Graphical Real-Valued Multi-Armed Bandit Using SVMs

CS221 Fall 2014: Homework [p-final]

Keenon Werling
keenon@stanford.edu

Amy Bearman
abearman@stanford.edu

Bhaven Patel
bhavenp@stanford.edu

1 Introduction

1.1 Motivation

A/B testing is a simple, popular way to make changes to website design. It involves testing two versions of a web page – an A version (the control) and a B version (the experimental) – with real users, and then measuring the effect that each version has on some metric, such as click rate or revenue. After the two versions are compared against each other, the design with the higher success rate is declared the winner.

There are several problems with A/B testing for website design. First, web designers have to create entire new versions of a site, without much insight into which variables to change and which to control. This is expensive and time-consuming. Second, statistical significance in A/B testing requires a large volume of visiting users, and on all but the largest websites this severely limits the speed with which experiments can be run. Third, A/B testing does not leverage the benefits of software development. Unlike most products, software is low cost to change and amenable to continuous improvement. Also, it is easy to observe and respond to user interaction with experimental versions of websites.

So, how can we improve A/B testing for website design? We can make use of a model which is based on what is known in probability theory as the "multi-armed bandit" problem.

1.2 What is a Multi-Armed Bandit Problem?

A "multi-armed bandit" problem is a type of hypothetical experiment where the goal is to find the most profitable action or sequence of actions. It can be imagined as a scenario in which a gambler faces a row of slot machines ("one-armed bandits"), and he must decide how many times to play each, and in what order, so that he maximizes his reward. He suspects that the slot machines have different payouts. Indeed, when played, each machine provides a random reward from its specific reward distribution. He wants to find the machine with the best payout rate, but he also wants to maximize his expected winnings. The gambler's objective is to maximize the sum of rewards earned through a series of lever pulls. He does this by "exploiting" some arms (playing the machines he knows have high rewards) and "exploring" other arms (playing untested or inferior machines in case they might perform even better). This tradeoff is referred to as "earn" vs. "learn."

In our project, we make two high-level changes to the classic multi-armed bandit problem, indicated by the title of this paper. First, we maximize expected reward over a website with multiple pages, so we have a **graph** of multi-armed bandits, one for each page in our website. Second, arms are **real-valued**, not discrete. Rather than choosing between several pre-designed versions of each page, we choose the optimal setting of some real-valued parameters for each page, such as font color and text size.

2 Related Work

There is currently a substantial amount of research in adaptive content, since multi-armed bandit problems were introduced by Robbins in 1952, and many corporations, including Microsoft Research and Google Analytics, use a multi-armed bandit approach to managing online experiments. While at Microsoft Research, Slivkins and Upfal studied a dynamic multi-armed bandit problem in which transitions and rewards gradually change, and they were able to characterize the cost of learning and adapting to changing environments [1]. This is useful because companies must be able to adapt their websites as user preferences and responses change over a longer period of time. Additionally, Mishra, Slivkins, and Syed devised an algorithm to adapt search results to shifts in user intent through the detection of various signals indicating a shift in user intent [5]. At Google, Dr. Scott has shown that using the multi-armed bandit problem for website conversions compared to A/B testing decreases the number of days of testing by on average 175 days [4]. Research on using the multi-armed bandit problem to optimize website features continues as more and more people use the Internet and companies aim to increase their profits.

3 Problem Definition

Our goal is to build a system for modeling and optimizing the customer lifecycle of a digital business. We aim to maximize the value a given customer brings, assuming that:

1. We pay a small amount for each user (i.e., with an ad)
2. The user may or may not purchase content from our site.

We do this by continuously tuning the parameters of each page on our website based on user behavior, as opposed to manually designing new versions of the site.

4 The Model

In this problem, we try to optimize stochastic behavior through some defined state space, where we have control over the parameters at each of the states that affect the stochastic transitions to child states, but we don't know in advance exactly how the parameters are related to the stochastic process. So, we utilize online learning to simultaneously build and optimize a model. It may be helpful to think of this as the "graphical bandit" problem, analogous to the traditional multi-armed bandit. Our algorithm converges on the optimal settings of each page in order to direct users toward high-reward pages/states in the graph.

We model the website as a Markov Decision Process. Each node is a state, representing the page of the website that the user is on. Our MDP is defined as follows:

- States: each page in the website.
- $s_{\text{start}} \in$ States: starting state: the "homepage."
- Actions(s): possible actions from state s, such as "go to product page," "leave website," and "purchase product."
- $T(s, a, s')$: probability that a user will take action a in state s to reach state s'.
- Reward(s, a, s'): reward received (can be positive, negative, or 0) for the user making a transition (s, a, s').
- IsEnd(s): whether or not at an end state: i.e., "left website" or "purchased product."

