A Recommendation System with Personalizable Distributed Collaborative Filtering

Dragan Vidaković, Milan Segedinac, Đorđe Obradović, Goran Savić
Department of Computing and Control Engineering
University of Novi Sad, Novi Sad, Serbia
{vdragan, milansegedinac, obrad, savicg}@uns.ac.rs

***Abstract*—Collaborative Filtering technique is a state-of-the-art method in recommender systems. This technique has proved to be successful both in research and industry. However, it is challenging to find an adequate algorithm for calculating user similarity, which enables the recommender system to generate the best recommendation for given domain and data set. In this paper, we proposed an architecture which allows users to additionally personalize the recommendation by letting them choose algorithm for calculating user similarities, or to implement their own algorithm on a distributed service. Online tests on implemented application for movie recommendation show that architecture proposed in this paper gives users the ability to personalize recommendation for more efficient results.**

***Keywords*–Machine Learning; Collaborative Filtering; Recommender system;**

I. INTRODUCTION

Collaborative Filtering [1] is a subfield of Machine Learning that aims at creating algorithms to predict user preferences based on past user behavior in purchasing or rating items. It is a technique that has been acclaimed both in research and industry.

The input for the Collaborative Filtering prediction algorithm is set of user's ratings on items. Collaborative Filtering algorithm is typically decomposed into three generic stages [2]:

(1) calculating the similarity of all users to the active user (the user for whom a recommendation is searched),

(2) selecting a certain number of most users most similar to the active user,

(3) computing the active user rating prediction. This is done for a target item whose rating is unknown, and is obtained by calculating the normalized and weighted average of the ratings of the certain number of most similar users, found at (2) on the target item according to the user-to-user similarity computed at (1).

In a Collaborative Filtering recommender system, the items that are finally recommended to the active user are typically those with maximum rating prediction. In some cases all the items are suggested, but in this case they are ranked according to the predicted ratings [2].

Over the last 20 years, a great effort has been invested in research on how to automatically recommend different things to people. During that period, a wide variety of methods have been proposed. The *Recommender System Handbook* [3] provides in-depth discussion of a variety of recommender methods and topics, focused primarily on Collaborative Filtering. There is also a growing interest in problems surrounding recommendation. Algorithms for understanding and predicting user preferences are merely one piece of a broader user experience. A recommender system must interact with the user, both to learn user's preferences and provide recommendations; these concerns pose challenges for user interface and interaction design. There is room for algorithm refinement and much work to be done on user experience, personalization, data collection and other problems which make up the whole of the recommender experience [4].

A wide variety of algorithms is used for calculating user similarities. Finding an adequate algorithm which enables the recommender system to generate the best recommendation for given domain and data set is challenging.

The specific problem our work addressed was to find an architecture which allow system users to additionally personalize the recommendation by letting them choose the algorithm for calculating user similarities, or to implement their own algorithm for the same task. In order to maintain the efficiency of the recommender system, users can use distributed execution of newly implemented algorithms for calculating user similarities. Research has shown that no other party has found a solution to this problem.

The rest of this paper is organized as follows. In Section II, several related works are presented and discussed. In section III, we first introduce architecture and design of our system, and after that we describe the implementation of our system. In Section IV the experimental results of our system are presented and analyzed. Finally we make a brief concluding mark and give the future work in Section V.

II. RELATED WORK

Berkovsky, Kuflik and Ricci [2] showed that reliability of user-to-user similarity computation can be solved, and the accuracy of Collaborative Filtering recommendations can be improved by partitioning the collaborative user data into distributed repositories and aggregation information coming from these repositories. Their work explores a content-depending partitioning of collaborative movie ratings, where the ratings are partitioned according to the genre of the movie and presents an evaluation of four aggregation approaches. Evaluation demonstrated that the aggregation improves the accuracy of a

centralized system containing the same ratings and proved the feasibility and advantages of a distributed Collaborative Filtering. Main shortcoming of their work is the difficulty to apply it in recommendation applications. It requires additional work to classify the items into several topics or domains.

Han, Xie, Yang and Shen [5] proposed a distributed Collaborative Filtering algorithm together with significance refinement and unanimous amplification to further improve scalability and prediction accuracy. Their experiments showed that distributed Collaborative Filtering-based recommender system has much better scalability than traditional centralized ones with comparable prediction efficiency and accuracy.

Ekstrand, Riedl and Konstan [4] presented various algorithms for calculating user similarity. They proved that the appropriate algorithm for a particular application should be selected based on a combination of the characteristics of the target domain and context of use, the computational performance requirements of the application, and the needs to the users of the system. They concluded that understanding what those needs are can simplify algorithm selection.

Liu, Hu, Mian, Tian and Zhu [6] analyzed the shortages of the existing similarity measures in Collaborative Filtering. Also, they proposed new similarity model to overcome drawbacks of existing algorithms. Comparison of many similarity measures on real data sets showed that Collaborative Filtering recommendation system can reach different performance on same dataset depending on used similarity measure.

