The Role of Negative Results for Choosing an Evaluation Approach – A Recommender Systems Case Study

Benjamin Heitmann and Conor Hayes

INSIGHT @ NUI Galway
National University of Ireland, Galway
Galway, Ireland
firstname.lastname@insight-centre.org

Abstract. We describe a case study, which shows how important negative results are in uncovering biased evaluation methodologies. Our research question is how to compare a recommender algorithm that uses an RDF graph to a recommendation algorithm that uses rating data. Our case study uses DBpedia 3.8 and the MovieLens 100k data set. We show that the most popular evaluation protocol in the recommender systems literature is biased towards evaluating collaborative filtering (CF) algorithms, as it uses the "rating prediction" task. Based on the negative results of this first experiment, we find an alternative evaluation task, the "top-k recommendation" task. While this task is harder to perform, our positive results show that it is a much better fit, which is not biased towards either CF or our graph-based algorithm. The second set of results are statistically significant (Wilcoxon rank sum test, $p < 0.01$).

1 Introduction

Personalisation has come to be an expected feature for many user facing web sites. This includes e-commerce sites such as Amazon [1], media discovery sites like Netflix [2], as well as social networking sites like Facebook.

Linked Open Data (LOD) and the Semantic Web strive to enable "smarter" and personalised user experiences, as described by Tim Berners-Lee et al. in the original vision statement [3]. Therefore, exploiting RDF and Linked Data for recommender systems has naturally become an active area of research. Notable examples include e.g. the work of Middleton et al. [4], Passant [5] and Di Noia et al. [6].

However, personalisation approaches based on RDF data need to be compared to the vast body of existing research on personalisation. Recommender systems have been a widely explored research topic since the early 90's, when the GroupLens project [7] produced one of the first recommender systems in order to filter net news from the usenet.

More importantly, in order to be comparable to the state of the art, RDF-based recommendation algorithms have to be compared to the best performing, existing recommendation algorithms. This requires adapting existing gold-standard data sets and evaluation protocols, in order to answer the following research question:

Research question: How can a recommendation algorithm that uses an RDF graph be compared to a recommendation algorithm that uses rating data?

In this paper, we present a case study which shows how important negative results are in uncovering biased evaluation methodologies. As we will show, the most well established evaluation methodology in the area of recommender systems is inherently biased towards a very specific and narrow evaluation task called "rating prediction". This makes it very difficult to compare novel recommendation algorithms to the state of the art, if the novel algorithm can be used for personalisation but not for performing this very narrowly defined evaluation task.

As part of this paper, we analyse the prevalent evaluation approach for evaluating recommender algorithms and we present the negative results of our evaluation. These results showed that the so-called "rating prediction task" cannot be used for evaluating our personalisation approach. We follow this by describing an alternative experiment protocol using the "top-k recommendation" task and our positive results. In addition, we discuss why our chosen evaluation task is harder to perform correctly than the established "rating prediction task".

The paper is structured as follows: Section 2 introduces background in the area of recommender algorithms. We also analyse the two most common evaluation tasks and compare their difficulty. Section 3 provides a high-level overview of our SemStim algorithm. Section 4 lists the data sets we used for our evaluations, while Section 5 lists the recommendation algorithms used for comparison with SemStim. Section 6 describes the failed evaluation using the rating prediction task. Then Section 7 describes the evaluation protocol and results of the successful evaluation using the top-k recommendation task. We conclude the paper in Section 8 and describe the lessons we learned from the two experiments.

2 Background and related work

In this Section we introduce the background which is required to understand how recommender algorithms are evaluated and compared. We first introduce the basic components of all recommender systems. Then we describe the three most important types of recommender algorithms, their data requirements and their limitations. Then we list notable related work on RDF-based recommender systems. This is followed by our analysis of the two main evaluation protocols and their accompanying performance metrics for evaluating recommender algorithms. We close this section with a comparison of the difficulty of these two evaluation protocols.

2.1 Recommender Systems

Personalised recommendations have proven themselves to greatly enhance the user experience of searching, exploring and finding new and interesting content. In the domain of e-commerce, recommender systems have the potential to support and improve the quality of customer decisions by reducing the information overload facing consumers as well as the complexity of online searches [8]. In

addition, e-commerce web sites benefit from recommender systems by converting browsers into buyers, increasing the average order size and improving consumer loyalty[9].

