Extracting Actionable Knowledge from Decision Trees

Qiang Yang, *Senior Member*, *IEEE*, Jie Yin, Charles Ling, and Rong Pan

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 discovered knowledge 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 in the objective function such as profit. In this paper, we present novel algorithms that suggest actions to change customers from an undesired status (such as attritors) to a desired one (such as loyal) while maximizing an objective function: the expected net profit. These algorithms can discover cost-effective actions to transform customers from undesirable classes to desirable ones. The approach we take integrates data mining and decision making tightly by formulating the decision making problems directly on top of the data mining results in a postprocessing step. To improve the effectiveness of the approach, we also present an ensemble of decision trees which is shown to be more robust when the training data changes. Empirical tests are conducted on both a realistic insurance application domain and UCI benchmark data.

Index Terms—Phrases decision making, data mining, machine learning.

1 Introduction

Extensive research in data mining has been done on discovering distributional knowledge about the underlying data. Models such as Bayesian models, decision trees, support vector machines, and association rules have been applied to various industrial applications such as customer relationship management (CRM) [3], [30], [32], [10]. Despite such phenomenal success, most of these techniques stop short of the final objective of data mining, which is to maximize the profit while reducing the costs, relying on such postprocessing techniques as visualization and interestingness ranking [23], [29]. While these techniques are essential to move the data mining result to the eventual applications, they nevertheless require a great deal of human manual work by experts. Often, in industrial practice, one needs to walk an extra mile to automatically extract the real "nuggets" of knowledge, the actions, in order to maximize the final objective functions.

In this paper, we present a novel postprocessing technique to extract actionable knowledge from decision trees. To illustrate our motivation, we consider customer relationship management CRM [6], [16], [20], in particular, where we take the telecommunications industry as an example. This industry is experiencing more and more competitions in recent years. The battle is over their most valuable customers. With massive industry deregulation across the world, each customer is facing an ever-growing number of choices in telecommunications and financial services. As a result, an increasing number of customers are switching from one service provider to another. This phenomenon is called customer "churning" or "attrition," which is a major problem for these companies and makes it hard for them to stay profitable. The data sets are often cost sensitive and unbalanced. If we predict a valuable customer who will be an attritor as loyal, the cost is usually higher than the case when we classify a loyal customer as an attritor. Similarly, in direct marketing, it costs more to classify a willing customer as a reluctant one. Such information is usually given by a cost matrix, where the objective is to minimize the total cost. In addition, a CRM data set is often unbalanced; the most valuable customers who actually churn can be only a small fraction of the customers who stay.

In the past, many researchers have tackled the direct marketing problem as a classification problem [28], [27], [14], [43], where the cost-sensitive and unbalanced nature of the problem is taken into account. In management and marketing sciences, stochastic models are used to describe the response behavior of customers [9], [12], [25], [7]. In the data mining area, a main approach is to rank the customers by the estimated likelihood to respond to direct marketing actions and compare the rankings using a lift chart or the area under curve measure from the ROC curve [27], [31], [22]. Domingos [14] proposed the MetaCost framework for adapting accuracy-based classification to cost-sensitive learning by incorporating a cost matrix and conditional risk. Elkan and Zadrozny [17], [43] examined general cases where a classification decision depends on both the destination class and the customer in question. Various ensemble-based methods are examined under the cost-sensitive learning framework; for example, Fan et al. [21]

- *Q. Yang, J. Yin, and R. Pan are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: {qyang, yinjie, panrong}@cse.ust.hk.*
- *C. Ling is with the Department of Computer Science, The University of Western Ontario, London, Ontario N6A 5B7, Canada. Email: cling@csd.wuo.ca.*

Manuscript received 26 May 2005; revised 15 Mar. 2006; accepted 28 Aug. 2006; published online 20 Nov. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0212-0505.

integrated boosting algorithms with cost considerations. An association-rule-based approach is proposed in [40], where a focus is placed on data preprocessing in which the cost-sensitive and data-imbalance knowledge about the data is used in a *data sampling* phase. The resultant prediction model produces higher net profits when compared against the winners of the ACM KDDCUP 1998 competition [1]. Wang et al. [41], [39] took the actions and a data set as input and generated utility-enhancing patterns which are then pruned to produce the cost-effective action sets.

Recognizing the sequential nature of some CRM problems, Pednault et al. applied the framework of reinforcement learning to address the issue of sequential decision making when interactions can occur among decision outcomes [34]. Reinforcement learning refers to a class of problems and associated techniques in which the learner is to learn how to make sequential decisions based on delayed reinforcement so as to maximize cumulative rewards. In a Markov Decision Process (MDP), the environment is assumed to be in some state at any given point in time. In the case of targeted marketing, such states would be represented as feature vectors comprised of both the categorical and numerical data fields on which decision on what next actions to take is made.

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. More specifically, most data mining algorithms (predictive or supervised learning algorithms) only aim at constructing customer profiles, which predict the characteristics of customers of certain classes. Examples of these classes are: What kind of customers (described by their attributes such as age, income, etc.) are likely attritors (who will go to competitors), and what kind are loyal customers? 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). This can be done in the telecommunications industry, for example, by reducing the monthly rates or increasing the service level for a valuable customer.

Unlike distributional knowledge, to consider actionable knowledge one must take into account resource constraints. Actions, such as direct mailing and sales promotion, cost money to the enterprise [42]. At the same time, enterprises are increasingly constrained by cost cutting. There is thus a strong limitation on the number of customer segments that the company can take on, or in the number of actions the company can exploit. To make a decision, one must take into account the cost as well as the benefit of actions to the enterprise. However, for each customer, there may be a large number of possible actions or action sets that can be applied to the customer. Which of the actions to take depends not only on the particular customers' situation, but also on other customers who might benefit from the same action.

In the past, several approaches have been designed to extract knowledge from the CRM data. When the actions and state information are given in the data, sequential mining methods can be applied using MDP-like approaches [34]. When the actions are initially unknown, however, few approaches have been designed to invent new actions that can minimize the total cost and bring about profitable changes. In this paper, we present novel algorithms for postprocessing decision trees to obtain actions that are associated with attribute-value changes, in order to maximize the profit-based objective functions. This allows a large number of candidate actions to be considered, complicating the computation.

More specifically, in this paper, we consider two broad cases. One case corresponds to the unlimited resource situation, which is only an approximation to the real-world situations, although it allows a clear introduction to the problem. Another more realistic case is the limited-resource situation, where the actions must be restricted to be below a certain cost level. 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 and designing a greedy heuristic algorithm to solve it efficiently is computationally hard. 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 one. In order to improve the robustness of the mined actions, we also describe an ensemble-based decision tree algorithm [26], using a collection of decision trees rather than a single tree, to generate the actions. We show that the resultant action sets are indeed more robust with respect to training data changes.

An important contribution of the paper is that it integrates data mining and decision making together, such that the discovery of actions is guided by the result of data mining algorithms (decision trees in our case). While decision-making and optimization techniques are not new, their application to data mining has resulted in a new generation of learning algorithms, such as support vector machines [38], [11], manifold regularization for subspace learning [5]. The work of [34] that applies MDP to customer databases allows new knowledge—probabilistic plans—to be extracted from the data sets, allowing a new type of knowledge to be discovered from the data sets. However, they require the actions and state changes to be given as part of the input. Our approach can be considered as a new step in this direction, which is to discover action sets from the attribute value changes in a nonsequential data set through optimization.

The rest of the paper is organized as follows: We first present our base algorithm for finding unrestricted actions in Section 2. We then formulate two versions of action-set extraction problems, and show that finding the optimal solution for the problems is computationally difficult in the limited resources case (Section 3). We show that our greedy algorithms are efficient while achieving results very close to the optimal ones obtained by the exhaustive search (which is exponential in time complexity). We also present an ensemble tree-based technique for making the technique robust. Conclusions and future work are presented in Section 4.

