Sequential Cost-Sensitive Decision Making with Reinforcement Learning

Edwin Pednault*
Mathematical Sciences Dept.
IBM T. J. Watson Res. Ctr.
Yorktown Hieghts, NY 10598
pednault@us.ibm.com

Naoki Abe
Mathematical Sciences Dept.
IBM T. J. Watson Res. Ctr.
Yorktown Hieghts, NY 10598
nabe@us.ibm.com

Bianca Zadrozny†
Dept. of Comp. Sci. and Eng.
University of Calif., San Diego
La Jolla, CA 92093
zadrozny@cs.ucsd.edu

ABSTRACT

Recently, there has been increasing interest in the issues of cost-sensitive learning and decision making in a variety of applications of data mining. A number of approaches have been developed that are effective at optimizing cost-sensitive decisions when each decision is considered in isolation. However, the issue of sequential decision making, with the goal of maximizing total benefits accrued over a period of time instead of immediate benefits, has rarely been addressed. In the present paper, we propose a novel approach to sequential decision making based on the reinforcement learning framework. Our approach attempts to learn decision rules that optimize a sequence of cost-sensitive decisions so as to maximize the total benefits accrued over time. We use the domain of targeted marketing as a testbed for empirical evaluation of the proposed method. We conducted experiments using approximately two years of monthly promotion data derived from the well-known KDD Cup 1998 donation data set. The experimental results show that the proposed method for optimizing total accrued benefits outperforms the usual targeted-marketing methodology of optimizing each promotion in isolation. We also analyze the behavior of the targeting rules that were obtained and discuss their appropriateness to the application domain.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

*Additional authors: Haixun Wang, Wei Fan, and Chidanand Apte

†The work presented herein was performed while this author was visiting IBM T.J.Watson Research Center.

SIGKDD '02 Edmonton, Alberta, Canada

1. INTRODUCTION

In the last several years, there has been an increasing interest in the machine learning community on the issue of cost-sensitive learning and decision making. Various authors have noted the limitations of classic supervised learning methods when the acquired rules are used for cost-sensitive decision making (e.g., [13, 4, 5, 17, 8]). A number of cost-sensitive learning methods have been developed [4, 17, 6] that have been shown to be superior to traditional classification-based methods.

However, these cost-sensitive methods only try to maximize the benefit (equivalently, minimize the cost) of a single decision, whereas in many applications sequences of decisions need to be made over time. In this more general setting, one must take into account not only the costs and benefits associated with each decision, but also the interactions among decision outcomes when sequences of decisions are made over time.

For example, in targeted marketing, customers are often selected for promotional mailings based on the profits or revenues they are expected to generate on each mailing when viewed in isolation. Profits or revenues are estimated using predictive models that are constructed based on historical customer-response data. To maximize expected profits for a given promotion, only those customers should be mailed whose predicted expected profit is nonzero when taking mailing costs into consideration [17].

However, the above decision policy of selecting customers to maximize expected profits on each promotion in isolation is not guaranteed to maximize total profits generated over time. It may be, for example, that the expected profit obtained by mailing the current promotion to a certain customer might exceed the current cost of mailing; however, it might also increase the profits generated by that customer in future mailings. More generally, marketing actions that are desirable from the perspective of maximizing customer loyalty over time may sacrifice immediate rewards in the anticipation of larger future revenues.

The opposite can also be true. Saturating profitable customers with frequent promotional mail might decrease the profitability of those customers, either because of the annoyance factor or because of the simple fact that everyone has budgetary limits on the amounts they are willing or are able to spend per unit time. The latter implies that a point of diminishing returns will necessarily be reached for each customer as the frequency of mail they receive increases.

In the present paper, we propose to apply the framework of reinforcement learning to address the issue of sequential decision making when interactions can occur among decision outcomes. 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.

More specifically, we adopt the popular *Markov Decision Process* model with function approximation. 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 comprising categorical and numerical data fields that characterize what is known about each customer at the time a decision is made.

When the learner takes an action, it receives a finite reward and the environment makes a probabilistic transition to another state. The goal of a learner is to learn to act so as to maximize the cumulative reward it receives (usually with future rewards discounted) as the learner takes actions and traverses through the state space.

In the example of targeted marketing, a customer, with all her past history of purchases and promotions, is in a certain state at any given point in time. When a retailer takes an action, the customer then makes a probabilistic transition to another state, possibly generating a reward. This process continues throughout the life of the customer's relationship with the retailer. The reward at each state transition is the net profit to the retailer. It takes into account both the purchases by the customer in response to the retailer's action and the cost of that action. The reward can thus be negative if the customer makes no purchases, which represents a net loss. Application of reinforcement learning to this problem amounts to maximizing the net present value of profits and losses over the life cycle of a customer.

As a proof of concept, we test our approach on the well-known donation data set from the KDD Cup 1998 competition. This data set contains approximately two years of direct-mail promotional history in each donor's data record. We transformed this data set and applied a reinforcement learning approach to acquire sequential targeting rules. The results of our experiments show that, in terms of the cumulative profits that are obtained, our approach outperforms straightforward (repeated) applications of single-event targeting rules. We also observe that the sequential targeting rules acquired by our proposed methods are often more cost-containment oriented in nature as compared to the corresponding single-event targeting rules.

The rest of the paper is organized as follows. In Section 2, we describe our proposed approach, including the reinforcement learning methods and the base learning algorithm we employ for function approximation. In Section 3, we explain our experimental setup, including the features employed and how the training data are generated from the original KDD Cup data set. We also describe the simulation model we use to evaluate the performance of our approach. In Section 4, we present the experimental results. Section 5 concludes with a discussion of results and future research issues.

