Theory and Algorithms for Bandit Problems

Dick Jessen William
A0200677H

March 1, 2023

Contents

Chapter 1

Abstract

In this report, we will discuss about a variation of a bandit problem that focuses on the evaluation of an unknown point on a known function rather than querying the point itself. We will first discuss in Chapter 2 the introduction and motivation of the problem, and why and how it is different than the original bandit problem. Next, Chapters 3 and 4 will delve on the results of [2] that outlines an algorithm to adaptively query the arms based on the gradients, a specialized algorithm for thresholding, and examine the gain (reduced number of pulls) compared to a naive uniform sampling. Then, Chapters 5 and 6 will discuss a few new problems about finding the maximum valuation given multiple vectors and functions. Finally, Chapter 7 supplements the theoretical discussion by doing some experiments to show empirical results. The back page of this report contains the list of notations used.

Chapter 2

Introduction and Motivation

The theory of (multi-armed) bandits has been discussed by many sources, and has a lot of practical use in real life [7]. The general form of a bandit problem is that we are given a unknown vector $\mu \in \mathbb{R}^{n\times 1}$, we could pick an index (or an arm), and then sample the arm based on the underlying probability distribution (that are hidden from us). The end goal of the problem is to minimize the regret function $T\mu^* - \sum_{i=1}^{T} r_i$, where each r_i denotes the result of the pull in round i and T denotes the total number of rounds. By a simple observation, this problem boils down to finding which arm has the largest expected value as fast as possible, and exclusively pull from there.

Now, we will consider the variation of the problem, which will be our main focus in the report. We still have the unknown μ that we can pull with some error, but now we are given an additional function $f : \mathbb{R}^n \to \mathbb{R}$, and we want to produce an approximation of $f(\mu)$ by giving a guess $\hat{\mu}$ such that $|f(\mu) - f(\hat{\mu})|$ is as small as possible.

In the case when $f = \max_{i\in[1,n]} \mu_i$, we can see that this is just a classic bandit problem, with the i-th element of μ act as the i-th arm with mean reward (value) μ_i. Then f means that we want to approximate the best reward we can get. The family of functions f that we will examine in the report will be functions that have Lipschitz gradients, which means that there exists L such that $|\nabla f(\mu_1) - \nabla f(\mu_2)| \leq L|\mu_1 - \mu_2|$ (using the suitable distance function).

We now examine some differences between the original bandit problem and this problem. From the description perspective, the presence of f means that we will have some information befor we begin our search. More importantly, because we only care about $f(\mu)$, this means that there may be more than one correct $\hat{\mu}$. As a simple example, if μ is a two dimensional vector equal to $[1,1]$, and our function is $f(\mu) = ||\mu||_2^2$, we have both $\hat{\mu}_1 = [-1,-1]$ and $\hat{\mu}_2 = [1,-1]$ as a perfect guess (with zero error) of $f(\mu)$ (in fact, there are infinitely many of them).

Now, we see some of the real life example of problems in this setting [2].

1. In software reliability testing, suppose that there are n tests, $C_1, C_2, \cdots, C_n$ for a software, and a test C_i have a (fixed, but unknown) probability μ_i of failing. We also know an operational profile $\{p_i\}_{i=1}^{n}$, where p_i denotes the fraction of time

spent by the software in test C_i. The unreliability of the software is then defined by $f(\mu) = \sum_{i=1}^{n} p_i \mu_i = p^\intercal \mu$.

2. In Monte-Carlo integration, the goal is to integrate f over a domain X by dividing X into some strata and use adaptive allocation. Suppose ω_i be the measure of a strata and μ_i is the expectation over strata i. Our guess is then be $f(\mu) = \sum_i \mu_i \omega_i$ (in one variable functions, this is akin to dividing the interval to several chunks, sample to get the expectation from each chunk, and sum all of them. However, while in integration, we want to find $\mathbb{E}[f(x)]$ for $x \in X$, our setting will focus on $f(\mathbb{E}[x])$, because in our case $\mu = E[X]$ is fixed (we can only get noisy samples). Fortunately, because f is linear, these two coincides.

   We can also use bandit algorithms as a computation tool. As nowadays datasets are much larger, these algorithms can be used to construct instance-optimal (best algorithm for any input) for computational tasks. Examples are Monte-Carlo tree search and hyperparameter tuning, which use adaptivity to spend more resource to better solutions. This is formalized in the work of Bandit-Based Monte Carlo Optimization [1], where it is used to problems such as finding a medoid (the most "centered" data point) on a dataset, finding a $k-$ nearest neighbour construction, Monte Carlo testing and mode estimation. Another example in linear algebra field is found in [4]. Later in this paper, we will apply this algorithm for the function $f(\mu) = ||A\mu||_2^2$, which appears often in linear algebra.

Chapter 3

Derivation of the Function Approximation Algorithm

This chapter focuses on the derivation of the main algorithm used in the report. We first look at the mathematical intuition to get the idea, and then look at the proof that our algorithm works as intended.

3.1 Algorithm Exposition

Before we start, lets state the problem setting for this chapter. We are given a known Lipschitz function $f : \mathbb{R}^n \to \mathbb{R}$ and an unknown $\mu \in \mathbb{R}^n$. Also, we have parameters ε (error bound) and δ (probability of failure). Each time step, we can query an arm (index) $i \in [n]$ and an oracle will give us $\mu_i + Z_{i,t}$ with $Z_{i,t}$ follows the standard normal distribution. Denote T as the number of queries needed to approximate $f(\mu)$ within ε distance with probability $1-\delta$. Our goal is to find an algorithm that minimizes $\mathbb{E}[T]$.

As the first step, we look at the bound $|f(\mu) - f(\hat{\mu})|$. We use the property of L-Lipschitz gradients from [3] that states that

$$|f(\mu) - f(\hat{\mu}) - \triangledown f(\mu)^{\intercal}(\hat{\mu} - \mu)| \leq \frac{L}{2}||\hat{\mu} - \mu||_2^2.$$

After T_i samples, our estimator for $\hat{\mu}_i$ can be described as $\hat{\mu}_i = \frac{1}{T_i}\sum_{j=1}^{T_i} \mu_i + Z_{i,j}$ (average of all predictions). By property of variances, we have $\hat{\mu}_i \sim N(\mu_i, 1/T_i)$ after T_i samples. This means that $||\hat{\mu} - \mu||_2^2$ can be bounded by uniform sampling each arms. Next, we focus on the term $|\triangledown f(\mu)^{\intercal}(\hat{\mu} - \mu)|$. We note that this follows $N(0, \sum_{i=1}^{n} \frac{g_i^2}{T_i})$. To make sure this number is less than ε, we use Hoeffding inequality to infer that we must sample until the inequality $\sum_{i=1}^{n} \frac{g_i^2}{T_i} \leq \frac{2\varepsilon^2}{\log(2/\delta)}$. So, we want to reach this as soon as possible. Suppose α is our $n-$dimensional simplex (which means $\sum_{i=1}^{n} \alpha_i = 1$ that represents our sampling distribution. In other words, $T_i = \alpha_i T$. Then, we want to minimize over α. More formally, we have this minimization problem

$$\min_{|\alpha|=1} \sum_{i=1}^{n} \frac{g_i^2}{\alpha_i T}.$$

By Cauchy-Schwarz inequality [5] α should be set to be propotional to $|g_i|$.

If this vector g is known, then so are g_i, and we only need to sample based on this vector. Hence, adaptivity is not necessary. However, in our setting, $g = \triangledown f(\mu)$ is unknown (in general, as if we know f is linear, then g is known). To combat this, we construct a good sample μ in rounds. In each round r, we have twice as much budget $B_r = 2^r B_0$. Each round will then refine our prediction until at some point, where sampling based on the optimal frequency will be accurate enough. After this point, we use the plug-in estimator based on the current frequency. In other words, we first guess the distribution (finding the g_i's) to optimal pull by querying every arm uniformly, and based on the percieved distribution, we then do that pull to obtain the final guess.

We record the estimators $\hat{\mu}^{(r)}$ for estimating $\triangledown f(\mu)$, where $\hat{\mu}_i^{(r)}$ is the average of all samples from arm i up to (and including) round r. We later show that there is a confidence interval with width C_r, where after r rounds, we have

$$|\hat{\mu_i}^{(r)} - \mu_i| \leq \sqrt{\frac{2\log(12nr^2/\delta)}{\tilde{B}_r}} \triangleq C_r$$

with high probability (proof later). Note that this is not tight.

For estimation, we keep track of our gradient bounds as follows. Define

$$\hat{g_{i,\varepsilon,\delta}}^{(r,L)} \triangleq \min_{y:||y-\hat{\mu}_r||_\infty \leq C_r} \triangledown_i f(y),$$

$$\hat{g_{i,\varepsilon,\delta}}^{(r,U)} \triangleq \max_{y:||y-\hat{\mu}_r||_\infty \leq C_r} \triangledown_i f(y).$$

Note that the ε, δ will be ignored for the default case. If the confidence bound holds, we have that g_i lies on the interval $(\hat{g}_i^{(r,L)}, \hat{g}_i^{(r,U)})$ for all i, r. Recalling the earlier discussion on minimizing $\min_{|\alpha|=1} \sum_{i=1}^{n} \frac{g_i^2}{\alpha_i T}$ by making α propotional to g, a small $|g_i|$ corresponds to a small amount of pulls for that arm. This is good because we want to focus on the most important (i.e. high g_i) arms.