Recommender systems require three components to provide recommendations [10]: (1) *background data* which is the information the system has before the recommendation process begins, (2) *input data*, which is the information provided about the user in order to make a recommendation, and (3) the *recommendation algorithm* which operates on background and input data in order to provide recommendations for a user. The recommendation algorithm families which are most frequently used are (1) collaborative filtering, (2) content-based filtering and (3) hybrid filtering approaches, according to [11].

Collaborative filtering (CF) aggregates feedback for items from different users and uses similarities between users or items to provide recommendations. The input data consists of a user profile providing ratings for one or more items. CF uses the background data to calculate the pair-wise similarity between all items or all users, and then uses the input data to recommend similar users or items. CF algorithms have reached a high maturity and are currently used by most major e-commerce vendors. The *main limitation* is the cold-start problem, as providing recommendations without ratings is not possible for new users, new items or very sparse rating matrices.

Content-based filtering (CBF) uses features of the items as the background data for the recommendation. These can either be directly derived from the content, e.g. keywords from text or tempo of the music piece, or derived from the meta-data of the items, e.g. author, title and genre. The input data needs to describe the user preferences in terms of content features. The *main limitation* is that the data about content features needs to be of very high quality (e.g. error-free and consistent) in order to provide good recommendations.

Hybrid filtering algorithms (Hybrid) combine two or more recommendation algorithms to provide better results with fewer of the drawbacks of an individual algorithm. This can be done e.g. by combining the scores of several algorithms with weights or chaining algorithms into a cascade [10].

Most commonly, CF is combined with a content-based algorithm in order to mitigate the shortcomings of both approaches. As an example, both Netflix [2] and Amazon.com [1] use CF and several content-based algorithms as part of their personalisation approach.

2.2 Related work on RDF-based recommender systems

We use Linked Data from DBpedia for personalisation. Previous research on using Semantic Web technologies for single-domain personalisation includes Middleton et al. [4], who introduce the concept of ontology-based user profiling and personalisation with the use case of scientific literature search. Passant [5] presents a music recommender system which uses Linked Data from DBpedia for its graph-based similarity measure called Linked Data Semantic Distance (LDSD).

2.3 Experiment protocols for evaluating recommender systems

In the absence of an online experiment with users, an offline evaluation of a recommender system, is always an approximate model of the user's response to the recommendation algorithm [12]. As with any model, it has implicit biases.

Most early recommendation research tended to view offline evaluation as just a *rating prediction task* or a classification task. The goal was to fill in the missing values in a sparsely filled user-item rating matrix. Thus, the main evaluation measure has been based on measures of rating prediction error such as Mean Average Error (MAE) and Round Mean Square Error (RMSE).

While the rating prediction task is still a core evaluation task in recommender research, it has been supplemented by the *top-k recommendation task* [13], which involves ranking a subset of recommendable items (e.g. from the test user profile) to produce a result set of the top-k items predicted to be of interest to the user. Typical measures are precision@k, recall@k and F1@k. The goal of the system is to suggest k items of interest, rather than predict ratings for all M unrated items in the inventory of the recommender system, with $k \ll M$.

While these two evaluation tasks provide different perspectives, both are still based on the notion of prediction of item rating, and in the case of the top-k recommendation task, ranking based on those predicted ratings.

The top-k recommendation task is the more universal task: Any algorithm capable of predicting item ratings can be used to rank those items, so both evaluation tasks are applicable. However, there are algorithms which only rank items without predicting ratings. For those algorithms, only the top-k recommendation task is applicable.

Any new algorithm that cannot be tested using variants of these two approaches will struggle to be accepted by the RecSys community. Furthermore, any algorithm whose computational model is not based on rating prediction may also struggle to be compared with baseline approaches that produce rating predictions.

2.4 Comparison of difficulty between evaluation tasks

For both evaluation tasks, each user profile is split into a training part and a test part. Then the training part is used to train the recommender. This is where the similarities end.

For the rating prediction task, a small number of items is withheld from the training profile to form the test profile. This is called the "hold-out", and it is usually very small. Common values for the hold-out range from 1 to 5. A hold-out of 10 is already considered to be big, c.f. [14]. The recommender system then has the task to compute ratings for all items in the test profile, without knowing the ground truth. However, the items for which to compute the rating prediction are provided.

For the top-k recommendation task, the recommender algorithm has to provide a list of recommendations selecting from all unrated items in the system. The algorithm does not know anything about the items in the test profile. In addition, usual values for the hold-out are provided as a percentage of the full user profile,

with values between 10% and 20% [12]. The overlap between recommendations and test profile is then used to compute precision and recall.

