Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits

Alekh Agarwal ALEKHA@MICROSOFT.COM
Microsoft Research, New York, NY

Daniel Hsu DJHSU@CS.COLUMBIA.EDU
Columbia University, New York, NY

Satyen Kale SATYEN@SATYENKALE.COM
Yahoo! Labs, New York, NY

John Langford JCL@MICROSOFT.COM
Microsoft Research, New York, NY

Lihong Li LIHONGLI@MICROSOFT.COM
Microsoft Research, Redmond, WA

Robert E. Schapire SCHAPIRE@CS.PRINCETON.EDU
Princeton University, Princeton, NJ

Abstract

We present a new algorithm for the contextual bandit learning problem, where the learner repeatedly takes one of K *actions* in response to the observed *context*, and observes the *reward* only for that action. Our method assumes access to an oracle for solving fully supervised cost-sensitive classification problems and achieves the statistically optimal regret guarantee with only $\tilde{O}(\sqrt{KT})$ oracle calls across all T rounds. By doing so, we obtain the most practical contextual bandit learning algorithm amongst approaches that work for general policy classes. We conduct a proof-of-concept experiment which demonstrates the excellent computational and statistical performance of (an online variant of) our algorithm relative to several strong baselines.

1. Introduction

In the contextual bandit problem, an agent collects rewards for actions taken over a sequence of rounds; in each round, the agent chooses an action to take on the basis of (i) *context* (or features) for the current round, as well as (ii) *feedback*, in the form of rewards, obtained in previous rounds. The feedback is *incomplete*: in any given round, the agent observes the reward only for the chosen action; the agent does not observe the reward for other actions. Contextual bandit problems are found in many important applications such as online recommendation and clinical trials, and represent a natural half-way point between supervised learning and reinforcement learning. The use of features to encode context is inherited from supervised machine learning, while *exploration* is necessary for good performance as in reinforcement learning.

The choice of exploration distribution on actions is important. The strongest known results (Auer et al., 2002; McMahan & Streeter, 2009; Beygelzimer et al., 2011) provide algorithms that carefully control the exploration distribution to achieve an optimal regret after T rounds of $O(\sqrt{KT\log(|\Pi|/\delta)})$, with probability at least $1-\delta$, relative to a set of policies $\Pi \subseteq A^X$ mapping contexts $x \in X$ to actions $a \in A$ (where K is the number of actions). The regret is the difference between the cumulative reward of the best policy in Π and the cumulative reward collected by the algorithm. Because the bound has only a logarithmic dependence on $|\Pi|$, the algorithm can compete with very large policy classes that are likely to yield high rewards, in which case the algorithm also earns high rewards. However, the computational cost of the above algorithms is linear in $|\Pi|$, which is tractable only for simple policy classes.

A sub-linear in $|\Pi|$ running time is possible for policy classes that can be efficiently searched. In this work, we

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

use the abstraction of an optimization oracle to capture this property: given a set of context/reward vector pairs, the oracle returns a policy in Π with maximum total reward. Using such an oracle in an i.i.d. setting (formally defined in Section 2.1), it is possible to create ϵ-greedy (Sutton & Barto, 1998) or epoch-greedy (Langford & Zhang, 2007) algorithms that run in time $O(\log|\Pi|)$ with only a single call to the oracle per round. However, these algorithms have suboptimal regret bounds of $O((K\log|\Pi|)^{1/3}T^{2/3})$ because the algorithms randomize uniformly over actions when they choose to explore.

The Randomized UCB algorithm of Dudík et al. (2011a) achieves the optimal regret bound (up to logarithmic factors) in the i.i.d. setting, and runs in time $\mathrm{poly}(T,\log|\Pi|)$ with $\tilde{O}(T^5)$ calls to the optimization oracle per round. Naively this would amount to $\tilde{O}(T^6)$ calls to the oracle over T rounds, although a doubling trick from our analysis can be adapted to ensure only $\tilde{O}(T^5)$ calls to the oracle are needed over all T rounds in the Randomized UCB algorithm. This is a fascinating result because it shows that the oracle can provide an exponential speed-up over previous algorithms with optimal regret bounds. However, the running time of this algorithm is still prohibitive for most natural problems owing to the $\tilde{O}(T^5)$ scaling.

In this work, we prove the following[1]:

Theorem 1. *There is an algorithm for the i.i.d. contextual bandit problem with an optimal regret bound requiring* $\tilde{O}\left(\sqrt{\frac{KT}{\ln(|\Pi|/\delta)}}\right)$ *calls to the optimization oracle over* T *rounds, with probability at least* $1-\delta$.

Concretely, we make $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ calls to the oracle with a net running time of $\tilde{O}(T^{1.5}\sqrt{K\log|\Pi|})$, vastly improving over the complexity of Randomized UCB. The major components of the new algorithm are (i) a new coordinate descent procedure for computing a very sparse distribution over policies which can be efficiently sampled from, and (ii) a new epoch structure which allows the distribution over policies to be updated very infrequently. We consider variants of the epoch structure that make different computational trade-offs; on one extreme we concentrate the entire computational burden on $O(\log T)$ rounds with $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ oracle calls each time, while on the other we spread our computation over $\sqrt{T}$ rounds with $\tilde{O}(\sqrt{K/\ln(|\Pi|/\delta)})$ oracle calls for each of these rounds. We stress that in either case, the total number of calls to the oracle is only sublinear in T. Finally, we develop a more efficient online variant, and conduct a proof-of-concept experiment showing low computational complexity and high reward relative to several natural baselines.

[1]Throughout this paper, we use the $\tilde{O}$ notation to suppress dependence on logarithmic factors in T and K, as well as $\log(|\Pi|/\delta)$ (i.e. terms which are $O(\log\log(|\Pi|/\delta))$.

