Recommendation Systems with Machine Learning

Alexandra Fanca
Automation Department
Technical University of Cluj Napoca
Cluj, Romania
alexandra.fanca@aut.utcluj.ro

Adela Puscasiu
Automation Department
Technical University of Cluj Napoca
Cluj, Romania
adela.puscasiu@aut.utcluj.ro

Dan-Ioan Gota
Automation Department
Technical University of Cluj Napoca
Cluj, Romania
dan.gota@aut.utcluj.ro

Honoriu Valean
Automation Department
Technical University of Cluj Napoca
Cluj, Romania
honoriu.valean@aut.utcluj.ro

***Abstract*—Recommender systems are a subclass of information filtering systems. These systems are specialized software components, which usually make part of a larger software system, but can also be standalone tools. A recommender system's main goal is to provide the user software suggestions for items that can be useful. The suggestions are related to different decision-making mechanisms, different techniques, such as, what product to buy, what movie to watch, or what vacation to reserve. In the context of recommender systems, the general term "item" refers to what the system is actually recommending to its users. The paper presents the development and the comparison of multiple recommendation systems, capable of making item suggestions, based on user, item and user-item interaction data, using different machine learning algorithms. Also, the paper deals with finding different ways of using machine learning models to create recommendation systems, training, evaluating and comparing the different methods in order to provide a general but accurate solution for ranking prediction.**

Keywords—recommendation systems, machine learning algorithms, decision-making mechanisms, ranking prediction.

I. Introduction

Given a system that has a huge amount of users and a similar amount of content to present for them, the filtering process becomes crucial. Nobody can expect a user to search manually through thousands or even hundreds of thousands of different items, whether these are movies, products or news, in order to find what he is looking for. Without recommendations, the users would come in contact only with the direct search result, that in the case of a tremendous amount of items, would limit the number of returned data to tens, maybe hundreds of items if the user looks through multiple pages. Even in the case of smaller e-commerce websites or news sites, where items are categorized properly, the number of items may exceed a user's ability to find what he is looking for.

Recommender systems usually focus only on a unique type of item, for example, videos, music, and with respect to their design, their main recommendation method used to make decisions and their graphical user interface are all tailored to that specific type of item [1, 2].

Different users or groups of users receive various suggestions because recommendations are usually made by taking into account the unique properties of the users. Non-personalized recommendations are easier to create and can be found mainly in magazines or newspapers [3].

Users may find some specific items a system has to offer compelling, but the problem is that they might never find out their existence if the system contains too many items. The goal of the recommender is to show the user a new set of items and possibilities, which they would not look up on their own.

A significant amount of world-leading companies are already using recommender systems in their everyday operations to make users spend more time or money on their websites [4, 5, 6].

One of the main problems is that most recommender systems are built by many researchers and developers for performing extremely well in a specific task. This means both a huge time and financial investment for anyone who is looking for such a system to integrate.

The paper presents a solution that uses different pre-trained machine learning models and traditional approaches as well and applies them on a dataset in order to provide such a recommender. The results are compared in order to identify possible problems, tradeoffs, shortcomings but also advantages these systems can have. An online dataset was used in order to train different models both on the local machine, using ML.NET and in the cloud with Azure Learning Studio. Once the models have been trained, the system should be able to recommend different movies for users, by predicting what rating the user would have given if he had already seen that specific movie.

The rest of the paper is organized as follows: chapter II presents theoretical information in the field, in chapter III describes the proposed solutions, and chapter IV exposed some future improvements and conclusions.

II. State of the Art

Nowadays artificial intelligence (AI), machine learning (ML) and deep learning (DL) are trending fields. These terms are frequently used as alternatives for each other, however, this is not always correct. It has to be mentioned that from all these terms AI is the most general concept, ML being just a subset of AI and DL making part of ML [7]. The main objective of any ML algorithm is to generalize beyond the training samples, to understand and interpret data that it has never 'seen' before with success.

There are multiple techniques of creating a recommender system, all based on different aspects of the collected data and the environment it is part of.

978-1-7281-1951-9/20/$31.00 ©2020 IEEE

Recommender systems can be broken into three main categories [8]: content-based filtering, collaborative filtering, and hybrid systems.

