# A Decision Tree Based Recommender System

Amir Gershman, Amnon Meisels
Department of Computer Science,
Ben-Gurion University of the Negev
Beer-Sheva, 84105, Israel
amirger, am@cs.bgu.ac.il

Karl-Heinz Lüke
Deutsche Telekom AG, Laboratories,
Innovation Development
Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
Karl-Heinz.Lueke@telekom.de

Lior Rokach, Alon Schclar, Arnon Sturm
Department of Information Systems Engineering,
Ben-Gurion University of the Negev
Beer-Sheva, 84105, Israel
liorrk, schclar, sturm@bgu.ac.il

**Abstract:** A new method for decision-tree-based recommender systems is proposed. The proposed method includes two new major innovations. First, the decision tree produces *lists of recommended items* at its leaf nodes, instead of single items. This leads to reduced amount of search, when using the tree to compile a recommendation list for a user and consequently enables a scaling of the recommendation system. The second major contribution of the paper is the splitting method for constructing the decision tree. Splitting is based on a new criterion - the least probable intersection size. The new criterion computes the probability for getting the intersection for each potential split in a random split and selects the split that generates the least probable size of intersection. The proposed decision tree based recommendation system was evaluated on a large sample of the MovieLens dataset and is shown to outperform the quality of recommendations produced by the well known information gain splitting criterion.

## 1 Introduction

Recommender Systems (RS) propose useful and interesting items to users in order to increase both seller's profit and buyer's satisfaction. They contribute to the commercial success of many on-line ventures such as Amazon.com or NetFlix [Net] and are a very active research area. Examples of recommended items include movies, web pages, books, news items and more. Often a RS attempts to predict the rating a user will give to items based on her past ratings and the ratings of other (similar) users.

Decision Trees have been previously used as a model-based approach for recommender systems. The use of decision trees for building recommendation models offers several benefits, such as: efficiency and interpretability [ZI02] and flexibility in handling a variety of input data types (ratings, demographic, contextual, etc.).

The decision tree forms a predictive model which maps the input to a predicted value based on the input's attributes. Each interior node in the tree corresponds to an attribute and each arc from a parent to a child node represents a possible value or a set of values of that attribute. The construction of the tree begins with a root node and the input set. An attribute is assigned to the root and arcs and sub-nodes for each set of values are created. The input set is then split by the values so that each child node receives only the part of the input set which matches the attribute value as specified by the arc to the child node. The process then repeats itself recursively for each child until splitting is no longer feasible. Either a single classification (predicted value) can be applied to each element in the divided set, or some other threshold is reached.

A major weakness in using decision trees as a prediction model in RS is the need to build a huge number of trees (either for each item or for each user). Moreover, the model can only compute the expected rating of a single item at a time. To provide recommendations to the user, we must traverse the tree(s) from root to leaf once for each item in order to compute its predicted rating. Only after computing the predicted rating of all items can the RS provide the recommendations (highest predicted rating items). Thus decision trees in RS do not scale well with respect to the number of items.

We propose a modification to the decision tree model, to make it of practical use for larger scale RS. Instead of predicting the rating of an item, the decision tree would return a weighted list of recommended items. Thus with just a single traverse of the tree, recommendations can be constructed and provided to the user. This variation of decision tree based RS is described in section 2.

The second contribution of this paper is in the introduction of a new heuristic criteria for building the decision tree. Instead of picking the split attribute to be the attribute which produces the largest information gain ratio, the proposed heuristic looks at the number of shared items between the divided sets. The split attribute which had the lowest probability of producing its number of shared items, when compared to a random split, is picked as the split attribute. This heuristic is described in further detail in section 3.

We evaluate our new heuristic and compare it to the information gain heuristic used by the popular C.45 algorithm ([Qui93]) in section 4.