2 Action Extraction in Decision Trees: Unlimited-Resource Case

2.1 A First Step in Postprocessing Decision Trees

Decision-tree learning algorithms, such as ID3 or C4.5 [36], are among the most powerful and popular predictive methods for classification. In CRM applications, a decision tree can be built from a set of examples (customers) described by a rich set of attributes including customer personal information (such as name, sex, and birthday, etc.), financial information (such as yearly income), family information (such as life style, number of children), and so on. Because decision trees can be converted to rules for explicit representation of the classification, one can easily obtain characteristics of customers belonging to a certain class (such as attritors). In this paper, we focus on the output of decision tree algorithms as the input to our postprocessing algorithms. Our algorithms rely on not only a prediction, but also a probability estimation of the classification, such as the probability of being loyal. Such information is available from decision trees.

Our first step is to consider how to extract actions when there is no restriction on the number of actions to produce. Our first industrial case study of an application corresponds to this case [10]. We call this the *unlimited-resource case*. Our data set consists of descriptive attributes and a class attribute. For simplicity, we consider a discrete-value problem, in which the class values are discrete values. Some of the values under the class attribute are more desirable than others. For example, in the banking application, the loyal status of a customer "stay" is more desirable than "not stay." The overall process of the algorithm can be briefly described in the following four steps:

1. Import customer data with data collection, data cleaning, data preprocessing, and so on.
2. Build customer profiles using an improved decision-tree learning algorithm [36] from the training data. In this case, a decision tree is built from the training data to predict if a customer is in the desired status or not. One improvement in the decision tree building is to use the area under the curve (AUC) of the ROC curve [22], [35] to evaluate probability estimation (instead of the accuracy). Another improvement is to use Laplace Correction to avoid extreme probability values.
3. Search for optimal actions for each customer (see Section 2.2 for details). This is a key component of the data mining system *Proactive Solution* [10].
4. Produce reports for domain experts to review the actions and selectively deploy the actions.

In the next section, we will discuss the `leaf-node search` algorithm used in Step 3 (search for optimal actions) in detail.

2.2 Leaf-Node Search in the Unlimited Resource Case

We first consider the simpler case of unlimited resources where the case serves to introduce our computational problem in an intuitive manner. The `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, the resulting decision tree can be used to classify, and more importantly, provide the probability of customers in the desired status such as being loyal or high-spending. When a customer, who can be either a 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.

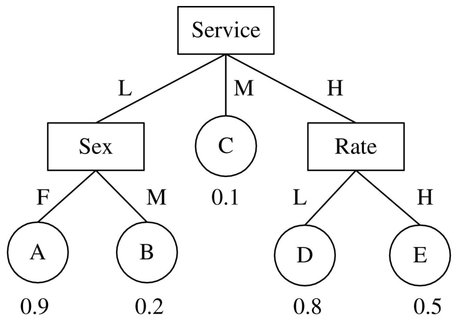


Fig. 1. An example of customer profile.

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 (see Section 2.3) 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 (Section 2.3) for actions, we define the net profit of an action to be as follows:

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

where P_{Net} 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. In Section 3.1, we extend this definition to a formal definition of the computational problem.

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

```
Algorithm leaf-node search
    1. For each customer $x$, do
    2. Let $S$ be the source leaf node in which $x$ falls into;
    3. Let $D$ be a destination leaf node for $x$ the maximum
       net profit $P_{Net}$;
    4. Output $(S, D, P_{Net})$;
```

To illustrate, consider an example shown in Fig. 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's record states that the Service = Low (service level is low), Sex = M (male), and Rate = L (mortgage rate is low). The customer is classified by the decision tree. It can be seen that Jack falls into the leaf node B, which predicts that Jack will have only a 20 percent chance of being loyal (or Jack will have an 80 percent chance to churn in the future). The algorithm will now search through all other leaves (A, C, D, and E) in the decision tree to see if Jack can be "replaced" into a best leaf with the highest net profit.

1. Consider leaf A. It does have a higher probability of being loyal (90 percent), 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 percent (80 percent - 20 percent) 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 \$1,000 from Jack when he is 100 percent loyal, then this probability gain (60 percent) is converted into \$600 ($1,000 \times 0.6$) of the expected gross profit. Therefore, the net profit would be \$400 ($600 - 200$).
4. Consider leaf E. The probability gain is 30 percent (50 percent - 20 percent), 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.

Notice that in the above example, the actions suggested for a customer-status change imply only correlations rather than causality between customer features and status. Similarly to other data mining systems, the actions should be reviewed by domain experts before deployment. This is the Step 4 of the *leaf-node search* algorithm given at the beginning of this section.

2.3 Cost Matrix

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, and 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.

For continuous attributes, such as interest rates that can be varied within a certain range, the numerical ranges can be discretized first using a number of techniques for feature transformation. For example, the entropy-based discretization method can be used when the class values are known [34]. Then, we can build a cost matrix for each attribute using the discretized ranges as the index values.

3 Postprocessing Decision Trees: The Limited Resource Case

3.1 A Formal Definition of BSP

Our previous case considered each leaf 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 *optimally* 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 several apply actions to increase the overall profit.

This limited-resource problem can be formulated as a precise computational problem. Consider a decision tree DT with a number of source leaf nodes that correspond to customer segments to be converted and a number of candidate destination leaf nodes, which correspond to the segments we wish customers to fall in. Formally, the bounded segmentation problem (BSP) is defined as follows: Given:

1. a decision tree DT built from the training examples, with a collection S of m source leaf nodes and a collection D of n destination leaf nodes (in Section 3.3, we extend from a single decision tree to multiple decision trees),
2. a prespecified constant k ($k \leq m$), where m is the total number of source leaf nodes,
3. a cost matrix $C(Attr_i, u, v), i = 1, 2, \ldots,$ which specifies the cost of converting an attribute $Attr_i$'s value from u to v, where u and v are indices for $Attr_i$'s values,
4. a unit benefit vector $B_C(L_i)$ denoting the benefit obtained from a single customer x when the x belongs to the positive class in a leaf node

$L_i, i = 1, 2, \ldots, N$, where N is the number of leaf nodes in the tree DT, and
5. a set C_{test} of test cases.

A solution is a set of k goals $\{G_i, i = 1, 2, \ldots, k\}$, where each goal consists of a set of source leaf nodes S_{ij} and one designation leaf node D_i, denoted as: $(\{S_{ij}, j = 1, 2, \ldots, |G_i|\} \rightarrow D_i)$, where S_{ij} and D_i are leaf nodes from the decision tree DT. The meaning of a goal is to transform customers that belong to the source nodes S to the destination node D via a number of attribute-value changing actions. Our aim is to find a solution with the maximal net profit (defined below).

Goals. Given a source leaf node S and a destination leaf node D, we denote the objective of converting a customer x from S to D as a goal, and denote it as $S \rightarrow D$. The concept of a goal can be extended to a set of source nodes: To transform a set of leaf nodes S_i to a designation leaf node D, the goal is $\{S_i, i = 1, 2, \ldots\} \rightarrow D$.

Actions. In order to change the classification of a customer x from S to D, one may need to apply more than one attribute-value changing action. An action A is defined as a change to an attribute value for an attribute $Attr$. Suppose that for a customer x, the attribute $Attr$ has an original value u. To change its value to v, an action is needed. This action A is denoted as $A = \{Attr, u \rightarrow v\}$.

Action Sets. A goal is achieved by a set of actions. To achieve a goal of changing a customer x from a leaf node S to a destination node D, a set of actions that contains more than one action may be needed. Specifically, consider the path between the root node and D in the tree DT. Let $\{(Attr_i = v_i), i = 1, 2, \ldots, N_D\}$ be set of attribute-values along this path. For x, let the corresponding attribute-values be $\{(Attr_i = u_i), i = 1, 2, \ldots, N_D\}$. Then, the actions of the form can be generated: $ASet = \{(Attr_i, u_i \rightarrow v_i), i = 1, 2, \ldots, N_D\}$, where we remove all null actions where u_i is identical to v_i (thus, no change in value is needed for an $Attr_i$). This action set $ASet$ can be used for achieving the goal $S \rightarrow D$.