In comparison, the top-k recommendation task is much more difficult to perform than the rating prediction task for the same dataset. The recommender system has a smaller amount of ratings in the training profile, as the hold-out is bigger for the top-k recommendation task. In addition, the recommender has no data from the test profile for the top-k recommendation task. In comparison, for the rating prediction task, all the items in the test profile are known to the algorithm, only their rating is withheld. As a result, the magnitude of e.g. precision and recall will generally be lower for the same algorithm and the same data set when using the top-k recommendation task, as compared to the rating prediction task.

3 High-level overview of the SemStim algorithm

Our SemStim algorithm is an enhanced version of spreading activation (SA) as described by Crestani in [15]. SA enables finding related entities in a semantic network in a fuzzy way without using reasoning, while still being deterministic and taking the semantics of the network into account. SemStim extends basic SA, by adding (1) *targeted activation*, which allows us to describe the target item set of the personalisation, and (2) *constraints* to control the algorithm *duration*. Due to the space constraints of this paper, we will only describe the intuition of the algorithm. The formal description of the algorithm can be found in [16], Chapter 4.

Algorithm intuition SemStim is an iterative algorithm. For SemStim, a *domain* is any set of items represented by vertices in a semantic network, such as an RDF graph. The *input* of the algorithm are the vertices in the user profile from the source domain. Each of these vertices will spread an amount of activation to his direct neighbors in the very first iteration. Whenever an unactivated vertex has accumulated more activation then the global threshold, it becomes activated. In each subsequent iteration, all vertices which became activated in the current iteration spread an amount of activation to their neighbours. This in turn might lead to other vertices becoming newly activated, which will spread activation in the following iteration.

When target number of vertices from the target domain have become activated, the algorithm terminates. The *output* of the algorithm, are the activated vertices from the target domain. They are returned as recommendations, ranked by their activation level. Vertices which are not from the target or source domain, can become activated as well. These vertices allow traversing the parts of the graph which indirectly connect the source and target domain.

4 Data sets

In both our experiments we use rating data from MovieLens to provide the training and test profiles, as well as DBpedia 3.8 to provide the background knowledge for the SemStim algorithm.

Rating data from MovieLens: For the single-domain accuracy experiment, we use the 100k ratings MovieLens[1] data set. It contains 100,000 ratings between 1 and 5, from 943 users on 1,682 movies, where each user has rated at least 20 movies. The links between MovieLens and DBpedia were created automatically by string matching. In cases without a match, it was manually added. This has resulted in 98% coverage (37 movies missing out of 1,682).

Background knowledge from DBpedia: In order to provide recommendations with SemStim, we use a subset of DBpedia 3.8 with 67 million edges and 11 million vertices, which includes all article categories, disambiguation links, instance types, mapping-based properties, transitive redirects, categories, and entity types. This represents all available data on edges between vertices on DBpedia. The data contains 72,000 movies.

5 Algorithms for comparison

We compare SemStim to two collaborative filtering algorithms, as well as two graph-based algorithms.

Collaborative filtering (CFknn50): We implemented a k-nearest neighbor user-based collaborative filtering algorithm from Herlocker et al. [14] with neighborhood size of 50. Cosine distance is used to calculate the pairwise similarity between users. For the rating prediction, we use the weighted average of deviations from the mean item rating in the neighbourhood of the active user. We used the same parameters as described in Herlocker et al. [14]. We verified the correctness of our implementation, by replicating the results published by Herlocker et al. [14] and by Cremonesi et al. [13].

SVD++ is a state of the art collaborative filtering algorithm, which is based on matrix factorisation. It is first described by Koren et al. [17]. We use the implementation of SVD++ which is provided by the GraphLab collaborative filtering toolkit[2]. GraphLab was originally developed at Carnegie Mellon University. We use the same parameters for SVD++ as in Cremonesi et al. [13], e.g. we use SVD++ with 200 latent factors.

Random movie selection (Random) This is a worst-case baseline. It recommends a set of random movies, which are unknown to the user.

Linked Data Semantic Distance (LDSD) LDSD is a state-of-the-art semantic similarity, presented by Passant [5]. LDSD provides a distance metric between entities of a semantic graph. It is motivated by the assumption that two resources are more related if they are the only ones sharing a particular property. Given a user profile with start nodes, LDSD returns the top-k most similar nodes, as determined by counting the number of direct and indirect links which are shared between each of the start nodes, and any 1st or 2nd degree neighbors from the target domain.

