Online Learning of k-CNF Boolean Functions

Joel Veness[1], **Marcus Hutter**[2], **Laurent Orseau**[1], **Marc Bellemare**[1]
[1]Google DeepMind, [2]Australian National University
{veness,lorseau,bellemare}@google.com, marcus.hutter@anu.edu.au

Abstract

This paper revisits the problem of learning a k-CNF Boolean function from examples, for fixed k, in the context of online learning under the logarithmic loss. We give a Bayesian interpretation to one of Valiant's classic PAC learning algorithms, which we then build upon to derive three efficient, online, probabilistic, supervised learning algorithms for predicting the output of an unknown k-CNF Boolean function. We analyze the loss of our methods, and show that the cumulative log-loss can be upper bounded by a polynomial function of the size of each example.

1 Introduction

In 1984, Leslie Valiant introduced the notion of Probably Approximately Correct (PAC) learnability, and gave three important examples of some non-trivial concept classes that could be PAC learnt given nothing more than a sequence of positive examples drawn from an arbitrary IID distribution [Valiant, 1984]. One of these examples was the class of k-CNF Boolean functions, for fixed k. Valiant's approach relied on a polynomial time reduction of this problem to that of PAC learning the class of monotone conjunctions. In this paper, we revisit the problem of learning monotone conjunctions from the perspective of universal source coding, or equivalently, online learning under the logarithmic loss. In particular we derive three new online, probabilistic prediction algorithms that: (i) learn from both positive and negative examples; (ii) avoid making IID assumptions; (iii) suffer low logarithmic loss for arbitrary sequences of examples; (iv) run in polynomial time and space. This work is intended to complement previous work on concept identification [Valiant, 1984] and online learning under the 0/1 loss [Littlestone, 1988; Littlestone and Warmuth, 1994].

The main motivation for investigating online learning under the logarithmic loss is the fundamental role it plays within information theoretic applications. In particular, we are interested in prediction methods that satisfy the following *power* desiderata, i.e. methods which: (p) make probabilistic predictions; (o) are strongly online; (w) work well in practice; (e) are efficient; and (r) have well understood regret/loss properties. Methods satisfying these properties can be used in a number of principled and interesting ways: for example, data compression via arithmetic encoding [Witten *et al.*, 1987], compression-based clustering [Cilibrasi and Vitányi, 2005] or classification [Frank *et al.*, 2000; Bratko *et al.*, 2006], and information theoretic reinforcement learning [Veness *et al.*, 2011; 2015]. Furthermore, it is possible to combine such online, log-loss predictors using various ensemble methods [Veness *et al.*, 2012b; Mattern, 2013]. The ability to rapidly exploit deterministic underlying structure such as k-CNF relations has the potential to improve all the aforementioned application areas, and brings the universal source coding literature in line with developments originating from the machine learning community.

Our contribution in this paper stems from noticing that Valiant's method can be interpreted as a kind of MAP model selection procedure with respect to a particular family of priors. In particular, we show that given n positive examples and their associated d-dimensional binary input vectors, it is possible to perform exact Bayesian inference over the 2^d possible monotone conjunction hypotheses in time $O(nd)$ and space $O(d)$ without making IID assumptions. Unfortunately, these desirable computational properties do not extend to the case where both positive and negative examples are presented; we show that in this case exact inference is #P-complete. This result motivated us to develop a hybrid algorithm, which uses a combination of Bayesian inference and memorization to construct a polynomial time algorithm whose loss is bounded by $O(d^2)$ for the class of monotone conjunctions. Furthermore, we show how to trade constant loss for logarithmic cumulative loss to get a more practical algorithm, whose loss is bounded by $O(d \log n)$. We also give an alternative method, based on the WINNOW algorithm [Littlestone, 1988] for 0/1 loss, which has better theoretical properties in cases where many of the d Boolean inputs are irrelevant. Finally, similarly to Valiant, we describe how to combine our algorithms with a reduction that (for fixed k) enables the efficient learning of k-CNF Boolean functions from examples.

