Postprocessing Decision Trees to Extract Actionable Knowledge

Qiang Yang and Jie Yin
Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Kowloon Hong Kong
{qyang, yinjie}@cs.ust.hk

Charles X. Ling and Tielin Chen
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
{ling, tchen}@csd.uwo.ca

Abstract

Most data mining algorithms and tools stop at discovered customer models, producing distribution information on customer profiles. Such techniques, when applied to industrial problems such as customer relationship management (CRM), are useful in pointing out customers who are likely attritors and customers who are loyal, but they require human experts to postprocess the mined information manually. Most of the postprocessing techniques have been limited to producing visualization results and interestingness ranking, but they do not directly suggest actions *that would lead to an increase the objective function such as profit. In this paper, we present a novel algorithm that suggest actions to change customers from an undesired status (such as attritors) to a desired one (such as loyal) while maximizing objective function: the expected net profit. We develop these algorithms under resource constraints that are abound in reality. The contribution of the work is in taking the output from an existing mature technique (decision trees, for example), and producing novel, actionable knowledge through automatic postprocessing.*

1. Introduction

Extensive research in data mining has been done on discovering distributional knowledge about the underlying data. Models such as the Bayesian models, decision trees, support vector machines and association rules have been applied to various industrial applications such as customer relationship management (CRM) [6, 1]. Despite such phenomenal success, most of these techniques stop short of the final objectives of data mining, such as maximizing profit while reducing costs, relying on such postprocessing techniques as visualization and interestingness ranking [2, 3, 5]. While these techniques are essential to move the data mining result to the eventual application, they nevertheless require a great deal of human manual work by experts.

In this paper, we present a novel postprocessing technique to mine actionable knowledge from decision trees. To illustrate our techniques, we focus on the application in CRM. Like most data mining algorithms today, a common problem in current applications of data mining in intelligent CRM is that people tend to focus on, and be satisfied with, building up the models and interpreting them, but not to use them to get profit explicitly. This knowledge is useful but it does not directly benefit the enterprise. To improve customer relationship, the enterprise must know what *actions* to take to change customers from an undesired status (such as attritors) to a desired one (such as loyal customers). To our best knowledge, no data mining algorithms have been made widely available to accomplish this important task in intelligent CRM.

In this paper, we present novel algorithms for postprocessing decision trees in order to extract *actions* to maximize a profit-based objective function. We consider two cases, one corresponds to unlimited resources, and the other limited resource constraints. In both cases, our aim is to maximize the expected net profit of all the customers. We show that finding the optimal solution for the limited resource problem is NP-complete, and design a greedy heuristic algorithm to solve it efficiently. We compare the performance of the exhaustive search algorithm with a greedy heuristic algorithm, and show that the greedy algorithm is efficient while achieving results with quality very close to the optimal ones.

2. Action Mining in Decision Trees under Unlimited Resources

Given a decision tree as input, our `leaf-node search` algorithm searches for optimal actions to transfer each leaf node to another leaf node with a higher probability of being in a more desirable class. After a customer profile is built in the form of a decision tree, the resulting decision tree can be used to classify, and more importantly, provide probability of customers in the desired status such as being loyal or high-spending. When a customer, who can be either an training example used to build the decision tree or an unseen testing example, falls into a particular leaf node with a certain probability of being in the desired status, the algorithm tries to "move" the customer into other leaves with higher probabilities of being in the desired status. The probability gain can then be converted into an expected gross profit.

However, moving a customer from one leaf to another means some attribute values of the customer must be changed. This change, in which an attribute A's value is transformed from v_1 to v_2, corresponds to an action. These actions incur costs. The cost of all changeable attributes are defined in a cost matrix by a domain expert. The `leaf-node search` algorithm searches all leaves in the tree so that for every leaf node, a best destination leaf node is found to move the customer to. The collection of moves are required to maximize the net profit, which equals the gross profit minus the cost of the corresponding actions.

Based on a domain-specific cost matrix for actions, we define the net profit of an action to be as follows.

$$P_N = P_E \times P_{gain} - \sum_i COST_i \tag{1}$$

where P_N denotes the net profit, P_E denotes the total profit of the customer in the desired status, P_{gain} denotes the probability gain, and $COST_i$ denotes the cost of each action involved.