III. ELABORATION

The only way to accurately assess recommendation efficiency is to do an online evaluation – to test a system with real users. Therefore, majority of recommendation systems nowadays are incorporated in Web applications [4]. In this paper we propose an architecture that allows the user to implement algorithm for calculating user similarity. Due to performance, this calculation is safe to be executed on distributed service.

A. System Architecture and Design

We present a recommendation system, implemented as a Web application using Model-View-Controller (MVC) design pattern [7]. The system modular design contains four modules, shown in Figure 1.

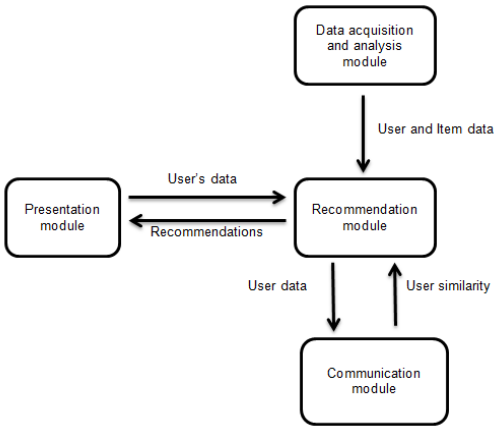

Figure 1. The system modular design

Data acquisition and analysis module is based on Linked data [8] and NoSQL databases [9]. This module contains data about user ratings from data set, and data about target items used for recommendations. Recommendation systems must handle accurate data in order to compute their recommendations and preferences, which is why it is important to collect reliable data and reduce the noise in user preferences data set.

Presentation module connects system user with Recommendation module. This module allows users to register themselves on the system and manage their distributed services. Beside this, this module forwards user's data and displays recommendations.

Communication module communicates with distributed services. This module attempts to conform to the design principles of Representational State Transfer (REST) [10].

Recommendation module communicates with other modules and performs recommendation. This module is basis of our architecture. System users can choose between already implemented user similarity algorithms, and their own algorithms implemented on distributed Web services.

B. Implementation

Data acquisition and analysis module stores data about users and target recommendation items in two separate collections in MongoDB [11] database. For every user in users collection, there are data about user ratings of items stored in JavaScript Object Notation – Linked Data (JSON-LD) [12] format. This format enables easier usage of data later in recommendation and eventual transfer to distributed user similarity calculation services via Communication module of the system. Target recommendation items data is stored in separated collection, using JavaScript Object Notation (JSON) format. According to information domain, there are libraries in this module for easier collection of the data. For movies as information domain, omdb [13] Python library was used to communicate with OMDb API[1] Web service to obtain movie data from the Internet Movie Database (IMDb)[2].

Presentation module is realized as a single page application using AngularJS framework [14]. Communication with Recommendation module relies on Representational State Transfer (REST). This module interacts with the user, collects it's preferences and provide recommendations. Therefore, our implementation of this module provided solution respecting good practices of user interface and interaction design.

Communication module performs user similarity calculation on distributed services. Using HTTP protocol this module sends data about two users, received from Recommendation module, to distributed services in JSON format. After user similarity calculation on distributed service, results are delivered to Communication module which forwards them back to Recommendation module. Although the implementation of Communication module was done using Django REST framework [15] and Python programming language, usage of HTTP protocol allows

[1] http://www.omdbapi.com/
[2] http://www.imdb.com/

user to implement distributed calculation service using any programing language and framewrok.

Recommendation module calculates recommendations using Collaborative Filtering, based on input data from Presentation and Data acquisition and analysis module. According to user's choice, this module calculates user similarity using already implemented algorithms (in this solution: *Euclidean Distance Score* and *Pearson Correlation Score* [4]) or using external service through Communication module. To complete active user rating prediction, we are calculating the normalized and weighted average of the ratings of most similar users according to calculated user similarity. Calculated recommendations are then sent to Presentation module in order to be displayed to system user. Implementation of Recommendation module was done using Python programming language and the Django Rest Framework. Active user's data, including account credentials and distributed services locations, are stored in SQLite relational database [16].

IV. EVALUATION

In this section, we describe data set and experimental results in online evaluation.

A. Data set

The information domain for system evaluation consists of a data set collected from Turi Machine Learning platform [3] with 82000 user preferences for movies represented as a *(User, Movie, Rating)* triple. Within Data acquisition and analysis module, collected data set was analyzed and filtered in order to provide accurate data for Recommendation module. Triplets with missing and duplicated values were removed from data set, and rating values were scaled to fit [0, 10] interval. Triplets were grouped by user in order to create user profile, and movie names are used to collect movie data from Internet Movie Database. After performed work, users' collection in MongoDB database was populated with 334 user profiles, and movies collection was populated with data of 5917 movies collected using omdb Python library.

B. Experimental results

Recommendation algorithms with similar numeric performance have been known to return observably different results, and a decrease in error may or may not make the system better at meeting the user's needs. The only way to accurately assess system accuracy is to do an online evaluation [4].

