Measuring the "on-lineness" of data streams

Manfred K. Warmuth
UC Santa Cruz

Jiazhong Nie
Google

Abstract

In on-line learning, the performance of the on-line algorithm is measured by its regret, which is the additional loss the on-line algorithm incurs over the loss of the best off-line comparator. We discuss methods for designing good off-line comparators that exploit the "on-lineness" and the statistical niceness of the data stream.

On-line learning proceeds in trials. In each trial, the on-line learner has to predict on the current instance based on the past examples. Then after receiving the label, the learner incurs a loss measuring the quality of its prediction. The regret is typically compared against the best off-line predictor that has access to all T examples.

In this note we ask ourselves what constitutes a good/fair comparator.

1. At trial $t = 1..T$ before predicting, the on-line algorithm has access only to the past t instances and the past $t-1$ labels, whereas the off-line comparator is based on all T examples. This is unfair to the on-line algorithm, since it has less information than the comparator.
2. The on-line algorithm can change its hypothesis from trial to trial and exploit local patterns of the data stream, whereas the off-line algorithm has to settle on a single hypothesis for all trials. This unnecessarily handicaps the comparator since its loss is invariant under permuting the examples and it cannot exploit local patterns.

Regarding 1, we seek comparators that are given less information about the future examples. Regarding 2, we let the comparator change its hypothesis as well. For each number of shifts k, we compute the minimum loss of any partition of the all T trails into k segments. Each of the k segments contributes the total loss of the best predictor for that segment. Computing the total loss of the best partition for each choice of k seems to be formidable, but it is manageable using dynamic programming.

We have explored this idea for two practical example problems in the Operating Systems domain: On-line tuning of the timeout for the disk of a laptop (See [HLS96] for a description of this problem) and building a combined caching strategy from a base set of known caching strategies. For the former problem, we fix a set of timeouts in a suitable range. The on-line algorithm predicts with a mixture of these timeouts. We let BestShift(k) denote the loss (here the energy use) of the best partition into k segments where in each segment the best timeout is used from the discretized set. This curve is computed via dynamic programming (See Figure 1).

We argue that the BestShift(k) curve is a good characterization of the "on-lineness" of the data stream. The total energy goes down with the number of shifts. If the initial drop is large (i.e. curve is roughly _ shaped), then we know that the data stream has the property that shifting between timeouts leads to great energy savings for small values of k. At some point the curve flattens out and further shifts are not as helpful. Ideally the total loss of an on-line algorithm should be as low as the bottom of the _ curve, i.e. it should exploit essentially all useful shifts in the data stream.

In past work, a similar curve has been computed for the caching problem [GWBA02]. We will show that in that application the on-line algorithms reached the bottom of the _ curve on natural

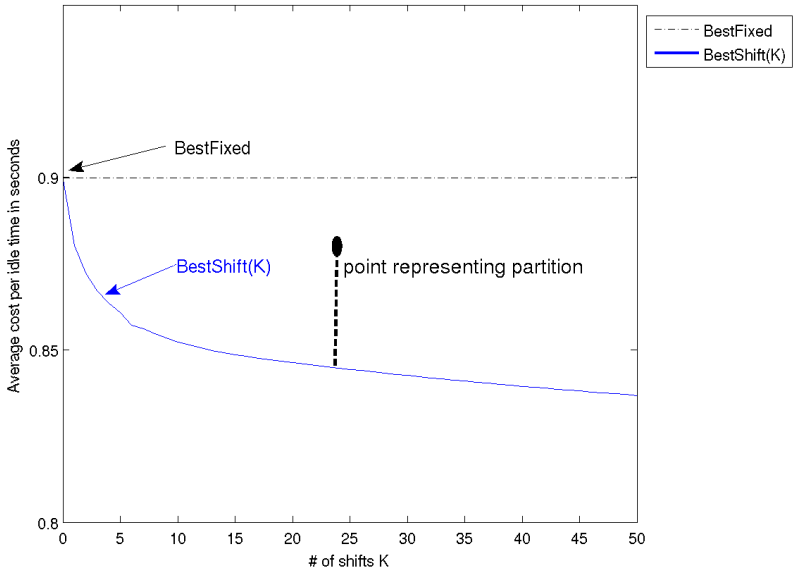


Figure 1: Dynamic programming costs $O(KN^2T)$ time,
where K # of partitions, N # of discrete idle times, T # of trials.

data streams. Inspired by this success we applied this dynamic programming approach to the disk spin down problem as well (not published) and systematically explored the performance of on-line algorithms in relation to this curve.