2. SEQUENTIAL DECISION MAKING

For illustrative purposes, we will make use of the domain of targeted marketing throughout this paper. However, it should be noted that the approach set forth in the present paper is applicable to a wide variety of applications. We will use the term single-event targeted-marketing approach to mean one in which customers are selected for promotions based on maximizing the benefits obtained from each promotion when each is considered in isolation. A sequential targeted-marketing approach, by contrast, is one in which a series of promotional actions are to be taken over time, and promotions are selected for each customer based on maximizing the cumulative benefits that can be derived from that customer. In an ideal sequential targeted-marketing approach, each decision would be made with the goal of maximizing the net present value of all profits and losses expected now and in the future. The challenge in implementing a sequential targeted-marketing approach lies in the fact that information about the future is available only in a delayed fashion. We appeal to the apparatus of reinforcement learning to resolve this difficulty.

2.1 Reinforcement Learning

As briefly explained in the introduction, we adopt the popular Markov Decision Process (MDP) model in reinforcement learning with function approximation. For an introduction to reinforcement learning see, for example, [11, 7]. The following is a brief description of an MDP.

At any point in time, the environment is assumed to be in one of a set of possible states. At each time tick (we assume a discrete time clock), the environment is in some state s, the learner takes one of several possible actions a, receives a finite reward (i.e., a profit or loss) r, and the environment makes a transition to another state s'. Here, the reward r and the transition state s' are both obtained with probability distributions that depend on the state s and action a.

The environment starts in some initial state s_0 and the learner repeatedly takes actions indefinitely. This process results in a sequence of actions $\{a_t\}_{t=0}^{\infty}$, rewards $\{r_t\}_{t=0}^{\infty}$, and transition states $\{s_t\}_{t=1}^{\infty}$. The goal of the learner is to maximize the total rewards accrued over time, usually with future rewards discounted. That is, the goal is to maximize the cumulative reward R,

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \,, \tag{1}$$

where r_t is the reward obtained at the t'th time step and γ is some positive constant less than 1. In financial terms, γ is a discount factor for calculating the net present value of future rewards based on a given interest rate.

Generally speaking, a learner follows a certain policy to make decisions about its actions. This policy can be represented as a function π mapping states to actions such that $\pi(s)$ is the action the learner would take in state s. A theorem of Markov Decision Processes is that an optimum policy π^* exists that maximizes the cumulative reward given by Equation 1 for every initial state s_0.

In order to construct an optimum policy π^*, a useful quantity to define is what is known as the *value function* Q^π of a policy. A value function maps a state s and an action a to the expected value of the cumulative reward that would be obtained if the environment started in state s, and the learner performed action a and then followed policy π for-

ever after. $Q^\pi(s,a)$ is thus defined as

$$Q^\pi(s,a) = E_\pi\left[\sum_{t=1}^{\infty}\gamma^t r_t \;\middle|\; s_0 = s, a_0 = a\right], \quad (2)$$

where E_π denotes the expectation with respect to the policy π that is used to define the actions taken in all states except the initial state s_0.

A remarkable property of Markov Decision Processes is that the value function Q^* of an optimum policy π^* satisfies the following recurrence relation, known as the *Bellman optimality equation*:

$$Q^*(s,a) = E_r[r \mid s,a] + \gamma E_{s'}\left[\max_{a'} Q^*(s',a') \;\middle|\; s,a\right], \quad (3)$$

where the first term $E_r[r \mid s,a]$ is the expected immediate reward obtained by performing action a in state s, and the second term $E_{s'}[\max_{a'} Q^*(s',a') \mid s,a]$ is the expected cumulative reward of performing the optimum action in the transition state s' that results when action a is performed in state s.

The Bellman equation can be solved via fixed-point iteration using the following system of equations:

$$\begin{aligned} Q_0(s,a) &= R(s,a) \\ Q_{k+1}(s,a) &= R(s,a) + \gamma\sum_{s'} P(s' \mid s,a)\max_{a'} Q_k(s',a'), \end{aligned} \quad (4)$$

where $R(s,a)$ is the expected immediate reward $E_r[r \mid s,a]$ and where $P(s' \mid s,a)$ is the probability of ending up in state s' when action a is performed in state s. This solution method is known as *value iteration*. In the limit, $Q_k(s,a)$ converges to $Q^*(s,a)$ as k tends to infinity. The optimum policy is then given by

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

The use of Equation 4, however, requires knowledge of both the expected reward $R(s,a)$ for each state-action pair as well as the state transition probabilities $P(s' \mid s,a)$. In learning situations, however, these functions are unknown. The problem faced by a learner, therefore, is to infer a (near) optimum policy over time through observation and experimentation.

Several approaches are known in the literature. One popular reinforcement-learning method known as *Q-learning*, due to Watkins [15], is based on the Bellman equation (Equation 3) and value iteration (Equation 4). Q-learning estimates optimum value functions in an online fashion when the sets of possible states and actions are both finite. The method starts with some initial estimates of the Q-values for each state and then updates these estimates at each time step according to the following equation:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\big(r_{t+1} + \gamma\max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\big). \quad (5)$$

It is known that, with some technical conditions, the above procedure probabilistically converges to the optimal value function [16]. The parameter α affects the rate of convergence of the update rule, as well as the asymptotic residual error in the estimate of the value function as the time step t tends to infinity. In order to obtain both a fast rate of convergence (which requires α to be large) and small asymptotic estimation error (which requires α to be small), the value of α is usually set up to be a decreasing function of time t. To ensure convergence, it is also necessary to repeatedly try every action in every reachable state in order to accurately estimate the value function for every state-action pair.

The policy that is followed during Q-learning must therefore balance the need to explore the state space (in order to ensure convergence) against the goal of maximizing cumulative rewards over time. One approach for achieving such a balance, known as the ϵ-greedy method, is to employ a stochastic policy that chooses an action at random with probability ϵ, and that otherwise follows the policy given by the following update equation with probability $(1-\epsilon)$:

$$\pi(s_t) \leftarrow \arg\max_a Q^\pi(s_t,a). \quad (6)$$

As each action is performed, Equation 5 is first used to update the Q-value for the state just visited, and Equation 6 is then used to update the action that is to be taken (with probability $(1-\epsilon)$) the next time that state is visited. Equation 6 is also used to define the initial policy given the initial Q-value estimates by applying the update to all possible states. As with α, the value of ϵ is usually set up to be a decreasing function of t.