Motivation and related work. The EXP4-family of algorithms (Auer et al., 2002; McMahan & Streeter, 2009; Beygelzimer et al., 2011) solve the contextual bandit problem with optimal regret by updating weights (multiplicatively) over all policies in every round. Except for a few special cases (Helmbold & Schapire, 1997; Beygelzimer et al., 2011), the running time of such *measure-based* algorithms is generally linear in the number of policies.

In contrast, the Randomized UCB algorithm of Dudík et al. (2011a) is based on a natural abstraction from supervised learning: efficiently finding a function in a rich function class that minimizes the loss on a training set. This abstraction is encapsulated in the notion of an optimization oracle, which is also used by ϵ-greedy (Sutton & Barto, 1998) and epoch-greedy (Langford & Zhang, 2007).

Another class of approaches based on Bayesian updating is Thompson sampling (Thompson, 1933; Li, 2013), which often enjoys strong theoretical guarantees in expectation over the prior and good empirical performance (Chapelle & Li, 2011). Such algorithms, as well as the closely related upper-confidence bound algorithms (Auer, 2002; Chu et al., 2011), are computationally tractable in cases where the posterior distribution over policies can be efficiently maintained or approximated. In our experiments, we compare to a strong baseline algorithm that uses this approach (Chu et al., 2011).

To circumvent the $\Omega(|\Pi|)$ running time barrier, we restrict attention to algorithms that only access the policy class via the optimization oracle. Specifically, we use a cost-sensitive classification oracle, and a key challenge is to design good supervised learning problems for querying this oracle. The Randomized UCB algorithm of Dudík et al. (2011a) uses a similar oracle to construct a distribution over policies that solves a certain convex program. However, the number of oracle calls in their work is prohibitively large, and the statistical analysis is also rather complex.[2]

Main contributions. In this work, we present a new and simple algorithm for solving a similar convex program as that used by Randomized UCB. The new algorithm is based on coordinate descent: in each iteration, the algorithm calls the optimization oracle to obtain a policy; the output is a sparse distribution over these policies. The number of iterations required to compute the distribution is small—at most $\tilde{O}(\sqrt{Kt/\ln(|\Pi|/\delta)})$ in any round t. In fact, we present a more general scheme based on epochs and warm start in which the total number of calls to the oracle is, with high probability, just $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ *over all* T *rounds*; we prove that this is nearly optimal for a certain class of optimization-based algorithms. The algorithm is natural

[2]The paper of Dudík et al. (2011a) is colloquially referred to, by its authors, as the "monster paper" (Langford, 2014).

and simple to implement, and we provide an arguably simpler analysis than that for Randomized UCB. Finally, we report proof-of-concept experimental results using a variant algorithm showing strong empirical performance.

2. Preliminaries

In this section, we recall the i.i.d. contextual bandit setting and some basic techniques used in previous works (Auer et al., 2002; Beygelzimer et al., 2011; Dudík et al., 2011a).

2.1. Learning Setting

Let A be a finite set of K actions, X be a space of possible contexts (*e.g.*, a feature space), and $\Pi \subseteq A^X$ be a finite set of policies that map contexts $x \in X$ to actions $a \in A$.[3] Let $\Delta^\Pi := \{Q \in \mathbb{R}^\Pi : Q(\pi) \geq 0 \, \forall \pi \in \Pi, \sum_{\pi \in \Pi} Q(\pi) \leq 1\}$ be the set of non-negative weights over policies with total weight at most one, and let $\mathbb{R}^A_+ := \{r \in \mathbb{R}^A : r(a) \geq 0 \, \forall a \in A\}$ be the set of non-negative reward vectors.

Let $\mathcal{D}$ be a probability distribution over $X \times [0,1]^A$, the joint space of contexts and reward vectors; we assume actions' rewards from $\mathcal{D}$ are always in the interval $[0,1]$. Let $\mathcal{D}_X$ denote the marginal distribution of $\mathcal{D}$ over X.

In the i.i.d. contextual bandit setting, the context/reward vector pairs $(x_t, r_t) \in X \times [0,1]^A$ over all rounds $t = 1, 2, \ldots$ are randomly drawn independently from $\mathcal{D}$. In round t, the agent first observes the context x_t, then (randomly) chooses an action $a_t \in A$, and finally receives the reward $r_t(a_t) \in [0,1]$ for the chosen action. The (observable) record of interaction resulting from round t is the quadruple $(x_t, a_t, r_t(a_t), p_t(a_t)) \in X \times A \times [0,1] \times [0,1]$; here, $p_t(a_t) \in [0,1]$ is the probability that the agent chose action $a_t \in A$. We let $H_t \subseteq X \times A \times [0,1] \times [0,1]$ denote the *history* (set) of interaction records in the first t rounds. We use $\widehat{\mathbb{E}}_{x \sim H_t}[\cdot]$ to denote expectation when a context x is uniformly chosen at random from the t contexts in H_t.

Let $\mathcal{R}(\pi) := \mathbb{E}_{(x,r)\sim\mathcal{D}}[r(\pi(x))]$ denote the expected (instantaneous) reward of a policy $\pi \in \Pi$, and let $\pi_\star := \arg\max_{\pi \in \Pi} \mathcal{R}(\pi)$ be a policy that maximizes the expected reward (the *optimal policy*). Let $\mathrm{Reg}(\pi) := \mathcal{R}(\pi_\star) - \mathcal{R}(\pi)$ denote the *expected (instantaneous) regret* of a policy $\pi \in \Pi$ relative to the optimal policy. Finally, the (empirical cumulative) regret of the agent after T rounds is defined as $\sum_{t=1}^T \big(r_t(\pi_\star(x_t)) - r_t(a_t)\big)$.