Set-based breadth first search (SetBFS) SetBFS is a baseline graph algorithm. It is a modified breadth first search, which starts from the set of nodes

[1] `http://www.grouplens.org/node/73`

[2] `http://docs.graphlab.org/toolkits.html`

in the user profile, and stops when any k nodes from the target domain have been found.

6 Evaluation using the rating prediction task

As stated in the introduction, our main research question, is how to compare SemStim to state of the art recommender algorithms, such as CF. Therefore we first evaluated the SemStim algorithm using the rating prediction task.

6.1 Experiment protocol

We use the following experiment protocol for our rating prediction experiment:

Type of experiment: Off-line experiment without user involvement.
Data set: MovieLens 100k ratings data set, with user profiles in which movies are associated with ratings.
Split of train-/test-data: We used a hold-out of 10 randomly selected items, so 10 items are in the test profile, the rest of the items go into the training profile.
N-fold cross-validation: If the split between train- and test-data is performed randomly, then **n** splits can be done in order to run the experiment n times. We used 5-fold cross validation.

6.2 Performance metrics

In the test phase, usually the following performance metrics are used to compare the ratings of the recommendation algorithm to the ratings in the ground truth:

Root-mean-square error (RMSE) Given the vector of original ratings and the vector of predictions, RMSE is the square root of the average squared distance between the rating and prediction for each item of the test data.
Mean absolute error (MAE) Given the vector of original ratings and the vector of predictions, MAE is the average of the absolute difference between the rating and prediction for each item from the test data.
Normalised discounted cumulative gain (nDCG) compares the ranking of the recommendation algorithm to the original ranking. It applies a heavier penalty to wrongly high ranked items, thus reflection the tendency of users to react negatively to irrelevant results at the top of a recommendation list. As the results are normalised, results can be compared between recommendation lists of different lengths and between results of different algorithms.

However, in order to compare CF and SemStim, we can only make use of nDCG. SemStim effectively performs a graph-search on a semantic network. SemStim starts with the items in the user profile and terminates when it has found the required number of items in the target domain. Due to this mode of operation, SemStim is not very well suited for the rating prediction task, as this basically requires starting from the items in the test profile. In addition, SemStim does not take ratings into account. However, an overlap between the recommendations and the test profile can be determined. So while MAE and RMSE are not well suited to evaluate SemStim, we can use nDCG to compare CF and SemStim.

6.3 Results

	nDCG
SemStim	9.5%
Collaborative Filtering	94%

The results show that SemStim performs significantly worse than CF. The performance of SemStim is so low compared to CF, that one possible interpretation is that the rating prediction evaluation task is a bad mismatch for the SemStim approach. In other words, it is possible that the popular evaluation protocol using the rating prediction task is biased towards CF, so that algorithms which are not optimised for the rating prediction task will have a worse performance when compared to CF.

7 Evaluation using the top-k recommendation task

The first experiment revealed that an evaluation using the rating prediction task is a mismatch for our proposed algorithm. So we went looking for another experiment protocol which would allow us to compare these two algorithms on equal terms, without any inherent bias for one of the two approaches.

We have adapted our experiment protocol for the top-k recommendation task from Cremonesi et al. [13], which is the most cited evaluation protocol based on this task. Cremonesi et al. propose an experiment protocol that allows using accuracy metrics such as precision and recall for evaluating rating predictions. The main idea is to form a test profile for each user by random sampling of the most positive ratings of the user, and then mixing them with a large number of random unrated movies. In this test profile, the originally highly ranked items are labelled as belonging to the "positive" class, while the random unrated movies are labelled as belonging to the "negative" class. The remaining part of the original user profile forms the training profile.

Labelling the items in the test profile allows evaluating the performance of each recommendation algorithm in the same way as a *binary classifier*. Each recommendation algorithm ranks all of the items in the test profile. The parameter of k determines the cut-off point for the ranked list of recommendations. The top-k items in the ranked list of recommendations are then used to determine the number of "true positives". E.g. when evaluating the top-15 recommendations for a specific user and algorithm, if the first 15 items which are recommended include 3 items labelled as "positive", then the algorithm has recommended 3 "true positives" and 12 "false positives".

Please note, that the classification labels for the items in the test profile are always withheld from the recommendation algorithms.