One drawback of Q-learning is that it has a tendency to aggressively pursue what appears to be the best possible policy based on current knowledge, even though parts of the state space have not yet been thoroughly explored. Sutton and Barto [11] provide an illustrative example of the consequences of this behavior wherein a simulated robot using ϵ-greedy Q-learning repeatedly runs itself of a cliff in order to better estimate the exact shape of the cliff edge in an attempt to find the best possible path to a goal state.

Another popular learning method, known as *sarsa* [10], is less aggressive than Q-learning in the assumptions it makes about the current knowledge of the state space. Like Q-learning, Sarsa-learning starts with some initial estimates for the Q-values that are then dynamically updated, but the update rule is somewhat different:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\big(r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\big). \quad (7)$$

In particular, there is no maximization over possible actions in the transition state s_{t+1}. Instead, the current policy π is used without updating to determine both a_t and a_{t+1}.

When the policy is not updated but is held fixed, it can be shown, with some technical conditions, that Equation 7 will probabilistically converge to the value function for the given policy. When the policy is updated according to Equation 6 in combination with ϵ-greedy search, improvements are made to the policy, but without the aggressive assumptions made by Q-learning.

2.2 Batch Reinforcement Learning with Function Approximation

In the foregoing description of reinforcement learning, two simplifying assumptions were made that are not satisfied in the current setting we are considering. The first assumption is that the problem space consists of a reasonably small number of atomic states and actions. Both the sarsa and Q-learning methods described above perform value updates

for each state-action pair, which requires that the number of such pairs be finite. In many practical applications, including targeted marketing, it is natural to treat the state space as a feature space with a large number of both categorical and real-valued features. In such cases, the state space is prohibitively large to represent explicitly, which renders the above methods impractical.

The second assumption that was made is the availability of online interaction with the environment. In applications like targeted marketing, this situation is typically not the case. In fact, it is quite the opposite. In targeted marketing, one usually has access to a very large amount of data accumulated from past transaction history from which an effective targeting strategy is to be derived. Moreover, the targeting strategy (i.e., the policy) must make simultaneous decisions for an entire population of customers, not one customer at a time. Online learning of policies, via reinforcement learning or otherwise, is not practical under these circumstances.

Bearing these factors in mind, we propose to use so-called *batch* reinforcement learning methods with *function approximation*. Batch reinforcement learning refers to a form of reinforcement learning in which the learning does not take place in an online fashion as the learner performs actions and the environment traverses states. Instead, batch leaning makes use of a potentially large volume of static training data that represents prior experience. The training data consists of sequences of states, actions, and resulting rewards. Batch learning thus reflects the realities of certain real-world applications like targeted marketing.

Function approximation amounts to representing the value function as some reasonable function of state features and actions (see, for example, [3, 12, 14]). The usual online learning approach, by contrast, assigns explicit values to explicit state-action pairs. For targeted marketing purposes, the state features can include everything that is known about a customer, such as demographic information and past transaction history.

Given such training data, batch reinforcement learning with function approximation attempts to estimate the value function $Q(s,a)$ by reformulating value iteration (i.e., Equation 4) as a supervised learning problem. In particular, on the first iteration, an estimate of the expected immediate reward function $R(s,a)$ is obtained by using supervised learning methods to predict the value of $R(s,a)$ based on the features that characterize the input state s and the input action a. On the second and subsequent iterations, the same supervised learning methods are used again to obtained successively improved predictions of $Q(s,a)$ by using variants of sarsa (Equation 7) or Q-learning (Equation 5) to recalculate the target values that are to be predicted for each iteration.

Figures 1 and 2 present pseudo-code for two versions of batch reinforcement learning: one based on sarsa, the other based on Q-learning. In both cases, the input training data D is assumed to consist of, or contain enough information to recover, *episode* data. An episode is a sequence of *events*, where each event consists of a state, an action, and a reward. Episodes preserve the temporal order in which events are observed. States $s_{i,j}$ are feature vectors that contain numeric and/or categorical data fields. Actions $a_{i,j}$ are assumed to be members of some prespecified finite set. Rewards $r_{i,j}$ are real-valued. The base learning module, Base, takes as input a set of event data and outputs a regression model Q_k that

Procedure Batch-RL(sarsa)
Premise:
A base learning module, Base, for regression is given.
Input data: $D = \{e_i | i = 1, ..., N\}$ where
$e_i = \{\langle s_{i,j}, a_{i,j}, r_{i,j}\rangle | j = 1, ..., l_i\}$
(e_i is the i-th episode, and l_i is the length of e_i.)
1. For all $e_i \in D$
$D_{0,i} = \{\langle s_{i,j}, a_{i,j}, r_{i,j}\rangle | j = 1, ..., l_i\}$
2. $D_0 = \bigcup_{i=1,...,N} D_{0,i}$
3. $Q_0 = \text{Base}(D_0)$.
4. For $k = 1$ to $final$
4.1 For all $e_i \in D$
4.1.1 For $j = 1$ to $l_i - 1$
4.1.1.1 $v_{i,j}^{(k)} = Q_{k-1}(s_{i,j}, a_{i,j}) + \alpha_k(r_{i,j} + \gamma Q_{k-1}(s_{i,j+1}, a_{i,j+1}) - Q_{k-1}(s_{i,j}, a_{i,j}))$
4.1.1.2 $D_{k,i} = \{\langle s_{i,j}, a_{i,j}, v_{i,j}^{(k)}\rangle | j = 1, ..., l_i - 1\}$
4.2 $D_k = \bigcup_{i=1,...,N} D_{k,i}$
4.3 $Q_k = \text{Base}(D_k)$
5. Output the final model, Q_{final}.