A coarser (larger width), but easier to compute definition of $\hat{g}_i^{(r,L)}$ is $(|\triangledown_i f(\hat{\mu}^{(r)})| - L\sqrt{n}C_r)_+$, with $(x)_+ \triangleq \max(x,0)$. This does not use the local geometry of the problem, but uses the Lipschitz bound L. Analogously, we can let $\hat{g}_i^{(r,U)}$ as $(|\triangledown_i f(\hat{\mu}^{(r)})| + L\sqrt{n}C_r)_+$. Using this, we now won't stop sampling important arms too soon because the upper bound of g_i (that we use) is more forgiving.

Now that we have laid out the motivations, we state the full exposition of the algorithm by [2]

Algorithm 1 Adaptive Function Approximation

```
Input: Arms $1,2,\cdots,n$, known function $f$, target $\varepsilon$, error bound $\delta$
$B_0 \leftarrow 17Ln^2\varepsilon^{-1}\log 12n/\delta$
for $r = 1,2,\cdots,n$ do
    $B_r \leftarrow 2^r B_0$.
    Pull every arm $\lfloor B_r/n \rfloor$ times.
    Use every sample so far to get $\hat{\mu}^{(r)}$.
    Find bounds $\{\hat{g_i}^{(r,L)}, \hat{g_i}^{(r,U)}\}$.
    Construct $\alpha_i^r$ proportional to $\hat{g_i}^{(r,L)}$.
    Set $\tilde{T}_i^{(r)} \triangleq \lceil(\alpha_i + 1/n)B_r\rceil$.
    if $\sum_i \frac{\hat{g_i}^{(r,U)2}}{\tilde{T}_i^{(r)}} \leq \frac{\varepsilon^2}{16\log(6/\delta)}$. then
        Pull arm $i$ $\tilde{T}_i^{(r)}$ times for $i = 1,2,\cdots n$ to construct $\tilde{\mu}$. return $f(\tilde{\mu})$
    end if
end for
```

Now, we outline the performance of the algorithm. First, here is a bound on the number of arms pulled as well as the performance guarantee.

Theorem 1. *[2] Algorithm 1 succeeds in finding $\tilde{\mu}$ with $|f(\tilde{(\mu)}) - f(\mu)| \leq \varepsilon$ with probability at least $1-\delta$ and using at most*

$$O\left(\frac{||\nabla f(\mu)||_1^2 \log 1/\delta}{\varepsilon^2} + \frac{n^2 L \log n/\delta}{\varepsilon}\right)$$

arm pulls.

To prove this, the approach is as follows:

1. Show that the mean estimators $\{\hat{\mu_i}^{(r)}\}$ satisfies the confidence bounds with high probability. We also show that with high probability, the algorithm terminates and returns the correct arm,

2. Show that our second order error (from the $\frac{L}{2}||\hat{\mu} - \mu||_2^2$ term in chapter 4, first equation) is less than $\varepsilon/2$. This is covered by the size of B_0.

3. Show that we sample important arms enough. More importantly, arms with small g_i are sampled enough from the uniform sampling stage and arms with large g_i will have large $\alpha_i^{(r)}$.

4. Bound the number of samples as claimed.

3.2 Proof for Theorem 1

The proof follows the previous plan.

3.2.1 Mean estimators on Confidence Bounds

Here, we define a good event ξ where the mean estimators $\{\mu_i^{(r)}\}$ stays in the confidence intervals, i.e.

$$\forall i, \forall r \{|\hat{\mu}_i{}^{(r)} - \mu_i| < C_r\}.$$

Our goal is to show the following.

Lemma 1. *The event ξ satisfies*

$$\mathbb{P}(\xi) \geq 1 - \frac{\delta}{3}.$$

Proof. We use union bound over the arms and all rounds. Denote $\bar{\xi}$ as the (bad) event, which means that there exists a round where there is one arm that lies outside the confidence bound on the current round. Note that we can use union bound [6]

$$\mathbb{P}(\bar{\xi}) = \mathbb{P}(\exists i \in [n], r \in \mathbb{N}, |\hat{\mu}_i{}^{(r)} - \mu_i| \geq C_r) \leq \sum_{i \in [n], r \in \mathbb{N}} 2e^{-\log 12nr^2/\delta} \leq \delta/3.$$

□

Then, we observe that from our definition of $g_i^{(r,L)}$,

$$\begin{aligned}
g_i^{(r,L)} &= \min_{y:||y-\hat{\mu}^{(r)}||_\infty \leq C_r} |\nabla_i f(y)| && \text{Definition} \\
&\geq g_i - \max_{y:||y-\hat{\mu}||_\infty \leq 2C_r} ||\nabla_i f(y) - \nabla_i f(\mu)|| && \text{Relax bound, and } g_i = \nabla_i f(\mu) \\
&\geq g_i - \max_{y:||y-\hat{\mu}||_\infty \leq 2C_r} ||\nabla f(y) - \nabla f(\mu)|| \\
&\geq g_i - 2L\sqrt{n}C_r. && \text{Invoke Lipschitzness}
\end{aligned}$$

Also, trivially, $g_i \geq g_i^{(r,L)}$. Hence, (by applying a similar bound for $g_i^{(r,U)}$), we have the chain of inequalities

$$g_u + 2L\sqrt{n}C_r \geq g_i^{(r,U)} \geq g_i \geq g_i^{(r,L)} \geq g_i - 2L\sqrt{n}C_r.$$

We will prove the following.

Lemma 2. *We define $H = \sum_i g_i$. Assume $g_i \geq 4L\sqrt{n}C_r$. Then, the following holds:*

$$\alpha_i^{(r)} = \frac{\hat{g}_i{}^{(r,L)}}{\sum_j \hat{g}_j{}^{(r,L)}} \geq \frac{g_i}{2H}.$$

Proof. Denote $A = 2L\sqrt{n}C_r$. Then, we have $g_i \geq 2A$ from our assumption and $g_i \geq \hat{g}_i{}^{(r,L)} \geq g_i - A$. Then, note that

$$\begin{aligned}
\frac{\hat{g}_i{}^{(r,L)}}{\sum_j \hat{g}_j{}^{(r,L)}} &\geq \frac{\hat{g}_i - A}{\sum_j \hat{g}_j} && \text{Smaller numerator, larger denominator} \\
&= \frac{\hat{g}_i - A}{H}
\end{aligned}$$

Because $g_i \geq 2A$, then $2(g_i - A) \geq g_i$. Hence, the last term is at least $\frac{g_i}{2H}$, and we are done. □

The next lemma bounds the second order error (which comes from sampling everything uniformly).

Lemma 3. *If each arm is sampled at least* $2nL\log(6n/\delta)\varepsilon^{-1}$ *times, then with probability at least* $1-\delta/3$*, we have*

$$\frac{L}{2}||\hat{\mu}-\mu||_2^2 \le \varepsilon/2.$$

Proof. First, note that $||x||_2^2 \le ||x||_\infty^2 n$, when x is a vector with dimension n. Hence,

$$\mathbb{P}\left(\frac{L}{2}||\hat{\mu}-\mu||_2^2 \ge \varepsilon/2\right) \le \mathbb{P}\left(||\hat{\mu}-\mu||_\infty^2 \ge \frac{\varepsilon}{nL}\right) \le 2\sum_{i=1}^{n} e^{-\frac{\tilde{T}_i^{(r)}\varepsilon}{2nL}} \le \delta/3.$$

The last part uses the assumption that $\tilde{T}_i^{(r)} \le 2nL\log 6n/\delta\varepsilon^{-1}$. This is enough from the initial budget $B_0 \ge 2n^2L\log 6n/\delta\varepsilon^{-1}$. $\square$

Now, we move on to the linear approximation. Note that using Chernoff/Hoeffding, $\sum_{i=1}^{n}\frac{g_i^2}{\tilde{T}_i^{(r)}} \le \frac{\varepsilon^2}{8\log 6/\delta} \implies \mathbb{P}(|\nabla f(\mu)^\intercal(\hat{\mu}-\mu)| > \varepsilon/2) \le \delta/3$.

However, the true value of $\nabla f(\mu)$ is unknown, so this cannot be computed. Hence, we need to use a stopping condition involving our estimators. Hence, we assume ξ to force the gradients in the bounds. Because of our assumption of ξ and the terminating condition that we uses, we have ($g_i^{(r)} \le g_i^{(r,U)}$)

$$\sum_{i=1}^{n}\frac{g_i^2}{\tilde{T}_i^{(r,U)}} \le \frac{\varepsilon^2}{8\log 6/\delta} \implies \sum_{i=1}^{n}\frac{g_i^2}{\tilde{T}_i^{(r)}} \le \frac{\varepsilon^2}{8\log 6/\delta}.$$

This means that if ξ happens, our algorithm will not end early. Hence, the algorithm works correctly as intended.

