3.0	100	4000	1.5M
2	2.0	1000	4000	7	2.5	1000	4000	12	3.0	1000	4000	
3	2.0	2000	4000	8	2.5	2000	4000	13	3.0	2000	4000	
4	2.0	4000	4000	9	2.5	4000	4000	14	3.0	4000	4000	
5	2.0	8000	4000	10	2.5	8000	4000	15	3.0	8000	4000	
16	1.5	4000	2000	17	1.5	4000	8000					
18	1.2	2000	4000									150K
19	1.2	2000	4000									500K
20	1.2	2000	4000									1M
21	1.2	2000	4000									3M

Experiments have been carried out using the neural DeepMF model. Training, validation, and testing sets have been obtained for all the real datasets (Netflix*, MyAnimeList, and Movielens 100K), and their corresponding synthetic datasets. The source code to train the model and test the results is the same for both the real and generated datasets; ensuring the consistency of the graphs in the comparative figures (Figs. 4a, 4b, 7b, 7e, 8b and 7e).

Table 3 shows the GAN generated synthetic datasets used to test the proposed GANRS method, using Netflix* as source data. The '#' columns show the number of the generated datasets; 'std' is the standard deviation used in the random noise Gaussian distribution; #users and #items are the total number of users and items chosen to generate each dataset; #samples is the number of fake samples created by the GAN generator. Please note that the final number of samples contained in each of the datasets is lower than #samples, due to the removing process of repeated samples. Cases 1 to 15 in Table 3 are used to test the effect of changing standard deviation and number of users. Cases 16 and 17 test the consequences of increasing the number of items. Finally, cases 18, 19 and 20 test the behavior of the synthetic datasets when they have different sizes (number of samples). All generated datasets and the source code of the proposed GANRS method are fully available in http://suleiman.ujaen.es:8061/gitlab-instance-981c80cc/ganrs. Additionally, Appendix B (Fig. 9) shows an example of the distribution graphs obtained for each of the synthetic datasets. Following the link provided, each generated dataset is located in its specific directory where a 'readme.txt' file is provided along the synthetic dataset distribution graphs.

Using the parameter values of Table 3, a variety of experiments have been conducted. The classification of the experiments is as follows:

1. Number of users
   (a) Distribution of users versus ratings
   (b) Distribution of the user ratings
   (c) Number of repeated samples
   (d) Proportion of samples with the same user and item
   (e) MAE and accuracy of the data set
   (f) Users' precision and recall
2. Number of items
   (a) MAE and accuracy of the dataset
   (b) Item's precision and recall
3. Number of samples
   (a) Number of samples generated
   (b) Precision and Recall

These experiments refer to well-known metrics in collaborative filtering.

Precision is focused on measuring the proportion of relevant recommendations (i.e. the user rated the item with a rating value equal or greater than a threshold θ) among the top N items recommended to the user u, collected in the list T_u^N (Eq. (4)). On the other hand, Recall measures the proportion of correctly predicted relevant recommendations among the total number of relevant votes of each user; therefore, recall is sensitive to the existing proportions of relevant ratings (Eq. (5)).

$$Precision = \frac{1}{\#U} \sum_{u \in U} \frac{|\{i \in T_u^N | r_{ui} \geq \theta\}|}{N} \tag{4}$$

$$Recall = \frac{1}{\#U} \sum_{u \in U} \frac{|\{i \in T_u^N | r_{ui} \geq \theta\}|}{|\{i \in T_u^N | r_{ui} \geq \theta\}| + |\{i \notin T_u^N | r_{ui} \geq \theta\}|} \tag{5}$$

Where U is the set of training users, r_{ui} is the rating of the training user u for the item i, N is the number of recommendations, and T_u^N is the set of N recommendations for the test user u: N highest predictions of the user u above the relevancy threshold θ.

Please note that Precision measures the proportion of recommendation hits (hits with respect to number of recommendations), whereas Recall measures the proportion of recommendation hits with respect to the total number of relevant items. Precision takes into consideration the number of true positives, whereas Recall combines both the true positives and the false negatives. The importance of the precision and the recall quality measures largely depends on the scenario in which they are applied, e.g. Recall seems to be crucial in medicine, where a false negative is a serious mistake (i.e. not detecting cancer). Nevertheless, Recall is less important in RS since missing a relevant film (false negative) is not serious; the objective is to maximize a correctly recommended film (true positives). The F1 quality measure combines both Precision and Recall (Eq. (6)).

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6}$$

Finally, this paper also tests accuracy (Eq. (7)), where true negatives are also considered. In this case both the positive and the negative hits contribute to the results (to positively recommend relevant items and to negatively recommend non relevant items).

$$Accuracy = \frac{|\{i \in S_t | p_{ui} \geq \theta \wedge r_{ui} \geq \theta\}| + |\{i \in S_t | p_{ui} < \theta \wedge r_{ui} < \theta\}|}{|S_t|} \tag{7}$$

Where S_t is the set of test samples, and each sample is the tuple $\langle u, i, r \rangle$ containing the user ID, item ID and rating of the user u for the item i (r_{ui}). The model prediction of the rating is p_{ui}.

Two values of the threshold θ will be explored across the experimental scenario when precision, recall, F1, and accuracy quality measures are tested

Note that the accuracy quality measure does not use the term T_u^N since the typical RS does not include negative recommendations. Accordingly, this accuracy formulation does not average users' results and acts on the entire training data, such as we have done with the Mean Absolute Error (Eq. (8)).