Figure 1: Batch reinforcement learning (sarsa-learning)

Procedure Batch-RL(Q)
Identical to Batch-RL(sarsa) except that 4.1.1.1 is replaced by:
4.1.1.1 $v_{i,j}^{(k)} = Q_{k-1}(s_{i,j}, a_{i,j}) + \alpha_k(r_{i,j} + \gamma \max_a Q_{k-1}(s_{i,j+1}, a) - Q_{k-1}(s_{i,j}, a_{i,j}))$

Figure 2: Batch reinforcement learning (Q-learning)

maps state-action pairs (s,a) to their estimated Q-values $Q_k(s,a)$. In the two procedures shown in these figures, and in all variants considered later in this paper, we set α_k to be α/k for some positive constant $\alpha < 1$.

Note that the only difference between the two methods is the equation that is used to recalculate target Q-values at each iteration. In the case of Figure 1, Equation 7 is used at line 4.1.1.1; in the case of Figure 2, Equation 5 is used.

2.3 Base Regression Method: ProbE

As the base learning method, we employ the multivariate linear-regression tree method implemented in the IBM ProbE data mining engine [9, 1]. This learning method produces decision trees with multivariate linear regression models at the leaves. Regression models are constructed as trees are built, and splits are selected to maximize the predictive accuracies of the regression models in the resulting child nodes. Feature selection is performed as part of both the tree building process (i.e., split selection) and the regression modeling process (i.e., variable selection). Likewise, pruning is performed both on the trees and on the regression models at the nodes.

In our experiments, we compared the conventional single-event targeting strategy of selecting customers for each marketing campaign so as to maximize the profit of each campaign when viewed in isolation, versus the proposed sequential targeting strategy of selecting campaigns for each customer so as to maximize the cumulative profits generated by

each customer. To ensure a fair comparison, ProbE's multivariate linear-regression tree method was used to construct models for both targeting strategies. A single-event targeting strategy was constructed by applying the procedure shown in Figure 1 with *final* set to a value of zero. Doing so causes the reinforcement learning loop at line 4 to be omitted, thereby producing a policy that maximizes immediate reward. Because the same base learning algorithm is used for constructing both single-event and sequential marketing strategies, any differences in performance that are observed should reflect inherent differences in the strategies.

2.4 Sampling for Enhanced Scalability

An important issue in any data mining application is that of scalability. This is especially critical in applying our approach to domains like targeted marketing. Not only can the volume of the business transaction data be huge (well over millions of records), but the iterative nature of reinforcement learning requires generating a sequence of models from such data.

In an effort to lighten the load of data size, we consider a series of sampling methods that are specifically designed for batch reinforcement learning. One obvious approach is random sampling. However, more efficient sampling methods can be obtained by taking into account the episodic nature of the data and the objectives of the learning strategy.

Recall that in batch reinforcement learning, training is performed on data that have already been collected, presumably using some *sampling* or *control* policy. This is to be contrasted with the online learning setting, in which the learner has control over the sampling policy. However, in domains that involve a potentially huge amount of data, it is possible to simulate online reinforcement learning with a particular policy by electing to use just those data that conform to the policy.

Based on this latter idea, we propose a sampling method we call *Q-sampling* in which only those states are selected that conform to the condition that the action taken in the next state is the best action with respect to the current estimate of the Q-value function. The value update is akin to Equation 7 used in sarsa-learning, but the effect of the learning that occurs corresponds to Equation 5 used in Q-learning because the sampling strategy ensures that $Q(s_{t+1}, a_{t+1}) = \max_{a'} Q(s_{t+1}, a')$.

Taking this line of reasoning a step further, it is also possible to look ahead an arbitrary number of states and select only those states in which optimal actions are taken in all of those subsequent states. In this case, it makes sense to take advantage of the lookahead that is being done for updating the Q-value. There is a well-known method of value update with lookahead known in the literature as TD(λ). This method updates the value function estimate using a weighted average of the Q-value estimate from the last state and the discounted partial sums of rewards obtained over the next several states. More precisely, TD(λ) uses the following update rule for estimating Q-values:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_t^{\lambda} - Q(s_t, a_t) \right) , \qquad (8)$$

where

$$R_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} ,$$

Procedure Batch-RL(Random-sampling)
Premise:
A base learning module, Base, for regression is given.
Input data: $D = \{e_i | i = 1, ..., N\}$ where
$e_i = \{\langle s_{i,j}, a_{i,j}, r_{i,j} \rangle | j = 1, ..., l_i\}$
(e_i is the i-th episode, and l_i is the length or number of events in e_i.)
1. For all e_i in *randomly selected* subset R_0 of D
$D_{0,i} = \{\langle s_{i,j}, a_{i,j}, r_{i,j} \rangle | j = 1, ..., l_i\}$
2. $D_0 = \bigcup_{e_i \in R} D_{0,i}$
3. $Q_0 = \text{Base}(D_0)$.
4. For $k = 1$ to *final*
4.1 For all e_i in *randomly selected* subset R_k of D
4.1.1 For $j = 1$ to $l_i - 1$
4.1.1.1 $v_{i,j}^{(k)} = Q_{k-1}(s_{i,j}, a_{i,j}) + \alpha_k(r_{i,j} + \gamma Q_{k-1}(s_{i,j+1}, a_{i,j+1}) - Q_{k-1}(s_{i,j}, a_{i,j}))$
4.1.1.2 $D_{k,i} = \{\langle s_{i,j}, a_{i,j}, v_{i,j}^{(k)} \rangle | j = 1, ..., l_i - 1\}$
4.2 $D_k = \bigcup_{e_i \in R} D_{k,i}$
4.3 $Q_k = \text{Base}(D_k)$
5. Output the final model, Q_{final}.

