Sequential Cost-Sensitive Decision-Making with Reinforcement Learning

Edwin Pednault

Naoki Abe

Bianca Zadrozny

(also Haixun Wang, Wei Fan and Chid Apte)

IBM T.J. Watson Research Center

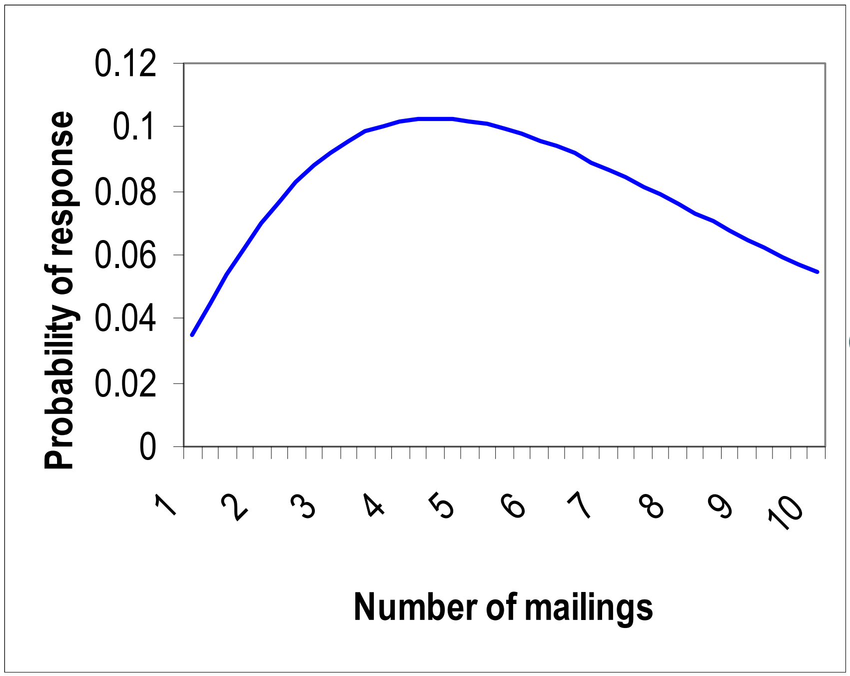
Introduction

- Cost-sensitive learning methods learn policies that attempt to minimize the cost of a single decision.
- However, in many applications, sequences of decisions must be made over time.
- In this case, the optimal policy must consider the interactions between decisions.

“Why do I receive so much junk mail?”

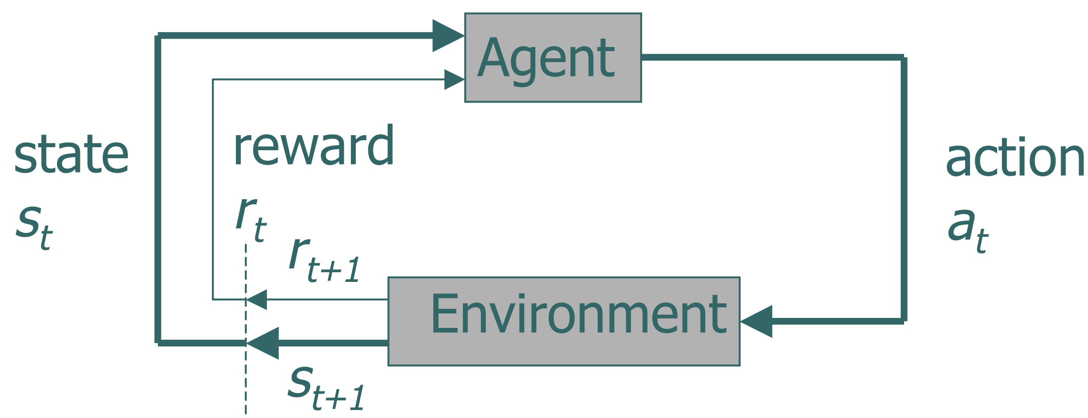
- Current approaches to targeted marketing attempt to maximize expected profit considering each campaign **in isolation**.
- This is a **greedy** approach which often results in over-mailing.
- A better approach is to maximize profit over **a series of campaigns**.

