Online Algorithms and Machine-Learned Advice*

Hsiang-Hsuan (Alison) Liu

Machine learning and online algorithms both deal with decision making under uncertainty. When the future input is unknown, online algorithms make decisions based on the prefix of input and previous decisions. The goal of an online algorithm is to perform not too much worse than the optimal solution with hindsight. On the other hand, machine learning tries to predict the future input based on historical data. Recently from 2017, people started combining online algorithms with machine-learned advice. The idea is, suppose that the future input can be predicted, how (and how much) can online algorithms' decision quality be improved?

Although it looks promising that machine learning techniques can predict the future input quite well if there are sufficient data, the prediction can hardly be completely correct. The online algorithm, which has only one shot for making decision upon a piece of revealed input, can be fooled if the prediction has even a small amount of error.

Take the Ski-Rental problem as an example, the prediction can be the weather forecast, which tells us how many skiing days we will have. If the forecast tells that there will be at least B skiing days (where B is the buying price for a pair of skis), a fully-trusting online algorithm may decided to buy the ski on the first day. In a worst case, the weather will not be better anymore in the future, and the online algorithm pay B times of the optimal cost.

Apart from the error, the prediction quality can also be affected if there are malicious data fed by adversarial components. The goal of an online algorithm with access to machine-learned advice is to be:

- *consistent*: given more accurate predictions, the online algorithm should perform close to the optimal offline algorithm, and
- *robust*: if the prediction is wrong, the online algorithm performance should be close to the online algorithm without predictions.

1 A simple example: Searching

We first have a basic idea about how an algorithm can accommodate the prediction that may contain errors. Consider the following searching problem, where given a sorted (but elements are unknown) array, the algorithm wants to access a requested value within a minimum number of queries (that is, accessing some element in A):

Searching problem

Given a sorted array A and a request r, access the element in A that contains r.

This problem can be seen as an online problem, our aim is minimizing the competitive ratio. That is, we hope to minimize the ratio between the number of queries incurred by our algorithm to the offline optimal one.

Now, consider that we are given a prediction p, which is the position of r in A. The prediction is not necessary to be correct. That is, $A[p]$ may not be r. We have the following algorithm with request r and prediction p:

*This lecture note is based on three papers published from 2018 to 2020 [1, 2, 3].

Algorithm 1 A searching algorithm with prediction of the position of the requested value in the array

```
Access A[p]
if A[p] = r then
    Return p
else
    Return Binary Search r in range A[1 ··· p] or A[p ··· n] according to the value of element A[p]
```

Analysis of Algorithm 1. To analyze algorithms with predictions, we consider two cases: 1) the prediction is correct, and 2) the prediction includes errors. If the prediction is correct, that is, $A[p] = r$, the algorithm needs one access to find the request. On the other hand, if the prediction is wrong, the algorithm will then perform a binary search, and it costs at most $O(\log n)$ accesses in the worst case.

Note that the optimal solution, which knows where exactly the requested number r is, only needs 1 access. On the other hand, the binary-search strategy of the pure online algorithm, which knows nothing about the array but the fact that it is sorted, needs $O(\log n)$ accesses to find r in the worst case. In fact, binary search is the best strategy for an algorithm which knows nothing about the instance.

You can see that when the prediction is correct, the online algorithm with prediction performs as well as the offline solution, which means that it is *consistent*. When the prediction is wrong, the online algorithm performs as well as the (best) pure online strategy, which means that it is *robust*.

2 Trusting the prediction is not always good

In the searching problem, we saw that with the prediction, a binary-search algorithm easily achieves consistency and robustness at the same time. However, in a more complicated setting, trusting the prediction completely causes a disaster.

SKI-RENTAL WITH PREDICTION ON THE NUMBER OF SKIING DAYS problem

Assume that the price for buying a pair of ski is B and the renting price is 1 per day. There is a weather forecast predicting that there will be p skiing days. At which day of skiing should you buy a ski?

Now, consider the naive strategy that follows the prediction. That is, if $p \geq B$, the online algorithm with prediction buys the ski on the first day; otherwise, it keeps renting:

Algorithm 2 A trusting online algorithm for the BUY-OR-RENT problem with prediction

```
if p ≥ B then
    Buy on the first day
else
    Keep renting for all skiing days
```

Analysis of Algorithm 2. For the consistency, there are two cases of the actual number of skiing days S: 1) $S \geq B$, and 2) $S < B$. When $S \geq B$, the optimal solution buys the ski on the first day and pays B. A correct prediction will suggest the algorithm to buy the ski. Algorithm 2 follows the advice and buy the ski on the first day. In this case, the algorithm behaves as well as the optimal solution. When $S < B$, the optimal solution keeps renting. Algorithm 2 follows the (correct) advice and keeps renting the ski. Therefore, it behaves as well as the optimal solution. The algorithm behaves exactly the same as the optimal solution as long as the prediction is correct and has good consistency.