Figure 3: Batch reinforcement learning (Random-sampling)

Procedure Batch-RL(Q-sampling)
Identical to Batch-RL(Random-sampling) except that Block 4.1 is replaced by:
4.1 For all e_i in *randomly selected* subset R_k of D
4.1.1 $D_{k,i} = \emptyset$
4.1.2 For $j = 1$ to $l_i - 1$
If $Q_{k-1}(s_{i,j+1}, a_{i,j+1})) = \max_a Q_{k-1}(s_{i,j+1}, a)$
Then
4.1.2.1 $v_{i,j}^{(k)} = Q_{k-1}(s_{i,j}, a_{i,j}) + \alpha_k(r_{i,j} + \gamma Q_{k-1}(s_{i,j+1}, a_{i,j+1}) - Q_{k-1}(s_{i,j}, a_{i,j}))$
4.1.2.2 $D_{k,i} = D_{k,i} \cup \{\langle s_{i,j}, a_{i,j}, v_{i,j}^{(k)} \rangle\}$

Figure 4: Batch reinforcement learning (Q-sampling)

and where $R_t^{(n)}$ is the so-called n-step return defined as follows.

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^n Q(s_{t+n}, a_{t+n}) .$$

We employ this update rule in our sampling method based on multistep lookahead, and thus name it TD(λ)-sampling.

Pseudo-code for the above three sampling methods is presented in Figures 3, 4, and 5. The Q-sampling and TD(λ)-sampling strategies are presented as variants of the basic random sampling method shown in Figure 3. Note that different random samples are selected at each iteration at line 4.1 of these algorithms. This is done to reduce overfitting effects as updated Q-value functions are learned at line 4.3.

3. EXPERIMENTAL SETUP

As mentioned in the introduction, we performed preliminary evaluation experiments using an existing benchmark data set in the general domain of targeted marketing, and using simulation. We use the well-known donation data set

Procedure Batch-RL(TD(λ)-sampling)
Identical to Batch-RL(Random-sampling) except that Input and Block 4.1 are replaced respectively by:

Input data: $D = \{e_i | i = 1, ..., N\}$ where
$e_i = \{\langle s_{i,j}, a_{i,j}, r_{i,j}\rangle | j = 1, ..., l_i\}$
λ $(0 < \lambda < 1)$ is a parameter for TD(λ)
L is a parameter for the number of lookahead steps

4.1 For all e_i in *randomly selected* subset R_k of D
4.1.1 $D_{k,i} = \emptyset$
4.1.2 For $j = 1$ to $l_i - L$
If $Q_{k-1}(s_{i,j+1}, a_{i,j+1})) = \max_a Q_{k-1}(s_{i,j+1}, a)$
and ... and
$Q_{k-1}(s_{i,j+L}, a_{i,j+L})) = \max_a Q_{k-1}(s_{i,j+L}, a)$
Then
4.1.2.1 For each n, $1 < n \geq L$
$R^{(n)}_{i,j} = r_{i,j+1} + \gamma r_{i,j+2} + ... + \gamma^n Q(s_{t+n}, a_{t+n})$
4.1.2.2 $v^{(k)}_{i,j} = Q_{k-1}(s_{i,j}, a_{i,j})$
$+\alpha_k((1-\lambda)\sum_{n=1}^{L} \lambda^{n-1} R^{(n)}_{i,j}$
$-Q_{k-1}(s_{i,j}, a_{i,j}))$
4.1.2.3 $D_{k,i} = D_{k,i} \cup \{\langle s_{i,j}, a_{i,j}, v^{(k)}_{i,j}\rangle\}$

Figure 5: Batch reinforcement learning (TD(λ)-sampling)

from KDD Cup 1998, which contains demographic as well as promotion history data as our *episode* data. The episode data are used in two ways: (1) A series of event data are generated from the episode data and are used for reinforcement learning to obtain a targeting policy; (2) Models of response probability and donation amount are estimated using similar event data generated from the episode data, which are used to obtain an MDP simulation model. We then use this MDP model to run simulation experiments for evaluating the acquired targeting policy.

3.1 Data Set

The donation data set we used from the KDD Cup 1998 competition is available from the UCI KDD repository [2] along with associated documentation. This data set contains information concerning direct-mail promotions for soliciting donations. The information includes demographic data as well as promotion history for 22 campaigns that were conducted monthly over an approximately two year period. The campaign information includes whether an individual was mailed or not, whether he or she responded or not, and how much was donated. Additionally, if the individual was mailed, the date of the mailing is available (month and year), and if the individual then responded, the date of the response is available.

We used the training data portion of the original data set, which contains data for approximately 100 thousand selected individuals.[1] Out of the large number of demographic features contained in the data set, we selected only age and income bracket. Based on the campaign information in the data, we generated a number of temporal features that are designed to capture the *state* of that individual at the time of each campaign. These features include the frequency of gifts, the recency of gifts and promotions, the number of recent promotions in the last 6 months, etc., and are summarized in Table 1.

[1]This is contained in "cup98lrn.zip" on the URL "http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html".

Table 1: Features Used in Our Experiments

Features	Descriptions
age	individual's age
income	income bracket
ngiftall	number of gifts to date
numprom	number of promotions to date
frequency	ngiftall / numprom
recency	number of months since last gift
lastgift	amount in dollars of last gift
ramntall	total amount of gifts to date
nrecproms	num. recent promotions (last 6 months)
nrecgifts	num. recent gifts (last 6 months)
totrecamt	total amount of recent gifts (6 months)
recamtpergift	recent gift amount per gift (6 months)
recamtpergift	recent gift amount per prom (6 months)
promrecency	months since last promotion
timelag	months between first promo and gift
recencyratio	recency / timelag
promrecratio	promrecency / timelag
action	was mailed in current promotion (0,1)

It should be noted that, because a 6 month history window is used to summarize recent promotion-response behavior, the first 6 monthly campaigns present in each data record are reserved as the first 6-month history window when extracting feature vectors. Hence, only the last 16 of the 22 campaigns are used for episode data.