Collaborative filtering algorithms are more often implemented than the others and often lead to better predictive performance. However, each technique has its advantages and disadvantages that must be taken into consideration before implementation [9].

A. Content-Based Filtering

The basic idea behind content-based filtering is that each item has some features. Recommender systems applies a content-based recommendation approach analyze, a set of documents and/or descriptions of items previously rated by the user, and creates a model or profile of user interests based on the features of the objects rated by that user [10]. Users are associated with a set of preferences related to item contents. A profile can be created explicitly by the user or automatically generated based upon his/her past actions. The profile is a structured representation of user interests, used to recommend new interesting items. The recommendation process can be described as comparing the attributes of the user profile with the attributes of an item. The outcome is a relevance score that represents how interested is the user in that given item. If a user profile precisely models user interests, it is of huge advantage for the effectiveness of an information retrieval operation.

The content-based recommendation requires proper techniques for representing the items and creating the user profiles, along with some algorithm for matching a user profile with an item's representation.

The use of content-based recommendation has several advantages, like independence from users, transparency, etc. As with any technique, the content-based approach also has some serious disadvantages, which must be taken into consideration: limited content analysis or over-specialization.

Content-based recommender systems are present in a variety of applications. LIBRA [11] uses a naive Bayes text categorization algorithm for recommending books by using product descriptions gathered from the Amazon online store. Another good example is Intimate [12], which creates movie recommendations by applying text categorization methods to learn from movie synopses collected from IMDB. The user has to categorize a minimum amount of films in order to get suggestions from the system: terrible, bad, below average, above average, good and excellent.

Recommenders are also present in news suggestions, where DailyLearner has to be mentioned. It keeps two user models, one for short-term interests and one for long-term interests. The short-term interest profile is based on nearest neighbor text classification, while for long-term interests a naïve Bayesian model is created.

A great web recommender is Letizia [13], which was developed as a browser extension that creates a model, tailored to the user. It builds from keywords from the user's interests by following the user's browsing. It is based on implicit observations to learn the user's interests. By putting a page in the bookmarks shows a strong sign that the user is interested in the content of that page.

The research of "mainstream" recommender systems conducted in the last 10-15 years shows that the keyword-based representation for users and items as well offer very good results, given that the necessary amount of user interest is available. The majority of content-based recommenders are created as text classifiers using training sets of documents, which represent user interests or the lack of it. As a result, for achieving high accuracy the training sets must contain a large amount of data. The main problem with this approach is the lack of "intelligence". If more complex characteristics have to be taken into account, keyword-based techniques meet their limits.

B. Collaborative Filtering

Collaborative filtering is realized by analyzing the behavior of a group of users to make provide suggestions to other users. The preferences of other users influence the recommendation. The main idea behind the collaborative filtering-based technique is that if a person has the same opinion as another person on a topic, then he is more likely to share that other person's opinion on another topic than that of a randomly chosen person. One of the simplest examples would be if a user gets a movie recommendation because his friend positively rated that movie and they have a similar history in rating movies.

Using a matrix the set of interactions can be visualized, where each entry (i, j) on the matrix represents the interaction between the user "i" and item "j". From another perspective, collaborative filtering can be viewed as a generalization of regression and classification.

Using collaborative filter-based recommendation has several advantages compared to content-based systems, like no domain knowledge necessary, serendipity, affinity to nuances, benefits of large user bases. Collaborative filtering-based approaches also come with some drawbacks, like complexity and expense, cold start (the system needs enough information (user-item interactions) to work properly).

Collaborative filtering is currently one of the most frequently used approaches and usually provides better results than content-based recommendations. A perfect example would be YouTube's recommendation system. As it is presented in [14], their system is composed of two neural networks working together to provide recommendations, one for candidate generation and one for ranking.

C. Hybrid Systems

The simplest and most direct way to build a hybrid recommender system is to take the independent result of content and a collaborative-based recommender system, then using a voting scheme combine their predictions (fig. 1).

