Electronic Colloquium on Computational Complexity, Report No. 131 (2007)

Boosting and hard-core set constructions: a simplified approach

Satyen Kale
Microsoft Research
One Microsoft Way
Redmond, WA 98052
satyen.kale@microsoft.com

Abstract

We revisit the connection between boosting algorithms and hard-core set constructions discovered by Klivans and Servedio. We present a boosting algorithm with a certain smoothness property that is necessary for hard-core set constructions: the distributions it generates do not put too much weight on any single example. We then use this boosting algorithm to show the existence of hard-core sets matching the best parameters of Klivans and Servedio's construction.

1 Boosting and Hard-Core sets

Hard-core set constructions are a form of hardness amplification of boolean functions. In such constructions, starting with a function that is mildly inapproximable by circuits of a certain size, we obtain a distribution on inputs such that the same function is highly inapproximable by circuits of size closely related to the original, when inputs are drawn from it.

Impagliazzo [3] gave the first hard-core set constructions and used them to give an alternative proof of Yao's XOR lemma. Klivans and Servedio [4] gave improved hard-core set constructions using a connection to the notion of boosting from learning theory. The goal of boosting is to "boost" some small initial advantage over random guessing that a learner can achieve in Valiant's PAC (Probabilistically Approximately Correct) model of learning. Klivans and Servedio describe how boosting algorithms which satisfy certain smoothness properties can be generically used to obtain good hard-core set constructions.

We give a new algorithm for boosting that enjoys the smoothness properties necessary for hard-core set constructions. This algorithm has the additional desirable property that it has the same number of iterations as the `AdaBoost` algorithm, and thus is more efficient Servedio's `SmoothBoost` algorithm [5]. For this reason, the boosting algorithm should be of independent interest, especially for the applications in Servedio's paper [5] on learning in the presence of malicious noise. Our boosting algorithm is inspired by Warmuth and Kuzmin's [7] technique (which, in turn, uses ideas that originated in the work Herbster and Warmuth [2]) of obtaining smooth distributions from any other distribution by projecting it into the set of smooth distributions using the relative entropy as a distance function.

We use this boosting algorithm to construct hard-core sets matching the best parameters of the Klivans-Servedio construction. Although we do not improve over the parameters of Klivans and Servedio's construction, our proof is arguably simpler for three reasons: (a) it obtains the best known parameters for hard-core set constructions directly by applying the boosting algorithm, rather than building it up incrementally by accumulating small hard-core sets, and (b) it obviates

ISSN 1433-8092

the necessity of composing two different boosting algorithms, with separate analyses of the distributions they produce, and, consequently, (c) the final circuit found in our construction is a simple majority over circuits found by the boosting algorithm, rather than a majority of majorities over circuits.

Finally, the improved efficiency of our boosting algorithm has ramifications for for an application in Klivans and Servedio's paper: it enables us to shave off a factor of $O(\log(1/\varepsilon))$ from the running time of the fastest known algorithm to learn DNF formulae with membership queries under the uniform distribution, while retaining the final hypothesis as a simple majority-of-parity circuit, instead of a majority-of-majority-of-parity circuit.

Our boosting algorithm is based on the same technology as Freund and Schapire's `AdaBoost` algorithm [1], viz. the Multiplicative Weights Update method. We augment this technique with the technique of Bregman projections (from [2, 7]) into the set of all smooth distributions as mentioned previously.

We now introduce this technique in the context of a repeated online decision making problem. In each round, we have to make a decision, and then we suffer an associated cost. To guide us in this task, we have n experts. In each round, every expert recommends a course of action, and our task is to pick an expert and use his advice. At this point the costs of all actions recommended by the experts are revealed, and we suffer the cost of the action recommended by the expert we chose.

We now set up some notation. Let t denote the current round, and let i be a generic expert. In each round t, we select a distribution $\mathbf{p}^{(t)}$ over the set of experts, and select an expert i randomly from it (and use his advised course of action). We impose the (crucial) restriction that the distribution $\mathbf{p}^{(t)}$ comes from some specified convex set $\mathcal{P}$ of distributions.

