Induction of Compact Decision Trees for Personalized Recommendation

Daniel Nikovski
Mitsubishi Electric Research Laboratories
MERL Technology Laboratory
201 Broadway, Cambridge, USA
nikovski@merl.com

Veselin Kulev
Massachusetts Institute of Technology
Research Science Institute
77 Massachusetts Ave., Cambridge, USA
veskok@gmail.com

ABSTRACT

We propose a method for induction of compact optimal recommendation policies based on discovery of frequent itemsets in a purchase database, followed by the application of standard decision tree learning algorithms for the purposes of simplification and compaction of the recommendation policies. Experimental results suggest that the structure of such policies can be exploited to partition the space of customer purchasing histories much more efficiently than frequent itemset discovery algorithms alone would allow.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

response modeling, product recommendation, frequent itemset mining

1. INTRODUCTION

Personalized recommendation is the process of deciding which product to recommend to individual customers based on their existing purchasing history, as recorded by a vendor in the past. This capability has been made possible by the wide availability of sales databases and the advancement of computationally-intensive statistical data mining techniques. Nowadays, personal recommendation is a major feature of online e-commerce sites such as Amazon.com [5], and plays a similarly important role in the direct marketing industry, where it is used to decide which customers to mail catalogs and promotional materials to, which products to include in such catalogs, etc.

1.1 Recommendation as response modeling

From a computational point of view, the problem usually reduces to estimating the probability $Pr(A_i = True|H)$ that a given product $A_i \in \aleph$ from the product assortment $\aleph$ of a company would be purchased by an existing customer with known purchasing history H, where $H \subset \aleph$ consists of only those items that the customers have purchased in the past. The main assumption here is that past purchases correlate well with future purchases, and information about customer preferences can be extracted from the known past purchasing history of that customer. (In the usual case, all evidence is positive — if a purchase of a product A_j has not been recorded, it is assumed that $A_j = False$, even though the customer might have purchased this product elsewhere.) This task is also known as response modeling, since it seeks to model quantitatively the likelihood that a prospective customer would respond to recommendation [1].

Once the probabilities for purchasing each available product have been estimated, the optimal product to be recommended can be determined in one of several ways. The simplest method is to recommend the product $A*$ with the highest probability of purchase $A* = argmax_{A_i=True} Pr(A_i|H)$. For this recommendation to be truly optimal, three conditions must hold: first, the profit from each product should be the same; second, the customer should be making only one product choice (or further purchases should be independent of that choice); and third, the probability of purchasing each product if it is *not* recommended should be constant. In practice, these three conditions almost never hold, which gives rise to several more realistic definitions of optimal recommendation.

Varying profits $r(A_i)$ among products are easily accounted for by recommending the product $A*$ with the highest expected profit: $A* = argmax_{A_i} Pr(A_i = True|H)r(A_i)$. When the probability of purchasing each product when it is not recommended is not constant, it would be more useful to recommend the product for which the *increase* in probability due to recommendation is the greatest. This would require separate estimation of customer response for the two cases when a product was recommended and the alternative case when it was not. Departures from the third condition can be dealt with, too, by solving a sequential Markov decision process (MDP) model that optimizes the *cumulative* profit resulting from a recommendation rather than the immediate profit [3] — although a lot more involved technically, this scenario also reduces to response modeling, since profit from individual products and transition probabilities are all that is required to specify an MDP.

1.2 Estimation of response probabilities

It would always be possible to infer $Pr(A_i = True|H)$ for any A_i and H, if the joint probability function (JPF) over all Boolean variables A_i, $i = 1, N$, $N = |\aleph|$ were known:

$$Pr(A_i = True|H) = \frac{Pr(A_i \cup H)}{Pr(H)}$$

where $Pr(A_i \cup H)$ and $Pr(H)$ can be read off the JPF. In practice, the JPF will not be known, and must be determined by means of a suitable computational method instead. When an existing purchase database is used for the estimation of the JPF, this reduces to the usual problem of density estimation, and is amenable to analysis by many existing data mining algorithms. Narrowly in the field of personalized recommendation, this approach is also known as collaborative filtering, because it leverages the recorded preferences and purchasing patterns of an existing group of customers to make recommendations to that same group of customers [6, 13].

