Transfer Learning for Content-Based Recommender Systems Using Tree Matching

Naseem Biadsy[1], Lior Rokach[2], and Armin Shmilovici[2]

[1] Deutsche Telekom Labs at Ben-Gurion University
Beer-Sheva 84105, Israel
naseem@cs.bgu.ac.il
[2] Department of Information System Engineering Ben-Gurion University
Beer-Sheva 84105, Israel {liorrk,armin}@bgu.ac.il

Abstract. In this paper we present a new approach to content-based transfer learning for solving the data sparsity problem in cases when the users' preferences in the target domain are either scarce or unavailable, but the necessary information for the preferences exists in another domain. Training a system to use such information across domains is shown to produce better performance. Specifically, we represent users' behavior patterns based on topological graph structures. Each behavior pattern represents the behavior of a set of users, when the users' behavior is defined as the items they rated and the items' rating values. In the next step, a correlation is found between behavior patterns in the source domain and target domain. This mapping is considered a bridge between the two. Based on the correlation and content-attributes of the items, a machine learning model is trained to predict users' ratings in the target domain. When our approach is compared to the popularity approach and KNN-cross-domain on a real world dataset, the results show that our approach outperforms both methods on an average of 83%.

Keywords: Recommender-Systems, Transfer Learning, Content-based, Behavior Patterns

1 Introduction

Cross domain recommenders [2] aim to improve recommendation in one domain (hereafter: the target) based on knowledge in another domain (hereafter: the source). There are two approaches for cross domain: 1) those that use a mediator to construct and initialize an empty user model, or that enrich the existing user model by using provided data or a partial user model in another domain or service; 2) those that do not use a mediator. In this paper we propose a content-based cross domain RS that uses a mediator to suggest items to **a new user** in a target domain who has only rated items in the source domain. We assume that: 1) the target domain has very high sparsity; 2) the source domain has low sparsity; and 3) the new user has already rated some items in the source domain. Based on these assumptions, traditional recommendation algorithms do

not work well in the target domain for two reasons: first, high sparsity; second, the new user and cold-start issues.
The problem: Given a set of items T in a target domain, and given a user u who has already rated certain items in the source domain but not in the target, what are the N most preferred items in T by user u?

We begin solving this problem by presenting a new graph-based transfer learning method for content-base recommendation. The concept of behavior tree is defined as a topological representation of users' behavior patterns. By ***users' behavior*** refers to the items rated by a user and the rating values the user has assigned to these items. Based on these trees, we find correlated behavior patterns in the source and target domains which are considered bridges between the two domains. Then, the data of each bridge is combined with the content data of the items in a one features vector. This vector is used as a sample in the training data for a machine learning algorithm. Here common users are assumed to exist between the source and target domains.

The paper's innovation is (1) the graph structure employed for dealing with the problem, instead of just using raw numbers. The trees structure provides a rich representation of the users' behavior. First, the structure enables items to be clustered according to the users' behavior, and the clusters to be represented by a forest of behavior trees. Second, the tree's structure defines a hierarchy among the items, this hierarchy is important because it shows how much the item represents the behavior pattern it belongs to. (2) Using the items' content data from a rich real world dataset with transfer learning recommenders that are employed to improve classifier performance. Finally, we find a correlation (mapping) between items in the source domain and items in the target domain, so the mapping is obtained by running a process of tree matching between trees in the source and trees in the target. Furthermore, from the content data, more features are gained to represent the training samples for building the recommendation model.

2 Related Work

Several recent studies apply cross domain techniques in order to achieve performance improvement over a single domain recommender [9]. One of the first studies [13] is a user study that examined the fundamental assumption that users share similar preferences across domains. The first case study presented in this work tested the correlations between related items in different domains. Their similarity across domain was determined according features or topical similarity. For example, "Harry Potter" the movie and the game are considered similar items, as are Andy Lau the singer and the song sung by her. This study found that positive correlations of user preferences usually exist across domains. For instance, users having positive evaluation of a person as an actor/actress also exhibit a strong interest in the same person as a singer.