At this point, the costs of all the actions recommended by the experts are revealed in the form of the vector $\mathbf{m}^{(t)}$ such that expert i incurs cost $m_i{}^{(t)}$. We assume that the costs lie in the range $[0, 1]$. The expected cost to the algorithm for choosing the distribution $\mathbf{p}^{(t)}$ is $\mathbb{E}_{i \in \mathbf{p}^{(t)}}[m_i{}^{(t)}] = \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$.

The total expected cost over all rounds is therefore $\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Our goal now is to design an algorithm which achieves a total expected cost not too much more than the cost of the best fixed distribution in $\mathcal{P}$ in hindsight, viz. $\min_{\mathbf{p} \in \mathcal{P}} \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}$. The following theorem will be proved in Section A. This theorem (and its analysis) appears in a slightly different form in [2, 7], and for completeness we include the proof of the theorem in Appendix A.

Theorem 1. *Let $0 < \delta \leq 1/2$ be an error parameter, and let T be any positive integer. There is an algorithm, which generates distributions $\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(T)} \in \mathcal{P}$ where each $\mathbf{p}^{(t)}$ is computed only based on $\mathbf{m}^{(1)}, \ldots, \mathbf{m}^{(t-1)}$, such that for any $\mathbf{p} \in \mathcal{P}$,*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; (1+\delta) \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\delta}.$$

Here, for distributions $\mathbf{p}$ and $\mathbf{q}$, the relative entropy between them is $\mathbf{RE}(\mathbf{p} \parallel \mathbf{q}) = \sum_i p_i \ln(p_i/q_i)$.

2 A Smooth Boosting algorithm

We work in the setting of Valiant's Probably Approximately Correct (PAC) learning framework [6]. In this framework, we are given a domain X of learning examples, and a concept class $\mathcal{C}$ of certain boolean functions over the domain X. We assume there is an unknown function $f \in \mathcal{C}$ which labels all points in X with boolean values. Also, we assume that there is an unknown distribution $\mathcal{D}$ over X such that we can efficiently sample points from X under $\mathcal{D}$ and obtain their corresponding labels.

We assume that we have access to a sample set S of m labeled examples $\langle x_j, y_j \rangle$ for $j = 1, 2, \ldots, m$, where $x_j \in X$, the domain of our samples, and $y_j \in \{0, 1\}$ is the label of x_j. Given a function $f : X \to [0, 1]$, and an example $\langle x_j, y_j \rangle$, the margin of f on x_j is defined to be $1/2 - |f(x_j) - y_j|$. Intuitively, if we use the function value on an example as a probability to predict the label of the example, then the margin is the advantage over random guessing. Given a distribution $\mathbf{p}$ over S, the function f is said to have advantage γ under $\mathbf{p}$ if the expected margin of f when examples are drawn from $\mathbf{p}$ is at least γ, i.e. $\sum_{j=1}^{m} p_j[1/2 - |f(x_j) - y_j|] \geq \gamma$, or equivalently, $\sum_{j=1}^{m} p_j |f(x_j) - y_j| \leq 1/2 - \gamma$. A γ-weak learning algorithm for S is one which, given any distribution $\mathbf{p}$ on S and a parameter $\gamma > 0$, returns a function $h : X \to [0, 1]$ with advantage γ under $\mathbf{p}$. Given an error rate ε, a boosting algorithm with margin θ is one which, given access to a γ-weak learning algorithm for S, runs it on a sequence of carefully constructed distributions on S, and combines the hypotheses it returns in some manner to obtain a function $f : X \to [0, 1]$ that has margin at least θ on at least $1 - \varepsilon$ fraction of examples in S.

