Extending RapidMiner with recommender systems algorithms

M. Mihelčić[1,3], N. Antulov-Fantulin[1], M. Bošnjak[1,2] and T. Šmuc[1]

[1] Ruđer Bošković Institute, Croatia

[2] Faculty of Engineering, University of Porto, Portugal

[3] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Netherlands

Abstract

Recommender systems are ubiquitous in today's information overloaded world. They help users to find and select products from a huge number available in various sources. RapidMiner can be used to construct various information filtering workflows using various data mining techniques. However, it does not explicitly support typical recommendation tasks. In this paper we present a RapidMiner Recommender Extension, developed in order to embed some of the state-of-the-art recommendation techniques into RapidMiner. We present the functionality of the extension, along with examples of diverse recommender system implementations. Integration of the extension with RapidAnalytics, which allows seamless construction of production level recommender systems, is demonstrated.

Keywords: recommendation systems, data mining, RapidMiner extension, e-LICO

1 Introduction

Whether buying technical equipment, books, watching movies, choosing restaurants or news to read, we often rely on recommendations from family, friends or even complete online strangers. We turn to recommendations whether there is too much or too little of information, or when the time cost of decision making is high. Recommender systems [9] facilitate this decision making through informed assistance and enhanced user experience by relying on the available information for the decision making process. If the data at hand is data on

items only, we are talking about content recommendation methods. If social information is available, i.e. interaction of other users with the items, we are talking about collaborative filtering methods. By combining both of these data sources, item data and social data, we can construct a hybrid system.

Recommender systems represent a broad field of research and were, from their start in nineties [5, 6, 8], somewhat detached from the data mining field. Collaborative filtering methods dealing with user-item usage information, were initially playing primary role in both research and real-world application of recommender systems. Requirements for real-world recommender systems are very similar to other contemporary data mining or machine learning tools in realistic settings: efficiency, scalability, ability to adapt models online, etc.

Rationale for introducing recommender systems into RapidMiner is multifold. RapidMiner has several important features that can be of advantage for solving recommendation problems and building efficient recommendation systems: (i) it provides a diverse set of operators for advanced data (pre)processing, (ii) one can easily optimize recommendation workflows by using operators for model optimization or by combining multiple models, and (iii) RapidMiner and RapidAnalytics provide seamless integration of models and data into real-world application setting. Though it is possible to create recommender systems in RapidMiner using only the standard operators [2], having a dedicated extension simplifies and speeds-up this procedure drastically.

We present the extension in the following sections, starting with the description of main operators available through the framework in Section 2. Section 3 specifies data formats used by the extension. Section 4 shows some applications of the extension, ranging from simple workflow construction, over building hybrid recommenders to exporting those workflows as web services. In section 5 we present part of the experimental results of iterative online updates and model application.

2 Recommender Extension operators

Recommender Extension operators are divided into two main categories: *Item Rating Prediction* operators (12 operators) and *Item Recommendation* operators (8 operators). Each of the categories are further divided in *Collaborative filtering based* operators and *Attribute based* operators. Each operator is a ported implementation of a specific algorithm from MyMedia library [4].

Collaborative based operators take as input an example set containing training data while returning a trained model and unchanged training data. *Attribute based* operators additionally take another example set containing attribute data. Operators return a trained attribute based model and unchanged training data. *Attribute based* operators use the attribute dataset to calculate

recommendations for users in the system, while the train data provide information about what items are already viewed by some user. Operator input data formats are explained in Section 3. Every operator has its unique parameters that enable fine tuning of its runtime and accuracy performance. We also created the *Apply Model* and the *Performance* operators for each category.

Rating prediction Apply Model operator takes a trained model and a test (query) set as input. The *Apply Model* operator applies the trained model on the query data and returns an example set containing rating predictions for each user-item pair in the query set. If the online update option is checked, the *Apply Model* operator will apply fast iterative model updates incorporating new query information into its input model. This feature is currently available for all operators except the *WRMF* that contains partial model retrain. We should also note that the *BMF* and the *BPRMF* operators contain online updates ported from MyMedia library. To perform online updates, query data must have a label role defined. *Apply Model* returns the updated or unchanged model, depending on the online update option, and an example set that contains the *prediction* attribute in addition to the attributes from a query set.