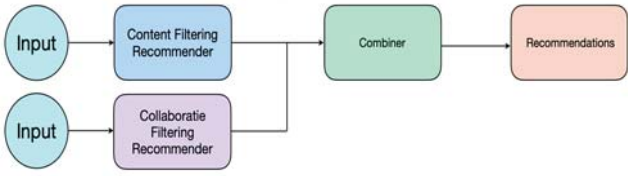

Fig. 1 Hybrid recommender system's basic architecture

[15] presents a method where the combination is done by choosing items that correspond to the user's profile and at the same time having positive ratings from the user's neighbors. In [16], the technique used compares users according to their content-based profiles and uses a collaborative filtering system where the generated similarity measures are used. In [17], the predictions based on the content are used to enrich

the rating matrix, and then collaborative filtering is run. In [18], item-based collaborative filtering is run, but before that uses the item's content descriptions and their associated rating vectors to calculate the similarity between them.

III. Proposed System

In this paper, two different movie recommender system designs are going to be presented, each based on a different recommendation method. The goal of the designed system is to predict what rating would a user give to a movie and based on this predicted rating to recommend movies.

The first type of system is implemented as an executable console application using the .NET Core and .NET Standard frameworks. It will be based on matrix factorization to provide recommendations. This is going to be a pure collaborative filtering approach.

The second approach will use Microsoft's cloud service, namely Azure Machine Learning Studio (AMLS). AMLS offers the possibility to create different machine learning models and train them directly in the cloud using Microsoft's immense cloud infrastructure [19]. Once a model is constructed it can be made available by exposing its functionality through a REST API. The model will use Microsoft's pre-trained hybrid model for the recommendation, Matchbox recommender.

A. Prerequisites of a Recommender System

The most important part of any machine learning model is data [20]. Typically some information about the users and items must be at disposal. Content-based filtering is a good start if only metadata is available about the users and items. If a sufficiently high amount of user-item interactions are accessible then more powerful collaborative or hybrid recommender systems can be implemented.

As a consequence of this, if more data is available then the system more probably becomes better. Also from a human resource perspective, one has to be sure that development team members are capable of understanding the data and manipulating it in a way to make it compatible with the methods that will be used to build the recommender system.

Interactions have to be defined with respect to the system so that data can be extracted easily.

B. Implementation

1) Structure of the Dataset

There are two main aspects of data that has to be taken into consideration: quantity and quality. First and foremost the dataset must contain enough entries to serve a learning algorithm well. The second aspect, quality, defines that data gathered and stored has to contain meaningful information in some way, even if its form is not directly compatible with machine learning algorithms, different transformations can be applied to achieve the desired structure.

Both of the recommendation systems are going to use the same dataset for training and evaluation. MovieLens 20M movie rating dataset is used from grouplens.org [21]. The dataset contains 20 million movie ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users. It also includes tag genome data with 12 million relevance scores across 1,100 tags.

The dataset contains the following files in .csv format:

- movies.csv: Stores the movie entities with movieId, title and genres fields;
- ratings.csv: Stores the rating entities with userId, movieId, rating and timestamp fields;
- tags.csv: Stores the tags added to movies by users with userId, movieId, tag and timestamp fields;
- genome-tags.csv: Stores the tag type entities with tagId and tag fields;
- genome-scores.csv: Stores the tag relevance score for each movie with movieId, tagId and relevance fields
- links.csv: Stores the link entities between imdb and tmdb databases with movieId, imdbId, and tmdbId fields.

The data provided by MovieLens20M is already structured very well, only some features have to be transformed from text to numbers in order to be compatible with machine learning algorithms. Also, the amount of data is more than enough for training and evaluating the model.

2) Collaborative Filtering in ML.NET

The first application is a console application built using .NET Core and C# programming language. The main goal of this application is to construct, train and tune a machine learning algorithm in order to make predictions of a specific data format.

The data comes from the MovieLens20M datasets ratings.csv file. It contains 20 million movie rating entries that will be used by the model during the learning phase and also for evaluation. A movie rating entry consists of the ID of the user who made the rating, the ID of the film that was rated, the rating itself and a timestamp when the rating happened. Since the prediction will return results in the form of movie IDs, a secondary file is necessary to make the recommendations human-readable, i.e. list the name of the movies instead of their IDs. As was mentioned in the previous section MovieLens20M contains such a file, movies.csv. This file holds the ID of the movie, its title and a list of tags that describe to what category the movie belongs to.

In order to be able to work with the data that comes from the data sources, specific classes must be created. Another class is also necessary to hold the predictions. It is having two fields, Label and Score.

