Theoretical Computer Science 519 (2014) 29–45

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Combining initial segments of lists ☆

Manfred K. Warmuth [a,1], Wouter M. Koolen [b,c,2], David P. Helmbold [a]

[a] Department of Computer Science, UC Santa Cruz, United States
[b] Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW200EX, United Kingdom
[c] Centrum Wiskunde en Informatica (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Online learning
Ranking
Worst-case analysis
Relative loss bounds

ABSTRACT

We propose a new way to build a combined list from K base lists, each containing N items. A combined list consists of top segments of various sizes from each base list so that the total size of all top segments equals N. A sequence of item requests is processed and the goal is to minimize the total number of misses. That is, we seek to build a combined list that contains all the frequently requested items. We first consider the special case of disjoint base lists. There, we design an efficient algorithm that computes the best combined list for a given sequence of requests. In addition, we develop a randomized online algorithm whose expected number of misses is close to that of the best combined list chosen in hindsight. We prove lower bounds that show that the expected number of misses of our randomized algorithm is close to the optimum. In the presence of duplicate items, we show that computing the best combined list is NP-hard. We show that our algorithms still apply to a linearized notion of loss in this case. We expect that this new way of aggregating lists will find many ranking applications.

1. Introduction

We propose a new approach for aggregating ranked lists. Assume we have K lists of N items each, as illustrated in Fig. 1. Our comparator is the best combined list of size N that is composed of the tops of the K lists. A combined list might take the top 20% of list 1, the top 0% of list 2, the top 60% of list 3, and so forth. Note that the contents of the base lists might change over time, and there are exponentially many (roughly N^K) combined lists altogether.

We seek efficient online algorithms that construct such combined lists on the fly. In each trial the following happens: First, the current contents of all base lists are provided to the algorithm. Then the algorithm assembles (either deterministically or randomly) its combined list. After that some item is requested. If it is not in the chosen combined list, then the algorithm incurs a miss (one unit of loss). Some or all of the base lists might also miss the requested item and might update themselves accordingly for the next trial.

The goal of the algorithm is to endure small additional loss (regret) over the best combined list chosen in hindsight once the entire request sequence and the time-varying contents of the base lists are known. We seek efficient online algorithms that implicitly maintain some information about all roughly N^K combined lists and can efficiently choose or sample a combined list from this information. As we shall see, randomness is a necessary ingredient for algorithms with good regret guarantees.

☆ An extended abstract of this paper appeared in [34].

E-mail addresses: manfred@cse.ucsc.edu (M.K. Warmuth), wouter@cs.rhul.ac.uk (W.M. Koolen), dph@cse.ucsc.edu (D.P. Helmbold).

[1] Supported by NSF grant IIS-0917397 and a Google gift grant.
[2] Supported by NWO Rubicon grant 680-50-1010.

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2013.09.021

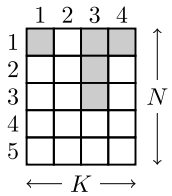

Fig. 1. We depict one combined list formed by taking the tops from $K = 4$ lists. Note that all lists have size $N = 5$ and the combined list has the same size. The list shown is $\boldsymbol{c} = (1, 0, 3, 1)$.

We claim that this setup has many applications. For example we can use our method to combine the listings produced by different search engines. Here a first goal is to combine the tops of the base lists for the purpose of maximizing hits. However in the case of search engines, we are also concerned with accumulating hits at the top of our chosen combined list and this aspect is not yet modeled by our approach. We briefly discuss such extensions in the conclusion Section 7.

Another important application is caching. In this case each list is the ranked list of items selected by a different caching strategy. We assume that all items have the same size and exactly N fit into the fast memory. The algorithm can simulate K caching strategies as "virtual caches" and then combines these virtual caches to form one "real cache". The virtual caches only record a few bytes of meta-data about each item in their cache: ID, link to the data, and calculated priority. Object data is only kept for the N items of the real combined cache. The memory cost for maintaining the virtual caches is negligible. The real combined cache can be updated as the virtual caching strategies change their item lists and the algorithm observes the performance of its combined cache.

Previous work. Methods for building a real cache by combining a number of virtually maintained caching strategies using exponential weights were explored in [15]. The employed heuristics performed well experimentally. However no performance bounds were proven. The idea for building a real cache by combining the tops of two virtual lists was first developed in [26,27]. In this case the first list contained the most recently requested items and the second contained those that were requested more than once. The goal for building a combined list was to optimally balance recency and frequency information. A deterministic heuristic was developed for adjusting the top portion taken from each list. In this paper we prove a lower bound for any deterministic algorithm that shows that any such algorithm can be forced to have loss at least KB where B is the loss of the optimal combined list chosen in hindsight. We also give a randomized online algorithm for the disjoint case whose expected loss is at most B plus an additional sublinear regret and show that the regret of this algorithm is optimal within a factor of $\min\{\sqrt{K}, \sqrt{\ln(N/K)}\}$.

This paper was motivated by our previous work on combining caching heuristics [15]. In general, there are always two approaches to a rich problem setting. Either we can restrict the setting enough so that theoretical bounds are obtainable, or we can explore heuristics without provable guarantees that perform well in practice. Ideally the two approaches will meet eventually. In this paper we focus on algorithms for which we can prove regret bounds and we believe we made significant progress towards addressing practically interesting problems.

Aggregating experts. One of the main metaphors of online learning has been the aggregation of many simple "experts" [25, 32,9]. In our application, each combined list serves as an expert. The online algorithm maintains its uncertainty over all experts as a mixture distribution and predicts based on this mixture. In more detail, the probability of a combined list $\boldsymbol{c}$ is proportional to $\beta^{\boldsymbol{M}(\boldsymbol{c})}$, where $\boldsymbol{M}(\boldsymbol{c})$ is the number of misses of list $\boldsymbol{c}$ and the update factor β lies in $[0, 1)$.

There are exponentially many mixture weights (one per expert). However, as we shall see later, we still manage to obtain very efficient algorithms by manipulating the weights implicitly. This paper is the next installment in a rich tradition of papers exploiting the combinatorial structure of the chosen comparator class (see references in [23]). In our case of combined lists, we were able to augment dynamic programming [30] with an additional speedup using Fast Fourier Transforms.

We first discuss how to obtain a combined list from a mixture. One could consider thresholding the mean, using the median (introduced below, but only works for 2 lists) or sampling.

The weighted average (mean) does not correspond to a combined list. A deterministic algorithm would naturally predict with the weighted majority (mean) of the mixture. That is, an item is in the chosen combined list if the total weight of all combined lists in the mixture that contain this item is at least 50%. Unfortunately, the weighted majority of the mixture does not always correspond to a list of size N. For $N = 4$ and $K = 3$, consider the case shown in Fig. 2, where we mix 3 combined lists each containing the top halves of 2 out of the 3 base lists. With the uniform mixture of these 3 combined lists, 6 items have total weight 2/3, more than the $N = 4$ allowed.

Deterministic algorithm for the case of two lists based on the median. In the case of $K = 2$ (i.e. two lists of size N), there are $N + 1$ combined lists $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_N$, where $\boldsymbol{c}_i = (i, N - i)$ contains the i top elements from the first list and the $N - i$ top elements from the second list. These combined lists are displayed in Fig. 3.

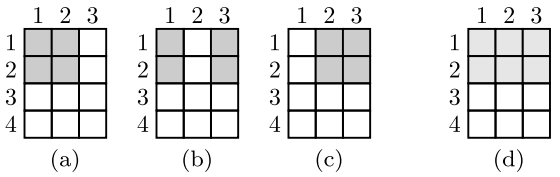

Fig. 2. Counterexample against obtaining a combined list by mean thresholding. Consider the combined lists (a), (b) and (c), which each use 4 items with weight 1. The uniform mixture over (a), (b) and (c) (displayed as (d)), sports 6 elements with weight 2/3 each. By including the items whose mean weight exceeds some threshold, we end up with either 0 or 6 elements, but not the desired 4.

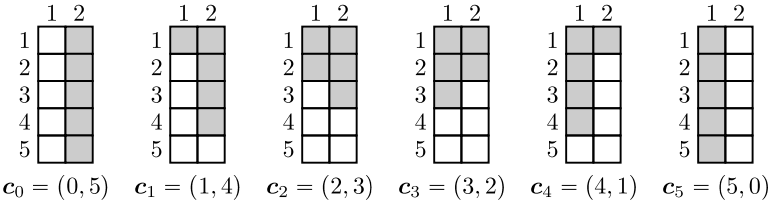

Fig. 3. The 6 combined lists of size $N = 5$ from $K = 2$ base lists.

For $K = 2$ disjoint lists there is a simple algorithm producing a single combined list of the right size N based on the median that circumvents the problem with the mean discussed in the previous section. It maintains the same exponential weight discussed before but chooses a combined list $\boldsymbol{c}_i$ s.t. the total weights of the lists $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{i-1}$ and the lists $\boldsymbol{c}_{i+1}, \ldots, \boldsymbol{c}_N$ are both at most half. In other words the algorithm picks the combined list from $\langle \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_N \rangle$ with the *median* weight and we call this deterministic algorithm the *Weighted Median algorithm*. Whenever a miss occurs, then at least half of the total weight is multiplied by β: If the item was on the first list and was missed by the median list $\boldsymbol{c}_i$, then at least $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_i$ are multiplied by β, which contain at least half of the total weight. Similarly, if the item was on the second list and missed by $\boldsymbol{c}_i$, then at least $\boldsymbol{c}_i, \ldots, \boldsymbol{c}_N$ are multiplied by β, which is again at least half of the total. The case when the item did not appear on either base list is trivial.