## 2 RS-Adapted Decision Tree

In recommender systems the input set for building the decision tree is composed of *Ratings*. *Ratings* can be described as a relation $< ItemID, UserID, Rating >$ (in which $< ItemID, UserID >$ is assumed to be a primary key). The attributes can describe the users, such as the user's age, gender, occupation. Attributes can also describe the items, for example the weight, price, dimensions. *Rating* is the target attribute which the decision tree classifies by. Based on the training set, the system attempts to predict the *Rating* of items the user does not have a *Rating* for, and recommends to the user the items with the highest predicted *Rating*.

The construction of a decision tree is performed by a recursive process. The process starts at the root node with an input set (training set). At each node an item attribute is picked as the split attribute. For each possible value (or set of values) child-nodes are created and the parent's set is split between child-nodes so that each child-node receives as input-set all items that have the appropriate value(s) that correspond to this child-node. Picking the split-attribute is done heuristically since we cannot know which split will produce the best tree (the tree that produces the best results for future input), for example the popular C4.5 algorithm ([Qui93]) uses a heuristic that picks the split that produces the largest information gain out of all possible splits. One of the attributes is pre-defined as the target attribute. The recursive process continues until all the items in the node's set share the same target attribute value or the number of items reaches a certain threshold. Each leaf node is assigned a label (classifying its set of items), this label is the shared target attribute value or the most common value in case there is more than one such value.

Decision trees can be used for different recommender systems approaches:

- Collaborative Filtering - Breese et al. [BHK98] used decision trees for building a collaborative filtering system. Each instance in the training set refers to a single customer. The training set attributes refer to the feedback provided by the customer for each item in the system. In this case a dedicated decision tree is built for each item. For this purpose the feedback provided for the targeted item (for instance like/dislike) is considered to be the decision that is needed to be predicted, while the feedback provided for all other items is used as the input attributes (decision nodes). Figure 1 (left) illustrates an example of such a tree, for movies.

- Content-Based Approach - Li and Yamda [LY04] and Bouza et al. [BRBG08] propose to use content features to build a decision tree. A separate decision tree is built for each user and is used as a user profile. The features of each of the items are used to build a model that explains the user's preferences. The information gain of every feature is used as the splitting criteria. Figure 1 (right) illustrates Bob's profile. It should be noted that although this approach is interesting from a theoretical perspective, the precision that was reported for this system is worse than that of recommending the average rating.

- Hybrid Approach - A hybrid decision tree can also be constructed. Only a single tree is constructed in this approach. The tree is similar to the collaborative approach, in that it takes user's attributes as attributes to split by (such as her liking/disliking of a certain movie) but the attributes it uses are general attributes that represent the user's preference for the general case, based on the content of the items. The attributes are constructed based on the user's past ratings and the content of the items. For example, a user which rated negatively all movies of genre comedy is assigned a low value in a "degree of liking comedy movies" attribute. Similarly to the collaborative approach, the tree constructed is applicable to all users. However, it is now also applicable to all items since the new attributes represent the user's preferences for all items and not just a single given item. Figure 2 illustrates such a hybrid tree.

Consider a general case with a data set containing *n* users, *m* items, and an average decision tree of height $h$. The collaborative filtering based RS requires $m$ trees to be constructed, one for each item. When a user likes to receive a recommendation on what movie to watch, the system traverses all trees, from root to leaf, until it finds an item the user would like to view. The time complexity in this case is therefore $O(h \cdot m)$. This might be too slow for a large system that needs to provide fast, on-demand, recommendations to users. The content based approach requires *n* trees to be constructed, one for each user. When a user likes to receive a recommendation, the system needs to traverse the user's tree from root to leaf once for each item, until it finds an item the user would like to view. The time complexity in this case is therefore $O(h \cdot m)$. Similarly, in the hybrid approach, the tree needs to be traversed once for each item, and the time complexity is also $O(h \cdot m)$.