After the necessary data structure is defined the next step is to import the data into the memory. Since the application is using only one data source for ratings, after it is loaded in the memory it must be split into two sets. The goal of this is to use the same file for both training and testing purposes. Once the data is in the memory the model configuration can be started. ML.NET offers several different options when it comes to recommendations, like one class matrix factorization, matrix factorization, and field-aware matrix factorization.

Figure 2 represents what data one should have in order to be able to use various matrix factorization methods. In this case, MovieLens database provides userId, movieId and rating fields. This makes it perfect for the simple matrix factorization method to use.

ML.NET offers a simple way of creating a matrix factorization recommender. First, the options for the matrix factorization have to be configured. There are five options that are necessary: MatrixColumnIndexColumnName, MatrixRowIndexColumnName, LabelColumnName,

NumberOfIterations and ApproximationRank. The first three options will tell the trainer algorithm which columns (fields of classes) represent the users, the items and were the result. The NumberOfIterations and ApproximationRank options can be used to tune the training of the model.

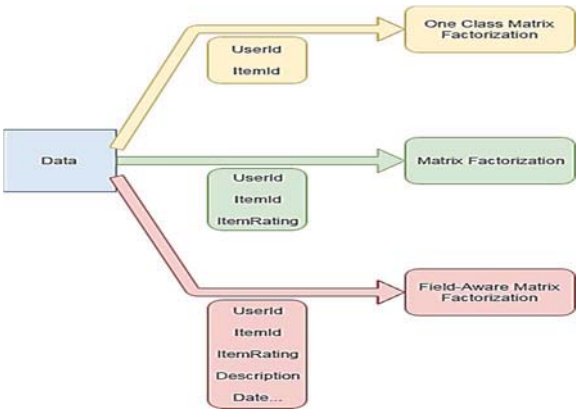

Fig. 2 Matrix factorization techniques available in ML.NET

After the options of the matrix factorization trainer are set up only the pipeline has to be configured. ML.NET works by the idea of the pipeline, the way of dataflow has to describe in order to be able to train a model. Figure 3 shows the complete structure of the pipeline.

```
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(
    inputColumnName: "userId",
    outputColumnName: "userIdEncoded")
.Append(mlContext.Transforms.Conversion.MapValueToKey(
    inputColumnName: "itemId",
    outputColumnName: "itemIdEncoded")
.Append(mlContext.Recommendation().Trainers.MatrixFactorization(options)));
```

Fig. 3 The complete structure of the pipeline.

As can be seen, it consists of three elements:

- First MapValueToKey: reads the userId column and builds a dictionary of unique Ids. Then creates an output column userIdEncoded containing an encoding for each Id. This step converts the Ids to numbers that the model understands;
- Second MapValueToKey: reads the itemId column and builds a dictionary of unique Ids. Then creates an output column itemIdEncoded containing an encoding for each Id. This step converts the Ids to numbers that the model understands;
- MatrixFactorization: performs matrix factorization on the encoded Id columns and the ratings. It uses the previously configured options to create the trainer. It is responsible for the calculation of item rating predictions.

The final step is evaluating the system. Evaluations are different for every system, and they depend only on the main objective of the system. For example, if a top n items recommender is implemented, there is no need to take into consideration the remaining items prediction score. The selected evaluation method can greatly influence the system's architecture.

There are two main types of evaluations in the case of recommender systems, offline and online methods. The presented model is evaluated using offline methods. As it was previously configured, 15% of the available data is going to be used for evaluation. In order to assess the performance of the system, the root means squared error (RMSE) value of the predictions will be taken into consideration. A comparison is presented in table 1 for a different number of training iterations of the model. These experiments were conducted on a laptop with Windows 7 64-bit operating system, with an Intel Core i5-3230M processor and 6GB of RAM.

TABLE I. COMPARISON OF RMSE BETWEEN ITERATIONSSSS

Number of iterations	Root mean squared error	Training times (sec)
5	0.8383	34
10	0.8248	42
15	0.8192	59
20	0.8163	62
25	0.8142	66
30	0.8127	75
35	0.8120	86
40	0.8108	98
45	0.8100	106
50	0.8094	115
55	0.8087	124
60	0.8082	134
65	0.8079	145
70	0.8074	150
75	0.8071	159
80	0.8069	165
85	0.8066	168

From table 1 it can be observed that for a not so powerful CPU the training time of such a model does not take up much time, even for the 17 million entries that are used for the training phase.