3.2.2 Runtime Claim

We split the arms into 2 sets: the ones with $g_i \ge 4L\sqrt{n}C_r$ and $g_i < 4L\sqrt{n}C_r$. Then, we use two different bounds for each groups of arms. Noting the stopping condition, we have (denote $\tau = 4L\sqrt{n}C_r$),

$$\begin{aligned}
\sum_{i=1}^{n}\frac{\hat{g}_i^{(r,U)^2}}{\tilde{T}_i^{(r)}} &\le \sum_{i=1}^{n}\frac{(g_i+2L\sqrt{n}C_r)^2}{\tilde{T}_i^{(r)}} && \text{Bound of } g_i^{(r,U)} \\
&\le \sum_{i|g_i<\tau}^{n}\frac{(g_i+2L\sqrt{n}C_r)^2}{B_r/n} + \sum_{i|g_i\ge\tau}\frac{(g_i+2L\sqrt{n}C_r)^2}{\alpha_i B_r} && \text{Divide arms into 2 cases} \\
&\le \sum_{i|g_i<\tau}^{n}\frac{36nL^2C_r^2}{B_r/n} + \sum_{i|g_i\ge\tau}\frac{9g_iH}{4B_r} && g_i<\tau\text{, Lemma 2 and } 2L\sqrt{n}C_r \le g_i/2 \\
&\le \frac{36n^3L^2C_r^2}{B_r} + \frac{9H^2}{4B_r} && \text{summands at most } n \text{ terms, } \textstyle\sum g_i < H
\end{aligned}$$

For the first term, we assume $B_0 \geq 17n^2 L \log(12n/\delta)\varepsilon^{-1}$, we have

$$\frac{36n^3L^2C_r^2}{B_r} \leq \frac{36n^3L^2C_1^2}{2B_0} \leq \frac{18n^4L^2\log(12n/\delta)}{B_0^2} \leq \frac{\varepsilon^2}{16\log(6/\delta)}.$$

For the second term, if $B_r \geq 36H^2\log(6/\delta)\varepsilon^{-2}$ then $\frac{9H^2}{4B_r} \leq \frac{\varepsilon^2}{16\log(6/\delta)}$. Hence, if this happens, termination triggers. Hence, the last round will satisfy $B_r \leq 72H^2\log(6/\delta)\varepsilon^{-2}$.

Note that in round r, the algorithm takes at most $B_r + n = 2^r B_0 + n$ (the n comes from ceilings). Hence, at round r, at most we have used $2B_r - B_0 + nr$ samples. From our algorithm termination, we have another $\sum_{i=1}^{r} \tilde{T}_i^{(r)} \leq 2B_r + n$ pulls. Summing both, we can see that at most $5B_r$ pulls will be used from our algorithm.

3.2.3 Correctness

Finally, we check the failure probability. Fortunately, this is not complicated. We see that by results in section 3.2.1,

$$\begin{aligned}
\mathbb{P}(|f(\hat{\mu}) - f(\mu)| \geq \varepsilon) &\leq \mathbb{P}(|f(\hat{\mu}) - f(\mu)| \geq \varepsilon|\xi) + \mathbb{P}(\not\xi) \\
&\leq \mathbb{P}(\nabla f(\hat{\mu})^{\intercal}|f(\hat{\mu}) - f(\mu)| \geq \varepsilon/2|\xi) + \mathbb{P}(\frac{L}{2}||\hat{\mu} - \mu||_2^2 \geq \varepsilon/2|\xi) + \delta/3 \\
&\leq \delta/3 + \delta/3 + \delta/3 = \delta.
\end{aligned}$$

Hence, we conclude our algorithm satisfies the correctness claim.

3.2.4 Gain of Adaptivity

We then discuss the gain we get from adaptivity in this algorithm. Let $H = ||\nabla f(\mu)||_1$. Observing the complexity from theorem 1, the first term will dominate the second term when $H^2 \geq Ln^2\varepsilon$. This naturally then occurs in a high-accuracy setting, as ε approaches 0.

Compare this with the uniform sampling baseline, if we sample each arm T/n times, then the budget is equal to approximately $n||\nabla f(\mu)||_2^2 \log(2/\delta)\varepsilon^{-2}$ pulls (this is by subbing $T_i = T/n$ in the earlier stopping condition). We then can define the reduction factor from the adaptive method as

$$gain(f;\mu) = \frac{n||\nabla f(\mu)||_2^2}{||\nabla f(\mu)||_1^2}.$$

This is bounded between 1 and n (by using Cauchy Schwarz [5]). The worst gain is when all partial derivatives are equal, which means our adaptavity cannot exploit any asymmetry. Conversely, if every entry is zero except for one arm i, we can say that f only depends on i, so we only need to sample i. This saves a factor of n samples from the naive uniform sampling.

One final thing to note is that this algorithm essentially enables us to get a 2ε interval on the true value of $f(\mu)$ with probability $1 - \delta$. Often, we use algorithm1 to get this interval in the later parts of this paper.

3.3 Number of Pulls per Arm

Another thing that we are interested to discuss is the amount of pulls on *each arm* in this algorithm. As discussed in the earlier part, the first r rounds will sample equally on each arms, and then sample based on the gradients to create the final sample (after the if condition is satisfied). To prepare on the setting of more than one function (which we will encounter later in the report), we need to modify some notations to differentiate which functions we are referring to. Hence, in Algorithm 1, when we want to learn f_k, we embed k to all indexes. In other words, we show the Algorithm 1 in context of multiple functions. This is identical to the previous algorithm, we just name the function so the parameters are clear.

Algorithm 2 Adaptive Function Approximation (For context of multiple functions)

```
        ▷ Identical to previous, this is for clarity        ▷ $L_j$ is the lipschitz constant for $f_j$.
Input: Arms $1, 2, \cdots, n$, known function $f_j$, target $\varepsilon$, error bound $\delta$
$B_{j,0} \leftarrow 17 L_j n^2 \varepsilon^{-1} \log 12n/\delta$        ▷ We need to take account that each function may end in different rounds.
for $r_j = 1, 2, \cdots, n$ do
    $B_{j,r_j} \leftarrow 2^r B_{j,0}$.
    Pull every arm $\lfloor B_{j,r_j}/n \rfloor$ times.
    Use every sample so far to get $\hat{\mu}^{(r)}$.
    Find bounds $\{\hat{g}_{i,j}^{(r,L)}, \hat{g_{i,j}}^{(r,U)}\}$.
    Construct $\alpha_{i,j}^r$ proportional to $\hat{g_{i,j}}^{(r,L)}$.
    Set $\tilde{T_{i,j}}^{(r)} \triangleq \lceil(\alpha_{i,j} + 1/n) B_{j,r}\rceil$.
    if $\sum_i \frac{\hat{g}_{i,j}^{(r,U)^2}}{\tilde{T_{i,j}}^{(r)}} \le \frac{\varepsilon^2}{16 \log(6/\delta)}$. then
        Pull arm $i$ $\tilde{T_{i,j}}^{(r)}$ times for $i = 1, 2, \cdots n$ to construct $\tilde{\mu}$. return $f(\tilde{\mu})$
    end if
end for
```

For simplicity, denote $S_{i,j,\varepsilon,\delta}$ as the number of arms needed to pull from arm i to learn f_j with ε accuracy and $1-\delta$ probability. If the choice of ε, δ is clear, we may omit these.

Note that here, $B_{j,r_j,i,\varepsilon,\delta}$ assumes that the error is ε and the probability is δ. We need to specify this because the value of B will depend on ε and δ, and these value will change depending on which ε and δ we are using. If the ε and δ are not present, we just assume that it is for plain ε, δ.

In the proof on the algorithm, note that the last round for function j, the value $B_{j,r_j,\varepsilon,\delta}$ satisfies $B_{j,r_j,\varepsilon,\delta} \le 72||\triangledown f_j(\mu)||_1^2 \log(6/\delta)\varepsilon^{-2}$. The value $S_{i,j,\varepsilon,\delta}$ is determined by the "pull everything equally" phase and "pull based on gradient" phase. The first case is easy to count, as it is at most $\frac{B_{j,r_j}}{n} + 1$ times each arm.

According to algorithm 1, the final pull for arm i in the final round (r_j) for function f_j needs $\lceil(\alpha_{i,j} + 1/n) B_{j,r_j,\varepsilon,\delta}\rceil$. Now, this $\alpha_{i,j}$ is proportional to $\hat{g}_{i,j,\varepsilon,\delta}^{(r,L)}$. Denote $g_{j,\varepsilon,\delta}^{(r,L)}$ as

$\sum_i \hat{g}_{i,j,\varepsilon,\delta}^{(r,L)}$ as the sum of these value over the arms. We see that $\tilde{T}_i^{(r)} = \lceil (\alpha_i + 1/n) B_{j,r_j,\varepsilon,\delta} \rceil \leq 2\alpha_i B_{j,r_j,\varepsilon,\delta} + 1 = 2\frac{\hat{g}_{i,j,\varepsilon,\delta}^{(r,L)}}{\hat{g}_{j,\varepsilon,\delta}^{(r,L)}} B_{j,r_j,\varepsilon,\delta} + 1$.

Combining, we conclude the following theorem.

Theorem 2. *We have*

$$S_{i,j,\varepsilon,\delta} \leq \left(2\frac{g_{i,\hat{j},\varepsilon,\delta}^{(r,L)}}{g_{j,\hat{\varepsilon},\delta}^{(r,L)}} + \frac{1}{n} \right) B_{j,r_j,\varepsilon,\delta} + 2 \leq \left(2\frac{g_{i,\hat{j},\varepsilon,\delta}^{(r,L)}}{g_{j,\hat{\varepsilon},\delta}^{(r,L)}} + \frac{1}{n} + 1 \right) B_{j,r_j,i,\varepsilon,\delta}.$$

We define S' and S'' to distinguish between equal pulls and adaptive pulls. In particular,

$$S_{i,j,\varepsilon,\delta} = S'_{i,j,\varepsilon,\delta} + S''_{i,j,\varepsilon,\delta},$$

$$S'_{i,j,\varepsilon,\delta} \leq \left(\frac{1}{n} + 1 \right) B_{j,r_j,\varepsilon,\delta},$$

$$S''_{i,j,\varepsilon,\delta} \leq \left(2\frac{g_{i,\hat{j},\varepsilon,\delta}^{(r,L)}}{g_{j,\hat{\varepsilon},\delta}^{(r,L)}} + \frac{1}{n} \right) B_{j,r_j,\varepsilon,\delta}.$$

In particular, for the same function f_i, the S' between each arms are equal. We essentially divide the pulls between the uniform and adaptive sampling.