Since at least half of the total weight is multiplied by β, an analysis paralleling the analysis of the deterministic Weighted Majority algorithm [25] gives the following loss bound after tuning β (as in [4]) based on N and the loss B of the best combined list in hindsight:

$$2B + O\left(\sqrt{B \ln(N+1)} + \ln(N+1)\right).$$

There are two problems with this deterministic algorithm: It has the factor 2 in front of the comparator loss and we don't know how to generalize it to more than 2 lists. This is not surprising because we show in this paper that any deterministic algorithm can be forced to incur loss KB, where B is the loss of the best combined list. Nevertheless in the case of $K = 2$, the deterministic Weighted Median based algorithms perform well experimentally [28] and beat the previous deterministic heuristics developed in [26,27].

Probabilistic algorithms based on sampling. Our approach is to apply the Hedge algorithm [9] to the exponentially many combined lists. It uses the same exponential weights that are proportional to $\beta^{\boldsymbol{M}(\boldsymbol{c})}$ for list $\boldsymbol{c}$ but now we simply pick a combined list at random according to the mixture coefficients on all the combined lists. This ensures that the randomly chosen combined list is of the right size and hence circumvents the fact that the mixture does not represent a single combined list. The expected loss of the mixture is the mixture of the losses of the individual combined lists. When β is properly tuned as a function of the loss B of the best combined list and the total number of combined lists, then the probabilistic algorithms achieve expected loss $B + O(\sqrt{BK \ln N})$. Note that now the factor in front of the comparator loss B is one (whereas any deterministic algorithm can be forced have loss at least KB, see Section 5). The caveat of the probabilistic algorithms is that the list they predict with may change significantly between trials and in applications where there is a cost for changing the prediction, such algorithms are less useful. We discuss this issue again in the conclusion Section 7.

Hedge vs Follow the Perturbed Leader. The two main flavors of efficient online algorithms for dealing with a linear loss are Hedge [9] and Follow the Perturbed Leader [22]. Hedge based algorithms usually have slightly better loss bounds, whereas FPL-type algorithms typically have computational advantages. In this paper, completely contrary to those general rules, we present a Hedge-based algorithm that is fundamentally *faster* for our combined lists problem than the best-known FPL algorithm. The reason for this anomaly is that the computations for Hedge can be accelerated using the Fast Fourier Transform, whereas only a slower acceleration is known for finding the best combined list.

Outline of the paper. In most of the paper we assume that all base lists are disjoint, i.e. the requested item appears on at most one base list. This assumption lets us express the 0–1 loss/miss of a combined list as a sum over the initial segments

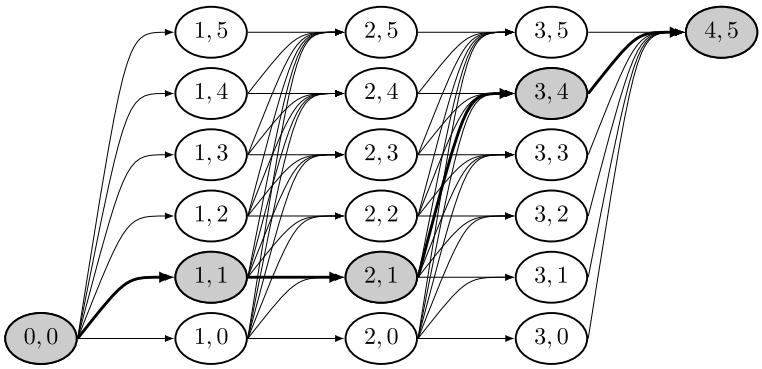

Fig. 4. Encoding combined lists ($K = 4$, $N = 5$) as paths. There is a 1–1 correspondence between $(0, 0) - (4, 5)$ paths in this graph and combined lists. The marked path corresponds to the combined list shown in Fig. 1. In general, the combined list $\boldsymbol{c} = (c_1, \ldots, c_K)$ corresponds to the path $(0, 0) - (1, c_1) - (2, c_1 + c_2) - \cdots - (k, \sum_{i=1}^{k} c_i) - \cdots - (K, N)$.

of the base lists. We describe the setting more formally in Section 2, and sketch the batch algorithm in Section 3. We then give our main randomized algorithm in Section 4 whose expected regret can be bounded by $O(\sqrt{BK \ln N})$, where B is the loss of the best combined list. It simulates the Hedge algorithm on the exponentially many combined lists. The straightforward implementation of this algorithm requires $O(KN^2)$ time per trial. However we found a way to speed up the algorithm using Fast Fourier Transforms for an improved time of $O(KN \ln N)$ per trial.

A number of lower bounds are given in Section 5. Any deterministic algorithm can be made to suffer loss at least KB, i.e. such algorithms cannot achieve sublinear regret. We also prove that any probabilistic algorithm can be forced to have regret at least $\Omega(\max(\sqrt{B \ln N}, \sqrt{BK}))$. This shows that our algorithm cannot be improved by much. (In fact we strongly believe it is optimal. Stronger lower bounds seem hard.)

In Section 6 we then address the case where duplicates are allowed between base lists. This apparently minor change in model has tremendous implications. In particular, we show that the question of whether there is a combined list with no misses is NP-hard. This implies that efficient algorithms with low regret do not exist (unless RP = NP). We then address the same problem with a surrogate loss. We "linearize" the loss much in the same way NP-completeness was avoided when learning disjunctions (by switching from 0–1 loss to attribute loss). In short, if the requested item occurs 5 times on the lists, then a combined list that hits this item 2 times now has loss 3. We can show that our algorithm now has a regret bound of $O(\sqrt{BK^2 \ln N})$ for the linearized loss. Note the additional factor of K which is due to the fact the range of the loss per trial is now $[0, K]$. We also discuss an alternate method for solving the problem with duplicates based on the online shortest path problem and using Component Hedge [23]. This algorithm achieves regret $O(\sqrt{BK \ln N})$ for the problem with duplicates (i.e. no additional range factor). However we do not know how to bound the running time for the iterative projection computation employed by the algorithm. We conclude with a number of open problems in the final section.

2. Setting

Recall that we have K base lists of N slots each. Until Section 6 we assume that each item appears at most once on all list. For $k \leqslant K$, we identify a partial *combined list* with a vector of counts $\boldsymbol{c} = (c_1, \ldots, c_k)$, where c_i denotes the number of items taken from the top of base list i. We denote the set of combined lists using n elements from k base lists by

$$\mathbb{C}_{n,k} := \left\{ \boldsymbol{c} \in \{0, \ldots, n\}^k \;\middle|\; \sum_{i=1}^{k} c_i = n \right\}.$$

We also think about $\mathbb{C}_{n,k}$ as the set of paths from $(0, 0)$ to (n, k) through the graph shown in Fig. 4. Representational issues aside, the number of combined lists is rather large (see Appendix A):

$$\ln |\mathbb{C}_{N,K}| = \ln \binom{N + K - 1}{K - 1} = \Theta\left(\max\left\{ K \ln \frac{N + K}{K}, N \ln \frac{N + K}{N} \right\} \right). \tag{1}$$

To appreciate the sheer number of lists, consider for example that $|\mathbb{C}_{10,20}| \approx 2 \cdot 10^7$, while $|\mathbb{C}_{100,20}| \approx 5 \cdot 10^{21}$.

Different trials can have different list contents and requested items. Note that if the requested item does not appear on any base list, then it will not appear on any combined list. Such trials will therefore not contribute to the regret, and will henceforth be ignored. What is important is where the requested item appears in the base lists and whether or not the combined list contains that position. In our model, the algorithm is charged a miss whenever its combined list does not contain the requested item. Therefore the combined list is treated as a *set* of items rather than a ranked list.

A single trial is summarized by a $(N + 1) \times K$ dimensional *miss matrix* $\boldsymbol{M}$, where for $0 \leqslant n \leqslant N$ and $1 \leqslant k \leqslant K$, the entry $M_{n,k}$ is one if the requested item appears on base list k in a position strictly greater than n and zero otherwise. The sum

$\boldsymbol{M}(\boldsymbol{c}) := \sum_{i=1}^{K} M_{c_i,i}$ is zero when combined list $\boldsymbol{c} \in \mathbb{C}_{N,K}$ contains the requested item, and one when $\boldsymbol{c}$ misses the requested item on the disjoint lists. We generalize this sum to partial combined lists $\boldsymbol{c} \in \mathbb{C}_{n,k}$ to $\boldsymbol{M}(\boldsymbol{c}) := \sum_{i=1}^{k} M_{c_i,i}$. Thus a partial combined list[3] that can only access the first $k < K$ base lists is not penalized for missing requested items that do not appear on these k lists. We often sum miss matrices over trials: $\boldsymbol{M}^t$ denotes the miss matrix for trial t and $\boldsymbol{M}^{<t}$ is the cumulative miss matrix before trial t, i.e. $M^{<t}_{c,k} := \sum_{s<t} M^s_{c,k}$. Therefore $\boldsymbol{M}^{<t}(\boldsymbol{c}) = \sum_{i=1}^{k} M^{<t}_{c_i,i}$ is the total number of misses made by the combined list $\boldsymbol{c} \in \mathbb{C}_{n,k}$ in the first $t-1$ trials.

We allow items to occur on multiple lists in Section 6. There, when $\boldsymbol{M}$ is a single trial miss matrix, $\boldsymbol{M}(\boldsymbol{c})$ can be greater than one. For example, if the requested item occurs on 5 different base lists and $\boldsymbol{c}$ has it twice, then $\boldsymbol{M}(\boldsymbol{c}) = 3$.

