Better Caching with Machine Learned Advice

Thodoris Lykouris
Cornell University
teddlyk@cs.cornell.edu

Sergei Vassilvitskii
Google Research
sergeiv@google.com

ABSTRACT

We investigate the use of machine learning to improve eviction strategies in caching applications. We begin by showing that simply following an ML oracle may be far from optimal. We then adapt classical online algorithms to incorporate ML input. Our PredictiveMarker algorithm has provable performance guarantees and performs well empirically.

Introduction. While machine learning has made impressive gains in the past decade, there are still a lot of hurdles that one needs to overcome to deploy an ML system in practice [13].

One perennial problem is that many of the machine learning methods are typically trained to minimize the *expected* loss, that is they provide guarantees on the average performance of the method. The challenge is that ML approaches can sometimes output answers that are wildly far away from the truth [14], especially if the test examples are drawn from a slightly different distribution than the training set. The fact that many of the modern methods are black-box only exacerbates the problem when the ML system returns embarrassingly inaccurate answers.

While machine learning algorithms are in the business of predicting the future, online algorithms tell us how to act without *any* knowledge of future inputs. These powerful methods show how to hedge decisions so that regardless of what the future holds, the online algorithm performs nearly as well as the best offline optimum that has access to all of the available information.

However, since they protect against the worst case, online algorithms may be overly cautious, which translates to high competitive ratios even for seemingly simple problems. A prototypical example is the online paging, or caching problem: a series of requests arrives one at a time to a server equipped with some small amount of memory. Upon processing a request, the server places the answer in a local memory (in case an identical request comes in the near future). Since the local memory has limited size, the server must decide which of the current items to evict.

It is well known that if the local memory or *cache* has size k, then any deterministic algorithm has a competitive ratio, defined as ratio of the online algorithm's performance to the offline optimum, of $\Omega(k)$. However, a matching $O(k)$ bound can also be achieved by almost any reasonable strategy; thus this metric fails to distinguish between algorithms that perform well in practice, and those that perform poorly. The competitive ratio of the best randomized algorithm is $\Theta(\log k)$, which is also much higher than observed in practice.

In this work we ask: What if the online algorithm is equipped with a machine learned oracle? Can we achieve better competitive ratios given a machine learned estimate of what the future holds?

We look for algorithms that:

- Make minimal assumptions on the accuracy of the oracle. Specifically, since most machine learning guarantees are on the *expected* performance, we only parametrize our results by the average error of the ML-oracle, η.
- Are *consistent*: a better oracle (one with lower η) should lead to lower competitive ratios. We call an algorithm *perfectly consistent* if it is optimal when $\eta = 0$.
- Are worst-case competitive: no matter the performance of the oracle on an instance, the algorithm should behave no worse than the best competitive algorithm for the problem.

Related work. The case for using machine learning in low level system applications has recently been made in [8]. The authors show that it is indeed possible to achieve acceptable levels of prediction at a speed that makes them viable in practice. However they are forced to consider a restricted setting of immutable indices, precisely to avoid the problems with ML described above. Specifically, they evaluate on the training set, and thus have fine grained guarantees on the performance of the model.

In contrast, our work gives performance guarantees as a function of the average model quality. To create our algorithms, we build upon the foundational work on competitive analysis, (see the text by Borodin and El-Yaniv [4]). Specifically for the caching problem, many variants of caching have been studied over the years; our main starting point will be the Marker algorithm introduced by Fiat et al. [7].

Critically, we show that simply following the advice of the oracle can be far from optimal. Instead, we modify the Marker algorithm so that it gets the best of both worlds: when the ML error is low, its performance will be nearly optimal; when it is high, it will be not much worse than the approach without machine learning at all. Formally, we prove a bound on the competitive ratio as a function of the average error of the oracle.

The prior work that is closest in spirit to ours achieves a similar goal—algorithms that do (almost) as well as their counterparts that assume additional structure about the data, but also have worst case guarantees. Such structural assumptions include that the data follow some predicted patterns [9] or arrive from a stochastic distribution [5, 11]. On a related note, Ailon et al. [2] consider "self-improving" algorithms that effectively learn the input distribution, and adapt to be nearly optimal in that domain. Finally, recently Medina and Vassilvitskii [10] show how to use a machine learned oracle to optimize revenue in repeated posted price auctions. Here we extend

, ,

their model to the more general area of online algorithms and show how to use tools from competitive analysis to achieve consistent, and competitive algorithms for online caching.

