Evalution of Multi-Armed Bandit Algorithms in a Web Recommendation Context

James Wang
Haas School of Business, UC Berkeley

December 5, 2014

Abstract

Evaluating multi-armed bandits empirically in real-world applications is difficult due to the off-policy problem (also known as partial label) where we cannot observe the feedback we'd get from unchosen paths in sequential decision problems. Li et al 2011 formulated a way of evaluating MABs in an unbiased manner and recently Yahoo! released the dataset used for that paper. In this work, I analyze the characteristics of that dataset and compare a number of popular MAB algorithms on the data. Notably, the data does not have consistent arms over time, so I also briefly describe a method for very simply adapting the evaluated bandit algorithms to this context.

Introduction

Multi-armed bandits (MABs) are an important, broadly applicable problem formulation in statistical learning. While their simplicity make them ideal for theoretical analysis, their structure also allow them to closely match the exploration-exploitation tradeoff faced in many real-world applications [5]. Operations research, medical treatment, and web content optimization are just a few of the domains where MABs have been applied to problems that exhibit this type of tradeoff [1, 10, 14].

However, empirical results for MABs have generally been limited to simulation data, toy datasets, or proprietary datasets that cannot be verified or replicated. The first two can potentially either introduce simulation bias (leading to an overly favorable underlying distribution, which trivially allows certain algorithms to perform well) or present scenarios that do not well represent those faced by real-world applications of MABs. It is common to see papers on MABs that utilize these types of experiments: numerical simulations generated from a set of known distributions, an adapted dataset that does not naturally exhibit a MAB structure (leading to a highly idealized scenario or even a form of bias introduced by the authors tailoring the scenario), and/or a dataset (usually in a web context) where results cannot be replicated as it is a proprietary/closed dataset [3, 11].

One of the main reasons for why these imperfect empirical evaluation techniques are used (specifically, when real-world performance is being assessed) is the difficulty of finding good benchmarks for evaluating bandit algorithms. While there are many good choices for general supervised learning [4, 15], most are not suitable for bandit algorithms. The main problem with most datasets is the "off-policy" problem (also known as "partial label") where the dataset does not observe the outcome of the action that the algorithm evaluated would have chosen [13, 16]. Unfortunately, this problem is most acute in cases where the dataset/application area naturally exhibits the characteristics of a bandit problem (sequential decisions with exploration-exploitation tradeoffs). In these cases, by definition, we only have partial information and usually no feedback from the choices that were not made by whatever policy generated the choices observed in data. This forces us to rely on expensive and often impractical evaluation using production systems for reliable real-world empirical tests of bandit algorithms (e.g. an online system for web recommendation on a website) [14].

A simple example that helps illustrate this problem is a website "A/B" test where one is trying to optimize some metric on the site with two different versions. We want to maximize conversions for a call-to-action (perhaps subscribing to a newsletter or clicks on an ad) on the website and want to both make sure we are showing the right version and we are maximizing our conversions. In an online, production context, we can

apply a policy that attempts to do this and evaluate its performance metric. If we try to use the dataset offline for other algorithms, we will find that the production algorithm may have chosen to show A, but our alternate algorithm in that context would have shown B. Unfortunately, we do not have the results for what would have happened with the counterfactual of showing B in this case, and hence we would not be able to properly evaluate our new algorithm on this dataset. We don't know what would have happened with the "on-policy" decision.

Li et al 2011 introduces an approach to solve this problem through an unbiased estimator [13]. The broad concept is similar to the process of rejection sampling where we sample from a superset of our desired set, and discard samples that do not fall within the desired set. The dataset described and used in Li et al 2011 was recently released by Yahoo!, and obtained for the purposes of this paper [18].

My aim in this paper is to evaluate the characteristics of dataset in question (and the unbiased estimator as applied to it), compare more popular algorithms that were not evaluated in the Li et al 2011 paper, and describe (very briefly) a simple procedure for modifying existing algorithms to work with a shifting number and set of arms. Finally, I hope to release the code used for processing the dataset (here, I took the raw data and processed it into an on-disk database), running the evaluations (with the unbiased estimator), and running the algorithms (with modifications for a shifting set of arms).

Formal MAB Problem Setup

For algorithm/policy P, each trial t to T trials:

1. Algorithm observes user u_t and a set of A_t arms (articles) together with context vector $x_{t,a}$ for $a \in A_t$. $x_{t,a}$ contains features and interactions from both user u_t and arm a_t. Specifically $x_{t,a} = \mathbf{ua}^T$ where $\mathbf{u}$ is the feature vector for user u_t and $\mathbf{a}$ is the feature vector for arm a_t
2. Based on A's observations of rewards per arm for previous trials, P chooses arm from A_t and recieves payoff $r_{t,a}$ (in our context, $r_{t,a}$ is binary and is either 0 or 1 depending on no click or click, respectively).
3. Algorithm updates strategy based on new observation. No feedback is received for unchosen arms.

Description / Characteristics of the Data

This data is drawn from the Today module on the Yahoo! homepage [18]. We see 45,811,883 distinct visit events where a user is shown an article in the feature box (drawn from a pool of articles picked by human editors) and the click (or lack of click) is recorded. Critically, as we will discuss later, this dataset displays articles from the aforementioned pool randomly.

In relation to the MAB problem, for each event t, I consider each article as an arm (a_t), each pool of potential articles that can be shown as the set of A_t arms for the round, and each user as distinct user u_t. Click events provide binary rewards (0 or 1 for no click or click, respectively) and represented by $r_{t,a}$ for each article in each round. The context vector $x_{t,a}$ and its associated features are described more below.

Users

Each user within the dataset is distinct, which mitigates a potential problem of dependent observations (users seeing an article multiple times, and having their CTR affected in one direction or another). With over 45 million observed events and a guarantee of distinctness in users, we have over 45 million individual users. Each users is analyzed using conjoint analysis and clustered using k-means into 5 clusters. The specific details are outlined in Chu et al 2009 [9].

Looking at the clustering behavior of user features, each datapoint is generally fairly close to the center of the K-means cluster. One way of roughly seeing this is to look at the relative dominance of features for each user.