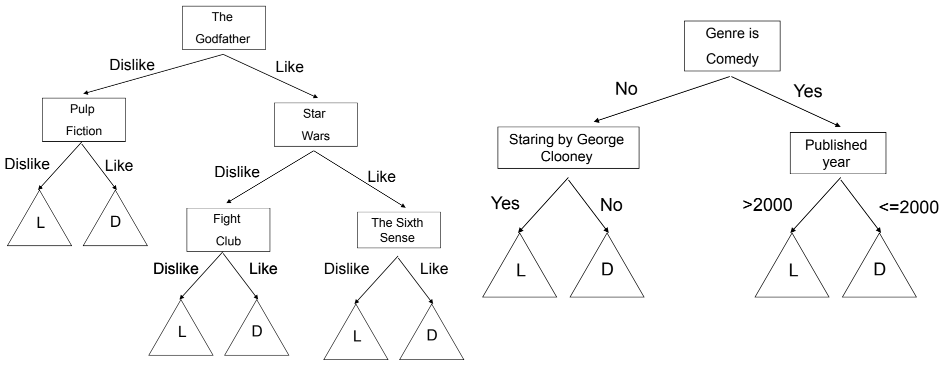

Figure 1: Left: A CF decision tree for whether users like the movie "The Usual Suspects" based on their preferences to other movies such as The Godfather, Pulp Fiction etc. A leaf labeled with "L" or "D" correspondingly indicates that the user likes/dislikes the movie "The Usual Suspects". Right: A CB decision tree for Bob.

In systems which require fast computation of recommendations and with many possible items to recommend, all the above decision tree based RS would be impractical. Therefore we propose a modification of the decision tree to better fit RS, and provide recommendations faster to users.

Our proposed algorithm is similar to the ID3 algorithm [Qui86] and uses the hybrid approach. Because we use the hybrid approach, only a single tree is needed, and the attributes to split by are only attributes that describe users. These attributes can be computed based on the user's past ratings and the content of the items, as shown in the example in figure 2 but they can also include user profile attributes such as age, gender etc. The major variation from the ID3 algorithm is in the leaf nodes of the tree. Instead of creating leaf nodes with a label that predicts the target attribute value (such as rating), we propose to construct a recommendation list out of the leaf's input set and save this list at the leaf node as its

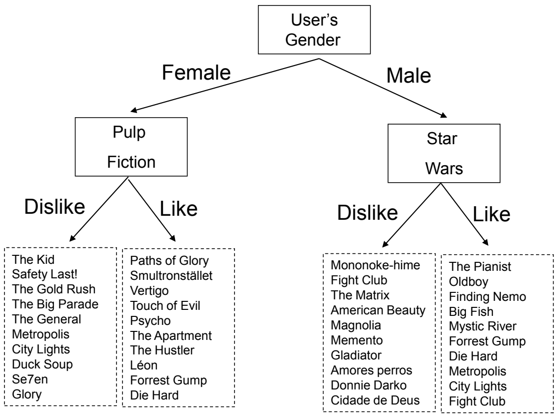

Figure 2: An example CF-CB hybrid decision tree

label. When a user wishes to receive recommendations, the tree is traversed based on the user's attributes until the leaf node is reached. The node contains a pre-computed recommendation list and this list is returned to the user. Thus the time complexity is reduced to only $O(h)$.

Building the recommendation list at the leaf node can be done in various ways. A simple solution selected here is to compute the weighted average of the ratings in all tuples at the leaf's *Ratings* set and to recommend the items with the highest weighted average first. Consider the rightmost leaf in the example tree in figure 2. For any tuple $< i, u, r >$ in the *Ratings* set (i,u,r denote an item, a user and a rating, respectively) at this leaf node we know that *u* has a degree of liking the genre Comedy more than 0.5 and a degree of liking movies with the actor George Clooney more than 0.7. All the items rated by users such as *u* appear in this *Ratings* set along with the ratings each user submitted. This leaf therefore contains the ratings of users similar to *u*. We assume that if we now pick the items which were rated the highest by the users similar to u, these would form a good recommendation for this user. Therefore, we order all items based on a weighted average (since items can appear more than once, when more than one user rated them) and set this list as the recommendation list of this leaf node. The algorithm is presented in detail in Algorithm 1.