The `leaf-node search` algorithm for searching the best actions can thus be described as follows:

Algorithm `leaf-node search`

1. For each customer c, do
2. Let L be the leaf node in which c falls into;
3. Let S be a leaf node for c the maximum net profit P_N;
4. Output (L, S, P_N);

To illustrate, consider an example shown in Figure 1, which represents an overly simplified, hypothetical decision tree as the customer profile of loyal customers built from a bank. The tree has five leaf nodes (A, B, C, D, and E), each with a probability of customers' being loyal. The probability of attritors is simply 1 minus this probability.

Consider a customer Jack who has the following attributes: Service (service level) = L (low), Sex = M (male), and Rate (mortgage rate) = L. The customer is classified by the decision tree. Clearly, Jack falls into the leaf B, which predicts that Jack will have only 20% chance of being loyal (or Jack will have 80% chance to churn in the future). The algorithm will now search through all other leaves (A, C, D, E) in the decision tree to see if Jack can be "replaced" into a best leaf with the highest net profit.

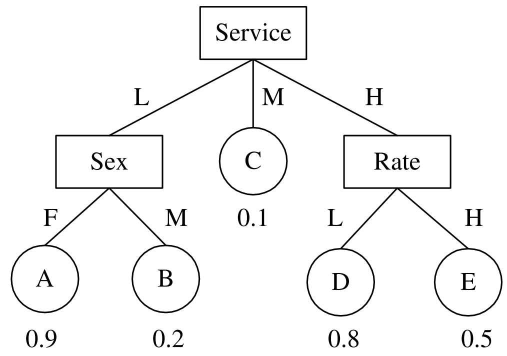

Figure 1. An example of customer profile

1. Consider leaf A. It does have a higher probability of being loyal (90%), but the cost of action would be very high (Jack should be changed to female), so the net profit is a negative infinity.

2. Consider leaf C. It has a lower probability of being loyal, so the net profit must be negative, and we can safely skip it.

3. Consider leaf D. There is a probability gain of 60% $(80\% - 20\%)$ if Jack falls into D. The action needed is to change Service from L (low) to H (high). Assume that the cost of such a change is \$200 (given by the bank). If the bank can make a total profit of \$1000 from Jack when he is 100% loyal, then this probability gain (60%) is converted into \$600 (1000×0.6) of the expected gross profit. Therefore, the net profit would be \$400 $(600 - 200)$.

4. Consider leaf E. The probability gain is 30% $(50\% - 20\%)$, which transfers to \$300 of the expected gross profit. Assume that the cost of the actions (change Service from L to H and change Rate from L to H) is \$250, then the net profit of moving Jack from B to E is \$50 $(300 - 250)$.

Clearly, the node with the maximal net profit for Jack is D, with suggested action of changing Service from L to H.

In our discussion so far, we assume that attribute-value changes will incur costs. These costs can only be determined by domain knowledge and domain experts. For each attribute used in the decision tree, a cost matrix is used to represent such costs. In many applications, the values of many attributes such as sex, address, number of children cannot be changed with any reasonable amount of money. Those attributes are called "hard attributes". For hard attributes, users must assign a very large number to every entry in the cost matrix.

If, on the other hand, some value changes are possible with reasonable costs, then those attributes such as the Service level, interest rate, promotion packages are called “soft attributes”. Note that the cost matrix needs not to be symmetric. One can assign $200 as the cost of changing service level from low to high, but infinity (a very large number) as the cost from high to low, if the bank does not want to “degrade” service levels of customers as an action.

One might ask why hard attributes should be included in the tree building process in the first place, since they can prevent customers from being moved to other leaves. This is because that many hard attributes are important in accurate probability estimation of the leaves. When the probability estimation is inaccurate, the reliability of the prediction would be low, or the error margin of the prediction would be high.

3. Post-processing Decision Trees: the Limited Resource Case

Our previous case considered each leave node of the decision tree to be a separate customer group. For each such customer group, we were free to design actions to act on it in order to increase the net profit. However, in practice, a company may be limited in its resources. For example, a mutual fund company may have a limited number k (say three) of account managers, each manager can take care of only one customer group. Thus, when such limitations exist, it is a difficult problem to merge all leave nodes into k segments, such that each segment can be assigned to an account manager. To each segment, the responsible manager can apply actions to increase the overall profit.