For the application to hard-core set constructions, we need the boosting algorithm to satisfy a certain smoothness property on the distributions it generates: it needs to ensure that these distribution do not put too much mass on any single example. For a distribution $\mathbf{p}$ on S, define $\mu(\mathbf{p}) := \frac{1}{|S| \mathcal{L}_\infty(\mathbf{p})}$ (here, $\mathcal{L}_\infty(\mathbf{p}) = \max_i p_i$). A distribution $\mathbf{p}$ on S is called ε-smooth if $\mu(\mathbf{p}) \geq \varepsilon$.

With this setup, we can now describe the properties of our boosting algorithm:

Theorem 2. *Given an error rate ε and access to a γ-weak learning algorithm, there is an efficient boosting algorithm with margin $\theta = \gamma/4$ which makes $T = O(\frac{\log(1/\varepsilon)}{\gamma^2})$ calls to the weak learning algorithm with ε-smooth distributions. The final output hypothesis is $f = \frac{1}{T} \sum_{t=1}^{T} h^{(t)}$, where $h^{(t)}$ is the hypothesis output by the weak learning algorithm in round t.*

PROOF: Let $\mathcal{P}$ be the set of ε-smooth distributions on the S. Note that this is a convex set. We now run the Multiplicative Weights algorithm of Theorem 1 with the experts corresponding to the m examples in S and with the parameter $\delta = \gamma/4$. The distributions on the experts are restricted to be in $\mathcal{P}$ (and are thus always ε-smooth).

We start with $\mathbf{p}^{(1)} = \mathbf{u}$, the uniform distribution on S. In every round t, the algorithm runs the γ-weak learning algorithm with the distribution $\mathbf{p}^{(t)}$ generated by the Multiplicative Weights algorithm to obtain a hypothesis $h^{(t)}$ that has advantage γ under $\mathbf{p}^{(t)}$. Then for $j = 1, \ldots, m$, we set ${m_j}^{(t)} = 1 - |h^{(t)}(x_j) - y_j|$ (thus, ${m_j}^{(t)}$ is just the margin of $h^{(t)}$ on x_j, plus 1/2). Note that ${m_j}^{(t)} \in [0, 1]$, and the expected loss in the round t is

$$\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;=\; \sum_j {p_j}^{(t)} [1 - |h^{(t)}(x_j) - y_j|] \;\geq\; 1/2 + \gamma,$$

since $h^{(t)}$ has advantage γ under $\mathbf{p}^{(t)}$.

After $T = \lceil \frac{8 \ln(1/\varepsilon)}{\gamma^2} \rceil + 1$ rounds, we set our final hypothesis $f = \frac{1}{T} \sum_{t=1}^{T} h^{(t)}$. Let $E \subseteq S$ be the subset of examples where f has margin less than $\theta = \gamma/4$. Suppose $|S| \geq \varepsilon m$. Let $\mathbf{p} = \mathbf{u}_E$, the uniform distribution on E. Note that $\mathbf{u}_E \in \mathcal{P}$, since $|S| \geq \varepsilon m$. Now, for any example $\langle x_j, y_j \rangle \in S$, we have

$$\sum_{t=1}^{T} {m_j}^{(t)} \;=\; \sum_{t=1}^{T} 1 - |h^{(t)}(x_j) - y_j| \;\leq\; T - \left| \sum_{t=1}^{T} h^{(t)} - T y_j \right| \;=\; T(1 - |f(x_j) - y_j|) \;\leq\; (1/2 + \theta) T,$$

since f has margin less than θ on $\langle x_j, y_j \rangle$. Thus,

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{u}_E \;=\; \frac{1}{|E|} \sum_{\langle x_j, y_j \rangle \in S} \sum_{t=1}^{T} {m_j}^{(t)} \;\leq\; (1/2 + \theta) T.$$

Also, since $|E| \geq \varepsilon m$, it is easy to check that

$$\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)}) \;=\; \mathbf{RE}(\mathbf{u}_E \parallel \mathbf{u}) \;\leq\; \ln(1/\varepsilon).$$