From the point of view of data mining and statistical machine learning, however, direct estimation of each and every entry of the JPF of a product domain is usually infeasible, for at least two reasons. First, there are exponentially many such entries, and the memory requirements for their representation grow exponentially with the size of the product assortment N. Second, even if it were somehow possible to represent all atomic entries of the JPF in memory, their values could not be estimated reliably by means of frequency counting from a purchase database, unless the size of this database also grew exponentially in N. However, the size of the available purchase database is usually linear in the time a vendor has been in business, rather than exponential in the size of its product assortment.

The usual method to deal with this problem is to impose some structure on the JPF. One particularly popular solution involves logistic regression, which has been dubbed "the workhorse of response modeling" [1]. The problem with logistic regression is that it fails to model the interactions among variables in the purchasing history H, and considers individual product influences independently. A significant improvement can usually be realized by the use of more advanced data mining techniques such as neural networks, decision trees, support-vector machines, or practically any other machine learning method for building classifiers. Although this idea has had practical impact on products, in particular the induction of dependency networks [15], it depends critically on progress in induction of classifiers on large databases, which is by no means a solved problem.

In this paper, we are pursuing a very different approach that is based on discovery of frequent itemset (FI) lattices, and subsequent extraction of direct compact recommendation policies expressed as decision trees. The main contribution of this paper is the idea that well-known algorithms for induction of decision trees can be leveraged to simplify considerably the optimal recommendation policies discovered by means of frequent itemset mining, and the experimental verification of this idea on a real data set of customer purchases.

Section 2 reviews how discovered FI lattices can be used for personalized recommendation, and section 3 discusses our idea for inducing compact decision trees from the FI lattices. Section 4 provides an experimental verification of the idea, and demonstrates that it indeed results in substantial reductions in the size and complexity of the recommendation policies. Section 5 concludes and presents some suggestions on how this idea can be extended to more sophisticated recommendation policies.

2. FREQUENT ITEMSET DISCOVERY

The problem of frequent itemset discovery is one of the most studied problems in the field of data mining, and our approach to personalized recommendation aims to leverage the existing solutions to this problem.

Let us consider first the typical information stored in sales databases. For example, let $T = \{A, B, C, D\}$ be a set of items in a store. The store also has a database with past transactions $\mathcal{T}$. Entries in $\mathcal{T}$ are (ID,item set) pairs (see Table 1).

Database ID	Item-set
100	{A,B,D}
200	{A,B}
300	{C,D}
400	{B,C}

Table 1: Example database $\mathcal{T}$

The *support* supp(X) of an item set $X \subseteq \mathcal{T}$ is the number of purchases Y in $\mathcal{T}$ such that $X \subseteq Y$. An item set $X \subseteq T$ is *frequent* if its support is greater than or equal to a user-defined threshold θ. Table 2 shows all frequent item-sets in $\mathcal{T}$ with a threshold $\theta = 1$.

Itemset	Cover ID	Support
{}	{100,200,300,400}	4
{A}	{100,200}	2
{B}	{100,200,300}	3
{C}	{300,400}	2
{D}	{100,300}	2
{A,B}	{100,200}	2
{A,D}	{100}	1
{B,C}	{400}	1
{B,D}	{100}	1
{C,D}	{300}	1
{A,B,D}	{100}	1

Table 2: Itemsets and their support in $\mathcal{T}$.

Before we describe how itemsets can be used for personalized recommendation, we will discuss the concept of an *adjacency lattice* of itemsets. We use a directed acyclic graph to visualize the adjacency lattice (see Figure 1) for all possible itemsets in T. A set of items X is adjacent to another set of items Y if and only if Y can be obtained from X by adding a single item. In the graph we use, X is the parent and Y is the child.

It is not hard to see that an adjacency lattice is one particular way of organizing all subsets of items in a store, which differs from other, alternative methods (such as N-way contingency tables, for example) in its progression from small subsets to large subsets. In particular, all subsets in the same level of the lattice have the same cardinality.

