Homework 3

CSE 599i: Interactive Learning
Instructor: Kevin Jamieson
Due 11:59 PM on March 13, 2020 (late homework not accepted)

Mirror Descent

1.1 Problem 26.4 of [SzepesvariLattimore].

Follow the Regularized Leader

2.1 Problem 28.4 of [SzepesvariLattimore]. For part b, implement both FTRL and mirror descent, state the explicit update rules for both algorithms, and comment on when they are different.

2.2 Problem 28.15 of [SzepesvariLattimore], excluding part h (just read that part).

Contextual Bandits

3.1 Problem 18.8 of [SzepesvariLattimore].

Experiments

4.1 Implement the algorithm from 28.15 (FTRL), EXP3, Follow-the-Leader, UCB, and Thompson Sampling (you can use implementations of UCB and Thompson Sampling from Homework 1 as a starting point). The updates for Follow-The-Leader are given by:

$$a_{t+1} = \arg\min_{a \in \Delta} \sum_{s=1}^{t} \langle a, \widehat{\mu}_s \rangle$$

where $\widehat{\mu}_s = \mathbf{e}_{I_s} X_s$ and $X_s \sim P_{I_s}$ vector at time s. Note that we're looking at *losses* so **lower is better**–modify Thompson sampling and UCB appropriately. Include plots and comment on the following:

- Run all algorithms on a standard stochastic bandit with $P_i = \text{Bernoulli}(\mu_i)$ in $\{0,1\}$, $n = 10$ arms, $\mu_i = 1/4$ for $i = 1$, $\mu_i = 3/4$ for $i \neq 1$. Use time horizon $T = 10000$. Note: if rewards are in $\{0,1\}$ how should you change the confidence bound of UCB, and variance of TS?
- Run all algorithms on a standard stochastic bandit with $P_i = \text{Bernoulli}(\mu_i)$ in $\{0,1\}$, $n = 10$ arms, $\mu_i = 1/4 + \frac{1}{2}\sqrt{(i-1)/(n-1)}$. Use time horizon $T = 10000$.
- With $T = 1000$, $n = 10$, find a set of losses such that UCB suffers roughly linear regret while EXP3 and FTRL both suffer sub-linear regret. The losses should be chosen in an oblivious fashion—they may rely on knowledge of the algorithms but cannot adapt to actions played by the algorithm (i.e. the losses should be chosen in advance).
- With $T = 1000$, $n = 10$, find an example where FTL suffers linear regret while FTRL suffers sub-linear regret. As above the losses should be chosen in an oblivious fashion.

4.2 Implement the EXP3(γ) algorithm discussed in class for linear bandits with a finite set of arms. Compare the performance of this algorithm to that of the LinUCB, Thompson Sampling, and Action Elimination algorithms from Homework 2 on a stochastic linear bandit. Fix $\theta^* = \mathbf{e}_1$ and draw arms from a Gaussian distribution. Recall that the EXP3(γ) algorithm is as follows:

Input: T, n arms, $\eta > 0$, $\gamma \in [0,1]$, exploration distribution λ
Init: $p_1 = (1/n, \ldots, 1/n)$, Adversary chooses $\{y_t\}_{t=1}^T \in [-h,h]^n$
for $t = 1, 2, ..., T$:

- Player draws $I_t \sim (1-\gamma)p_t + \gamma\lambda$
- Player suffers loss y_{t,I_t}
- Player computes $\hat{y}_{t,i}$ satisfying $\mathbb{E}[\hat{y}_{t,i}|p_t] = y_{t,i}$
- Update $\tilde{p}_{t+1,i} = \exp(-\eta \sum_{s=1}^{t} \hat{y}_{s,i})$, $p_{t+1,i} = \tilde{p}_{t+1,i}/\|\tilde{p}_{t+1}\|_1$

Implement this algorithm with both the original action set, and the action set augmented to map stochastic bandits to the adversarial setting (see Section 29.2 of [SzepesvariLattimore]). If $\mathcal{X}$ is our original action set and $x \in \mathcal{X}$ an arm, the augmented action set is defined as:

$$\mathcal{X}_{aug} = \{[x, 1] \ : \ x \in \mathcal{X}\}$$

Note that the theoretical regret bounds we proved for EXP3(γ) only hold in the stochastic setting when we use the augmented action set.