Chapter 4

The Thresholding Algorithm

Here, we look at another related problem on [2]. Instead finding $\hat{\mu}$ such that $f(\hat{\mu})$ is near $f(\mu)$, we want to verify whether $f(\mu)$ is greater than a threshold τ. This is useful for decision tasks, like deciding whether to run a policy (if the f is some linear combination of factors, we want this value to be larger than some τ to classify as yes). The objective provides an adaptive ε, as in if $f(\mu)$ is far from τ, then we only need a few samples as we only interested in its size relative to τ. The algorithm here is similar to the first one. The difference is now we explicitly estimate $f(\hat{\mu}^{(r)})$ to check whether the gap $|f(\hat{\mu}^{(r)}) - \tau|$ is large enough relative to $|f(\hat{\mu}^{(r)}) - f(\mu)|$.

Algorithm 3 Thresholding Variant

```
Input: Arms $1, 2, \cdots, n$, known function $f$, threshold $\tau$, error bound $\delta$
for $r = 1, 2, \cdots, n$ do
    $B_r \leftarrow 2^r B_0$.
    Pull every arm $\lceil B_r/n \rceil$ times.
    Use every sample so far to get $\hat{\mu}^{(r)}$.
    Find bounds $\{\hat{g}_i^{(r,L)}, \hat{g}_i^{(r,U)}\}$.
    Construct $\alpha_i^r$ proportional to $\hat{g}_i^{(r,L)}$.
    Set $\tilde{T}_i^{(r)} \triangleq \lceil (\alpha_i + 1/n) B_r \rceil$.
    Pull arm $i$ $\tilde{T}_i^{(r)}$ times to create $\tilde{\mu}^{(r)}$.
    if $C_r^{(f)} \leq |f(\tilde{\mu}^{(r)}) - \tau|$ then
        Use $\tilde{\mu}^{(r)}$ as the final estimator. return $|\tilde{\mu}^{(r)} - \tau|$.
    end if
end for
```

For this, we define and utilize confidence intervals C_r^f for the function estimator $f(\tilde{\mu}^{(r)})$ such that $|f(\tilde{\mu}^{(r)}) - f(\mu) \leq C_r^f$ with high probability. Here,

$$\begin{aligned} C_r^f &= C_{r,1}^f + C_{r,2}^f \\ C_{r,1}^f &= \sqrt{2\log(24r^2/\delta)\sum_i \frac{\hat{g}_i^{(r,U)^2}}{\tilde{T}_i^{(r)}}} \\ C_{r,2}^f &= \frac{Ln\log(24nr^2\delta)}{\tilde{B}_r} \end{aligned}$$

Here, $C_{r,1}$ is the first order error, and $C_{r,2}$ is the second order error. Assume that the borderline cases never occur ($f(\mu) \neq \tau$). (The algorithm can be modified slightly to accomodate this case). The claim is as follows. ($\tilde{O}$ supresses the $\log\log$ terms on $poly(n, \delta, ||\nabla f(\mu)||_1, (f(\mu) - \tau)^{-1})$.

Theorem 3. *Algorithm 2 succeeds in predicting whether $f(\mu) > \tau$ with probability at least $1 - \delta$ and using at most*

$$\tilde{O}\left(\frac{H^2\log(1/\delta)}{(f(\mu)-\tau)^2} + \frac{n^2L\log n/\delta}{|f(\mu)-\tau|}\right)$$

pulls.

4.1 Proof of Theorem 2

Similar to the first proof, we also do this step by step

4.1.1 Algorithm Terminates Correctly (and on time)

As before, we define a good event ξ where the mean estimators $\{\mu_i^{(r)}\}$ stays in the confidence intervals. We define another good event $\hat{\xi}$ as the event that our estimates $f(\hat{\mu^{(r)}})$ falls in their intervals (the distance from $f(\mu)$ in each round is within C_r^f).

Lemma 4. *For Algorithm 2, we have $\mathbb{P}(\hat{\xi}|\xi) \geq 1 - 2\delta/3$*

Proof. Due to $L-$lipschitz gradient, we have

$$|f(\mu) - f(\hat{\mu})| \leq |\nabla f(\mu)^\intercal(\hat{\mu} - \mu)| + \frac{L}{2}||\hat{\mu} - \mu||_2^2.$$

Hence,

$$\begin{aligned} &\mathbb{P}(|f(\tilde{\mu}^{(r)}) - f(\mu)| \geq C_r^f|\xi) \leq \mathbb{P}(|\nabla f(\mu)^\intercal(\hat{\mu} - \mu)| + \frac{L}{2}||\hat{\mu} - \mu|| \geq C_r^f|\xi) \\ &\leq \mathbb{P}\left(|\nabla f(\mu)^\intercal(\hat{\mu} - \mu)| \geq \sqrt{2\log(24r^2/\delta)\sum_i \frac{g_i}{\tilde{T}_i^{(r)}}}\middle|\xi\right) + \mathbb{P}(\frac{L}{2}||\hat{\mu} - \mu|| \geq C_{r,2}^f) \\ &\leq 2\mathbb{P}\left(|\nabla f(\mu)^\intercal(\hat{\mu} - \mu)| \geq \sqrt{2\log(24r^2/\delta)\sum_i \frac{g_i}{\tilde{T}_i^{(r)}}}\middle|\xi\right) + 2\mathbb{P}(\frac{L}{2}||\hat{\mu} - \mu|| \geq C_{r,2}^f) \end{aligned}$$

The last line comes from the fact that because $\xi \geq 2/3$ (from the previous proof), we have for any event E,

$$\mathbb{P}(E|\xi) = \frac{\mathbb{P}(E) - \mathbb{P}(E|\neg\xi)\mathbb{P}(\neg\xi)}{\mathbb{P}(\xi)} \leq 2\mathbb{P}(E).$$

Now, we analyze the first term of the inequality. Because $\nabla f(\mu)^\intercal(\hat{\mu} - \mu)$ follows $N\left(0, \sum_i \frac{g_i^2}{\tilde{T}_i^{(r)}}\right)$, by Hoeffding we have that this quantity is at most $2\exp - \log(24r^2/\delta) \leq \frac{\delta}{12r^2}$. For the second term, we also use Hoeffding to see that it is also less than $\frac{\delta}{12r^2}$. Plugging these two shows that $\mathbb{P}(|f(\tilde{\mu}^{(r)}) - f(\mu)| \geq C_r^f|\xi) \leq \frac{\delta}{3r^2}$. Hence, $\mathbb{P}(\hat{\xi}|\xi)$ is bounded by $1 - \frac{1}{\delta}\sum_{i=1}^n \frac{1}{i^2} > \delta/3$. □

Now, if both ξ and $\hat{\xi}$ happen, then the estimator falls on the confidence intervals. We claim that this will make the algorithm correct. If otherwise, then for all r we have $f(\mu^{(r)}) - \tau \geq f(\mu) - \tau - C_f^r > -C_f^r$. However, termination condition gives us $f(\mu^{(r)}) - \tau \leq C_f^r$, contradiction. Hence,

$$\mathbb{P}(\text{Success}) \leq \mathbb{P}(\xi \wedge \hat{\xi}) = \mathbb{P}(\xi)\mathbb{P}(\hat{\xi}|\xi) \geq (1 - \delta/3)(1 - 2\delta/3) \geq 1 - \delta.$$

We have proven that the algorithm terminates and has the accuracy claim.

4.1.2 Number of Sample Bound

Intuitively, the number of queries should be not too large because the algorithm stops when $2C_r^f < |f(\mu) - \tau|$. This is because assuming both ξ and $\hat{\xi}$, $|f(\tilde{\mu}^{(r)} - \tau| \geq |f(\mu) - \tau| - |f(\tilde{\mu}^{(r)}) - f(\mu)| \geq C_r^f$. Hence, we bound on how many samples required to get $C_r^f < |f(\mu) - \tau|/2$ given ξ and $\hat{\xi}$.