The second case study examined the correlation in users' preferences on items of the same genre but of different domains (e.g. preferences of action movies and

action games). The results showed a positive correlation of users' preferences of the same genres across domains. The results reinforced the assumption that user interest exists across domains.

The study's evaluation was based on user's questionnaires (the users were asked for their preferences) and not on an actual recording of system usage, with a limited number of items and users. Moreover, it was based on the aggregate ratings that users provided rather than on an individual's ratings. Therefore, we can only infer the existence of general trends towards preferences across domains. However, this is the only user study that has attempted to examine the assumption that all other studies assume about the relatedness of preferences across domains.

Among the studies dealing with cross domain recommenders, one can distinguish between two main approaches for applying data from different domains are discernible. One aims at constructing a unified user model from the different domains, while the second approach uses "Transfer Learning" techniques. The following sections review the characteristics of the two approaches.

User-Model Mediation Perhaps the simplest approach to cross domain recommendations is to import relevant data from a different domain and aggregate it with the original target data. The aggregation is simple when the domains share information on the same users. Berkovsky et al. [3] refer to this problem as cross user-model mediation and suggest the following definition: A user model is the data that the RS has collected on the user in order to provide him with personalized services. Most of the recommender systems maintain UMs that are tailored to their specific application or domain. According to Berkovsy et al. [1], since the quality of the personalized services depends heavily on the UM's characteristics and accuracy, different services would benefit from enriching their UMs by importing, translating and aggregating partial UMs from related domains or services.

Several UM mediation techniques are represented in [4] and [3]. The target RS sends a request for rating prediction; the other RSs in the different domains send information back to the target system; and each suggested technique differs in the information returned to the target system:

Centralized Prediction - Each RS sends back its local users-items-ratings matrix. The target RS unifies all the given matrices with its own local matrix, and run collaborative Filtering (CF) on the unified matrix. Distributed Peer Identification - this technique assumes that if two users are similar in some domain, they may also be similar in another related domain. In practical terms, each RS computes the k-nearest neighbors of the given user-id according to its local matrix and returns their identities. The target RS unifies the given answers with its local nearest neighbors and computes the rating using CF. Distributed Neighborhood Formation - Each RS computes the similarity between the active user i and the other users using their own local rating matrix. A set of K nearest-neighbors is selected, and their identifiers, together with their similarity are sent to the target RS. The target RS averages the domain-specific similarity values

into the overall similarity metric using inter-domain correlation values. When the overall similarity value is computed, K nearest neighbors can be selected and the predictions generated. Distributed Prediction- Each RS that contains rating data on the desired item computes locally the rating using CF, and sends back the rating to the target. The target RS calculates its local CF rating, and calculates the average of all the ratings.

Experiments were conducted to compare these approaches to a baseline approach that used only the target domain data. Results showed that in some conditions integrating neighbors (2nd approach) and integrating recommendation (3rd approach) can be beneficial, as they may improve, respectively, the coverage and accuracy of the generated recommendations. However, in other cases, when the target system has enough rating records, it is preferable to use only the target data since the remote systems data may be interpreted as noise.

The study has certain limitations. The method assumes that the different domains share the same users who are identifiable in different applications. Yet, in many applications no such overlapping exists and even if there are common users, they are unidentifiable. In addition, some of the suggested approaches, for example the Distributed Prediction approach, assume that at least a large portion of the domains share the same items with the target domain, which means that such methods cannot be applied to totally different domains like movies and songs.

A similar approach is described in [6]. The authors generated a uniform UM approach that aggregated features from different domains, and mapped them to relevant domains.

In conclusion, the main advantage of the user-model mediation is by its simplicity and intuitive implementation. However, it can be applied only to particular problems when the different sources share implicit resources like users, items or features. Since one of our goals in this work has been to avoid restricting users or items that overlap between domains, we did not use this approach.

Transfer Learning Transfer learning (TL) is a relatively new area of research (since 1995) in machine learning whose aim is to extract knowledge that was learned for one task in a domain and use it for a target task in a different domain [11]. In the field of machine learning we usually train a model based on available data for the problem that we are interested in (training data). The model is used for predicting behavior in examined domain (using the testing data). For example, in recommender systems the model can help us predict whether or not the user will like a movie.