2 Preliminaries

Notation. A Boolean variable x is an element of $\mathcal{B} := \{\bot, \top\} = \{0, 1\}$. We identify false $\bot$ with 0 and true $\top$ with 1, since it allows us to use Boolean functions as likelihood functions for deterministically generated data. We keep the boolean operator notation whenever more sugges-

tive. The unary not operator is denoted by $\neg$, and is defined as $\neg : 0 \mapsto 1; 1 \mapsto 0$ ($\neg x = 1 - x$). The binary conjunction and disjunction operators are denoted by $\wedge$ and $\vee$ respectively, and are given by the maps $\wedge : (1,1) \mapsto 1$; or 0 otherwise ($x \wedge y = x \cdot y$). and $\vee : (0,0) \mapsto 0$; or 1 otherwise ($x \vee y = \max\{x, y\}$). A literal is a Boolean variable x or its negation $\neg x$; a positive literal is a non-negated Boolean variable. A clause is a finite disjunction of literals. A monotone conjunction is a conjunction of zero or more positive literals. For example, $x_1 \wedge x_3 \wedge x_6$ is a monotone conjunction, while $\neg x_1 \wedge x_3$ is not. We adopt the usual convention with conjunctions of defining the zero literal case to be vacuously true. The power set of a set $\mathcal{S}$ is the set of all subsets of $\mathcal{S}$, and will be denoted by $\mathcal{P}(\mathcal{S})$. For convenience, we further define $\mathcal{P}_d := \mathcal{P}(\{1, 2, \ldots, d\})$. We also use the Iverson bracket notation $[\![P]\!]$, which given a predicate P, evaluates to 1 if P is true and 0 otherwise. We also use the notation $x_{1:n}$ and $x_{<n}$ to represent the sequences of symbols $x_1 x_2 \ldots x_n$ and $x_1 x_2 \ldots x_{n-1}$ respectively. Furthermore, base two is assumed for all logarithms in this paper. Finally, we use the notation a^i to index the ith component of a Boolean vector $a \in \mathcal{B}^d$.

Problem Setup. We consider an online, sequential, binary, probabilistic prediction task with side information. At each time step $t \in \mathbb{N}$, a d-dimensional Boolean vector of side information $a_t \equiv (a_t^1, ..., a_t^d) \in \mathcal{B}^d$ is presented to a probabilistic predictor $\rho_t : \mathcal{B}^d \to (\mathcal{B} \to [0,1])$, which outputs a probability distribution over $\mathcal{B}$. A label $x_t \in \mathcal{B}$ is then revealed, with the predictor suffering an instantaneous loss of $\ell_t := -\log \rho_t(x_t; a_t)$, with the cycle continuing ad infinitum. It will also prove convenient to introduce the joint distribution $\rho(x_{1:n}; a_{1:n}) := \prod_{t=1}^n \rho_t(x_t; a_t)$, which lets us express the cumulative loss $\mathcal{L}_n(\rho)$ in the form

$$\mathcal{L}_n(\rho) \;:=\; \sum_{i=1}^{n} \ell_t \;=\; -\log \rho(x_{1:n}; a_{1:n}).$$

We later use the above quantity to analyze the theoretical properties of our technique. As is usual with loss or regret based approaches, our goal will be to construct a predictor ρ such that $\mathcal{L}_n(\rho)/n \to 0$ as $n \to \infty$ for an interesting class of probabilistic predictors $\mathcal{M}$. The focus of our attention for the remainder of this paper will be on the class of monotone conjunctions.

