УДК 004.855.5
DOI: 10.25559/SITITO.16.202002.398-406

Automatic Evaluation of Recommendation Models

O. A. Alieva[a], E. S. Gangan[b], E. A. Ilyushin[a*], A. I. Kachalin[c]
[a] Lomonosov Moscow State University, Moscow, Russia
1 Leninskie gory, Moscow 119991, Russia
[*] john.ilyushin@gmail.com
[b] Babes-Bolyai University, Cluj-Napoca, Romania
1 Mihail Kogălniceanu St., Cluj-Napoca 400084, Romania
[c] PJSC "Sberbank of Russia", Moscow, Russia
19 Vavilova St., Moscow 117997, Russia

Abstract

The paper presents an overview of state-of-the-art algorithms used in recommender systems. We discuss the goal of collaborative filtering (CF) as well as different approaches to the method. Specifically, we talk about Singular Value Decomposition (including optimizations, bias, time sensitive Singular Value Decomposition (SVD) and enhanced SVD methods as SVD++), clustering approaches (using K means clustering). We also discuss deep learning methods applied to recommender systems, such as Autoencoders and Restricted Boltzmann Machines. We also go through qualitative evaluation metrics of the algorithms, with a special emphasis on the classification quality metrics, as recommender systems are usually expected to have an order in which the recommendations are delivered. At the same time, we propose a tool that automates the processes of CF algorithms launch and evaluation, that contains data pre-processing, metrics selection, training launch, quality indicators checks and analyses of the resulted data. Our tool demonstrates the impact that parameter selection has on the quality of the algorithm execution. We observed that classical matrix factorization algorithms can compete with new deep learning methods, giving the correct tuning. Also, we demonstrate a significant gain in time between the manual (involving a person that launches all the algorithms individually) and the automatic (when the tool launches all the algorithms) algorithm launch.

Keywords: Automatization, Collaborative Filtering, Recommender Systems, Recommender Tool.

For citation: Alieva O.A., Gangan E.S., Ilyushin E.A., Kachalin A.I. Automatic Evaluation of Recommendation Models. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2020;16(2):398-406. DOI: https://doi.org/10.25559/SITITO.16.202002.398-406

Автоматическая оценка моделей рекомендаций

О. Алиева[1], Е. Ганган[2], Е. Ильюшин[1*], А. Качалин[3]
[1] ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова», г. Москва, Россия
119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1
[*] john.ilyushin@gmail.com
[2] Университет Бабеш-Боляи, г. Клуж-Напока, Румыния
400084, Румыния, г. Клуж-Напока, ул. Михаила Когэлничану, д. 1
[3] ПАО «Сбербанк России», г. Москва, Россия
117997, Россия, г. Москва, ул. Вавилова, д. 19

Аннотация

В статье представлен обзор современных алгоритмов, используемых в рекомендательных системах. Мы обсуждаем цель коллаборативной фильтрации (CF), а также различные подходы к этому методу. В частности, мы говорим о сингулярном разложении (включая оптимизацию, смещение, чувствительное ко времени сингулярное разложение (SVD) и расширенные методы SVD как SVD++), подходах к кластеризации (с использованием метода K-средних). Мы также обсуждаем методы глубокого обучения, применяемые к рекомендательным системам, такие как Автоэнкодеры и ограниченные машины Больцмана. Мы также рассматриваем качественные метрики оценки алгоритмов, уделяя особое внимание метрикам качества классификации, поскольку рекомендательные системы обычно должны иметь порядок, в котором выполняются рекомендации. В то же время мы предлагаем инструмент, автоматизирующий процессы запуска и оценки алгоритмов коллаборативной фильтрации, содержащий предварительную обработку данных, выбор метрик, запуск обучения, проверку показателей качества и анализ полученных данных. Наш инструмент демонстрирует влияние выбора параметров на качество выполнения алгоритма. Мы наблюдали, что классические алгоритмы матричного факторизации могут конкурировать с новыми методами глубокого обучения, давая правильную настройку. Кроме того, мы демонстрируем значительный выигрыш во времени между ручным (с участием человека, который запускает все алгоритмы индивидуально) и автоматическим (когда инструмент запускает все алгоритмы) запуском алгоритма.

Ключевые слова: автоматизация, коллаборативная фильтрация, рекомендательные системы, рекомендательный инструмент.

Для цитирования: Алиева, О. А. Автоматическая оценка моделей рекомендаций / О. А. Алиева, Е. С. Ганган, Е. А. Ильюшин, А. И. Качалин. – DOI 10.25559/SITITO.16.202002.398-406 // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16, № 2. – С. 398-406.

Introduction

Recommender systems represent a specific type of software that are designed to predict user behavior and suggest items that are of most interest for a user at a given time (items can be movies, articles, music, etc.). As a result, user satisfaction grows and so does the service's income. The income does not have to be direct revenue; the service can capitalize on gaining user loyalty and retention, that in a long run might turn into revenue from ads, subscriptions, or others [1]. With giants like Netflix, YouTube, Amazon and many others, recommender systems (RecSys) gained influence in the industry and became an important part of service-user interaction. Depending on the business model, RecSys may be the core of the service (think TripAdvisor or Booking) or can be a complimentary service that eases navigation and decision making. There are 2 general approaches in RecSys: *content based filtering* and *collaborative filtering*.