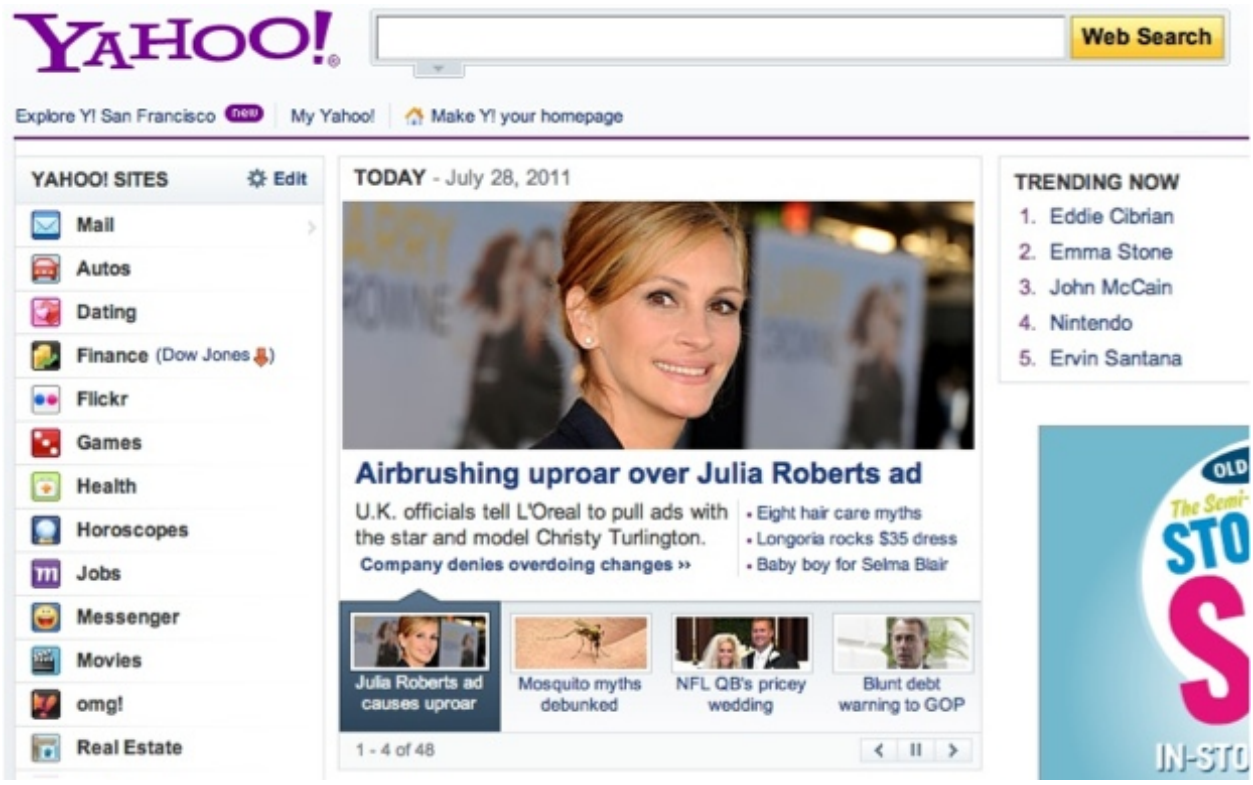

Figure 1: Yahoo! homepage and Today module

All features (besides one constant feature) must sum to one, and if we look at Figure 2, we see that most users have one predominant feature that corresponds with their cluster membership. Practically speaking, we would expect this would improve the performance of indexed policies (e.g. indexed UCB), since the user feature vectors act similarly to discrete segments.

Given this characteristic, for this paper I average across the feature vectors for each cluster to get the user feature vectors, both to make each user more interchangeable (which they largely are anyway) and also save on computation, as this makes it possible to cache feature vectors between user and article pairs (all users otherwise have slightly different conjoint features). As Figure 2 suggests, this doesn't make much of an impact due to how close the features are to their cluster centers for individual users and the similarity between users in the same cluster.

Articles

There are 271 articles observed within the course of the 10 days covered by the dataset. A random article is drawn from a pool of around 20 available articles for each event (varying from 19 to 25, with new articles introduced and old articles retired throughout the period observed. We ultimately observe 433 distinct sets of articles, A_t, shown to users throughout the 10 days.

Each article has features determined by the same conjoint analysis process, which is described in more detail in Chu et al 2009 [9]. Together, the user and article feature vectors give us a $\mathbb{R}^{36}$ vector for each user/article pair, $x_{t,a}$–this is obtained from the outer product of the user and article $\mathbb{R}^6$ feature vectors, which are then flattened into $\mathbb{R}^{36}$

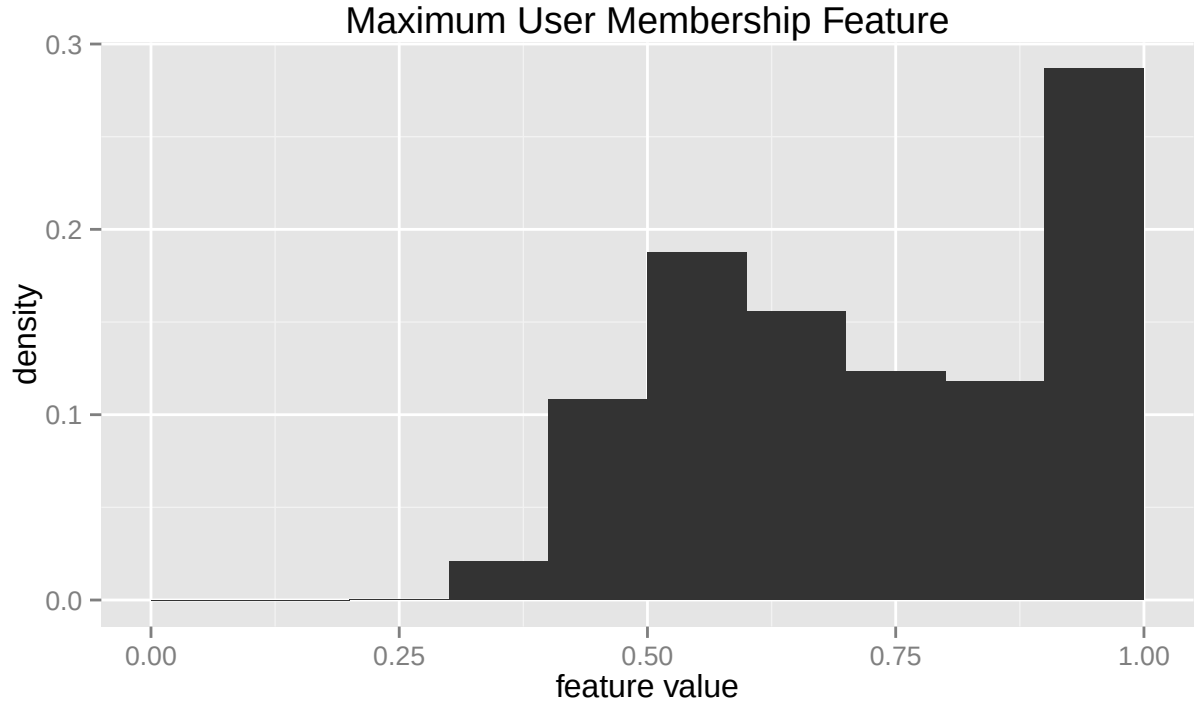

Figure 2: Maximum features for each user vector

Clickthrough Rates

Each user cluster is fairly distinct in terms of which arms are pulled. Note that from an application domain perspective, while the CTRs look fairly close (see Figure 3), the differences are significant in a web context given generally low CTRs and with small changes in CTRs leading to large business impacts. That being said, the nature of CTRs – which can be thought of as a stream of highly unbalanced binary rewards – mean that we need a huge number of observations in order to start distinguishing between arms.