Priming and Saturation



- Priming: choosing an action that is not profitable immediately but that increases the probability of response in the future.
- Saturation: after a certain number of mailings, the probability of response per mailing decreases as more mail is sent.
  - Budgetary limits
  - Annoyance factor

Reinforcement learning



- In state s_t, the agent chooses action a_t according to a policy $\pi(s)$, and the environment transitions probabilistically.
- By the Markov assumption, the next state s_{t+1} and the reward r_{t+1} depend only on s_t and a_t ,
- RL methods specify how to change the policy $\pi(s)$ as a result of experiments to maximize the cumulative reward:

$$R = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$$

Reinforcement learning for targeted marketing

- States: contain customer's demographic and behavioral features, and possibly environment features such as seasonal information and inventory data.
- Actions
  - mail
  - do not mail

  (possible have different types of mailings)
- Rewards
  - Positive: revenue received from customer
  - Negative: cost of mailing

Value function

- A value function gives the expected return for taking action a in state s and following a policy π thereafter:

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

- The optimal policy π^* has a value function $Q^*(s,a)$ such that $Q^*(s,a) \geq Q^{\pi}(s,a)$ for all s and a.
- If the expected reward and the transition probabilities are known for every state and action, we can compute the optimal value function $Q^*(s,a)$.
- Using $Q^*(s,a)$ we can compute the optimal policy:

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

Q-learning

- In learning situations where the environment parameters, the learner needs to infer a good policy through observation.
- Q-learning starts with an initial guess of $Q(s,a)$ and then updates it at each time step according to

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

which can be rewritten as

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

- Convergence to the optimal policy is guaranteed if every action is repeatedly tried in every reachable state and α decreases with time (use ε-greedy policy).

Sarsa

- Instead of maximizing over possible actions, we can choose the next action based on the current policy:

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

- The name "sarsa" comes from the quintuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ used in the update rule.
- Sarsa also converges to the optimal policy given the same conditions as needed for Q-learning convergence.
- However, the policies generated during the learning process tend to be more conservative.

Function Approximation

- Standard RL methods assume that the number of states is finite.
- But in targeted marketing each state consists of a large number of categorical and real-valued features representing a customer, resulting in a large state space.
- A regression method is used to approximate the value function, generalizing it to states that have never been seen.

Batch Reinforcement Learning

- Standard RL methods assume that on-line interaction with the environment is possible.
- In targeted marketing and other applications, it is not possible to directly interact with the environment.
- But a large amount of data describing past transactions is available.
- Batch RL uses static training data consisting of episodes, which are sequences of state-action-reward triples:

$$\langle (s_0, a_0, r_0), (s_1, a_1, r_1), \ldots, (s_l, a_l, r_l) \rangle$$

where l is the length of an episode.

Batch-RL (sarsa)

- Using regression, we learn an initial Q-function mapping states and actions to immediate rewards.
  1. Let $e_i = \langle (s_0, a_0, r_0), \ldots, (s_l, a_l, r_l) \rangle$ be an episode.
  2. For $j=1$ to $l-1$
     $$v_j = (1-\alpha)Q(s_j, a_j) + \alpha(r_j + \gamma Q\ (s_{j+1}, a_{j+1}))$$
  3. $D_i = \left\{ \left\langle s_j, a_j, v_j \right\rangle \mid j = 1, \ldots, l_i - 1 \right\}$

- We repeat this procedure for each episode e_i, and obtain $D = \bigcup_{i=1,\ldots,N} D_i$.
- We then learn a new Q-function using D.
- This process is repeated for a number of iterations.

Regression Method: ProbE

- The IBM ProbE™ learning method produces decision trees with multivariate linear regression models at the leaves.
- Feature selection and pruning are performed both at the tree level and at the node level.

Evaluation by Simulation (1)

- Because we cannot directly interact with the environment, it is not straightforward to evaluate a learned policy.
- We construct a model of the environment by estimating the following functions:
  - $P(s,a)$: the probability of response as a function of the state and action.
  - $A(s,a)$: the amount of donation given that there is a response, as a function of the state and action.