$$MAE = \frac{1}{|S_t|} \sum_{s \in S_t} |p_{ui} - r_{ui}|, u, i \in S \tag{8}$$

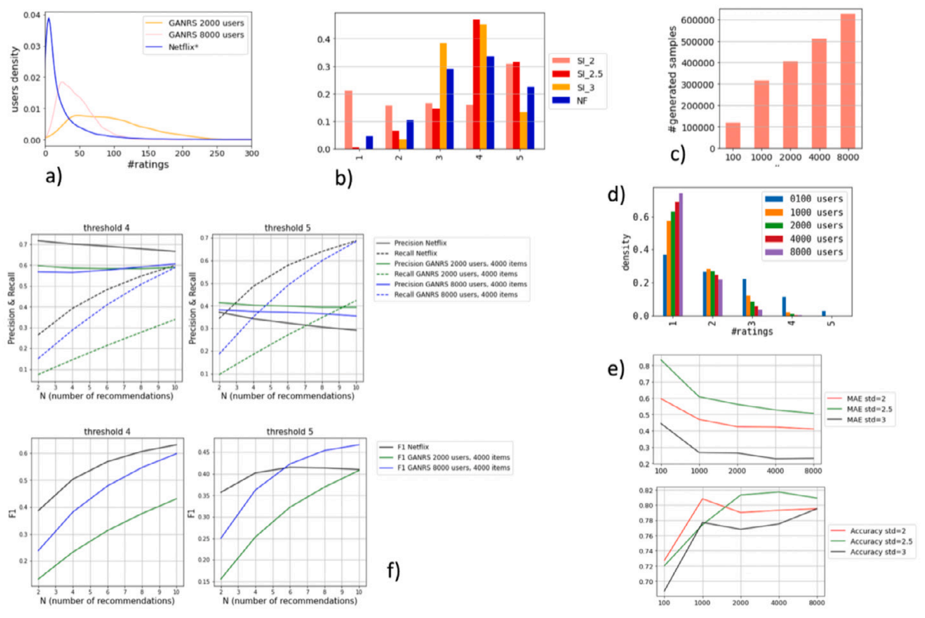

Fig. 4. (a) Distribution of users versus ratings. Number of items: 4000. Datasets 8 and 10 in Table 3;(b) Distribution of user ratings. Number of users: 2000; number of items: 4000. Datasets 3, 8 and 13 in Table 3;(c) Number of samples remaining after removing the repeated ones. items: 4000. Datasets 6 to 10 in Table 3;(d) Proportion of samples in which the same user has cast different votes for the same item. items: 4000. Datasets 6 to 10 in Table 3;(e) MAE and accuracy. Number of users: 2000; number of items: 4000; 'std' is the standard deviation of the Gaussian random noise distribution. Datasets 1 to 15 in Table 3;(f) Precision, recall, and F1. Standard deviation of the random noise Gaussian distribution: 2.5. Number of recommendations $N = [2,4,6,8,10]$. Datasets 6 to 10 in Table 3.

4.2. *Results*

This subsection shows the graphs obtained when the designed experiments (previous subsection) are run. The synthetic datasets described in Table 3 are used to obtain results that allow us: (1) to compare the distributions of users, items and ratings belonging to the source datasets, in relation to those obtained using the synthetic datasets, (2) to measure the number of repeated samples returned in the clustering stage, and (3) to test the prediction and recommendation qualities and trends obtained by running the proposed RSGAN method and comparing them to those shown by the source datasets.

4.2.1. *Experiment 1a. Number of users: Distribution of users versus ratings*

Fig. 4a shows the density of users (y-axis) that have cast different numbers of votes (x-axis). (Selected datasets: 8 and 10 in Table 3). As expected, for a fixed number of ratings in the dataset, we can observe that the higher the number of users, the lower the number of ratings. If the fixed number of samples in the dataset is distributed among a high number of users, each user centroid in the clustering stage receives a lower number of samples. Please, note that Netflix* contains around 23,000 users.

4.2.2. *Experiment 1b. Distribution of the user ratings*

Fig. 4b shows the proportion of each rating 1,...,5 (x-axis) when different random noise Gaussian distributions are applied. (Selected datasets: 3, 8 and 13). It can be observed that the standard deviation 2.5 generates a more similar distribution of votes, compared to the Netflix* original, than the adjacent standard distributions 2 and 2.5. Fig. 4b also shows the impact of the Gaussian standard deviation in the layout of the individual values of the GAN-generated samples.

4.2.3. *Experiment 1c. Number of repeated samples*

As explained in the 'Method' section, the trained GAN generator predicts from random noise vectors as many dense samples as we want; all these samples are then converted from continuous dense values to discrete sparse ones. In the discretization process, repeated samples will appear that must be removed (Table 1 contains an example). Fig. 4c shows the number of samples remaining in the dataset after the removal process. The lower the number of users, the higher the number of samples assigned to each user (to its centroid in the clustering process), and therefore the higher the probability of repeating discrete samples. Overall, the smaller the number of users, the smaller the number of remaining samples. Selected datasets: 6 to 10 in Table 3.

4.2.4. *Experiment 1d. Proportion of samples with the same user and item*

The GANRS generated datasets possess one attribute that does not exist in the source datasets (Movielens, etc.): they contain a proportion of samples where the same user has cast different votes for the same item; e.g.: $\langle 348, 90, 5\rangle$, $\langle 348, 90, 4\rangle$, as explained in the 'Method' section. This can be seen as a mechanism to allow intermediate votes (4.5 in the example) or to allow users to change their minds. This makes sense if, the number of repeated votes is two or three. The rare cases of four or five repeated votes should be removed, just as we have done in all the generated datasets.

From the standard quality metrics to measure the accuracy of predictions: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), we have chosen the former since it is the most widely used in RS state-of-art research. Some papers provide both measures, but experimental research shows that in the CF field, results for RMSE and MAE are very similar. This is because the distribution of the errors in the CF field usually has little variance. The MAE returns the absolute difference between the predicted values and the real values in the testing set: $MAE = \frac{1}{n}\sum_i |y_i - \hat{y}_i|$. The lower the MAE, the better the model fits a dataset. The RMSE uses the square of the error instead of the absolute value: $MAE = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$; therefore, the RMSE is more sensitive to observations that are further from the mean, and this is not the case in CF.

Fig. 4d shows that for regular CF RS (1000 or more users), the proportion of four or five repetitions is not significant, and as the number of users increases, the proportion of repetitions drops very fast.

4.2.5. *Experiment 1e. MAE and accuracy of the dataset*

Whereas the previous experiments analyze the internal composition and distribution of the synthetic datasets, this experiment and the following experiment test the behavior of the generated datasets on the prediction and recommendation tasks. Fig. 4e shows the prediction quality (MAE) and the accuracy of the recommendation obtained from each set of individual samples in Datasets 1 to 15 in Table 3. Please note that these measures are not obtained by analyzing and averaging the results of users. The graphs in Fig. 4e show an improvement in accuracy (and its corresponding decrease in MAE error) as the number of users increases. This behavior is expected in the CF RS, where a high number of users leads to better predictions, and it tells us that the GAN-generated samples follow a CF convenient pattern. The MAE values in the top graph of Fig. 4e are closely related to the distribution of ratings for each of the standard deviations 2.0, 2.5 and 3.0. MAE/accuracy results can be used to select the most appropriate standard deviation; in this case: std = 2.5.

4.2.6. Experiment 1f. Users' precision and recall

This experiment provides the most significant results to test the generated datasets: we extract the values and evolutions of two representative recommendation quality measures: precision and recall. The top graphs in Fig. 4f show the quality values obtained testing several numbers of recommendations N: [2,4,6,8,10] (x-axis), two different relevancy thresholds θ: [4,5], and two number of users: 2000 (green lines), and 8000 (blue lines). The standard deviation of the Gaussian random noise has been set to 2.5. Selected datasets: 6 to 10 in Table 3. The values and evolutions obtained from the synthetic datasets fit with the source dataset: Netflix* (black lines). Additionally, as expected, the overall results of the dataset generated by 8000 users outperform those of the 2000 users and are closer to the Netflix* reference (please note that Netflix* contains around 23,000 users). The two bottom graphs in Fig. 4f represent the F1 combination of precision and recall; they clearly show the similarity in the behavior of the generated datasets compared to the source dataset.