This limited resources problem can be formulated as a precise computational problem. Our input is as follows:

1. a decision tree built from training examples
2. a pre-specified constant k ($k \leq m$)
3. a set of testing examples

Given these, our problem is to find k customer groups and their corresponding action sets from a decision tree such that when these actions are applied to the corresponding group in testing examples, the total net profit is maximized.

To illustrate, consider an example in Figure 2. Assume that for leaf nodes L_1 to L_4, the probability values of being in the desired class are 0.9, 0.2, 0.8, 0.5, respectively. Now consider the task of transforming all leaf nodes from a lower probability value node to a higher one, such that the net benefit of such transformation is maximized. To illustrate this point, consider a test data set such that there is exactly one member that falls in each leaf node in this decision tree.

In order to calculate the net profit, we assume all leaf nodes to have a profit of one. We also assume that the cost of transferring a customer is equal to the number of attribute value changes multiplied by 0.1. Thus, to change from L_2 to L_1, we need to modify the value of the attribute `Status`, with a profit gain of $(0.9 - 0.2) \times 1 - 0.1 = 0.6$.

To illustrate the limited resources problem, consider again our decision tree in Figure 2. Suppose that we wish to find one single customer segment ($k = 1$). One such group is $\{L_2, L_4\}$, with a selected action set $\{Service \leftarrow H, Rate \leftarrow C\}$ which can be applied on it. Assume L_2 and L_4 only contain one example. If we consider transferring this group to leaf node L_3, L_2 has a profit gain of $(0.8 - 0.2) \times 1 - 0.2 = 0.4$ and L_4 has a profit gain of $(0.8 - 0.5) \times 1 - 0.1 = 0.2$. Thus the net benefit for this group is $0.2 + 0.4 = 0.6$.

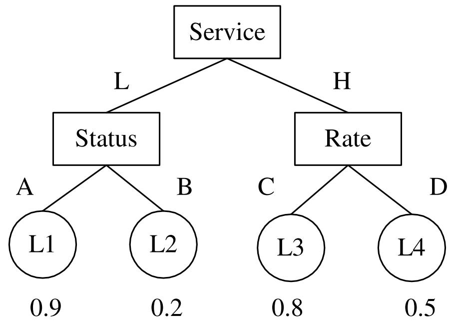

Figure 2. An example decision tree

How difficult is it to find an optimal solution for any value of k? Below, we show that when there is limited resources, the problem NP-Complete to solve. Because this problem requires that the number of customer segments to be limited, we call it the bounded segmentation problem (BSP).

To solve the problems optimally, it requires an exponential search. To avoid the computational complexity, we have developed a greedy algorithm which can reduce the computational cost but also guarantee the quality of the solution at the same time. Consider the following generalization of the maximum coverage problem. Given a set with m leaf nodes $L_s = \{L_1, L_2, \ldots, L_m\}$, each associated with a different profit P_{ij} ($P_{ij} \geq 0$) for each action set A_j, $1 \leq j \leq n$. Each A_j can be denoted as a subset of L_s which only contains the covered leaf nodes L_i for $P_{ij} \neq 0$, $1 \leq i \leq m$. The goal is to choose k action sets so as to maximize the net profit of covered leaf nodes. We can solve this problem using a greedy algorithm which is a variation of a set covering algorithm. We call it the Greedy-BSP algorithm below.

We evaluated our algorithm using a dataset from an insurance company in Canada. It consists of over 25,000 records for customers who have the status of “stay” or “leave” the insurance company. We will refer to them as positive and negative respectively. The dataset is described by over 60 attributes, many of which are not hard attributes.

About 20 attributes are soft attributes with reasonable costs for value changes.

Since the data distribution in the training set is highly unbalanced, we first perform data sampling with the ratio of positive and negative examples is about one to one [4], in order to prevent the decision tree from predicting all the customers to be negative. In this setting, we have built a decision tree with 153 leaf nodes. 87 of them are considered as negative leaf nodes because their probability of being positive is less than 50%, while the other 66 as positive leaf nodes. Furthermore, we have constructed a cost matrix for each attribute contained in the dataset according to their semantics in the real domain. The testing set consists of about 300 examples. We use the built decision tree to classify them into corresponding leaf nodes.