3. Batch algorithm

Let $\boldsymbol{M} = \sum_{t=1}^{T} \boldsymbol{M}^t$ denote the cumulative miss matrix after T trials. The hindsight-optimal list $\boldsymbol{c}^*$ and its loss B are given by

$$\boldsymbol{c}^* := \operatorname*{argmin}_{\boldsymbol{c} \in \mathbb{C}_{N,K}} \boldsymbol{M}(\boldsymbol{c}), \qquad B := \boldsymbol{M}(\boldsymbol{c}^*) = \min_{\boldsymbol{c} \in \mathbb{C}_{N,K}} \boldsymbol{M}(\boldsymbol{c}).$$

Brute-force evaluation of the minimum is intractable, viz (1). But we can exploit the structure of $\mathbb{C}_{N,K}$ and $\boldsymbol{M}(\boldsymbol{c})$ to compute $\boldsymbol{c}^*$ and B efficiently. Let $B_{n,k}$ denote the loss of the best combined list with n elements from the first k base lists:

$$B_{n,k} := \min_{\boldsymbol{c} \in \mathbb{C}_{n,k}} \boldsymbol{M}(\boldsymbol{c}).$$

Hence $B = B_{N,K}$. Now observe that $B_{\star,\star}$ satisfies the recurrence for $0 \leqslant n \leqslant N$:

$$B_{n,1} = M_{n,1} \quad \text{and} \quad B_{n,k} = \min_{0 \leqslant c \leqslant n} B_{n-c,k-1} + M_{c,k}, \quad \text{for } 1 < k \leqslant K.$$

By straightforward tabulation, the loss $B = B_{N,K}$ of the best combined list can be computed in time $O(KN^2)$. Interestingly, we can tabulate even faster. The column $B_{\star,k}$ is the infimal convolution[4] of $B_{\star,k-1}$ and $\boldsymbol{M}_{\star,k}$. The best-known algorithm for general infimal convolution is $O(\frac{N^2}{\ln N})$ due to [2]. In our setting, both $B_{\star,k-1}$ and $M_{\star,k}$ are non-increasing. However it is an open problem whether these special properties lead to an improved time bound. At any rate, once the full table $B_{\star,\star}$ is available we can recover the optimal combined list $\boldsymbol{c}^*$ in $O(KN)$ time from back to front as follows. The minimizer of the recurrence for $B_{N,K}$ is c^*_K. The minimizer of the recurrence for $B_{N-c^*_K,K-1}$ is c^*_{K-1}, and so on.

The above dynamic programming algorithm for computing the best combined list immediately leads to an online algorithm for lists, called Follow the Perturbed Leader (FPL) [22], which has small regret compared to the best list chosen in hindsight. At trial t, FPL adds a random perturbation matrix to $\boldsymbol{M}^{<t}$ and chooses the best combined list with respect to that perturbed miss matrix. For combinatorial prediction problems FPL is usually faster than Hedge, but has slightly weaker regret bounds [17]. Surprisingly we show in the next section that for combined list the Hedge algorithm is actually faster: it requires $O(KN \ln N)$ time per round instead of $O(\frac{KN^2}{\ln N})$ for FPL.

4. The randomized online algorithm based on sampling

In this section we develop an efficient randomized online algorithm and prove that its loss is not much larger than B, the loss of the best combined list chosen in hindsight. The algorithm is the well-known Hedge algorithm [9] where the possible combined lists function as the experts. The algorithm (implicitly) maintains weights proportional to $\beta^{\boldsymbol{M}(\boldsymbol{c})}$ for each combined list $\boldsymbol{c}$. There are two versions of the Hedge algorithm. The first one outputs a mixture vector over the experts and incurs loss equal to dot product between the mixture vector times the loss vector. Since the mixture vector is intractably large (see (1)) we have to use the second version. This version, like the Randomized Weighted Majority algorithm [25], outputs an expert drawn at random from the mixture. The loss is the loss of the drawn combined list and the goal is to bound the expected loss of the algorithm in relation to the loss of the best list chosen in hindsight.

Our contribution lies in the efficient implementation of the sampling version of the Hedge algorithm for combined lists. Following [30], we use the crucial fact that in our application, the loss $\boldsymbol{M}(\boldsymbol{c})$ of a combined list $\boldsymbol{c}$ decomposes into a sum: $\boldsymbol{M}(\boldsymbol{c}) = \sum_{k=1}^{K} M_{c_k,k}$. Thus the weight of $\boldsymbol{c}$ is proportional to the product $\prod_{k=1}^{K} \beta^{M_{c_k,k}}$. This property of the weights allows us to efficiently sample a combined list according to the mixture defined by the exponential weights of Hedge. The computation is based on dynamic programming, similar to the batch algorithm, but it is faster: $O(KN \ln N)$ instead of $O(\frac{KN^2}{\ln N})$ per trial.

Before showing this speedup, recall that the expected regret of the Hedge algorithm after tuning the learning rate β [9] is bounded by $\sqrt{2B \ln E} + \ln E$ where E is the number of experts with loss range $[0,1]$ per trial and B is the loss of the best expert. In our application, $\ln E$ is $O(K \ln \frac{N}{K})$ by (1), giving us the following regret bound:

[3] Notice the subtle but important difference in $\boldsymbol{M}(\boldsymbol{c})$ between partial combined lists in $\mathbb{C}_{N,k}$ that have only k components, and complete lists in $\mathbb{C}_{N,K}$ that have K components, but whose last $K-k$ components are all 0.

[4] Also known as $(\min, +)$ convolution, or min-convolution, or inf-convolution, or epigraphical sum.

Fact 4.1. *The regret of the tuned Hedge algorithm is as follows when the experts are combined lists formed from k base lists of size N and the loss of the best combined list is B*:

$$O\left(\sqrt{BK\ln\frac{N}{K}}+K\ln\frac{N}{K}\right).$$

Note that on each trial the loss of a combined list lies in $\{0,1\}$ since we assumed that the base lists are disjoint. Thus one can use the number of trials T as an overestimate of B and obtain regret bounds of the form $O(\sqrt{TK\ln\frac{N}{K}}+K\ln\frac{N}{K})$.

We now give the efficient implementation of the sampling. Let $\boldsymbol{M}=\boldsymbol{M}^{<t}$ denote the cumulative miss matrix before trial t. In trial t, Hedge samples combined list $\boldsymbol{c}\in\mathbb{C}_{N,K}$ with probability

$$\mathbb{P}(\boldsymbol{c})=\frac{\beta^{\boldsymbol{M}(\boldsymbol{c})}}{Z_{N,K}}\quad\text{where } Z_{n,k}:=\sum_{\boldsymbol{c}\in\mathbb{C}_{n,k}}\beta^{\boldsymbol{M}(\boldsymbol{c})}.$$

The first step of our method is to compute the normalization $Z_{n,k}$ for all problem sizes $1\leqslant k\leqslant K$ and $0\leqslant n\leqslant N$. To efficiently do this, we again derive a recurrence. For all $0\leqslant n\leqslant N$,

$$Z_{n,1}=\beta^{M_{n,1}}\quad\text{and}\quad Z_{n,k}=\sum_{c_k=0}^{n}Z_{n-c_k,k-1}\beta^{M_{c_k,k}}\quad\text{for } 1<k\leqslant K.$$

The right expression can be written more compactly in vector form as follows

$$Z_{\star,k}=Z_{\star,k-1}\circledast\beta^{\boldsymbol{M}_{\star,k}}\quad\text{for } 1<k\leqslant K$$

where $\circledast$ denotes real discrete convolution, i.e. $(x\circledast y)_n=\sum_i x_{n-i}y_i$. Since the convolution of two vectors of length N each can be performed in time $O(N\ln N)$ using the Fast Fourier Transform [7], we can tabulate $Z_{\star,\star}$ in time $O(KN\ln N)$.[5]

Once we have $Z_{\star,\star}$ tabulated, sampling a list $\boldsymbol{c}$ from $\mathbb{C}_{N,K}$ with probability proportional to $\beta^{\boldsymbol{M}(\boldsymbol{c})}$ is done as follows: First draw the last element $c_K=j$ with probability equal to $Z_{N-j,K-1}\beta^{M_{j,K}}/Z_{N,K}$. Then recursively sample the initial part of the list from $\mathbb{C}_{N-j,K-1}$. The probability of drawing the full $\boldsymbol{c}\in\mathbb{C}_{N,K}$ telescopes to $\beta^{\boldsymbol{M}(\boldsymbol{c})}/Z_{N,K}$ as desired. The sampling only takes $O(KN)$ time.

4.1. Relation to shortest path

We already mentioned in Section 2 that it is possible to identify combined lists with paths through the specific graph shown in Fig. 4. Therefore any algorithm that has small regret compared the best path chosen in hindsight is a competitor to our algorithm. The online shortest path problem has received considerable attention both in the full-information and in the bandit setting [30,22,1,5,23]. However, for that problem it is assumed that the adversary can control the loss of each individual edge and as a result any algorithm requires an update time that is at least on the order of the number of edges. This is not the case in our setting. We have $O(KN^2)$ edges in our graph, and yet we achieve update time $O(KN\ln N)$. How is this possible? First note that a miss matrix $\boldsymbol{M}$ has only KN parameters. Put another way, the losses of the edges are partitioned into equivalence classes of size $O(N)$ and all edges of the same class always have the same loss. That is, edge $(n,k)\to(n+i,k+1)$ and edge $(n',k)\to(n'+i,k+1)$ always have the same loss, because they both encode using i elements from base list k. This feature of our problem allows us to use convolutions and obtain update time that is lower than the number of edges in the shortest path formulation.

4.2. Computing expected loss and variance

We show how to compute two additional quantities: the expected loss of the Hedge algorithm and the variance of its loss. These quantities are of importance, because they are used in schemes for adaptively tuning the learning rate online. In particular, [6] introduces a parameter-free online tuning scheme based on the variance, for which the expected regret is at most of the order

$$\sqrt{\frac{B(T-B)}{T}\ln E},$$

where E is the number of experts, T is the number of trials and B is the loss of the best expert. This bound improves (4.1), which is obtained by taking the worst-case $T\to\infty$. Also [31] and the subsequent [8] give an alternate tuning scheme based

[5] A similar approach was used in [20] to obtain a similar speedup in the completely different context of computing the stochastic complexity of the Multinomial model.

on both the expected loss and $\ln Z_{N,K}$, for which the regret is shown to be constant in several natural settings including stochastic losses.