From Figure 4, we can see that each cluster in Figure 3 does start picking out some of the best articles in CTRs generally, despite still being distinct in the overall set of top 5 most preferred articles.

One challenge that we will face in this dataset is that CTRs will not be stable over time (a simple example is off-hours, during the middle of the night in certain highly represented time zones). However, this type of impact affects all articles/arms similarly. More challenging from the perspective of existing algorithms is handling a shifting number of arms. There is a fairly simple solution for the algorithms I examine, which I will describe in later sections.

CTR Unbiased Estimation

Overall, this dataset still does not solve the "off-policy" problem, where the chosen arm by the policy does not correspond with the arm observed by the dataset. We cannot observe the counterfactual, but we can get around the off-policy problem by estimating the policy's performance on the dataset in an unbiased manner. This is covered much more extensively in Li et al 2011, but the essential idea takes advantage of the fact that the dataset's policies are randomly assigned [13]. This means that there is no inherent bias in the policy choices of the data itself. Taking advantage of this property, we can simply discard observations where the

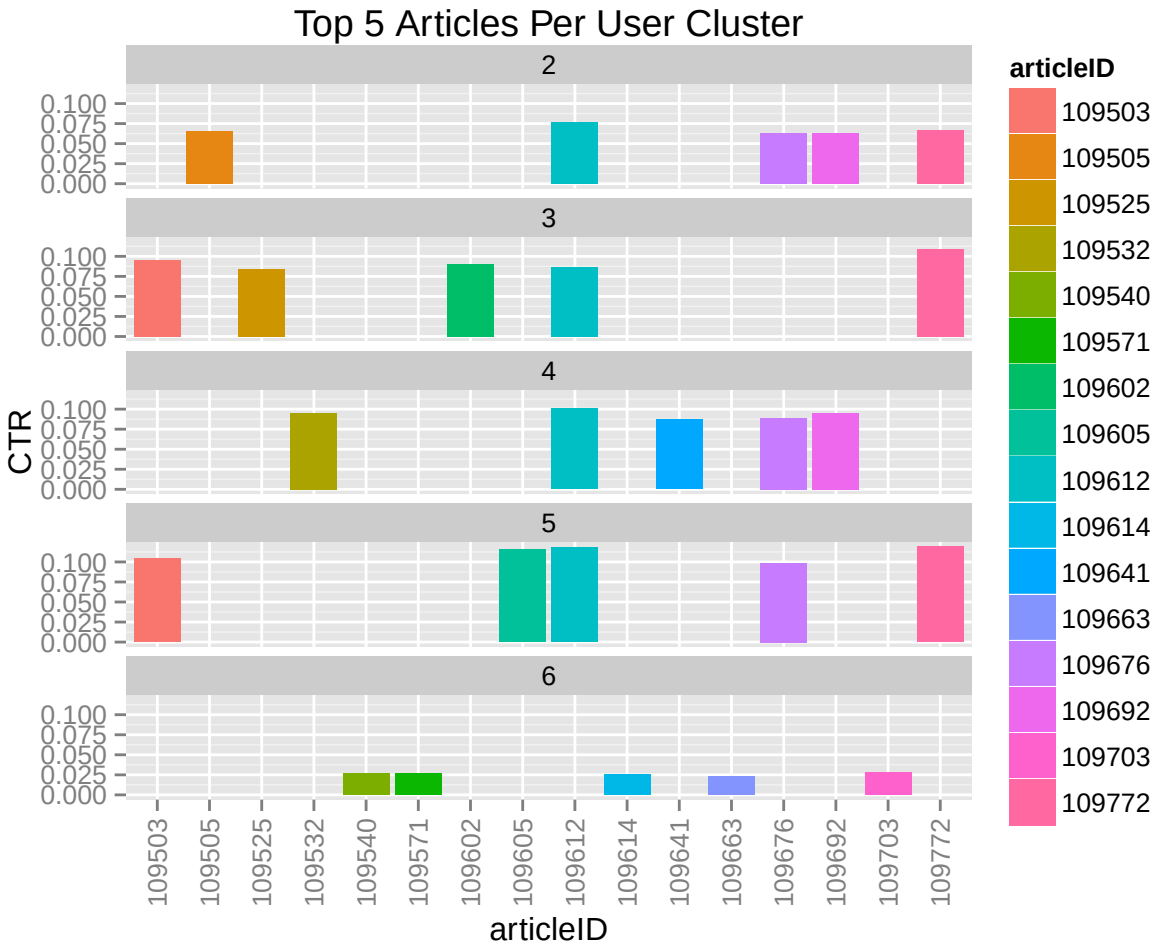

Figure 3: Top arms and CTRs for each user cluster.