Net Profits. The net profit of converting one customer x from a leaf node S to a destination node D is defined as follows: Consider a set of actions $ASet$ for achieving the goal $S \rightarrow D$. For each action $Attr_i, u \rightarrow v$ in $ASet$, there is a cost as defined in the cost matrix: $C(Attr_i, u, v)$. Let the sum of the cost for all of $ASet$ be $C_{\text{total}}, S \rightarrow D(x)$.

Let the probability of x to belong to the positive class in S be $p(+|x, S)$. Likewise, let the probability of a customer in D be in the positive class be $p(+|x, D)$. Recall that from the input, we have a benefit vector $B_C(L)$ for the leaf nodes L. Thus, we have $B_C(S)$ as the benefit of belonging to node S and $B_C(D)$ as the benefit of belonging to node D. Then, the *unit net profit* of converting one customer x from S to D is:

$$P^{unit}(x, S \rightarrow D) = (B_C(D) * p(+|x, D) - B_C(S) * p(+|x, S)) - C_{\text{total}}, S \rightarrow D(x). \tag{2}$$

Then, for a collection C_{test} of all test cases that fall in node S, the total net profit of applying an action set for achieving the goal $S \rightarrow D$ is:

$$P(C_{test}, S \rightarrow D) = \Sigma_{x \in C_{test}} P^{unit}(x, S \rightarrow D). \tag{3}$$

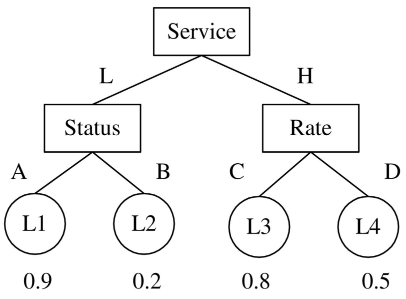


Fig. 2. An example decision tree.

When the index of S is i, and the index of D is j, we denote $P(C_{test}, S \rightarrow D)$ as P_{ij} for simplicity.

Thus, the BSP problem is to find the best k groups of source leaf nodes $\{Group_i, i = 1, 2, \ldots, k\}$ and their corresponding goals and associated action sets to maximize the total net profit for a given test data set C_{test}.

An Example. To illustrate, consider an example in Fig. 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, and 0.5, respectively. Now consider the task of transforming L_2 and L_4 to a higher probability node, such that the net profit of making all transformations is maximized. To illustrate the process, consider a test data set such that there is exactly one member that falls in each leaf node of this decision tree.

In order to calculate the net profit, we assume all leaf nodes to have an initial benefit of one unit. For simplicity, 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 $(0.9 \times 1 - 0.2 \times 1) - 0.1 = 0.6$.

To illustrate the limited resources problem, consider again our decision tree in Fig. 2. Suppose that we wish to find a single customer segment ($k = 1$). A candidate group is $\{L_2, L_4\}$, with a selected action set $\{Service \leftarrow H, Rate \leftarrow C\}$ which can transform the group to node L_3. Assume that L_2 and L_4 only contain one example each. Transferring this group to leaf node L_3, L_2 changes the service level only and, thus, has a profit gain of $(0.8 - 0.2) \times 1 - 0.1 = 0.5$ 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 $0.2 + 0.5 = 0.7$.

The BSP problem has an equivalent matrix representation. From (2) and (3), we obtain a profit matrix $M = (P_{ij}), i = 1, \ldots, m; j = 1, \ldots, n$ formed by listing all source leaf nodes S_i as the row index and all the action sets $ASet_j$, for achieving the goal $(S_i \rightarrow D_j)$, as the column index (here, we omit S_i in the column headings). In this matrix M, $(P_{ij} \geq 0)$, $1 \leq i \leq m$ (where m is the number of source leaf nodes), and $1 \leq j \leq n$ (n is the number of destination leaf nodes). P_{ij} denotes the profit gain computed by applying $ASet_j$ to S_i for all test-case customers that falls in S_i. If $P_{ij} > 0$, that is, applying $ASet_j$ to transfer S_i to the corresponding destination leaf node can bring about a net profit, we say that the source leaf node S_i can be *covered* by the action set $ASet_j$. From (2) and (3), the computation of the profit matrix $M(.,.)$ can be done in $O(m * n)$.

As an example of the profit matrix computation, a part of the profit matrix corresponding to the source leaf node

TABLE 1
An Example of the Profit Matrix Computation

	$ASet_1(L_2)$(Goal=$\rightarrow L_1$)	$ASet_2(L_2)$ (Goal=$\rightarrow L_3$)	$ASet_3(L_2)$ (Goal=$\rightarrow L_4$)
L_2	0.6	0.4	0.1
L_4	...	...	...

TABLE 2
Illustrating the Greedy-BSP Algorithm

Source Nodes	$ASet_1$ (Goal= $\rightarrow D_1$)	$ASet_2$ (Goal= $\rightarrow D_2$)	$ASet_3$ (Goal= $\rightarrow D_3$)	$ASet_4$ (Goal= $\rightarrow D_4$)
S_1	2	0	1	1
S_2	0	1	0	0
S_3	0	1	0	0
S_4	0	1	0	0
Column Sum	2	3	1	1
Selected Actions		x		

L_2 is as shown in Table 1 where $ASet_1 = \{Status = A\}$, $ASet_2 = \{Service = H, Rate = C\}$, and $ASet_3 = \{Service = H, Rate = D\}$ (here, for convenience, we ignore the source value of the attributes, which is dependent on the actual test cases).

Then, the BSP problem becomes one of picking the best k columns of matrix M such that the sum of the maximum net profit value for each source leaf node *among the k columns* is maximized. When all P_{ij} elements are of unit cost, this is essentially a maximum coverage problem [19], which aims at finding k sets such that the total weight of elements covered is maximized, where the weight of each element is the same for all the sets. A special case of the BSP problem is equivalent to the maximum coverage problem with unit costs. Thus, we know that the BSP problem is NP-Complete. Our aim will then be to find approximation solutions to the BSP problem.

3.2 Algorithms for BSP

Our first solution is an exhaustive search algorithm for finding the k optimal action sets with maximal net profit.

Algorithm Optimal-BSP

```
1. for each $ASet_l \in A$, $1 \leq l \leq n$, choose any combination of
   $k$ action sets, do
       1.1. Group the leaf nodes into $k$ groups
       2.1. Evaluate the net benefit of the action sets on the
            groups
   end for
2. return the $k$ action sets with associated leaf node groups
   that have the maximal net benefit using (2) and (3).
```

Since the optimal-BSP needs to examine every combination of k action sets, the computational complexity is $O(n^k)$, which is exponential in the value of k.

To avoid the exponential worst-case complexity, we have also developed a greedy algorithm which can reduce the computational cost and 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 $ASet_j$, $1 \leq j \leq n$. Each $ASet_j$ can be denoted as a subset of L_s which only contains the covered leaf nodes L_i for $P_{ij} \geq 0$, $1 \leq i \leq m$. 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 below, where C is the resulting k action sets.

We consider the intuition of the Greedy-BSP algorithm using an example profit matrix M as shown in Table 2, where we assume a $k = 2$ limit. In this table, each number is a profit P_{ij} value computed from the input parameters. The greedy algorithm processes this matrix in a sequential manner for k iterations. In each iteration, it considers adding one additional column of the M matrix, until it has considered all k columns. Initially, Greedy-BSP starts with an empty result set $C = \emptyset$. The algorithm then compares all the column sums that corresponds to converting all leaf nodes S_1 to S_4 to each destination leaf node D_i in turn. It found that $ASet_2 = (\rightarrow D_2)$ has the current maximum profit of three units. Thus, the resultant action set C is assigned to $\{ASet_2\}$.

Next, Greedy-BSP considers how to expand the customer groups by one. To do this, it considers which additional column will increase the total net profit to a highest value, if we can include one more column. As can be seen from Table 3, if we in addition consider the first column, then the node S_1 can choose to be converted to D_1, instead of D_2 as in Table 2. In that case, the profit of $C = \{ASet_1, ASet_2\}$ can be increased by two units, which is the maximum increase among all other columns. Thus, we choose this subset to be the current action set C.