Note that when the number of experts is small, it is trivial to evaluate the expectation and variance of the Hedge loss. In our case of intractably many combined lists we again obtain efficient algorithms by evaluating recurrences using Fast Fourier Transforms.

4.2.1. Expected loss

Recall that $\boldsymbol{M}^{<t}$ denotes the cumulative miss matrix of all trials before trial t, and $\boldsymbol{M}^t$ denotes the miss matrix of trial t. The expected loss of our randomized algorithm in trial t is

$$\mathbb{E}\big[\boldsymbol{M}^t(\boldsymbol{c})\big] = \sum_{\boldsymbol{c}\in\mathbb{C}_{N,K}} \boldsymbol{M}^t(\boldsymbol{c})\mathbb{P}(\boldsymbol{c}) = \sum_{\boldsymbol{c}\in\mathbb{C}_{N,K}} \boldsymbol{M}^t(\boldsymbol{c})\frac{\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})}}{Z_{N,K}}.$$

To simplify the exposition, we first compute the *unnormalized expected loss*, which we define for problems of all sizes $0 \leqslant n \leqslant N$ and $1 \leqslant k \leqslant K$ as

$$L_{n,k} := \sum_{\boldsymbol{c}\in\mathbb{C}_{n,k}} \boldsymbol{M}^t(\boldsymbol{c})\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})}.$$

With this definition $\mathbb{E}[\boldsymbol{M}^t(\boldsymbol{c})] = L_{N,K}/Z_{N,K}$. The structure of combined lists allows us to write this as

$$L_{n,k} = \sum_{c_k=0}^{n} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \big(\boldsymbol{M}^t(\boldsymbol{c}) + M^t_{c_k,k}\big)\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})}\beta^{M^{<t}_{c_k,k}}.$$

We split the binary sum and expand the two parts individually. First

$$\sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \boldsymbol{M}^t(\boldsymbol{c})\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})} = \sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} L_{n-c_k,k-1}$$

and second

$$\sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} M^t_{c_k,k} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})} = \sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} M^t_{c_k,k} Z_{n-c_k,k-1}.$$

Now observe that both right-hand sides are convolutions. We derived

$$L_{\star,k} = L_{\star,k-1} \circledast \beta^{\boldsymbol{M}^t_{\star,k}} + Z_{\star,k-1} \circledast \beta^{\boldsymbol{M}^t_{\star,k}} \boldsymbol{M}^t_{\star,k}.$$

Computing $L_{N,K}$ requires two N-length convolutions for each base list, and can hence be done in $O(KN\ln N)$ time.

4.2.2. Variance

The variance of the loss in trial t can be expanded as usual:

$$\mathbb{V}\big[\boldsymbol{M}^t(\boldsymbol{c})\big] = \mathbb{E}\big[\boldsymbol{M}^t(\boldsymbol{c})^2\big] - \mathbb{E}\big[\boldsymbol{M}^t(\boldsymbol{c})\big]^2.$$

We already computed the expectation in the previous section. Here we focus on the expectation of the squared loss. To simplify the exposition, we first compute the *unnormalized expected squared loss*, which we define for all problem sizes $1 \leqslant k \leqslant K$ and $0 \leqslant n \leqslant N$ as

$$V_{n,k} := \sum_{\boldsymbol{c}\in\mathbb{C}_{n,k}} \boldsymbol{M}^t(\boldsymbol{c})^2\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})}.$$

This allows us to compute the variance using

$$\mathbb{V}\big[\boldsymbol{M}^t(\boldsymbol{c})\big] = V_{N,K}/Z_{N,K} - (L_{N,K}/Z_{N,K})^2.$$

Using the structure of combined lists, we obtain for all $k > 1$

$$V_{n,k} = \sum_{c_k=0}^{n} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \big(\boldsymbol{M}^t(\boldsymbol{c}) + M^t_{c_k,k}\big)^2\beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})+M^t_{c_k,k}}.$$

We expand the square, and rewrite the three cross terms individually. First

$$\sum_{c_k=0}^{n} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \boldsymbol{M}^t(\boldsymbol{c})^2 \beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})+M^t_{c_k,k}} = \sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} V_{n-c_k,k-1},$$

second

$$\sum_{c_k=0}^{n} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} 2\boldsymbol{M}^t(\boldsymbol{c}) M^t_{c_k,k} \beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})+M^t_{c_k,k}} = 2\sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} M^t_{c_k,k} L_{n-c_k,k-1}$$

and third

$$\sum_{c_k=0}^{n} \sum_{\boldsymbol{c}\in\mathbb{C}_{n-c_k,k-1}} \left(M^t_{c_k,k}\right)^2 \beta^{\boldsymbol{M}^{<t}(\boldsymbol{c})+M^t_{c_k,k}} = \sum_{c_k=0}^{n} \beta^{M^t_{c_k,k}} \left(M^t_{c_k,k}\right)^2 Z_{n-c_k,k-1}.$$

Again, each of these fragments is a convolution. Summing them up, we obtain

$$V_{\star,k} = V_{\star,k-1} \circledast \beta^{\boldsymbol{M}^t_{\star,k}} + 2L_{\star,k-1} \circledast \beta^{\boldsymbol{M}^t_{\star,k}} \boldsymbol{M}^t_{\star,k} + Z_{\star,k-1} \circledast \beta^{\boldsymbol{M}^t_{\star,k}} \left(\boldsymbol{M}^t_{\star,k}\right)^2.$$

The variance can hence also be computed in $O(KN\ln N)$ time.

5. Lower bounds

In the noise-free case (there is a combined list with no misses), the minimax regret for deterministic algorithms was proven to be $\Omega(K\ln N)$ with the first author's students in a class project [29]. We include a version in Section 5.1. Then we turn to the noisy case. We begin with a simple adversary argument against any deterministic algorithm in Section 5.2, and then present our two lower bounds for randomized algorithms in Sections 5.3 and 5.4.

5.1. Minimax analysis for the deterministic noise-free case

We consider a sequential game in which Learner chooses a combined list and Adversary requests an item slot. We restrain Adversary to ensure that there is a combined list with no loss. That is, once an item slot is requested, it is forced to be on the best combined list. At the start, each base list can contribute between 0 to N items to the best combined list, giving $N+1$ possible candidate cutoffs. If slot i is requested from some base list, then all slots above it are also forced to be on the best combined base list. Ensuring that these i slots are all in the best combined list means that there is no longer room for the slots at position $N-i+1$ and further down on all of the other base lists. This has the effect that on each base list there are now $N-i+1$ candidate cutoffs left – the same number for every base list. This simplifies the analysis greatly, since we only have to track this number of remaining candidate cutoffs per base list. Each miss results in a strictly reduced game, so after at most N misses the game is over.

Fix K. Let $V(n)$ denote the loss the Learner can be forced to incur in a game with $n+1$ candidate cutoffs remaining per base list. That is $V(0)=0$ and for $n\geqslant 1$

$$V(n) := \min_{\boldsymbol{c}} \max_{\substack{1\leqslant i\leqslant n\\ 1\leqslant k\leqslant K}} \mathbb{1}_{i>c_k} + V(n-i)$$

where $\boldsymbol{c}$ ranges over combined lists of size n. Each trial, Learner's goal is to minimize the number of remaining candidate cutoffs after a miss. To this end, he must spread his n items as equally as possible over the K lists, that is, taking either $\lfloor n/K\rfloor$ or $\lceil n/K\rceil$ items from the base lists. Adversary will then force a miss by choosing $i=\lfloor n/K\rfloor+1=\lceil (n+1)/K\rceil$. We find

$$V(n) = 1 + V\left(n - \left\lfloor \frac{n}{K}\right\rfloor - 1\right) = 1 + V\left(n - \left\lceil \frac{n+1}{K}\right\rceil\right).$$

Recall that the number of lists K is at least 2. We now define two monotone increasing functions, which we will show to be lower and upper bounds of $V(n)$, respectively:

$$L(n) := \log_{\frac{K}{K-1}}\left(\frac{n}{K}+1\right), \qquad U(n) := \log_{\frac{K}{K-1}}(n+1).$$

Note that L and U each solve a relaxation of the recurrence for V: $L(0)=U(0)=0$ and for any real $n\geqslant 1$,

$$L(n) = 1 + L\left(n - \frac{n}{K} - 1\right), \qquad U(n) = 1 + U\left(n - \frac{n+1}{K}\right). \tag{2}$$

The sandwich $L(n)\leqslant V(n)\leqslant U(n)$, for any integer $n\geqslant 0$, now follows by the following inductive argument: For the base case $n=0$, all three functions are 0, while for any integer $n\geqslant 1$,

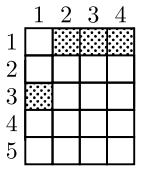

Fig. 5. For $N = 5$, $K = 4$ the special items are put in the marked positions. One must be missed by any combined list of size N.

$$V(n) = 1 + V\left(n - \left\lfloor \frac{n}{K} \right\rfloor - 1\right) \overset{\text{Induction}}{\geqslant} 1 + L\left(n - \left\lfloor \frac{n}{K} \right\rfloor - 1\right) \overset{\text{Monotonicity}}{\geqslant} 1 + L\left(n - \frac{n}{K} - 1\right) \overset{(2)}{=} L(n),$$

$$V(n) = 1 + V\left(n - \left\lceil \frac{n+1}{K} \right\rceil\right) \overset{\text{Induction}}{\leqslant} 1 + U\left(n - \left\lceil \frac{n+1}{K} \right\rceil\right) \overset{\text{Monotonicity}}{\leqslant} 1 + U\left(n - \frac{n+1}{K}\right) \overset{(2)}{=} U(n).$$

This shows that $V(N) \approx \ln_{\frac{K}{K-1}}(N) \approx K \ln(N)$. We now turn to the noisy case, where the best list can have loss.