It should also be noted that many of these features are not explicitly present in the original data set, but instead are computed from the data by traversing through the campaign history data. In the terminology of general batch reinforcement learning introduced in Section 2, the demographic and campaign history data for each individual constitute an episode, from which the sequence of events—state, action and reward triples—may be *recovered*. For example, the feature named *numprom* in the original KDD Cup data takes on a single value for each individual, and equals the total number of promotions mailed to that individual prior to the last campaign. In our case, *numprom* is computed for *each campaign* by traversing the campaign history data backwards from the last campaign and subtracting one every time a promotion was mailed in a campaign. Similarly, *ngiftall* in the original data set is just the total number of gifts to date as of the last campaign, but here this is computed for each campaign by starting at the last campaign and subtracting one each time a gift was made.

We note that we did not make use of the RFA codes included in the original data, which contain the so-called Recency/Frequency/Amount information for the individuals, since they did not contain enough information to recover their values for campaigns that were not mailed to the individuals.

3.2 Evaluation by Simulation

We evaluated our approach via simulation using an estimated MDP for the donation data set. The MDP we constructed consists mainly of two estimation models: one model $P(s, a)$ for the probability of response as a function of

the state features and the action taken, and the other $A(s,a)$ for the amount of donation *given* that there is a response, as a function of the state features and the action. The $P(s,a)$ model was constructed using ProbE's naive-Bayes tree algorithm, while $A(s,a)$ was constructed using ProbE's linear-regression tree algorithm.

Given models for $P(s,a)$ and A(s,a), it is possible to construct an MDP in the following way. First, the immediate reward $r(s,a)$, for a given state, action pair can be specified using the two models as follows: Flip a coin with bias $P(s,a)$ to determine if there is a response. If there is no response, then the amount of donation is zero. If there is a response, then determine the amount of donation as $A(s,a)$. The reward obtained is the amount of donation minus the mailing cost, if any. Next, the state transition function can be obtained by calculating the transition of each feature using the two models. For example, ngiftall (number of gifts to date) is incremented by one if the above coin with bias $P(s,a)$ came up heads; otherwise, it remains unchanged. Similarly, numprom (number of promotions to date) is incremented if the action taken was 1, and remains constant otherwise. Using the above two features, frequency (i.e., ngiftall / numprom) can be computed. Updates for other features are computed similarly.

Given the above functional definition of an MDP, we conducted our evaluation experiment as follows. Initially, we selected a large enough subset (5,000) of the individuals, and set their initial states to correspond to their states prior to a fixed campaign number (in experiments reported here, campaign number 7 was used). We then *throw* all these individuals to the MDP and use the value-function output of our batch reinforcement learning procedure to make decisions about what actions to take for each individual. Utilizing the response probability model and the expected amount model, we compute the resulting rewards and next states. We record the rewards thus obtained, and then go on to the next campaign. We repeat this procedure 20 times, simulating a sequence of 20 virtual campaigns.

The use of a simulation model for evaluation raises a question concerning our premise that online interaction with an MDP is infeasible. A natural inclination may be to use the above MDP as a model of the environment, and use an online learning method (such as online versions of sarsa and Q-learning) to estimate the value function from interactions with it. Our view is that the human behavior in application domains such as targeted marketing is too complicated to be well captured by such a simplified model of MDP. We are using the simulation model to evaluate the policy obtained by our method, only as a preliminary experiment prior to a real-world evaluation experiment.

4. EXPERIMENTAL RESULTS

We report on the results of our preliminary experiments using a simulation model. We evaluate our proposed approach with respect to a number of performance measures, including the total life-time profits obtained and the qualitative behaviors of the acquired targeting rules.

4.1 Life-Time Profits

We first consider the most obvious, and arguably most important, measure of total cumulative benefits (life-time profits) obtained by the competing methods. In particular, we compare the life-time profits obtained by two variants of reinforcement learning to that obtained by the single-event targeting method. Here, the single-event method is obtained by using the base regression module to learn a model of the expected immediate rewards (profits) as a function of state features and the action, and then mailing to an individual just in case the expected immediate reward for mailing exceeds that for not mailing, at each campaign. Notice that, since the state features contain temporal information, such as recency, frequency, and the number of recent promotions, the targeting decisions obtained this way are sensitive to the past history and, hence, to the campaign number.

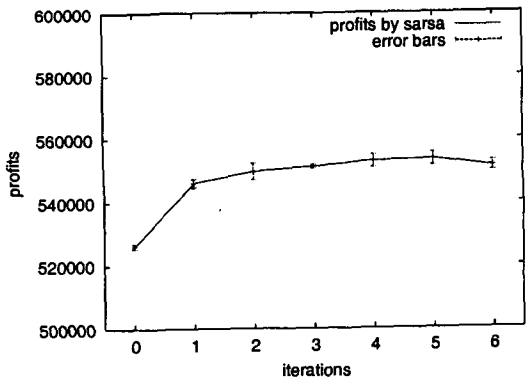

Figure 6: Total life-time profits obtained by BatchRL(sarsa).

Figure 6 shows the total life-time profits obtained by the sarsa-learning version of batch reinforcement learning, plotted as a function of the number of value iterations performed. The plots were obtained by averaging over 5 runs, each run with episode data size 10,000, which translates to training data size of 160,000 for reinforcement learning (i.e., 10,000 episodes times 16 campaigns). The total profits are obtained using the simulation model as described in the previous section, and totaled over 20 campaigns. The error bars shown in the graph are the *standard errors* calculated from the total profits obtained in the five independent runs, namely

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(P_i - \bar{P})^2/n - 1}{n}} \qquad (9)$$

where P_i is the total profit obtained in the i-th run, $\bar{P}$ is the average total profit, and n is the number of runs (5 in this case). Note that the iteration number 0 corresponds to the single-event targeting method. Thus, the total life-time profits obtained in later iterations represent statistically significant improvements over the single-event approach.