Output of the *Apply Model* operator is used to calculate performance using the *Performance* operator. The *Performance* operator calculates the value of rating prediction error measures: Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Normalized Mean Absolute Error (NMAE) [4]. These error measure values are returned as a performance vector and an example set.

Item Recommendation Apply Model operator also takes a trained model, and a test/query set as input. The *Apply Model* operator applies the trained model on the query data and returns the list of the first n ranked items for each user in the query set, where n is a user defined parameter. Online updates have the same functions as in the Rating Prediction category, for them to be applied query data must have a user-item pairs in the query set. *Item Recommendation Apply Model* output cannot be used for Performance calculation.

Item Recommendation Performance operator takes a training set, a test set and a trained model as input. Performance operator returns item recommendation error measures: the Area Under the Curve (AUC), Precision at k (prec@k), Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) [4].

Since we require only the n best items to be recommended for each user in a test set, we implemented a partial sorting algorithm into the *Apply Model* operator of the *Item Recommendation* category thus significantly decreasing application time.

Fast iterative online updates have been implemented for *Rating Prediction* and *Item Recommendation* operator. We built upon MyMedia library to support online update capability for *Item Recommendation* operators. Online

updates implemented for some *Rating Prediction* operators in the library recalculate all the information for every user in the query set constantly allocating new memory. That caused performance problems for larger query sets that we resolved using iterative online updates.

We constructed the weighted *Model Combiner* operator for each category. The *Model Combiner* operator takes as input multiple trained models and returns a weighted grouped model. When applying grouped model or performing online updates, operation is applied to each input model. The *Model Combiner* enables ensemble testing on various data examples and comparing results with each of the individual models. It also enables us creating various hybrid based models by combining collaborative filtering and attribute based operators. A full list of operators supported by the extension can be found in the extension user guide [7].

The extension is well integrated with RapidMiner so we can use existing operators to perform various standard operations like model saving[1] and parameter optimization[2]. Recommendation operators can be used along other RapidMiner operators in advanced experimentation tasks[3] and recommendation process, including online updates can be emulated by using a combination of standard RapidMiner operators and extension operators[4].

3 Data formats for the Recommender Extension

The *Item Recommendation* operators use a matrix that contains information about user viewing history in the system. The *Rating Prediction* operator matrix contains additional information: ratings describing users affinity towards particular items. Input datasets used to learn a recommender system model must be formatted in two columns for the *Item Recommendation*, or in three columns for the *Rating Prediction* problem. Attributes names and their positioning can be arbitrary. In the first column of Table 1 we have user IDs, in second column item IDs and in third column ratings. Prior to applying input datasets to recommendation operators, we have to set the roles for these columns, in our example: user identification, item identification, and label, respectively. An example of an `AML` and the related `DAT` file for rating prediction are given in Table 1.

The *Item Recommendation* operators do not require a rating attribute in the `AML` file, nor the third column of the `DAT` file, therefore any additional attributes will be ignored during execution of those operators.

[1] `http://www.myexperiment.org/workflows/2705.html`
[2] `http://www.myexperiment.org/workflows/2730.html`
[3] `http://www.myexperiment.org/packs/246.html`
[4] `http://www.myexperiment.org/packs/248.html`

Table 1: Rating Prediction input data example consisting of an AML and the related DAT file

AML file

```
rating_prediction.aml:
<?xml version="1.0" encoding="windows-1252"?>
<attributeset default_source="sample.dat">
  <attribute
    name         = "user_id"
    sourcecol    = "1"
    valuetype    = "integer"/>
  <attribute
    name         = "item_id"
    sourcecol    = "2"
    valuetype    = "integer"/>
  <attribute
    name         = "rating"
    sourcecol    = "3"
    valuetype    = "real"/>
</attributeset>
```

DAT file

```
sample.dat :
1 71 67.0
1 169 76.0
2 211 56.0
2 562 99.0
3 670 100.0
4 576 10.0
```

The data matrix for attribute based operators contains the user or item identification and the attribute identification. Concretely in our example, by changing the attribute *"item_id"* in Table 1 to *"attribute_id"* and removing the rating attribute and the third column from the corresponding DAT file, we would get an AML input file for attribute based operators.

The meaning of the DAT file in Table 1 in this case would be that the user with ID=1 contains attributes with ID=71, ID=169 and no other attributes. We developed a RapidMiner workflow to transform user/item description datasets into binomial form necessary for our attribute based operators[5].