For the robustness, there are again two cases of the actual number of skiing days. When $S \geq B$, a *bad* prediction may suggest the algorithm to keep renting the ski. In this case, Algorithm 2 pays S while the optimal solution pays B. On the other hand, when $S < B$, a bad prediction may suggest the algorithm to buy the ski. Algorithm 2 buys the ski on the first day and pays B. In the worst case,

there is only one skiing day, and the optimal solution is to rent the ski for one day and to pay 1. Thus, the competitive ratio of Algorithm 2 under the wrong prediction is $\max\{\frac{S}{B}, B\}$, which can be infinite when S tends to infinity. The algorithm has a bad robustness.

A small improvement. One can improve Algorithm 2 by modifying its behavior when the prediction tells us we should not buy the ski:

Algorithm 3 A modified trusting online algorithm for the BUY-OR-RENT problem with prediction

1: **if** $p \geq B$ **then**
2: `Buy` on the first day
3: **else**
4: Keep `renting` until the B-th skiing day

Algorithm 3 performs better than Algorithm 2 when the prediction is wrong, and there are at least B skiing days. In this case, the algorithm buys the ski on the B-th day and pays $(B-1)+B$. The robustness is improved to $\max\{2-\frac{1}{B}, B\}$. Unfortunately, it is still an increasing function of the instance.

3 Balance the trustness and the risk

In the previous section, we saw the unfortunate outcome of fully trusting the advice. More specifically, the online algorithm performance may be extremely bad when the prediction is wrong. Therefore, we have to balance our trustness of the prediction and the robustness.

Trustness parameter. To express the non-binary trustness of the algorithm to the prediction, we introduce the concept of *trustness parameter*, which indicates how much the algorithm trusts the prediction. In practical, the range of the trustness parameter can be defined for the convenience of the online algorithm. For example, it can be between $[0, 1]$, where 1 means fully trusting and 0 means not trusting the advice at all. By tuning the value of the trustness parameter, one can tune the level of trustness and the behavior of the online algorithm.

Take the Ski-Rental algorithm as an example. We can introduce the trustness parameter $k \in [1, B]$ and have the following algorithm:

Algorithm 4 An online algorithm with a trustness parameter k

1: **if** $p \geq B$ **then**
2: `Buy` on the k-th day
3: **else**
4: Keep `renting` until the B-th skiing day

You can see that when $k = 1$, the algorithm buys the ski on the first day when the advice says that one should buy the ski. On the other hand, when $k = B$, the algorithm buys the ski on the ski on the B-th day even when the advice tells it to buy the ski. That is, the online algorithm does not trust the prediction at all and behaves like the pure online algorithm without advice.

Analysis of Algorithm 4. For the consistency, Algorithm 4 pays $(k-1)+B$ if there are at least B skiing days and pays S if there are $S < B$ skiing days. In these cases, the competitive ratio of the online algorithm with prediction is $\max\{\frac{B+k-1}{B}, \frac{S}{S}\} = 1 + \frac{k-1}{B}$.

For the robustness, Algorithm 4 pays $(B-1)+B$ when there are at least B skiing days, but the prediction says that there will be fewer than B skiing days. On the other hand, Algorithm 4 pays $(k-1)+B$, while the optimal pays k in the worst case when there are fewer than B skiing days. In the worst case, the competitive ratio of the algorithm is $\max\{2-\frac{1}{B}, \frac{k-1+B}{k}\} \leq \max\{2-\frac{1}{B}, 1+\frac{B-1}{k}\}$.

4 A more complicated online algorithm with prediction

The huge difficulties for making online decisions are *irrevocability* and the *uncertainty of instance*. Some researchers study the effect of uncertainty by studying semi-online algorithms, which knows

partial information of the instance. These semi-online algorithms may perform better than the pure online algorithms, given that they know more than the pure online algorithms. For example, no pure deterministic online algorithm can be better than 1.5-competitive for the Bin-Packing problem. But with a small piece of information, a semi-online algorithm can achieve 1.5-competitiveness:

Online algorithm with *trusted* prediction. First, we classify the items by their size:

- *Tiny* items: items with size in $(0, \frac{1}{3}]$,
- *Small* items: items with size in $(\frac{1}{3}, \frac{1}{2}]$,
- *Critical* items: items with size in $(\frac{1}{2}, \frac{2}{3}]$, and
- *Large* items: items with size in $(\frac{2}{3}, 1]$.