Next, we compare the total profits obtained by different versions of batch reinforcement learning methods, sarsa and Q-learning. Figure 7 shows the total profits obtained by these two versions, again using 10,000 episode data and averaged over five runs. These results show that, in this particular case, Q-learning resulted in a more profitable policy than sarsa-learning, although the statistical significance of the difference was unconvincing with our limited experimentation. This is indeed not surprising considering that Q-learning attempts to obtain the optimal policy, whereas sarsa-learning is trying to perform a local improvement based on the current policy. In the context of batch reinforcement learning,

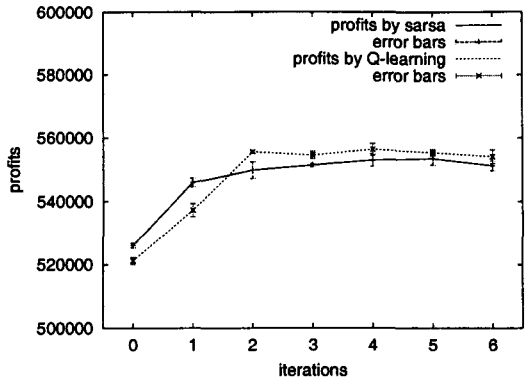

Figure 7: Total profits obtained by BatchRL(sarsa) and BatchRL(Q).

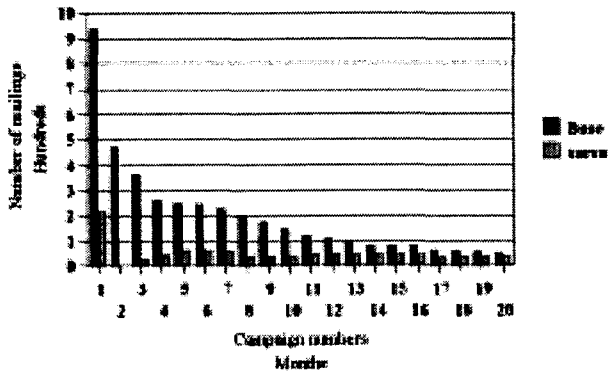

Figure 8: The number of mailings by a policy obtained by BatchRL(sarsa).

this current policy is in fact the policy that was used in practice when the data was obtained.

4.2 Rule Behavior: Number of Mailings

In addition to analyzing the profits that were attained, we also examined how the behavior of the obtained models differ. Figure 8 shows the number of individuals mailed in each of the twenty campaigns. The number of individuals considered in each campaign was 10 thousand for this experiment. Clearly, the policy obtained by sarsa-learning is significantly more cost-containment oriented than the policy produced by the single-event strategy. It is also interesting to note that the model produced by reinforcement learning seems to exhibit rather sophisticated temporal characteristics. That is, it initially mails to a large number of individuals, waits to observe the responses, and then starts sending again to very selected segments. This type of sequential targeting strategy seems to make sense intuitively, but it also appears highly unlikely that a real-world retailer actually employs a strategy like this. It appears to be a rather surprising and nontrivial discovery made by our approach to sequential targeted marketing.

We also examined the policy obtained by the Q-learning version of batch reinforcement learning. In many cases, it was outputting policies that mail to almost all individuals. It was indeed the case that the simulation model we used was crediting even more profits to this strategy.[2] Since Q-learning deviates more from the current policy and searches for a global optimum, it appears to have found a policy that was significantly different in nature from the current policy but was more profitable. Sarsa-learning, on the other hand, works more closely with the current policy and tries to improve it, and as a result it seems to obtain a similar policy to the current one, but is more cost-containment oriented and is more practical.

The profitability of a policy is obviously an important criterion in the choice of marketing strategies in targeted marketing. There are, however, other considerations that impact decision making in practice. Observations such as the one made above may prove that, in practice, the more conservative sarsa-learning may be more readily accepted than the more aggressive Q-learning method.

4.3 Rule Behavior: Profits per Campaign

How is it possible that the cost-containment oriented policies generated by our reinforcement learning can approach achieve greater profits? To probe into this question further, we examined how the amount of profits obtained changes over time as the campaigns proceed. Figure 9 shows the profits obtained by each policy per campaign, for the twenty campaigns considered in the simulation. In the graph, it is clearly seen that the policy produced by the reinforcement learning approach is settling for lower profits initially in order to achieve greater profits in later campaigns. This is an indication that the reinforcement learning approach, which takes into account the long-term effects, is indeed successful at finding targeting rules that maximize life-time profits rather than immediate profits.

Note also that profits are obtained during campaign 2 even though almost no one was mailed during this campaign. These profits represent delayed responses to previous campaigns that had already been mailed. Thus, donations are credited to the months in which they received, not to the campaigns that triggered those donations. This delayed-response approach to credit assignment is necessary in order to correctly formulate sequential decision making as a Markov decision process.

4.4 Comparison of Sampling Methods

We also conducted experiments to examine the effect of using the various sampling methods we proposed in Section 2 with respect to the quality of the output models and the required computational resources. Figure 10 plots the total life-time profits attained using different sampling methods as a function of the number of value iterations that were performed. The sampling methods employed are random sampling, Q-sampling, TD(λ)-sampling with 2-step lookahead, and TD(λ)-sampling with 3-step lookahead. Similarly, Figure 11 shows how the sample size (i.e., the number of data

[2]We note that this is not as unlikely as it may seem. The KDD Cup 98 data set contains data about individuals who used to actively contribute and then stopped being active. The test data used for the cup competition was the data for the last campaign. So the models were trying to find a strategy that somehow determines who among these defectors are likely to be won back. This is not the case in our simulation. We took a certain campaign number in the first half of the two years, when most individuals were still active, and then started simulating from there.