4.2.7. Experiment 2a. MAE and accuracy when the number of items varies.

Experiment 1e tested MAE and accuracy quality measures on datasets with different numbers of users. Now we will test both quality measures on datasets with different numbers of items: [100, 1K, 2K, 4K, 8K]. The results in Fig. 5 show adequate values for both MAE and accuracy, and consistent evolutions where accuracy increases and MAE decreases as the number of items (x-axis) increases. Thus, the higher the number of items, the better the accuracy: this shows that the GAN generator can enrich the data. The Netflix * source dataset contains 1750 items and we can observe in Fig. 5a how the improvement slows down around this value (x-axis). Selected datasets: 16 and 17 in Table 3, and the 100, 1000, 4000 user versions not included in Table 3.

4.2.8. Experiment 2b. Items' precision and recall

Experiment 2b is similar to Experiment 1f; now we will test the behavior of datasets that contain different numbers of items (instead of different numbers of users). Fig. 5b shows the performance of Netflix* (1750 items), represented using black lines, and compares it with the 2000 item dataset (green lines) and the 8000 item dataset (blue lines). We can observe that evolutions and values are consistent with the source datasets (black lines); furthermore, both the 2K and 4K items versions perform well: the first one conveniently captures the Neflix* patterns of items, since both contain a similar number of items. The dataset generated second (8K items) can enrich the data and show better accuracy than the 2K items version. Selected datasets: 16 and 17 in Table 3, and the 100, 1000, 4000 user versions not included in Table 3.

4.2.9. Experiment 3a. Number of samples generated in datasets with different sizes

Here we will test the number of samples that the GANRS method obtains when different numbers of generated samples and different numbers of users have been set. For this purpose, we define four different numbers of samples: 150K, 500K, 1M and 3M (Datasets 18 to 21 in Table 3, and their equivalent datasets for 100, 1000, 4000 and 8000 users) in the GAN generation process. The number of items is fixed at 4K for all experiments. In Fig. 6a we can observe that the smaller the number of users, the smaller the number of generated samples; this is due to the fact that the smaller the number of users, the higher the number of samples assigned to each user (to each centroid in the clustering stage), and therefore the higher the probability of repeated samples that will be removed. As an example, Fig. 6 shows that the 8K user dataset preserves, approximately, 1M samples from the GAN generated (version 3M), and 600K in version 1M.

4.2.10. Experiment 3b. Precision and recall on datasets with different sizes

This experiment shows the impact of increasing the number of samples in datasets with fixed parameters, in this case: 2000 users, 4000 items, and a standard deviation of 1.2 (Table 3; Datasets 18, 19 and 21). It is important to realize that we are using the same source dataset Netflix* to generate the three cases shown in Fig. 6b: 150K samples (yellow lines), 500K samples (magenta lines), and 3M samples (red lines). Please note that 150K, 500K and 3M samples refer to the dense and continuously generated samples, prior to the removal stage to convert them into their sparse, discrete version. Fig. 6a shows the final sizes of the datasets in the 2000 user data (x-axis).

Fig. 6b compares the precision and recall values obtained in the Netflix* dataset (black lines) with the generated values. Overall: (1) precision increases and recall decreases; (2) the bigger the generated dataset, the better its precision; (3) the higher the dataset, the lower its recall. Precision results improve when using large datasets, as there are more relevant samples to choose from, and therefore it is easier to succeed in the fixed number N of recommended predictions. On the other hand, recall gets worse using large datasets because they contain more variability in the samples, particularly when large standard deviations have been chosen for the random noise Gaussian distribution. Unlike precision, whose denominator is the constant N (number of recommendations), the recall quality measure depends on the variable: 'number of relevant votes' in the set of test items for each user tested. As the number of samples increases, the number of user votes also increases (and, from them, the number of relevant votes); this is the reason why recall is lower in the 3M synthetic dataset in Fig. 6b, and higher in the 150K version.

Figs. 7 and 8 show, respectively, the results obtained from the MyAnimeList and Movielens 100K test datasets. Graph '(a)' compares the rating distribution of each source dataset (in blue) with the generated rating distributions obtained by setting different values of the Gaussian random noise standard deviation. We have chosen the standard deviation value of 1.2 for MyAnimeList, and the standard deviation value of 2.5 for Movielens 100K, since the obtained distributions of ratings are closest to their respective baselines. Results '(b)', '(c)' and '(e)' are obtained using the selected standard deviation values. Graph '(b)' shows the distribution of users according to their number of casted ratings (x-axis). As expected, they follow the same pattern as the one in Netflix*. To compare results, please note that MyAnimelist dataset contains 19,179 users, and Movielens 100K contains 943 users. Graph '(c)' shows the number of samples left after removing repeated instances. The higher the number of users, the lower the probability of generating samples containing the same user ID, item ID, and rating. In the MyAnimeList case, we started with 1.5 million generated samples, whereas for Movielens we selected 1 million generated samples. Graph '(d)' refers to MAE error and accuracy values obtained by processing the individual samples contained in each dataset. As usual in the CF context, the higher the number of users, the lower the error, and the higher the accuracy. Finally, Graphs '(e)' tests the recommendations obtained by processing the users in each dataset. As is with Netflix*, compared to baselines, precision improves and recall gets worse.