Transfer Learning is a very intuitive notion since human nature is to transfer learning from different aspects of life. People can intelligently apply previously learned knowledge to solve new problems faster or better. For example, when learning a foreign language such as French, knowledge of Romanian can be used to pick up new words and skills that we acquired in the past can be employed to assimilate the new language. Another example, when learning a new programming language like C++, our knowledge of Java and our general programming

skills can be exploited to our benefit. TL also enables us to retain and reuse previously learned knowledge [11], and this may reduce the need and effort to recollect training data in new applications.

An important TL characteristic is that it does not always require content overlap between the different domains. Regarding the recommender systems application, TL does not require that the source and target domain share users or items, since it aims at finding common consumption patterns that exist in related domains by recognizing latent behavior groups [10]. Consider, for example, the music and games domains. Although they do not appear strongly connected, the same latent groups of users are still found in both domains. For example, there might be a group of consumers that always purchase new trendy items, or another group which likes to consume low cost products, and still other users who especially enjoy children's items.

Transferring knowledge between domains is a daunting task because the knowledge of one domain is not guaranteed to be useful for another domain. The success of transfer learning depends on a variety of factors, e.g. the degree of domain correlation (for example, are movies and books more closely correlation than movies and jokes), the data characteristics (sparsity level, rating scale etc.), and whether the domains share common resources such as items or users.

There are only a few transfer learning applications for the recommender system. Most of the works that used TL for recommender systems are based on collaborative filtering. Next several works are reviewed that employ TL for recommender systems and discuss their limitations and difference from our work.

Li et al [10] may be the most relevant work. They suggested that when insufficient data in the target domain prevents training an accurate recommendation model, useful knowledge can be borrowed from a different domain and its data used to train the model. They introduced the idea that rating matrices of different domains may share similar user-item rating patterns. Thus, they adopted a user-item rating matrix of the source domain, referred to as a "codebook", and transferred the rating patterns to the target domain in order to fill in the target domain's missing values. Their algorithm consists of two steps. First, a rating pattern (codebook) of the dense domain is created, which summarizes the original rating data. Second, the codebook is expanded in order to learn the missing values in the target domain.

3 The Recommendation Framework

Our framework deals with the abovementioned problem in three phases: The first is described in (3.1) and aims to preprocess the users' data in the source and target domains. Preprocessing consists of three steps: 1) building behavior trees for each domain; 2) tree matching; 3) building training samples. The second phase (3.2) is for training a model on the training set. The third phase, described in (3.3), is for recommending items to new users based on their behavior in the source domain.

3.1 Preparing Training Data

Building Behavior Graphs Constructing the graphs requires four steps. **First**, the same item with different rating values is separated by expanding each item into k items, where k equals the number of possible rating values in the domain. For example, if item i was rated as $r1$ by one group of users and the same item received an $r2$ rating from a different group of users, we consider them as different items and represent them by i_{r1} and i_{r2}. Note: the number of items after this step is $k \times number\ of\ origin\ items$. **Second step.** The separated items are sorted by their popularity, that is, by the number of the user ratings. Table (1) is an example of a rating matrix, where the popularity of item 1_2 (item 1 with rating 2) in this matrix is 3.

	item1	**item2**	**item3**	**item4**	**item5**
user1	2			1	1
user2	1		1	1	3
user3	2	1	2	3	
user4	3		2		1
user5		3	3	1	1
user6	1	3		1	
user7	1		1	2	3
user8	2	3	3	2	1
user9	1	3		1	
user10	1		1		3

Table 1. *Rating matrix, possible ratings [1, 2, 3] (k=3).*

Third step: A topological representation is adopted by taking the sorted items set of each domain and representing them by a disconnected weighted graph. The graphs are constructed as follows: **Nodes**: Each item on the sorted items list is represented by one node (note: the node represents a pair of an item and one rating value). **Edges**: An edge is found between two nodes if there are common users exist who rated both items represented by both nodes (note: rating must be the same as in the node). **Weight**: The weight of the edge between two nodes is defined as the Jaccard coefficient between the sets of users who rated the items represented by the nodes. The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets (Wikipedia):

Similarity(A, B) = J(A, B) $= \frac{|A \bigcap B|}{|A \bigcup B|}$. If the two sets A and B are empty then we define $J(A, B) = 0$. The max value of this similarity when A, B are finite is 1, and the minimal value is 0.