Figure 1 shows an example website represented as a MDP, with a start state, end states, and directed arcs representing valid transitions. Note that the graph is directed and acyclic.

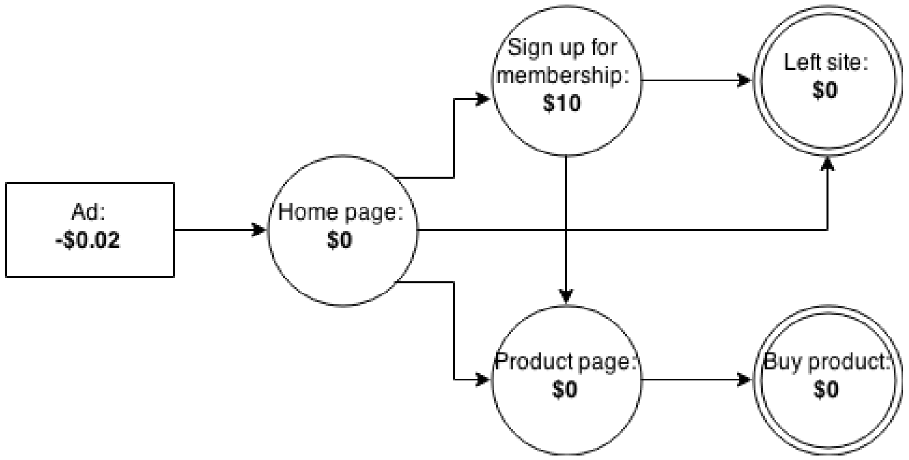

Figure 1

4.1 States

Each state (page) has the following associated with it: a reward, a set of valid transitions to new states, and a feature vector x. The feature vector represents a set of n parameters $(x_1, \ldots, x_n)$ that describe the format and layout of the page, where n and the type of features may be different for each page. A toy example of a feature vector with seven features is:

$$\begin{pmatrix} \text{Font color R} \\ \text{Font color G} \\ \text{Font color B} \\ \text{Font size} \\ \text{Background color R} \\ \text{Background color G} \\ \text{Background color B} \end{pmatrix} \tag{1}$$

Larger features (such as colors) are split up (i.e., into R, G, B values) so that each feature can be represented as a real value. We don't store the names of features explicitly in the feature vector, only the values themselves, and we associate each feature name with an index into the vector. So, our toy feature vector for one page looks like:

$$x = \begin{pmatrix} 50 \\ 0 \\ 255 \\ 20 \\ 255 \\ 255 \\ 255 \end{pmatrix} \tag{2}$$

where values have been assigned to each feature. The goal is to pick the setting of the features for each state that maximize expected value (revenue per customer).

4.2 Transitions

Each transition from state s to state s' has the following associated with it: the next state it points to (s'), and a set of learned support vectors based on user behavior representing a mapping from s's feature vector ($s.x$) to the probability of a given user taking that transition. We generate the transition probability $T(s, a, s')$ by dividing the output of transition a's SVM by the sum of the SVM outputs for all transitions out of state s:

$$\frac{f(a)}{\sum_{t \in \text{Actions}(s)} f(t)} \tag{3}$$

where f is the output of an SVM:

$$f(x) \propto \sum_{i=1}^{m} a_i y_i K(x_i, x) + b \tag{4}$$

We use these support vectors in order to pick the feature vector x for each state that maximizes the expected value of a user traversing our site.

5 The Algorithm

There are two versions of this problem: discrete and continuous. In the discrete case (the classic multi-armed bandit problem), the model chooses between a number of preset versions for each node on the graph, akin to an A/B testing framework choosing between versions of a page. In the continuous case, the model chooses a vector of real-valued parameters for each node on the graph. We implement the latter problem. We use an epsilon-greedy approach to exploration, in order to pick the setting of the features for each state that optimizes revenue per customer.