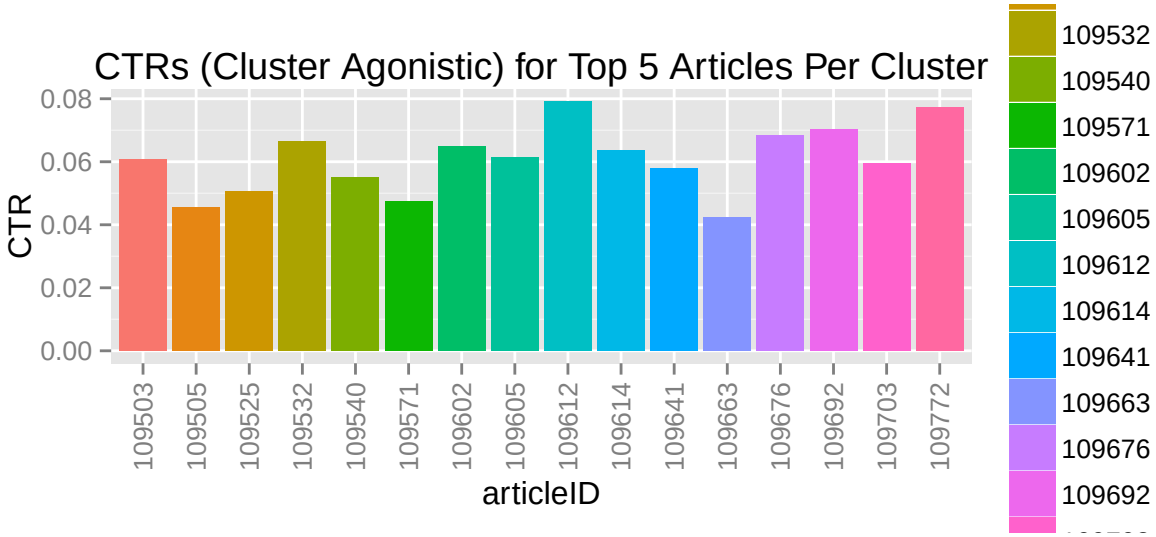

Figure 4: Top 5 arms from each cluster and CTRs, without regard to cluster

policy's desired arm does not match the observed arm. The notion here is similar to the idea of rejection sampling.

Intuitively, we sample uniformally from a context-action set that is a superset our desired context-action set, and reject samples that fall outside of the space. Normally, rejection sampling's difficulty in finding a good way to evaluate membership in the desired space. In our case, this is simple: we observe whether or not the arm that the algorithm would pull given the context (time, user group, and/or feature vector depending on the algorithm) matches the actual arm observed.

This process, however, means that we may have to discard a fairly large number of articles to get our desired T observations, relating to T and the number of arms. With a large T and a fairly large arm space of 271 articles, we would expect an order of magnitude more rejected than accepted observations: specifically, with only $\frac{1}{|A_t|}$ probability of the data exhibiting the right arm, where $|A_t|$ is the cardinality of the set of arms available each round t.

`Policy Evaluator` (from Li et al., 2011)

---

0. Inputs $T > 0$; bandit algorithm P; stream of events S
1. $h_0 \leftarrow \emptyset$ {empty history}
2. $\hat{G_P} \leftarrow 0$ {zero payoff to start}
3. **for** $t = 1, 2, 3, ..., T$**do**
4. **repeat**
5. Get next event $(\mathbf{x}, a, r_a)$ from S
6. **until** $P(h_{t-1}, \mathbf{x}) = a$
7. $h_t \leftarrow \texttt{CONCATENATE}(h_{t-1}, (\mathbf{x}, a, r_a))$
8. $\hat{G_P} \leftarrow \hat{G_P} + r_a$
9. **end for**
10. Output: $\hat{G_P}/T$

Limitations

The main limitation of this evaluator is that it is not unbiased in cases where we have a finite data stream [13]. All real world data sets, including this one, are finite. Additionally, it takes a significant number of total observed events to reach our desired T.

Empirically, it took over 220,000 data points to get 10,000 useable observations, over 6.7 million data points to get 300,000 useable observations and nearly 25 million data points to get 1 million useable observations. Fortunately, for values of T in this range,the data set is effectively infinite with 46 million events. However, this does limit the size of T that we can ultimately test. Here, for both this limitation and reasons of time, I do not exceed $T = 1$ million.

Brief Review of Algorithms Compared

Below, I briefly describe each algorithm I evaluate. These are fairly well-known algorithms that are extensively described in literature (and our seminar), so I won't outline them in detail and leave that for their respective referenced papers.

UCB, KL-UCB, and Thompson Sampling are adapted from Cappé and Garivier's pyBandits code distribution on mloss [6], with adaptations most significantly to allow code to run with a constantly shifting pool of available arms, A_t, and to allow the code to run with my testing code. The adaptations for these algorithms was fairly simple. As UCB and KL-UCB are index policies, I simply restricted the index set from all arms to A_t. Unseen new arms are always pulled (at random if there are multiple previously unseen arms), and arms that "drop out" for an extended period of time will naturally grow in attractiveness with t, as they

will not have been pulled and UCB's "exploration bonus" would inflate and make them more attractive. For Thompson Sampling, I take a similar conceptual approach of restricting the argmax comparison for most attractive arms to only those within A_t: choose only among arms within A_t to determine which action/arm a maximizes our sampled parameter from the posterior distribution (here, for non-contextual Thompson, the Beta distribution).

Non-Contextual

- **Random** - this algorithm simply picks an arm uniformally from A_t.
- **UCB** - UCB was described originally in Lai & Robbins 1985. This implementation is as a special case of KL-UCB with Gaussian divergence as per Cappé et al 2013 and with an α constant of 1/2 instead of 2 as per Auer et al 2002 [7, 12].
- **KL-UCB** - As per Cappé et al 2013 [7].
- **Thompson Sampling** - As originally described in Thompson 1933 [17].

Indexed

- **Indexed UCB** - This algorithm uses $\mathcal{X}$ independent UCB instances, where $\mathcal{X}$ is the number of distinct contexts/clusters (in the case of this paper, 5).

Contextual

- **Thompson Sampling** As described in Agrawal & Goyal 2014, including with Gaussian prior [2]. Modified to work in the same way as non-contextual Thompson Sampling described above. I describe this version as "Contextual Thompson."
- **LinUCB** As per Li, Chu et al 2010 [14]. Uses *only* the context vector $x_{a,t}$, so needs no modification to operate with a shifting pool of arms – it is already designed to work in the more general linear bandit context. We will see the impact of this in the results below.

Results and Discussion

Non-Contextual

The results for the non-contextual bandit algorithms evaluated here match those found more broadly in literature (see Figure 5). KL-UCB and Thompson Sampling do the best out of this group, matching simulation results found in Cappé et al 2013 and Chapelle & Li 2011, where these algorithms outperform different variants of the UCB algorithm [7, 8]. UCB in turn has been shown to outperform ϵ-greedy [3].

Interestingly, ϵ-greedy performed nearly as poorly as an entirely random strategy. However, this is just one run. My results do not capture error bars well, and past work has shown that ϵ-greedy is highly variable in its results [11, 14]. While ϵ-greedy has mostly done poorly in my own experiments, Figure 8 illustrates what may be happening. With so many (and continually shifting) arms, ϵ-greedy can get "stuck" exploiting "best" arms that are, in actuality, suboptimal (its peak arms are relatively low in rank). Exploration takes a long time to correct this problem after early successes and a large arm set to explore (but too high an ϵ would cause ϵ-greedy to begin approaching a purely random policy anyway).