Proof. Similar to last time,

$$\begin{aligned}
C_r^f &= \sqrt{2\log(24r^2/\delta)\sum_i \frac{\hat{g}_i^{(r,U)^2}}{\tilde{T}_i^{(r)}}} + C_{r,2}^f && \text{Formula of } C_r^f \\
&\leq \sqrt{2\log(24r^2/\delta)\sum_i \frac{(g_i + 2L\sqrt{C_n})^2}{\tilde{T}_i^{(r)}}} + \frac{Ln\log(24nr^2/\delta)}{\tilde{B}_r} && \text{Bound of } \hat{g}_i^{(r,U)^2} \\
&\leq \sqrt{\frac{2\log(24r^2/\delta)(36n^3L^2C_r^2 + 9H^2/4)}{B_r}} + \frac{Ln\log(24nr^2/\delta)}{\tilde{B}_r} && \text{Similar bound with 4.1.2} \\
&\leq \sqrt{\frac{9H^2\log(24r^2/\delta)}{2B_r}} + \sqrt{\frac{144n^3L^2\log^2(24r^2/\delta)}{B_r^2}} + \frac{Ln\log(24nr^2/\delta)}{B_r} && \sqrt{a+b} \leq \sqrt{a} + \sqrt{b}
\end{aligned}$$

Notice that when $B_r \geq \frac{18\log(24r^2/\delta)H^2}{(f(\mu)-\tau)^2} + \frac{24Ln^2\log 24nr^2/\delta}{|f(\mu)-\tau|}$, we have the quantity is less than $|f(\mu) - \tau|/2$ by noting that $1/sqrtx + y <= 1/2(1/sqrtx + 1/sqrty)$.

Hence, when a round r met these conditions, our algorithm will terminate. Note that $\log r = \log\log(B_r/n)$. So, our algorithm stops at

$$B_r = \tilde{O}\left(\frac{H^2\log(1/\delta)}{(f(\mu) - \tau)^2} + \frac{n^2L\log n/\delta}{|f(\mu) - \tau|}\right)$$

Because Algorithm 2 takes at most $3B_r + n$ arms per round, the total budget is at most $\sum_{i=1}^{r} 3B_i + n \leq 7B_r$ rounds. Hence, there are at most

$$\tilde{O}\left(\frac{H^2 \log(1/\delta)}{(f(\mu) - \tau)^2} + \frac{n^2 L \log n/\delta}{|f(\mu) - \tau|}\right)$$

pulls, as needed. $\square$

Chapter 5

Finding Maximum Point Evaluation

In this section, we will use the algorithms used in the last two sections to solve a related problem. We first discuss the problem setting. Here, we are given one (lipschitz) function f and multiple vectors $\mu_1, \mu_2, \cdots, \mu_l$. Note that all vectors are sampled iid, hence the information on one vector does not tell anything about other vectors . Our goal now is to find the maximum value of the evaluation of μ_i over f, i.e. finding

$$argmax_{i=1,2,\cdots,l} f(\mu_i).$$

A naive way to do this is to compare $f(\mu_1)$ and $f(\mu_2)$, and the winner will be compared to $f(\mu_3)$, and so on. However, this may have a bad performance: Suppose that $f(\mu_1), f(\mu_2)$ are very tightly closed, with $|f(\mu_1) - f(\mu_2)|$ being very near to 0. This will take a long time due to this small value (as intuitively, small gaps means that it is generally harder to decide). However, if $f(\mu_3)$ is much larger than both $f(\mu_1)$ and $f(\mu_2)$, then we should notice it a lot sooner and not waste time on f_1 and f_2 (as neither are the correct functions anyway).

The idea for this case is to invoke algorithm 1 many times for one arm to learn confidence intervals, but the main idea is that the previous invocation of the algorithm may be used for the future invocations (no need to relearn from scratch). For example, if we learn $f(\mu_1)$ with accuracy 1 (which means that we get $\hat{\mu}_1$ such that $|f(\hat{\mu}_1) - f(\mu_1)| < 1$, then learn $f(\mu_2)$ with accuracy 1, and finally learn $f(\mu_1)$ with accuracy 0.5, it is the equivalent as learning $f(\mu_1)$ with accuracy 0.5 then learning $f(\mu_2)$ with accuracy 1. We also add an additional condition for convenience, which is $|f(\mu_1)|, \cdots, |f(\mu_k)| \leq M$ for some $M > 0$. This means that $\Delta \leq 2M$.

The main idea here is to learn each function in parallel, and giving up on arms that are getting outclassed by another arm. For example, if the upper bound of $f(\mu_1)$ is smaller than the lower bound of $f(\mu_2)$ in a round, then we can safely assume that $f(\mu_1)$ is not the maximum.

Algorithm 4 Find Maximum

```
Input: Sets of arms $\{\mu_i\}_{i=1}^k$, known function $f$, threshold $\tau$, error bound $\delta$
candidates = $[n]$
$d = 1$
while |candidates| > 1 do
    for c in candidates do
        $x_c = Algorithm1(\mu_c, f, d, \frac{\delta}{\log(1/\Delta)k})$
        $L_c = x_c - d$ , $U_c = x_c + d$
    end for
    Remove all arms in $c$ such that $\exists d, U_c < L_d$.
    $d = d/2$
end while
return The remaining candidate.
```

We start by proving the correctness. This is done by using union bound [6].

Theorem 4. *Algorithm4 will produce the correct maximum arm with probability* $1-\delta$.

Proof. Denote $X_{i,j}$ be the event of failure when invoking algorithm1 on the i-th arm on the j-th iteration in the while loop. We have $j \le \log(\frac{1}{\Delta})$ as d halves each while loop and we need to reach Δ distance to confirm an arm. From our algorithm settings, $\mathbb{P}(X_{i,j}) \le \frac{\delta}{k\log(1/\Delta)}$. By union bound,

$$\mathbb{P}\left(\bigcup_{1\le i\le k, 1\le j\le \log(1/\Delta)} X_{i,j}\right) \le \sum_{1\le i\le n, 1\le j\le \log(1/\Delta)} \mathbb{P}(X_{i,j}) \le \delta.$$

□

Next, we analyze the arm pulls needed.

Theorem 5. *The algorithm will need at most*

$$O\left(\frac{kF\log\frac{\log(1/\Delta)k}{\delta}}{\Delta^2} + \frac{kn^2L\log n\frac{\log(1/\Delta)k}{\delta}}{\Delta}\right)$$

pulls, where $F = \max_{i\in[n]} ||\nabla f(\mu)||_1^2$.

Proof. Note that for a candidate, the last time we invoke algorithm 1 on that candidate is when $\varepsilon < \Delta$. Due to our way of setting epsilon (dividing by 2), we must have $\varepsilon \ge \Delta/2$. Also, we only need to find the time needed to reach this epsilon, which is

$$O\left(\frac{||\nabla f(\mu_i)||_1^2\log\frac{\log(1/\Delta)k}{\delta}}{\Delta^2} + \frac{kn^2L\log n\frac{\log(1/\Delta)k}{\delta}}{\Delta}\right).$$

(We can substitute the ε with Δ). Then, summing through all k candidates (in the worst case), and letting $F = \max_{i\in[n]} ||\nabla f(\mu)||_1^2$ for cleaner notation, we have at most

$$O\left(\frac{kF\log\frac{\log(1/\Delta)k}{\delta}}{\Delta^2} + \frac{kn^2L\log n\frac{\log(1/\Delta)k}{\delta}}{\Delta}\right).$$

as desired. □

We then compare the performance of the algorithm compared with the naive algorithm where we compare two points pair-wise in a tournament setting (eliminate the smaller one each round). From the previous chapter, this can be done on $\tilde{O}\left(\frac{H^2\log(1/\delta)}{(g(\mu))^2} + \frac{n^2L\log n/\delta}{|g(\mu)|}\right)$ pulls and succeeds with probability $1-\delta$. To help, define

$$\Delta^* = \min_{i,j\in[k],i\neq j} f(\mu_i) - f(\mu_j).$$

By definition, $\Delta^* \leq \Delta$. Now, if we do the comparison tournament style, the probability of each round to fail must be kept at δ/k by union bound (we do $k-1$ rounds. Hence, one comparison will have $\tilde{O}\left(\frac{H^2\log(k/\delta)}{\Delta^{*2}} + \frac{n^2L\log k/\delta}{|\Delta^*|}\right)$. Hence, summing through $k-1$ comparisons, we have $\tilde{O}\left(\frac{kF^2\log(k/\delta)}{\Delta^{*2}} + \frac{kn^2L\log k/\delta}{|\Delta^*|}\right)$. Although they are similar, the key is the fact that $\Delta^* \leq \Delta$, and it is easy to construct cases when the naive algorithm is punished hard (to a unbounded factor, as we take the arms arbitrarily). For example, we easily can make $\Delta^* = 10^{-10}$ and $\Delta = 1$ and our algorithm will be much better than the naive algorithm.

To be more precise, we will write the time complexity of our new algorithm in terms of relative distance of each arm. Now, define Δ_i as the distance of the best arm with the $i-$th best arm (hence, $\Delta_2 = \Delta$). For notation consistency, $\Delta_1 = 0$. Hence, for the $i-$th best arm, using the same logic, needs $\frac{1}{\log\Delta_i}$ pulls to be eliminated. This is the sharper (but less nicer) form of the total number of pulls.

Theorem 6. *The total pulls needed is equal to*

$$O\left(k\sum_i \frac{||\nabla f(\mu_i)||_1^2\log\frac{\log(1/\Delta_i)k}{\delta}}{\Delta_i^2} + kn^2L\sum_i\frac{\frac{\log(1/\Delta_i)k}{\delta}}{\Delta_i}\right)$$

Proof. This is an immediate extension leveraging the fact that we add the runtime for each arm (now with the distance depending on their distance with the first arm). Note that the proof of accuracy still also holds. □

Chapter 6

Finding Maximum Function in a Point

Now, we consider another extension of the problem, in a similar spirit to the previous chapter. Now, we have a single point, but multiple known functions. As before, we want to find the maximum value of the evaluation of f_i over μ, i.e. finding

$$argmax_{i=1,2,\cdots,l} f_i(\mu).$$

First, we solve the case when there are only two functions. Suppose f and g are functions known and has $L-$Lipschitz gradients. Suppose also that μ is an unknown vector. Similar to the previous configurations, we can ask an oracle for noisy samples for μ_i We would like to create an algorithm that determines whether $f(\mu) > g(\mu)$.