2.2. Inverse Propensity Scoring

An unbiased estimate of a policy's reward $\mathcal{R}(\pi)$ may be obtained from a history of interaction records H_t using *inverse propensity scoring* (IPS; also called *inverse probability weighting*): the expected reward of policy $\pi \in \Pi$ is estimated as

$$\widehat{\mathcal{R}}_t(\pi) := \frac{1}{t} \sum_{i=1}^{t} \frac{r_i(a_i) \cdot \mathbb{1}\{\pi(x_i) = a_i\}}{p_i(a_i)}. \qquad (1)$$

This technique can be viewed as mapping $H_t \mapsto \mathsf{IPS}(H_t)$ of interaction records $(x, a, r(a), p(a))$ to context/reward vector pairs $(x, \hat{r})$, where $\hat{r} \in \mathbb{R}^A_+$ is a fictitious reward vector that assigns to the chosen action a a scaled reward $r(a)/p(a)$ (possibly greater than one), and assigns to all other actions zero rewards. We may equivalently write $\widehat{\mathcal{R}}_t(\pi) = t^{-1} \sum_{(x,\hat{r}) \in \mathsf{IPS}(H_t)} \hat{r}(\pi(x))$; it is easy to verify that $\mathbb{E}[\hat{r}(\pi(x))|(x,r)] = r(\pi(x))$, as $p(a)$ is indeed the agent's probability (conditioned on (x,r)) of picking action a. This implies $\widehat{\mathcal{R}}_t(\pi)$ is an unbiased estimator for any history H_t.

Let $\pi_t := \arg\max_{\pi \in \Pi} \widehat{\mathcal{R}}_t(\pi)$ denote a policy that maximizes the expected reward estimate based on inverse propensity scoring with history H_t (π_0 can be arbitrary), and let $\widehat{\mathrm{Reg}}_t(\pi) := \widehat{\mathcal{R}}_t(\pi_t) - \widehat{\mathcal{R}}_t(\pi)$ denote *estimated regret* relative to π_t. Note that $\widehat{\mathrm{Reg}}_t(\pi)$ is generally *not* an unbiased estimate of $\mathrm{Reg}(\pi)$, because π_t is not always $\pi_\star$.

2.3. Optimization Oracle

One natural mode for accessing the set of policies Π is enumeration, but this is impractical in general. In this work, we instead only access Π via an optimization oracle which corresponds to a cost-sensitive learner. Following Dudík et al. (2011a), we call this oracle AMO[4].

Definition 1. *For a set of policies Π, the* $\arg\max$ *oracle (*AMO*) is an algorithm, which for any sequence of context and reward vectors, $(x_1, r_1), (x_2, r_2), \ldots, (x_t, r_t) \in X \times \mathbb{R}^A_+$, returns $\arg\max_{\pi \in \Pi} \sum_{\tau=1}^t r_\tau(\pi(x_\tau))$.*

2.4. Projections and Smoothing

In each round, our algorithm chooses an action by randomly drawing a policy π from a distribution over Π, and then picking the action $\pi(x)$ recommended by π on the current context x. This is equivalent to drawing an action according to $Q(a|x) := \sum_{\pi \in \Pi : \pi(x) = a} Q(\pi)$, $\forall a \in A$. For keeping the variance of reward estimates from IPS in check, it is desirable to prevent the probability of any action from being too small. Thus, as in previous work, we also use a smoothed projection $Q^\mu(\cdot|x)$ for $\mu \in [0, 1/K]$, $Q^\mu(a|x) := (1 - K\mu) \sum_{\pi \in \Pi : \pi(x) = a} Q(\pi) + \mu$, $\forall a \in A$. Every action has probability at least μ under $Q^\mu(\cdot|x)$.

For technical reasons, our algorithm maintains non-negative weights $Q \in \Delta^\Pi$ over policies that sum to at

[3] Extension to VC classes is simple using standard arguments.

[4] Cost-sensitive learners often need a cost instead of reward, in which case we use $c_t = \mathbb{1} - r_t$.

most one, but not necessarily equal to one; hence, we put any remaining mass on a default policy $\bar{\pi} \in \Pi$ to obtain a legitimate probability distribution over policies $\tilde{Q} = Q + \left(1 - \sum_{\pi \in \Pi} Q(\pi)\right) \mathbb{1}_{\bar{\pi}}$. We then pick an action from the smoothed projection $\tilde{Q}^{\mu}(\cdot|x)$ of $\tilde{Q}$ as above.

3. Algorithm and Main Results

Our algorithm (ILOVETOCONBANDITS) is an epoch-based variant of the Randomized UCB algorithm of Dudík et al. (2011a) and is given in Algorithm 1. Like Randomized UCB, ILOVETOCONBANDITS solves an optimization problem (OP) to obtain a distribution over policies to sample from (Step 7), but does so on an *epoch schedule*, *i.e.*, only on certain pre-specified rounds $\tau_1, \tau_2, \ldots$. The only requirement of the epoch schedule is that the length of epoch m is bounded as $\tau_{m+1} - \tau_m = O(\tau_m)$. For simplicity, we assume $\tau_{m+1} \leq 2\tau_m$ for $m \geq 1$, and $\tau_1 = O(1)$.

The crucial step here is solving (OP). Before stating the main result, let us get some intuition about this problem. The first constraint, Eq. (2), requires the average estimated regret of the distribution Q over policies to be small, since b_π is a rescaled version of the estimated regret of policy π. This constraint skews our distribution to put more mass on "good policies" (as judged by our current information), and can be seen as the exploitation component of our algorithm. The second set of constraints, Eq. (3), requires the distribution Q to place sufficient mass on the actions chosen by each policy π, in expectation over contexts. This can be thought of as the exploration constraint, since it requires the distribution to be sufficiently diverse for most contexts. As we will see later, the left hand side of the constraint is a bound on the variance of our reward estimates for policy π, and the constraint requires the variance to be controlled at the level of the estimated regret of π. That is, we require the reward estimates to be more accurate for good policies than we do for bad ones, allowing for much more adaptive exploration than that of ϵ-greedy style algorithms.