## 3 Least Probable Intersections

Decision trees seek to provide good results for new unseen cases, based on the model constructed with the training set. To accomplish this, the construction algorithm strives for a small tree (in number of nodes) that performs well on the training set. It is believed that a small tree would generalize better, avoids over fitting, and forms a simpler representation for humans to understand [Qui86]. The C4.5 [Qui93] is a popular algorithm for construct-

**Algorithm 1** RS-Adapted-Decision-Tree(Ratings)

```
create root node
if (Ratings.size < threshold)
    root.recommendations ← recommendation_list(Ratings)
    return root
else
    Let A be the user attribute that best classifies
    the input
    For each possible value v of A
        Add a branch b below root, labeled (A=v)
        Let $Ratings_v$ be the subset of Ratings that have
        the value v for A
        add RS-Adapted-Decision-Tree($Ratings_v$) below this
        new branch b
return root
```

ing such decision trees. It uses the criterion of normalized information gain [Mit97] to pick the attribute by which to split each node. The attribute with the largest normalized information gain is picked, as it provides the largest reduction in entropy. Since this is a heuristic, it does not guarantee the smallest possible tree.

In recommender systems the input set for building the decision tree is composed of *Ratings*. *Ratings* can be described as a relation $< ItemID, UserID, Rating >$ (in which $< ItemID, UserID >$ is assumed to be a primary key). The intuition behind the heuristic proposed in the present paper is as follows. Consider a split into two subsets, *A* and *B*. The less *ItemIDs* are shared between the sets *A* and *B*, the better the split is since it forms a better distinction between the two groups. However, different splits may result in different group sizes. Comparing the size of the intersection between splits of different sub-relation sizes would not be good since an even split for example (half the tuples in group *A* and half in group *B*) would probably have a larger intersection than a very uneven split (such as one tuple in group *A* and all the rest in group *B*). Instead, we look at the probability of our split's item-intersection size compared to a random (uniform) split of similar group sizes. A split which is very likely to occur even in a random split is considered a bad split (less preferred) since it is similar to a random split and probably does not distinguish the groups well from each other. A split that is the least probable to occur is assumed to be a better distinction of the two subgroups and is the split selected by the heuristic.

More formally let us denote:

- $items(S) = \pi_{ItemID}(S)$, where $\pi$ is the projection operation in relational algebra, the set of all *ItemIDs* that appear in a set *S*.

- $O_i(S) = |\sigma_{ItemID=i}(S)|$, where $\sigma$ is the select operation in relation algebra, is the number of occurrences of the *ItemID i* in the set *S*.

Let $S_q$ ($q$ denotes the number of ratings) be a random binary partition of the tuples in *Ratings* into two sub-relations *A* and *B* consisting of *k* and *q-k* tuples, respectively. We are interested in the probability distribution of $S_q \equiv |(items(A) \bigcap items(B)|$.

First, let us find the probability of an item belonging to the intersection. The probability of all $o_i$ occurrences of any item *i* to be in the set *A* is $\left(\frac{k}{q}\right)^{o_i}$. Similarly the probability for all $o_i$ occurrences of item *i* to be in the set *B* is $\left(\frac{q-k}{q}\right)^{o_i}$. In all other cases item *i* will appear in the intersection, thus the probability $P_i$ that item *i* belongs to $items(A) \bigcap items(B)$ is:

$$P_i = 1 - \left(\frac{k}{q}\right)^{o_i} - \left(\frac{q-k}{q}\right)^{o_i} \tag{1}$$