The most obvious approach is to define $h = f - g$, and run algorithm 2 with $\tau = 0$. This is possible because of the following lemma.

Lemma 5. *f and g has L-lipschitz gradients, then h = f − g is also has 2L-lipschitz gradients*

Proof. We have $|f(x) - f(y)| \leq L||x - y||$ and $|g(x) - g(y)| \leq L||x - y||$. Then,

$$|h(x) - h(y)| = |(f(x) - f(y)) - (g(x) - g(y))| \leq |f(x) - f(y)| + |g(x) - g(y)| \leq 2L|x - y|.$$

$\square$

In particular, by theorem 3, the time complexity is $\tilde{O}\left(\frac{H^2 \log(1/\delta)}{(h(\mu))^2} + \frac{n^2 L \log n/\delta}{|h(\mu)|}\right)$, where $H = ||\nabla h(\mu)||_1 = ||\nabla f(\mu) - \nabla g(\mu)||_1$.

A more interesting question is to generalize to the k L-lipschitz function $f_1, f_2, \cdots, f_k$. The direct attempt is to do it tournament style, where we create compare f_1 and f_2, keep the winner, and then compare with f_3 and so on. We do this process $k - 1$ times. However, this may have a bad performance: Suppose that f_1, f_2 are very close in evaluation to μ, with $|f_1(\mu) - f_2(\mu)|$ being very near to 0. This will cause Algorithm 2 to take a long time. However, if f_3 is much larger than both f_1 and f_2 in μ, then we should notice it a lot sooner and do not waste time on f_1 and f_2.

The difference of this problem and the previous version is that learning the value of μ with respect to f_1 has an influence for (known functions) $f_2, f_3, \cdots, f_k$, as opposed to the earlier version, as $\mu_1, \mu_2, \cdots$ are independent. Hence, in theory, while we can adapt the fast algorithm in the previous chapter, it will leave some room for optimization as we disregarded the additional information of μ by learning with respect one function. We just learn to estimate $|f_i(\hat{\mu}) - f_i(\mu)|$ for each f_i. This will give a bound of $f_i(\mu)$ that can be used. Another important thing is that now, each query for an arm for one function can provide information about another function (as the query vector μ is the same). Hence, by right, the amount of queries is smaller (in principle, the complexity is changed from $\sum$ to max).

Now, we discuss on the algorithm that does adaptive sampling with a similar spirit to the previous chapter. Suppose that L_i is the lipschitz constant for f_i and we keep the assumption that there exists a positive number M such that $\forall i, f_i(\mu) < M$. Also, now we denote ω_i as the absolute difference between the values of the best function and the i-th function, with $\omega_1 = 0$. Denote further $\omega = \omega_2$ to simplify notations. Another remark is that in the algorithm below, we reuse arm pulls implicitly, meaning that if we have done 3 pulls in the first candidate, and needs 2 pulls on the second candidate, we do not do any pulls. Finally, we assume that ω is known (or at least, there exists a minimum bound $w > 0$ of ω). If ω is unknown, we can use w as our ω.

Algorithm 5 Find Maximum Function

```
Input: Sets of functions $\{f_i\}_{i=1}^k$, with lipschitz constants $L_i$, unknown vector $\mu$, threshold $\varepsilon$,
error bound $\delta$
candidates = $[k]$
$d = 1$
while |candidates| > 1 do
    for c in candidates do
        $x_c = Algorithm1(\mu, f_c, d, \frac{\delta}{\log(1/\omega))}$ ▷ Reuse pulls for same arm in different functions
        $L_c = x_c - d$ , $U_c = x_c + d$
    end for
    Remove all functions in candidates such that $\exists d, U_c < L_d$.
    $d = d/2$
end while
return The sole remaining candidate.
```

Firstly, lets check that this algorithm meets the correctness claim. Note that the algorithm always stops trivially (assuming no ties). The difference here is due to "reusability", we do not need to repeat for each function, hence the relaxed bound.

Theorem 7. *The algorithm will produce the correct maximum arm with probability* $1 - \delta$.

Proof. Denote X_j be the event of failure when invoking Algorithm1 on the j-th iteration in the while loop. We have $j \leq \log(1/\omega)$ as d halves each while loop and we need to reach ω

distance to confirm a function as the maximum. From our algorithm settings, $\mathbb{P}(X_j) \leq \frac{\delta}{\log(1/\omega)}$. By union bound,

$$\mathbb{P}\left(\bigcup_{1 \leq j \leq \log(1/\omega)} X_j\right) \leq \delta.$$

□

Next, we discuss the total number of pulls. Note that we will count for each arm. To pick the best f, we need $\varepsilon < \omega$ to separate it from others. hence last round has $\varepsilon \geq \omega/2$. By Theorem 2, for the j-th function, the i-th arm needs no more than $S_{i,j,\omega_j,\frac{\delta}{\log(1/\omega_j)}}$ pulls. This means that arm i needs at most $\max_j S_{i,j,\omega_j,\frac{\delta}{\log(1/\omega_j)}}$, which means the total of arm pulls is at most

$$\sum_i \max_j (S_{i,j,\omega_j,\frac{\delta}{\log(1/\omega_j)}}).$$

Let's briefly talk about the sampling complexity compared to the naive algorithm. Similar to before, denote $\omega_* = \min_{i \neq j} |f_i(\mu) - f_j(\mu)|$. If we have l functions, we need to invoke the thresholding algorithm (as in the 2-function case) $l-1$ times. For union bound, we need a success probability of δ/l (because we do this l times). We need to check for each arm. Note that each arm i needed at most $S_{i,j,\omega^*,\frac{\delta}{\log(1/\omega^*)}}$ pulls. Summing each arm in the similar manner (note that even in a naive setting, we still save arms), the total arm pull is

$$\sum_i \max_j \left(S_{i,j,\omega^*,\frac{\delta}{\log(1/\omega^*)}}\right).$$

We can see that similar to the multiple μ situation, we have $\omega > \omega^*$ as the difference maker here.

Chapter 7

Numerical Experiments

To show the application of the algorithms in this report, a series of experiments are made for practical showing. We will replicate some of the experiments done on the original paper [2], and also adding some of our own for the new algorithms. The implementations can be found in the code attached on the submission.

7.1 Matrix-Vector Product Norm Error

Here, given a known matrix $A \in \mathbb{R}^{n \times d}$ and a vector $x \in \mathbb{R}^{d \times 1}$, our goal is to spend as few computational errors as possible to estimate $||Ax||_2^2$. For simplicity, denote $\mu = Ax$. The observation is that the value $dA_{i,j}x_j$ where j is chosen uniformly from 1 to d is an unbiased estimator for μ_i. Hence, we can think the calculation of $||Ax||_2^2$ as calculating $||\mu||_2^2$ where the entries of μ can only be sampled by a noisy oracle.

We will show first the performance of the algorithm compared to the naive uniform setting (given T arm pulls, sample all arm T/n times and then find the predicted $\hat{\mu}$). Due to the nature of the problem, we have several modifications for the algorithm, also noted in [2]:

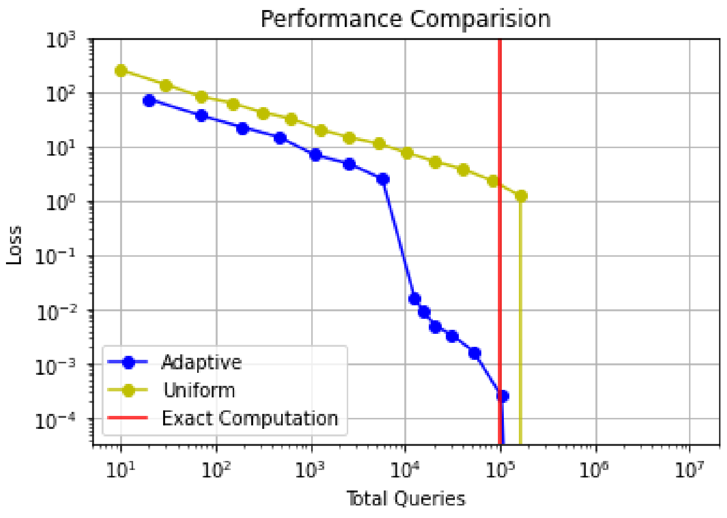
1. Because we know the sample space $dA_{i,j}x_j$ is finite, any picked j should not be considered anymore. To mimic this, we first permute the columns of A and entries of x randomly, and simulate the uniform sampling by iterating the entries from left to right. We **do not** add noise to the sampling, as the "randomness" of μ_i essentially comes from the sample space.

2. If we already sampled the entire sample space $dA_{i,j}x_j$, we then already know μ_i with zero error (simply take the average). This means that we do not need to waste pulls in the row anymore.

3. We multiply a constant of 10 in the adaptive sampling T_i to have a better performance and use more aggressive bounds. Also, we redefine $C_r = \sqrt{\log(1/\delta)/\tilde{B}_r}$.