If the weight of edge is less than a given threshold [3] we drop the edge. Note, a one-to-one mapping exists between the nodes and the items, thus the term node or item can be used to refer to the same element. Figure (1) shows the behavior graphs (disconnected) obtained from table (1) when threshold = 0.5.

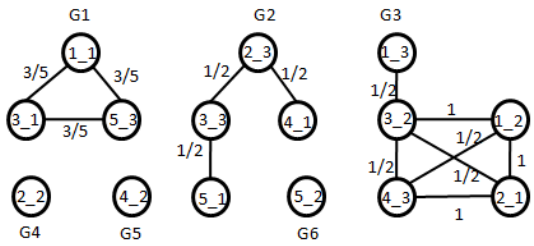

Fig. 1. *Behavior graphs (disconnected) of table (1) when threshold = 0.5.*

Fourth step: Each component in the disconnected graph is converted into a topological tree which we refer to as *a behavior tree*. This is another representation of the users' behavior in the graph based on the graph's structure. Transferring a graph into a tree is a known problem. For example, in [5] the authors showed a general transformation between graphs, trees and generalized trees by introducing a special transformation that uses the DIJKSTRA distance between nodes to define a multi-level function which assigns each node of the graph a level value that introduces a hierarchy.

This process takes place by adding an artificial node as the root for each tree, and connecting all the popular [4] nodes in the graph to the node. The weight of each child in level one is equal to the child's popularity. Then a recursive greedy algorithm is used to complete the tree by adding all the other nodes on the graph to it. This is done by connecting each child (who is not in the tree yet) to the parent who is connected to the highest edge weight in the original graph.

The weight of each node (except in first level) in the tree is the same as the edge from the parent to the child in the graph. The idea behind adding an artificial node is to make sure that all the popular items in the graph appear in the first level of the tree and the less popular items appear in deeper levels.

The motivation for moving from graphs to trees is the tree's simpler hierarchical structure than that of graphs, and the tree's greater ability to represent sufficient data of the behavior graphs. Figure (1) represents the graphs obtained from the rating matrix in Table (1), and Figure (2) illustrates the behavior trees

[3] This threshold depends on different parameters, such as the number of users in the domain, the number of items, the number of ratings, etc. We set it as 0.5 in our experiments.

[4] The most popular items in each graph are those which have greater popularity than the average popularities in the same graph.

based on the behavior graphs in Figure (1), where T_i is the behavior tree of graph G_i, $i : 1 \rightarrow 6$, and the nodes with labels A1-A6 are artificial.

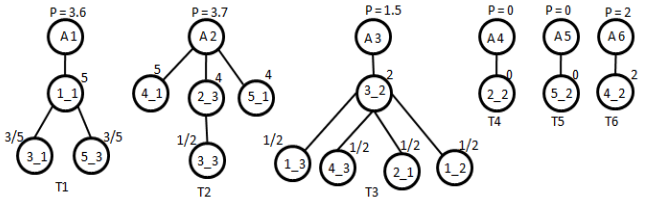

Fig. 2. *Constructed behavior trees based on the behavior graphs (1). "P" is the popularity threshold.*

Tree Matching The first task is to decide for each tree in the source domain which tree in the target domain is the best for matching. Tree matching appears in various fields such as image clustering, RNA, chemical structure analysis, and so fourth. Thus, many method were defined to measure similarity between trees, for instance largest or smallest common subtree, tree edit distance, or the transferable ratio between two trees [8].