5.2. Linear regret for deterministic learners

We first show a simple loss lower bound of KB against any deterministic learner, where $B \geqslant 0$ is the number of misses of the best combined list in hindsight. We show that this loss can be forced upon any learner, even if B is disclosed in advance. We assume that $N \geqslant K$. As illustrated by Fig. 5, the adversary tags the following K items as "special": item $N - K + 2$ from the 1st list and the top items from the remaining $K - 1$ lists. Any combined list of size N must miss at least one of these K special items because to contain them all would require a list of size $N - K + 2 + K - 1 = N + 1$. In each trial the adversary simply hits one of the special items missed by the combined list produced by the deterministic learner. Since combined lists have size N, at least one special item will be missed in each trial.

In T trials, the learner incurs T misses, but the number of misses of the best combined list is at most $\frac{T}{K}$. The reason is that the best combined list leaves out the special item with the lowest number of hits and the lowest number is at most $\frac{T}{K}$. So when the best has loss B then any deterministic learner can be forced to have loss at least KB, and thus regret at least $B(K - 1)$.

5.3. Lower bound using two lists

We now build techniques for proving our lower bound against any randomized learner in stages.

The basic component of our lower bound constructions will be the standard *binary prediction* game. There are two players, the Learner and the Adversary, and the game proceeds in rounds. Each round, the Learner chooses a prediction from {a, b}, and the Adversary chooses an outcome from {a, b}. Both players are allowed to randomize. The Learner suffers a unit of loss when his prediction does not match the outcome. The difficulty of the outcomes is controlled by the parameter B, which we will refer to as the budget: Adversary must ensure that there is at least one *static* prediction with loss $\leqslant B$. Since the budget will always be exhausted, B will as before be the loss of the comparator, i.e. the best decision in hindsight.

Fact 5.1. (*See e.g.* [21, Theorem 2.7.1].) *Let $g_2(B)$ be the largest total expected loss the Adversary can force the Learner to incur in the binary prediction game. For any integer budget B*

$$g_2(B) \geqslant B + \sqrt{\frac{B}{\pi}}.$$

Moreover, Adversary can inflict $g_2(B)$ in $2B + 1$ rounds.

We use this building block to prove a lower bound on the expected loss in the $K = 2$ list case of

$$B + \sqrt{\frac{B \log_2(N+1)}{\pi}}$$

against any randomized learner A. We do this by constructing a trial sequence using the above binary case as a building block on which A incurs at least this much loss while there is at least one combined list with loss at most B. For the sake of simplicity of the presentation we assume that $N = 2^S - 1$ for some integer S that divides the budget B.

The game consists of $S = \log_2(N+1)$ stages, each stage using up a proportion B/S of the loss budget and causing a loss of at least $g_2(B/S)$ to the learner. Therefore, the total loss of the learner will be at least the desired result

$$S\left(\frac{B}{S} + \sqrt{\frac{B}{S\pi}}\right) = B + \sqrt{\frac{BS}{\pi}}.$$

We have two lists, each with N elements. During stage 1, we access only two elements, the middle elements (those in position $(N+1)/2$ on the first and position $(N+1)/2$ on the second list). Notice that any combined list misses at least one of these middle elements. We now use the adversary from the binary prediction game with budget B/S to play these two middle items. This results in a sequence of accesses enforcing loss $g_2(B/S)$ on any Learner, while ensuring that one middle item is missed at most B/S times. Call this item the *selected item* of stage 1.

We now commit the selected item to be on the best combined list. Hence the loss of the best list during stage 1 is at most B/S. Say the selected item is on list 1. This means we have removed from consideration the first half of elements of list 1 (they are included) and the last half of elements of list 2 (they are not included).

During stage 2 we similarly select between the middle element of the second half of list 1, and the middle element of the first half of list 2. Suppose we now decide in favor of list 2. This means that we have decided to include at least $(N+1)/2$ elements from list 1, at least $(N+1)/4$ elements from list 2, and $(N-3)/4$ elements are yet to be decided. We continue this process, halving the remaining range at each stage, until only one good combined list remains. This argument proves the following lower bound:

Theorem 5.2. *Let $N+1$ be a power of 2 and B be a budget that is divisible by $\log_2(N+1)$. Then for any randomized online learner A for the $K=2$ list problem there is a sequence of requests on which A incurs an expected loss at least*

$$B + \sqrt{\frac{B\log_2(N+1)}{\pi}}$$

but at least one combined list has at most B misses.

Note that the lower bound construction reuses the same base list contents in each trial and the base lists are disjoint.

The lower bound for $K=2$ also holds when $K>2$, since the 2 special items that the Adversary uses in each trial retain the property that no combined list over $K>2$ lists can include them both. Increasing K does introduce many combined lists that include neither, but playing those is clearly suboptimal for the Learner against this 2-list adversary.

We now turn to our second lower bound, which does make nontrivial use of $K>2$ lists.

5.4. Lower bound using the first row of items

In this section we construct an adversary that only hits the first items of the base lists. The adversary partitions the lists into k *pairs*, (so that $K=2k$) each of length $N=k$. In this section we will call our base lists $\{1\mathrm{a}, 1\mathrm{b}, \ldots, k\mathrm{a}, k\mathrm{b}\}$ instead of $\{1,\ldots,K\}$. The target combined list will contain one element from each pair (the top element from either of the lists).

Recall that c_i is the random variable indicating the number of elements on list i taken by the learner. At each trial, define the *usage* u_j to be the learner's expected number of slots taken from pair j:

$$u_j := \mathbb{E}(\#\text{ elements selected from the pair}) = \mathbb{E}(c_{j\mathrm{a}} + c_{j\mathrm{b}})$$

where $\mathbb{E}$ denotes expectation with respect to the distribution $\mathbb{P}$ over combined lists used by the learner at the trial. Note that the distribution $\mathbb{P}$ used by the learner depends on the learner and the entire past.

Some immediate consequences of these definitions:

1. Since the Learner predicts with an N-element combined list, $\sum_{i=1}^{k} u_i = N$.
2. If $u_j \leqslant 1+\epsilon$ (for $\epsilon>0$) then for list pair j

$$\mathbb{P}(c_{j\mathrm{a}} \geqslant 1) + \mathbb{P}(c_{j\mathrm{b}} \geqslant 1) \leqslant 1+\epsilon.$$

Given a K-list mistake budget B, the adversary uses a parameter ϵ and simulates k binary prediction problems with mistake budget B/k (assume this is an integer) one for each pair. A pair is *live* if its mistake budget has not been reached, and *dead* if its mistake budget has been reached. The adversary will keep on playing as long as there are live pairs. At the end of the process, the target combined list will contain from each pair the first item from the list corresponding to the prediction with least loss.

On each trial the learner produces its distribution $\mathbb{P}$ over combined lists, which is known to the adversary. The adversary then:

1. Picks any live pair j with usage $u_j \leqslant 1+\epsilon$. Play Lemma 5.3 adversary on that pair, or
2. Picks any dead pair i with usage $u_j \leqslant 1-\epsilon/(k-1)$. Use the item known to be in the target.

Note that at least one of these cases must hold as long as there is at least one live pair, since

$$1+\epsilon+(k-1)\left(1-\frac{\epsilon}{k-1}\right) = N = \sum_j u_j.$$

Also note that each occurrence of case 1 brings its game one step closer to the end. Hence case 1 cannot occur infinitely often.

In case 1, by Lemma 5.3 below, the total expected loss incurred while the pair is live is at least $(1-2\epsilon)(B/k+\sqrt{B/(k\pi)})$. In case 2, the (expected) loss on the trial is at least $\epsilon/(k-1)$, since with probability at least $\epsilon/(k-1)$ no element from the list pair is chosen by the algorithm.

If the learner plays so that some pair remains alive forever, then case 2 occurs infinitely often and the learner incurs a constant loss (at least $\epsilon/(k-1)$) an unbounded number of times. Thus, to guarantee finite regret any learner must eventually let all pairs die. When all pairs are dead, the total expected loss is thus at least

$$k(1-2\epsilon)\big(B/k+\sqrt{B/(k\pi)}\big) = (1-2\epsilon)(B+\sqrt{Bk/\pi}).$$

We have obtained an adversary for each small (but nonzero) ϵ, and hence no algorithm can have an expected loss less than the supremum of these loss lower bounds, i.e.

$$B+\sqrt{B\lfloor K/2\rfloor/\pi}$$

on every sequence of requests where the best combined list has loss at most B.

Interestingly, the argument does not work for the "limit adversary" with $\epsilon=0$. To see this, consider the learner that on trial t uses a randomized combined list where the first pair has usage $u_1 = 1-2^{-t}$ and each other pair has usage $u_j = 1+2^{-t}/(k-1)$. The limit adversary always plays the first pair. After the first pair dies, the learner incurs at most loss $1+1/2+1/4+\cdots\leqslant 2$, even over an infinite series of trials. So we get the lower bound for one pair, not k.

We now prove the main lemma showing that regret lower bounds for the binary prediction setting can be transferred to what we call the *restricted powerset* game. In the binary prediction setting the learner chooses a prediction from {a, b}, the adversary chooses an outcome from {a, b}, and the learner incurs a unit of loss if the prediction and outcome differ. In the $(1+\epsilon)$-*powerset* setting the learner chooses a subset of {a, b} (i.e. the prediction can be neither, a, b or both), with the restriction that $\mathbb{P}(\text{a})+\mathbb{P}(\text{b})\leqslant 1+\epsilon$. The adversary again chooses an outcome as before, and the learner now suffers a unit of loss when its prediction does not include the outcome.