Brute force Bayesian learning. Consider the monotone conjunction $h_{\mathcal{S}}(a_t) := \bigwedge_{i \in \mathcal{S}} a_t^i$ for some $\mathcal{S} \in \mathcal{P}_d$, classifying $a_t \in \mathcal{B}^d$ as $h_{\mathcal{S}}(a_t) \in \mathcal{B}$. This can be extended to the function $h_{\mathcal{S}} : \mathcal{B}^{n \times d} \to \mathcal{B}^n$ that returns the vector $h_{\mathcal{S}}(a_{1:n}) := (h_{\mathcal{S}}(a_1), ..., h_{\mathcal{S}}(a_n))$. One natural Bayesian approach to learning monotone conjunctions would be to place a uniform prior over the set of 2^d possible deterministic predictors that are monotone conjunctions of the d Boolean input variables. This gives the Bayesian mixture model

$$\xi_d(x_{1:n}; a_{1:n}) \;:=\; \sum_{\mathcal{S} \in \mathcal{P}_d} \frac{1}{2^d} \nu_{\mathcal{S}}(x_{1:n}; a_{1:n}), \qquad (1)$$

where $\nu_{\mathcal{S}}(x_{1:n}; a_{1:n}) \;:=\; [\![h_{\mathcal{S}}(a_{1:n}) = x_{1:n}]\!]$ is the deterministic distribution corresponding to $h_{\mathcal{S}}$. Note that when $\mathcal{S} = \{\}$, the conjunction $\bigwedge_{i \in \mathcal{S}} a_t^i$ is vacuously true. From here onwards, we will say hypothesis $h_{\mathcal{S}}$ generates $x_{1:n}$ if $h_{\mathcal{S}}(a_{1:n}) = x_{1:n}$. For sequential prediction, the predictive probability $\xi_d(x_t | x_{<t}; a_{1:t})$ can be obtained by computing the ratio of the marginals, that is $\xi_d(x_t | x_{<t}; a_{1:t}) = \xi_d(x_{1:t}; a_{1:t}) \;/\; \xi_d(x_{<t}; a_{<t})$. Note that this form of the predictive distribution is equivalent to using Bayes rule to explicitly compute the posterior weight for each $\mathcal{S}$, and then taking a convex combination of the instantaneous predictions made by each hypothesis. The loss of this approach for an arbitrary sequence of data generated by some $h_{\mathcal{S}^*}$ for $\mathcal{S}^* \in \mathcal{P}_d$, can be upper bounded by

$$\begin{aligned} \mathcal{L}_n(\xi_d) \;&:=\; -\log \xi_d(x_{1:n}; a_{1:n}) \\ &=\; -\log \sum_{\mathcal{S} \in \mathcal{P}_d} \tfrac{1}{2^d} [\![h_{\mathcal{S}}(a_{1:n}) = x_{1:n}]\!] \\ &\leq\; -\log \tfrac{1}{2^d} [\![h_{\mathcal{S}^*}(a_{1:n}) = x_{1:n}]\!] \;=\; d. \end{aligned}$$

Of course the downside with this approach is that a naive computation of Equation 1 takes time $O(n\, 2^d)$. Indeed one can show that no polynomial-time algorithm in d for ξ_d exists (assuming P$\neq$NP).

Theorem 1 (ξ_d is #P-complete)**.** *Computing the function $f : \{0,1\}^{n \times d} \to \{0, \ldots, 2^d\}$ defined as $f(a_{1:n}) := 2^d \xi_d(0_{1:n}; a_{1:n})$ is #P-complete.*

We prove hardness by a two-step reduction: counting independent sets, known to be #P-hard, to computing the cardinality of a union of power sets to computing ξ_d.

Definition 2 (UPOW)**.** *Given a list of n subsets $\mathcal{S}_1, \ldots, \mathcal{S}_n$ of $\{1, \ldots, d\}$, compute $A := |\mathcal{P}(\mathcal{S}_1) \cup \cdots \cup \mathcal{P}(\mathcal{S}_n)|$, i.e. the size of the union of the power sets of $\mathcal{S}_1, \ldots, \mathcal{S}_n$.*

Lemma 3 (UPOW$\to$ ξ_d)**.** *If a_t is defined as the d-dimensional characteristic bit vector describing the elements in $\mathcal{S}_t$, i.e. $a_t^i := [\![i \in \mathcal{S}_t]\!]$, then $A = 2^d[1 - \xi_d(0_{1:n} | a_{1:n})]$.*