Plugging these observations into the inequality of Theorem 1, since $\delta = \gamma/4$, we get

$$(1/2+\gamma)T \;\leq\; (1+\gamma/4)(1/2+\theta)T + \frac{4\ln(1/\varepsilon)}{\gamma}.$$

But since $\theta = \gamma/4$, we get that $(1+\gamma/4)(1/2+\theta) \leq 1/2+\gamma/2$. Now since $T = \lceil \frac{8\ln(1/\varepsilon)}{\gamma^2} \rceil + 1$, we get a contradiction to this inequality. □

A note of comparison: our boosting algorithm improves over Servedio's `SmoothBoost` algorithm [5] in running time, reducing the polynomial dependence on $1/\varepsilon$ to logarithmic dependence. `SmoothBoost` needs $O(\frac{1}{\varepsilon\gamma^2})$ calls to the weak learning algorithm, whereas ours needs $O(\frac{\log(1/\varepsilon)}{\gamma^2})$.

3 Hard-core Set Construction

Let f be a function from $\{0,1\}^n \to \{0,1\}$. Let $\mathbf{u}$ be the uniform distribution on $\{0,1\}^n$. f is said to be *ε-hard* for circuits of size g if no such circuit can match f on more than $1-\varepsilon$ fraction of inputs. For a distribution $\mathbf{p}$ on $\{0,1\}^n$, f is said to be *γ-hard-core* under $\mathbf{p}$ for circuits of size g if no such circuit can match on f on a subset of inputs of measure at least $1/2+\gamma$ when inputs are sampled from $\mathbf{p}$. A *γ-hard-core set* for f is a set of inputs S such that f is γ-hard-core under the uniform distribution $\mathbf{u}_S$ on S for circuits of size g.

Given a function f that is ε-hard for circuits of size g, our goal is to find hard-core sets S for f that are as large as possible, for circuits of size g' which is as close to g as possible. The size of S, as a fraction of the 2^n inputs, is exactly $1/\mathcal{L}_\infty(2^n\mathbf{u}_S)$. We relax our objective a bit, and aim to find a distribution $\mathbf{p}$ on $\{0,1\}^n$ such that f is γ-hard-core on $\mathbf{p}$, so that $\mu(\mathbf{p}) := 1/\mathcal{L}_\infty(2^n\mathbf{p})$ is as large as possible. It can be shown that by randomly sampling inputs using $\mathbf{p}$ sufficiently many times, we can construct an $O(\gamma)$-hard-core set of size $\Omega(\mu(\mathbf{p})\cdot 2^n)$.

We now describe our hard-core set constructions. Note that the following theorem can also be proved directly using the Theorem 1, but we prefer to prove it using the boosting algorithm of the previous section to make the connection to boosting explicit.

Theorem 3. *Let f be a function from $\{0,1\}^n \to \{0,1\}$ that is ε hard for circuits of size g. Then there is a distribution $\mathbf{p}$ with $\mu(\mathbf{p}) \geq \varepsilon$ such that f is γ-hard-core for $\mathbf{p}$ for circuits of size $O(\frac{\gamma^2}{\log(1/\varepsilon)}g)$.*

Proof: Assume by way of contradiction that for any distribution $\mathbf{p}$ with $\mu(\mathbf{p}) \geq \varepsilon$, there is a circuit of size $g' = O(\frac{\gamma^2}{\log(1/\varepsilon)}g)$ that computes f on a subset of inputs of measure at least $1/2+\gamma$ under $\mathbf{p}$.