Since we have an advantaged by the common users, the similarity between two trees is defined the same way that similarity between two items is defined in (3.1), but here it is referred to on a set of users instead of items:

$$\textit{Tree-Similarity(T1, T2)} = \textit{J(U(T1), U(T2))} = \frac{|U(T1) \bigcap U(T2)|}{|U(T1) \bigcup U(T2)|}$$

$$\textit{U(T)} = \bigcup_{i=1}^{|T|} (u \mid \text{user } u \text{ rated item } i \text{ in } T).$$

The similarity measure relates to the content of the trees instead of the structure, so if two trees have the same set of users then they receive a higher similarity value than trees that do not have common users. However, after a tree in the target has been found for each tree in the source that it should be matched with, we run the tree matching process to match the nodes in both trees, and obtain the advantages of the structure of the trees as well. Eventually, a set of pairs of matched nodes is arrived at that are the bridges between the items in the source and the items in the target.

Building the Training Samples This is the final step in preparing training data. Here the training set must be prepared for a supervised machine learning task. A special structure for features vectors is found that combines the data of users' behavior in the source and target domains, and the items content data. This is done by taking all the matched nodes from the previous step. Then, for each two matched nodes n_s and n_{tr} that represent item t_s in the source with rate r_s and item t_{tr} in the target with rating r_{tr}, respectively, one features vector is

defined as following:
$[f_1(t_s), ..., f_n(t_s), r_s; f_1(t_{tr}), ..., f_m(t_{tr}), r_{tr}]$, where $f_i(t)$ is the value of feature i for item t. This structure is meaningful for cross-recommendation because it provides a full image of the user's behavior from the previous step and the features of the items they rated, since it combines the features with the rating values for each group of users, For example, if the users who like books with features $[b_1, b_2, ..., b_n]$, like movies with features $[m_1, m_2, ..., m_m]$, then a new user u with similar behavior in the books domain will be in the same behavior tree and probably has similar behavior toward movies with features $[m_1, m_2, ..., m_m]$.

3.2 Model Training

Here a model of a multiclass classifier is built on the training samples that combine features of items in the source, ratings in the source, features of items in the target, and ratings in the target. Since our goal is to predict the user's rating of an item in the target domain, the class of the classifier is the last attribute in the features vector, where all the other attributes in the vector represent the features. This model is expected to predict rating for an item t in the target domain for a new user u that has rated only items in the source domain. The number of the classes equals the number of possible rating values in the target domain.

3.3 Recommendation

The recommendation task is based on the model described in the previous task (3.2). When the system is asked to recommend the top N items in the target domain for a new user u, the system ranks each item tr in the target domain based on the items rated by the user in the source domain. The ranking is carried out by building *a features matrix* for each item tr in the target as follows:

- The number of rows in the matrix equals the number of items the user rated in the source.
- The number of the column in the matrix equals $n + m + 2$, where n is the number of the item's features in the source, m is the number of the item's features in the target domain and + 2 for the rating in the source and the class (rating in target).
- The value in row i, column j, equals to:
  If $j < n + 1$ Then: The value of feature j in item i (in the source)
  If $j = n + 1$ Then: The rating value that the user u rated item i (in the source)
  If $j > n + 1$ Then: The value of feature $j - (n + 1)$ of the item tr (in the target)
  If $j = n + 2$ Then: The value is "?" (missing)

To find the rank of a features matrix M, we first run the classifier on each row in the matrix in order to return the vector of the predicted probability for each

class. This vector is called *the distribution vector* and represents the probability for each class, so the value in entry i equals the probability that this sample is classified as i, and the vector's size equals the number of the classes (possible ratings). For each row we take the distribution vector P returned by the classifier and compute the ***expected rating***[5] as follows: ***expected rating*** $= \sum_{j=1}^{k} j \times P[j]$, when k is the size of P. The rank of the features matrix M is defined as the average of the expected ratings of all the rows in M: $Rank(M) = \frac{\sum_{j=1}^{z} ER(j)}{z}$, where $ER(j)$ is the expected rating for row j and z is the number of the rows in M. Then the matrices are sorted by the rank value, and the target items represented by the top N matrices are sent back as the recommended items for the user u.