Normally, if we want to represent the full JPF of a problem domain, we can use this adjacency lattice to store the probabilities of each subset, and this representation will be

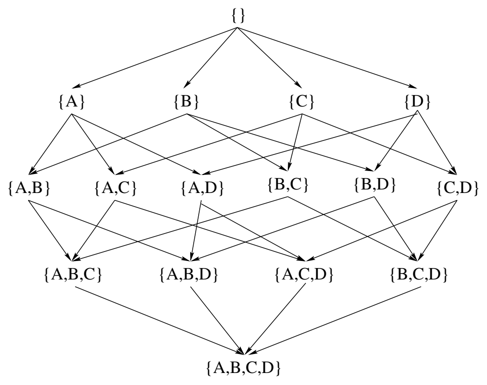

Figure 1: Adjacency lattice for T

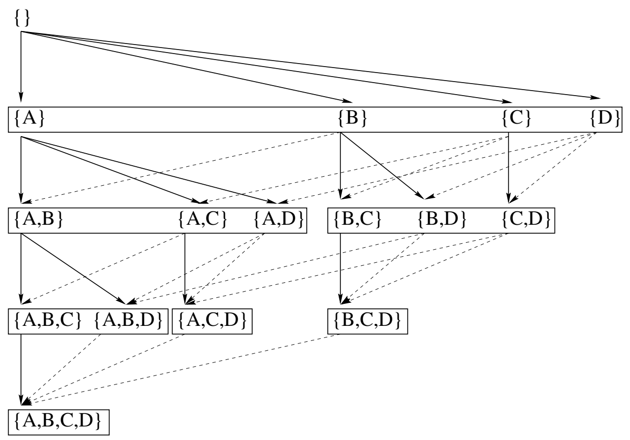

Figure 2: Prefix tree representation of a lattice — the missing edges are shown in dashed lines.

neither substantially more efficient, nor substantially less efficient than a contingency table, because it still stores the probabilities of all atomic events.

However, we can realize major economies in terms of memory requirements if we store only those subsets whose probabilities are above a given threshold. Such subsets of items are called frequent itemsets, and an active sub-field of data mining is concerned with efficient algorithms for frequent itemset mining (FIM). Given a threshold, these algorithms find all itemsets whose support exceeds the threshold, and record for each of them the exact number of transactions that support it. Note that this representation is not lossless: by storing only frequent itemsets and discarding less frequent ones, we are trading the accuracy of the JPF for memory compactness.

One of the first algorithms for mining frequent item sets was the *Apriori* algorithm[14]. This algorithm creates an adjacency lattice for a given transaction database $\mathcal{T}$ and threshold value θ. The algorithm operates by creating first all frequent item-sets X where $|X| = 1$, then building all frequent itemsets Y where $|Y| = 2$, and so on. After every itemset generation, the algorithm runs a pruning function to delete the itemsets with support lower than the threshold θ. The threshold is chosen so that all frequent itemsets can fit in memory. Note that while the full JPF of a problem domain would typically not fit in memory, we can always make the frequent itemset lattice fit in the available memory by raising the support threshold. (Certainly, the lower the threshold, the more complete the JPF.)

Once a (sparse) FI lattice has been extracted in memory, it can be used to define a recommendation policy much like a full JPF could be used, with some provisions for handling missing entries. The easiest case is when the itemset H corresponding to the purchasing history of a customer is represented in the lattice, and at least one of its descendants in the lattice is also present. Then, the optimal recommendation is the extension $A = Q \setminus H$ of the set H that maximizes the support of the direct descendants Q of H in the lattice. (By definition, the descendant FIs of H in the lattice differ from H by only one element, which makes search for optimal recommendations very easy.) Note that only the *existing* descendant FIs have to be examined in order to find the optimal recommendation — if all other possible descendants are not frequent, then their support must be below that of the frequent itemsets, so the extensions leading to them could not possibly be optimal.

