Title

JT Cho
joncho@
seas.upenn.edu

Karinna Loo
kloo@
seas.upenn.edu

Veronica Wharton
whartonv@
seas.upenn.edu

1 Introduction

For our CIS 625 final project, our team — JT Cho, Karinna Loo, and Veronica Wharton — took a closer look at the topic of fairness in machine learning. The paper that piqued our interest was *Rawlsian Fairness for Machine learning* (Joseph et al., 2016), which describes two online algorithms in the linear contextual bandit framework that both learn at a rate comparable to (but necessarily worse than) the best algorithms absent of a fairness constraint and also satisfy a specified fairness constraint. The authors present theoretical and empirical results. Our team sought to reimplement the algorithms presented by Joseph et al. (2016) and then expand upon their empirical analyses. We were also interested in exploring further fairness analyses using real-world data.

2 Project overview

Our project consisted of the following steps:

1. We read the paper *Rawlsian Fairness for Machine Learning* (Joseph et al., 2016).
2. We implemented the `TopInterval`, `IntervalChaining`, and `RidgeFair` algorithms from the paper in Python.
3. We ran our implementations on a Yahoo! dataset containing a fraction of the user click log for news articles displayed in the Featured Tab of the Today Module on the Yahoo! Front Page during the first ten days in May 2009 (Yahoo!, 2009), to see how well they performed on real data.
4. To empirically evaluate our implementations, we ran experiments similar to those in (Joseph et al., 2016) with randomly-drawn contexts.
5. We compiled our findings into a written report.

3 Algorithm implementations

The code for our implementations can be found here: `https://github.com/jtcho/FairMachineLearning/blob/master/fairml.py`

All algorithms and code were written using Python 3, along with numpy, SciPy, and various other Python libraries.

4 Implementation: TopInterval

The `TopInterval` learning algorithm was implemented true to form as presented in (Joseph et al., 2016). Particular details of note - to ensure that all matrices used in computation were nonsingular, the first d rounds are always chosen to be exploration rounds, where d is the number of features. Additionally, we found it necessary to pick each arm once in order to observe data for each.

5 Implementation: IntervalChaining

The implementation for `IntervalChaining` was simple given `TopInterval`, as it sufficed to alter the strategy for picking arms in each round to that of picking uniformly at random from the chain containing the top interval.

6 Implementation: RidgeFair

The `Ridge Fair algorithm` was also implemented as presented in (Joseph et al., 2016). This algorithm is very similar in implementation to Interval Chaining, save that its narrower confidence intervals allow for derivation of tighter regret bounds. Some details to note are that we assume for simplicity (and without loss of generality) that our parameter R = 1 and that we play uniformly randomly among all arms.

7 Yahoo! Dataset

To expand upon the initial work done by Joseph et. al, we endeavored to test the presented algo-

rithms on a real dataset. A Yahoo! dataset containing logs of user-visits to the front page was procured to evaluate our contextual bandits algorithms. Each log entry details the following:

unix_timestamp	displayed_id	user_clicked	user_features	article_pool
1241162400	109513	0	. . .	[. . .]

In each event, a user specified by 6 features is presented an article from a pool of around 20 distinct articles, each of which has their own 6-dimensional feature vector. The event also tracks whether the user clicked the featured article or not.

In a fashion similar to that presented in (Li et al., 2010), we devised an evaluation scheme for the various learning algorithms. In our procedure, a random sample is drawn from the set of logged events. The learning algorithm scans through the sampled events linearly, evaluating its predictions for each one. If there happens to be a match between the algorithm's picked arm and the article displayed in the event, the logged event is added to the history.

Initial attempts to use this approach failed for a couple of reasons. First, the Yahoo! dataset contains a highly disproportionate number of negative samples with respect to positive ones. Therefore, our learning algorithm would not retain useful information over a number of iterations due to only being trained on negative samples. Second, a direct application of the `TopInterval` and `IntervalChaining` algorithms relies on the assumption of 20 distinct underlying groups from which the articles were chosen to be in the article pool, each with their own distinct quality function. This assumption was found to be unreasonable, as we found that an article's index in the article pool had no bearing on its actual likelihood of being clicked by the user when picked. The initial context also does not work well with a fairness analysis. As a consequence, we saw that direct applications of the learning algorithms saw very poor performance.

To mitigate the first issue, we elected to alter our sampling procedure to separately sample positive and negative samples, and then shuffle them together. A brief argument towards the validity of this approach follows. While the underlying distribution of observed user visits saw mostly negative results, the algorithms performance should be independent of whatever underlying distribution there is - taking into account exclusively the user's features and the articles it is choosing from. Hence, curating the input to the learning algorithm such that it learns equally from both the positive and negative events suffices.

To resolve the second issue, we made a simplification to the problem context by clustering the articles. Across the million and a half logged events, there are approximately 20 distinct articles in the article pools. In choosing a smaller number of clusters, we altered the scenario such that a successful event would be if the user clicked an article that was from the same pool chosen by the algorithm. In grouping the articles together, we reduced the number of available arms and also developed the notion of 'groups' implicit in Joseph et. al.'s contextual bandits framework. The emergent notion of fairness then lies in discrimination against any particular cluster of articles.

These modifications resulted in significant improvements in the performance of our implementations on the Yahoo! dataset, as shown in Figure 1 below.