To evaluate our system, we decided to perform comparison of Collaborative Filtering recommendations using different user similarity calculation algorithms. Therefore, we implemented *Jaccard Similarity Coefficient* algorithm [4] on distributed service using Flask framework [17] and register it in our application. We then performed recommendation calculation for the same input data. We rate couple of movies and get Collaborative Filtering recommendations with three different similarity calculation algorithms:

(1) Euclidean Distance Score,
(2) Pearson Correlation Score,
(3) Jaccard Similarity Score (on distributed service).

Input	- Forrest Gump (1994), 10 - A Beautiful Mind (2001), 10 - Sin City (2005), 10		
	Pearson	**Euclidean**	**Jaccard**
Output	- Rory O'Shea Was Here (2004) - The Manchurian Candidate (2004) - The Terminal (2004)	- Seal Team Six: The Raid on Osama Bin Laden (2012) - Crash (2004) - Secret Window (2004)	- The Imitation Game (2014) -Mission: Impossible – Ghost Protocol (2011) - Tower Heist (2011)

Table 2. Online evaluation results

Results in Table 1. shows that for the same input data, user gets different recommendations, confirming that architecture provided in this paper gives user the ability to personalize recommendation by implementing distributed services using any programming language and framework to personalize recommendation for more efficient results.

V. CONCLUSION

In this paper, we propose an architecture which allows users to additionally personalize Collaborative Filtering recommendation by letting them choose already implemented algorithm for calculating user similarity, or to implement their own algorithm for the same task with preserved efficiency of the recommendation by enabling distributed execution of newly implemented algorithm. Proposed architecture allows users to implement distributed service using any programming language and any framework which conforms with the design principles of Representational State Transfer (REST). Online evaluation for movies as information domain confirms that provided architecture gives users the ability to additionally personalize Collaborative Filtering recommendation results by selecting appropriate algorithm for given information domain and data set. Also, distributed execution of that algorithm preserves system efficiency.

Our future work includes investigation on a data set extension by using Linked data benefits to include data from social media. We would also like to investigate abilities of collecting and analyzing feedback of system users to join Collaborative Filtering and Content Based recommendation into Hybrid recommender system.

REFERENCES

[1] J. Herlocker, J. A. Konstan and J. Riedl, *"Explaining Collaborative Filtering Recommendations"*, in Proceedings of ACM Conference on Computer Supported Cooperative Work, 2000.

[2] S. Berkovsky, T. Kuflik and F. Ricci, *"Distributed collaborative filtering with domain specialization"*, in Proceedings of the 2007 ACM conference on Recommender Systems, pp 33-40, 2007.

[3] F. Ricci, L. Rokach, B. Shapira and P. B. Kantor, *"Recommender System Handbook"*, Springer, 2010.

[3] https://turi.com/

[4] M. Ekstrand, J. Riedl and J. Konstan, *"Collaborative Filtering Recommendation Systems"*, in Journal Foundations and Trends in Human-Computer Interaction, vol. 4, pp 81-173, 2011.

[5] P. Han, B. Xie, F. Yang and R. Shen, *"A scalable P2P recommender system based on distributed collaborative filtering"*, in Expert Systems with Applications, vol. 27, pp 203-210, 2004.

[6] H. Liu, Z. Hu, A. Mian, H. Tian and X. Zhu, *"A new user similarity model to improve the accuracy of collaborative filtering"*, in Journal Knowledge-Based Systems, vol. 56, pp 156-166, 2014.

[7] R. Grove and E. Ozkan, *"The MVC-Web Desing Pattern"*, in Proceedings of the 7th International Conference on Web Information Systems and Technologies, 2011.

[8] C. Bizer, T. Heath and T. Berners-Lee, *"Linked Data – The Story So Far"*, in International Journal on Semantic Web and Information Systems, vol. 5, pp 1-22, 2009.

[9] C. Strauch, *"NoSQL Databases"*, Hochschule der Medien, Stuttgart, 2011.

[10] R. T. Fielding, *"Architectural Styles and the Design of Network-based Software Architectures"*, University of California, Irvine, 2000.

[11] *"MongoDB Documentation"*, 2017, [Online]. Available: https://docs.mongodb.com/, [Accessed: 13-Apr-2017].

[12] M. Lanthaler and C. Gutl, *"On using JSON-LD to create evolvable RESTful services"*, in Proceedings of the Third International Workshop on RESTful Design, pp 25-32, 2012.

[13] *"omdb 0.2.0"*, 2017, [Online]. Available: https://pypi.python.org/pypi/omdb/0.2.0, [Accessed: 13-Apr-2017].

[14] *"Angular Docs"*, 2017, [Online]. Available: https://angular.io/docs/ts/latest/, [Accessed: 13-Apr-2017].

[15] *"Django REST framework"*, 2017, [Online]. Available: http://www.django-rest-framework.org/, [Accessed: 13-Apr-2017].

[16] *"SQLite Documentation"*, 2017, [Online]. Available: https://www.sqlite.org/docs.html, [Accessed: 13-Apr-2017].

[17] *"Flask (A Python Microframework)"*, 2017, [Online]. Available: http://flask.pocoo.org/, [Accessed: 13-Apr-2017].