4 Extension applications

In this section we point out some possible applications of the Recommender Extension.

4.1 Simple workflows

We start our survey by demonstrating and explaining some basic recommendation workflows. Figure 1 depicts such a basic model application workflow for the *Item Recommendation* category.

The train data and the query data are read from the AML file using the *Read AML* operators. We train the *Item k-NN* model on the training data,

[5] http://www.myexperiment.org/workflows/2711.html

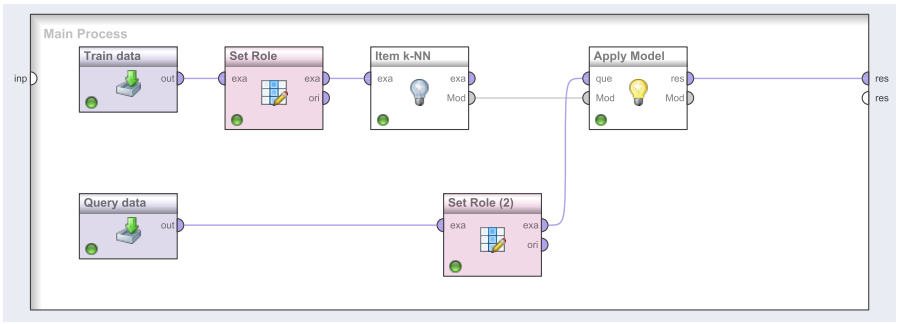

Figure 1: Item recommendation model application workflow

and pass it to the *Apply Model* operator that applies it to the query data. To calculate the performance of our model, instead of applying the trained model on the query set, we test our trained model on the test set using the *Performance* operator.

The performance calculation workflow for the *Rating Prediction* category is shown in Figure 2. By simply omitting the *Performance* operator, we obtain the model application workflow.

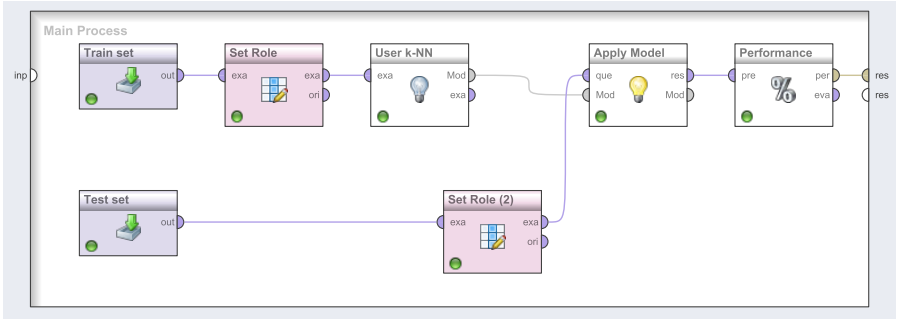

Figure 2: Rating Prediction performance calculation workflow

4.2 Building hybrid solutions

Hybrid approach to recommendation systems consists of combining different recommendation techniques [3] in order to achieve better performance and circumvent limitations of individual techniques [1].

Building hybrid recommendation systems using the Recommender Extension is easy. User can combine multiple learned recommendation models using

the *Model Combiner* operator. That allows using and combining specific features of different techniques in order to achieve better recommendation performance. Depending on the input information, user can add weights to each trained model in combined model to change the significance that model has on the overall recommendation.

We demonstrate results of combining three different recommendation models. The *Weighted Item k-NN* and the *User k-NN* model are a neighbourhood based recommendation models that try to find users with a similar preferences as the user to whom we make some recommendation. The *Weighted Matrix Factorization* is a matrix factorization based recommendation model that uses a set of factors combined with collaborative information to make recommendations for some user. By using the *Model Combiner*, we increase the performance of each of the individual models. Results gained on a recommendation problem with 5000 users is demonstrated in Table 2.