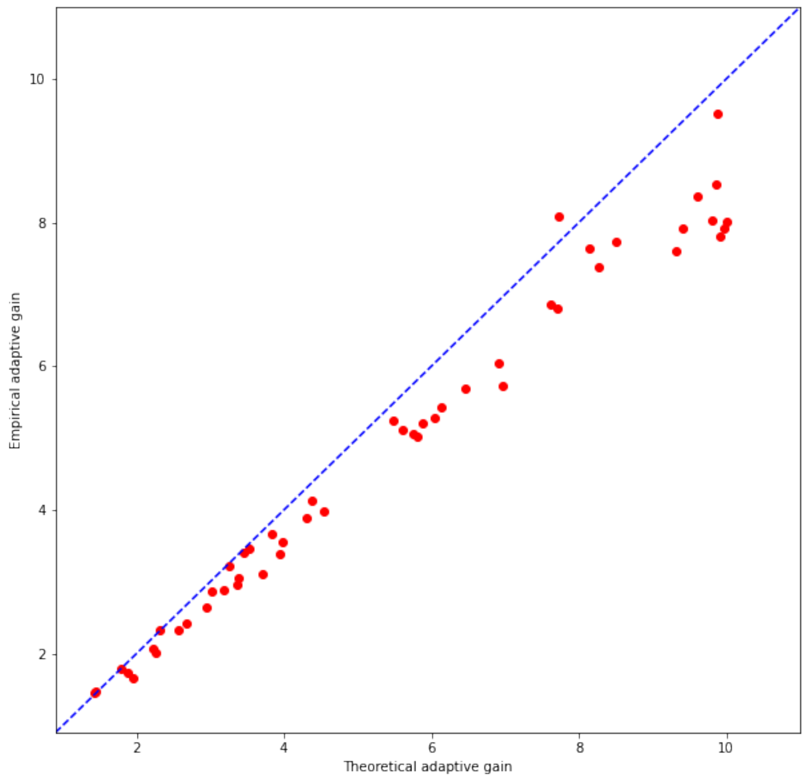
We also discuss briefly how we get the A and x data. First, we create μ by sampling from $Pareto(0.5)$ distribution and x from $Uniform(0.5, 1)$. We then normalize μ so the largest entry is 20. Then, write $A = \mu x^\intercal/||x||_2^2 + Z$ where Z is sampled from a normal distribution with variance $1/\delta^2$. Finally, set $A \leftarrow A - (Ax - \mu)\mathbf{O}^\intercal/||x||_1$ to ensure $Ax = \mu$, where $\mathbf{O}$ denotes the vector of all ones with the correct size.

Now, the first experiment is shown below where we try to predict $||Ax||_2^2$ with $n = 10, d = 10000, \varepsilon = 10^{-4}$ and $\delta = 10^{-2}$. We run 100 rounds of the algorithm and plot the average losses per round. No failures (experiments ending with an invalid $\hat{\mu}$) are recorded In this graph, the point for round r in Algorithm1 is taken after taking samples according to $\tilde{T}_r$. These samples are then **not** used for the subsequent samples. This is done to show the gain versus the uniform algorithm. The exact computation is just to detect every term and add them normally, which takes nd samples (which is 10^5 in our graph).



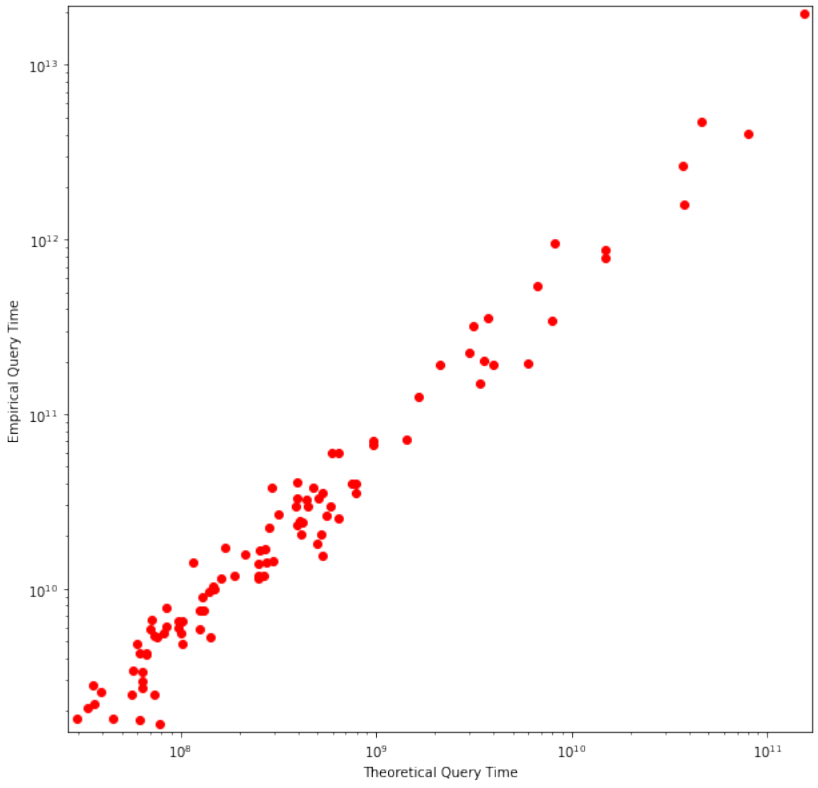
As we can see, the gains become noticeable when we spend around 10^4 queries. This is because our adaptive algorithm focuses on the largest entry (hence contributes to the largest loss count) and allows it to get near exact value way faster than the uniform algorithm.

We then check the gain of adaptivity for this test. To see this, we constructed 50 test cases of μ with $n = 10$ and $d = 10000$, and for each test case, run algorithm1 and uniform 100 times to find the total number of queries before termination. (Here, we ignore point 1 and 2 in the previous assumption to find the true stopping time). We then compare the expected gain from the formula, and the actual gain by finding the ratio of the mean of stopping time for uniform and mean of stopping time for Algorithm1. To add variability and show the trend better, we randomly change the budget per round to $n(2 - c/200)^r$, where c is uniform from 0 to 99.



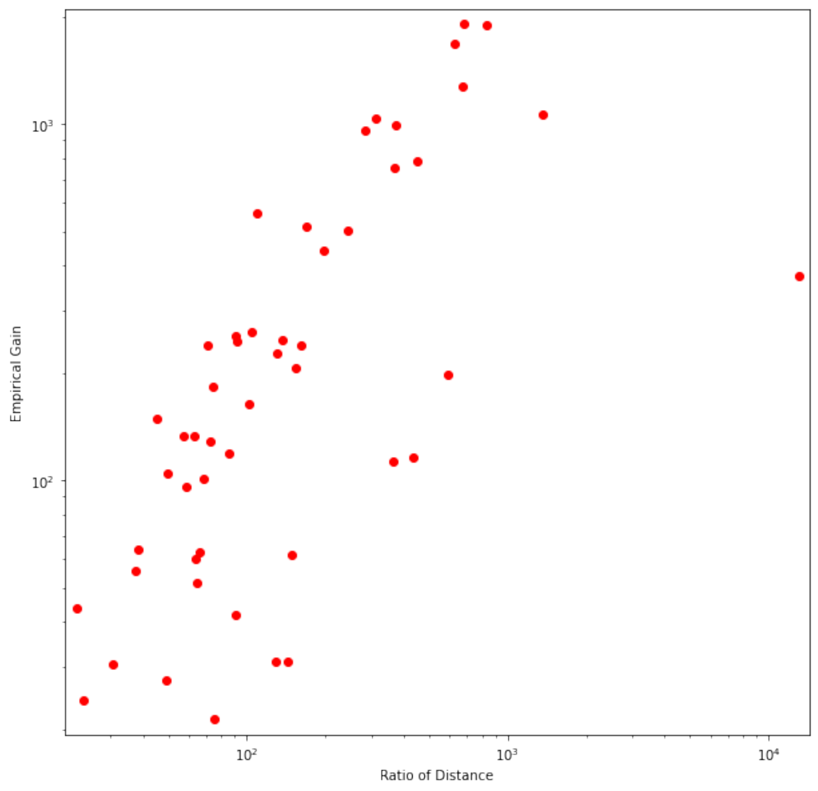
One red dot represents a test case, and ideally, all points are on the dashed line $y = x$. However, by using a simple linear regression model, we find that the slope of these points are around 0.82. This is actually explainable by the choice of 10 earlier in the algorithm. Indeed, each round, the adaptive algorithm makes at most $2B_r$ uniform pulls, and $10B_r$ adaptive pulls, so the optimal sampling frequencies are used on $\frac{10}{10+2} = 0.83$ pulls, which is close with our empirical analysis.

Next, we show the performance of the adaptive thresholding algorithm. Similar to the first two experiments, we run the experiment to predict $||Ax||_2^2$, but we supply a boundary of $||\mu||_2 + Uniform(-1,1) \times 0.01$. We set $n = 100, d = 10000$ here and run 100 trials. Our implementation shows 100 percent success (returns the correct thresholding decision) rate and has a consistent run time with the theoretical bound shown earlier, as shown in this linear relationship.