Content based filtering uses previous knowledge and items features to recommend users similar items to what they have already liked. The recommended content is based on user interest and their explicit feedback [2]. Product description of all available items is required. Usually, content-based filtering is applied to movies, books and articles. It is best suited for cold-start problems as it does not require big amounts of data to work. However, the method is highly influenced by user's (or item's) features, so the diversity of recommendations may suffer [3].

Collaborative filtering looks at the user's interaction history with different items (purchases, ratings), but also observes similar decisions in other users. Thus, the recommendation is constructed based on the user-item interaction as well as the interactions of users with similar characteristics with the same item. By using another user's information to decide, CF employs group knowledge to construct the recommendation based on user similarity [3, 4]. In such context, recommendations are based on filtering users with related behavioral patterns. One of the shortcomings of CF is the cold-start problem, because the method requires vast prior knowledge of user's habits or user-item interaction. On the other hand, it is not bound by user (item) features, making the recommendation span wider.

Problem Formulation

A. Scope of the paper

The scope of this work is to review all the major CF algorithms and their evaluation metrics and to propose a tool that automates the processes of CF algorithms training and evaluation.

B. Related work

The closest tool to our approach is Cornac, written in Python by Preferred.Ai[1]. It was designed to work with models that use as input data: image, text, social networks and enables fast experimentation. Cornac is compatible with the most important ML frameworks, such as TensorFlow, PyTorch. It allows users to pick any model from their model zoo and test them on available datasets - MovieLens, Netflix Prize, Amazon, Tradesy etc., and set test-training splits. It also offers evaluation metrics such as MAE, RMSE, Recall, NDCG, AUC (ROC-curve). Finally, the user can see the statistics of the model's performance in a generated table.

C. Requirements

Requirements for our tool are based on Cornac's specs, defined in section *I.B,* as well as our own considerations: the program must have as input parameters: algorithm model, configuration, evaluation metrics and dataset; the output should be a table with the experiment's statistics in a configurable directory. Must have the possibility to input different algorithm parameters for comparative evaluation.

Collaborative filtering goal

As mentioned before, CF algorithms work based on analyzing and comparing user preferences. They need vast amounts of data to work properly. Regardless, they have gained popularity because of their diversity in recommendations and are widely used at Netflix, Amazon, YouTube. There are 2 types of CF: heuristic (*memory-based*) and *model-based*.

A. Heuristic approach

In its turn, the heuristic approach can be split into *user-item filtering* and *item-item filtering*.

User-item filtering choses an user and then finds other users with similar behavior and recommends the former items that were previously liked by the latter [5]. *Item-item filtering* choses an object instead of an user, then identifies users who liked this item, after which it finds other items that were also liked by these users (or ones similar to them).

The main difference between the heuristic approach and the model-based one is that it does not use gradient descent (or any other optimizator) to compute weights. The similarity between the users is calculated using cosine similarity or Pearson correlation coefficient, both being based only on arithmetical operations. Thus, algorithms that do not use weight computation and don't employ optimization are categorized as heuristic methods and considered easy to use. However, their performance drops when using sparse data and make this approach not scalable for real-world problems.

B. Model-based approach

In this approach, CF models employ machine learning algorithms and techniques for constructing recommendations. These algorithms can be sub-categorized into *matrix factorization, clustering*, and *deep learning*.

Matrix factorization Let A be a matrix of dimension (m,n). This matrix can be viewed as a product of 2 matrices of sizes (m,k) and (k,n). For example, A matrix can be the ratings users gave to movies [6]. Each row represents a user, while each column represents a movie. Obviously, this is a sparse matrix (not all users rated all movies). One of the benefits of Matrix Factorization (MF) is that it can infer user preferences when explicit feedback is not available [7]. Using *implicit feedback*, it can infer if a user might like a movie they have not yet seen and recommend it. MF can be formulated as an optimization problem with loss functions and constraints. Constraints are selected based on the model's features [8]. MF can be executed using various techniques, such as: Singular Value Decomposition (SVD), Probabilistic Matrix Factorization (PMF), or Non-negative matrix factorization (NMF) [9].

Clustering The idea of clustering is similar to the heuristic approach in RecSys. Clustering algorithms also utilize user similarity in order to predict ratings. The difference is that the similarity is computed using unsupervised learning methods and not cosine similarity or

[1] Cornac: A Comparative Framework for Multimodal Recommender Systems [Электронный ресурс]. - URL: https://cornac.preferred.ai/ (дата обращения: 14.07.2020).

the Pearson correlation coefficient [10]. Also, the number of similar users is constrained to *k*, making this approach more scalable. In order to deal with sparse data, K-means clustering algorithm was proposed, and became the most popular algorithm that uses this approach [11].

Deep learning Recently, deep learning has been actively used for building RecSys. Deep learning algorithms gained popularity because they can overcome the shortcoming of traditional methods and achieve high recommendation quality [12]. Due to its nature to capture nonlinear relationships between users and items, deep learning provides better representations for them, which leads to better recommendations [13]. With neural networks, users and items can be modeled into low dimensional vectors in a latent space [14]. There are many models based on deep learning; some of the most popular are Autoencoders, convolutional neural networks (CNN) based recommendations, Restricted Boltzmann machine-based recommendations (RBM) and also recommendations based on Recurrent Neural Networks (RNN) [15].