This problem is very similar to the one in Dudík et al. (2011a), and our coordinate descent algorithm in Section 3.1 gives a constructive proof that the problem is feasible. As in Dudík et al. (2011a), we have the following regret bound:[5]

Theorem 2. *Assume the optimization problem (*OP*) can be solved whenever required in Algorithm 1. With probability at least $1 - \delta$, the regret of Algorithm 1 (*ILOVETOCONBANDITS*) after T rounds is*

$$O\left(\sqrt{KT\ln(T|\Pi|/\delta)} + K\ln(T|\Pi|/\delta)\right).$$

[5]Omitted proofs and other details are given in the full version of the paper (Agarwal et al., 2014).

Algorithm 1 Importance-weighted LOw-Variance Epoch-Timed Oracleized CONtextual BANDITS algorithm (ILOVETOCONBANDITS)

input Epoch schedule $0 = \tau_0 < \tau_1 < \tau_2 < \cdots$, allowed failure probability $\delta \in (0, 1)$.

1: Initial weights $Q_0 := \mathbf{0} \in \Delta^{\Pi}$, initial epoch $m := 1$.
Define $\mu_m := \min\{1/2K, \sqrt{\ln(16\tau_m^2|\Pi|/\delta)/(K\tau_m)}\}$ for all $m \geq 0$.
2: **for round** $t = 1, 2, \ldots$ **do**
3: Observe context $x_t \in X$.
4: $(a_t, p_t(a_t)) := \mathsf{Sample}(x_t, Q_{m-1}, \pi_{\tau_m - 1}, \mu_{m-1})$.
5: Select action a_t and observe reward $r_t(a_t) \in [0, 1]$.
6: **if** $t = \tau_m$ **then**
7: Let Q_m be a solution to (OP) with history H_t and minimum probability μ_m.
8: $m := m + 1$.
9: **end if**
10: **end for**

Optimization Problem (OP)

Given a history H_t and minimum probability μ_m, define $b_\pi := \frac{\widehat{\mathrm{Reg}}_t(\pi)}{\psi\mu_m}$ for $\psi := 100$, and find $Q \in \Delta^{\Pi}$ such that

$$\sum_{\pi \in \Pi} Q(\pi) b_\pi \leq 2K \quad (2)$$

$$\forall \pi \in \Pi : \widehat{\mathbb{E}}_{x \sim H_t}\left[\frac{1}{Q^{\mu_m}(\pi(x)|x)}\right] \leq 2K + b_\pi. \quad (3)$$

3.1. Solving (OP) via Coordinate Descent

We now present a coordinate descent algorithm to solve (OP). The pseudocode is given in Algorithm 2. Our analysis, as well as the algorithm itself, are based on a potential function which we use to measure progress. The algorithm can be viewed as a form of coordinate descent applied to this same potential function. The main idea of our analysis is to show that this function decreases substantially on every iteration of this algorithm; since the function is non-negative, this gives an upper bound on the total number of iterations as expressed in the following theorem.

Theorem 3. *Algorithm 2 (with $Q_{\mathrm{init}} := \mathbf{0}$) halts in at most $\frac{4\ln(1/(K\mu_m))}{\mu_m}$ iterations, and outputs a solution Q to (*OP*).*

3.2. Using an Optimization Oracle

We now show how to implement Algorithm 2 via AMO (c.f. Section 2.3).

Lemma 1. *Algorithm 2 can be implemented using one call to* AMO *before the loop is started, and one call for each iteration of the loop thereafter.*

Proof. At the very beginning, before the loop is started,

Algorithm 2 Coordinate Descent Algorithm

Require: History H_t, minimum probability μ, initial weights $Q_{\text{init}} \in \Delta^{\Pi}$.

1: Set $Q := Q_{\text{init}}$.
2: **loop**
3: Define, for all $\pi \in \Pi$,

$$\begin{aligned} V_\pi(Q) &= \widehat{\mathbb{E}}_{x\sim H_t}[1/Q^\mu(\pi(x)|x)] \\ S_\pi(Q) &= \widehat{\mathbb{E}}_{x\sim H_t}\left[1/(Q^\mu(\pi(x)|x))^2\right] \\ D_\pi(Q) &= V_\pi(Q) - (2K + b_\pi). \end{aligned}$$

4: **if** $\sum_\pi Q(\pi)(2K + b_\pi) > 2K$ **then**
5: Replace Q by cQ, where

$$c := \frac{2K}{\sum_\pi Q(\pi)(2K + b_\pi)} < 1. \tag{4}$$

6: **end if**
7: **if** there is a policy π for which $D_\pi(Q) > 0$ **then**
8: Add the (positive) quantity

$$\alpha_\pi(Q) = \frac{V_\pi(Q) + D_\pi(Q)}{2(1 - K\mu)S_\pi(Q)}$$

to $Q(\pi)$ and leave all other weights unchanged.
9: **else**
10: Halt and output the current set of weights Q.
11: **end if**
12: **end loop**

we compute the best empirical policy so far, π_t, by calling AMO on the sequence of historical contexts and estimated reward vectors; *i.e.*, on $(x_\tau, \hat{r}_\tau)$, for $\tau = 1, 2, \ldots, t$.