Evaluation by Simulation (2)

- The immediate reward $r(s,a)$ can be determined by flipping a coin with bias $P(s,a)$ to determine if there is a response:
  - If there is no response $r(s,a)=0$
  - If there is a response $r(s,a)=A(s,a) - c$, where c is the cost of mailing.
- The next state can be found by updating each state variable.
  - For example, **`ngiftall`** is incremented by one if there was a response.
- We select a number of individuals and start the simulation by setting their initial states to be their actual states prior to a certain campaign.
- From then we use the policy to select actions and the model to calculate the rewards and next state for each individual, repeating this for the sequence of campaigns.

Experimental Setup

- We use the donation dataset from the KDD-98 competition.
- It contains demographic data for about 100K individuals (training set), along with the promotion history of 22 campaigns:
  - whether the individual was mailed or not
  - whether the individual responded or not
  - if the individual was mailed: date of mailing
  - if the individual responded: date of response
- Based on the campaign information in the data, we compute a number of temporal features that capture the state of the individual at the time of each campaign.

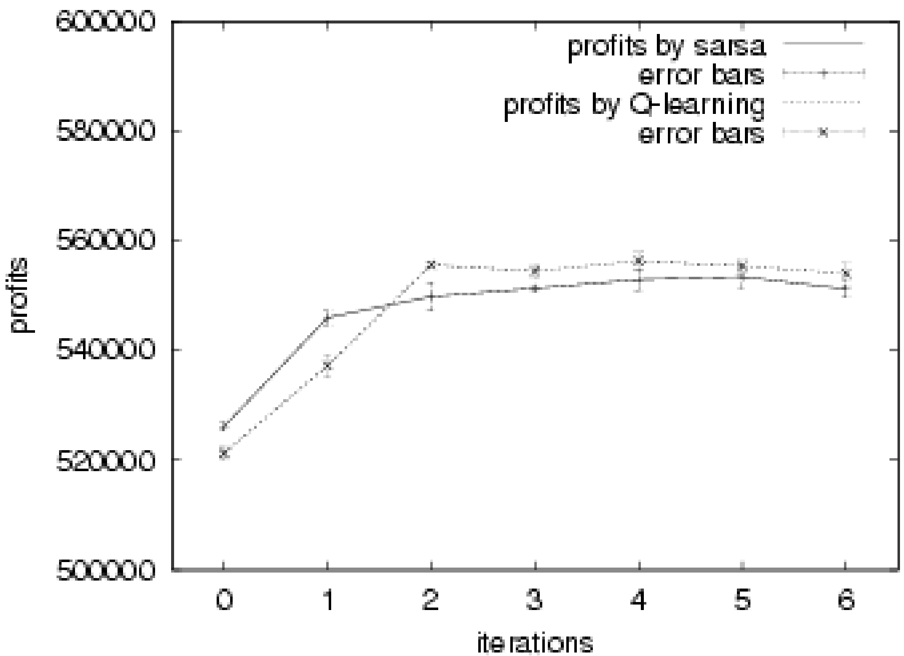
State representation

Variable	Description
`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 promotions since last gift
`lastgift`	amount of dollars of last gift
`ramntall`	total amount of gifts to date
`nrecproms`	number or recent promotions (last 6 months)
`nrecgifts`	number of recent gifts (last 6 months)
`totrecamnt`	total amount of recent gifts (last 6 months)
`recamntpergift`	recent amount per gift (last 6 months)
`recamptperprom`	recent amount per promotion (last 6 months)
`promrecency`	number of months since last promotion
`timelag`	number of months between first promotion and gift
`recencyratio`	recency/timelag
`promrecratio`	promrecency/timelag