A more complicated case occurs when the complete purchasing history H is not an FI. There are several ways to deal with this case, but before describing them, we should note that they are by far not as important as the main case described above, because they happen infrequently. Still, one reasonable approach is to find the largest subset of H that is frequent and has at least one frequent descendant, and use the optimal recommendation for that largest subset. (In practice, the algorithm would find the largest frequent subset present in the lattice, and use the optimal recommendation for its parent.) In case several largest subsets of the same cardinality exist, ties can be broken randomly, or more sophisticated algorithms for accommodating several local models into one global can be used [9].

The extraction of the optimal recommendation has to be performed only once, and after that it can be stored in the lattice, together with the support of that set. Table 3 shows the recommendations extracted form the lattice for every item-set with a minimum support threshold of 1. We will call the mapping from past purchases to optimal products to be recommended a *recommendation policy*. As discussed in the Introduction, this definition of optimality corresponds to the simplest objective of product recommendation, namely the maximization of the probability of the immediately following purchase. However, any of the more elaborate formulations of optimality discussed above can be used here. They would result in different recommendation policies, which are nevertheless of the same form: a mapping from purchasing histories to products to be recommended.

As an implementation detail, we will note that the lattice is usually stored as a *prefix tree* which does not represent all the lattice edges explicitly [7]. The missing edges are displayed as dashed lines in Figure 2. For example, the set $\{A, B, C\}$ will be a parent to the set $\{A, B, C, D\}$ but $\{B, C, D\}$ will not be a parent to the $\{A, B, C, D\}$ set. The set $\{A, B, C, D\}$ is called an indirect child to the set $\{B, C, D\}$. Searching indirect children, however, is not a major problem — in practice, the algorithm would generate in turn all possible extensions, use the prefix tree to find the corresponding itemset, and consider it in determination of the optimal policy, if it is frequent.

Before discussing our idea for representation and compaction of the recommendation policy by means of decision trees, we will note that the approach to personalized recom-

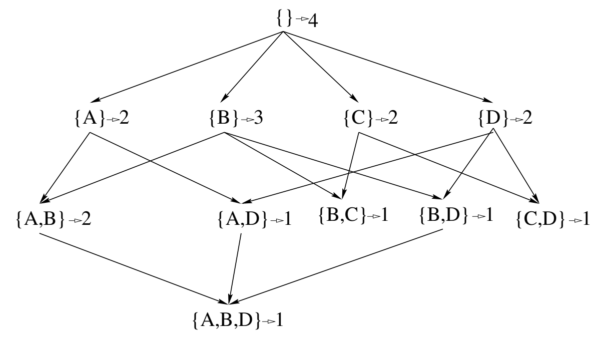

Figure 3: Adjacency lattice for T and the support values of the item-sets for $\mathcal{T}$.

Itemset		Recommendation	Purchase Prob.
{}	⇒	{B}	0.75
{A}	⇒	{B}	1.00
{B}	⇒	{A}	0.66
{C}	⇒	{B} or {D}	0.50
{D}	⇒	{A} or {C}	0.50
{A,B}	⇒	{D}	0.50
{A,D}	⇒	{B}	1.00
{B,C}	⇒	{}	1.00
{B,D}	⇒	{A}	1.00
{C,D}	⇒	{}	1.00
{A,B,D}	⇒	{}	1.00

Table 3: Recommendation policy derived from $\mathcal{T}$.

mendation based on frequent itemsets has close connections to personalized recommendation based on association rules, as proposed by Mobasher [11] and Lin et al. [10]. Those authors proposed to derive association rules of the form "If H then y with probability P", match the antecedents of all rules to a customer's purchasing history, and use the most specific rule to estimate the probabilities of product purchases. (Or, for the last step, use some other arbitration mechanism to resolve conflicting rules.) These algorithms are not essentially (mathematically) different from our method, since the discovery of association rules is also based on the same algorithms for discovery of frequent itemsets that we discussed, and the induced JPF is essentially the same. Note that our objective in this paper is not to improve on the accuracy of these algorithms in estimating the customer response probabilities, nor to compare the accuracy of FI-based recommenders with that of alternative methods based on logistic regression, neural nets, etc. Our objective here is to to improve on the time and space required to store and producde optimal recommendations derived by means of discovery of frequent itemsets.