Because we have now reached the resource limit $k = 2$, we will terminate with the action set C and two groups of leaf nodes: $G_1 = \{S_1\}$, and $G_2 = \{S_2, S_3, S_4\}$. This example

TABLE 3
Illustrating the Greedy-BSP Algorithm (Continued)

Source Nodes	$ASet_1$ (Goal= $\rightarrow D_1$)	$ASet_2$ (Goal= $\rightarrow D_2$)	$ASet_3$ (Goal= $\rightarrow D_3$)	$ASet_4$ (Goal= $\rightarrow D_4$)
S_1	2	0	1	1
S_2	0	1	0	0
S_3	0	1	0	0
S_4	0	1	0	0
Column Sum	**2**	**3**	1	1
Selected Actions	x	x		

returns a total net profit of 2 + 3 = 5 units, which is also the optimal solution.

The algorithm Greedy-BSP is now described as follows:

```
Algorithm Greedy-BSP
1. $C \leftarrow \emptyset$; Compute the matrix $M = (P_{ij})$ using (2) and (3);
2. for $l$ = 1 to $k$
        2.1. select $ASet_l \in A$ that maximizes
             $$\sum_{S_i \in \text{cover}(C \cup ASet_l)} max\{P_{ij}\}, j = 1, 2, \ldots, l \quad (4)$$
        2.2. $C \leftarrow C \cup ASet_l$
   end for
3. return $C$
```

This algorithm can be shown to perform close to the optimal result in our subsequent analysis. In particular, we can exploit the complexity analysis of the approximate maximum coverage algorithm given in [21] to reach this result. In order to prove the approximation ratio of the solution returned by Greedy-BSP to one by Optimal-BSP, we first need to establish the following two lemmas.

If we let $Profit(Greedy)$ and $Profit(OPT)$ be the net profit returned by the Greedy-BSP algorithm and the Optimal-BSP algorithm, respectively, we have the following property.

Lemma 1. *For $l = 1, 2, \ldots, k$, let $ASet$ be an action set. We have*

$$\begin{aligned} &Profit(\cup_{i=1}^{l} ASet_i) - Profit(\cup_{i=1}^{l-1} ASet_i) \\ &\geq \frac{(Profit(OPT) - Profit(\cup_{i=1}^{l-1} ASet_i)}{k}. \end{aligned} \quad (5)$$

Proof. Let the optimal solution returned by Optimal-BSP consists of k optimal action sets. Suppose Greedy-BSP has already selected $(l-1)$ action sets so far, m of which are contained in the optimal solution. Now, we consider the situation where Greedy-BSP selects the next $ASet_l$ action set. Because of the heuristic strategy used in the Step 2.1 of the Greedy-BSP algorithm, $Profit(\cup_{i=1}^{l} ASet_i) - Profit(\cup_{i=1}^{l-1} ASet_i)$ represents the additional profit gain achieved by $ASet_l$. In addition, $ASet_l$ should be a set that can achieve maximal additional profit gain. On the other hand, assume Greedy-BSP selects those $(k-m)$ optimal action sets in the optimal solution and yet have not chosen by itself, the profit gain of this batch procedure is at least $Profit(OPT) - Profit(\cup_{i=1}^{l-1} ASet_i)$. According to the *pigeonhole principle*, there must exist one single action set in the remaining $(k-m)$ optimal action sets, whose profit is at least $\frac{Profit(OPT)-Profit(\cup_{i=1}^{l-1} ASet_i)}{k-m}$. Since this action set is also a candidate for selecting the next $ASet_l$, we have

$$\begin{aligned} &Profit(\cup_{i=1}^{l} ASet_i) - Profit(\cup_{i=1}^{l-1} ASet_i) \\ &\geq \frac{Profit(OPT) - Profit(\cup_{i=1}^{l-1} ASet_i)}{k-m} \\ &\geq \frac{Profit(OPT) - Profit(\cup_{i=1}^{l-1} ASet_i)}{k}. \end{aligned}$$

□

Lemma 2. *For $l = 1, 2, \ldots, k$, we have*

$$Profit(\cup_{i=1}^{l} ASet_i) \geq \left[1 - \left(1 - \frac{1}{k}\right)^l\right] Profit(OPT). \quad (6)$$

Proof Sketch. The proof can be done by induction using Lemma 1, similar to the proof of Lemma 3.13 in [21]. □

Based on the above two established lemmas, we have the following theorem:

Theorem 1. *The Greedy-BSP is a $(1 - \frac{1}{e})$-approximation algorithm.*

$$\begin{aligned} Profit(Greedy) &\geq \left[1 - \left(1 - \frac{1}{k}\right)^k\right] Profit(OPT) \\ &> \left(1 - \frac{1}{e}\right) Profit(OPT). \end{aligned} \quad (7)$$

Proof. Theorem 1 follows directly from Lemma 2 by letting $l = k$. In addition, because $\lim_{k\to\infty} 1 - (1 - \frac{1}{k})^k = 1 - \frac{1}{e}$ and $1 - (1 - \frac{1}{k})^k$ is decreasing, it follows that $1 - (1 - \frac{1}{k})^k > 1 - \frac{1}{e}$. □

3.3 Improving the Robustness Using Multiple Trees

The major advantage of the Greedy-BSP algorithm is that it can significantly reduce the computational cost while also guaranteeing the high quality of the solution at the same time. In Greedy-BSP, the built decision tree always choose the most informative attribute as the root node. However, as pointed out in [26], many top-ranked attributes may exhibit similar discriminating merits with little difference. It is worthwhile to employ different top-ranked attributes as the root nodes for building multiple decision trees. Therefore, we have also proposed an algorithm referred to as Greedy-BSP-Multiple which is based on integrating an ensemble of decision trees in this paper. Ensemble-based methods have been shown to improve the robustness of machine learning systems greatly [45], [26], [44]. The basic idea is to construct multiple decision trees using different top-ranked attributes as their root nodes. For each set of test cases, the ensemble decision trees return the median net profit and the corresponding leaf nodes and action sets as the final solution. We believe that this method will be more robust when the training data set changes, because each change can only alter a small number of trees. Thus, we expect that when the training data are unstable, the ensemble-based decision tree methods can perform much more stable as compared to results from the single decision trees.

```
Algorithm Greedy-BSP-Multiple
1. Given a training data set described by $p$ attributes
        1.1. calculate gain ratios to rank all the attributes in
             an descending order
        1.2. for $i = 1$ to $p$
                use the $i$th attribute as root node to
                construct the $i$th decision tree
             end for
2. Take a set of testing examples as input
        2.1. for $i = 1$ to $p$
                use the $i$th decision tree to calculate the net
                profit by calling algorithm Greedy-BSP
             end for
        2.2. return $k$ action sets corresponding to the median
             net profit
```

The added advantage of Greedy-BSP-Multiple is that it is more robust with respect to different sampled training data. Since Greedy-BSP-Multiple relies on building multiple decision trees to calculate the median net profit, different sampling can only affect the construction of a small portion of decision trees. Therefore, Greedy-BSP-Multiple can produce net profit with less variance.

3.3.1 Difference from Previous Works

In business marketing, most of the marketing planning activities have been done in a human-heavy process, which is carried out by hand [4], [13]. Collecting customer data and using the data for direct marketing operations has increasingly become possible, thanks to the popularity of data mining technology. One approach is known as "database marketing," which is defined as creating a bank of information about individual customers from their orders, enquiries, and other activity, using it to analyze the customer behavior and developing intelligent strategies [37], [33], [15]. An important computational aspect is to segment a customer group into subgroups, often in terms a binary decision variable such as customer's willingness to buy a product or not to buy. This segmentation corresponds to traditional classification learning in machine learning, such as decision trees [36], where the aim is to generate a ranking function for the customers sorted on their likelihood to buy a product (or stay loyal to the company), so that a "gain chart" or "lift chart" can be created for human analysis. Analytical techniques such as linear and logistic regression can be used to implement the ranking functions [15]. The decision then is on which subset of the customers to market a certain product to, in order to maximize the total *net* profit [27]. However, even though the segmentations can be discovered using an machine learning algorithm, the actions are still to be discovered by human experts.