Lemma 5.3. *Let $g_2(B)$ be the largest total expected loss the Adversary can force the Learner to incur in the binary prediction game. There is an adversary that forces expected loss $g_2(B)(1-2\epsilon)$ in the $(1+\epsilon)$-powerset setting while keeping the cumulative loss of the best static* singleton *prediction at most B. This loss can be forced in $2B+1$ rounds.*

Proof. Consider an arbitrary $(1+\epsilon)$-powerset learner. Recall that the expected loss of this learner is the probability that it does not include the correct outcome.

Use a trial-by-trial simulation of the binary prediction setting adversary. Consider an arbitrary trial and let $p_\text{a}=\mathbb{P}(\text{a})$ and $p_\text{b}=\mathbb{P}(\text{b})$ be the probabilities chosen by the powerset learner (so $p_\text{a}+p_\text{b}\leqslant 1+\epsilon$).

Since increasing the p_i's can only decrease the expected loss, we assume WLOG that $p_\text{a}+p_\text{b}=1+\epsilon$. Now, since each $p_i\leqslant 1$, each $p_i\geqslant\epsilon$. Let

$$\hat{p}_\text{a} := \begin{cases} p_\text{a}-\epsilon & \text{if } p_\text{a}\leqslant p_\text{b}, \\ p_\text{a} & \text{if } p_\text{a}>p_\text{b} \end{cases} \quad\text{and}\quad \hat{p}_\text{b} := \begin{cases} p_\text{b} & \text{if } p_\text{a}\leqslant p_\text{b}, \\ p_\text{b}-\epsilon & \text{if } p_\text{a}>p_\text{b}. \end{cases}$$

Note that this ensures that $\hat{p}_\text{a}+\hat{p}_\text{b}=1$ by decreasing the *smallest* p_i.

Have the binary prediction learner use $(\hat{p}_\text{a},\hat{p}_\text{b})$ as its distribution over the two predictions. Assume the binary prediction adversary chooses outcome a; the other case is symmetrical. The expected loss in the binary prediction game is $\hat{p}_\text{b}$. The adversary in the powerset setting uses the same outcome, so the expected loss of the powerset learner is

$$\mathbb{P}(\text{a not in set}) = 1-p_\text{a} = p_\text{b}-\epsilon.$$

If $p_\text{b}\leqslant p_\text{a}$ then $\hat{p}_\text{b}=p_\text{b}-\epsilon$ and the binary and powerset losses on the trial are equal. If $p_\text{b}>p_\text{a}$ then $\hat{p}_\text{b}=p_\text{b}>1/2$. This means that the binary setting loss is:

$$p_\text{b}-\epsilon=\hat{p}_\text{b}-\epsilon\geqslant\hat{p}_\text{b}(1-2\epsilon)$$

and is at least $(1-2\epsilon)$ times the binary setting loss. Note that the cumulative loss of each prediction in the binary setting is the same as the cumulative loss of the corresponding singleton prediction in the powerset setting. Accumulating this over all trials gives that the total loss of the $(1+\epsilon)$-powerset learner is at least $(1-2\epsilon)g_2(B)$ as desired. □

Extension to larger N. The preceding construction can be extended to lengths $N>k$ by adding another base list, called the *padding list*, so now $K=2k+1$. We include the first $N-k$ items from the padding list in the target combined list. When the learner's total usage of the k pairs exceeds $k+\epsilon$, the adversary requests the *padding item*, which sits in the padding list at position $N-k$. This padding item must then have probability below $1-\epsilon$, so the learner incurs regret at least ϵ on the trial.

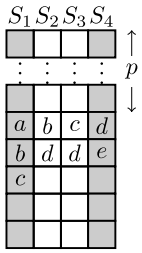

Fig. 6. Reduction for Set Cover problem with $m = 2$, $U = \{a, b, c, d, e\}$, $\mathcal{C} = \{S_1, S_2, S_3, S_4\}$ where $S_1 = \{a, b, c\}$, $S_2 = \{b, d\}$, $S_3 = \{c, d\}$ and $S_4 = \{d, e\}$. Cover shown in gray.

5.5. Discussion

We already observed that the above $\Omega(\sqrt{B \ln N})$ expected regret bound for any randomized learner holds for all $K \geqslant 2$ lists, and that the $\Omega(\sqrt{BK})$ expected regret bound holds for lists of length $N \geqslant K/2$. We conjecture that the lower bound on the expected regret is $\Omega(\sqrt{BK \ln(N/K)})$, i.e. that the expected regret bound of the learner of Section 4 is optimal within a constant factor.

6. Combining lists with duplicates

We now turn to the scenario where the base lists are not disjoint any more, i.e. items can occur on multiple lists. This apparently minor difference in the setup has major implications. For one, finding the list that minimizes the number of misses is NP-hard, even if the base lists contain the same items in every trial. So we cannot hope for efficient algorithms with low regret unless RP = NP. We sketch a reduction from Set Cover in Section 6.1 and then work around this hardness result in Section 6.2 by linearizing the loss.

In the presence of duplicates we may either ask the learner to pick N distinct items (possibly from $> N$ slots), or to choose N item slots (possibly containing $< N$ distinct items). We choose the latter rule for simplicity. Our hardness result holds for both.

6.1. NP-hardness by reduction from Set Cover

An instance of the Set Cover problem consists of a collection $\mathcal{C}$ of subsets of some universe U, and a number m. The question is whether a subcollection $\mathcal{D} \subseteq \mathcal{C}$ of size m exists such that the union of the subcollection is U. We transform this instance into a zero miss combined list existence problem with $K = |\mathcal{C}|$ base lists of size $N = m(p + |U|)$ and a sequence of $|U|$ requests. For each subset $S \in \mathcal{C}$, we introduce a base list that starts with p many padding items, then includes item x for each $x \in S$, and ends with padding to length N. By design, N is such that a combined list can contain all non-padding items from m base lists. To ensure that it cannot contain non-padding items from $m + 1$ base lists, we must have that $(m + 1)(p + 1) > N$. We hence fix the least such p, which equals $m(|U| - 1)$. The request sequence hits x for each $x \in U$.

If an m-cover exists, there is a combined list with zero loss. (See Fig. 6.) If not, then any combined list must miss at least one item. So an m-cover exists iff there is a combined list without any misses. Also unless RP = NP, there cannot exist a polynomial time randomized algorithm with polynomial expected loss on request sequences for which there exists a combined list with no misses. (By polynomial we mean polynomial in N and K.)

If such an algorithm existed and achieved regret $p(N, k)$ then one could present it with random requests chosen with replacement from the given request sequence. If there does not exist a combined list with zero misses, then for such random request the expected loss must be at least 1 over the length of the request sequence. By presenting the algorithm with polynomially many random requests, the expected loss is either bounded by $p(N, k)$ or the expected loss must grow linearly in the number of requests. This clearly can be used to design a polynomial-time randomized decision procedure for the combined list problem we just showed to be NP-complete.

6.2. Working around the hardness

When the lists are disjoint then in any trial and for any combined list $\boldsymbol{c} \in \mathbb{C}_{N,K}$, the miss count $\boldsymbol{M}(\boldsymbol{c})$ is either 0 or 1 and the 0/1 loss is identical to $\boldsymbol{M}(\boldsymbol{c})$. When items can appear on multiple list, then $\boldsymbol{M}(\boldsymbol{c})$ can be larger than 1, whereas the 0/1 loss is $I(d - \boldsymbol{M}(\boldsymbol{c}))$, where $I(h) = 1$ if $h = 0$ and 0 if $h \geqslant 1$, and d is the number of times the requested item appears in the base lists. The reduction in Section 6.1 shows that we cannot hope for an efficient algorithm with small expected regret measured by the 0/1 loss when the items can appear on multiple lists. Observe that $I(d - \boldsymbol{M}(\boldsymbol{c})) \leqslant \boldsymbol{M}(\boldsymbol{c})$. Therefore, in this section we propose to replace the 0/1 loss $I(d - \boldsymbol{M}(\boldsymbol{c}))$ by its upper bound $\boldsymbol{M}(\boldsymbol{c})$. This amounts to linearizing the loss because $\boldsymbol{M}(\boldsymbol{c}) = \sum_{k=1}^{K} M_{c_k,k}$. Note that the 0/1 loss $I(d - \boldsymbol{M}(\boldsymbol{c}))$ decidedly does not decompose into a sum over the component of $\boldsymbol{c}$.

This approach parallels how NP-hardness was avoided when learning disjunctions. There the 0/1 loss was replaced by the attribute loss which decomposes into a sum of the literals in the disjunction [24]. This linearization of the loss is extensively discussed in [30].

Our efficient implementation of Hedge is based on the decomposition of the miss count and remains unchanged in the setting when items can appear on multiple lists. However, $\boldsymbol{M}(\boldsymbol{c})$ now lies in $[0, K]$ and the straightforward regret bounds have an additional factor of K due to the extended range. This means that in the case of combined lists the resulting regret bound for the linearized loss has leading term

$$\sqrt{2BK \ln |\mathbb{C}_{N,K}|} \approx \sqrt{2BK\left(K \ln \frac{N+K}{K} + N \ln \frac{N+K}{N}\right)}.$$

The additional range factor also appears in the regret of the FPL algorithm. Several method have been developed for eliminating the range factor. First, the range factor can be eliminated if the so called *unit rule* holds for the given loss and algorithm [23]. However, there are simple counter examples for combined lists. Second, we can change the algorithm to the Component Hedge algorithm which has a range factor free regret bound with a leading term of

$$\sqrt{2B\left(K \ln \frac{N+K}{K} + N \ln \frac{N+K}{N}\right)}.$$

This bound will be derived in Appendix B.