Next, we show that each iteration in the loop of Algorithm 2 can be implemented via one call to AMO. Going over the pseudocode, first note that operations involving Q in Step 4 can be performed efficiently since Q has sparse support. Note that the definitions in Step 3 don't actually need to be computed for all policies $\pi \in \Pi$, as long as we can identify a policy π for which $D_\pi(Q) > 0$. We can identify such a policy using one call to AMO as follows.

First, note that for any policy π, we have $V_\pi(Q) = t^{-1}\sum_{\tau=1}^t 1/Q^\mu(\pi(x_\tau)|x_\tau)$, and $b_\pi = \widehat{\mathcal{R}}_t(\pi_t)/(\psi\mu) - (\psi\mu t)^{-1}\sum_{\tau=1}^t \hat{r}_\tau(\pi(x_\tau))$. Now consider the sequence of historical contexts and reward vectors, $(x_\tau, \tilde{r}_\tau)$ for $\tau = 1, 2, \ldots, t$, where for any action a we define

$$\tilde{r}_\tau(a) := \frac{1}{t}\left(\frac{\psi\mu}{Q^\mu(a|x_\tau)} + \hat{r}_\tau(a)\right). \tag{5}$$

Observe that $D_\pi(Q) = (\psi\mu)^{-1}\sum_{\tau=1}^t \tilde{r}_\tau(\pi(x_\tau)) +$ a constant independent of π. Therefore, $\arg\max_{\pi\in\Pi} D_\pi(Q) = \arg\max_{\pi\in\Pi}\sum_{\tau=1}^t \tilde{r}_\tau(\pi(x_\tau))$, and hence, calling AMO once on the sequence $(x_\tau, \tilde{r}_\tau)$ for $\tau = 1, 2, \ldots, t$, we obtain a policy that maximizes $D_\pi(Q)$, and thereby identify a policy for which $D_\pi(Q) > 0$ whenever one exists. □

3.3. Epoch Schedule

Recalling the setting of μ_m in Algorithm 1, Theorem 3 shows that Algorithm 2 solves (OP) with $\tilde{O}(\sqrt{Kt/\ln(|\Pi|/\delta)})$ calls to AMO in round t. Thus, if we use the epoch schedule $\tau_m = m$ (*i.e.*, run Algorithm 2 in every round), then we get a total of $\tilde{O}(\sqrt{KT^3/\ln(|\Pi|/\delta)})$ calls to AMO over all T rounds. This number can be dramatically reduced using a more carefully chosen epoch schedule.

Lemma 2. *For the epoch schedule $\tau_m := 2^{m-1}$, the total number of calls to* AMO *is $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$.*

Proof. The epoch schedule satisfies the requirement $\tau_{m+1} \le 2\tau_m$. With this epoch schedule, Algorithm 2 is run only $O(\log T)$ times over T rounds, leading to $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ total calls to AMO. □

3.4. Warm Start

We now present a different technique to reduce the number of calls to AMO. This is based on the observation that practically speaking, it seems terribly wasteful, at the start of a new epoch, to throw out the results of all of the preceding computations and to begin yet again from nothing. Instead, intuitively, we expect computations to be more moderate if we begin again where we left off last, *i.e.*, a "warm-start" approach. Here, when Algorithm 2 is called at the end of epoch m, we use $Q_{\text{init}} := Q_{m-1}$ (the previously computed weights) rather than $\mathbf{0}$.

We can combine warm-start with a different epoch schedule to guarantee $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ total calls to AMO, spread across $O(\sqrt{T})$ calls to Algorithm 2.

Lemma 3. *Define the epoch schedule $(\tau_1, \tau_2) := (3, 5)$ and $\tau_m := m^2$ for $m \ge 3$ (this satisfies $\tau_{m+1} \le 2\tau_m$). With high probability, the warm-start variant of Algorithm 1 makes $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$ calls to* AMO *over T rounds and $O(\sqrt{T})$ calls to Algorithm 2.*

3.5. Computational Complexity

So far, we have only considered computational complexity in terms of the number of oracle calls. However, the reduction also involves the creation of cost-sensitive classification examples, which must be accounted for in the net computational cost. With some natural bookkeeping of probabilities, the computational complexity of our algorithm, modulo the oracle running time, can be made to be $\tilde{O}(\sqrt{(KT)^3/\ln(|\Pi|/\delta)})$. Details are given in the full version of the paper.

3.6. A Lower Bound on the Support Size

An attractive feature of Algorithm 2 is that the number of calls to AMO is directly related to the number of policies in the support of Q_m. For instance, with the doubling schedule of Section 3.3, Theorem 3 implies that we never have non-zero weights for more than $\frac{4\ln(1/(K\mu_m))}{\mu_m}$ policies in epoch m. The support size of the distributions Q_m in Algorithm 1 is crucial to the computational cost of sampling an action.

We now demonstrate a lower bound showing that it is not possible to construct substantially sparser distributions that also satisfy the low-variance constraint (3) in the optimization problem (OP). To formally state the lower bound, for a given an epoch schedule (τ_m), define the following set of non-negative vectors over policies:

$$\mathcal{Q}_m := \{Q \in \Delta^\Pi : Q \text{ satisfies Eq. (3) in round } \tau_m\}.$$

(The distribution Q_m computed by Algorithm 1 is in $\mathcal{Q}_m$.) Recall that $\mathrm{supp}(Q)$ denotes the support of Q (the set of policies where Q puts non-zero entries). We have the following lower bound on $|\mathrm{supp}(Q)|$.