Major collaborative filtering algorithms review

A. Top-N Recommendations

Top-N recommendations are a good example of heuristic algorithms. The idea of the algorithm is to recommend a set of N items with highest ratings, which would be of interest for a user. Top-N recommendations analyze the user-item interaction matrix to find correlations between them to produce recommendations. Top-N are divided into 2 subgroups: *user-based Top-N recommendations* and *item-based Top-N recommendations* [16].

User-based Top-N recommendations first identify k most similar users (nearest neighbors) for a specific user, using cosine similarity or Pearson's correlation coefficient. After the neighbors have been identified, the corresponding rows in the interaction matrix are concatenated to obtain the set of items C, that interested similar users; the frequency of item incidence is also accounted for. Next, the most frequent N items previously unseen by the user from set C are recommended. This algorithm has limitations in terms of scalability and real-time execution [17].

Item-based Top-N recommendations were developed to overcome the scalability issue. The algorithm first identifies k most similar items and, reasoning from their features, creates the set C of potential recommendation candidates. Then the set U of items already seen by the user is deleted from C and all the new items similar to C and U, but unseen by the user, are collected into the final set. This set is sorted in descending order by item frequency and the user is recommended the first N objects [18].

B. SVD

1) Classical SVD

SVD algorithm is based on MF and is one of the most popular in RecSys; it was proposed by Simon Fank at the Netflix Prize competition[2]. It is partially based on the SVD algorithm from linear algebra, where the main formula is:

$$A = U\lambda V^T \quad (1)$$

Where matrix U is of size $m \times m$, V^T size of $n \times n$, and their columns are orthonormal. λ is a diagonal matrix of size $m \times n$, containing singular values of A. Because λ is a diagonal matrix, it can be combined with either U or V^T, which resulted into the classic formula of SVD recommender algorithm [7]:

$$\widetilde{r_{ui}} = q_i^T p_u \quad (2)$$

Each item and user is represented as a latent vector (q_i for object *i* and p_u for user *u*, respectively) of size *k*, which is a parameter of the model corresponding to the number of features. r_{ui} in this scenario is the expected rating by a user *u* on an item *i,* for example a movie rating. It is worth mentioning that linear algebra SVD can be computed only for dense matrices. If the initial matrix has a missing value, SVD cannot be calculated; in this case the initial matrix can be approximated. In order to obtain a dense initial matrix, missing values were filled in. Soon it turned out this scenario is not feasible, as it distorted the data. Thus, a special normalized model was proposed, that made use only of known ratings. In order to get the latent vectors q_i and p_u for users and items and predict the future rating r_{ui}, the system minimizes the normalized root square error on multiple known ratings:

$$min \sum_{(u,i)\in D} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \quad (3)$$

where *D* is the number of (u, i) pairs for which the rating r_{ui} is known. The system trains the model using previously known ratings. Its goal is to generalize these figures to predict future ratings. In order to avoid overfitting, the length of latent vectors is penalized. Regularization is executed by optimizing the loss and some other feature function, for example vector norms. The $\lambda\lambda$ responsible for regularization is computed using cross-validation.

2) Optimizations

The main mechanism of optimizing the formula to find factor vectors (3) is gradient descent. Simon Fank [19] used stochastic gradient descent, where the algorithm goes through all the user ratings in the training set. For each case, the system predicts r_{ui} r_{ui} and computes the error:

$$e_{ui} \stackrel{\text{def}}{=} r_{ui} - q_i^T p_u \quad (4)$$

Then the vectors are updated by taking a step in the opposite direction to the gradient of the loss function with some parameter γ γ and thus:

$$q_i \leftarrow q_i + \gamma(e_{ui}p_u - \lambda q_i) \quad (5)$$

$$p_u \leftarrow p_u + \gamma(e_{ui}q_i - \lambda p_u) \quad (6)$$

3) Bias

One of the strengths of MF is its flexibility in working with different data. Formula (4) analyzes the interaction between users and items and comes up with the rating. It happens that some users over-rate or under-rate some items, which in their turn can be less or more popular. That is why SVD's performance might suffer. To solve this problem, bias is introduced:

$$b_{ui} = \mu + b_i + b_u \quad (7)$$

where b_{ui} is the bias of the rating r_{ui}, b_i item bias, b_u user bias, and μ — global bias or the general rating mean. With this, formula (2) becomes:

$$\widetilde{r_{ui}} = \mu + b_i + b_u + q_i^T p_u \quad (8)$$

Finally, SVD learns by minimizing the square root error.

[2] MovieLens data [Электронный ресурс]. — URL: http://www.grouplens.org (дата обращения: 14.07.2020).

4) SVD ++

In «Factorization Meets the Neighborhood» [17] the authors describe a modification of the SVD model. Each user is associated with 2 groups of items: *R(u)* — the set of items with known ratings and *N(u)* — the set of items which do not have an explicit rating. SVD++ method uses *implicit feedback*. Because implicit feedback may sometimes be unavailable, *N(u)* can be replaced *R(u)*, as $R(u) \subset N(u)$ always holds.