Indexed UCB and Contextual Thompson

On the other hand, indexed UCB and contextual Thompson do far worse than I would expect. While both do better than random, as we can see in Figure 6, they are both worse than any non-contextual bandit (see

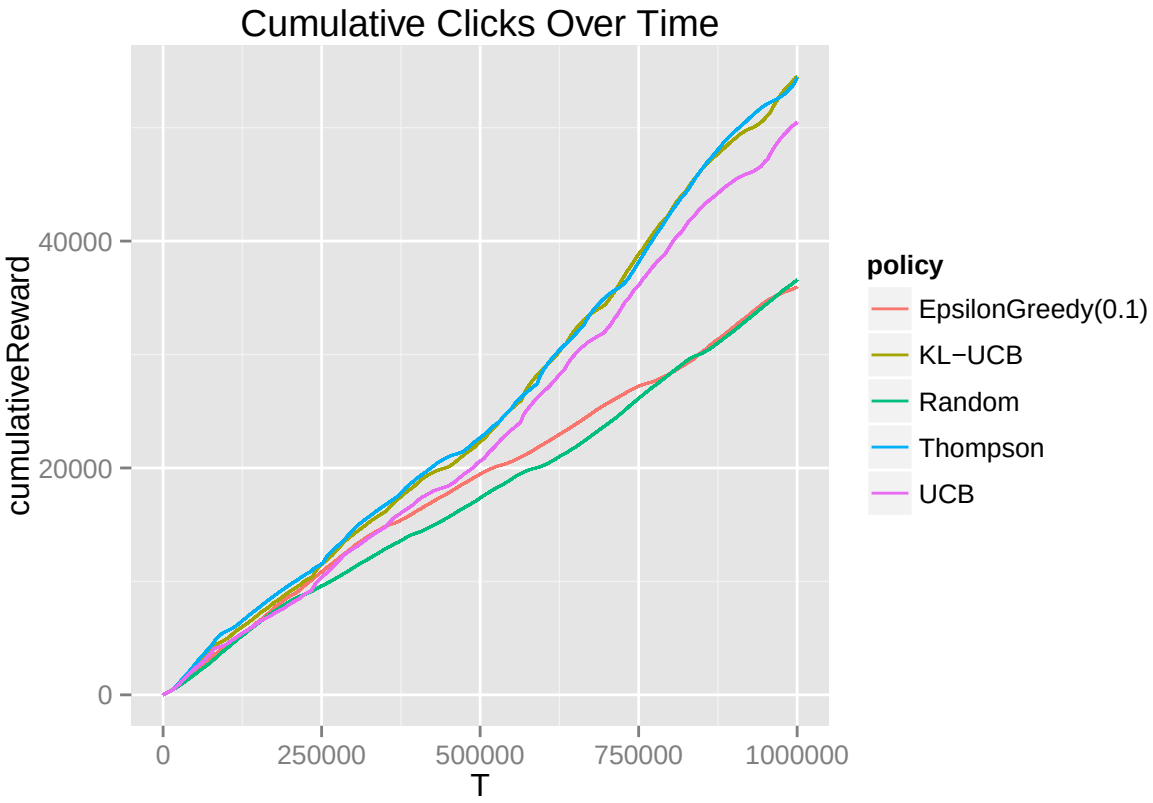

Figure 5: Results for non-contextual bandit algorithms

the table of CTRs for all algorithms in the Comparison of Clickthrough Rates section), suggesting that either the context is not informative (which isn't true as we saw previously, and will see with LinUCB) or whatever "overhead" is incurred from including context is impacting performance.

I believe the "overhead" is at play for indexed UCB. For Indexed UCB, the constantly shifting pool of arms imposes a heavy penalty, as it must still explore suboptimal arms for each individual UCB instance every time a new arm appears. As we can see in Figure 9, we have a significant number of pulls from lowly ranked arms for every context.

On the other hand, Figure 9 also shows extremely poor learning on the part of the contextual Thompson, in stark contrast with non-contextual Thompson on Figure 8. The reason this is likely the case is because the prior used in Contextual Thompson, following from Agrawal & Goyal 2014, is Gaussian and does not fit this problem context as well [2].