Machine learning and data mining research has contributed to the business practice by addressing some new issues in marketing. One issue is that the typical marketing data are *cost sensitive*, in that the false positive and false negative costs are different. To solve this problem, [43], [14] proposed the framework of cost-sensitive learning by incorporating a cost matrix for balancing the total expected misclassification costs. In addition to the cost issues, the number of items belonging to one class may greatly outnumber those in another class. As mentioned above, machine learning methods have traditionally been applied to produce a ranking of customers by the estimated probability to respond to a marketing action, and selecting some top portion of the ranked list [27], [31]. When the data are biased, traditional accuracy-based measurements are no longer adequate. In order to solve this problem, Wang et al. [22] presented an association rule-based approach that differentiates between the positive class and the negative class members *the most*, and use these rules to for segmentation.

Another direction in machine learning is to apply sequential learning techniques using reinforcement learning and Markov decision processes. For example, [43], [34], [2] aimed to find a *policy* with which to direct an optimal action based on a customer's current status. Here, the objective is to maximize the total benefits accrued over a period of time, after a sequence of actions is taken, when deciding whether to take an action or not.

However, all of the above research works are aimed at either finding a segmentation of the customer database, or deciding to take a predefined action for every customer based on that customer's current status. None of them addressed the issue of *discovering* actions that might be taken from a customer database. To the best of our knowledge, ours is the first such work in machine learning and business marketing that addressed this action-discovery issue.

3.4 Experimental Evaluation

In order to evaluate the performance of our proposed algorithm, experiments were carried out on a real data set from an insurance company and four data sets from the UCI ML repository [8].

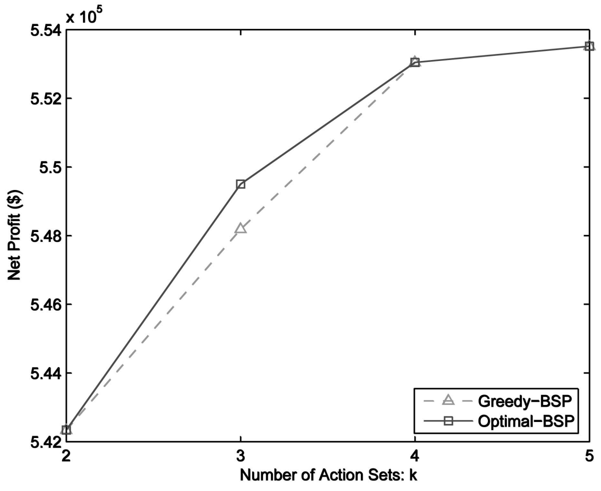


Fig. 3. Net profit versus number of action sets.

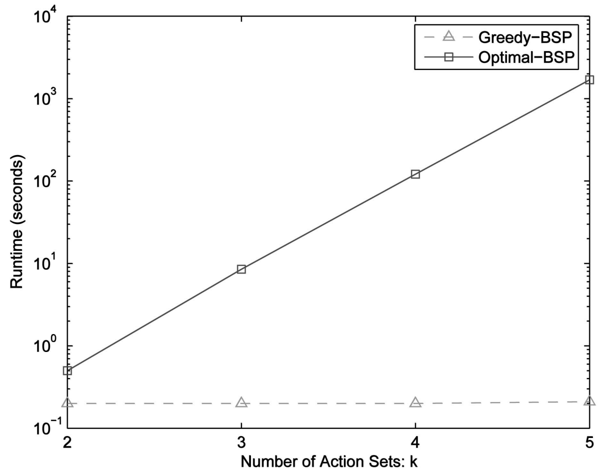


Fig. 4. Runtime versus number of action sets.

3.4.1 Experiments on Real Data

Experiments were first performed on a real data set obtained from an insurance company in Canada. This data set consists of more than 25,000 records for customers who have the status of "stay" or "leave" the insurance company, which are referred to as positive and negative examples, respectively. Each example is described by more than 60 attributes, many of which are not hard attributes. About 20 attributes are soft attributes with reasonable costs for value changes. We constructed a cost matrix for each attribute contained in the data set according to their semantics in the real domain.

In our experiment, we first selected 10 attributes based on the *Gain Ratio* criterion. Since this data set has a highly unbalanced data distribution, we randomly sampled 6,000 examples as the training data, with the ratio of positive and negative as one to one, in order to prevent a decision tree from predicting all the customers to be negative. We also randomly sampled 300 examples from the rest of the data to be used as the testing data. In this setting, we built a decision tree with 153 leaf nodes. Eighty-seven of them are considered negative leaf nodes because their probability of being positive is less than 50 percent, while the other 66 positive leaf nodes.

We applied Greedy-BSP and Optimal-BSP to calculate the prespecified k action sets with maximal net profit. Fig. 3 shows the net profit obtained by the two algorithms with respect to different numbers of action sets k. As shown in the figure, the net profit increases for both Greedy-BSP and Optimal-BSP with an 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.

Table 4 compares the k action sets selected by both Greedy-BSP and Optimal-BSP with respect to different numbers of action sets k. There are a total of 66 action sets provided by the decision tree built in our experiments. As shown in the table, the action sets selected by Greedy-BSP are very close to that by Optimal-BSP for the same number of action sets k.

In our example, each action set contains a number of actions; for example, the action set A_3 consists of four different actions (attribute changes), that is,

TABLE 4
Selected Action Sets versus Number of Action Sets

#Action	Selected Action Sets	
Sets	Greedy-BSP	Optimal-BSP
$k=2$	$\{\mathbf{A_3}, A_{29}\}$	$\{\mathbf{A_1}, A_{29}\}$
$k=3$	$\{\mathbf{A_3}, A_{20}, A_{29}\}$	$\{\mathbf{A_1}, A_{20}, A_{29}\}$
$k=4$	$\{A_1, A_3, A_{20}, A_{29}\}$	$\{A_1, A_3, A_{20}, A_{29}\}$
$k=5$	$\{A_1, A_3, A_{20}, A_{29}, A_{42}\}$	$\{A_1, A_3, A_{20}, A_{29}, A_{42}\}$

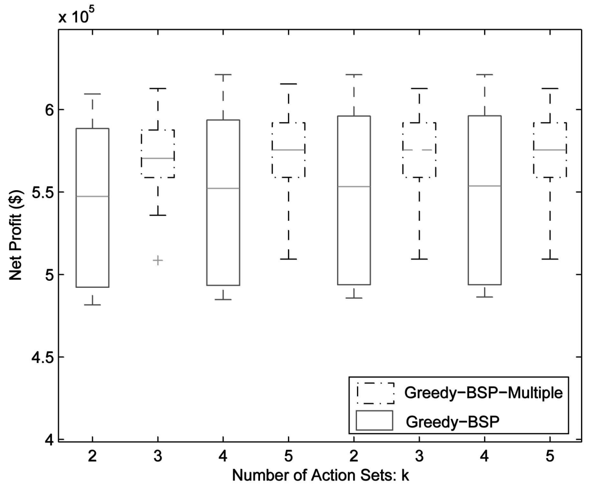


Fig. 5. Net profit versus number of action sets.