Algorithm 1 below demonstrates the basic pseudocode of our algorithm:

Algorithm 1 Expectation Reward Optimization

```
1:  graph := new Graph()
2:  procedure ObserveUser(traversal)
3:      for <state, action> in transitions do
4:          Append <state.x, action> to state.trainingSet
5:          state.retrainMulticlassSVM()
6:      $\epsilon := 0.1$                    ▷ Initialize epsilon to prioritize exploration vs. exploitation
7:      for state in graph.reverseTopologicalSort() do
8:          $r :=$ Uniform($[0,1]$)          ▷ A uniformly distributed random variable
9:          if $r < \epsilon$ then           ▷ Exploration phase
10:             state.x := random
11:         else                             ▷ Exploitation phase
12:             state.x = $\arg\max_x \sum_i$ state.getTransProb($i$) $\cdot$ state.child[$i$].expectedReward
13:         state.expectedReward = $\sum_i$ state.getTransProb($i$) $\cdot$ state.child[$i$].expectedReward
```

5.1 Data

We train the algorithm with synthetic data, because it is expensive to collect data from real users. Users are all drawn independently and identically from a probability distribution that represents their preferences. A user has a given "gold" support vector for each outgoing arc on a state. We use these support vectors to map the state's feature vector to its probability distribution over outgoing transitions. We keep these gold support vectors hidden from the system.

To simulate a user clicking through the MDP, we draw a random transition from our hidden model for the given state. Then the pair (current feature vector x, action) becomes a new training example that we add to the training dataset for that state. Each time a "user" takes an action, we go through one cycle of ObserveUser using the larger training set, retraining the multiclass SVM for each state to pick new support vectors, and updating the feature vector for each state to maximize expected value over the entire graph.

The algorithm parses its inputs from a plain text file. It accepts as input:

- S: the list of states in the MDP, with indications as to which one(s) are start and terminal states.
- V: a dictionary that associates a state with its reward.
- T: the possible transitions from each state (with unknown probabilities), represented as an $|n \times n|$ adjacency matrix.

6 Optimization

6.1 Choosing an optimal feature vector in a nonlinear transition model

We choose to model the transition probabilities from a node to a given child by the output of an SVM, normalized across the multiple possible outputs:

$$f(x) \propto \sum_{i=1}^{m} a_i y_i K(x_i, x) + b \tag{5}$$

We have a set of n children, where each child i has expected value E_i. We are interested in our expected value, given a choice of x, representing the feature vector for creating a website:

$$E[x] = \frac{\sum\limits_{i=1}^{n} E_i \Big[\sum\limits_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i \Big]}{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i} \tag{6}$$

We wish to choose an x that maximizes our expected value under the model, given our evidence so far. This is non-convex, so we'd like a solution in closed form. Begin by taking the derivative:

$$\frac{d}{dx} E[x] = \frac{d}{dx} \frac{\sum\limits_{i=1}^{n} E_i \Big[\sum\limits_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i \Big]}{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i} \tag{7}$$

In order to fit the entire derivative on one line, let:

$$e(x) = \sum_{i=1}^{n} E_i \Big[\sum_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i \Big] \tag{8}$$

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_j^{(i)} y_j^{(i)} K(x_j^{(i)}, x) + b_i \tag{9}$$

Then, by the quotient rule, we have:

$$\frac{d}{dx} E[x] = \frac{f(x) \frac{d}{dx} e(x) - e(x) \frac{d}{dx} f(x)}{(f(x))^2} \tag{10}$$

Following convention, let's choose the Gaussian kernel as our kernel:

$$K(y, x) = e^{\frac{\|x-y\|^2}{2\sigma^2}} \tag{11}$$

We'll also need the derivative. We use the fact: $\|x-y\|^2 = \|x\|^2 - 2x^T y + \|y\|^2$

$$\frac{d}{dx} K(y, x) = \frac{d}{dx} e^{\frac{\|x\|^2 - 2x^T y + \|y\|^2}{2\sigma^2}} \tag{12}$$

Then, we apply the chain rule and arrive at:

$$\frac{d}{dx}K(y,x) = \left(e^{\frac{\|x\|^2 - 2x^T y + \|y\|^2}{2\sigma^2}}\right) \cdot \left(\frac{y-x}{\sigma^2}\right) \tag{13}$$

And simplifying:

$$\frac{d}{dx}K(y,x) = \left(e^{\frac{\|x-y\|^2}{2\sigma^2}}\right)\left(\frac{y-x}{\sigma^2}\right) \tag{14}$$

We can substitute in this expression to get our final derivative:

$$\frac{d}{dx}E[x] = \frac{f(x)\frac{d}{dx}e(x) - e(x)\frac{d}{dx}f(x)}{(f(x))^2} \tag{15}$$

where

$$e(x) = \sum_{i=1}^{n} E_i\Big[\sum_{j=1}^{m} a_j^{(i)} y_j^{(i)} e^{\frac{\|x - x_j^{(i)}\|^2}{2\sigma^2}} + b_i\Big] \tag{16}$$