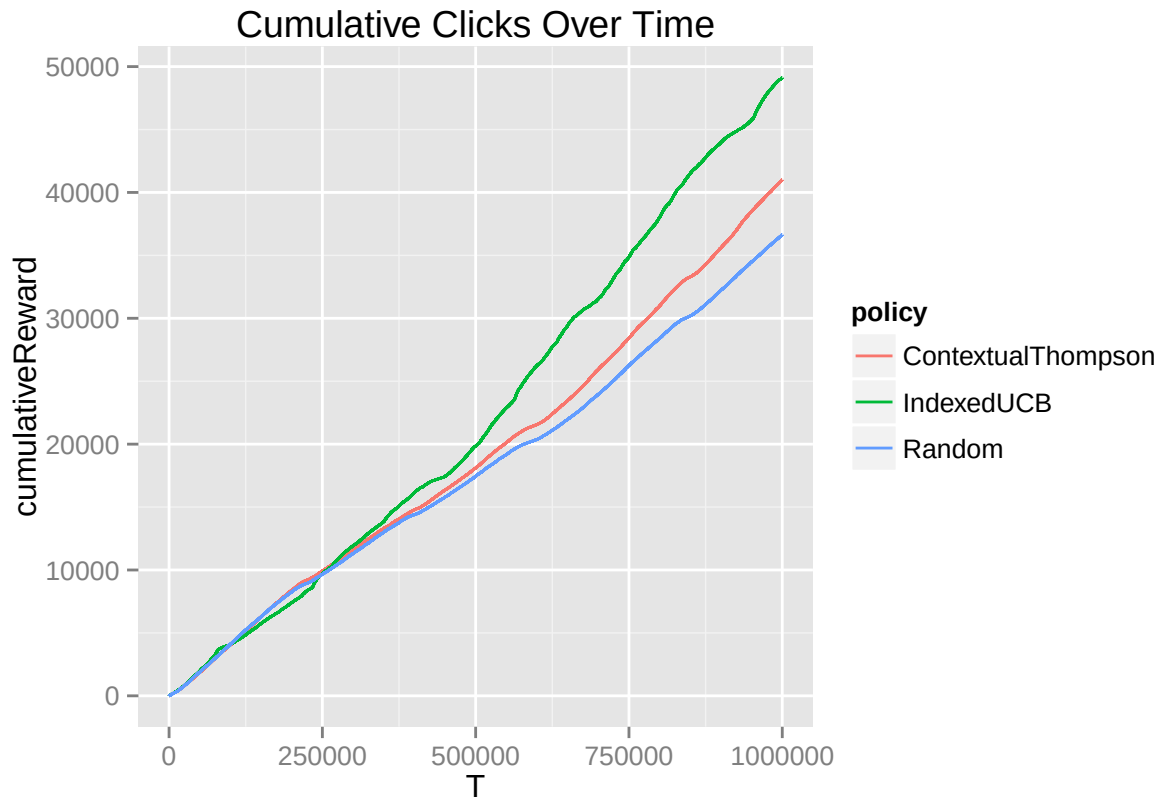

Figure 6: Results for indexed UCB and contextual thompson

LinUCB

Finally, we can see in Figure 7 that LinUCB very quickly does better than all other bandit algorithms compared. This is fortunate, since due to greater computational demands from this algorithm, I was not able to run LinUCB for 1,000,000 T like the other algorithms within the time constraints of this paper. However, we can conceptually understand why this is the case and see the reason in Figure 10. LinUCB shares information between arms and contexts and quickly zeroes in on a good arm for every context, ignoring most subpar arms entirely. But we also see some weaknesses in this approach in Figure 10. LinUCB doesn't actually pull the best arm within its available set for each context. It settles on arms quickly due to its structure of using a given context vector to learn a set of unobserved parameters that linearly generate

rewards for each arm. There is no need to explore obviously subpar arms, except in this case some of the "subpar arms" within the model were actually superior. If this were a theoretical analysis, LinUCB would be penalized a fair amount for constantly incurring regret. Practically speaking, LinUCB looks like for this type of application domain to be "good enough" – learning one of the the better choices very quickly. However, I also would expect that LinUCB's exploration bonus structure (similar to classical UCB) would cause it to eventually start learning the context-reward structure more precisely with a higher T (i.e. similar to what I was able to run for the other algorithms).

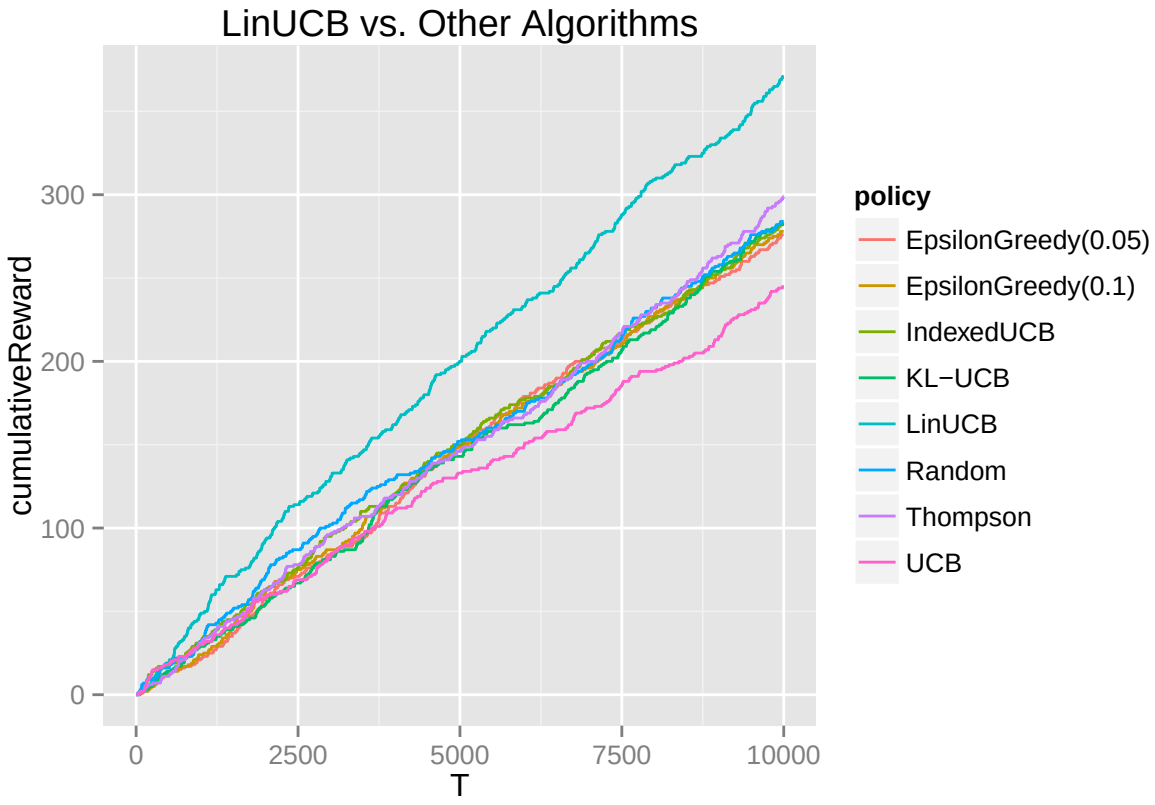

Figure 7: Results for LinUCB

Comparison of Clickthrough Rates

Note that if we ran this experiment to 1,000,000 T for LinUCB like the other algorithms, it would most likely outperform. Instead, its learning was restricted at 10,000 T. In future work, I would want to confirm this conjecture since it is also possible that constantly missing the best arms like described above would cause it to underperform some of these other algorithms.

```
##                Policy      CTR
## 1:             KL-UCB 0.054508
## 2:           Thompson 0.054392
## 3:                UCB 0.050476
## 4:         IndexedUCB 0.049134
## 5: ContextualThompson 0.041023
## 6:             Random 0.036651
## 7: EpsilonGreedy(0.1) 0.035946
```