Experiment with the choices of d and n. Can you find values of d and n for which EXP3(γ) with augmented actions significantly outperforms LinUCB? Provide plots comparing the performance of each of these algorithms in settings of n and d where:

- LinUCB outperforms Action Elimination and EXP3(γ) with augmented actions.
- EXP3(γ) with augmented actions outperforms both, and LinUCB outperforms Action Elimination.
- EXP3(γ) with augmented actions outperforms both, and Action Elimination outperforms LinUCB

(Hint: how do the regret bounds we proved for each of these algorithms scale with d and n? You may need to use a different learning rate for EXP3(γ) than the theory motivates to attain this performance. All of these objective should be attainable with values of d and n below 200.). Also note that while we proved the correctness of EXP3(γ) for rewards in $[-1, 1]$ is is straightforward to show that it holds for Gaussian as well.

4.3 In this exercise we will implement several contextual bandit algorithms to perform classification with bandit feedback. In most practical settings where contextual bandits are used, we do not have access to the rewards for each action (we only know the reward of the action taken – bandit feedback). But to benchmark a contextual bandit algorithm we need to know the rewards of every action so we can evaluate how the algorithm would have performed if it took some other sequence of actions. To simulate such a process we can use a multi-class classification dataset in the following way: each feature vector is the context, each "arm" is recommending a particular class label, and the reward of an arm is 1 if the correct class was predicted and 0 otherwise. This way, while the bandit algorithm only knows whether they get it right or not, as an evaluator we know what reward the algorithm would have received if they had taken any arm. This is, of course, not something we would do if we actually cared about solving the classification task, since in that case we should incorporate the knowledge of the reward obtained for each action. We are only using this classification task as a way to benchmark our contextual bandit algorithm.

We will run our classification task on the MNIST dataset[1]. The MNIST dataset contains 28x28 images of handwritten digits from 0-9. Download this dataset and use the python-mnist library[2] to load it into Python. Rather than using the full images, you may run PCA on the data to come up with a lower dimensional representation of each image. You will have to experiment with what dimension, d, to use.

We will reduce the classification task to regression—given a context, predict the reward you will receive for playing each action, and choose the action with the highest predicted reward. Let the d dimensional representation of the tth image in the dataset, x_t, be our "context", and for each $a \in \{0, 1, \ldots, 9\}$ let $\phi_a \in \mathbb{R}^d$. Then $x_t^\top \phi_a$ is the predicted value of the "reward" for context x_t when action a is played. Here our true reward $r(x_t, A_t)$ will be equal to 1 if a is the correct label for x_t and 0 otherwise. As noted above, the player only observes the reward for the action they chose—unlike the standard supervised learning setting, they do not observe what the true label is when they guess incorrectly.

Implement the Explore-Then-Commit, LinUCB, Follow-The-Leader, and Thompson Sampling algorithms for this problem. Let:

$$\hat{\phi}_{a,T} = \arg\min_{\phi \in \mathbb{R}^d} \sum_{t=1}^{T} \mathbb{I}\{A_t == a\}(\phi^\top x_t - r(x_t, A_t))^2$$

In this setting, the algorithms work as follows:

[1] `http://yann.lecun.com/exdb/mnist/`

[2] `https://pypi.org/project/python-mnist/`

- Explore-Then-Commit algorithm uniformly samples actions for the first T steps, then computes $\hat{\phi}_{a,T}$ as above, and then for the remaining points in the dataset, it chooses the action A_t such that $\hat{\phi}_{A_t,T}^\top x_t$ is maximized.
- For LinUCB, the algorithm is essentially the standard LinUCB algorithm but now it forms confidence sets, $\mathcal{C}_{T,a}$ for each $\hat{\phi}_{a,T}$. At each iteration, it determines which action corresponds to the highest UCB, that is, it chooses action A_t where:

$$A_t = \underset{a \in \{0,1,\ldots,9\}}{\arg\max} \max_{\phi \in \mathcal{C}_{T,a}} \phi^\top x_t$$

- Follow-The-Leader chooses actions according to the following rule:

$$A_t = \underset{a \in \{0,1,\ldots,9\}}{\arg\max} \hat{\phi}_{a,t-1}^\top x_t$$

  where ϕ_a is obtained by solving the above.
- Thompson Sampling is similar to in standard linear bandits, but now it maintains a posterior distribution on ϕ_a for each action. At time t, it draws $\tilde{\phi}_a$ from the posterior for each a, and chooses the action A_t such that $\tilde{\phi}_{A_t}^\top x_t$ is maximized.

Implement each of these algorithms and plot the regret when run on MNIST. Experiment with different choices of d. Find values of d (and include corresponding plots) where LinUCB outperforms Explore-Then-Commit, and where Explore-Then-Commit outperforms LinUCB.

References