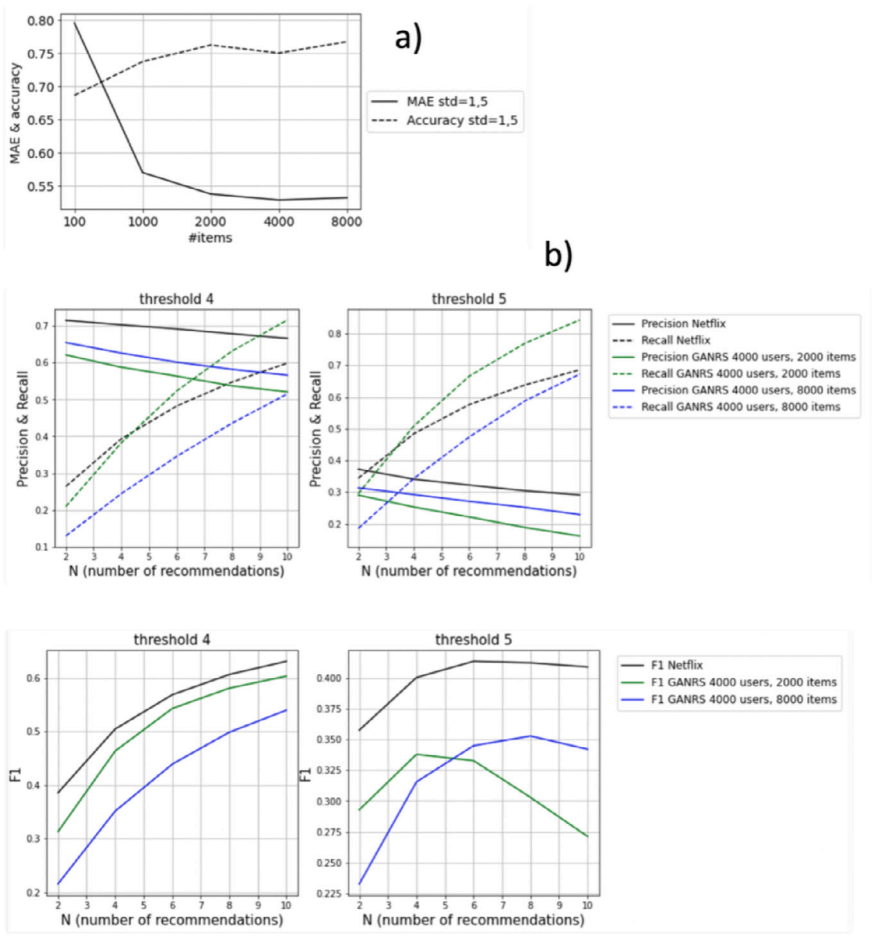

Fig. 5. (a) MAE and accuracy obtained from the dataset samples when the number of items varies. Number of users: 4000. Standard deviation of the Gaussian random noise: 1.5. Datasets 16 and 17 in Table 3;(b) Precision, recall, and F1 when the number of items varies. Standard deviation of the random noise Gaussian distribution: 1.5. Number of recommendations $N = [2, 4, 6, 8, 10]$. Datasets 16 and 17 in Table 3.

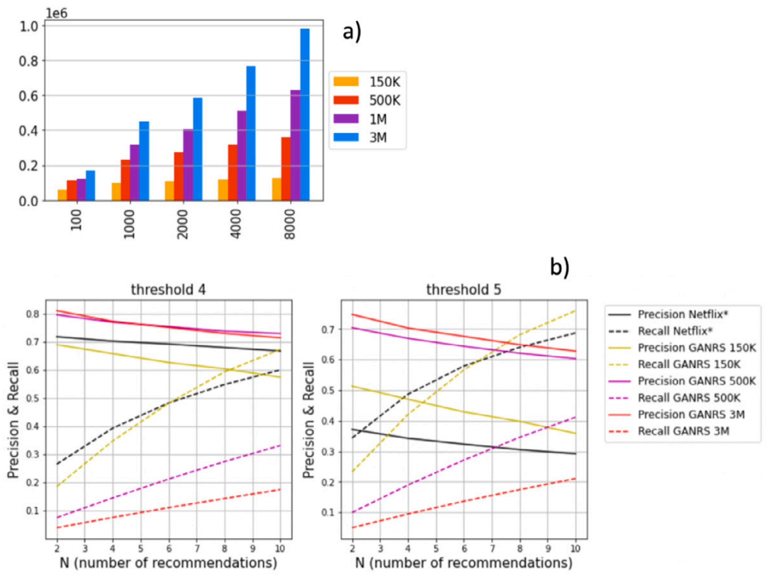

Fig. 6. (a) Number of generated samples using different number of users (x axis) and different number of GAN generated samples (legend). Standard deviation of the random noise Gaussian distribution: 1.2. Number of items: 4000. Datasets 18 to 21 in Table 3;(b) Precision and recall using a different number of recommendations (x axis) and a different number of GAN generated samples (legend). Standard deviation of the random noise Gaussian distribution: 1.2. Datasets 18, 19 and 21 in Table 3.

The results obtained in this section highlight the importance of those that test the performance of the synthetic datasets against the source datasets, particularly when specific RS metrics are used. To check the consistency between synthetic and real data, two types of

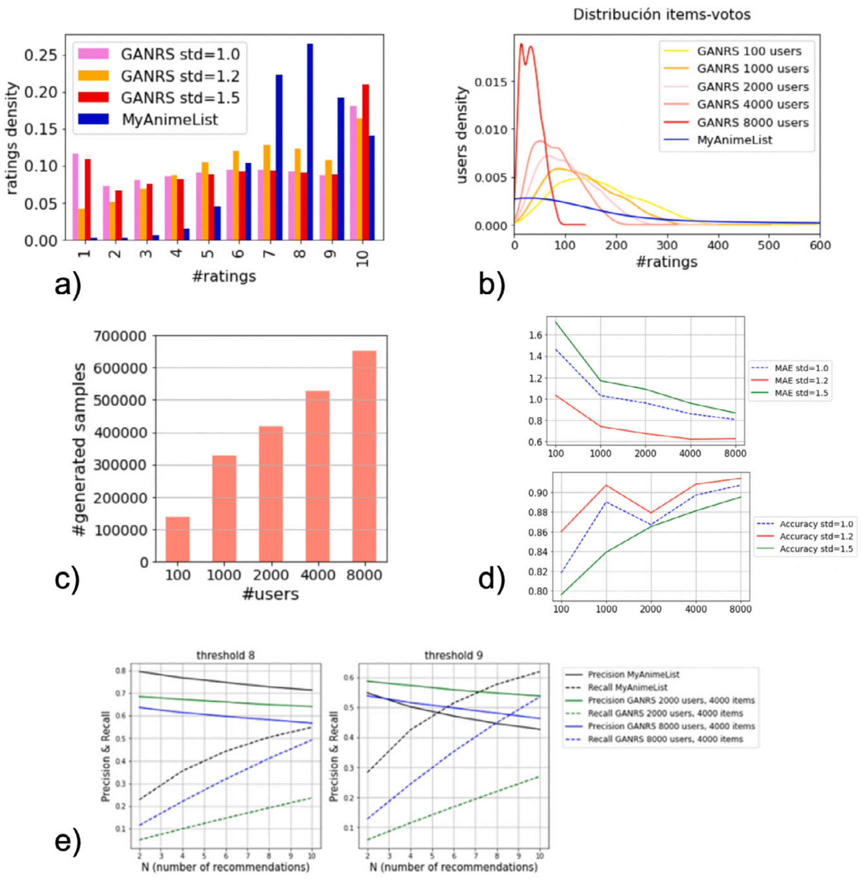

Fig. 7. MyAnimeList. 1.5 million generated samples, (a) distribution of the MyAnimeList ratings 1 to 10, (b) distribution of users according to their number of casted ratings, (c) number of samples after the removing process of the repeated ones, (d) error and accuracy by processing the samples of the dataset, (e) CF precision and recall (by testing the dataset users). The GANRS std = 1.2 value has been set to test experiments (b) to (e).

experiments have been conducted: direct and indirect. In direct comparisons, rating distributions have been obtained and compared from both source datasets and their synthetic versions. Figs. 4b and 4c show the Netflix* results by varying the number of generated users and the standard deviation of the random Gaussian distribution used to feed the proposed GAN. Figs. 7b and 8b show, respectively, comparison of the MyAnimeList and the Movielens 100K datasets, in this case by varying the number of users in the synthetic datasets versus their equivalent source counterparts. Indirect experiments tested and compared the recommendation performance on both the synthetic and the source datasets. We have chosen the recommendation quality measures of precision, recall and F1, obtained using the classical neural model DeepMF. Results can be found in Fig. 4f (Netflix* vs. its synthetic version), Fig. 7e (MyAnimeList vs. its synthetic version), and Fig. 8e (Movielens 100K vs. its synthetic version). Overall, as expected, the results show that synthetic datasets behave like their source datasets. The more similar the results are, the more suitable the generated datasets will be, as this means that the original datasets can be effectively replaced by synthetic ones.