To compare the performance of the Greedy-BSP algorithm and the Optimal-BSP algorithm, we have again carried out experiments on the dataset from the insurance company. We apply Greedy-BSP and Optimal-BSP respectively to find the pre-specified k action sets with maximal net profit.

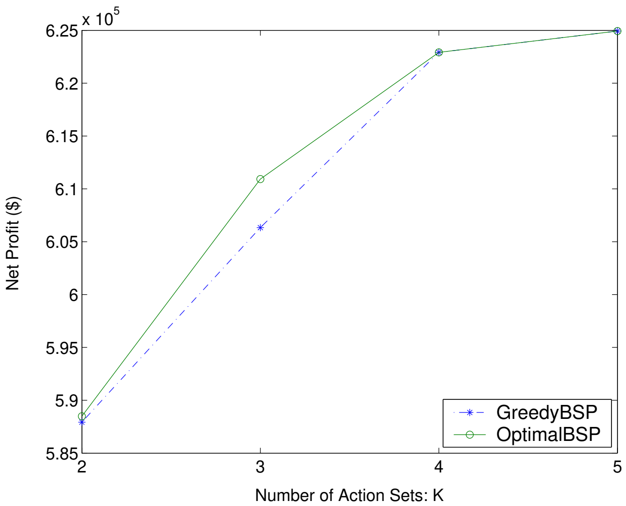

Figure 3. Net Profit vs. Number of action sets

Figure 3 shows the net profit obtained by the two algorithms over the number of action sets k. As shown in the figure, the net profit increases for both Greedy-BSP and Optimal-BSP with increasing number of action sets k. This is because if more customers are transformed to a desired status, it is more possible to obtain higher profit. In addition, an important property to note is that, for a specific k, the net profit obtained by Greedy-BSP is very close to or the same as that by Optimal-BSP, which can guarantee the quality of solution provided by Greedy-BSP.

For Greedy-BSP, The runtime is always around 0.20 seconds irrespective of the number of action sets k. On the other hand, the runtime for Optimal-BSP increases exponentially with the increasing number of action sets k. This is because it needs to compare much more combinations for larger k in order to obtain maximal net profit.

We conclude from our experiments that, Greedy-BSP can find k action sets with maximal net profit, which is very close to those found by Optimal-BSP, at least for small values of k for which Optimal-BSP terminates in a reasonable amount of time. At the same time, Greedy-BSP can scale well with the increasing number of action sets k, which is more efficient than Optimal-BSP.

4. Conclusions and Future Work

Intelligent CRM improves customer relationship from the data about customers. This paper contributed to the data mining field by considering mining the best k action groups to maximize the net profit. We formulated two versions of the problem and described a greedy algorithm that efficiently discovers the best k groups efficiently. The results discussed in this paper offer effective solutions to intelligent CRM for Enterprises. In our future work, we will research on other forms of limited resources problem as a result of postprocessing data mining models, and evaluate the effectiveness of our algorithms in the real-world deployment of the action-oriented data mining.

Acknowledgment Qiang Yang and Jie Yin are supported by a Hong Kong Government RGC grant. Charles X. Ling and Tielin Chen are supported by a Canadian NSERC grant.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases(VLDB'94)*, pages 487–499. Morgan Kaufmann, September 1994.

[2] D. Keim and H. Kriegel. Issues in visualizing large databases, 1995.

[3] D. A. Keim and H.-P. Kriegel. Visualization techniques for mining large databases: A comparison. *Transactions on Knowledge and Data Engineering, Special Issue on Data Mining*, 8(6):923–938, 1996.

[4] C. X. Ling and C. Li. Data mining for direct marketing – specific problems and solutions. In *Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 73 – 79. 1998.

[5] B. Liu, W. Hsu, L.-F. Mun, and H.-Y. Lee. Finding interesting patterns using user expectations. *Knowledge and Data Engineering*, 11(6):817–832, 1999.

[6] P. S. Usama Fayyad, Gregory Piatetsky-Shapiro. From data mining to knowledge discovery in databases. *AI Magazine*, 17(11), Fall 1996.