Figure 4 shows a better overview of how the RMSE value is changing when the number of iterations of training is modified. As can be seen, there is a high increase in predictions if the amount of iterations is increased to at least 35. From 35 to 75 the decrease in the RMS value is only marginal, less than 0.001. At this point, the number of iterations must be increased carefully in order to avoid overfitting. Anyhow these RMSE values show that after at least 35 training iterations the recommendation system based on the evaluation data performs well.

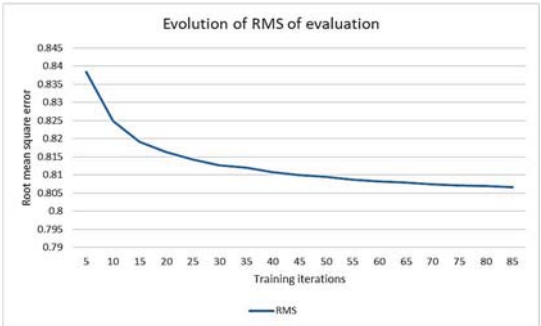

Fig. 4 Evolution of RMS of evaluation

3) The hybrid model in Azure Machine Learning Studio

For this approach, the same data is going to be used as before, the MovieLens20M dataset. This time a hybrid model will be built and evaluated, which extends the previous pure collaborative approach.

The first step is loading the data into the studio's workspace. The uploaded datasets are the movies.csv and the ratings.csv files, that hold the name of the movies and the ratings of the movies given by the users respectively.

The first element of the model is going to be the dataset paired together with a "Select Columns" data transformation element. This filters out all the columns that are needed for the training, skipping over the ones with hold no useful information. Just like before, in the case of the ML.NET application, from the ratings.csv the userId, movieId and rating columns are going to be used. The timestamp column does not seem relevant for predicting the rating of the movie. Furthermore, this time the genre of the movie is also going to be taken into consideration during training and prediction. The movies.csv holds this information along with the movieId and title attributes. Also, in this case, the initial dataset has to be split up for training and evaluation sets. The chosen model is Azure's Matchbox recommender, which creates a Bayesian recommender using the matchbox algorithm.

The "Train Matchbox Recommender" component uses a dataset of user-item-rating triple and can also handle user and item features if there are any. In this case, item features are available in the form of movie genres. These item features will help to solve the cold-start problem. This component has the options to configure the number of training iterations, which is set to 5 and the number of training batches, i.e. the number of batches to divide the data during training, which is set to 4 and the number of traits, set to 4, which is the number of latent traits that should be learned for every user and item.

After training the model has to be scored. There is a "Score Matchbox Recommender" module that does this job. Its inputs are the output of the trained model, the evaluation dataset and the item features, i.e. the movie genres.

The last step of every machine learning algorithm is evaluated and this is no different in this case. The final component of the model is an "Evaluate Recommender" module which, as its name suggests, is going to make the evaluation of the model.

Figure 5 shows both the structure and the flow of information on the complete system.

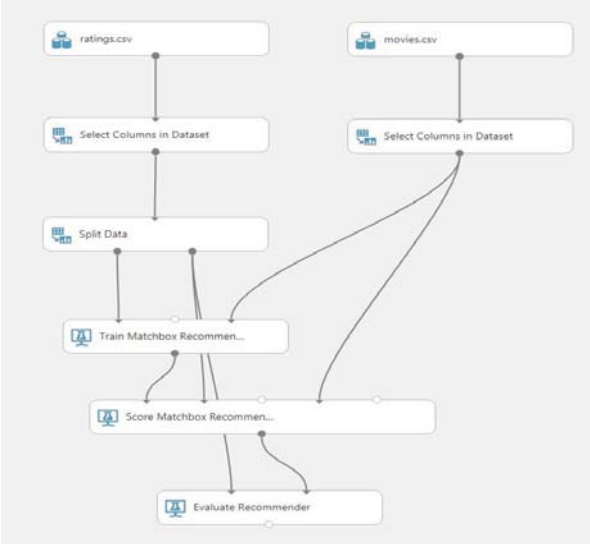

Fig. 5 Structure of the complete model

The evaluation module compares the test dataset ratings with the predicted ratings of the scored dataset. It offers some metrics about the results, namely the mean absolute error (MAE) and the RMSE. The results of the evaluation are presented in figure 6.