4.3. Comparison of the proposed framework with previous work

The Related Works section identifies some previous work focused on data generation methods for recommender systems. In this subsection, a brief analysis will be performed, which will focus on showing how this previous work is not truly comparable with our current proposal in a fair way, since it is focused on different objectives and also generates data of a different nature.

- Mladenov et al. [53] presented RecSim NG, an architecture centered on the generation of synthetic profiles of users and items as part of the recommendation environment. Overall, the goal of the work is the development of a configurable platform for both authoring and learning RS simulation environments. The aim of this simulation is to evaluate existing RS policies, or generate data to train new policies (in either a tightly coupled online fashion, or in batch mode). Furthermore, this paper lacks information about the presented method, and therefore does not allow reproducibility.
- Shi et al. [54] introduce a multi-agent reinforcement learning architecture tailored to Taobao-specific website search improvement, and uses a GAN to simulate the internal rating distribution. Therefore, considering that it is focused on data generation for a specific context, it is not comparable with the framework proposed in the current paper.
- Del Carmen et al. [56] introduce DataGenCars, a Java-based generator of RS synthetic data. Here it is important to remark that this work is specifically focused on the context-aware recommendation scenario. In this sense, even though the proposed tool supports the generation of synthetic datasets of users, items, contexts, and ratings; this generation always relies on context-related characteristics through criteria introduced throughout the work, such as the uncertainty of the content, the user's expectations or the item's attributes. As result, this work is not comparable with the methodology presented in our current paper, which mainly uses rating values as input and does not consider datasets with contextual information.

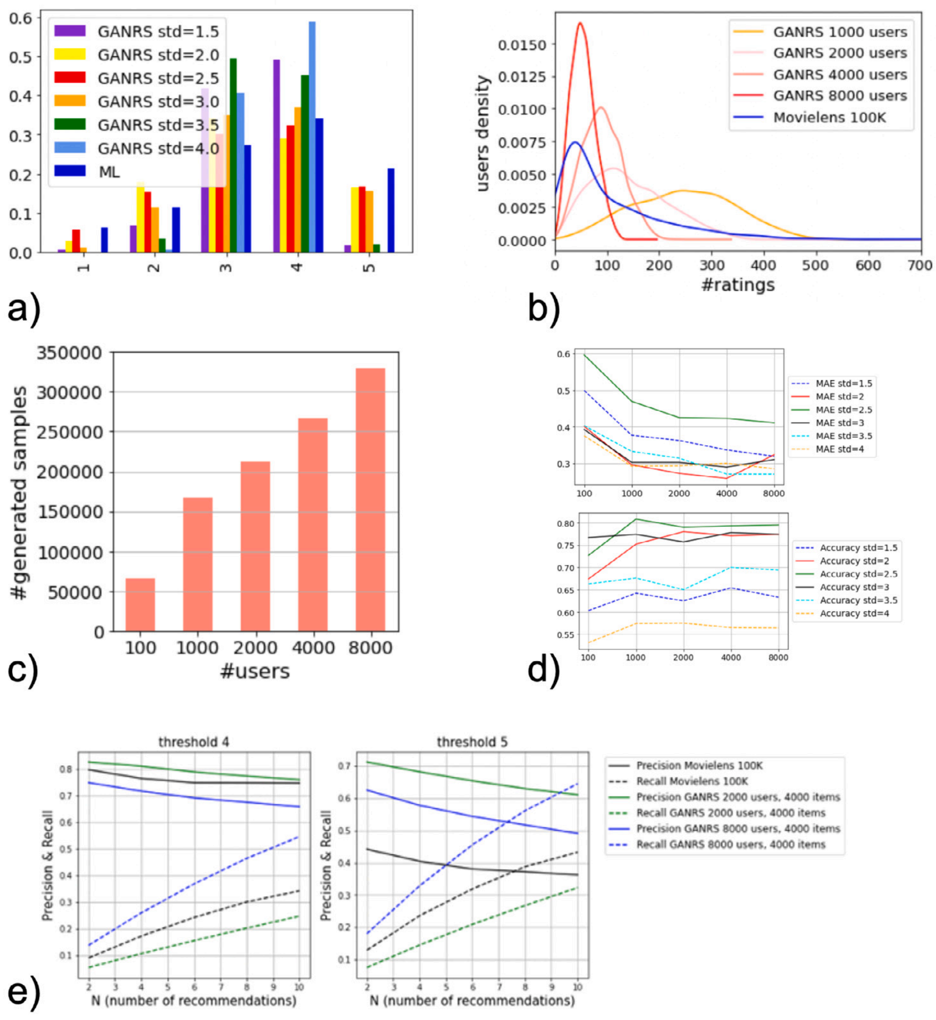

Fig. 8. Movielens 100K results. 1 million generated samples, (a) Distribution of the Movielens 100K ratings 1 to 5, (b) Distribution of users according to their number of casted ratings, (c) Number of samples after the removal process of the repeated ones, (d) error and accuracy by processing the samples of the dataset, (e) CF precision and recall (by testing the dataset users). The GANRS std = 2.5 value has been set to test experiments (b) to (e).

- Provalov et al. [57] introduce the SynEvaRec framework, focused on the presentation of a novel paradigm for evaluating recommendations based on the generation of synthetic RS datasets. In contrast to our current paper, this approach is mainly focused on generating synthetic user and item profiles that are internally used by SynEvaRec to guarantee user privacy protection, mitigate the data insufficiency problem, and measure the effect of the no-free-lunch problem. Regarding the aim of the architecture proposed in Provalov et al. [57] is not the proper retrieval of the whole synthetic rating datasets to be used in further evaluations (i.e. an evaluation protocol is presented rather than a dataset generation method), a major transformation of their work is needed to make it comparable with this paper. A fair comparison is then not possible at this stage.

4.4. Overall discussion

A large number of synthetic datasets have been generated to test the performance of the proposed GANRS method. These datasets have been created setting different values for the main parameters of the method: number of users, number of items, Gaussian random noise variation, and number of generated samples. To generalize the conclusions of this paper, three open and representative CF datasets have been used as sources for the generative process. Finally, a variety of quality measures have been tested on the generated datasets; of these, precision and recall are the most relevant. A key issue is that we are not able to visually test the quality of the generated samples, as can be done, for example, with the popular fake faces; in fact, in the CF context, we only can adequately test the generated datasets by comparing their CF quality results with those typically obtained in real CF datasets. For this reason, we have focused on the designed experiments in which the quality measures of precision and recall are tested: using datasets containing different numbers of users, different numbers of items, and different numbers of samples (sizes). In all cases, comparatively, we obtain excellent precision results and moderate recall values. Overall, it can be considered positive in the CF context, where precision errors are serious and recall errors are less important: it is worse to recommend a trip you will not like (sorry, no refunds!) than not to recommend a trip that you probably would enjoy. Please note that it is the opposite for a deep learning model detecting malignant tumors: it is worse to make precision errors (no early detection of the tumor) than making recall errors (to erroneously detect a tumor).