7.1 Experiment protocol

For our specific experiment, we generate the training and test profile from the full MovieLens 100k rating data set as follows: First we sample 10% of each MovieLens user profile into a probe set P. The remaining 90% of the data set form the training profile M. We then construct the test profile T as follows:

1. We take all highly rated items (all 4 and 5 star ratings) from the probe set P of a user, labeled as belonging to the positive class. The mean number of highly rated items per user is 6.48.
2. Then we add random, unrated items for each user, labeled as belonging to the negative class, so that each user profile in P contains 200 items. The resulting user profiles contain 96% items from the negative class on average. Cremonesi suggests a sample size of 1.4% for the probe set, however this resulted in just 2 highly rated items per user profile on average, so we increased the sample size to 10%, which results in 6.48 highly rated items per user on average.

We then generate a ranked list of recommendations for each of the different algorithms.

It is important to note that all algorithms have access to the same training data. However, while CF uses positive *and* negative ratings to find the neighbourhood of a user, the graph algorithms are each essentially implementing a similarity based approach to find similar items. Therefore the graph algorithms only use positive preferences to make recommendations.

We generate a ranked list of recommendations for each of the different algorithms as follows:

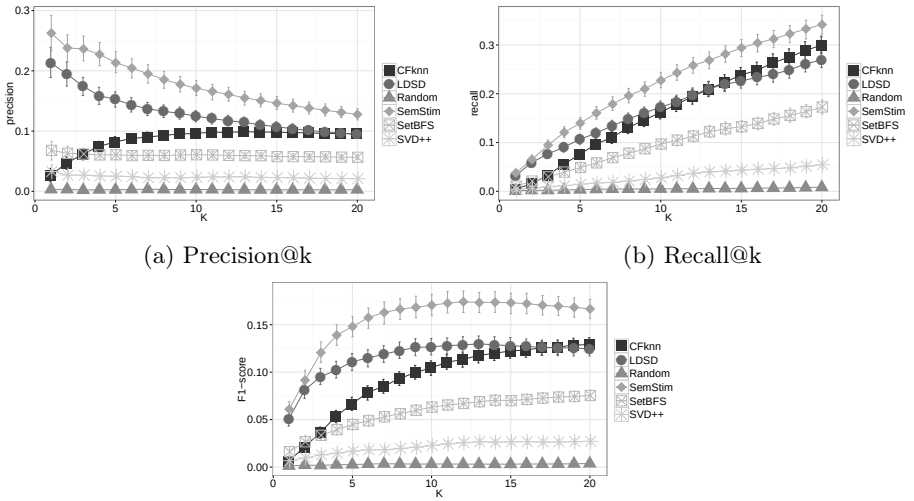
SemStim: The DBpedia URIs of all positively rated movies (4 and 5 stars) in the training profile M of the active user are used as the set P of start nodes for SemStim. The target domain D is the set of DBpedia URIs for all movies in the test profile T of the active user. The target domain only includes items from the test profile of the active user. When the algorithm terminates, we rank the activated nodes from the test profile by their activation value in descending order and return the ranked list.

Linked Data Semantic Distance (LDSD): Given the training profile M and the test profile T for the active user, we determine the minimum LDSD distance for all positively rated items from the test profile to any item in the training profile: $\forall t_j \in T: \ LDSD(t_j) = \min_{m_i \in M} LDSD_c(m_i, t_j)$. We then rank all the items from the test profile by their smallest distance to the training profile, in ascending order, and return this ranked list.

Set-based Breadth First Search (SetBFS): For each user, SetBFS starts from the positively rated items in the training profile M of the active user, and terminates when 20 nodes from the test profile T have been found in the DBpedia network. The nodes are then ranked in the same order in which SetBFS found them in the network, and the ranked list is returned.

SVD++ and Collaborative Filtering (CFknn): First a model is generated for both algorithms, by using all ratings in the training profiles of all users in the data set. Then for each user, ratings are predicted for each item in the test profile of the active user. The test profile items are then ranked in descending order by their predicted rating, and one ranked list for each of both algorithms is returned.

Random selection (Random): The Random algorithm returns 20 random items from the test profile of the active user.



(a) Precision@k

(b) Recall@k

(c) F1-score@k

Fig. 1: Accuracy results for the single-domain top-k recommendation experiment, which is described in Section 7. MovieLens 100k rating data is used as source and target domain. Error bars show the 95% confidence interval.