We now run the smooth boosting algorithm of Theorem 2 with the set of examples $S = \{\langle x, f(x)\rangle : \; x \in \{0,1\}^n\}$, and with the error parameter ε. The weak learning algorithm, given a distribution $\mathbf{p}$ with $\mu(\mathbf{p}) \geq \varepsilon$, outputs a circuit (which exists, by our assumption above) of size g' which computes f on a subset of inputs of measure at least $1/2+\gamma$ under $\mathbf{p}$. It can be checked easily that the function computed by this circuit has advantage γ under $\mathbf{p}$. Let $C^{(t)}$ be the circuit found in round t for distribution $\mathbf{p}^{(t)}$, and let $h^{(t)}$ be the function computed by it. Theorem 2 implies that the function $f = \frac{1}{T}\sum_{t=1}^{T} h^{(t)}$ has margin at least $\theta = \gamma/4$ on at least $1-\varepsilon$ fraction of inputs. This implies that the circuit $\texttt{MAJORITY}(C^{(1)}, \ldots, C^{(T)})$ matches f on at least $1-\varepsilon$ fraction of inputs. The size of this majority of circuits is $\Theta(Tg') < g$, for a suitable choice of g', which contradicts the assumption that f is ε hard for circuits of size g. □

Acknowledgments

I thank Manfred Warmuth for showing me the Bregman projection trick, and for interesting discussions of the uses of this idea in other applications.

References

[1] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[2] Mark Herbster and Manfred K. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.

[3] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 October 1995. IEEE.

[4] Adam R. Klivans and Rocco A. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003.

[5] Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

[6] Leslie G. Valiant. A theory of the learnable. In *STOC*, pages 436–445, 1984.

[7] Manfred K. Warmuth and Dima Kuzmin. Randomized pca algorithms with regret bounds that are logarithmic in the dimension. In *NIPS*, 2006.

A The Multiplicative Weights algorithm with Restricted Distributions

In this section we prove Theorem 1. We restate it here for convenience:

Theorem 1. *Let $0 < \delta \leq 1/2$ be an error parameter, and let T be any positive integer. There is an algorithm, which generates distributions $\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(T)} \in \mathcal{P}$ where each $\mathbf{p}^{(t)}$ is computed only based on $\mathbf{m}^{(1)}, \ldots, \mathbf{m}^{(t-1)}$, such that for any $\mathbf{p} \in \mathcal{P}$,*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; (1+\delta) \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\delta}.$$

Proof: The algorithm is shown in Figure 1.

Multiplicative Weights Update algorithm with Restricted Distributions

Initialization: Fix a $\delta \leq \frac{1}{2}$. Start out with an arbitrary probability distribution $\mathbf{p}^{(t)}$ on the experts. initialized to 1.
For $t = 1, 2, \ldots, T$:

1. Choose expert i using $\mathbf{p}^{(t)}$.
2. Observe the costs of the experts $\mathbf{m}^{(t)}$.
3. Compute the probability vector $\hat{\mathbf{p}}^{(t+1)}$ using the following multiplicative update rule: for every expert i,
$$p_i{}^{(t+1)} \;=\; p_i{}^{(t)}(1-\delta)^{m_i{}^{(t)}}/Z^{(t)} \tag{1}$$
where $Z^{(t)} = \sum_i p_i{}^{(t)}(1-\delta)^{m_i{}^{(t)}}$ is the normalization factor.
4. Set $\mathbf{p}^{(t+1)}$ to be the projection of $\hat{\mathbf{p}}^{(t)}$ on the set $\mathcal{P}$ using the **RE** as a distance function, i.e.
$$\mathbf{p}^{(t+1)} \;=\; \arg\min_{\mathbf{p}\in\mathcal{P}} \mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{(t)}).$$

Figure 1: The Multiplicative Weights algorithm with Restricted Distributions

We use the relative entropy between $\mathbf{p}$ and $\mathbf{p}^{(t)}$, $\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) := \sum_i p_i \ln(p_i/p_i{}^{(t)})$ as a "potential" function. We have

$$\begin{aligned}
\mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{t+1}) - \mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) \;&=\; \sum_i p_i \ln \frac{p_i{}^{(t)}}{\hat{p}_i{}^{(t+1)}} \\
&=\; \sum_i p_i \ln \frac{Z^{(t)}}{(1-\delta)^{m_i{}^{(t)}}} \\
&=\; \ln \frac{1}{1-\delta} \sum_i p_i m_i{}^{(t)} + \ln Z^{(t)} \\
&\leq\; \delta(1+\delta)(\mathbf{m}^{(t)} \cdot \mathbf{p}) + \ln Z^{(t)}.
\end{aligned}$$