In both applications the current analysis methods for on-line learning methods do not apply. For the disk spin down problem, the loss function is not even convex and in the caching problem the actual cache corresponds to an unusual "delayed mixture" of the base strategies. Nevertheless standard on-line algorithms based on a suitably chosen set of experts with combined exponential and share updates (See e.g. [BW02]) become useful heuristics. In the absence of theoretical guarantees, we found that the use of the BestShift(k) curve particularly useful. The curves give a clear goal to shoot for on the particular data stream at hand. If the curve is flat, then no on-line algorithm is expected to predict well. However if the curve is _ shaped, then the on-line algorithm better exploit the initial drop of the curve.

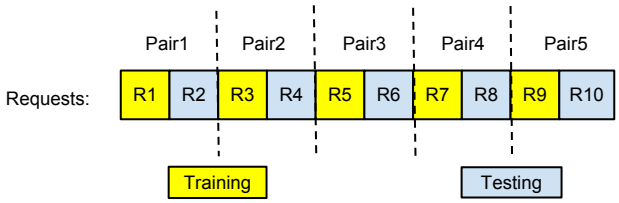
We propose to greatly expand the usage of such curves for practical on-line learning problems that are out of the reach of theoretical analysis and discuss pros and cons of this approach:

- Computing the BestShift(k) is expensive. However it can be done for small typical segments of the data stream and can therefore still be used as a diagnostic tool.
- We simply chose the best predictor, as a base algorithm for each segment, However if the segments are expected to be iid, then one should use the Follow the Leader algorithm for each segment.
- Shifting for free may be seen as unfair. In that case one might incorporate a cost for shifting between segments. This cost should measure how far the predictors in the segments are apart, ie. shifting between close predictors should be cheaper. We already did this in the caching application with great success.
- The goal is to find a "fair" comparator that essentially models the performance of the on-line algorithm on a large variety of real data streams. The comparator then becomes a good predictor of the performance of the on-line algorithm.

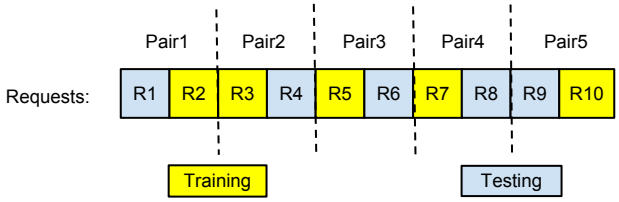
The first key idea that we will present is the BestShift(k) curve and its many uses. We will then describe a refinement of these curves. Consider an on-line data stream with a pronounced _ shaped

curve. Now randomly permute the data stream. This surely will obliterate all on-lineness of the data stream. The BestShift(k) curve should be flat or increasing. However since the curve is computed with full information of all examples, further segments always improve the overall loss. So even if the date stream is randomly permuted, the BestShift(k) curve is slowly decreasing. This phenomenon is due to over fitting: The comparator knows too much about the future data stream (See Part 1 above).

In the refinement of the curve, we group the trials (requests in the caching application) into consecutive pairs.



The odd requests (first in each pair) constitute the training stream and even requests the test stream. The best partition is chosen based on the training stream, but the total loss (number of misses) of the chosen partition is measured by the total loss of on the test stream. An alternate is to randomly choose for each pair one as training and one as test.



However in our real data streams this alternate (and a number of other variations we tried) were not necessary.

The upshot is that now the refined BestShift(k) curves for randomly permuted data streams slowly increases from $k = 1$ onward, ie. the best partition has one segment. For on-line data streams, the initial drop is still there in the refined BestShift(k) curve, but then the total loss is now slowly increasing. This parallels the situation for standard batch learning where the data is split into a training and test set: The performance on the training set continues to decrease, while the performance on the test set drops to a low point and then starts to increase.

We hope that this talk will encourage our community to work on messier but practically relevant on-line learning problems where the on-line heuristics can still be evaluated with carefully crafted comparators.

References

[BW02] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

[GWBA02] Robert B. Gramacy, Manfred K. Warmuth, Scott A. Brandt, and Ismail Ari. Adaptive caching by refetching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 1465–1472. MIT Press, 2002.

[HLS96] D.P. Helmbold, D.D.E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the Second Annual ACM International Conference on Mobile Computing and Networking*. ACM/IEEE, November 1996.