Theorem 4. *For any epoch schedule $0 = \tau_0 < \tau_1 < \tau_2 < \cdots$ and any $M \in \mathbb{N}$ sufficiently large, there exists a distribution $\mathcal{D}$ over $X \times [0,1]^A$ and a policy class Π such that, with probability at least $1-\delta$,*

$$\inf_{\substack{m\in\mathbb{N}:\\ \tau_m \geq \tau_M/2}} \inf_{Q\in\mathcal{Q}_m} |\mathrm{supp}(Q)| = \Omega\left(\sqrt{\frac{K\tau_M}{\ln(|\Pi|\tau_M/\delta)}}\right).$$

In the context of our problem, this lower bound shows that the bounds in Lemma 2 and Lemma 3 are unimprovable, since the number of calls to AMO is at least the size of the support, given our mode of access to Π.

4. Regret Analysis

In this section, we outline the regret analysis for our algorithm ILOVETOCONBANDITS.

The deviations of the policy reward estimates $\widehat{\mathcal{R}}_t(\pi)$ are controlled by (a bound on) the variance of each term in Eq. (1): essentially the left-hand side of Eq. (3) from (OP), except with $\widehat{\mathbb{E}}_{x\sim H_t}[\cdot]$ replaced by $\mathbb{E}_{x\sim\mathcal{D}_X}[\cdot]$. Resolving this discrepancy is handled using deviation bounds, so Eq. (3) holds with $\mathbb{E}_{x\sim\mathcal{D}_X}[\cdot]$, with worse right-hand side constants.

The rest of the analysis, which deviates from that of Randomized UCB, compares the expected regret $\mathrm{Reg}(\pi)$ of any policy π with the estimated regret $\widehat{\mathrm{Reg}}_t(\pi)$ using the variance constraints Eq. (3):

Lemma 4 (Informally)**.** *With high probability, for each m such that $\tau_m \geq \tilde{O}(K\log|\Pi|)$, each round t in epoch m, and each $\pi \in \Pi$, $\mathrm{Reg}(\pi) \leq 2\widehat{\mathrm{Reg}}_t(\pi) + O(K\mu_m)$.*

This lemma can easily be combined with the constraint Eq. (2) from (OP): since the weights Q_{m-1} used in any round t in epoch m satisfy $\sum_{\pi\in\Pi} Q_{m-1}(\pi)\widehat{\mathrm{Reg}}_{\tau_m-1}(\pi) \leq \psi \cdot 2K\mu_{\tau_m-1}$, we obtain a bound on the (conditionally) expected regret in round t using the above lemma: with high probability, $\sum_{\pi\in\Pi}\widetilde{Q}_{m-1}\mathrm{Reg}(\pi) \leq O(K\mu_{m-1})$. Summing these terms up over all T rounds and applying martingale concentration gives the bound in Theorem 2.

5. Analysis of the Optimization Algorithm

In this section, we give a sketch of the analysis of our main optimization algorithm for computing weights Q_m on each epoch as in Algorithm 2. As mentioned in Section 3.1, this analysis is based on a potential function.

Since our attention for now is on a single epoch m, here and in what follows, when clear from context, we drop m from our notation and write simply $\tau = \tau_m$, $\mu = \mu_m$, etc. Let $\mathcal{U}_A$ be the uniform distribution over the action set A. We define the following potential function for use on epoch m:

$$\Phi_m(Q) = \tau\mu\left(\frac{\widehat{\mathbb{E}}_x[\mathrm{RE}\left(\mathcal{U}_A \| Q^\mu(\cdot \mid x)\right)]}{1-K\mu} + \frac{\sum_{\pi\in\Pi} Q(\pi)b_\pi}{2K}\right).$$

This function is defined for all vectors $Q \in \Delta^\Pi$. Also, $\mathrm{RE}\left(p\|q\right)$ denotes the unnormalized relative entropy between two nonnegative vectors p and q over the action space (or any set) A: $\mathrm{RE}\left(p\|q\right) = \sum_{a\in A}(p_a\ln(p_a/q_a) + q_a - p_a)$, which is always nonnegative. Here, $Q^\mu(\cdot|x)$ denotes the "distribution" (which might not sum to 1) over A induced by Q^μ for context x as given in Section 2.4. Thus, ignoring constants, this potential function is a combination of two terms: The first measures how far from uniform are the distributions induced by Q^μ, and the second is an estimate of expected regret under Q since b_π is proportional to the empirical regret of π. Making Φ_m small thus encourages Q to choose actions as uniformly as possible while also incurring low regret — exactly the aims of our algorithm. The constants that appear in this definition are for later mathematical convenience.

For further intuition, note that, by straightforward calculus, the partial derivative $\partial\Phi_m/\partial Q(\pi)$ is roughly proportional to the variance constraint for π given in Eq. (3) (up to a slight mismatch of constants). This shows that if this constraint is not satisfied, then $\partial\Phi_m/\partial Q(\pi)$ is likely to be negative, meaning that Φ_m can be decreased by increasing $Q(\pi)$. Thus, the weight vector Q that minimizes Φ_m satisfies the variance constraint for every policy π. It turns out that this minimizing Q also satisfies the low regret constraint in Eq. (2), and also must sum to at most 1; in other words, it provides a complete solution to our optimization problem. Algorithm 2 does not fully minimize Φ_m, but it

is based roughly on coordinate descent. This is because in each iteration one of the weights (coordinate directions) $Q(\pi)$ is increased. This weight is one whose corresponding partial derivative is large and negative.

To analyze the algorithm, we first argue that it is correct in the sense of satisfying the required constraints, provided that it halts.

Lemma 5. *If Algorithm 2 halts and outputs a weight vector Q, then the constraints Eq. (3) and Eq. (2) must hold, and furthermore the sum of the weights $Q(\pi)$ is at most 1.*

What remains is the more challenging task of bounding the number of iterations until the algorithm does halt. We do this by showing that significant progress is made in reducing Φ_m on every iteration. To begin, we show that scaling Q as in Step 4 cannot cause Φ_m to increase.