TABLE 5
UCI Data Sets Used in the Experiments

Name of data sets	No. of attributes	No. of training data	No. of testing data
German	20	300	100
Australian	14	500	100
Adult	14	10000	10000
Endgame	9	400	100

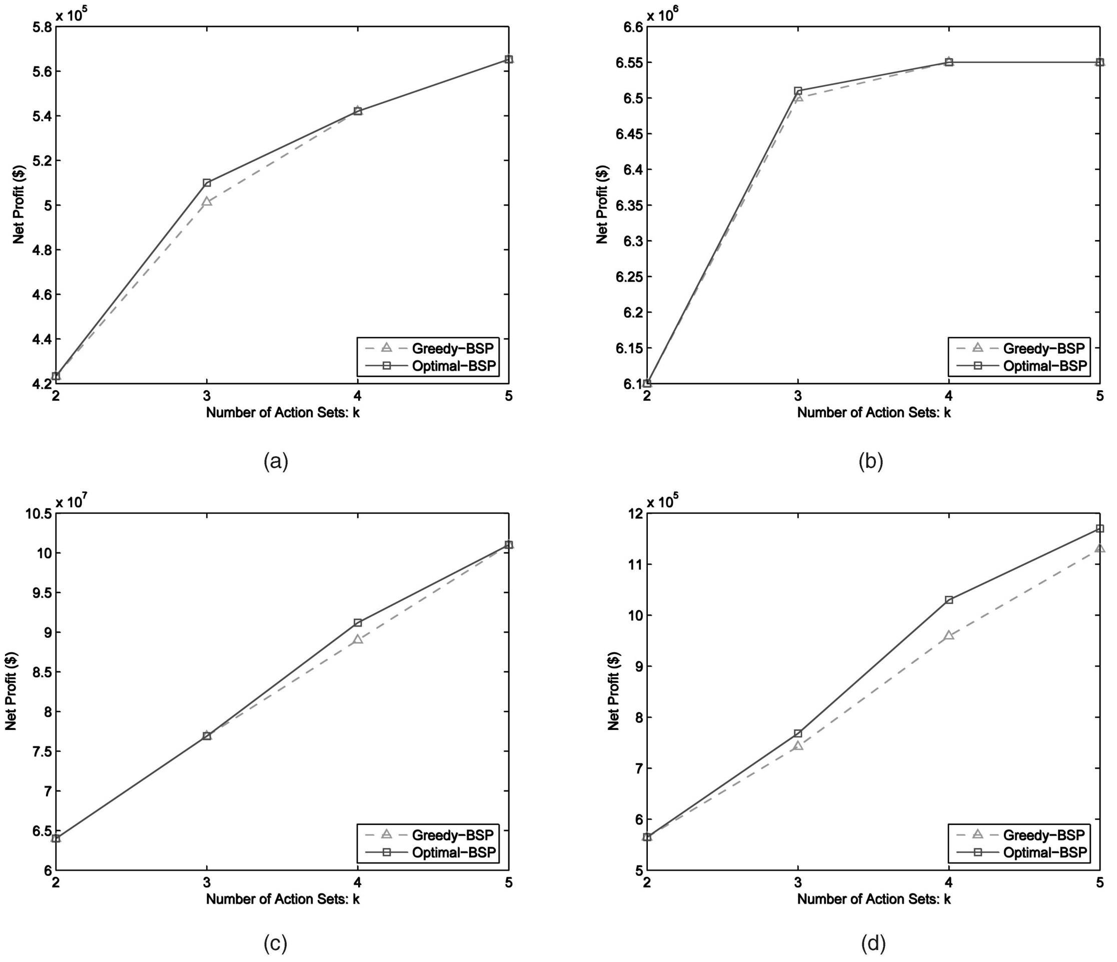


Fig. 6. (a) German. (b) Australian. (c) Adult. (d) Endgame. Net profit comparisons of Optimal-BSP and Greedy-BSP on four UCI data sets.

$$\{a_4 : \$0 - \$249 \rightarrow \$500 - \$999, a_7 : F \rightarrow E,$$
$$a_8 : DIV \rightarrow WID, a_9 : 1980 - 1989 \rightarrow 1990 - 1994\}.$$

As described above, there are a total of 87 negative leaf nodes in our decision tree. By applying the action set A_3, a group of customers falling into five negative leaf nodes $\{L_2, L_3, L_{13}, L_{56}, L_{75}\}$ can be transferred to third positive leaf node, which corresponds to a customer segmentation.

Fig. 4 compares the runtime of the two algorithms with respect to different numbers of action sets k. Note that the runtime of y-axis uses a logscale. As expected, Greedy-BSP is much more efficient and scalable than Optimal-BSP. For Greedy-BSP, the runtime is 0.20 seconds irrespective of the number of action sets k. In contrast, the runtime for Optimal-BSP increases exponentially with the increasing number of action sets k. This is because the optimal algorithm needs to compare many more combinations for larger values of k in order to obtain maximal net profits.

We performed another set of experiments to evaluate *the robustness* of Greedy-BSP and Greedy-BSP-Multiple (where 10 decision trees are used in each ensemble). In this experiment, we randomly selected 300 examples from the whole data set as the testing data. From the rest of the data, we randomly generated 10 training data sets of 6,000 examples, where the ratio of positive and negative equals 1:1. We applied the two algorithms on these 10 groups of data. Fig. 5 compares the performance of the two algorithms with respect to different numbers of actions sets k in a box plot (the figure shows a box and whisker plot for each set of

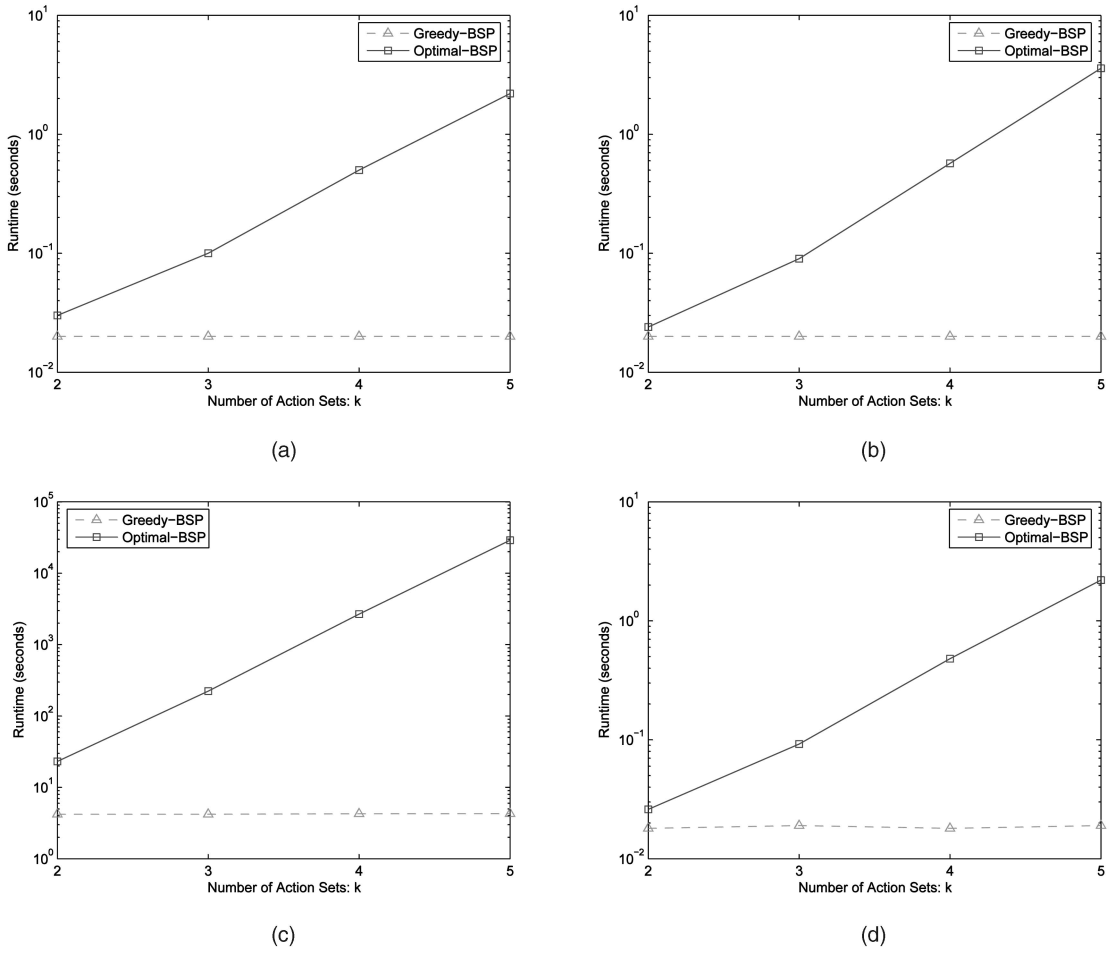


Fig. 7. Runtime comparisons of Optimal-BSP and Greedy-BSP on four UCI data sets. (a) German. (b) Australian. (c) Adult. (d) Endgame.

experiments, where a box has lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the rest of the data. Outliers are data with values beyond the ends of the whiskers.). We can see from the figure that Greedy-BSP-Multiple can produce the net profit values with much less variance than Greedy-BSP when 10 sets of randomly sampled examples are used for training. Since Greedy-BSP-Multiple relies on building multiple decision trees to calculate the median net profit, different sampling can only affect the construction of a small portion of decision trees, while the rest are left unchanged. Therefore, we can conclude that Greedy-BSP-Multiple is more robust than Greedy-BSP.