Additionally, experiments show the relevant impact of the standard deviation on the quality of the results. The GAN network learning has been based on a vector containing noise values that serves as a seed to generate the different samples in the synthetic dataset. Each 'fake' sample is generated from the list of random values in the 'noise' vector. As usual in the GAN context, random values have been created from a Gaussian distribution with a mean of 0 and a standard deviation of 1. Each generated sample contains a dense item representation, a dense user representation, and an individual value that codes the user's rating of the item. Once the GAN has learned, its generative model can be used, in a feedforward process, to generate as many samples as we want, starting from a different random noise vector for each sample generated. Experimental results show that using a Gaussian distribution with a standard deviation of 1 leads to many ratings in the middle of the voting range: rating 3 and closest in the 1 to 5 voting range (Movielens and Netflix), and rating 5 and closests in 1 to 10 voting range (MyAnimeList). Several experiments in this section demonstrate that we can modulate the standard deviation of the Gaussian distribution of random noise to generate a wider range of ratings using feedforward. As expected, when the standard deviation increases, the range of ratings also increases proportionally.

Finally, the experiments presented include the existing relationship between the number of fake samples generated for the GAN and the number of samples that the dataset will eventually contain. As explained in the 'Method' section, the conversion from dense and continuous values to sparse and discrete values leads to a probability of sample repetitions. Results show that, as expected, the larger the number of users and items in the synthetic dataset, the lower the number of repeated samples. It has also been shown that for a typical number of users, say 4000 or more, the probability of more than two different ratings from one user for the same item can be considered negligible.

5. Conclusions

This paper provides an innovative method for generating synthetic parameterized collaborative filtering datasets from real datasets. Synthetic datasets can be generated by selecting different numbers of users, items, samples, and distribution variability. This means that comparative experiments can be designed on the basis of a whole 'family' of generated datasets, for example, to test the accuracy of a new matrix factorization model when the number of users increases. A GAN is used to obtain 'fake' samples from real samples, benefitting from the inherent capacity of GAN networks to capture complex patterns in the source datasets. The GAN learns from dense and continuous embedding representations of items and users, rather than the sparse and discrete representations of the collaborative filtering datasets. The effect is a fast and accurate learning process.

The proposed GANRS method contains a clustering stage to convert from the dense generated 'fake' samples to the sparse and discrete values necessary to fill the generated dataset. This clustering stage implements a k-means algorithm to group items and another k-means to group users. In a natural way, both 'k' parameters set the chosen number of users and items in the dataset. A drawback of the discretization process is the generation of identical samples that our method merely removes. A complete set of experiments have been made using three representative source datasets. We have tested the distribution values and evolutions of the results, as well as prediction and recommendation qualities. Although precision tends to improve, while recall tends to worsen, overall accuracy can be considered correct, since precision is more relevant than recall in the RS context. The results show that the generated datasets conveniently mimic the behavior of the source datasets Movielens, MyAnimeList, etc.

The source code for the proposed GANRS method is available to ensure the reproducibility of the experiments. Similarly, a complete set of generated datasets has been made available for research. This paper and its related documentation open the door to address some future work, such as designing alternative options to the clustering stage, implementing the PacGAN concept in the GAN discriminator, testing generated datasets using a complete range of machine learning and deep learning collaborative filtering models, replacing the GAN model with a CGAN one, generating demographically balanced datasets, and performing an in-depth study of the impact of the random noise vector variations in the generated set of samples.

CRediT authorship contribution statement

Jesús Bobadilla: Conceptualization, Validation, Formal analysis, Investigation, Software, Writing – original draft, Writing – review & editing, Visualization. **Abraham Gutiérrez:** Conceptualization, Validation, Formal analysis, Investigation, Software, Writing – original draft, Visualization. **Raciel Yera:** Methodology, Validation, Formal analysis, Writing – original draft , Visualization. **Luis Martínez:** Methodology, Validation, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to the data and code has been shared in the manuscript.

Acknowledgments

This work was partially supported by Ministerio de Ciencia e Innovación of Spain under the project PID2019-106493RB-I00 (DL-CEMG); the Comunidad de Madrid, Spain under Convenio Plurianual with the Universidad Politécnica de Madrid, Spain in the actuation line of Programa de Excelencia para el Profesorado Universitario; and the Plan Andaluz de Investigación, Desarrollo e Innovación (PAIDI 2020), Spain under the project PROYEXCEL_00257.

Appendix A

See Table 4.

Table 4
Main parameter and hyperparameter values set for the neural models involved in the RSGAN method.

DeepMF	**values**
Embedding size (both for users an items)	5
Optimizer	Adam
Loss function	Mean squared error
Epochs	20
GAN generator	
Input shape, noise vector size	100
Block 1 dense layer #neurons	10
Block 1 activation function	LeakyRelu, alpha 0.2
Block 1 normalization	BatchNormalization, momentum 0.8
Block 2 dense layer #neurons	20
Block 2 activation function	LeakyRelu, alpha 0.2
Block 2 regularization	Dropout 0.2
Block 3 dense layer #neurons	$2 * \ embedding\ size\ +\ 1$
Block 3 activation function	linear
GAN discriminator	
Input: shape	$2 * \ embedding\ size\ +\ 1$
Block 1 dense layer #neurons	6
Block 1 activation function	LeakyRelu, alpha 0.2
Block 2 dense layer	1
Block 2 activation function	Sigmoid
GAN train	
Epochs	20
Batch size	64
Stochastic noise	Gaussian (0,1)
Loss function	$(real\ samples\ loss\ +\ fake\ samples\ loss)/2$

Appendix B

See Fig. 9.