The Reserve-Critical Algorithm 5 also labels the bins by *tiny*, *small*, *critical*, and *large*. With the information about the number of critical items will be in the instance, c, the algorithm first open c critical bins, reserved for critical items. In principle, the algorithm is very similar to First-Fit but with some other rules. Once a bin is opened by a large item, the algorithm will not put any other item in it. Similarly, once a bin is opened by a small/tiny item, the algorithm only put small/tiny items in it. The critical bins are the most complicated. In each of the critical bins, there will be one critical item. Meanwhile, in each critical bin, there will be a space of size $\frac{1}{6}$ reserved for tiny items. More specifically, when a tiny item arrives, the algorithm first seeks if there is a critical bin that can accommodate it. A tiny bin is opened only when no critical bin or tiny bin can accommodate the released tiny item.

Algorithm 5 Reserve-Critical(c): A semi-online algorithm knowing the number of critical items c

```
Open c critical bins. In each critical bin, pack one critical item and tiny items with total size ≤ 1/6
while Item i arrives do
    if i is a large item then
        Open a large bin and pack i into it
    if i is a small item then
        pack i into a small bin by First-Fit
    if i is a critical item then
        pack i into an opened critical bin that has no critical item
    if i is a tiny item then
        if There is a critical bin with total size of tiny items less than 1/6 − size(i) then
            pack i into that bin
        else
            pack i into a tiny bin by First-Fit
```

With the (correct) information of the number of critical items, the algorithm performs very well:

Theorem 1. *The Reserve-Critical Algorithm is* 1.5*-competitive.*

Now, let's see what happens if the information of the number of critical items is learned by machine-learning and may contains errors. The risk of trusting the machine-learned prediction of the number of critical items is that, if we open too many critical bins and reserved them for the (never arriving) critical items, they may only contain tiny items and be at least $\frac{5}{6}$-empty. On the other hand, if we don't use this piece of prediction, we can only perform as good as the pure online algorithm.

The following Robust-Reserve-Critical Algorithm is given the prediction $\frac{c}{c+t}$, where c is the number of critical bins, and t is the number of tiny bins of Reserve-Critical Algorithm, respectively. The trustness parameter α is in the range $[0, 1]$.

You can see that Robust-Reserve-Critical Algorithm is more reserved about opening critical bins than Reserved-Critical Algorithm. Instead of opening c critical bins in the very beginning as Reserved-Critical Algorithm, Robust-Reserve-Critical Algorithm sets a "threshold" β according to the predicted $\frac{c}{c+t}$ and the trustness parameter α and tries to keep the ratio of the number of critical bins and the number of tiny bins closed to β.

The detailed analysis can be found in the paper by Angelopoulos et al. [1].

Theorem 2. *If the prediction on* $r = \frac{c}{c+t}$ *is correct, Robust-Reserve-Critical is at most* $(1.5 + \frac{1-\alpha}{4-3\alpha})$*-competitive. Otherwise, it is at most* $(1.5 + \max\{\frac{1}{4}, \frac{9\alpha}{8-6\alpha}\})$*-competitive.*

Algorithm 6 Robust-Reserve-Critical($r = \frac{c}{c+t}, \alpha$)

Let $\beta \leftarrow \min\{r, \alpha\}$
When a critical item arrives, pack it into a critical bin
When a tiny item arrives, pack it into an available critical bin
if There is no available critical bin **then**
    Pack it into a tiny bin
    if There is no available tiny bin **then**
        Open a new bin B
        Let c' and t' be the number of currently opened critical bins and tiny bins, respectively
        B is a critical bin if $\frac{c'}{c'+t'} < \beta$; otherwise, B is a tiny bin
Large items and small items are packed into large bins and small bins respectively

5 Competitiveness analysis taking learning error into account

For analyzing the performance of online algorithms with prediction, one can take the quality of the prediction into account. More specifically, we want quantify the effect of the wrong prediction on the robustness of the online algorithm.

The most common ways to measure the prediction error are (where p is the prediction, and t is the true value of the predicted thing):

- *Absolute error* $\eta_1 = |p - t|$,
- *Squared error* $\eta_2 = |p - t|^2$, and
- *Classifica error* $\eta_c = 1$ if $p \neq t$.

Next, we use the absolute error to analysis the robustness of the Ski-Rental algorithm. For the sake of analysis, we modify the algorithm as follows:

Algorithm 7 An online algorithm with a trustness parameter $\alpha \in (0, 1)$

if $p \geq B$ **then**
    `Buy` on the $\lceil B(1-\alpha) \rceil$-th day
else
    Keep `renting` until the $\lceil \frac{B}{1-\alpha} \rceil$-th skiing day

Figure 1 is an illustration of Algorithm 7. If the prediction suggests that the algorithm should buy the ski, the algorithm with trustness parameter α will buy the ski on the day marked by teal; otherwise, the algorithm will keep renting until the day marked by blue. If the trustness parameter is closer to 0, both the teal day and the blue day will be closer to B. That is, the algorithm behaves more similar to the pure online algorithm.