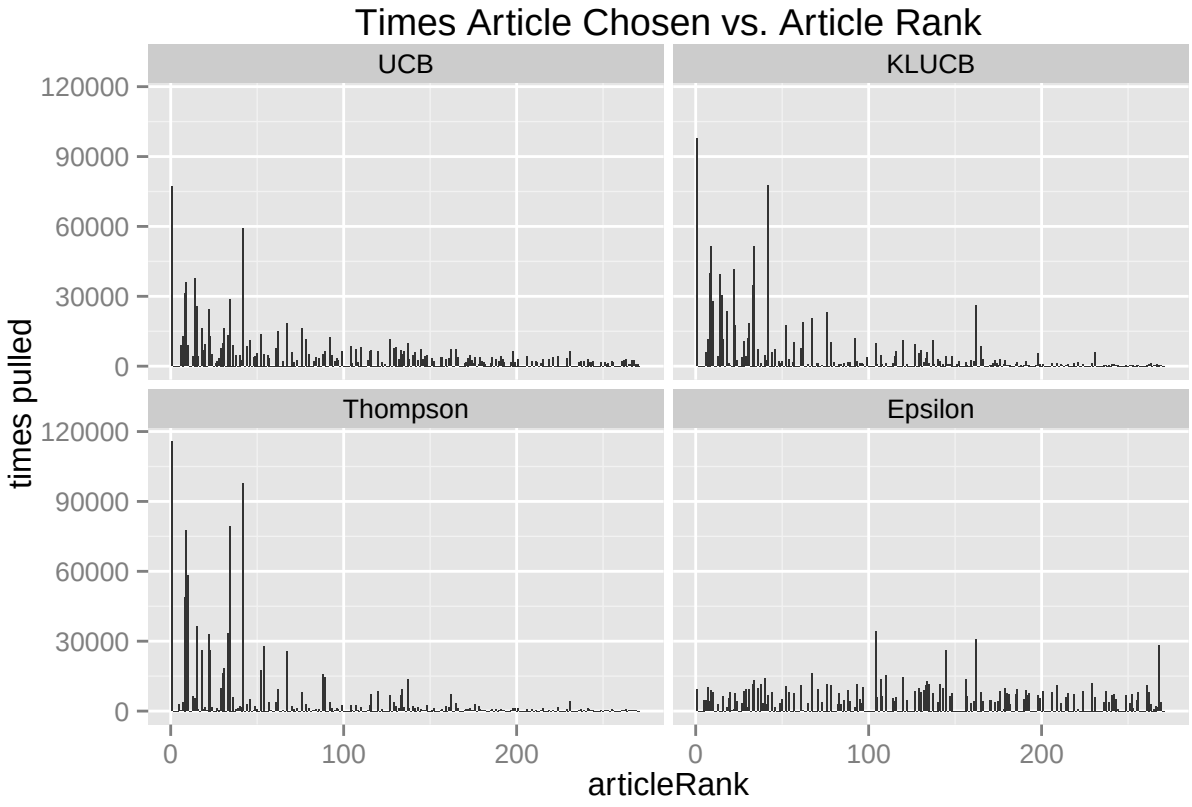

Figure 8: Arm pulls against article rank for non-contextual bandits

Future Work and Conclusions

Like I mentioned previously for ϵ-greedy in results, this analysis does not take into account the range of outcomes we may see. Future work should put "error bars" into place to get a sense of how much variation we may see in the results. From testing the algorithms, I got a sense that most results were fairly consistent except ϵ-greedy, but it would be good to have a more rigorous analysis done.

Analytically speaking, in addition to quantifying the variation we may see in the results, I would want to run LinUCB for $T = 1{,}000{,}000$ to see whether or not it behaves as I conjectured it might in the Results section. Finally, I would also want to clean up my loading, evaluation, and algorithm code and have it available to do these benchmark tests more easily.

Most results I found in this paper were as I would expect based on prior numerical simulations and work (besides indexed UCB and contextual Thompson for the reasons I described previously). However, it was helpful to verify these results on the Yahoo! dataset as a context more representative of what real-world applications may face. Ultimately, I would hope that continued work in this area, the availability of this dataset, more awareness of Li et al 2011's work over time will allow us to see more datasets like the Yahoo! Today Module data, especially in other application domains. Over time, I would hope that more standardized benchmark datasets, like we currently have in supervised learning applications, start to emerge for bandit problems.