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} a_j^{(i)} y_j^{(i)} e^{\frac{\|x - x_j^{(i)}\|^2}{2\sigma^2}} + b_i \tag{17}$$

$$\frac{d}{dx}e(x) = \sum_{i=1}^{n} E_i\Big[\sum_{j=1}^{m}\Big(a_j^{(i)} y_j^{(i)} e^{\frac{\|x - x_j^{(i)}\|^2}{2\sigma^2}} + b_i\Big)\Big(\frac{x_j^{(i)} - x}{\sigma^2}\Big)\Big] \tag{18}$$

$$\frac{d}{dx}f(x) = \sum_{i=1}^{n}\sum_{j=1}^{m}\Big(a_j^{(i)} y_j^{(i)} e^{\frac{\|x - x_j^{(i)}\|^2}{2\sigma^2}}\Big)\Big(\frac{x_j^{(i)} - x}{\sigma^2}\Big) \tag{19}$$

Our expectation in x is non-concave and has many local optima. In practice, we use gradient descent with several restarts per problem to find the best value we can, with no optimality guarantees. Figure 2 is a graph of a gradient check on a toy dataset, with 6 SVMs resulting in 2 distinct (identical) peaks:

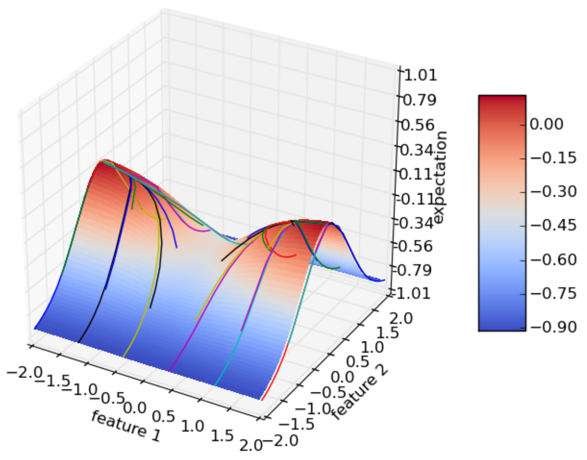

Figure 2

The three-dimensional graphs presented throughout this paper represent a simple optimization problem, with two real valued parameters ("feature 1", and "feature 2"), each constrained to take on values between (-2, 2). The graphs represent an expected value of user behavior at a combination of those features by the z coordinate. This expected value abstracts away the details of the distribution over transitions a user will take at a given vector and the expected value of each of those transitions, in favor of the global expected value of the user transition distribution.

7 Experiments

Next, we run a full experiment: we generate synthetic preferences for a user and have the system try to learn those preferences in order to maximize the expected value of that user traversing our site. To simplify our experiment, our website will have only one page, but the experiment generalizes to any directed acyclic graph. In order to better visualize our results in a three-dimensional plot, we choose to optimize two free variables: feature 1 and feature 2. Figure 3 is a graph of a synthetic user's actual expected value z over the parameterization of these two real-valued features. Given a parameterization, the true expected value drawn from the synthetic, hidden distribution corresponds to the height of the graph at this point. We keep this mapping of (feature 1, feature 2) $\rightarrow$ expectation hidden from the system, since it is the algorithm's job to use gradient descent to maximize the expected value of a given user.

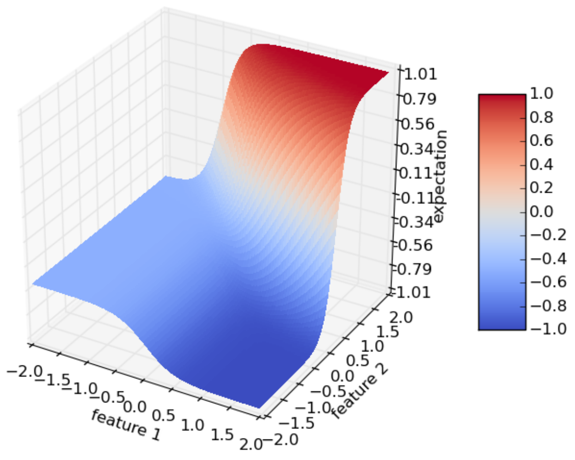

Figure 3

For each outcome (in our experiment there are three), there is an SVM with a single support vector. Near each support vector, the graph takes on the expected value corresponding to the outcome that the support vector predicts (i.e., it has the highest probability of the outcomes). There are support vectors at parameterizations (2.0, 2.0), (-2.0, 2.0), and (-2.0, 2.0) with expected values near 1.0, -1.0, and -0.5, respectively. The graph takes on the expected value predicted by the support vector at each of these parameterizations, because the probability that the user takes the transition resulting in this expected value is very close to 1 at those settings of feature 1 and feature 2.