Lemma 6. *Let Q be a weight vector such that $\sum_\pi Q(\pi)(2K + b_\pi) > 2K$, and let c be as in Eq. (4). Then $\Phi_m(cQ) \leq \Phi_m(Q)$.*

Next, we show that substantial progress will be made in reducing Φ_m each time that Step 8 is executed.

Lemma 7. *Let Q denote a set of weights and suppose, for some policy π, that $D_\pi(Q) > 0$. Let Q' be a new set of weights which is an exact copy of Q except that $Q'(\pi) = Q(\pi) + \alpha$ where $\alpha = \alpha_\pi(Q) > 0$. Then $\Phi_m(Q) - \Phi_m(Q') \geq \tau\mu^2/(4(1 - K\mu))$.*

So Step 4 does not cause Φ_m to increase, and Step 8 causes Φ_m to decrease by at least the amount given in Lemma 7. This immediately implies Theorem 3: for $Q_{\text{init}} = \mathbf{0}$, the initial potential is bounded by $\tau\mu \ln(1/(K\mu))/(1 - K\mu)$, and it is never negative, so the number of times Step 8 is executed is bounded by $4 \ln(1/(K\mu))/\mu$ as required.

5.1. Epoching and Warm Start

As shown in Section 2.3, the bound on the number of iterations of the algorithm from Theorem 3 also gives a bound on the number of times the oracle is called. To reduce the number of oracle calls, one approach is the "doubling trick" of Section 3.3, which enables us to bound the total combined number of iterations of Algorithm 2 in the first T rounds is only $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$. This means that the average number of calls to the arg-max oracle is only $\tilde{O}(\sqrt{K/(T\ln(|\Pi|/\delta))})$ per round, meaning that the oracle is called far less than once per round, and in fact, at a vanishingly low rate.

We now turn to warm-start approach of Section 3.4, where in each epoch $m+1$ we initialize the coordinate descent algorithm with $Q_{\text{init}} = Q_m$, i.e. the weights computed in the previous epoch m. To analyze this, we bound how much the potential changes from $\Phi_m(Q_m)$ at the end of epoch m to $\Phi_{m+1}(Q_m)$ at the very start of epoch $m + 1$. This, combined with our earlier results regarding how quickly Algorithm 2 drives down the potential, we are able to get an overall bound on the total number of updates across T rounds.

Lemma 8. *Let M be the largest integer for which $\tau_{M+1} \leq T$. With probability at least $1 - 2\delta$, for all T, the total epoch-to-epoch increase in potential is*

$$\sum_{m=1}^{M} (\Phi_{m+1}(Q_m) - \Phi_m(Q_m)) \leq \tilde{O}\left(\sqrt{\frac{T\ln(|\Pi|/\delta)}{K}}\right),$$

where M is the largest integer for which $\tau_{M+1} \leq T$.

This lemma, along with Lemma 7 can be used to further establish Lemma 3. We only provide an intuitive sketch here, with the details deferred to the appendix. As we observe in Lemma 8, the total amount that the potential increases across T rounds is at most $\tilde{O}(\sqrt{T\ln(|\Pi|/\delta)/K})$. On the other hand, Lemma 7 shows that each time Q is updated by Algorithm 2 the potential decreases by at least $\tilde{\Omega}(\ln(|\Pi|/\delta)/K)$ (using our choice of μ). Therefore, the total number of updates of the algorithm totaled over all T rounds is at most $\tilde{O}(\sqrt{KT/\ln(|\Pi|/\delta)})$. For instance, if we use $(\tau_1, \tau_2) := (3, 5)$ and $\tau_m := m^2$ for $m \geq 3$, then the Q is only updated about $\sqrt{T}$ times over T rounds; on each of those rounds, Algorithm 2 requires $\tilde{O}(\sqrt{K/\ln(|\Pi|/\delta)})$ iterations, on average, giving the claim in Lemma 3.

6. Experimental Evaluation

In this section we evaluate a variant of Algorithm 1 against several baselines. While Algorithm 1 is significantly more efficient than many previous approaches, the overall computational complexity is still at least $\tilde{O}((KT)^{1.5})$ plus the total cost of the oracle calls, as discussed in Section 3.5. This is markedly larger than the complexity of an ordinary supervised learning problem where it is typically possible to perform an $O(1)$-complexity update upon receiving a fresh example using online algorithms.

A natural solution is to use an *online* oracle that is stateful and accepts examples one by one. An online cost-sensitive classification (CSC) oracle takes as input a weighted example and returns a predicted class (corresponding to one of K actions in our setting). Since the oracle is stateful, it remembers and uses examples from all previous calls in answering questions, thereby reducing the complexity of each oracle invocation to $O(1)$ as in supervised learning. Using several such oracles, we can efficiently track a distribution over good policies and sample from it. We detail this approach (which we call Online Cover) in the full version of the paper. The algorithm maintains a uniform distribution over a fixed number n of policies where n is a parameter of the algorithm. Upon receiving a fresh example, it updates

Table 1. Progressive validation loss, best hyperparameter values, and running times of various algorithm on RCV1.