5) Asymmetric SVD++

Along SVD++, authors in [17] propose an alternative model. The idea behind this model is also to use implicit feedback.

6) Time SVD++

One of the well-known flexible models is TimeSVD++. In datasets like MovieLens[3] [28] and Netflix Prize[4], besides the history of user ratings, there is also information about the time they were given. In order to make use of these data, the paper «Matrix Factorization Techniques for Recommender Systems» [7] suggested plugging into SVD++ some knowledge about time. Time SVD++ makes use of bias. 2 types of bias can be used: *item bias* and *user bias*. With *item bias,* time intervals in which ratings are observed are split into bins (30 in this paper) and for each item a time changing bias parameter - $b_{i,Bin(t)}$ - is added. The parameter is chosen based on the bin of variable *t*. *User bias* is more difficult to compute; similarly, the user's bias $b_{u,t}$ is added for each user, along with a term α_u.

C. K-means clustering

Clustering considers objects as vectors in a high-dimensional space and forms groups such that objects inside every group are most similar to one another and most different from objects in other clusters. In clustering, the dimensionality of input data is divided into k partitions. Choosing the right initial k is crucial, because incorrect choice of the k parameter may lead to fault results. The main concept of the algorithm consists of minimizing the sum of the squared distance between the data points and centroids:

$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2 \quad (9)$$

where k is the number of clusters, S_i — resulted clusters, $i = 1,2, \dots, k$, and μ_i — centroid af all $x$vectors from cluster S_i. The algorithm was first described in «An improved parallel K-means clustering algorithm with MapReduce»[5] and consists of 4 steps: chose k, which will denote the centroids of the cluster; calculate the Euclidean distance between the k centroids and the rest of the vectors; the closest vectors are assigned to that one cluster; centroids is re-computed; the algorithm stops after centroids stop changing or on another specific condition.

D. Autoencoders

The Autoencoder is an artificial neural network with input, hidden and output layers. The input and output layers have the same number of neurons. It is an unsupervised learning method that encodes input data and outputs learned data representations, by ignoring the "noise" in the data. The basic idea is that it learns to copy its input to its output. The autoencoder first maps data to some latent, more compact representation, only to restore the data to the original dimensionality, but free of noise, with only the most important information retained. Hidden layer has the functions to $encode(x) : R^n \to R^d$ and $decode(z) : R^d \to R^n$ [21]. The goal of the autoencoder is to obtain a *d*-dimensional representation of data in such a way that the error between x and $f(x) = decode(encode(x))$is minimal. During training, the encoder gets as input, for example, the vector of movie ratings and computes its latent representation z. In forward-pass, it takes a user from the training data (represented by their rating-vector $x \in R^n$, where *n* is the number of rated items). Notice that *x* is sparse, while the input of the decoder, $f(x) \in R^n$, is dense, and contains predicted ratings for all items. The latent representation is calculated by:

$$l = f(W * x + b) \quad (10)$$

Where *f* is some non-linear activation, W is the weight matrix and b is the bias. It has been found that it is important that the activation function *f* contains a non-zero negative part in the hidden layers [21]. It's impossible to fully restore input vector *x* in the output *y* by representation *z*; that's why after obtaining vector *y*, training resumes to using stochastic gradient descent for minimizing the loss (for example the MSE - Mean Squared Error). In «Denoising Autoencoder-Based Deep Collaborative Filtering Using the Change of Similarity» [22], the autoencoder has both the encoder and the decoder consisting of feed-forward neural networks with fully connected layers calculated by formula (1).

An extension of the simple autoencoder is the Deep Autoencoder, that has multiple hidden layers, which proves useful when working with complex data because of their tendency to learn higher order features.

E. Restricted Boltzmann Machines

Restricted Boltzmann Machine (RBM) is a generative stochastic neural net that determines the probability distribution over the input data. RBM is based on binary elements in a Bernoulli distribution that constitute the visible $v_i$and hidden $h_i$network layers. The connection between the layers is given by a weight matrix $W = (w_{i,j})$, with bias a_i for the visible layer and $b_i$for the hidden one. The probability distribution over the vectors of the visible and hidden layers is given by:

$$p(V) = \sum_h \frac{exp\ (-E(V,h))}{\sum_{V',h'}\ exp\ (-E(V',h'))} \quad (11)$$

where is called the energy function for the network (V,h). In RBM, the neurons form a bipartite graph, with visible units on one side (inputs) and hidden on the over, fully connected. The structure of the graph allows it to be trained using gradient descent. «Restricted Boltzmann Machines for Collaborative Filtering» [23] considers using RBM for recommendations. Say we have *M* items; *N* users and ratings are from 1 to *K*. In order to use RBM, the first problem to be addressed are the possible missing ratings. If all *N* users rated the same set of *M* items, then each user could be considered as a training instance for a RBM, with *M* SoftMax visible units. Each hidden unit could learn to model the dependency between ratings for different films. In the case when the majority of the ratings are missing, every user should be modeled with a different RBM. Each RBM has the same number of hidden units, but the visible SoftMax units correspond to the ratings given by the user.