7. Open problems

In summary, we hope that we opened up a new approach for combining lists (of the same size N). We invented a new notion of mixture of such base lists in which the sizes of the top segments that were chosen from each list sum to N. We developed an efficient algorithm that implicitly maintains a mixture over exponentially many combined lists and proved regret bounds that are tight within a factor of $\min(\sqrt{K}, \sqrt{\ln(N/K)})$. Besides proving a tight lower bound for the $K > 2$ list case (notoriously hard), our approach leaves many open questions:

- We have ignored the question of how to rank the items within the combined list. Our notion of loss simply charges one unit depending on whether the requested item is in the combined list or not. Ideally, the combined list should be ranked and we would like to develop online algorithms for the case when the cost is proportional to how close each request is to the top of the list. One idea is to overlay N algorithms optimized for combined list sizes $1, 2, \ldots, N$. Intuitively the item i on the list would miss the combined lists of size $1, 2, \ldots, i-1$ and incur loss proportional to $i-1$. However this approach will only work when the combined lists of size less than i are contained in the list of size i. So far we were not able to achieve good regret bounds with this approach.
- In caching applications one could need generalizations of our algorithms to the case when the items have different sizes, but the total size of a combined list is N. Here we expect to run into various online packing problems, as studied e.g. in [12].
- In caching applications it costs to change the "real cache" because items need to be reloaded. Our online algorithms currently ignore the reloading costs and this is particularly egregious for the probabilistic algorithms. The Shrinking Dartboard Algorithm [14], a method for lazily updating the expert that is followed by the Hedge algorithm seems quite readily applicable to our setting. The Follow-the-Lazy-Leader algorithm [22] is another method requiring fewer updates. Also some good practical heuristics for reloading were given in [13,15]. However no performance guarantees were provided for these heuristics.
- In this paper we implement the Hedge algorithm by maintaining exponentially many weights implicitly. There is a second method, developed in [33,19,23], that represents each of the exponentially many experts by its binary "component usage" vector (see Appendix B). In our case we would have one component per base list k and per count c, which, if set to one, would indicate that the combined list uses c items from base list k. The resulting "Component Hedge" algorithm for our list problem then maintains one weight for each of the $K(N+1)$ many components. The weight vector must lie in a polytope which is the convex hull of the component usage vectors of the possible combined lists.
  As we will discuss in the next bullet, the advantage of the Component Hedge is that it can be adapted to the shifting target case more easily. However this algorithm is much more complex than the simple dynamic programming algorithm used in this paper that employs FFT for efficiency.
- At this point the algorithms of this paper are designed to achieve small regret compared to the best fixed combined list chosen in hindsight. In the expert setting there is a long history of algorithms that can handle the case when the best expert is allowed to shift over time. This is achieved by first doing an exponential update and then mixing in a bit of either the uniform distribution over all experts or the average of all past distributions over the experts [18,3,10]. In all experimental evaluations that the authors are aware of, it was crucial to extend the online algorithms to the shifting expert case [16,15].

It is a tedious but straightforward exercise to mix in a bit of the uniform distribution into the dynamic programming algorithm of Section 4, thus implementing the Fixed Share algorithm from [18]. The method is again based on recurrences, and maintains all weights implicitly. However, the overall running time changes from $O(T)$ to $O(T^2)$.[6] We don't know how to do the fancier method efficiently, which mixes in a bit of the past average distribution. The reason is that the exponential weight updates on the combined lists seem to be at loggerheads with mixing in the past average weight. The resulting update is neither multiplicative nor additive and makes it difficult to implicitly maintain the weights.

In this respect Component Hedge [23] may have the advantage (see Appendix B), as mixing in the past average usage vector can be done in constant time per component per trial. However, for this approach to be viable, an efficient implementation of the required relative entropy projections must be found. It is still an open problem how to combine, in a simple and efficient way, the mixing of the past average method with the main algorithm of this paper (efficient Hedge for k-lists).

- This paper is theoretical in that we focus on models for which we can prove regret bounds. However it would be useful to do practical experiments of the type done in [13,15]. This would require us to blend the methods developed here with heuristics for handling for example the reloading issue.

Acknowledgements

Thanks to Anindya Sen and Corrie Scalisi for valuable ground work on this problem as part of a class and a Master's project [29,28]. Thanks to Jyrki Kivinen for generous brainstorming.

Appendix A. Number of lists: proof of (1)

Throughout assume $K \geqslant 2$ and $N \geqslant 1$. For the lower bound

$$\ln \binom{N+K-1}{K-1} \geqslant (K-1)\ln\frac{N+K-1}{K-1} \geqslant (K-1)\ln\frac{N+K}{K} \geqslant \frac{K}{2}\ln\frac{N+K}{K}$$

and

$$\ln \binom{N+K-1}{K-1} \geqslant N\ln\frac{N+K-1}{N} \geqslant \left(\min_{N\geqslant 1, K\geqslant 2} \frac{\ln\frac{N+K-1}{N}}{\ln\frac{N+K}{N}}\right) N\ln\frac{N+K}{N} = \frac{N}{2}\ln\frac{N+K}{N}.$$

For the upper bound, we have

$$\ln \binom{N+K-1}{K-1} \leqslant \ln \binom{N+K}{K} \leqslant K\ln\frac{N+K}{K} + N\ln\frac{N+K}{N} \leqslant 2\max\left\{K\ln\frac{N+K}{K}, N\ln\frac{N+K}{N}\right\}.$$

The first inequality follows from the log-convexity of the gamma function. The second overestimates the log-choose by the binary entropy.

Appendix B. Component Hedge

There are two natural extensions of the combined list setting in which the Component Hedge [23] algorithm may be more suited than Hedge:

1. Loss range of $[0, 1]$ per item slot. We saw that in the presence of duplicates between base lists the loss range of a combined list becomes $[0, K]$. In general we may even allow it to be $[0, N]$. The Hedge algorithm may not be appropriate for this extension because its regret bound now includes an undesirable range factor. Component Hedge avoids the range factor.
2. Competing with a shifting target combined list. Here we measure the regret w.r.t. a *sequence* of combined lists with few switches. Typical algorithms for shifting [18,3,10] operate by mixing weight vectors on experts. Unfortunately, our recurrence-based implementation does not allow mixing. It can be bolted on externally for an additional overhead of $O(t)$ per trial, leading to an overall quadratic algorithm. Component Hedge maintains its weights explicitly, and these can hence be mixed. So although Component Hedge is more expensive per trial, a shifting algorithm based on it would need only constant time per trial.

In this appendix we analyze the regret bound for Component Hedge on combined lists. The standard symmetry argument of [23] does not apply: the item slots can not be freely permuted due to the initial segment constraint. Instead we optimize directly.

[6] The $O(T^2)$ factor may be improved to $O(T\log T)$ by employing the techniques of [11].

Component Hedge maintains its uncertainty as a *usage vector*, i.e. an element of the convex hull of the $(N+1) \times K$ indicator matrices corresponding to the combined lists. The weight update involves a relative entropy projection onto this convex hull, which can be expressed using few linear equality constraints as a linear image of the flow polytope. This projection can be computed using a convex optimization library. One could also apply the iterative algorithm of [23], but its convergence properties have not yet been analyzed.

We identify each combined list $\boldsymbol{c}$ with its $(N+1) \times K$ *indicator vector*, which takes values in $\{0,1\}$ and has $c_{n,k} = 1$ iff combined list $\boldsymbol{c}$ uses exactly n items from base list k. Let $\mathbb{U} \subseteq [0,1]^{(N+1)\times K}$ denote the convex hull of (indicator vectors of) combined lists. Component Hedge (CH) maintains a weight vector $\boldsymbol{c}$ in $\mathbb{U}$. The initial weight vector is a parameter of the algorithm called the *prior*. We will discuss how to set it below. Each round CH operates as follows:

1. First CH issues a prediction. To this end, it decomposes the usage vector $\boldsymbol{c}$ into a convex combination of indicator vectors $\boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v}_i$, where the weights $\boldsymbol{\alpha}$ form a probability distribution. Such a decomposition always exists, but it need not be unique. CH then randomly plays the combined list associated to indicator vector $\boldsymbol{v}_i$ with probability α_i.
2. Second the miss matrix $\boldsymbol{M}$ is revealed, and CH updates the weights to $\boldsymbol{c}'$, where

$$\boldsymbol{c}' := \inf_{\boldsymbol{u}} \Delta(\boldsymbol{u} \| \boldsymbol{c}) + \eta \boldsymbol{M}(\boldsymbol{u})$$

where $\boldsymbol{u}$ ranges over all usage vectors, $\Delta(\boldsymbol{u}\|\boldsymbol{c})$ is the relative entropy

$$\Delta(\boldsymbol{u}\|\boldsymbol{c}) = \sum_{n=0}^{N} \sum_{k=1}^{K} u_{n,k} \ln \frac{u_{n,k}}{c_{n,k}}.$$

η is a parameter called the *learning rate* and $\boldsymbol{M}(\boldsymbol{u}) = \sum_{n=0}^{N} \sum_{k=1}^{K} u_{n,k} M_{n,k}$ is the expected loss when playing a combined list according to weights $\boldsymbol{u}$. This update step is called the *projection step*. Computing $\boldsymbol{c}'$ is a convex optimization problem, for which a numerical library may be used.

Hence for T rounds Component Hedge will issue T decompositions and T projections. To implement shifting, we may between trials perform a *share update* akin to [18], that is, update the usage vector to a convex combination of the current usage vector and the prior usage vector. We may also add in a bit of the past average as in [3]. In either case, this additional step is cheap and hence does not affect the time complexity.