Table 2: Performance comparison of the combiner operator with the following weights: WRMF 0.1, User k-NN 0.2, Weighter Item k-NN 0.7

	WRMF	User k-NN	Item k-NN	Combiner
AUC	0.855	0.915	**0.940**	**0.940**
prec@5	0.118	0.129	0.137	**0.138**
prec@10	0.083	0.091	0.098	**0.099**
prec@15	0.066	0.072	0.076	**0.077**
NDCG	0.416	0.436	0.452	**0.455**
MAP	0.229	0.245	0.261	**0.264**

Combiner weights for this example were manually chosen and could be optimized further, train set used has enough data to efficiently train *Item k-NN* model, so it is hard to further increase performance using only collaborative filtering models. In addition *WRMF* operator parameters could be further optimized for better performance. Despite that, probably it is not efficient to use the *Model Combiner* for such datasets.

A potential benefit could be to combine collaborative operators with attribute based operators incorporated in the extension to improve performance when dealing with little collaborative information. Since we lack real datasets that contain attribute data, we could test model combination only on artificially created attribute datasets. Performance comparison of *Item k-NN* model, trained on a dataset with little collaborative information, and combination of that model with *Item attribute k-NN* model gave AUC: $0.621 \mapsto 0.700$, prec@5: $0.066 \mapsto 0.073$, NDCG: $0.316 \mapsto 0.329$ and MAP: $0.090 \mapsto 0.099$.

It is also possible to combine Recommender extension models with RapidMiner regression or classification models into a large hybrid recommendation system[6].

[6] `http://www.myexperiment.org/packs/262.html`

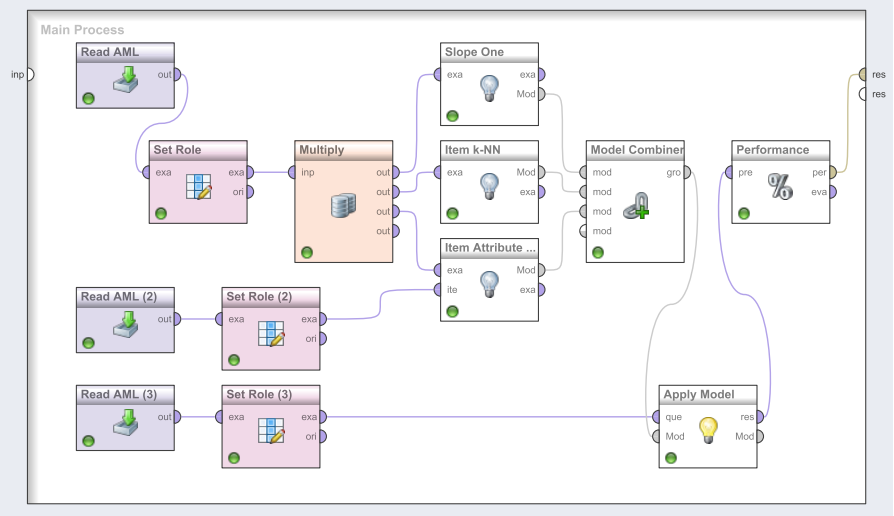

Figure 3: Hybrid rating prediction workflow

4.3 Exporting recommenders using RapidAnalytics

In order to build a recommendation web service for serving a website in real time, we need to assure response time in milliseconds. Therefore, we need a more complex architecture of recommender web services. Our recommender system web engine architecture, depicted in Figure 4, consists of the following parts in RapidAnalytics server: front-end recommendation, write activity, online update recommendation and offline update recommendation web service.

These services communicate with the RapidMiner repository on RapidAnalytics and with the SQL database. To ensure low recommendation response time, we cache the top-n recommendations for each user in the SQL database. Our SQL database has only two tables: item recommendation table (with columns *"user_id", "item_id", "rank"*), and train set table (with columns *"user_id", "item_id"*). The job of the front-end recommendation web service[7] is to query the cached recommendations from the *item recommendation table*. This web service has around 100 millisecond response time. When a web page needs a recommendation for a specific user, it calls the front-end recommendation web service. When a specific user i consumes a certain item j, the write activity web service[8] is invoked. This web service writes the activity (i, j) to the *train set table* and removes the recommendation j for the user i from the *item recommendation table* in the SQL database.