Next, we can construct a random variable $x_i$ which takes the value 1 when item *i* belongs to the intersection of *A* and *B*, and the value 0 otherwise. Using the above equation, the variable $x_i$ is distributed according to a Bernoulli distribution with a success probability $P_i$. Thus, $S_q$ is distributed as the sum of $|items(Ratings)|$ non-identically distributed Bernoulli random variables which can be approximated by a Poisson distribution [Cam60]:

$$Pr(S_q = j) = \frac{\lambda^j \cdot e^{-\lambda}}{j!} \tag{2}$$

where

$$\lambda = \sum_{i \in items(Ratings)} P_i \tag{3}$$

The cumulative distribution function (CDF) is therefore given by:

$$Pr(S_q \leq j) = \frac{\Gamma(\lfloor k+1 \rfloor, \lambda)}{\lfloor k \rfloor!} \tag{4}$$

where $\Gamma(x, y)$ is the incomplete gamma function and $\lfloor k \rfloor$ is the floor function.

To summarize, given a binary split of *Ratings* into two sub-relations *A* and *B*, of sizes *q* and *q-k* respectively. Our proposed heuristic first computes the size of the item-intersection, $|(items(A) \bigcap items(B)| = j$. Next, we compute the probability of receiving such intersection size in a similar-size random split using the probability$Pr(S_q \leq j)$. Out of all possible splits of *Ratings*, our heuristic picks the one with the lowest probability $Pr(S_q \leq j)$ to be the next split in the tree.

## 4 Experimental Evaluation

### 4.1 Evaluation Measures

To assess the quality of the resulting recommendations list, we evaluated the *half-life utility* metric ([BHK98]) of the movies that were ranked by the user but were not used in the profile construction. The above metric assumes that successive items in the list are less likely to be viewed with an exponentially decreasing rate. The utility is defined as the difference between the user's rating for an item and the "default rating" for an item. The grade is then divided by the maximal utopian grade. Specifically, the grade of the recommendations list for user $a$ is computed as follows:

$$R_\alpha = \sum_j \frac{\max(r_{a,j} - d, 0)}{2^{(j-1)/(\alpha-1)}} \tag{5}$$

where $r_{a,j}$ represents the rating of user $a$ on item $j$ of the ranked list, $d$ is the default rating, and $\alpha$ is the viewing half-life that in this experiment was set to 10. The overall score for a dataset across all users ( $R$) is

$$R = 100 \frac{\sum_a R_a}{\sum_a R_a^{\max}} \tag{6}$$

where $R_a^{\max}$ is the maximum achievable utility if the system ranked the items in the exact order that the user ranked them.

### 4.2 Experimental Setup

For our experimental evaluation we used the MovieLens ([RL10]) data set, which holds 1 million ratings of 6040 users rating 3900 distinct movies. For the training set, about 10% of the movies were selected, and all ratings associated with them. These ratings accumulate to roughly 50,000 ratings for half the user set (3063 in number). The users' attributes consisted of age, occupation and gender. The same attribute can serve as a split attribute at multiple junctions in the tree as we confine the tree to use only binary splits. The RS is asked to provide recommendations for all remaining users (2977 in number), and the *half-life utility* metric is used to evaluate the results. In order to evaluate the new heuristic, we compare two recommendation systems identical in all details except for the split heuristic used. One system is using the standard information gain heuristic, and the other is using the proposed least probable intersection size heuristic. The same training set is given as input to both systems.

## 4.3 Results

Figure 3 compares the performance of the new intersection size heuristic criterion with the well known information gain. The horizontal axis of this plot indicates the depth of the decision tree, as the tree is being constructed. The vertical axis indicates the average utility of recommendations produced by the tree. The solid line shows the utility of the decision tree using the new criterion, whereas the broken line shows utility obtained by the information gain criterion. Both lines follow the well-known over-fitting pattern, in which the utility first increases, then decreases. Note that the new criterion dominates the information gain criterion over all depths. Specifically, the best performance of the two criteria are 75.88% and 72.71% respectively which implies a 4.35% relative improvement.