The motivation for this objective is the observation that these algorithms are fairly inefficient in matching customer histories to rules, since the whole rule base has to be searched sequentially. (Unless additional data structures are used, but it is not likely that they would be any simpler than a prefix tree.) In contrast, search in an adjacency matrix represented by a prefix tree is only logarithmic in the number of itemsets represented in it. Furthermore, general algorithms for induction of association rules generate far too many rules — while there are 2^N itemsets in a domain, there are 3^N possible association rules, which makes a big difference in memory requirements.

However, a recommendation policy stored in the lattice has its inconveniencies, too. First, it is not very portable — unlike sets of association rules, which can be stored and exchanged in the Predictive Model Markup Language (PMML), there is no convenient PMML representation of a prefix tree or adjacency lattice. Second, and even more important, the lattice encodes a (sparse) JPF, while what we need is only the recommendation policy. (This shortcoming is shared with the mentioned algorithms for personalized recommendation based on association rules.)

In fact, it can be expected that large discrepancies might exist between the complexity of a JPF and the complexity of the optimal recommendation policy implied by that JPF. As an example, let's consider a domain of N products whose purchases are completely uncorrelated. Still, not knowing this, the JPF will have on the order of 2^N entries. (Representing only frequent itemsets will help us sparsify their representation, but if their individual purchase frequencies are similar, this would not help us a lot.) On the other hand, the optimal recommendation policy in this case is very simple, because past purchasing histories have no correlation to future purchases, and the optimal strategy is to recommend the most popular item not already owned by the customer — if a customer has not purchased the most popular item, recommend it, otherwise if the customer has not purchased the second most popular item, recommend it instead, and so on until the least popular item, to be recommended to a customer who already has purchased everything else. Clearly, such a recommendation policy is only linear in N, while the JPF of the problem domain is exponential in N.

While this is an extreme constructed example, and inter-item correlations certainly do exist in real purchasing domains (otherwise the whole idea of personalized recommendation would be futile!), our hypothesis is that this discrepancy between the complexity of the JPF and that of the recommendation policy still exists in real domains to a large extent. The question then becomes how to discover and exploit it, and the next section describes how this can be achieved by means of decision trees.

3. EFFICIENT INDUCTION OF DECISION TREES FROM FREQUENT ITEMSET LATTICES

Decision trees are a very popular data mining method for classification and regression, and can be conveniently induced, exchanged, and visualized by many tools. A decision tree consists of intermediate nodes, where attributes (variables) are tested, and leaves, where decisions are stored. Since a recommendation policy is a mapping between purchasing histories (inputs) and optimal product recommendations (output), a decision tree is a perfectly viable structure for representing a recommendation policy.

If we want to represent a recommendation policy as a decision tree, the easiest approach is to convert directly the prefix tree of the adjacency lattice to a decision tree. To this end, each node of the prefix tree that has n descendants must be represented as n binary nodes, testing in sequence whether the customer has purchased each of the corresponding n items that label the edges leading to the descendants. Clearly, if this approach is followed, the resulting decision tree would be much larger than the original lattice.

Instead, our approach is to treat the problem of encoding the recommendation policy as a machine learning problem of its own right, and address it with existing algorithms for induction of decision trees, such as ID3 and C4.5. Our expectation is that the optimal partitioning of the itemset space for the purposes of representing the recommendation policy is very different from the optimal partitioning of that space for the purposes of storing the JPF of purchasing patterns, and existing algorithms for induction of decision trees would be able to discover the former partition.

In order to use these algorithms for induction of decision trees, we generate training examples directly from the recommendation policy — one example per itemset in the lattice. Each frequent itemset is represented as a complete set of Boolean variables, which are used as input variables. The optimal product to be recommended is given as the class label of the output. Table 4 shows an example data transformation. We use this list of itemsets and recommendations as training examples for building a decision tree.