[7] `http://www.myexperiment.org/workflows/2904.html`
[8] `http://www.myexperiment.org/workflows/2906.html`

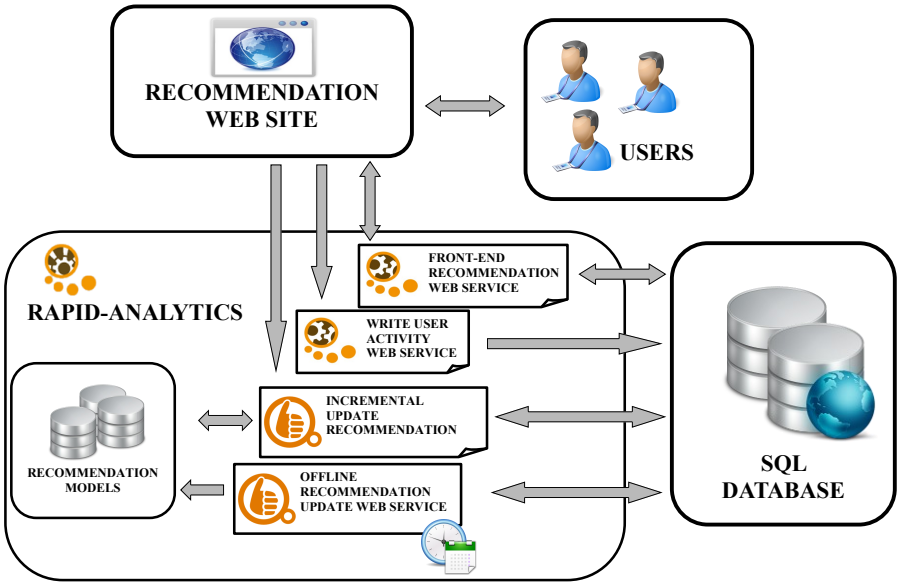

Figure 4: Architecture of recommendation web engine

After a number of recommendations to a specific users, system has to update recommendations in the *item recommendation table*. This is accomplished by calling the online update recommendation web service[9], which updates the recommendation model in RapidAnalytics repository and updates the recommendations for specific users in the *item recommendation table*. The online update procedure updates recommendations only for users for which some recommendation and feedbacks have been obtained. Therefore, we have to periodically do a full re-training on the whole train set with the offline update recommendation web service[10].

5 Results

We tested the iterative online updates both in accuracy and time execution performance. We give results for the *Item k-NN* operator in Figure 5. The x-axis denotes the number of cumulative iterative online updates performed on various query datasets before applying them on a test set, while the y-axis holds the performance result. We must note that in general we expect model retrain to outperform iterative online updates.

[9] `http://www.myexperiment.org/workflows/2955.html`
[10] `http://www.myexperiment.org/workflows/2954.html`

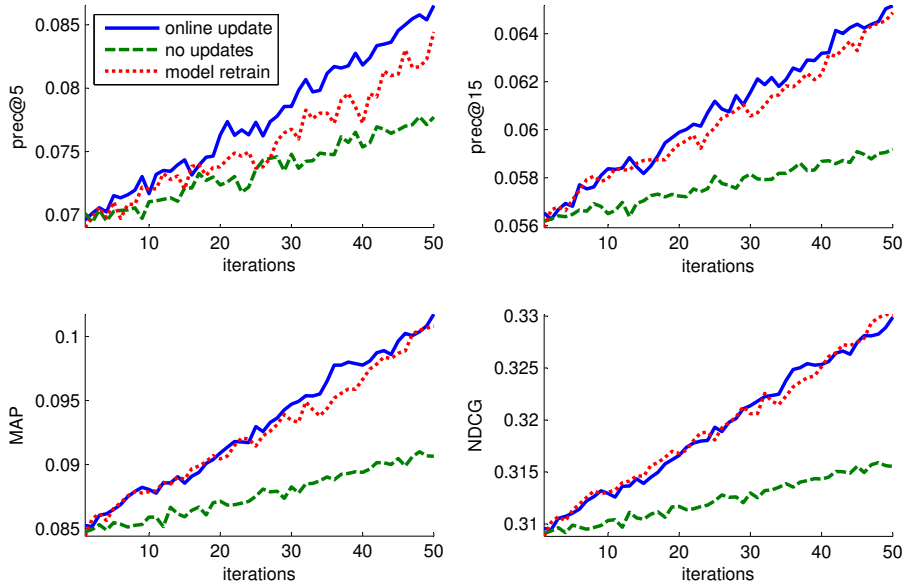

Figure 5: `prec@5`, `prec@15`, `MAP` and `NDCG` performance of iterative online updates on the *Item k-NN* operator.