[3] MovieLens data [Электронный ресурс]. URL: http://www.grouplens.org (дата обращения: 14.07.2020).

[4] MovieLens data [Электронный ресурс]. - URL: http://www.grouplens.org (дата обращения: 14.07.2020).

[5] Funk S. "SVD algorithm" [Электронный ресурс]. — URL: https://sifter.org/simon/journal (дата обращения: 14.07.2020).

Algorithm Quality Evaluation

The quality of the algorithm based on various metrics is the usual method in evaluating CF algorithms. Most common datasets used in RecSys evaluation are public: MovieLens, Netflix Prize Competition, Goodreads and BookCrossing, although in many cases proprietary, unpublic data is used. The quality criteria are accuracy of the forecast or of the classifications, computed by specific metrics.

A. Forecast Quality

Forecast quality looks at the difference between the predicted rating by the recommender system and the real rating from the test set.

Mean Absolute Error MAE computes the difference between the prediction and the real rating. A normalized version of *MAE* called *NMAE* is also used. It normalizes the error by the difference between minimum and maximum ratings.

Root Mean Square Error RMSE was popularized after Netflix Prize. One of RMSE characteristics is that it disproportionally penalizes big errors, thus making it sensitive to incorrect predictions. As in the case of MAE the smaller the value, the better.

B. Classification Quality

The most important metrics to evaluate how good the algorithm distinguishes good examples from bad ones are Precision, Recall, F-score, and ROC curve. They measure if the recommended item is suitable, without which one is better [24]. Of course, this is not always convenient, because the user usually focuses on best matches. In such cases, NDCG is employed.

Precision and Recall Considering 4 types of possible output scenarios for predictions: True Positive, True Negative, False Positive and False Negative, *Precision* characterizes the aptitude of the model to make correct predictions, while *Recall* gives the number of correct predictions out of all predictions.

F-score This metric was introduced to balance the need of maximizing Precision and Recall [25]. It includes the information about both Precision and Recall, proving itself as the best candidate for evaluating classification quality.

ROC-curve The metric based on the ROC-curve is an alternative to Precision and Recall. It represents the dependency graph between true positives rates (also called *sensitivity*; X-axis) and the total number of false positives (also called *specificity*; Y-axis) for different classification thresholds from point (0,0) (min) to (1,0) (max)[6]. One major characteristic of the method is AUC (area under the curve) that is used to identify the capacity of the model to attribute to a random correct example a higher rank rather than a lower. It is an aggregate measure of model's performance; a model that outputs 100% correct classifications will have an AUC of 1.0. A good model's graph will flatten in the top left corner [26, 27].

DCG Discounted cumulative gain, usually noted as $DCG@k$, is used if the order or the recommendations is important. The metric accounts for relevancy and position of the recommended item. Relevant objects will rank higher. Also, there is the possibility to calculate the normalized version of the metric, denoted as $NDCG@k$. In this case the value of $DCG@k$ is divided by $IDCG@k$, which is the maximum possible value for the $DCG@k$ for user u if all the recommendations are ranked in correct order.

T a b l e I. **Comparative analysis of different algorithms**

rmse	mae	model	batch_size	epoch	optimizer	dataset	train_loss	eval_loss	ndcg
0.897662	0.707663	Embedding	1024	40	Adam	ml-1m	0.783518	0.805796	0.99172
0.91333	0.721112	Embedding	2048	40	Adam	ml-1m	0.815496	0.834172	0.991212
0.913936	0.721648	Embedding	4096	40	Adam	ml-1m	0.815425	0.835279	0.99121
0.913981	0.721799	Embedding	5096	40	Adam	ml-1m	0.815205	0.835361	0.991221
0.921221	0.732688	Mf	2048	40	Adam	ml-1m	0.952845	0.885261	0.99106
0.924686	0.740538	Mf	4096	40	Adam	ml-1m	0.964502	0.890183	0.991041
0.919099	0.737387	Mf	5096	40	Adam	ml-1m	0.957858	0.881622	0.991163
0.916629	0.72576	NFC	1024	40	Adam	ml-1m	0.976101	0.978186	0.991013
0.920278	0.729483	NFC	2048	40	Adam	ml-1m	1.016503	1.017418	0.990979
0.923661	0.736542	NFC	4096	40	Adam	ml-1m	1.069894	1.069072	0.990804
0.908309	0.716352	SVD	1024	40	Adam	ml-1m	0.809234	0.825025	0.991179
0.908263	0.71652	SVD	2048	40	Adam	ml-1m	0.807266	0.824942	0.991184
0.908069	0.716626	SVD	4096	40	Adam	ml-1m	0.805809	0.824589	0.991202
0.908197	0.717079	SVD	5096	40	Adam	ml-1m	0.806059	0.824822	0.991209
0.890178	0.701217	Embedding	1024	50	Adam	ml-1m	0.76251	0.792417	0.991969
0.906517	0.715482	Embedding	2048	50	Adam	ml-1m	0.801086	0.821773	0.991433
0.913532	0.721314	Embedding	4096	50	Adam	ml-1m	0.814093	0.83454	0.991211
0.923723	0.737501	Mf	1024	50	Adam	ml-1m	0.964439	0.888988	0.990987
0.924147	0.740093	Mf	4096	50	Adam	ml-1m	0.966159	0.884982	0.991166
0.924065	0.740294	Mf	5096	50	Adam	ml-1m	0.967744	0.886099	0.99121
0.915574	0.727545	NFC	1024	50	Adam	ml-1m	0.962916	0.960318	0.990916
0.915199	0.727524	NFC	2048	50	Adam	ml-1m	0.975107	0.97591	0.990985
0.920537	0.732639	NFC	5096	50	Adam	ml-1m	1.042509	1.041333	0.990921
0.908367	0.716569	SVD	1024	50	Adam	ml-1m	0.809244	0.825131	0.991173
0.90823	0.716479	SVD	2048	50	Adam	ml-1m	0.80724	0.824882	0.991187