There are certainly many possible decision trees that classify correctly a given set of training examples, and some of them are larger than others. For example, if we are given the examples in Table 4, a possible decision tree may be the one in Figure 4. However, this tree is rather large — the one in Figure 5 is just as good, and is significantly smaller.

While finding the most compact decision tree is not a trivial problem, our approach is to use greedy algorithms such as ID3 and C4.5 ([2, 4]) that have been shown time and again to produce very compact decision trees with excellent classification properties. So, after we generate training examples as described above, we rely on these general algorithms for induction of decision trees to build a compact representation of the recommendation policy.

For our example, we end up with 11 nodes in the adjacency lattice and 11 nodes in the decision tree, which means that we have not achieved reduction of the number of nodes. However, the experimental comparison results described below showed that on larger datasets, our algorithm performs better in terms of number of nodes, and creates simpler data structures represented with decision trees compared to the lattice representation for the same data.

4. EXPERIMENTAL VERIFICATION

We tested our algorithm and hypothesis on the *retail* data set that is commonly used in research on frequent itemset mining [8]. The database originates from an automated convenience store in Belgium, and contains 41, 373 records. In this evaluation, we used the implementation of Apriori due to Bart Goethals [7]. After generating training examples, decision trees were built within the MineSet algorithmic environment. During decision tree induction, split attributes were selected using the Mutual Information (entropy) criterion. In all cases, completely homogeneous trees were built — this is always possible, since each training example has unique input.

Figure 6 shows a comparison between the number of nodes in the prefix tree and that of the nodes and leaves of the decision tree, both plotted against the support threshold. The experimental results suggest that indeed the use of decision trees indeed results in more compact recommendation policies. Furthermore, the percentage savings are not constant, but increase with the size of the policy. In some cases, the decision tree learning algorithm is able to reduce the number of nodes necessary to encode the policy by up to 80%. This shows that there is indeed significant structure in the discovered recommendation policy, and the learning algorithm was able to discover it.

Moreover, we will note that storing a binary decision tree is much cheaper than storing a prefix tree with the same number of nodes, because, in general, the prefix tree is not binary. Furthermore, a decision tree can be transported in standard PMML format. And, finally, the induced tree handles all novel customers directly, even those whose full purchasing histories are not explicitly represented in the original lattice.

5. CONCLUSION

In this paper, we discussed the application of frequent itemset discovery algorithms for product recommendation and personalization technology, and a method for compaction of the derived recommendation policy by means of standard decision tree induction algorithms. Since the adjacency matrix of all frequent item sets consumes a lot of memory and results in relatively long look-up times, we investigated the idea to compress the recommendation policy by means of a decision tree. To this end, several algorithms for learning decision threes were applied to a training set consisting of all recommendations encoded in the adjacency matrix. We discovered that decision trees indeed resulted in more compact recommendation policies.

Future research in this area will focus on more advanced algorithms for building decision trees. It can be expected that alternative, problem-specific measures for introducing splits in the decision trees would perform better than the traditional measures such as entropy, information gain, the Gini index, etc., which were originally developed for the purposes of classification by decision trees. We believe that further progress would depend critically on gaining further insight on the nature and structure of optimal recommendation policies for real data sets.

Another possible future direction is to try and apply this approach to more sophisticated recommendation policies, for example ones based on the extraction of frequent sequences. Such policies have been shown to model the sequential nature of customer choice significantly better than atemporal associations [11], and since the discovery of frequent sequences has been shown to be not much more difficult than the discovery of frequent itemsets, it can be expected that the adjacency lattice of frequent sequences can be compressed similarly to that of frequent itemsets. However, such an extension is by no means trivial, since the decision tree would have to use attributes that not only test whether an item has been purchased in the past by a given customer, but also whether this item has been purchased before or after other relevant items. Still, it can be expected that our approach could be generalized to sequential recommendation policies, possibly at the expense of considering more complex decision trees.

6. REFERENCES

[1] Bruce Ratner (2003). *Statistical Modeling and Analysis for Database Marketing*. Boca Raton: Chapman and Hall/CRC.

[2] J.R. Quinlan (1986). Induction of decision trees. *Machine Learning*, 1(1), pp. 81-106.