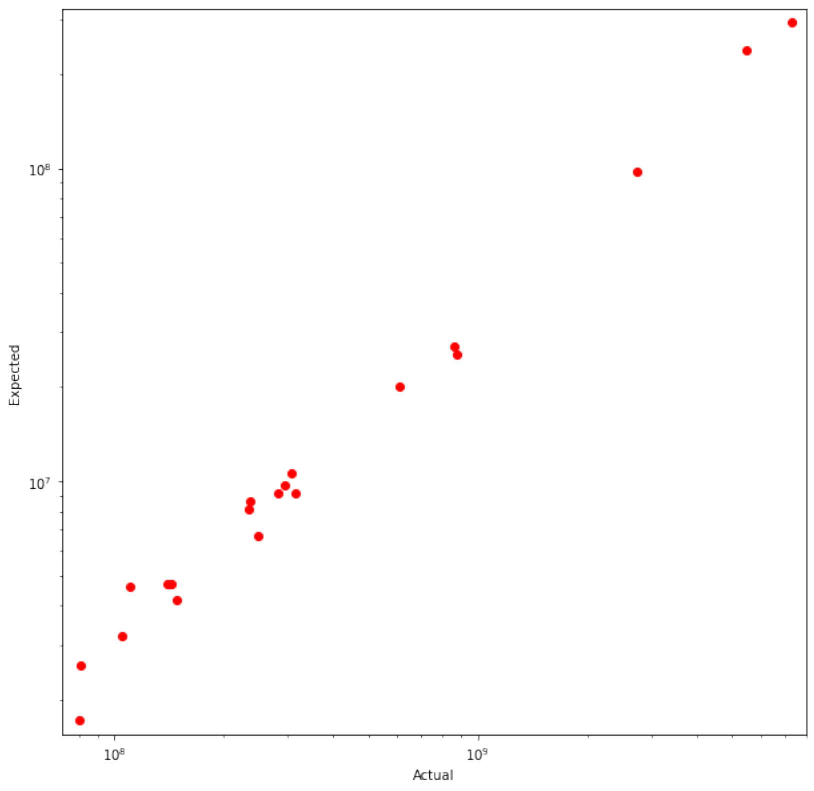
7.2 Finding Maximum Function in a Point

We now turn our attention to experiment our algorithm to find the maximum function given μ. Here, we do a simple test to demonstrate the effect of bad orderings can punish naive algorithm. To do this, we create a family of k functions $f(\mu) = x^{\intercal}\mu/n^4$, where $x \in \mathbb{R}^{n\times 1}$ taken from $N(n/2, 1)$. To give raise of a condition where there is a clear winner, we multiply one of them by n. We then present all the functions on the array for both the adaptive and naive algorithm to run. We set $n = 5, k = 5$ and do 50 trials. The results are plotted below, where the x-axis shows the ratio $\frac{\omega}{\omega^*}$ and the y-axis show the gain from adaptivity. Recall that ω is the distance between the best and the second best functions and ω^* denotes the smallest distance between two functions on μ. Hence, when this ratio gets larger, we should expect an increase in gain. The plot below confirms our theoretical hypothesis.



7.3 Finding the Maximum Point

Finally, we also consider the experiment on our algorithm to find the best μ given f. We construct a function similar to the chapter above, and generate multiple $\mu \in \mathbb{R}^{n\times 1}$ taken from $N(n/2, 1)$. Then, we do another 50 tests where $n = 5$ with 5 μ candidate. We observe a linear relationship between empirical observation and theoretical bound (we overestimate the theoretical bound, but the linearity is still observed in the picture).


10^8
10^7
Expected
10^8
10^9
Actual

References

[1] Vivek Bagaria et al. *Bandit-Based Monte Carlo Optimization for Nearest Neighbors*. 2018. DOI: 10.48550/ARXIV.1805.08321. URL: https://arxiv.org/abs/1805.08321.

[2] Tavor Z. Baharav et al. *Approximate Function Evaluation via Multi-Armed Bandits*. 2022. DOI: 10.48550/ARXIV.2203.10124. URL: https://arxiv.org/abs/2203.10124.

[3] Sébastien Bubeck, Nicolò Cesa-Bianchi, and Gábor Lugosi. *Bandits with heavy tail*. 2012. DOI: 10.48550/ARXIV.1209.1727. URL: https://arxiv.org/abs/1209.1727.

[4] Govinda M. Kamath, Tavor Z. Baharav, and Ilan Shomorony. *Adaptive Learning of Rank-One Models for Efficient Pairwise Sequence Alignment*. 2020. DOI: 10.48550/ARXIV.2011.04832. URL: https://arxiv.org/abs/2011.04832.

[5] Radmila Bulajich Manfrino, José Antonio Gómez Ortega, and Rogelio Valdez Delgado. *Inequalities: a mathematical olympiad approach*. Springer Science & Business Media, 2009.

[6] *Probability oer*. URL: https://probability.oer.math.uconn.edu/.

[7] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits*. 2019. DOI: 10.48550/ARXIV.1904.07272. URL: https://arxiv.org/abs/1904.07272.

Appendix A

Notations

Below are the notations used over the report

- $f_1, f_2, \cdots, f_k$ are known Lipschitz functions. These functions have Lipschitz constants $L_1, L_2, \cdots, L_k$.
- We also define $\nabla_i f$ as the derivative with respect to the i-th variable of f. We also write these as g_i to simplify notation.
- $\mu_1, \mu_2, \cdots, \mu_l$ are the unknown vectors, which we call points. We further assume that $\mu_i \in \mathbb{R}^n$. We call the j-th entry of μ_i as the j-th arm of μ_i.
- For a point μ_i, we denote $\hat{\mu}_i$ as our estimate of μ_i.
- We ignore the indexing where there is only one point/function of interest.
- In most of the algorithms, $\mathbb{E}[T]$ is the expected number of total pulls δ is the probability of success, and ε is the confidence bound.
- Z denotes a normal distribution with mean 0 and variance 1. The subscripts mostly meant that the Z are reset each pull.
- One query or one an arm pull to a point μ is defined as choosing an arm i, and get back $\mu + Z_i$.
- The variable r denotes the number of rounds, usually starts with 1 and ends in r. Also, in multiple points case, denote r_j as the final round of j-th point.
- Following the previous definition, the budget of a round $B_{j,k,\varepsilon,\delta}$ is the number of arm pulls for function j in round k to learn with ε, δ parameters in Algorithm 1. The ε, δ might be ignored if the context is clear.
- Furthermore, the superscript $\cdot^{(r)}$ means in context of r-th round.
- Define the confidence interval for round r as $C_{r,\varepsilon,\delta} = \sqrt{\frac{2\log(12nr^2/\delta)}{\sum_{i=1}^{r} \lceil B_{i,\varepsilon,\delta}/n \rceil}}$

- We can also write down the lower and upper bound for gradients as follows.

$$\hat{g_{i,\varepsilon,\delta}}^{(r,L)} \triangleq \min_{y:||y-\hat{\mu}_r||_\infty \le C_r} \nabla_i f(y),$$

$$\hat{g_{i,\varepsilon,\delta}}^{(r,U)} \triangleq \max_{y:||y-\hat{\mu}_r||_\infty \le C_r} \nabla_i f(y).$$

  We might add j in the case of multiple function (hence, $\hat{g}_{i,1,\varepsilon,\delta}$ refers to the first function). The ε, δ might be ignored.

- Denote $\tilde{T_i}^{(r)}$ be the allocation of pulls in round r for arm i. In multiple points case, $\tilde{T_{i,j}}^{(r)}$ be the allocation of pulls in round r for arm i when learning point j.

- Denote $H = \sum_i g_i$, i.e. sum of all partial derivatives of f.

- ξ denotes the good event, usually in correctness analysis.

- $gain(f;\mu)$ is defined as the overall speedup by adaptivity to learn $f(\mu)$ compared to uniform approach.

- Define C_r^f as the confidence interval for the round r when learning f. We also divide into $C_{r,1}^f$ for the first order error and $C_{r,2}^f$ for the second order error. Moreover,

$$C_r^f = C_{r,1}^f + C_{r,2}^f$$

$$C_{r,1}^f = \sqrt{2\log(24r^2/\delta)\sum_i \frac{\hat{g_i}^{(r,U)^2}}{\tilde{T_i}^{(r)}}}$$

$$C_{r,2}^f = \frac{Ln\log(24nr^2\delta)}{\tilde{B_r}}$$

- Denote $S_{i,j,\varepsilon,\delta}$ as the number of arms needed to pull from arm i to learn f_j with ε accuracy and $1-\delta$ probability.

- We define S' and S'' to distinguish between equal pulls and adaptive pulls. In particular,

$$S_{i,j,\varepsilon,\delta} = S'_{i,j,\varepsilon,\delta} + S''_{i,j,\varepsilon,\delta},$$

$$S'_{i,j,\varepsilon,\delta} \le \left(\frac{1}{n} + 1\right) B_{j,r_j},$$

$$S''_{i,j,\varepsilon,\delta} \le \left(2\frac{\hat{g_{i,j}}^{(r,L)}}{\hat{g_j}^{(r,L)}} + \frac{1}{n}\right) B_{j,r_j}.$$

- We define Δ_i as the distance of the best arm with the $i-$th best arm (hence, $\Delta_2 = \Delta$). For example, if $f(\mu_1) = 3, f(\mu_2) = 2$ and $f(\mu_3) = 5$, then $\Delta = 2$ and $\Delta_3 = 3$.

- Similarly, we define ω_i as the distance of the best function evaluated at μ with the $i-$th best function (hence, $\omega = \omega$). For example, if $f_1(\mu) = 3, f_2(\mu) = 2$ and $f_3(\mu) = 5$, then $\omega = 2$ and $\omega_3 = 3$.