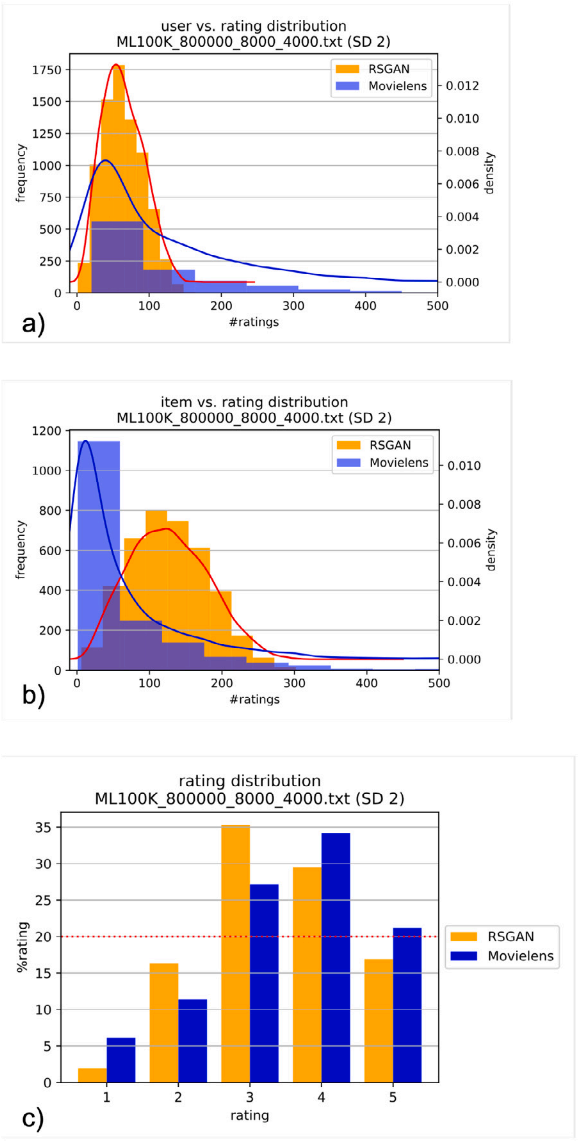

Fig. 9. Main distributions of the data in the synthetic dataset generated from Movielens 100K compared to the distributions of the data in the source dataset. Number of users: 8000, number of items: 4000, initial number of samples: 800,000, standard deviation of the Gaussian noise: 2.5. Graph (a) shows the distribution of the fake users (y axis) versus the number of ratings belonging to each of the users (x axis). Graph (b) shows the distribution of the fake items (y axis) versus the number of ratings belonging to each of the items (x axis). Graph (c) shows the percentage of ratings (y axis) for each of the available vote values 1, 2, 3, 4, 5 (x axis) in the dataset.

References

[1] Z. Fang, L. Zhang, K. Chen, A behavior mining based hybrid recommender system, in: 2016 IEEE International Conference on Big Data Analysis, ICBDA, IEEE, 2016, pp. 1–5.

[2] R. Yera, L. Martínez, Fuzzy tools in recommender systems: A survey, Int. J. Comput. Intell. Syst. 10 (1) (2017) 776–803.

[3] R. Yera, A.A. Alzahrani, L. Martínez, A fuzzy content-based group recommender system with dynamic selection of the aggregation functions, Internat. J. Approx. Reason. 150 (2022) 273–296.

[4] L. Zheng, V. Noroozi, P.S. Yu, Joint deep modeling of users and items using reviews for recommendation, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, 2017, pp. 425–434.

[5] Y. Gong, Q. Zhang, Hashtag recommendation using attention-based convolutional neural network., in: IJCAI, 2016, pp. 2782–2788.

[6] H. Kanwal, M. Assam, A. Jabbar, S. Khan, et al., Convolutional neural network and topic modeling based hybrid recommender system, Int. J. Adv. Comput. Sci. Appl. 11 (7) (2020).

[7] K. McNally, M.P. O'Mahony, B. Smyth, A comparative study of collaboration-based reputation models for social recommender systems, User Model. User-Adapt. Interact. 24 (3) (2014) 219–260.

[8] N.M. Villegas, C. Sánchez, J. Díaz-Cely, G. Tamura, Characterizing context-aware recommender systems: A systematic literature review, Knowl.-Based Syst. 140 (2018) 173–200.

[9] M. Moradi, J. Hamidzadeh, Ensemble-based top-k recommender system considering incomplete data, J. AI Data Min. 7 (3) (2019) 393–402.

[10] M. Jalili, S. Ahmadian, M. Izadi, P. Moradi, M. Salehi, Evaluating collaborative filtering recommender algorithms: a survey, IEEE Access 6 (2018) 74003–74024.

[11] B. Zhu, R. Hurtado, J. Bobadilla, F. Ortega, An efficient recommender system method based on the numerical relevances and the non-numerical structures of the ratings, IEEE Access 6 (2018) 49935–49954.

[12] R. Yera, A.A. Alzahrani, L. Martínez, Exploring post-hoc agnostic models for explainable cooking recipe recommendations, Knowl.-Based Syst. 251 (2022) 109216.

[13] E. D'Amico, G. Gabbolini, C. Bernardis, P. Cremonesi, Analyzing and improving stability of matrix factorization for recommender systems, J. Intell. Inf. Syst. 58 (2) (2022) 255–285.

[14] M.H. Aghdam, A novel constrained non-negative matrix factorization method based on users and items pairwise relationship for recommender systems, Expert Syst. Appl. 195 (2022) 116593.

[15] G. Ayci, A. Köksal, M.M. Mutlu, B. Suyunu, A.T. Cemgil, Active learning with Bayesian nonnegative matrix factorization for recommender systems, in: 2019 27th Signal Processing and Communications Applications Conference, SIU, IEEE, 2019, pp. 1–4.

[16] J. Bobadilla, R. Bojorque, A.H. Esteban, R. Hurtado, Recommender systems clustering using Bayesian non negative matrix factorization, IEEE Access 6 (2017) 3549–3564.

[17] H.-J. Xue, X. Dai, J. Zhang, S. Huang, J. Chen, Deep matrix factorization models for recommender systems, in: IJCAI, Vol. 17, Melbourne, Australia, 2017, pp. 3203–3209.

[18] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 173–182.

[19] J. Bobadilla, R. Lara-Cabrera, A. Gonzalez-Prieto, F. Ortega, DeepFair: Deep learning for improving fairness in recommender systems., Int. J. Interact. Multimed. Artif. Intell. 6 (6) (2021) 86–95.

[20] Y. Himeur, A. Alsalemi, A. Al-Kababji, F. Bensaali, A. Amira, C. Sardianos, G. Dimitrakopoulos, I. Varlamis, A survey of recommender systems for energy efficiency in buildings: Principles, challenges and prospects, Inf. Fusion 72 (2021) 1–21.

[21] J. Bobadilla, J. Dueñas, A. Gutiérrez, F. Ortega, Deep variational embedding representation on neural collaborative filtering recommender systems, Appl. Sci. 12 (9) (2022) 4168.

[22] J. Bobadilla, Á. González-Prieto, F. Ortega, R. Lara-Cabrera, Deep learning approach to obtain collaborative filtering neighborhoods, Neural Comput. Appl. 34 (4) (2022) 2939–2951.

[23] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Comput. Surv. 52 (1) (2019) 1–38.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Commun. ACM 63 (11) (2020) 139–144.

[25] D. Sacharidis, Diversity and novelty in social-based collaborative filtering, in: Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization, 2019, pp. 139–143.

[26] A. Gogna, A. Majumdar, DiABlO: Optimization based design for improving diversity in recommender system, Inform. Sci. 378 (2017) 59–74.

[27] J. Bobadilla, A. Gutierrez, S. Alonso, Á. González-Prieto, Neural collaborative filtering classification model to obtain prediction reliabilities, Int. J. Interact. Multimed. Artif. Intell. 7 (4) (2022) 18–26.