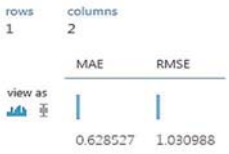

Fig. 6 Results of the evaluation component

Figure 7 shows the input data, which is the user with id 5, the movie with id 48, which is Pocahontas, and the given rating to the movie is 5.

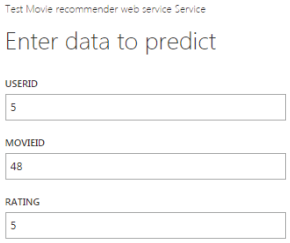

Fig. 7 Web service test input form

Pocahontas is an animated children's movie tagged with the following genres in the dataset: animation, children, drama, musical, romance. Based on all this information the expected recommendations should include similar movies that are mainly animated and address children. The top ten recommendations are presented in the next figure.

MovieId	Title	Genre
5	Father of the Bride Part II (1995)	Comedy
158	Casper (1995)	Adventure|Children
313	The Swan Princess (1994)	Animation|Children
317	TheSanta Clause (1994)	Comedy|Drama|Fantasy
1064	Aladdin and the King of Thieves (1996)	Animation|Children|Comedy|Musical|Romance
1367	101 Dalmatians (1996)	Adventure|Children|Comedy
3157	Stuart Little (1999)	Children|Comedy|Fantasy
3615	Dinosaur (2000)	Adventure|Animation|Children
91286	The Little Colonel (1935)	Children|Comedy|Crime|Drama
112355	A Kid for Two Farthings (1955)	Children|Comedy|Drama|Fantasy

Fig. 8 Top 10 movie recommendations

As can be observed the results are perfectly in balance with the expectations, as most of the recommendations are indeed animated children movies. As a conclusion, it can be said that all the recommendations are completely relevant regarding the given input.

4) Comparison of the Systems

Even though both of the implemented approaches addressed the sample problem, two different ways of implementation were presented each with its advantages and disadvantages.

The first approach of directly implementing a collaborative filtering model had the main advantage of coding the model itself, having control over every single parameter regarding the preprocessing, training and evaluation of the data. At the same time, it also leverages the power of the open-source ML.NET framework, which is constantly growing. Also, the model created with ML.NET can be exported in ONNX format to be later used by other applications. On the other hand, one of the main disadvantages is that C# coding skills are required, and since the ML.NET project is very new, it is exposed to frequent changes, making

refactorization of old code a must in order to use the features of the newest versions.

The best way of comparing the performances of the two models would be to deploy them in as part of an already released system and observe their impact on user behavior, performing this way online evaluation. However such a system is not available for the sake of this research and also gathering the necessary data would take a considerable amount of time, the results of offline evaluations are going to be compared. Using the RMSE values obtained during validation can offer some insight. It must be noted that the two systems implement two different approaches, so the final decision cannot be based completely on the RMSE value.

The pure collaborative approach yielded a lower RMSE value than the model trained in Azure, 0.81 for 45 iterations vs. 1.03 from Azure, but it must be taken into account that it suffers from the cold start problem.

The hybrid approach of the matchbox recommender eliminates this problem and at the same time can make use of additional features that come from the datasets. Using extra features in the first approach would have required a field aware matrix factorization model and more work to process the data.

IV. Conclusion

The systems were tested on the MovieLens20M database, which contains 20 million movie ratings and user and movie data. The tests were performed multiple times to adjust the parameters of the training algorithm in order to achieve better results. A collaborative filtering-based recommendation model was created, using a new and open-source machine learning framework, ML.NET.

The second model was built by using Microsoft's Azure Machine Learning Studio, which allowed a completely different approach towards developing machine learning solutions.

Although the first approach of a pure collaborative filtering method reached a better RMSE value, it does not mean that it is better. The matchbox recommender is a hybrid model, which solves the cold start problem. This in itself can mean a huge real-life difference, which cannot be measured with synthetic tests. The best recommendation method can be chosen by carefully observing what type of data is available and what problem should the recommendation address.