[6] Guo S. Analysis and Evaluation of Similarity Metrics in Collaborative Filtering Recommender System: Thesis of the Degree Programme in Business Information Technology. Tornio, 2014. [Электронный ресурс]. - URL: https://www.theseus.fi/bitstream/handle/10024/80193/Shuhang%20Guo_BIT10_K0951349_FinalThesis.pdf?sequence=1 (дата обращения: 14.07.2020).

Implementation Details

In this chapter we will describe the program package we created. It corresponds to the requirements from II.*C*, containing 4 functional modules. The package is developed on Python 3 using Tensorflow.

A. Workflow description

First step consists of defining the input parameters into the command line, which can be used with «—eval_only» mode (no training) or «—config» parameters. The config file contains information about the datasets (MovieLens, Goodreads, BookCrossing); the model parameters (batch size, epoch, learning rate); model architecture (keras layer description and directory path); used optimizers; usage of *learning rate schedule* ; evaluation metrics; possibility to save results into MLFlow; data input directory. In the second part, the program performs training. The third step consists in computing the model's performance. Finally, the last step outputs the result into the selected directory in the form of a .csv file containing the name of the model, its parameters, used datasets as well as the evaluation metrics. The possibility of multiple launch for evaluating parameter tuning is executed by a special loop that creates a multitude of parameter combinations and automatically launches the created configuration.

B. Available models and evaluation metrics

The following models were considered: *Matrix Factorization* with: SVD, SVD ++, Variational Autoencoder, Bayesian personalized ranking; *Neural collaborative Filtering*; *Turicreate* (CF algorithm developed by Apple); *Autoencoder*.

As evaluation metrics we used *Accuracy, Precision, Recall, RMSE, AUC-ROC curve* (with implementation from Keras); *DCG score* with k=5 most relevant results and *NDCG score* with k=5 most relevant results.

C. Input parameter reading, training and results output

Input parameters are sent to the main program using the command line and loaded using a yaml.load file. This triggers the train_model function (with arguments parsed from the config file) which trains each model form the config file for all batch_size, epoch, learning_rate values and selected optimizers. The model is compiled by the model.compile function and trained in model.fit (skips this step if eval_only was selected). After training is completed, the model is evaluated using model.evaluate (with users, items and ratings from datasets as parameters). The results are saved in a .csv file as well as in MLFlow if that was opted for. The results are saved after each training loop.

Experimental evaluation

This experiment considers the models described in chapters IV and V. We used the MovieLens dataset, as it is the most used in evaluating recommendation algorithms performance (but Goodreads and BookCrossing are also available). Experiments were run on 2 CPU Intel Xeon Silver Cascade Lake-SP 4208 8C/16T; 4 GPU NVIDIA PNY Quadro P5000 16GB GDDR5xPCIe3; RAM 128 GB; 2 Tb SSD. Source code for the experiment we published in https://github.com/Ilyushin/rec-tool.

A. Parameter tuning and algorithm performance

We demonstrated the influence of parameter tuning on the quality of the model by using the multiple launch strategy that automatically combines parameters over the fixed datasets and evaluation metrics to get a result. Because of the large number of possible combinations, we will mention only the most interesting results and will shoe a small portion of the results in the comparative analysis section. The dataset has 6040 unique users and 3706 unique items. For example, the best performance shown by Embedding was for batch size 1024 for 50 epochs, while MF performed best on batch size of 5096 on same number of epochs. Neural Collaborative Filtering got good result with 50 epochs and a batch size of 2048. For The performance of each model is determined by the correct parameter tuning. Our package demonstrates this dependency and offers the possibility to obtain the best configurations of parameters of datasets and evaluation metrics.

B. Comparative analysis of different algorithms

For comparative analysis, we launch the algorithms in the best configurations obtained in VII.*A*. A part of the results for comparative analysis on Movielens are presented in Table I. As we can see, the developed tool gives the possibility to demonstrate the difference in quality for various Collaborative Filtering algorithms on fixed evaluation metrics and datasets. Also, we can observe that classical MF algorithms can compete with new deep learning methods, giving the correct tuning. These results demonstrate once more that the problem of parameter tuning is still actual and needs proper investigation.

C. Testing time gain analysis

We have implemented experiments to demonstrate the time gain for manual vs multiple launch. Using a special time-measurement utility, we calculated the time needed for a person to manually launch all the tests for all the configurations and compared it with quality, we do not care about the real metrics outputs and used only random parameter values, while the datasets for both manual and automatic launches are the same. We found that for the Movielens dataset, the researcher's times for launching the recommendation models and looking for the best parameters' combinations are:

- Manual launch - approx. 6 hours.
- Automatic launch - about 30 minutes.