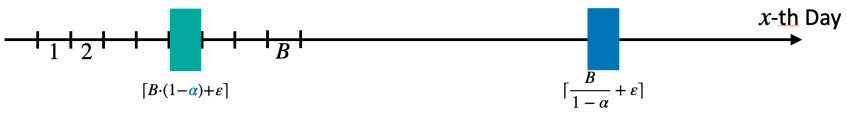

Figure 1: Illustration of Algorithm 7

Analysis of Algorithm 7. For the consistency, the algorithm pays $B(1-\alpha) - 1 + B$ when the actual skiing day $S \geq B$ and pays S otherwise. In this case, the algorithm has competitive ratio $\frac{B(1-\alpha)-1+B}{B} = 2 - \alpha - \frac{1}{B}$.

For the robustness, there are two cases: 1) $S < B$, and $S \geq B$. In the first case, OPT $= S$, and $B \leq p = S + \eta_1$ (see Figure 2). The algorithm buys the ski on day $\lceil B(1-\alpha) \rceil$. In the worst case, the online algorithm cost can be charged to OPT + (OPT + η_1). More precisely the online algorithm pays at most $(2-\alpha)(\eta_1 + \text{OPT})$.

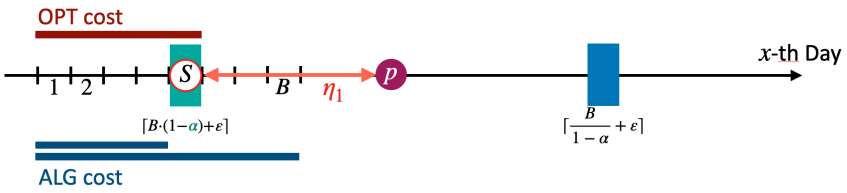

Figure 2: Illustration of the case where $S < B$, but $p \geq B$. The length of the red bar visualizes the optimal cost, while the blue bars visualize the algorithm cost.

In the second case, $p < B$, and there are three sub-cases depending on the actual number of skiing days: $S < B$, $B \leq S < \lceil \frac{B}{1-\alpha} \rceil$, and $S \geq \lceil \frac{B}{1-\alpha} \rceil$. The first sub-case is simple as the algorithm behaves the same with the optimal solution. In the second sub-case, $\eta_1 = S - p$. The algorithm keep renting, while the optimal solution buys the ski on the first day. Thus, the algorithm pays at most $S = p + \eta_1 \leq \mathrm{OPT} + \eta_1$. (See Figure 3.)

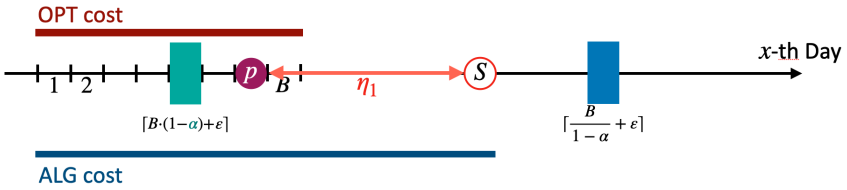

Figure 3: Illustration of the case where $S \geq B$, but $B \leq S < \lceil \frac{B}{1-\alpha} \rceil$.

In the third sub-case, $\eta_1 \geq \frac{B}{1-\alpha} - B$. The algorithm first rents the ski for $\frac{B}{1-\alpha} - 1$ days and eventually buys the ski. In total, the algorithm pays $\frac{B}{1-\alpha} - 1 + B$, which can be charge to $\mathrm{OPT} + (\mathrm{OPT} + \eta_1)$. More precisely, the algorithm cost is at most $\mathrm{OPT} + \frac{\eta_1}{\alpha}$ since $\eta_1 + B \geq \frac{B}{1-\alpha}$.

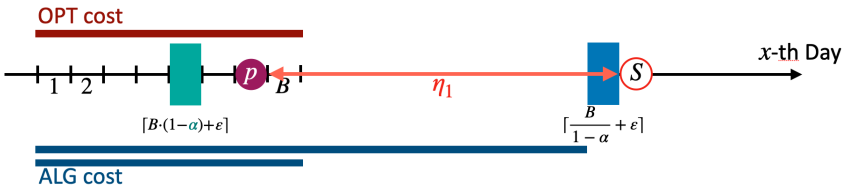

Figure 4: Illustration of the case where $S \geq B$, but $S \geq \lceil \frac{B}{1-\alpha} \rceil$

References

[1] S. Angelopoulos, C. Dürr, S. Jin, S. Kamali, and M. P. Renault. Online computation with untrusted advice. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[2] T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

[3] M. Purohit, Z. Svitkina, and R. Kumar. Improving online algorithms via ML predictions. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018.