The last inequality follows because $-\ln(1-\delta) \leq \delta(1+\delta)$ for $\delta \leq \frac{1}{2}$.

$$\begin{aligned}
\ln Z^{(t)} \;&=\; \ln[\sum_i p_i{}^{(t)}(1-\delta)^{m_i{}^{(t)}}] \\
&\leq\; \ln[\sum_i p_i{}^{(t)}(1-\delta m_i{}^{(t)})] && \because (1-\delta)^x \leq (1-\delta x) \text{ for } x \in [0,1] \\
&=\; \ln[1-\delta \sum_i p_i{}^{(t)} m_i{}^{(t)}] \\
&\leq\; -\delta \sum_i p_i{}^{(t)} m_i{}^{(t)}. && \because \ln(1-x) \leq -x \text{ for } x < 1
\end{aligned}$$

Thus, we get

$$\mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{t+1}) - \mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) \;\leq\; \delta(1+\delta)(\mathbf{m}^{(t)} \cdot \mathbf{p}) - \delta(\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).$$

Now, projection on the set $\mathcal{P}$ using the relative entropy as a distance function is a Bregman projection, and thus it satisfies the following Generalized Pythagorean inequality (see, e.g. [2]), for any $\mathbf{p} \in \mathcal{P}$:

$$\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t+1)}) + \mathbf{RE}(\mathbf{p}^{(t+1)} \parallel \hat{\mathbf{p}}^{(t+1)}) \ \leq \ \mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{(t+1)}).$$

Since $\mathbf{RE}(\mathbf{p}^{(t+1)} \parallel \hat{\mathbf{p}}^{(t+1)}) \geq 0$, we get the following bound:

$$\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t+1)}) - \mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) \ \leq \ \delta(1+\delta)(\mathbf{m}^{(t)} \cdot \mathbf{p}) - \delta(\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).$$

Summing up from $t = 1$ to T, and using the fact that $\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(T+1)}) \geq 0$, and simplifying, we get

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \ \leq \ (1+\delta) \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\delta}.$$

□

A.1 Computing the Projection

In general, the projection of a probability distribution $\mathbf{p}$ on the experts is a convex program, since the $\mathbf{RE}(\mathbf{q} \parallel \mathbf{p})$ is a convex function of $\mathbf{q}$, and so it can be computed in polynomial time using standard convex programming techniques.

In one special case, however, we can compute the projection far more efficiently. Let $\varepsilon > 0$ be a given parameter. For a distribution $\mathbf{p}$ on the experts, define $\mu(\mathbf{p}) := 1/(n\mathcal{L}_\infty(\mathbf{p}))$, and let $\mathcal{P}$ be the (convex) set of distributions $\mathbf{p}$ on the experts such that $\mu(\mathbf{p}) \geq \varepsilon$.

In [2], [7], the following theorem is proved, by recursively using the median:

Theorem 4. *There is a $O(n)$ time algorithm that, given a probability distribution $\mathbf{p}$ over S, computes the projection of $\mathbf{p}$ on $\mathcal{P}$ with the $\mathbf{RE}$ as a distance function.*

A very simple algorithm can be obtained as follows. Find the least index k such that clipping the largest k coordinates of $\mathbf{p}$ to $\frac{1}{\varepsilon n}$ and rescaling the rest of the coordinates to sum up to $1 - \frac{k}{\varepsilon n}$ ensures that all coordinates are at most $\frac{1}{\varepsilon n}$, and output the probability vector thus obtained. This is the required projection. This can be implemented by sorting the coordinates, and so it takes $O(n \log n)$ time.

ECCC ISSN 1433-8092
http://eccc.hpi-web.de/