[28] F. Pajuelo-Holguera, J.A. Gómez-Pulido, F. Ortega, Evaluating strategies for selecting test datasets in recommender systems, in: International Conference on Hybrid Artificial Intelligence Systems, Springer, 2019, pp. 243–253.
[29] K.D. Bollacker, S. Lawrence, C.L. Giles, CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications, in: Proceedings of the Second International Conference on Autonomous Agents, 1998, pp. 116–123.
[30] W. Choochaiwattana, Usage of tagging for research paper recommendation, in: 2010 3rd International Conference on Advanced Computer Theory and Engineering, Vol. 2, ICACTE, IEEE, 2010, pp. V2–439.
[31] J. Shokeen, C. Rana, Social recommender systems: techniques, domains, metrics, datasets and future scope, J. Intell. Inf. Syst. 54 (3) (2020) 633–667.
[32] Y. Xing, I. Mohallick, J.A. Gulla, Ö. Özgöbek, L. Zhang, An educational news dataset for recommender systems, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2020, pp. 562–570.
[33] F. Ortega, J. Bobadilla, A. Gutiérrez, R. Hurtado, X. Li, Artificial intelligence scientific documentation dataset for recommender systems, IEEE Access 6 (2018) 48543–48555.
[34] D. Liang, R.G. Krishnan, M.D. Hoffman, T. Jebara, Variational autoencoders for collaborative filtering, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 689–698.
[35] S. Zamany, D. Li, H. Fei, P. Li, Towards deeper understanding of variational auto-encoders for binary collaborative filtering, in: Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval, 2022, pp. 254–263.
[36] M. Gao, J. Zhang, J. Yu, J. Li, J. Wen, Q. Xiong, Recommender systems based on generative adversarial networks: A problem-driven perspective, Inform. Sci. 546 (2021) 1166–1185.
[37] Y. Deldjoo, T.D. Noia, F.A. Merra, A survey on adversarial recommender systems: from attack/defense strategies to generative adversarial networks, ACM Comput. Surv. 54 (2) (2021) 1–38.
[38] D.-K. Chae, J.-S. Kang, S.-W. Kim, J.-T. Lee, Cfgan: A generic collaborative filtering framework based on generative adversarial networks, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 2018, pp. 137–146.
[39] Z. Wang, M. Gao, X. Wang, J. Yu, J. Wen, Q. Xiong, A minimax game for generative and discriminative sample models for recommendation, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2019, pp. 420–431.
[40] W. Zhao, B. Wang, J. Ye, Y. Gao, M. Yang, X. Chen, Plastic: Prioritize long and short-term information in top-n recommendation using adversarial training, in: Ijcai, 2018, pp. 3676–3682.
[41] H. Bharadhwaj, H. Park, B.Y. Lim, Recgan: recurrent generative adversarial networks for recommendation systems, in: Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 372–376.
[42] G. Guo, H. Zhou, B. Chen, Z. Liu, X. Xu, X. Chen, Z. Dong, X. He, IPGAN: Generating informative item pairs by adversarial sampling, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[43] J. Zhao, H. Li, L. Qu, Q. Zhang, Q. Sun, H. Huo, M. Gong, DCFGAN: An adversarial deep reinforcement learning framework with improved negative sampling for session-based recommender systems, Inform. Sci. 596 (2022) 222–235.
[44] J. Sun, B. Liu, H. Ren, W. Huang, NCGAN:: A neural adversarial collaborative filtering for recommender system, J. Intell. Fuzzy Systems 42 (4) (2022) 2915–2923.
[45] Y. Lin, Z. Xie, B. Xu, K. Xu, H. Lin, Info-flow enhanced GANs for recommender, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2021, pp. 1703–1707.
[46] Q. Wang, Q. Huang, K. Ma, X. Zhang, A recommender system based on model regularization wasserstein generative adversarial network, in: 2021 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2021, pp. 2043–2048.
[47] J. Wen, X.-R. Zhu, C.-D. Wang, Z. Tian, A framework for personalized recommendation with conditional generative adversarial networks, Knowl. Inf. Syst. 64 (10) (2022) 2637–2660.
[48] G. Deng, C. Han, D.S. Matteson, Extended missing data imputation via GANs for ranking applications, Data Min. Knowl. Discov. (2022) 1–23.
[49] H. Chen, S. Wang, N. Jiang, Z. Li, N. Yan, L. Shi, Trust-aware generative adversarial network with recurrent neural network for recommender systems, Int. J. Intell. Syst. 36 (2) (2021) 778–795.
[50] G. Van Houdt, C. Mosquera, G. Nápoles, A review on the long short-term memory model, Artif. Intell. Rev. 53 (2020) 5929–5955.
[51] W. Shafqat, Y.-C. Byun, A hybrid GAN-based approach to solve imbalanced data problem in recommendation systems, IEEE Access 10 (2022) 11036–11047.
[52] Z. Lin, A. Khetan, G. Fanti, S. Oh, Pacgan: the power of two samples in generative adversarial networks, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 1505–1514.
[53] M. Mladenov, C.-w. Hsu, V. Jain, E. Ie, C. Colby, N. Mayoraz, H. Pham, D. Tran, I. Vendrov, C. Boutilier, Demonstrating principled uncertainty modeling for recommender ecosystems with RecSim NG, in: Fourteenth ACM Conference on Recommender Systems, 2020, pp. 591–593.
[54] J.-C. Shi, Y. Yu, Q. Da, S.-Y. Chen, A.-X. Zeng, Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33–01, 2019, pp. 4902–4909.
[55] C.-N. Ziegler, S.M. McNee, J.A. Konstan, G. Lausen, Improving recommendation lists through topic diversification, in: Proceedings of the 14th International Conference on World Wide Web, 2005, pp. 22–32.
[56] M. del Carmen Rodríguez-Hernández, S. Ilarri, R. Hermoso, R. Trillo-Lado, DataGenCARS: A generator of synthetic data for the evaluation of context-aware recommendation systems, Pervasive Mob. Comput. 38 (2017) 516–541.
[57] V. Provalov, E. Stavinova, P. Chunaev, SynEvaRec: A framework for evaluating recommender systems on synthetic data classes, in: 2021 International Conference on Data Mining Workshops, ICDMW, IEEE, 2021, pp. 55–64.
[58] A. Cossu, A. Carta, V. Lomonaco, D. Bacciu, Continual learning for recurrent neural networks: an empirical evaluation, Neural Netw. 143 (2021) 607–627.
[59] M. Ahmed, R. Seraj, S.M.S. Islam, The k-means algorithm: A comprehensive survey and performance evaluation, Electronics 9 (8) (2020) 1295.
[60] F.M. Harper, J.A. Konstan, The movielens datasets: History and context, ACM Trans. Interact. Intell. Syst. (TIIS) 5 (4) (2015) 1–19.
[61] F. Ortega, B. Zhu, J. Bobadilla, A. Hernando, CF4j: Collaborative filtering for java, Knowl.-Based Syst. 152 (2018) 94–99.