Another novel modification we made was the use of a logit model instead of the simple linear regression used in Joseph et al. (2016). We preserve the original fairness argument of the `IntervalChaining` algorithm by simply rescaling the output of the OLS estimator and the confidence intervals to $[0, 1]$ via the inverse logit. That is,

$$w_{t,i} = \mathcal{Q}_{\mathcal{F}_{t,i}}(\frac{\delta}{2kT})$$

$$[\ell_i^t, u_i^t] = [\Phi(\hat{y}_{t,i} - w_{t,i}), \Phi(\hat{y}_{t,i} + w_{t,i})]$$

where $\Phi(x) = \frac{e^x}{1+e^x} = \text{logistic}(x)$. It suffices to note that both OLS and logistic regression are variations of the generalized linear model (GLM).

8 Experimental results

We ran experiments that compared the regret of INTERVALCHAINING (IC) with the regret of TOPINTERVAL (TI). As in Joseph et al. (2016), we present three sets of empirical results:

- Varying T (the number of rounds) - we measured the average regret of INTERVALCHAINING and TOPINTERVAL as a function of increasing T.
- Varying k (the number of arms/groups) - we measured the average regret of INTER-

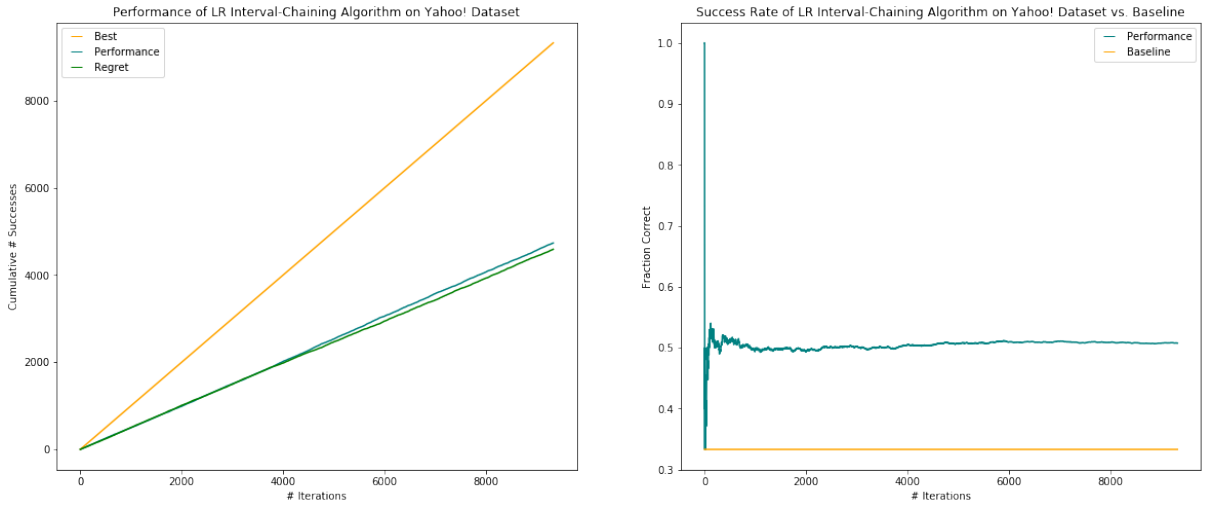

Figure 1: *Performance metrics of the logistic-regression based interval-chaining algorithm with* 3 *clusters over 10,000 iterations. Shown on the left is a graph depicting the performance of the learning algorithm vs that of the 'best' player, whose picked article is clicked by the user in every round. The regret is simply the difference in the cumulative number of successes between the two. In practice, this is an unfair comparison to make, as it is unreasonable to expect that the user would click the featured article every visit - and our results stand even stronger in comparison. On the right is a graph denoting the cumulative fraction of successful picks by the algorithm vs. the baseline (randomly selecting one out of the three pools at each step). The learning algorithm appears to converge to approximately* 50% *accuracy, which is considerably higher than the baseline.*

VALCHAINING and TOPINTERVAL as a function of increasing k.

- Varying d (the number of features) - we measured the average regret of INTERVALCHAINING and TOPINTERVAL as a function of increasing d.

For each increasing variable (T, k, or d), we present nine metrics as a function of the variable, each averaged over 500 trials. Contexts are drawn uniformly at random from $[0, 1]^d$ and standard Gaussian noise. Joseph et al. (2016) only present the average regret difference (metric #3).

1. Average regret (TI) - the average regret of TOPINTERVAL across all rounds.

2. Average regret (IC) - the average regret of INTERVALCHAINING across all rounds.

3. Average regret difference (TI vs. IC) - the difference between the average regrets of TOPINTERVAL and INTERVALCHAINING across all rounds.

4. Cumulative regret (TI) - the cumulative regret of TOPINTERVAL across all rounds.

5. Cumulative regret (IC) - the cumulative regret of INTERVALCHAINING across all rounds.

6. Cumulative regret difference (TI vs. IC) - the difference between the cumulative regrets of TOPINTERVAL and INTERVALCHAINING across all rounds.

7. Final regret (TI) - the regret of TOPINTERVAL in the final round.

8. Final regret (IC) - the regret of INTERVALCHAINING in the final round.

9. Final regret difference (TI vs. IC) - the difference between the final regrets of TOPINTERVAL and INTERVALCHAINING.

9 Conclusion

References

[Joseph et al.2016] Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. 2016. Rawlsian fairness for machine learning. *CoRR*, abs/1610.09559.

[Li et al.2010] Lihong Li, Wei Chu, and John Langford. 2010. An unbiased, data-driven, offline evaluation

method of contextual bandit algorithms. *CoRR*, abs/1003.5956.

[Yahoo!2009] Yahoo! 2009. Yahoo! front page today module user click log dataset. `https://webscope.sandbox.yahoo.com/catalog.php?datatype=r`. Accessed: 2017-04-03.