4 Experiment

In this chapter we investigate whether the additional knowledge gained from the source domain and the content of the items can improve the recommendation in the target domain. Our approach is compared with the popularity, which is a single domain approach generally used for recommendation to new users, and with the KNN-cross-domain approach that uses both domains but does not use the content data.

4.1 Dataset

We used the Loads data-set in our experiments. Loads data is a real-world content dataset that includes different domains such as videos, music, games (common users among the domains). Music-loads was chosen as the source domain and game-loads as the target domain, and 600 common users were extracted from both domains, 817 items from the music domain with 18,552 ratings, and 1264 items from the games domain with 17,640 ratings. This dataset is event-based, thus we manipulated it and converted the events to ratings by weighting the events by their type. For example, the event *user buys an item* that was converted to the max rate value. For this experiment we used a binary rating.

4.2 Evaluation and Metrics

Our goal is to recommend a set of N items that may find the interest of the new user in the target domain. This kind of recommendation refers to recommending good items [7] or *top-N* items. The correct method of evaluating the recommender in this case is by measuring the precision at N (or Top-N precision) which is the number of interesting items from the recommended items [7]. Since we have content data, a recommended item is considered to be a true positive if it is similar to 80% of the positively rated items

[5] Ratings here are natural numbers between *1* and k when *1* is the minimal value, and k is the maximal value.

4.3 Baseline

Our method, referred to as BGM (Behavior Graph Matching), is compared with two base-line recommenders:

KNN-cross-domain: This recommender is a cross domain recommender based on collaborative filtering with Pearson's correlation coefficient method. The main idea behind the method is to find the K-nearest-neighbors of the active user in the source domain who also rated items in the target domain, and to consider them as the K-nearest-neighbors of the active user, who is also in the target domain, and then predict the user's ratings in the target domain based on his/her neighbors' ratings in the target domain.

Popularity: This is a naive method that recommends the popular items to the new users. This method is the simplest to implement and sometimes outperforms other complex algorithms [12] especially in a domain where most of the users have similar preferences.

4.4 BGM versus Popularity

We evaluate the two methods by 10-fold cross validation on the Loads dataset, when each fold includes 60 users for testing and 540 users for training. The test set was considered as the new users' set in the target domain who have ratings just in the source domain. The behavior patterns were based on all the users in the source domain, and users who belong to the training set in the target domain. For each user in the test set, the recommender was asked to recommend the *Top-N* items when *N= 5, 10, 15, 20, 50, and 100*, and for each set of recommended items the top-N precision was measured per user. The popularity algorithm employs the training set in the target domain to find the popular item that was recommended for each of the test users, then the top-N precision was compared for each N, and for each N the average of the top-N precision was found for all of the users.

Figure (3) shows the results, where we divided into 6 groups, each group representing a different N value and containing three columns, the first one representing the average *Top-N* precision value for the BGM, and the second representing the average of *Top-N* precision value for the popularity algorithm. Note that BGM outperforms the popularity in all the cases except when *N= 100*. The reason is BGM tries to recommend items which are similar to the user's behavior, thus, with small values of N BGM knows better than popularity what items are related to the user, but sometimes users behave in a different way than their normal behavior, such as one decide to buy a book because it is a little bit popular and there is a good offer about it. Here, popularity recommender will recommend this item specially when N is big, while BGM will not recommend this item because it is not similar to the user's preferences. Thus, our results-based conclusion: it is better to use the source domain to improve recommendation in the target domain, specially in cases there is a limitation of the number of items to recommend.

4.5 BGM Vs. KNN cross-domain

This experiment was designed to compare the performance of our approach with the collaborative-filtering cross-domain approach which also uses the knowledge in the source domain to return recommendations to the target domain. The main goal was to determine whether the BGM method makes maximum use of the content data of the items, and to check whether the proposed behavior patterns work well for representing the users' behavior and knowledge transfer. As in the first experiment, this on was conducted with a 10-fold cross validation on Loads dataset.