Experimental Results

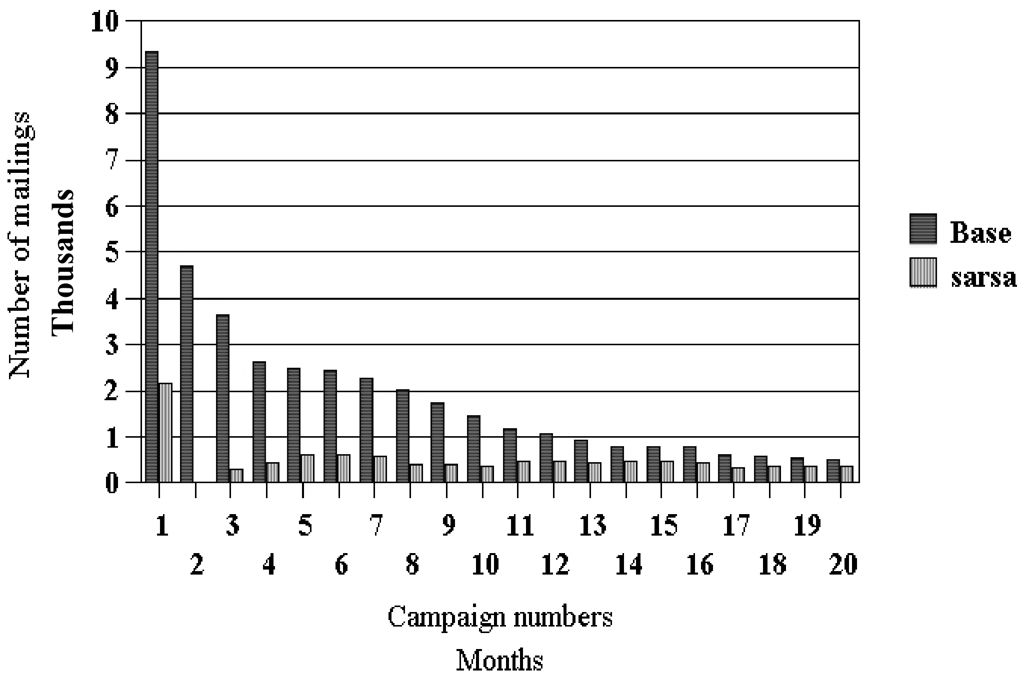
- We compare the policies learned by Q-learning and sarsa to the single-event targeting method.
- As the single-event method we use ProbE to predict immediate rewards (profits) as a function of state and action.
- We mail an individual if the expected reward for mailing exceeds that for not mailing.

Life Time Profits



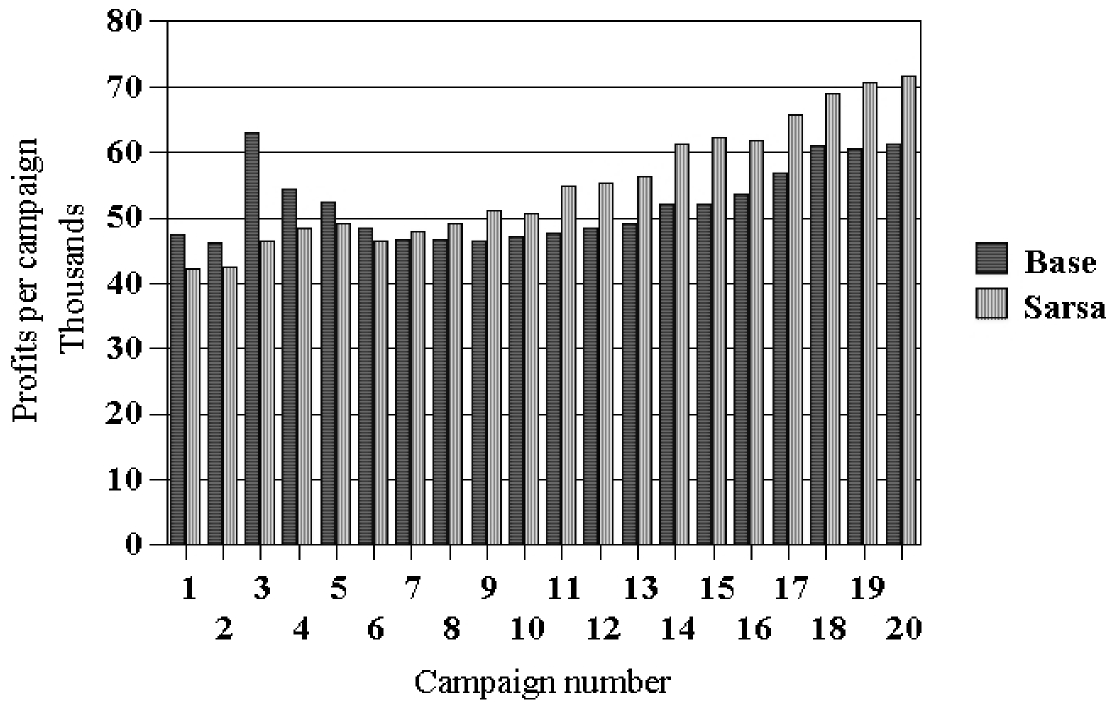
- Iteration number 0 corresponds to the single-event method.
- The plots were obtained by averaging over 5 runs, using 10,000 individuals and 16 campaigns for training.
- Both Q-learning and sarsa are significantly better than the single-event method.
- Q-learning is slightly better than sarsa, which is not surprising given that sarsa performs a local improvement based on the current policy.