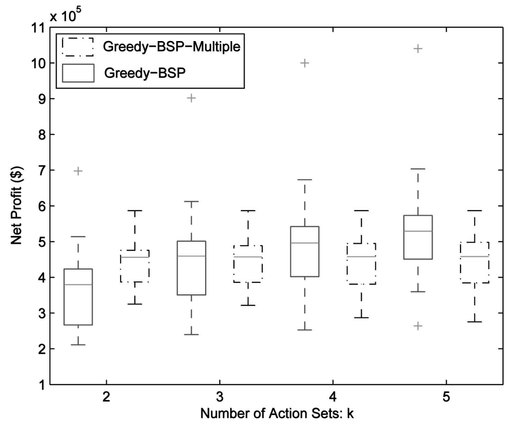
3.4.2 Experiments on UCI Data

We also performed experiments on four UCI data sets to evaluate the performance of the algorithms. The four data sets used in our experiments are listed in Table 5. These data sets were chosen because they have binary classes and a sufficient number of examples. In our experiments, we also used *Gain Ratio* [32] criterion to select eight significant attributes for each data set. For Greedy-BSP and Optimal-BSP, we calculated the net profit based on a decision tree whose root node is the attribute with the maximal gain ratio value. For Greedy-BSP-Multiple, the net profit is computed using *eight* decision trees with different attributes as the root node. For each data set, we randomly sampled the training data set with the ratio of positive and negative as one to one. Another independent data set generated randomly is used for testing.

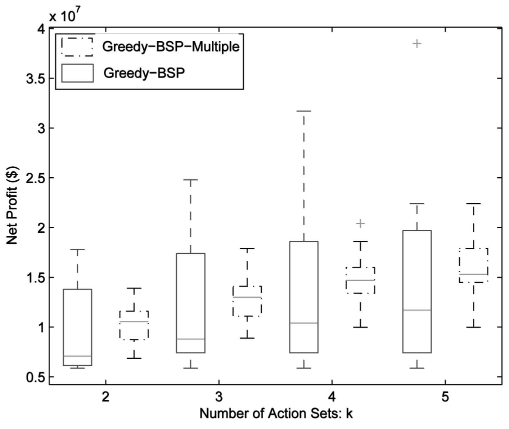
Fig. 6 shows the net profit obtained by Optimal-BSP and Greedy-BSP on the four data sets. We can also observe that, for each data set, the values of net profit computed by Greedy-BSP are very close to or the same as those found by Optimal-BSP with respect to the same number of action sets k.

Fig. 7 compares the efficiency of Optimal-BSP and Greedy-BSP on the four data sets. Note that the values of y-axis use a logscale in the figure. We can observe that the runtime of Optimal-BSP increases exponentially as the number of action sets k increases. This makes Optimal-BSP intractable especially when the decision tree has a large number of positive and negative leaf nodes. In contrast, the runtime of Greedy-BSP remains approximately the same regardless of different numbers of action sets k. Therefore, Greedy-BSP is much more efficient and scalable than Optimal-BSP.

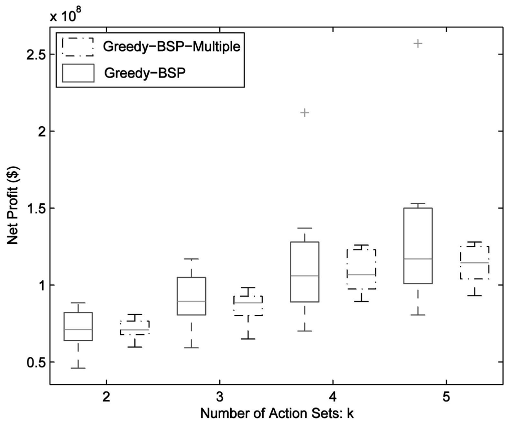
We also performed experiments to evaluate the robustness of Greedy-BSP and Greedy-BSP-Multiple on the four data sets. In this experiment, for each data set, we randomly generated 10 training data sets, each of which has the same positive and negative examples. The same testing data set



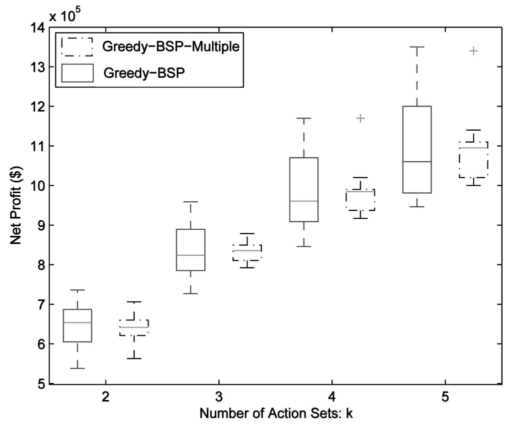
(a)



(b)



(c)



(d)

Fig. 8. Robustness comparisons of Greedy-BSP and Greedy-BSP-Multiple on four UCI data sets. (a) German. (b) Australian. (c) Adult. (d) Endgame.

was used for evaluation. Fig. 8 shows the net profit obtained by the two algorithms with respect to different sampled training data. We can also observe that, when different sampled examples are used for training, Greedy-BSP-Multiple can produce the net profit with much less variance than Greedy-BSP. Therefore, Greedy-BSP-Multiple is more robust than Greedy-BSP.

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 an increasing number of action sets k, which is more efficient than Optimal-BSP. In addition, by building multiple decision trees, Greedy-BSP-Multiple is more robust than Greedy-BSP when different sampled examples are used for training.

4 Conclusions and Future Work

Most data mining algorithms and tools produce only the segments and ranked lists of customers or products in their outputs. In this paper, we present a novel technique to take these results as input and produce a set of actions that can be applied to transform customers from undesirable classes to desirable ones. For decision trees, we have considered two broad cases. The first case corresponds to unlimited resources, and the second case corresponds to the limited resource-constraint situations. In both cases, our aim is to maximize the expected net profit of all the customers. We have found a greedy heuristic algorithm to solve both problems efficiently and presented an ensemble-based decision-tree algorithm that use a collection of decision trees, rather than a single tree, to generate the actions. We show that the resultant action set is indeed more robust with respect to training data changes.

The results discussed in this paper offer effective solutions to intelligent CRM for enterprises. Our initial applications case study (in unlimited resources case, reported in [28], [42]) has shown strong promise in applying this class of postprocessing techniques in practice.

In our future work, we will research 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.

Acknowledgments

The authors would like to thank Hong Kong RGC for supporting this work under grant HKUST6187/04E. They also thank the anonymous referees for their comments.

REFERENCES

[1] The kdd-cup-98 result: http://www.kdnuggets.com/meetings/kdd98/kdd-cup-98-results.html, 2005.

[2] N. Abe, E. Pednault, H. Wang, B. Zadrozny, W. Fan, and C. Apte, "Empirical Comparison of Various Reinforcement Learning Strategies for Sequential Targeted Marketing," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM '02),* 2002.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94),* pp. 487-499, Sept. 1994.

[4] Bank Marketing Association, *Building a Financial Services Plan: Working Plans for Product and Segment Marketing,* Financial Sourcebooks, 1989.

[5] M. Belkin, P. Niyogi, and V. Sindhwani, "On Manifold Regularization," *Proc. 10th Int'l Workshop Artificial Intelligence and Statistics,* pp. 17-24, Jan. 2005.

[6] A. Berson, K. Thearling, and S.J. Smith, *Building Data Mining Applications for CRM.* McGraw-Hill, 1999.