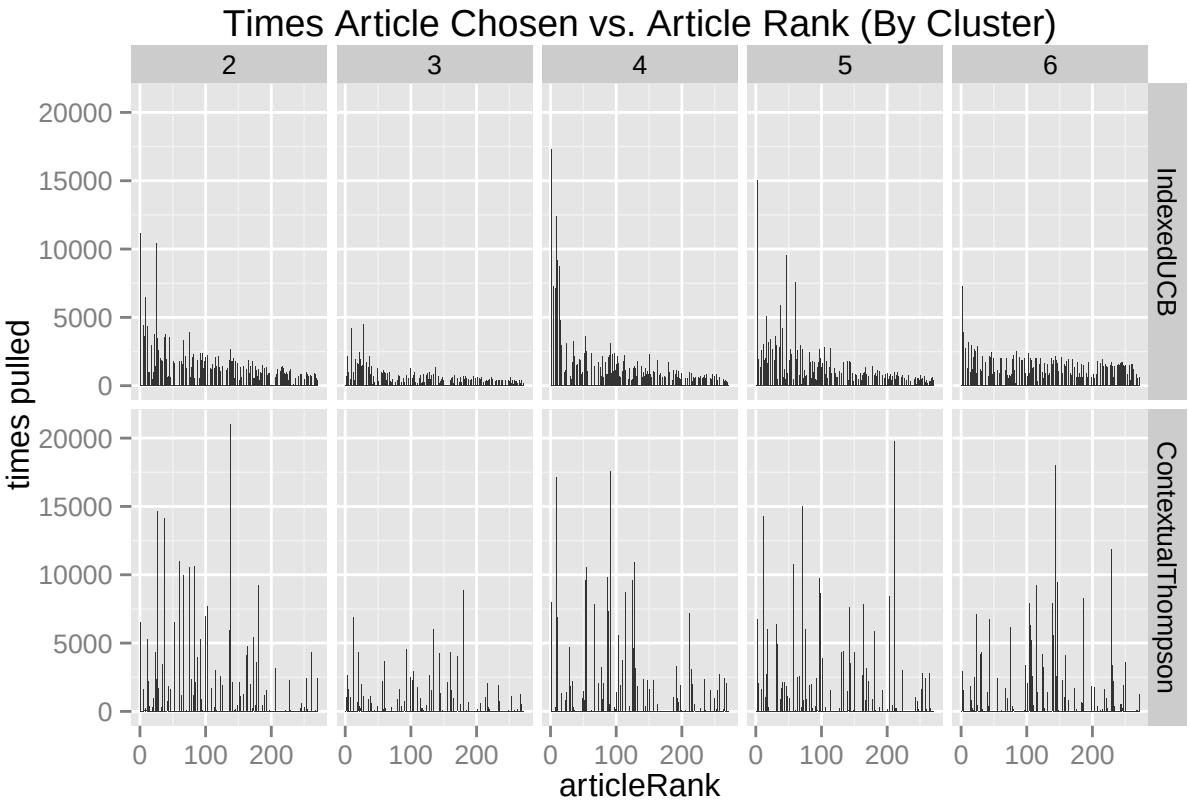

Figure 9: Arm pulls against article rank for contextual bandits

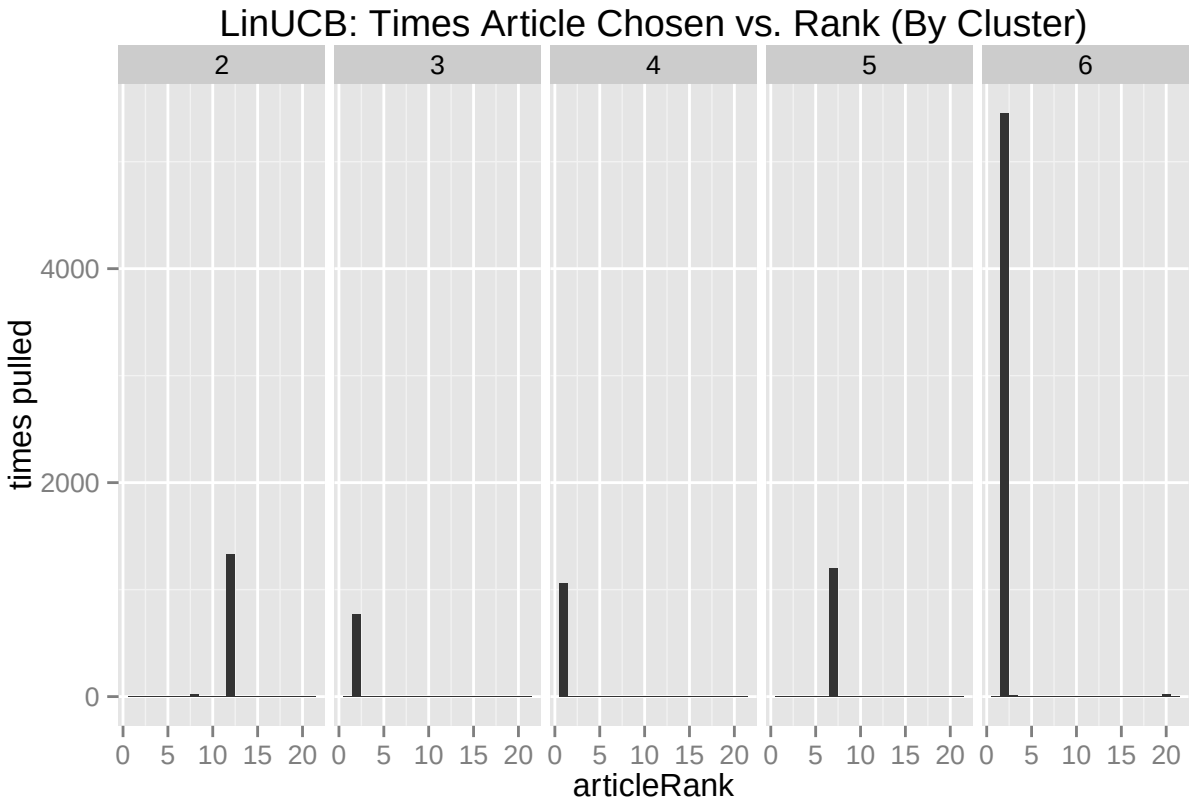

Figure 10: Arm pulls against article rank for LinUCB

References

[1]Agarwal, D., Chen, B.-C. and Elango, P. 2009. Explore/exploit schemes for web content optimization. *Proceedings of the 2009 ninth iEEE international conference on data mining* (Washington, DC, USA, 2009), 1–10.

[2]Agrawal, S. and Goyal, N. 2012. Thompson sampling for contextual bandits with linear payoffs. *CoRR*. abs/1209.3352, (2012).

[3]Auer, P., Cesa-Bianchi, N. and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* 47, 2-3 (May 2002), 235–256.

[4]Bache, K. and Lichman, M. 2013. UCI machine learning repository. University of California, Irvine, School of Information; Computer Sciences.

[5]Berry, D.A. and Fristedt, Bert 1985. *Bandit problems : sequential allocation of experiments / donald a. berry, bert fristedt*. Chapman; Hall London ; New York.

[6]Cappe, O., Garivier, A. and Kaufmann, E. 2012. pymaBandits.

[7]Cappé, O., Garivier, A., Maillard, O.-A., Munos, R. and Stoltz, G. 2013. Kullback–leibler upper confidence bounds for optimal sequential allocation. *The Annals of Statistics*. 41, 3 (Jun. 2013), 1516–1541.

[8]Chapelle, O. and Li, L. 2011. An empirical evaluation of thompson sampling.

[9]Chu, W., Park, S.-t., Beaupre, T., Motgi, N., Phadke, A., Chakraborty, S. and Zachariah, J. 2009. A case study of behavior-driven conjoint analysis on yahoo! front page today module. *In proc. of kDD* (2009).

[10]Farias, V.F. and Madan, R. 2011. The irrevocable multiarmed bandit problem. *Operations Research*. 59, 2 (2011), 383–399.

[11]Filippi, S., Cappe, O., Garivier, A. and Szepesvári, C. 2010. Parametric bandits: The generalized linear case. *Advances in neural information processing systems 23*. J. Lafferty, C. Williams, J. Shawe-taylor, R. Zemel, and A. Culotta, eds. 586–594.

[12]Lai, T.L. and Robbins, H. 1985. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*. 6, 1 (1985), 4–22.

[13]Li, L., Chu, W. and Langford, J. 2010. An unbiased, data-driven, offline evaluation method of contextual bandit algorithms. *CoRR*. abs/1003.5956, (2010).

[14]Li, L., Chu, W., Langford, J. and Schapire, R.E. 2010. A contextual-bandit approach to personalized news article recommendation. *CoRR*. abs/1003.0146, (2010).

[15]M. Braun, C.O., S. Sonnerburg 2014. Machine learning open source software. ML Group of the TU Berlin.

[16]Precup, D., Sutton, R.S. and Singh, S.P. 2000. Eligibility traces for off-policy policy evaluation. *Proceedings of the seventeenth international conference on machine learning (iCML 2000)* (San Francisco, CA, USA, 2000), 759–766.

[17]Thompson, W.R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *j-biometrika*. 25, 3/4 (Dec. 1933), 285–294.

[18]Yahoo! Yahoo! Webscope dataset ydata-frontpage-todaymodule-clicks-v1$_0$.