One of the biggest improvements of the above-mentioned systems would be to modify them to be capable of continuous training, i.e. learning continuously from the new data as it comes. The current systems must be retrained periodically in order to incorporate information from freshly delivered data.

An additional feature would be the implementation of a big data module, that would hold all the user, item and user-item interaction related information. This module then could be connected to the recommender system which would perform online learning, continuously improving and adapting to users.

References

[1] G.C. Capelleveen, C. Amrit, D.M. Yazan, W.H.M. Zijm, “The recommender canvas: A model for developing and documenting recommender system design”. Expert systems with applications, pp. 97-117, 2019.

[2] F. Ricci, L. Rokach, B. Shapira, *Introduction to Recommender Systems Handbook*. Boston, Massachusetts, United States of America: Springer, 2010.

[3] Y. Lim, “A Primer to Recommendation Engines”, Sep 10, 2019.

[4] J. Erickson and S. Wang. (2017, June) www.alizila.com. [Online]. https://www.alizila.com/at-alibaba-artificial-intelligence-is-changing-how-people-shop-online/

[5] I. MacKenzie, C. Meyer, and S. Noble. (2013, Oct.) www.mckinsey.com. [Online]. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

[6] C.A. Gomez-Uribe and N. Hunt, "The Netflix Recommender System: Algorithms, Business Value," *ACM Transactions on Management Information Systems (TMIS)*, vol. VI, no. 4, p. 7, Jan. 2016.

[7] J. Schmidt, M.R.G. Marques, S. Botti, et al. “Recent advances and applications of machine learning in solid-state materials science”. npj Comput Mater 5, 83, 2019.

[8] F. Ricci, L. Rokach, B. Shapira, “Recommender Systems: Introduction and Challenges”. In: Ricci F., Rokach L., Shapira B. (eds) Recommender Systems Handbook. Springer, Boston, MA, 2015.

[9] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh, “Recommendation systems: Principles, methods and evaluation”. Egyptian Informatics Journal, Volume 16, Issue 3, pp. 261-273, November 2015.

[10] P. Lops, M. Gemmis, G. Semeraro, “Content-based Recommender Systems: State of the Art and Trends”. In book: Recommender Systems Handbook, pp. 73-105, 2011.

[11] E. Cano and M. Morisio, “Hybrid Recommender Systems: A Systematic Literature Review”. Intelligent Data Analysis, vol. 21, no. 6, pp. 1487-1524, 2017.

[12] H. Mak, I. Koprinska, and J. Poon, "INTIMATE: A Web-Based Movie Recommender Using Text Categorization," Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence, 2003.

[13] L.H. Li, R. Hsu, F. Lee, “Review of Recommender Systems and Their Applications”. Computer Science, 2012.

[14] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," Mountain View, California, 2016.

[15] A. Bellogín, P. Castells, I. Cantador, “Neighbor Selection and Weighting in User-Based Collaborative Filtering: A Performance Prediction Approach”. ACM Transactions on the Web, No. 12, 2014.

[16] S.C. Stephen, H. Xie, and S. Rai, “Measures of Similarity in Memory-Based Collaborative Filtering Recommender System: A Comparison”, 4th Multidisciplinary International Social Networks Conference, 2017.

[17] G.S. Milovanovic, “Hybrid content-based and collaborative filtering recommendations with {ordinal} logistic regression (1): Feature engineering”, Data Science Central, 2017.

[18] M.G. Vozalis and K.G. Margaritis, "On the Enhancement of Collaborative Filtering by Demographic Data," in *Web Intelligence and Agent Systems*, Vol. 1, 2006

[19] A.M. Caulfield, E.S. Chung, A. Putnam, H.Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, D. Burger, “A Cloud-Scale Acceleration Architecture”. MICRO-49: The 49th Annual IEEE/ACM International Symposium on Microarchitecture, No. 7, pp 1–13, 2016.

[20] J. Mizgajski and M. Morzy, “Affective recommender systems in online news industry: how emotions influence reading choices”, User Model User-Adap Inter 29, pp.345–379, 2019.

[21] F. Maxwell Harper and J. A. Konstan, “The MovieLens Datasets: History and Context”. ACM Transactions on Interactive Intelligent Systems (TiiS), No. 19, 2015.