It is easily observable that the time gain constitutes over 5 and half hours, which demonstrates the capability of our program to dramatically decrease the time needed to manually pick parameter configurations and launches for algorithm testing. Also, this approach eliminates the need of a person to be physically present to relaunch an experiment after the previous terminates, as our program automates this process and frees valuable human resources.

Conclusion

In this paper we investigated sophisticated functionalities of complex programs, collaborative filtering algorithms and metrics for their evaluation. We developed and implemented a program that automates tests with these algorithms. We also conducted experiments that compared the quality of different algorithms and demonstrated the influence of specific parameter tuning on their performance. Also, we have shown that matrix factorization algorithms can compete with deep learning models, given the right parameter tuning. Besides that, we demonstrated a significant time gain when testing the algorithm using our program rather than launching each experiment by hand.

References

[1] Miller B.N., Konstan J.A., Riedl J. PocketLens: Toward a personal recommender system. *ACM Transactions on Information Systems*. 2004; 22(3):437-476. (In Eng.) DOI: https://doi.org/10.1145/1010614.1010618

[2] Hofmann T. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*. 2004; 22(1):89-115. (In Eng.) DOI: https://doi.org/10.1145/963770.963774

[3] Yu K., Schwaighofer A., Tresp V., Xu X., Kriegel H.-P. Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*. 2004; 16(1):56-69. (In Eng.) DOI: https://doi.org/10.1109/TKDE.2004.1264822

[4] Resnick P., Iacovou N., Suchak M., Bergstrom P., Riedl J. GroupLens: an open architecture for collaborative filtering of netnews. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW '94).* Association for Computing Machinery, New York, NY, USA; 1994. p. 175-186. (In Eng.) DOI: https://doi.org/10.1145/192844.192905

[5] Balabanović M., Shoham Y. Fab: content-based, collaborative recommendation. *Communications of the ACM*. 1997; 40(3):66-72. (In Eng.) DOI: https://doi.org/10.1145/245108.245124

[6] Cacheda F., Carneiro V., Fernández D., Formoso V. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web*. 2011; 5(1):1-33. Article No. 2. (In Eng.) DOI: https://doi.org/10.1145/1921591.1921593

[7] Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. *Computer*. 2009; 42(8):30-37. (In Eng.) DOI: https://doi.org/10.1109/MC.2009.263

[8] Bell R.M., Koren Y. Improved Neighborhood-based Collaborative Filtering. In: *Proceedings of KDDCup and Workshop.* San Jose, California, USA; 2007. p. 7-14. Available at: https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Neighbor-Koren.pdf (accessed 14.07.2020). (In Eng.)

[9] Sarwar B., Karypis G., Konstan J., Riedl J. Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. In: *Fifth International Conference on Computer and Information Science*. 2002. Available at: http://files.grouplens.org/papers/sarwar_SVD.pdf (accessed 14.07.2020). (In Eng.)

[10] Ungar L.H., Foster D.P. Clustering Methods for Collaborative Filtering. In: *Proceedings of the 1998 Workshop on Recommender Systems*. AAAI Press, Menlo Park; 1998. (In Eng.)

[11] O'Connor M., Herlocker J. Clustering items for Collaborative Filtering. In: *Proceedings of the ACM SIGIR Workshop on Recommender Systems: Algorithms and Evaluation*. Berkeley, California, USA; 1999. (In Eng.)

[12] He X., Liao L., Zhang H., Nie L., Hu X., Chua T. Neural Collaborative Filtering. In: *Proceedings of the 26th International Conference on World Wide Web (WWW'17).* International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE; 2017. p. 173-182. (In Eng.) DOI: https://doi.org/10.1145/3038912.3052569

[13] Zhang S., Yao L., Sun A., Tay Y. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*. 2019; 52(1):1-38. (In Eng.) DOI: https://doi.org/10.1145/3285029

[14] LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015; 521:436-444. (In Eng.) DOI: https://doi.org/10.1038/nature14539

[15] Xue H.-J., Dai X., Zhang J., Huang S., Chen J. Deep Matrix Factorization Models for Recommender Systems. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17).* Melbourne, Australia; 2017. p. 3203-3209. (In Eng.) DOI: https://doi.org/10.24963/ijcai.2017/447

[16] Wang H., Wang N., Yeung D.-Y. Collaborative Deep Learning for Recommender Systems. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15).* Association for Computing Machinery, New York, NY, USA; 2015. p. 1235-1244. (In Eng.) DOI: https://doi.org/10.1145/2783258.2783273

[17] Sarwar B., Karypis G., Konstan J., Riedl J. Item-based collaborative filtering recommendation algorithms. In: *Proceedings of the 10th international conference on World Wide Web (WWW '01).* Association for Computing Machinery, New York, NY, USA; 2001. p. 285-295. (In Eng.) DOI: https://doi.org/10.1145/371920.372071

[18] Koren Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '08).* Association for Computing Machinery, New York, NY, USA; 2008. p. 426-434. (In Eng.) DOI: https://doi.org/10.1145/1401890.1401944