Algorithms. It is well known that the optimal offline algorithm for the caching problem evicts the element from the cache that will arrive the furthest in the future. The popular *Least Recently Used*, LRU, policy is a heuristic approximation to this optimum, relying on locality of access that is often present in real world examples. From a theoretical analysis standpoint, any deterministic caching algorithm achieves a competitive ratio of $\Omega(k)$, and any randomized caching algorithm achieves a competitive ratio of $\Omega(\log k)$, see [12].

In this work, we assume access to an oracle h, which at every time t, predicts the *next* time $t' > t$ this element will arrive in the future. For an element coming at time t, let $y(t)$ be the actual time it appears next, and $h(t)$ be the predicted time. We consider the average error of the oracle, $\eta_1 = \frac{1}{n}\sum_{t=1}^{n} |h(t) - y(t)|$, but note that our results generalize to other metrics, such as squared loss.

It is tempting to simply trust the oracle, and evict the element predicted to appear furthest in the future. Unfortunately this may lead to arbitrarily bad performance. For a simple example, consider a sequence on three elements, $c, a, b, a, b, a, b, \ldots$, where the oracle returns $h(1) = 2$ for c, and is always correct regarding a and b. Then with $k = 2$, following the oracle will keep c in the cache, and iteratively evict a and b, resulting in repeated misses. On the other hand, the *average* error of the oracle is low; the trouble is that it is concentrated on one element. While this example is contrived, simple fixes do not work: even if the oracle only overestimates the arrival time we can come up with examples with competitive ratios that grow to infinity. As we will see, trusting the oracle also performs poorly empirically.

Results. Our main technical contribution is an oracle-based adaptation of the Marker algorithm [7] that achieves a competitive ratio of $\min(2 + 2\sqrt{\eta_1/OPT}, 4\log k)$ when using an oracle with loss η_1. Note that this competitive ratio is 2-consistent and 4-competitive.

The standard Marker algorithm runs in phases. In the beginning of each phase, all elements are unmarked. When an element arrives and it is already in the cache, the element is marked. If it is not in the cache, a *random unmarked* element is evicted, the newly arrived element is placed in the cache and is marked. Once all elements are marked, the phase ends and we unmark all of the elements.

The Marker algorithm never evicts marked elements when there are unmarked elements present. This property gives an upper bound of $O(k)$ on the competitive ratio for *any* tie-breaking rule that evicts an unmarked element. Fiat et al. [7] showed that evicting a random unmarked element gives an $O(\log k)$ competitive ratio.

We propose using the predictions made by ML for tie-breaking, specifically by evicting the element whose predicted next appearance time is furthest in the future. We show that doing so reduces the competitive ratio to $O(2 + 2\sqrt{\eta_1/OPT})$, but defer the proofs to the full version of the paper. With an extra modification, essentially switching to random eviction when the oracle performs poorly, we can cap the performance ratio at $\min(2 + 2\sqrt{\eta_1/OPT}, 4\log k)$

Empirical Evaluation. We give preliminary results showing the efficacy of our algorithm in practice. We consider sequences of

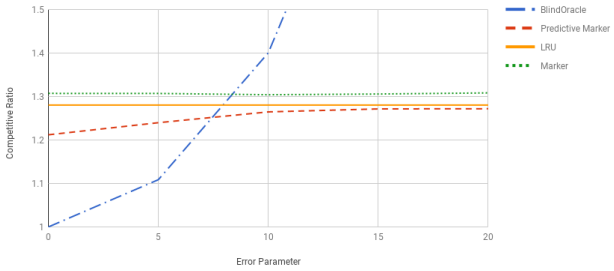

Figure 1: Ratio of average number of evictions as compared to optimum for varying levels of oracle error.

Algorithm	Competitive Ratio
Blind Oracle	2.049
LRU	1.280
Marker	1.310
Predictive Marker	1.266

Table 1: Competitive Ratio using PLECO model.

checkins from BrightKite, a now defunct social network, and extract the top 100 users with the longest non-trivial sequences. (Those where the optimum eviction policy makes at least 50 mistakes.) The dataset is publicly available at [1, 6].