Algorithm	ϵ-greedy	Explore-first	Bagging	LinUCB	Online Cover	Supervised
P.V. Loss	0.148	0.081	0.059	0.128	0.053	0.051
Searched	$0.1 = \epsilon$	2×10^5 first	16 bags	10^3 dim, minibatch-10	cover $n = 1$	nothing
Seconds	17	2.6	275	212×10^3	12	5.3

all n policies with the suitable CSC examples (Eq. (5)). The specific CSC oracle we use is a reduction to squared-loss regression (Algorithms 4 and 5 of Beygelzimer & Langford (2009)) which is amenable to online updates. Our implementation is included in Vowpal Wabbit.[6]

Due to lack of public datasets for contextual bandit problems, we use a simple supervised-to-contextual-bandit transformation (Dudík et al., 2011b) on the CCAT document classification problem in RCV1 (Lewis et al., 2004). This dataset has 781265 examples and 47152 TF-IDF features. We treated the class labels as actions, and one minus 0/1-loss as the reward. Our evaluation criteria is progressive validation (Blum et al., 1999) on 0/1 loss. We compare several baseline algorithms to Online Cover; all algorithms take advantage of linear representations which are known to work well on this dataset. For each algorithm, we report the result for the best parameter settings (shown in Table 6).

1. ϵ-greedy (Sutton & Barto, 1998) explores randomly with probability ϵ and otherwise exploits.

2. Explore-first is a variant that begins with uniform exploration, then switches to an exploit-only phase.

3. A less common but powerful baseline is based on bagging: multiple predictors (policies) are trained with examples sampled with replacement. Given a context, these predictors yield a distribution over actions from which we can sample.

4. LinUCB (Auer, 2002; Chu et al., 2011) has been quite effective in past evaluations (Li et al., 2010; Chapelle & Li, 2011). It is impractical to run "as is" due to high-dimensional matrix inversions, so we report results for this algorithm after reducing to 1000 dimensions via random projections. Still, the algorithm required 59 hours[7]. An alternative is to use diagonal approximation to the covariance, which runs substantially faster ($\approx$1 hour), but gives a worse error of 0.137.

5. Finally, our algorithm achieves the best loss of 0.0530. Somewhat surprisingly, the minimum occurs for us with a cover set of size 1—apparently for this problem the small decaying amount of uniform random sampling imposed is adequate exploration. Prediction performance is similar with a larger cover set.

All baselines except for LinUCB are implemented as a simple modification of Vowpal Wabbit. All reported results use default parameters where not otherwise specified. The contextual bandit learning algorithms all use a doubly robust reward estimator instead of the importance weighted estimators used in our analysis (Dudík et al., 2011b).

Because RCV1 is actually a fully supervised dataset, we can apply a fully supervised online multiclass algorithm to solve it. We use a simple one-against-all implementation to reduce this to binary classification, yielding an error rate of 0.051 which is competitive with the best previously reported results. This is effectively a lower bound on the loss we can hope to achieve with algorithms using only partial information. Our algorithm is less than 2.3 times slower and nearly achieves the bound. Hence on this dataset, very little further algorithmic improvement is possible.

7. Conclusions

In this paper we have presented the first practical algorithm to our knowledge that attains the statistically optimal regret guarantee and is computationally efficient in the setting of general policy classes. A remarkable feature of the algorithm is that the total number of oracle calls over all T rounds is sublinear—a remarkable improvement over previous works in this setting. We believe that the online variant of the approach which we implemented in our experiments has the right practical flavor for a scalable solution to the contextual bandit problem. In future work, it would be interesting to directly analyze the Online Cover algorithm.

Acknowledgements

We thank Dean Foster and Matus Telgarsky for helpful discussions. Part of this work was completed while DH and RES were visiting Microsoft Research.

References

Agarwal, Alekh, Hsu, Daniel, Kale, Satyen, Langford, John, Li, Lihong, and Schapire, Robert E. Taming the

[6] `http://hunch.net/~vw`. The implementation is in the file `cbify.cc` and is enabled using `--cover`.

[7] The linear algebra routines are based on Intel MKL package.

monster: A fast and simple algorithm for contextual bandits. *CoRR*, abs/1402.0555, 2014.

Auer, Peter. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

Auer, Peter, Cesa-Bianchi, Nicolò, Freund, Yoav, and Schapire, Robert E. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32(1):48–77, 2002.

Beygelzimer, Alina and Langford, John. The offset tree for learning with partial labels. In *KDD*, 2009.

Beygelzimer, Alina, Langford, John, Li, Lihong, Reyzin, Lev, and Schapire, Robert E. Contextual bandit algorithms with supervised learning guarantees. In *AISTATS*, 2011.

Blum, Avrim, Kalai, Adam, and Langford, John. Beating the holdout: Bounds for k-fold and progressive cross-validation. In *COLT*, 1999.

Chapelle, Olivier and Li, Lihong. An empirical evaluation of Thompson sampling. In *NIPS*, 2011.

Chu, Wei, Li, Lihong, Reyzin, Lev, and Schapire, Robert E. Contextual bandits with linear payoff functions. In *AISTATS*, 2011.

Dudík, Miroslav, Hsu, Daniel, Kale, Satyen, Karampatziakis, Nikos, Langford, John, Reyzin, Lev, and Zhang, Tong. Efficient optimal learning for contextual bandits. In *UAI*, 2011a.

Dudík, Miroslav, Langford, John, and Li, Lihong. Doubly robust policy evaluation and learning. In *ICML*, 2011b.

Helmbold, David P. and Schapire, Robert E. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, 1997.

Langford, John. Interactive machine learning, January 2014. URL http://hunch.net/~jl/projects/interactive/index.html.

Langford, John and Zhang, Tong. The epoch-greedy algorithm for contextual multi-armed bandits. In *NIPS*, 2007.

Lewis, David D, Yang, Yiming, Rose, Tony G, and Li, Fan. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

Li, Lihong. Generalized Thompson sampling for contextual bandits. *CoRR*, abs/1310.7163, 2013.

Li, Lihong, Chu, Wei, Langford, John, and Schapire, Robert E. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

McMahan, H. Brendan and Streeter, Matthew. Tighter bounds for multi-armed bandits with expert advice. In *COLT*, 2009.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement learning, an introduction*. MIT Press, 1998.

Thompson, William R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.