[19] Karypis G. Evaluation of Item-Based Top-N Recommendation Algorithms. In: *Proceedings of the tenth international conference on Information and knowledge management (CIKM '01).* Association for Computing Machinery, New York, NY, USA; 2001. p. 247-254. (In Eng.) DOI: https://doi.org/10.1145/502585.502627

[20] Babu S. Towards automatic optimization of MapReduce programs. In: *Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10).* Association for Computing Machinery, New York, NY, USA; 2010. p. 137-142. DOI: (In Eng.) https://doi.org/10.1145/1807128.1807150

[21] Liao Q., Yang F., Zhao J. An improved parallel K-means clustering algorithm with MapReduce. In: *2013 15th IEEE International Conference on Communication Technology.* Guilin; 2013. p. 764-768. (In Eng.) DOI: https://doi.org/10.1109/ICCT.2013.6820477

[22] Barbieri J., Alvim L.G.M., Braida F., Zimbrão G. Autoencoders and recommender systems: COFILS approach. *Expert Systems with Applications.* 2017; 89:81-90. (In Eng.) DOI: https://doi.org/10.1016/j.eswa.2017.07.030

[23] Suzuki Y., Ozaki T. Stacked Denoising Autoencoder-Based Deep Collaborative Filtering Using the Change of Similarity. In: *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA).* Taipei; 2017. p. 498-502. (In Eng.) DOI: https://doi.org/10.1109/WAINA.2017.72

[24] Salakhutdinov R., Mnih A., Hinton G. Restricted Boltzmann machines for collaborative filtering. In: *Proceedings of the 24th international conference on Machine learning (ICML*

'07). Association for Computing Machinery, New York, NY, USA; 2007. p. 791-798. (In Eng.) DOI: https://doi.org/10.1145/1273496.1273596

[25] McLaughlin M.R., Herlocker J.L. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '04).* Association for Computing Machinery, New York, NY, USA; 2004. p. 329-336. (In Eng.) DOI: https://doi.org/10.1145/1008992.1009050

[26] Brzezinski D., Stefanowski J. Prequential AUC: properties of the area under the ROC curve for data streams with concept drift. *Knowledge and Information Systems*. 2017; 52:531-562. (In Eng.) https://doi.org/10.1007/s10115-017-1022-8

[27] Davis J., Goadrich M. The relationship between Precision-Recall and ROC curves. In: *Proceedings of the 23rd international conference on Machine learning (ICML '06).* Association for Computing Machinery, New York, NY, USA; 2006. p. 233-240. (In Eng.) DOI: https://doi.org/10.1145/1143844.1143874

[28] Herschtal A., Raskutti B. Optimising area under the ROC curve using gradient descent. In: *Proceedings of the twenty-first international conference on Machine learning (ICML '04).* Association for Computing Machinery, New York, NY, USA; 2004. p. 49. DOI:https://doi.org/10.1145/1015330.1015366

Submitted 14.07.2020; revised 20.08.2020; published online 30.09.2020.
Поступила 14.07.2020; принята к публикации 20.08.2020; опубликована онлайн 30.09.2020.

About the authors:

Olga A. Alieva, MA student of the Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1, Leninskie gory, Moscow 119991, Russia), ORCID: http://orcid.org/0000-0002-9503-2042

Elena Gangan, Junior ML Specialist of the Faculty of Mathematics and Computer Science, Babes-Bolyai University (1 Mihail Kogălniceanu St., Cluj-Napoca 400084, Romania), ORCID: http://orcid.org/0000-0002-6471-6492, lenagangan2@gmail.com

Eugene A. Ilyushin, Senior Software Developer of Open Information Technologies Lab, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1, Leninskie gory, Moscow 119991, Russia), ORCID: http://orcid.org/0000-0002-9891-8658, john.ilyushin@gmail.com

Alexey I. Kachalin, Deputy Head of the Cyber Defense Center, PJSC "Sberbank of Russia" (19 Vavilova St., Moscow 117997, Russia), ORCID: http://orcid.org/0000-0003-3039-7160, a.kachalin@gmail.com

All authors have read and approved the final manuscript.

Об авторах:

Алиева Ольга, магистрант факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова» (119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1), ORCID: http://orcid.org/0000-0002-9503-2042

Ганган Елена, младший специалист по машинному обучению, факультета математики и информатики, Университет Бабеш-Боляи (400084, Румыния, г. Клуж-Напока, ул. Михаила Когэлничану, д. 1), ORCID: http://orcid.org/0000-0002-6471-6492, lenagangan2@gmail.com

Ильюшин Евгений, ведущий программист лаборатории открытых информационных технологий, факультет вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова» (119991, Россия, г. Москва, ГСП-1, Ленинские горы, д. 1), ORCID: http://orcid.org/0000-0002-9891-8658, john.ilyushin@gmail.com

Качалин Алексей, заместитель руководителя центра кибербезопасности, ПАО «Сбербанк России» (117997, Россия, г. Москва, ул. Вавилова, д. 19), ORCID: http://orcid.org/0000-0003-3039-7160, a.kachalin@gmail.com

Все авторы прочитали и одобрили окончательный вариант рукописи.