We run two sets of experiments. First, to showcase the sensitivity to learning errors, we run a synthetic test. For each element, we compute the true next arrival time, setting it to $n + 1$ if it does not appear in the future. To simulate the performance of an ML system, we set $h(t) = y(t) + \epsilon$, where ϵ is drawn i.i.d. from a lognormal distribution with mean parameter 0 and standard deviation σ.

Second, we use the machinery introduced by Anderson et al. [3] to model repeat consumption. We implement their PLECO model, and use $h(t) = t + 1/p(t)$, where $p(t)$ represents the probability that element that appeared at time t will re-appear at time $t + 1$.

We compare our approach against the all-seeing ex-post optimal, the standard marker algorithm with the random tie-breaking rule, as well as LRU. Finally, to demonstrate the pitfall in simply following the oracle, we add an algorithm that evicts the element predicted to be furthest in the future, which we refer to as BlindOracle.

We set $k = 10$, and summarize the synthetic results in Figure 1, where we plot the competitive ratio of all algorithms. First, observe that the performance of Predictive Marker is consistently better than LRU and standard Marker, and degrades slowly as the average error increases, as captured by the theoretical analysis. Second, we empirically verify that blindly following the oracle works well when the error is very low, but quickly becomes incredibly costly.

The results using the PLECO predictor from [3] are shown in Table 1. We can again see that the Predictive Marker algorithm outperforms all others. While the gains appear modest, we note they are statistically significant at $p < 0.001$. Moreover, the off the shelf PLECO model was not tuned or optimized for predicting the *next* appearance of each element.

Conclusion. In this work we showed how to use ML advice to augment classical caching algorithms. Our proposed approach is efficient, has provable performance guarantees and performs well in simulation.

REFERENCES

[1] [n. d.]. Brightkite data. http://snap.stanford.edu/data/loc-brightkite.html. ([n. d.]).

[2] Nir Ailon, Bernard Chazelle, Kenneth L. Clarkson, Ding Liu, Wolfgang Mulzer, and C. Seshadhri. 2011. Self-Improving Algorithms. *SIAM J. Comput.* 40, 2 (2011), 350–375. https://doi.org/10.1137/090766437

[3] Ashton Anderson, Ravi Kumar, Andrew Tomkins, and Sergei Vassilvitskii. 2014. The Dynamics of Repeat Consumption. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. ACM, New York, NY, USA, 419–430. https://doi.org/10.1145/2566486.2568018

[4] Allan Borodin and Ran El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA.

[5] Sébastien Bubeck and Aleksandrs Slivkins. 2012. The Best of Both Worlds: Stochastic and Adversarial Bandits. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*. 42.1–42.23. http://www.jmlr.org/proceedings/papers/v23/bubeck12b/bubeck12b.pdf

[6] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and Mobility: User Movement in Location-based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*. ACM, New York, NY, USA, 1082–1090. https://doi.org/10.1145/2020408.2020579

[7] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. 1991. Competitive Paging Algorithms. *J. Algorithms* 12, 4 (Dec. 1991), 685–699. https://doi.org/10.1016/0196-6774(91)90041-V

[8] Tim Kraska, Alex Beutel, Ed H. Chi, Jeff Dean, and Neoklis Polyzotis. 2017. The Case for Learned Index Structures. https://arxiv.org/abs/1712.01208

[9] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. 2012. Online Optimization with Uncertain Information. *ACM Trans. Algorithms* 8, 1 (2012), 2:1–2:29. https://doi.org/10.1145/2071379.2071381

[10] Andrés Muñoz Medina and Sergei Vassilvitskii. 2017. Revenue Optimization with Approximate Bid Predictions. *CoRR* abs/1706.04732 (2017). http://arxiv.org/abs/1706.04732

[11] Vahab S. Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. 2012. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. 1690–1701. http://portal.acm.org/citation.cfm?id=2095250&CFID=63838676&CFTOKEN=79617016

[12] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA.

[13] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511. http://dl.acm.org/citation.cfm?id=2969442.2969519

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199 (2013). http://arxiv.org/abs/1312.6199