When comparing the results in Figure (3), where the third column in each group represents the value of top-N precision of the KNN-cross-domain, note that BGM outperforms the KNN-cross-domain in 83% of the cases. The only case where KNN-cross-domain outperforms BGM is when *N=100*. This is because of a similar reason to the one described above: sometimes people make exceptions in their behavior, such as one decide to go to watch a movie just to join friends, so BGM will not perform well with these cases, however, people usually have a consistence behavior, and recommendation is done with small values of *N*.

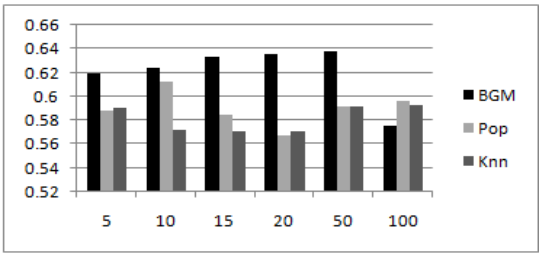

Fig. 3. *Top-N precision values for BGM, Popularity, and Knn-cross-domain, with different N values*

4.6 Statistical Analysis

Paired t-tests were performed on the same results that were received by running 10-fold cross-validation with BGM, Popularity and KNN-cross-domain on Loads dataset, and their statistically significant was checked. Each user was considered a participant, and each of the recommenders was considered a different method of the test.

A table of 600 users (rows) was obtained, and for each user the top-N precision value was received when $TopN = 5$, 10, 15, and 20 with each of the methods (12 columns per user). Then the t-test was performed for every two relevant columns (with the same N) . All the results are statistically significant, which means that our method performance is decidedly better than the other two when the

number of recommended items is 5, 10, 15, and 20, and the size of the sample is 600 participants.

5 Conclusions

Our paper has presented and evaluated a novel method of transfer learning in content-based recommenders by using a new topological structure that we call a behavior graph. The main idea of using such structure is its ability to represent rich data about the users' behavior, and the relation between items in a structure. By employing tree matching methods we discovered a correlation between items in the source and items in the target.

We compared our method with the popularity approach, which is generally used with recommendations to new users, and with the KNN-cross-domain method. The comparison was based on a real-world dataset called Loads dataset, and the Top-N precision metric was evaluated by 10-fold cross validation.

The results show that our method (referred to as BGM) outperforms the popularity and KNN-cross-domain methods in the majority of cases. Our conclusion: it is preferable to use the data in the source domain and the item's content data when dealing with this kind of recommendation problem.

References

1. S. Berkovsky, T. Kuflik, and F. Ricci. Cross-technique mediation of user models. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 21–30. Springer, 2006.
2. S. Berkovsky, T. Kuflik, and F. Ricci. Cross-domain mediation in collaborative filtering. *User Modeling 2007*, pages 355–359, 2007.
3. S. Berkovsky, T. Kuflik, and F. Ricci. Cross-domain mediation in collaborative filtering. In *User Modeling 2007*, pages 355–359. Springer, 2007.
4. S. Berkovsky, T. Kuflik, and F. Ricci. Distributed collaborative filtering with domain specialization. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 33–40. ACM, 2007.
5. F. Emmert-Streib and M. Dehmer. Topological mappings between graphs, trees and generalized trees. *Applied Mathematics and Computation*, 186(2):1326–1333, 2007.
6. G. González, B. López, and J. L. de la Rosa. A multi-agent smart user model for cross-domain recommender systems. *Proceedings of Beyond Personalization*, 2005.
7. J. L. Herlocker, J. A. Konstan, L. G. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.
8. T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 75–86, London, UK, 1994. Springer-Verlag.
9. B. Li. Cross-domain collaborative filtering: A brief survey. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 1085–1086. IEEE, 2011.

10. B. Li, Q. Yang, and X. Xue. Can movies and books collaborate? cross-domain collaborative filtering for sparsity reduction. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 2052–2057. Morgan Kaufmann Publishers Inc., 2009.
11. S. J. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.
12. A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. CROC: A New Evaluation Criterion for Recommender Systems.
13. P. Winoto and T. Tang. If you like the devil wears prada the book, will you also enjoy the devil wears prada the movie? a study of cross-domain recommendations. *New Generation Computing*, 26(3):209–225, 2008.