Proof. Since $h_{\mathcal{S}}(a_t) = 1$ iff $\mathcal{S} \subseteq \mathcal{S}_t$ iff $\mathcal{S} \in \mathcal{P}(\mathcal{S}_t)$ we have

$$\begin{aligned} &[\![h_{\mathcal{S}}(a_{1:n}) = 0_{1:n}]\!] \iff \bigwedge_{t=1}^{n} [h_{\mathcal{S}}(a_t) = 0] \\ &\iff \neg \exists t : \mathcal{S} \in \mathcal{P}(\mathcal{S}_t) \iff \mathcal{S} \notin \mathcal{P}(\mathcal{S}_1) \cup ... \cup \mathcal{P}(\mathcal{S}_n) \end{aligned}$$

which implies $\sum_{\mathcal{S} \in \mathcal{P}_d} \nu_{\mathcal{S}}(0_{1:n} | a_{1:n}) = 2^d - A$. □

The intuition behind Lemma 3 is that since ξ_d uses a uniform prior over $\mathcal{P}_d$, the number of hypotheses consistent with the data is equal to $2^d \xi_d(0_{1:n} | a_{1:n})$, and therefore the number of hypotheses inconsistent with the data is equal to $2^d[1 - \xi_d(0_{1:n} | a_{1:n})]$. One can easily verify that the set of hypotheses inconsistent with a single negative example is $I_t := \mathcal{P}\left(\left\{i \in \{1, \ldots, d\} \;:\; [\![a_t^i = 1]\!]\right\}\right)$, hence the set of hypotheses inconsistent with the data is equal to $|\cup_{t=1}^n I_t|$.

Theorem 4 (IS$\to$UPOW, Brendan McKay, private communication)**.** *UPOW is #P-hard.*

Proof. Let $G = (V, E)$ be an undirected graph with vertices $V = \{1, ..., d\}$ and edges $E = \{e_1, ..., e_n\}$, where edges are $e = \{v, w\}$ with $v, w \in V$ and $v \neq w$. An independent set I is a set of vertices no two of which are connected by an edge. The set of independent sets is IS $:= \{I \subseteq V : \forall e \in E : e \not\subseteq I\}$. It is known that counting independent sets, i.e. computing |IS| is #P-hard [Vadhan, 2001].

We now reduce IS to UPOW: Define $\mathcal{S}_t := V \setminus e_t$ for $t \in \{1, ..., n\}$ and consider any $W \subseteq V$ and its complement $\overline{W} = V \setminus W$. Then

$$\begin{aligned} W \notin \text{IS} &\iff \exists e \in E : e \subseteq W \iff \exists t : e_t \subseteq W \\ &\iff \exists t : \overline{W} \subseteq \mathcal{S}_t \iff \exists t : \overline{W} \in \mathcal{P}(\mathcal{S}_t) \\ &\iff \overline{W} \in \mathcal{P}(\mathcal{S}_1) \cup ... \cup \mathcal{P}(\mathcal{S}_n). \end{aligned}$$

Since set-complement is a bijection and there are $2^{|V|}$ possible W, this implies $|\text{IS}| + |\mathcal{P}(\mathcal{S}_1) \cup ... \cup \mathcal{P}(\mathcal{S}_n)| = 2^{|V|}$. Hence an efficient algorithm for computing $|\mathcal{P}(\mathcal{S}_1) \cup ... \cup \mathcal{P}(\mathcal{S}_n)|$ would imply the existence of an efficient algorithm for computing |IS|. □