Item-set		Rec.
{}	⇒	{B}
{A}	⇒	{B}
{B}	⇒	{A}
{C}	⇒	{B} or {D}
{D}	⇒	{A} or {C}
{A,B}	⇒	{D}
{A,D}	⇒	{B}
{B,C}	⇒	{}
{B,D}	⇒	{A}
{C,D}	⇒	{}
{A,B,D}	⇒	{}

⇒

A	B	C	D	Rec.
No	No	No	No	{B}
Yes	No	No	No	{B}
No	Yes	No	No	{A}
No	No	Yes	No	{B} or {D}
No	No	No	Yes	{A} or {C}
Yes	Yes	No	No	{D}
Yes	No	No	Yes	{B}
No	Yes	Yes	Yes	{}
No	Yes	No	Yes	{A}
No	No	Yes	Yes	{}
Yes	Yes	No	Yes	{}

Table 4: Recommendation tables for T and ID3 input table for T

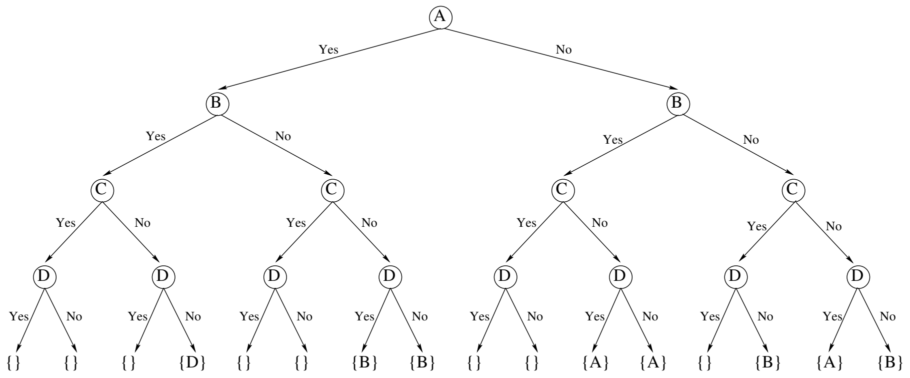

Figure 4: Full decision tree for T.

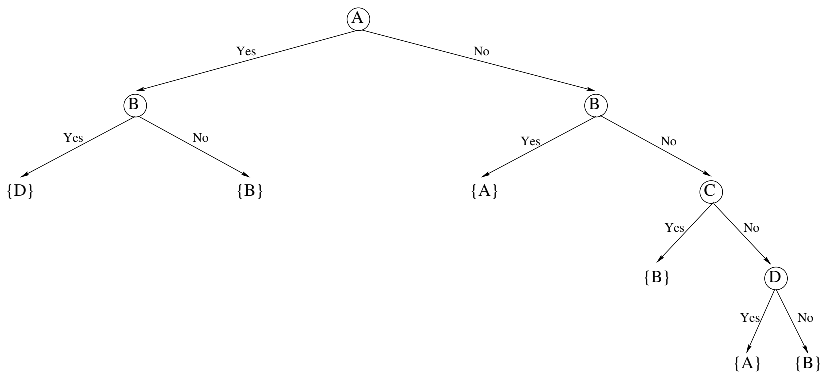

Figure 5: Optimized decision tree for T.