It is shown in [23] that the expected regret of CH with prior $\boldsymbol{u}$ and properly tuned learning rate η compared to any combined list $\boldsymbol{c}$ with loss B is bounded by

$$\sqrt{2B\,\Delta(\boldsymbol{c}\|\boldsymbol{u})} + \Delta(\boldsymbol{c}\|\boldsymbol{u}).$$

To get a good regret bound for Component Hedge, we need to choose a smart prior usage vector. The part of the regret bound that is affected by the choice of prior is the relative entropy term $\Delta(\boldsymbol{c}\|\boldsymbol{u})$, where $\boldsymbol{c}$ is the 0/1 usage vector of the comparator, and $\boldsymbol{u}$ is the usage vector of the chosen prior. The best prior is the one attaining

$$\inf_{\boldsymbol{u}\in\mathbb{U}} \sup_{\boldsymbol{c}\in\mathbb{U}} \Delta(\boldsymbol{c}\|\boldsymbol{u}) = \inf_{\boldsymbol{u}\in\mathbb{U}} \sup_{\boldsymbol{c}\in\mathbb{U}} \sum_{n=0}^{N} \sum_{k=1}^{K} (\overbrace{c_{n,k} \ln c_{n,k}}^{0} - c_{n,k} \ln u_{n,k}).$$

By Sion's minimax theorem, we may exchange inf and sup, and then rewrite

$$\begin{aligned}
\inf_{\boldsymbol{u}\in\mathbb{U}} \sup_{\boldsymbol{c}\in\mathbb{U}} -\sum_{n=0}^{N} \sum_{k=1}^{K} c_{n,k} \ln u_{n,k} &= \sup_{\boldsymbol{c}\in\mathbb{U}} \inf_{\boldsymbol{u}\in\mathbb{U}} -\sum_{n=0}^{N} \sum_{k=1}^{K} c_{n,k} \ln u_{n,k} \\
&= \sup_{\boldsymbol{c}\in\mathbb{U}} \inf_{\boldsymbol{u}\in\mathbb{U}} \Delta(\boldsymbol{c}\|\boldsymbol{u}) - \sum_{n=0}^{N} \sum_{k=1}^{K} c_{n,k} \ln c_{n,k} \\
&= \sup_{\boldsymbol{c}\in\mathbb{U}} \underbrace{-\sum_{n=0}^{N} \sum_{k=1}^{K} c_{n,k} \ln c_{n,k}}_{:=H(\boldsymbol{c})}.
\end{aligned}$$

That is, we are looking for the usage that maximizes the entropy. To find an upper bound on the maximal entropy, we relax the constraint $\boldsymbol{c} \in \mathbb{U}$ to the following:

$$\sum_{n=0}^{N} c_{n,k} = 1 \quad \text{for all } k \quad \text{and} \quad \sum_{n=0}^{N} \sum_{k=1}^{K} n c_{n,k} = N. \tag{B.1}$$

The quantity of interest can now be computed by the method of Lagrange multipliers, which yields

$$\sup_{\boldsymbol{c} \text{ s.t. (B.1)}} H(\boldsymbol{c}) = \inf_{\mu} \left(K \ln \sum_{i=0}^{N} e^{\mu i} - \mu N \right).$$

It is difficult to determine the optimal μ analytically. However, for an upper bound we may substitute the suboptimal choice $\mu = \ln \frac{N}{N+K-1}$, in which case the value becomes

$$K \ln \frac{N+K-1-N(N/(N+K-1))^N}{K-1} - N \ln \frac{N}{N+K-1} \leqslant K \ln \frac{N+K-1}{K-1} + N \ln \frac{N+K-1}{N}.$$

Now observe that the bound for Hedge features the term

$$\ln \binom{N+K-1}{K-1} \leqslant (K-1) \ln \frac{N+K-1}{K-1} + N \ln \frac{N+K-1}{N}.$$

In summary we get the following regret bound:

Fact B.1. *The regret of the tuned Component Hedge algorithm is as follows when learning combined lists formed from K base lists of size N, the usage prior is chosen as above, and the loss of the best combined list is at most B:*

$$\sqrt{2B\Delta} + \Delta, \quad \textit{where } \Delta = K \ln \frac{N+K-1}{K-1} + N \ln \frac{N+K-1}{N}.$$

Note that this regret bound for Component Hedge bound is very close to the regret bound for the Hedge algorithm (Fact 4.1).

References

[1] J. Abernethy, E. Hazan, A. Rakhlin, Competing in the dark: An efficient algorithm for bandit linear optimization, in: Proceedings of the 21st Annual Conference on Learning Theory, July 2008.
[2] D. Bremner, T.M. Chan, E.D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, P. Taslakian, Necklaces, convolutions, and $x+y$, in: Algorithms – ESA 2006, in: Lect. Notes Comput. Sci., vol. 4168, Springer, 2006, pp. 160–171.
[3] O. Bousquet, M.K. Warmuth, Tracking a small set of experts by mixing past posteriors, J. Mach. Learn. Res. 3 (2002) 363–396.
[4] N. Cesa-Bianchi, Y. Freund, D. Haussler, D.P. Helmbold, R.E. Schapire, M.K. Warmuth, How to use expert advice, J. ACM 44 (3) (May 1997) 427–485.
[5] N. Cesa-Bianchi, G. Lugosi, Combinatorial bandits, in: Proceedings of the 22nd Annual Conference on Learning Theory, June 2009.
[6] N. Cesa-Bianchi, Y. Mansour, G. Stoltz, Improved second-order bounds for prediction with expert advice, Mach. Learn. 66 (2007) 321–352.
[7] J.W. Cooley, J.W. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (90) (1965) 297–301.
[8] S. de Rooij, T. van Erven, P.D. Grünwald, W.M. Koolen, Follow the leader if you can, hedge if you must, arXiv, January 2013.
[9] Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1997) 119–139.
[10] A. György, T. Linder, G. Lugosi, Tracking the best of many experts, in: Proceedings of the 18th Annual Conference on Learning Theory, Springer, 2005, pp. 51–74.
[11] A. György, T. Linder, G. Lugosi, Efficient tracking of large classes of experts, in: Proceedings of the IEEE International Symposium on Information Theory, ISIT, 2012.
[12] A. György, G. Lugosi, G. Ottucsàk, On-line sequential bin packing, J. Mach. Learn. Res. 11 (January 2010) 89–109.
[13] R.B. Gramacy, Adaptive caching by experts, PhD thesis, University of California at Santa Cruz, 2003.
[14] S. Geulen, B. Vöcking, M. Winkler, Regret minimization for online buffering problems using the Weighted Majority algorithm, in: Proceedings of the 23rd Annual Conference on Learning Theory, 2010.
[15] R.B. Gramacy, M.K. Warmuth, S.A. Brandt, I. Ari, Adaptive caching by refetching, in: Proceedings of the 15th Annual Conference on Neural Information Processing Systems, NIPS 2002, MIT Press, 2002, pp. 1465–1472.
[16] D.P. Helmbold, D.D.E. Long, T.L. Sconyers, B. Sherrod, Adaptive disk spin-down for mobile computers, Mob. Netw. Appl. (2000) 285–297.
[17] M. Hutter, J. Poland, Adaptive online prediction by following the perturbed leader, J. Mach. Learn. Res. 6 (April 2005) 639–660.
[18] M. Herbster, M.K. Warmuth, Tracking the best expert, Mach. Learn. 32 (1998) 151–178.
[19] D.P. Helmbold, M.K. Warmuth, Learning permutations with exponential weights, J. Mach. Learn. Res. 10 (July 2009) 1705–1736.
[20] P. Kontkanen, P. Myllymäki, A fast normalized maximum likelihood algorithm for multinomial data, in: IJCAI, 2005, pp. 1613–1615.
[21] W.M. Koolen, Combining strategies efficiently: high-quality decisions from conflicting advice, PhD thesis, Institute of Logic, Language and Computation (ILLC), University of Amsterdam, January 2011.
[22] A. Kalai, S. Vempala, Efficient algorithms for online decision problems, J. Comput. Syst. Sci. 71 (3) (2005) 291–307.
[23] W.M. Koolen, M.K. Warmuth, J. Kivinen, Hedging structured concepts, in: Proceedings of the 23rd Annual Conference on Learning Theory, June 2010, pp. 93–105.
[24] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Mach. Learn. 2 (4) (1988) 285–318.
[25] N. Littlestone, M.K. Warmuth, The Weighted Majority algorithm, Inf. Comput. 108 (2) (1994) 212–261.
[26] N. Megiddo, D.S. Modha, One up on LRU, ;login: The Magazine of USENIX and SAGE 28 (4) (August 2003) 6–11.
[27] N. Megiddo, D.S. Modha, Outperforming LRU with an adaptive replacement cache algorithm, IEEE Comput. 37 (4) (April 2004) 58–65.
[28] C.A. Scalisi, An adaptive caching algorithm, Master's Project, University of California Santa Cruz, June 2007.
[29] A. Sen, C. Scalisi, Making online predictions from k-lists, Project report for CMPS 290C, Advanced Topics in Machine Learning, June 2007.
[30] E. Takimoto, M.K. Warmuth, Path kernels and multiplicative updates, J. Mach. Learn. Res. 4 (2003) 773–818.
[31] T. van Erven, S. de Rooij, W.M. Koolen, P. Grünwald, Adaptive hedge, in: Proceedings of the 25th Annual Conference on Neural Information Processing Systems, NIPS 2011, December 2011, pp. 1656–1664.
[32] V. Vovk, A game of prediction with expert advice, J. Comput. Syst. Sci. 56 (2) (1998) 153–173.

[33] M.K. Warmuth, D. Kuzmin, Randomized online pca algorithms with regret bounds that are logarithmic in the dimension, J. Mach. Learn. Res. 9 (October 2008) 2287–2320.
[34] M.K. Warmuth, W.M. Koolen, D.P. Helmbold, Combining initial segments of lists, in: Proceeding of the 22nd International Conference on Algorithmic Learning Theory, ALT 11, Springer-Verlag, October 2011, pp. 219–233.