7.2 Performance metrics

We determine *precision*, *recall* and *F1-score* for each algorithm and each user, by determining the number of true positives in each set of recommendations using the withheld classification labels. #true-positives@k refers to the number of items labelled as belonging to the "positive" class, among the first k items of the ranked list of recommendations. #true-positives-in-user-test-profile refers to the total number of items in the test profile of the active user, which belong to the "positive" class.

$$\text{precision@k} = \frac{\#\text{true-positives@k}}{k} \tag{1}$$

$$\text{recall@k} = \frac{\#\text{true-positives@k}}{\#\text{true-positives-in-user-test-profile}} \tag{2}$$

$$\text{F1-score@k} = 2\frac{\#\text{precision@k} \cdot \#\text{recall@k}}{\#\text{precision@k} + \#\text{recall@k}} \tag{3}$$

7.3 Results

The results in Figure 1 show that when using the top-k recommendation task on MovieLens data, SemStim outperforms the other algorithms. SemStim can provide top-k recommendations, which are more accurate than the results of the baseline algorithms, including collaborative filtering. SemStim dominates the performance results for all three metrics of precision, recall and F1-score.

However, the differences between the performance of SemStim and the CF algorithms are not too big, which suggests that the evaluation task is a good match for both algorithm types, without being biased for either of both approaches.

The results for SemStim are statistically significant for values of $k > 5$ (Wilcoxon rank sum test, $p < 0.01$). All algorithms show a trend of decreasing precision with growing k, except for CFknn. We believe that the increasing precision@k of CFknn is due to the low average number of true positives in the test profiles of the users. Our experiment protocol adds a large number of random items to the test profiles (up to 200 items).

8 Conclusions

In this paper we have presented a case study which shows how important negative results are in choosing an evaluation approach.

We learned that the performance of an algorithm for one evaluation task does not have to correlate with the performance of the same algorithm for another evaluation task. Notably, Cremonesi at al. in [13] found that algorithms such as CF which are optimised for minimising RMSE do not necessarily perform as expected in terms of the top-k recommendation task.

We also learned that sometimes a research community can request comparing an apple to an orange. In our case, we had to compare our approach to the CF algorithm as a baseline, because all other recommender algorithms have been compared to CF. We ultimately managed to compare the two approaches, however we had to use an evaluation task which is harder to perform than the established rating prediction task, as the magnitude of e.g. precision and recall is smaller than for the same algorithm and same data set when using the rating prediction task (if the rating prediction task is applicable to the algorithm).

In summary, the negative results in the first experiment were very important, as they suggested a mismatch between the most established evaluation task and the proposed algorithm. This in turn, lead us to find a better evaluation protocol and evaluation task.

Acknowledgements: This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 (Insight).

References

1. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet computing **7**(1) (2003) 76–80
2. Amatriain, X., Basilico, J.: Netflix Recommendations: Beyond the 5 stars (Part 2). The official Netflix Tech Blog (2012)
3. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Scientific american **284**(5) (2001) 28–37
4. Middleton, S., Shadbolt, N., De Roure, D.: Ontological user profiling in recommender systems. ACM Transactions on Information Systems (TOIS) **22**(1) (2004) 54–88

5. Passant, A.: dbrec – music recommendations using dbpedia. The Semantic Web–ISWC 2010 (2010) 209–224
6. Di Noia, T., Mirizzi, R., Ostuni, V.C., Romito, D., Zanker, M.: Linked open data to support content-based recommender systems. In: I-Semantics. (2012)
7. 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, ACM (1994) 175–186
8. Xiao, B., Benbasat, I.: E-commerce product recommendation agents: Use, characteristics, and impact. Management Information Systems Quarterly **31**(1) (2007) 137
9. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The adaptive web. Springer (2007) 291–324
10. Burke, R.: Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction **12**(4) (2002) 331–370
11. Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A.: Recommender systems survey. Knowledge-Based Systems (2013)
12. Shani, G., Gunawardana, A.: Evaluating recommendation systems. In: Recommender systems handbook. Springer (2011) 257–297
13. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: ACM Conference on Recommender Systems. (2010) 39–46
14. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: ACM SIGIR conference. (1999) 230–237
15. Crestani, F.: Application of spreading activation techniques in information retrieval. Artificial Intelligence Review **11**(6) (1997) 453–482
16. Heitmann, B.: An open framework for multi-source, cross-domain personalisation with semantic interest graphs. PhD thesis, National University of Ireland, Galway (2014)
17. 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, ACM (2008) 426–434