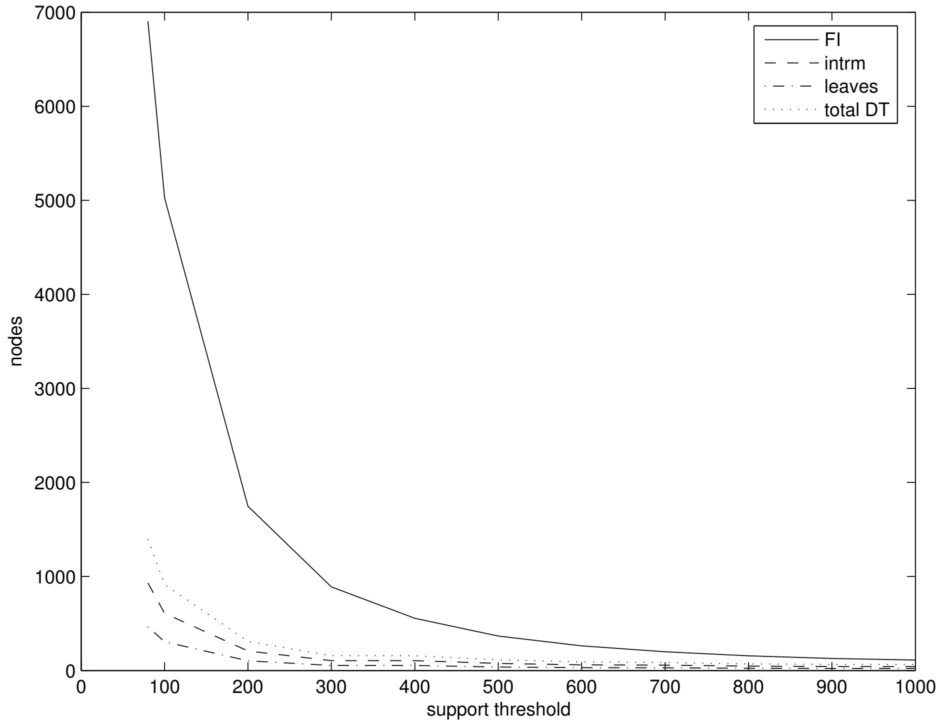

Figure 6: Experimental comparison between the number of nodes needed to store the recommendation policy in a prefix tree ("FI") or a decision tree ("total DT"). For the case of decision trees, the nodes are broken down into intermediate (decision) nodes, denoted by "intrm", and leaves (recommendations), denoted by "leaves". It can be argued that leave nodes should not even be counted, because recommendations recorded in them can easily be recorded in their immediate parents, instead, and comparison should be between "FI" and "intrm".

[3] Brafman, R.I., Heckermann, D., and Shani, G. (2003). Recommendation as a stochastic sequential decision problem, in *Proc. of ICASP 13*, Trento (Italy), June 6-10, 2003, pp. 164–173.

[4] J.R. Quinlan (1993). *C4.5: Programs for Machine Learning* San Mateo, CA: Morgan Kaugmann.

[5] Greg Linden, Brent Smith, and Jeremy York (2003). Amazon.com recommendations: item-to-item collaborative filtering, *Industry Report*, March, 2003.

[6] Pattie Maes (1994). Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30-40 July 1994.

[7] Bart Goethals (2002). *Efficient Frequent Pattern Mining*, PhD thesis, December ,2002, Transnational University of Limburg, Diepenbeek, Belgium.

[8] Brijs T., Swinnen G., Vanhoof K., and Wets G. (1999). The use of association rules for product assortment decisions: a case study, in *Proc. of the Fifth International Conference on KDD*, San Diego, August 15-18, 1999, pp. 254-260.

[9] H. Mannila, D. Pavlov, and P. Smyth. (1999). Predictions with local patterns using cross-entropy, in *Proc. of Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 357–361, New York, ACM Press, 1999.

[10] W. Lin, S. A. Alvarez, and C. Ruiz (2002). Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery*, 6(1): 83–105, 2002.

[11] Mobasher, B., Dai, H., Luo, T., and Nakagawa, M. (2001). Effective personalization based on association rule discovery from web usage data. In *Proc. of the 3rd International Workshop on Web information and Data Management*, Atlanta, Georgia, ACM Press, New York, pp. 9–15.

[12] Charu C.Aggarwal and Philip S. Yu (1994). *Online Generation of Association Rules* IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

[13] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and Jonh Riedl (1999). An Algorithmic Framework for Performing Collaborative Filtering, *Proc. of SIGIR 1999*, pp.230–237.

[14] Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in very large databases (1993). *Proc. of the ACM SIGMOD Conference on Management of data*, pp. 207–216, Washington D.C., May 1993.

[15] Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., and Kadie, C. (2001). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, vol. 1, Sep. 2001, pp.49–75.