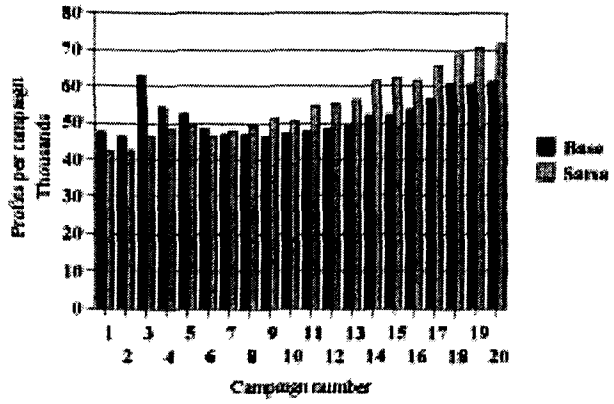

Figure 9: Profits per campaign obtained by BatchRL(sarsa) and single-event method.

records selected at each iteration) changes as a function of the iteration number. In these experiments, an episode data set of size 10,000 was used, and in each iteration 5,000 out of them were randomly sampled, resulting in 80,000 event data. The latter three sampling methods further reduced the sample size by filtering out those data that did not meet the respective conditions (as specified in Figures 4 and 5.)

Comparing these two graphs clearly shows the advantage of the proposed sampling methods. That is, confining the training data to those that conform to the currently estimated greedy policy can result in a substantial saving in the sample size and therefore in the required computational resources (time and space), without compromising the quality of the output model in terms of the total life-time profits obtained. In fact, the three sampling methods resulted in more profitable policies in these particular experiments. As one increases the number of lookahead steps by one (1 for Q-sampling, 2 for TD-2, and 3 for TD-3), the sampling size is roughly cut in half. This is natural considering that there are two possible actions (mail or do not mail), and only one of them conforms to the current policy. In many practical applications, there are likely to be a large number of possible actions to choose from. In such cases, one can expect correspondingly large savings in computational resources.

5. CONCLUSIONS

We presented a novel approach to sequential cost-sensitive decision making based on the framework of reinforcement learning, and reported on the results of preliminary experiments performed using a simulation model. In the future, we plan to conduct large-scale, real-world experiments to further demonstrate the effectiveness of the proposed approach, as well as to assess its practicality in specific application domains.

6. ACKNOWLEDGMENTS

We thank Fateh Tipu for his generous help in setting up the environment for our experiments.

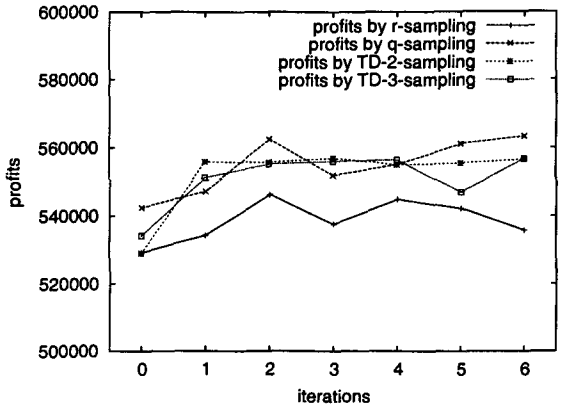

Figure 10: Life-time profits obtained by competing sampling methods as a function of value iteration number: (1) Random sampling, (2) Q-sampling, (3) TD-2-sampling, and (4) TD-3-sampling.

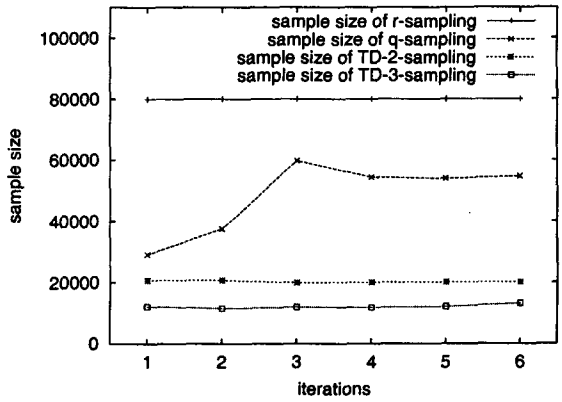

Figure 11: Sample size for competing sampling methods as a function of value iteration number: (1) Random sampling, (2) Q-sampling, (3) TD(λ)-sampling(2), and (4) TD(λ)-sampling(3).

7. ADDITIONAL AUTHORS

Haixun Wang (Distributed Computing Dept., IBM T. J. Watson Research Center, Hawthorne, NY 10532, haixun@us.ibm.com), **Wei Fan** (Distributed Computing Dept., IBM T. J. Watson Research Center, Hawthorne, NY 10532, weifan@us.ibm.com), and **Chidanand Apte** (Mathematical Sciences Dept., IBM T. J. Watson Research Center, Yorktown Hieghts, NY 10598, apte@us.ibm.com).

8. REFERENCES

[1] C. Apte, E. Bibelnieks, R. Natarajan, E. Pednault, F. Tipu, D. Campbell, and B. Nelson. Segmentation-based modeling for advanced targeted marketing. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 408–413. ACM, 2001.

[2] S. D. Bay. UCI KDD archive. Department of Information and Computer Sciences, University of California, Irvine, 2000. http://kdd.ics.uci.edu/.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[4] P. Domingos. MetaCost: A general method for making classifiers cost sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164. ACM Press, 1999.

[5] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Aug. 2001.

[6] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. 16th International Conf. on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[8] D. D. Margineantu and T. G. Dietterich. Bootstrap methods for the cost-sensitive evaluation of classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 583–590. Morgan Kaufmann, San Francisco, CA, 2000.

[9] R. Natarajan and E. Pednault. Segmented regression estimators for massive data sets. In *Second SIAM International Conference on Data Mining*, Arlington, Virginia, 2002. to appear.

[10] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Departement, 1994. Ph.D. thesis.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[12] J. N. Tsitsiklis and B. V. Roy. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[13] P. Turney. Cost-sensitive learning bibliography. Institute for Information Technology, National Research Council, Ottawa, Canada, 2000. http://extractor.iit.nrc.ca/bibliographies/cost-sensitive.html.

[14] X. Wang and T. Dietterich. Efficient value function approximation using regression trees. In *Proceedings of the IJCAI Workshop on Statistical Machine Learning for Large-Scale Optimization*, 1999.

[15] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, 1989.

[16] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[17] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, 2001.