To examine the effects of the tree's depth and of the splitting criteria, a two-way analysis of variance (ANOVA) with repeated measures was performed. The dependent variable was the mean utility. The results of the ANOVA showed that the main effects of the tree's depth $F = 34.2, p < 0.001$ and the splitting criteria $F = 7.24, p < 0.001$ were both significant.

The Post-Hoc Duncan test was conducted in order to examine when the proposed criterion outperforms the information gain criterion. With $\alpha = 0.05$, starting from tree of four levels and up to the a tree of 21 levels the proposed method is significantly better than the information gain.

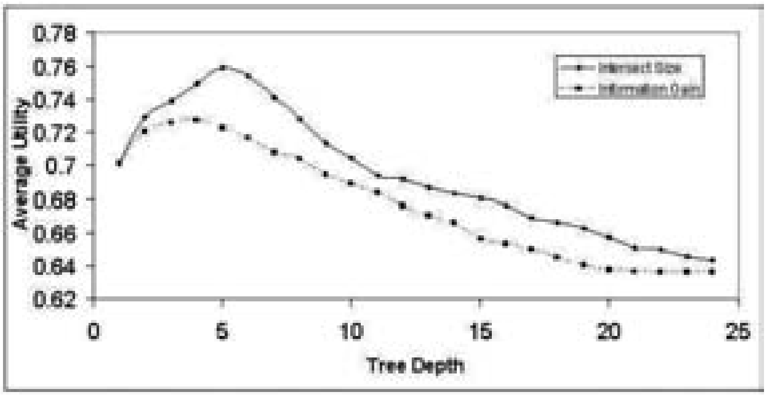

Figure 3: Comparing the average utility of the two splitting criteria on different tree's sizes.

## 5 Conclusions

A new decision tree based recommender technique was presented. The new technique requires only a single traversal of the tree for producing the recommendation list, by holding lists of recommended items in the tree's nodes.

A new splitting criterion for guiding the induction of the decision tree is also proposed. The experimental study shows that the new criterion outperforms the well known information gain criterion.

Additional issues to be further investigated include: (a) Add a pruning phase which will follow the growing phase and will help to avoid over-fitting; (b) Compare the proposed technique to other decision tree based techniques; and (c) Examine the proposed method on other benchmark datasets.

## Acknowledgements

The project was funded and managed by Deutsche Telekom Laboratories as part of the Context-Aware Service Offering and Usage (CASOU) project.

## References

[BHK98] J. S. Breese, D. Heckerman, and C. Kadie. Emperical analysis of predictive algorithms for collaborative filtering. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[BRBG08] A. Bouza, G. Reif, A. Bernstein, and H. Gall. Semtree: ontology-based decision tree algorithm for recommender systems. In *International Semantic Web Conference*, 2008.

[Cam60] L. Le Cam. An Approximation Theorem for the Poisson Binomial Distribution. *Pacific Journal of Mathematics, volume 10*, pages 1181–1197, 1960.

[LY04] P. Li and S. Yamada. A Movie Recommender System Based on Inductive Learning. In *IEEE Conference on Cybernetics and Intelligent Systems*, 2004.

[Mit97] T. M. Mitchell. *Machine Learning (The McGraw-Hill Companies, Inc.).* 1997.

[Net] NetFlix. The NetFlix prize, www.netflixprize.com.

[Qui86] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, pages 81–106, 1986.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning).* Morgan Kaufmann, January 1993.

[RL10] GroupLens Research-Lab. ”The MovieLens Data Set”, http://www.grouplens.org/node/73, February 2010.

[ZI02] T. Zhang and V. S. Iyengar. Recommender Systems Using Linear Classifiers. *Journal of Machine Learning Research, volume 2*, pages 313–334, 2002.