[7] G. Bitran and S. Mondschein, "Mailing Decisions in the Catalog Sales Industry," *Management Science,* vol. 42, pp. 1364-1381, 1996.

[8] C.L. Blake and C.J. Merz, "UCI Repository of Machine Learning," www.ics.uci.edu/~mlearn/mlrepository.html, 1998.

[9] J.R. Bult and T. Wansbeek, "Optimal Selection for Direct Mail," *Marketing Science,* vol. 14, pp. 378-394, 1995.

[10] M.-S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. Knowledge And Data Eng.,* vol. 8, pp. 866-883, 1996.

[11] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines.* Cambridge Univ. Press, 2000.

[12] W. Desarbo and V. Ramaswamy, "Crisp: Customer Response Based Iterative Segmentation Procedures for Response Modeling in Direct Marketing," *J. Direct Marketing,* vol. 8, pp. 7-20, 1994.

[13] S. Dibb, L. Simkin, and J. Bradley, *The Marketing Planning Workbook.* Routledge, 1996.

[14] P. Domingos, "Metacost: A General Method for Making Classifiers Cost Sensitive," *Proc. ACM Conf. Knowledge Discovery and Data Mining,* pp. 155-164, 1999.

[15] R.G. Drozdenko and P.D. Drake, *Optimal Database Marketing.* Sage Publications, 2002.

[16] J. Dyche, *The CRM Handbook: A Business Guide to Customer Relationship Management.* Addison-Wesley, 2001.

[17] C. Elkan, "The Foundations of Cost-Sensitive Learning," *Proc. 17th Int'l Joint Conf. Artificial Intelligence (IJCAI '01),* 2001.

[18] W. Fan, S.J. Stolfo, J. Zhang, and P.K. Chan, "Adacost: Misclassification Cost-Sensitive Boosting," *Proc. 16th Int'l Conf. Machine Learning,* pp. 97-105, 1999.

[19] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NPCompleteness.* WH Freeman, 1979.

[20] B.J. Goldenberg, *CRM Automation.* Prentice Hall, 2002.

[21] D.S. Hochbaum, "Approximation Algorithms for Np-Hard Problems," chapter 3, p. 136. PWS Publishing Company, 1995.

[22] J. Huang and C.X. Ling, "Using Auc and Accuracy in Evaluating Learning Algorithms," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 3, pp. 299-310, 2005.

[23] D.A. Keim and H.-P. Kriegel, "Visualization Techniques for Mining Large Databases: A Comparison," *IEEE Trans. Knowledge and Data Eng.,* special issue on data mining, vol. 8, no. 6, pp. 923-938, Dec. 1996.

[24] R. Kohavi and M. Sahami, "Error-Based and Entropy-Based Discretization of Continuous Features," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining,* pp. 114-119, 1996.

[25] N. Levin and J. Zahavi, "Segmentation Analysis with Managerial Judgment," *J. Direct Marketing,* vol. 10, pp. 28-37, 1996.

[26] J. Li and H. Liu, "Ensembles of Cascading Trees," *Proc. IEEE Int'l Conf. Data Mining (ICDM '03),* pp. 585-588, 2003.

[27] C.X. Ling and C. Li, "Data Mining for Direct Marketing—Specific Problems and Solutions," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining (KDD '98),* pp. 73-79, 1998.

[28] C.X. Ling, T. Chen, Q. Yang, and J. Cheng, "Mining Optimal Actions for Intelligent CRM," *Proc. IEEE Int'l Conf. Data Mining (ICDM),* 2002.

[29] B. Liu, W. Hsu, L.-F. Mun, and H.-Y. Lee, "Finding Interesting Patterns Using User Expectations," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 6, pp. 817-832, 1999.

[30] H. Mannila, H. Toivonen, and A.I. Verkamo, "Efficient Algorithms for Discovering Association Rules," *Proc. Workshop Knowledge Discovery in Databases (KDD '94),* pp. 181-192, 1994.

[31] B. Masand and G.P. Shapiro, "A Comparison of Approaches for Maximizing Business Payoff of Prediction Models," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (ACM KDD '96),* pp. 195-201, 1996.

[32] T. Mitchell, "Machine Learning and Data Mining," *Comm. ACM,* vol. 42, no. 11, pp. 30-36, Nov. 1999.

[33] E.L. Nash, *Database Marketing.* McGraw-Hill Inc., 1993.

[34] E. Pednault, N. Abe, and B. Zadrozny, "Sequential Cost-Sensitive Decision Making with Reinforcement Learning," *KDD '02: Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 259-268, 2002.

[35] F. Provost, T. Fawcett, and R. Kohavi, "The Case against Accuracy Estimation for Comparing Induction Algorithms," *Proc. 15th Int'l Conf. Machine Learning,* pp. 445-453, 1998.

[36] J.R. Quinlan, *C4.5 Programs for Machine Learning.* Morgan Kaufmann, 1993.

[37] R. Shaw and M. Stone, *Database Marketing.* John Wiley and Sons, 1988.

[38] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[39] K. Wang, Y. Jiang, and A. Tuzhilin, "Mining Actionable Patterns by Role Models," *Proc. IEEE Int'l Conf. Data Eng.,* 2006.

[40] K. Wang, S. Zhou, Q. Yang, and J.M.S. Yeung, "Mining Customer Value: From Association Rules to Direct Marketing," *Data Mining and Knowledge Discovery,* vol. 11, no. 1, pp. 57-79, 2005.

[41] A. Tuzhilin, Y. Jiang, K. Wang, and A. Fu, "Mining Patterns that Respond to Actions," *Proc. IEEE Int'l Conf. Data Mining,* pp. 669-672, 2005.

[42] Q. Yang, J. Yin, C.X. Ling, and T. Chen, "Postprocessing Decision Trees to Extract Actionable Knowledge," *Proc. IEEE Conf. Data Mining (ICDM '03),* pp. 685-688, 2003.

[43] B. Zadrozny and C. Elkan, "Learning and Making Decisions When Costs and Probabilities Are Both Unknown," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (ACM SIGKDD '01),* pp. 204-213, 2001.

[44] X. Zhang and C.E. Brodley, "Boosting Lazy Decision Trees," *Proc. Int'l Conf. Machine Learning (ICML),* pp. 178-185, 2003.

[45] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling Neural Networks: Many Could Be Better Than All," *Artifical Intelligence,* vol. 137, nos. 1-2, pp. 239-263, 2002.

Qiang Yang received the PhD degree from the University of Maryland, College Park. He is a faculty member in the Hong Kong University of Science and Technology's Department of Computer Science and Engineering. His research interests are AI planning, machine learning, case-based reasoning, and data mining. He is a senior member of the IEEE and an associate editor for the *IEEE Transactions on Knowledge and Data Engineering* and *IEEE Intelligent Systems*.

Jie Yin received the BE degree in computer science from the Xi'an Jiaotong University in 2001. Starting from the Fall of 2001, she has been a PhD student in the Department of Computer Science and Engineering at the Hong Kong of Science and Technology. Her research interests include artificial intelligence, data mining, and pervasive computing.

Charles Ling received the MSc and PhD degrees from the Department of Computer Science at the University of Pennsylvania in 1987 and 1989, respectively. Since then, he has been a faculty member of computer science at the University of Western Ontario, Canada. His main research areas include machine learning (theory, algorithms, and applications), cognitive modeling, and AI in general. He has published more than 100 research papers in journals (such as *Machine Learning*, *JMLR*, *JAIR*, *TKDE*, and *Cognition*) and international conferences (such as IJCAI, ICML, and ICDM). He has been an associate editor for the *IEEE Transactions on Knowledge and Data Engineering*, and guest editor for several journals. He is also the director of Data Mining Lab, leading data mining development in CRM, bioinformatics, and the Internet. He has managed several data mining projects for major banks and insurance companies in Canada.

Rong Pan received the BSc and PhD degrees in applied mathematics from Zhongshan University, China, in 1999 and 2004, respectively. He is a postdoctoral fellow at the Hong Kong University of Science and Technology. His research interest includes machine learning, data mining, and case-based reasoning.