In Figure 6 we give time performance for the *Item k-NN* operator on two different train datasets. The first dataset contains 395.497, while the second contains 146.741 user-item pairs. Our results show comparable performance in accuracy, while gaining time performance when comparing iterative online model updates with full model retrain. By formal analysis of the problem it can be shown that the absolute error between iterative online update and model retrain correlation value converges to zero when the problem size rises and it is to be expected that the time performance gain should increase as well, however that is one point for a further study.

Item recommendation operators within the MyMedia library [4] use a sorting algorithm to find the top-k items for a specific user. For this extension we developed a small algorithm for ranking the top-k items without the need to use a sorting algorithm on the whole item set predictions. For example, if we need to recommend k items with the highest score, the straightforward way is to sort the vector of n items scores in the descending order and choose the first k items. Our algorithm uses the min priority queue to maintain k items with the highest score throughout the vector score iteration. The approximate number of operations for the straightforward way is $n * log(n)$ and the approximate number of operations for our partial ranking algorithm is $n + c * log(k)$, where the constant c represents number of insert and delete operations in the

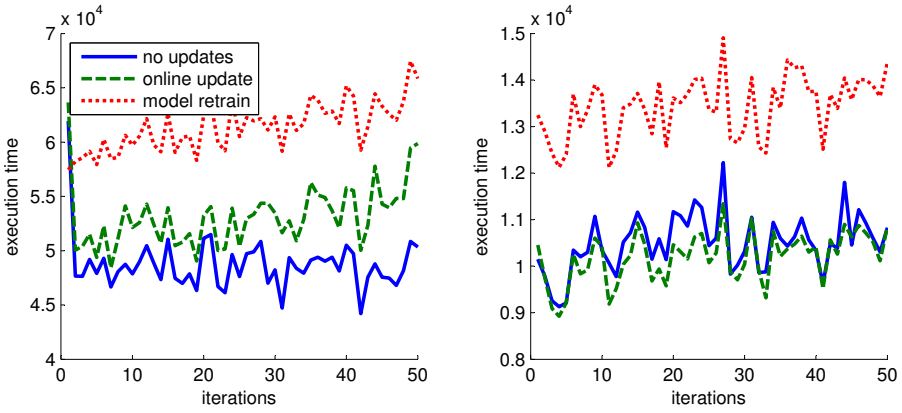

Figure 6: Time performance of the *Item k-NN* operator on two datasets.

priority queue. The expected number of insert and delete operations in the priority queue turns out to be much less than the total number of items n. Our experiments in Table 3 show improvement in the execution time.

Table 3: Execution time speed up as the ratio of execution times for apply model operator with the partial ranking algorithm and with the normal sorting algorithm

Item recommendation model	No. of users	No. of items	Execution time speed up (ratio)
WRMF	10 000	4 775	4,59
BPRMF	10 000	4 775	5.07
WRMF	20 000	3 963	5,73
BPRMF	20 000	3 963	4.74
WRMF	49 999	22 745	6,84
BPRMF	49 999	22 745	2.21
WRMF	199 999	11 083	7,08

6 Summary

In this paper we presented the RapidMiner Recommender Extension. We showed various applications of the extension, and various ways of incorporating its operators in RapidMiner. The applications shown range from simple workflows to a real time application of the extension in a form of a recommendation service run on RapidAnalytics. This system can be used and set up by

the users and would provide them with many customizations based on their requirements.

7 Acknowledgement

This work is supported by the European Community 7th framework ICT-2007.4 (No 231519) "e-LICO: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science".

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[2] M. Bošnjak, N. Antulov-Fantulin, T. Šmuc, and D. Gamberger. Constructing recommender systems workflow templates in rapidminer. *Proc. of the 2nd RapidMiner Community Meeting and Conference*, pages 101–112, 2011.

[3] R. Burke. The adaptive web. chapter Hybrid web recommender systems, pages 377–408. 2007.

[4] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: a free recommender system library. In *Proc. of the fifth ACM conference on Recommender systems*, RecSys '11, pages 305–308, 2011.

[5] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[6] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 194–201, 1995.

[7] M. Mihelčić, N. Antulov-Fantulin, and T. Šmuc. *Rapid Miner Recommender Extension - user guide*, 2011.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of the ACM conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, 1994.

[9] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, Mar. 1997.