Rule behavior: Number of Mailings



- The policy obtained by sarsa is significantly more cost-containment than the single-event one.
- Q-learning creates a policy that mails to almost all individuals.

Rule Behavior: Profits per Campaign

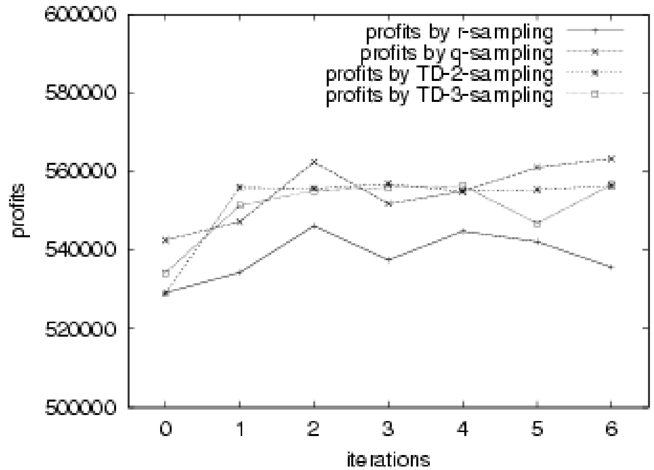


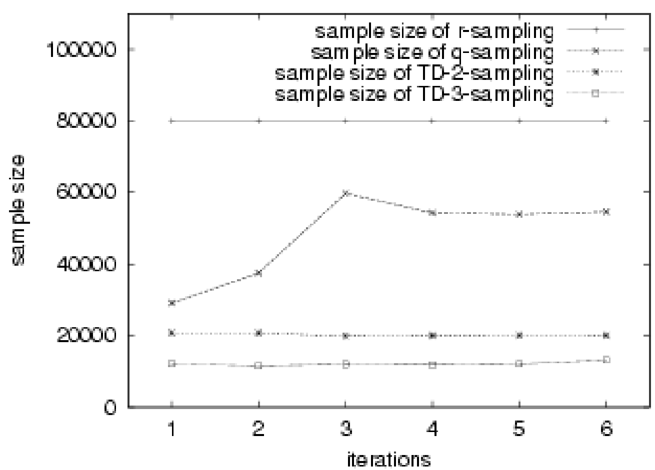
- The policy produced by sarsa generates less profits in the beginning and more profits in the end, as expected for RL methods.

Sampling for Enhanced Scalability

- We can make our methods scale to a huge number of records by using random sampling.
- We can also simulate on-line reinforcement learning with a particular policy by using just the data that conform to the policy.
  - Q-sampling: use only states for which the action taken in the next state is optimal according to the current estimate of the Q-value function.
  - TD(λ)-sampling: look ahead an arbitrary number of states and select only those states in which optimal actions are taken in all subsequent states.

Comparison of Sampling Methods





- By using Q-sampling and TD-sampling, we can substantially reduce the data set size, without compromising performance.
- As the look-ahead size increases by 1, the sampling size is roughly cut in half (because there are two possible actions).

Conclusions

- We presented a novel approach to sequential cost-sensitive decision-making, based on the reinforcement learning framework.
- The simple model used for evaluation may not be capturing the behavior of customers, so experimentation in the real-world is needed.
- Another possibility is to use the simulation model to learn a policy and compare it to the policies learned by the batch methods.