Figure 4 shows what our system believes about the user's expected value at different values of feature 1 and feature 2 after 30 iterations. This belief is a good approximation of the true distribution. Also note the success of gradient descent in converging to the local optima (as indicated by the lines in the plot).

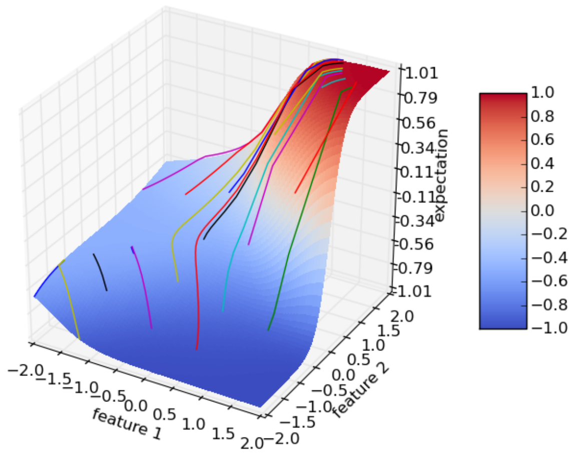

Figure 4

Figure 5 shows the true expected value of the user's traversal through our website, as a function of number of iterations. The dips in the graph are the epsilon-greedy explorations. Note that the system quickly discovers the optimal solution, and sticks with it.

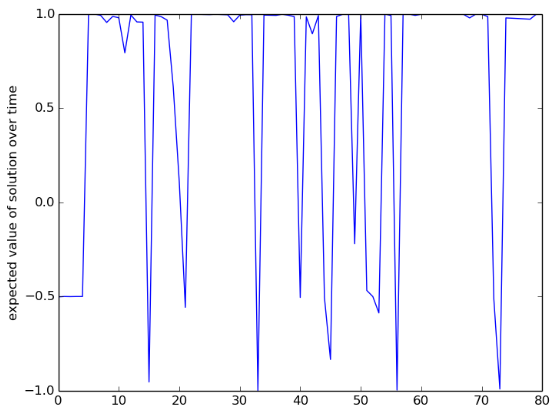

Figure 5

For this small test case (simplified so that we can analyze every aspect of the pipeline), the system works according to the theory laid out in this paper. Our algorithm would most likely work in practice, but its computational performance suffers as it accumulates training examples. Any real deployment would require batching and optimization in order to make updates tractable.

8 Future Work

Our simple solution to the problem of optimizing continuous parameters over an MDP suggests many obvious improvements.

In theory, SVMs with a Gaussian kernel can approximate any function, but it would be worthwhile to apply this to a real website, in order to judge whether or not it is possible for SVMs to converge to a reasonable approximation of the relationship between parameters and user behavior in a reasonable number of trials. It would be useful to test different exploration strategies in the presence of real users, on the hypothesis that epsilon-greedy is not the optimal solution (although it is the simplest).

It may also be possible to automatically segment the population into clusters that behave similarly, and learn the optimal parameter settings for each group. For example, the elderly may have a higher expected value with a website using a larger size of font. In order for clustering work optimally, the behavior of an arriving user would have to determine within a transition or two which cluster they belong to, in order for us to optimize future pages served to this user during their stay. This may mean that the landing page would have to be specially optimized to provide maximum information value, as a means to maximum expected value. That dramatically complicates the mathematics, so we leave it to future work.

Another area for improvement is loosening restrictions on graphical structure. Our requirement that we only consider DAGs is unrealistic. In the real world, users cycle back and forth between pages, and an adaptation of EM that converges to ideal parameter settings in a loopy MDP setting would be useful. The theoretical proof involved in order to guarantee convergence in the non-linear setting is non-trivial (and may be impossible, due to lack of convexity).

References

[1] A. Slivkins and E. Upfal. “Adapting to a Changing Environment: the Brownian Restless Bandits,” in *Proceedings of the 21st Annual Conference on Learning Theory*, Helsinki, Finland, 2008, pp. 343-354.

[2] S. Hanov. *20 lines of code that will beat A/B testing every time* [Online]. Available: http://sevehanov.ca/blog/index.php?id=132

[3] S. L. Scott. “A modern Bayesian look at the multi-armed bandit.” *Applied Stochastic Models in Business and Industry*, vol. 26 (2010), pp. 639-658.

[4] S. L. Scott. (2014). *Overview of Content Experiments: Multi-armed bandit experiments* [Online]. Available: http://support.google.com/analytics/answer/2844870?hl=en

[5] U. Syed *et. al.* “Adapting to the Shifting Intent of Search Queries,” in *Advances in Neural Information Processing Systems*, Vancouver, B.C., 2009, pp. 1829-1837.