Proof. **of Theorem 1.** Lemma 3 and Theorem 4 show that f is #P-hard. What remains to be shown is that f is in $\#P$. First consider UPOW function $u : \mathcal{P}_d^n \to \{0, ..., 2^d\}$ defined as $u(\mathcal{S}_1, ..., \mathcal{S}_d) := A$. With identification $\{0, 1\}^d \cong \mathcal{P}_d$ via $a_t^i = [\![i \in \mathcal{S}_t]\!]$ and $\mathcal{S}_t = \{i : a_t^i = 1\}$, Lemma 3 shows that $f(a_{1:n}) + u(\mathcal{S}_1, ..., \mathcal{S}_n) = 2^d$. Since $\mathcal{S} \in \mathcal{P}(\mathcal{S}_1) \cup ... \cup \mathcal{P}(\mathcal{S}_n)$ iff $\exists t : \mathcal{S} \in \mathcal{P}(\mathcal{S}_t)$ iff $\exists t : \mathcal{S} \subseteq \mathcal{S}_t$, the non-deterministic polynomial time algorithm "Guess $\mathcal{S} \in \mathcal{P}_d$ and accept iff $\exists t : \mathcal{S} \subseteq \mathcal{S}_t$" has exactly A accepting paths, hence u is in $\#P$. Since this algorithm has 2^d paths in total, swapping accepting and non-accepting paths shows that also f is in $\#P$. □

One interesting feature of our reduction was that we only required a sequence of negative examples. As we shall see in Section 3, exact Bayesian inference is tractable if only positive examples are provided. Finally, one can also show that the Bayesian predictor ξ_d obtains the optimal loss.

Proposition 5. *There exists a sequence of side information $a_{1:2^d} \in \mathcal{B}^{2^d \times d}$ such that for any probabilistic predictor $\rho_t : \mathcal{B}^d \to (\mathcal{B} \to [0, 1])$, there exists an $\mathcal{S} \in \mathcal{P}_d$ such that $h_{\mathcal{S}}$ would generate a sequence of targets that would give $\mathcal{L}_{2^d}(\rho) \geq d$.*

Proof. Consider the sequence of side information $a_{1:2^d} \in \mathcal{B}^{2^d \times d}$, where a_t^i is defined to be the ith digit of the binary representation of t, for all $1 \leq t \leq 2^d$. As

$$|\{x_{1:2^d} : x_{1:2^d} \text{ is generated by an } \mathcal{S} \in \mathcal{P}_d\}| = 2^d, \quad (2)$$

to have $\mathcal{L}_{2^d}(\rho) < \infty$ for all $x_{1:2^d}$, we need $\rho(x_{1:2^d}) > 0$ for each of the 2^d possible target strings, which implies that $\mathcal{L}_{2^d}(\rho) \geq d$. □

Memorization. As a further motivating example, it is instructive to compare the exact Bayesian predictor to that of a naive method for learning monotone conjunctions that simply memorizes the training instances, without exploiting the logical structure within the class. To this end, consider the sequential predictor that assigns a probability of

$$m_d(x_n | x_{<n}; a_{1:n}) = \begin{cases} [\![x_n = l(a_{1:n}, x_{<n})]\!] & \text{if } a_n \in \{a_t\}_{t=1}^{n-1} \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

to each target, where $l(a_{1:n}, x_{<n})$ returns the value of x_t for some $1 \leq t \leq n-1$ such that $a_n = a_t$. Provided the data is generated by some $h_{\mathcal{S}}$ with $\mathcal{S} \in \mathcal{P}_d$, the loss of the above memorization technique is easily seen to be at most 2^d. This follows since an excess loss of 1 bit is suffered whenever a new a_k is seen, and there are at most 2^d distinct inputs (of course no loss is suffered whenever a previously seen a_k is repeated). While both memorization and the Bayes predictor suffer a constant loss that is independent of the number of training instances, the loss of the memorization technique is exponentially larger as a function of d.

3 Exact Bayesian learning of monotone conjunctions from positive examples

We now show how exact Bayesian inference over the class of monotone conjunctions can be performed efficiently, provided learning only occurs from positive examples $x_{1:n} \equiv 1_{1:n}$. Using the generalized distributive law [Aji and McEliece, 2000] we derive an alternative form of Equation 1 that can be straightforwardly computed in time $O(nd)$.

Proposition 6. *For all $n, d \in \mathbb{N}$, for all $a_{1:n} \in \mathcal{B}^{n \times d}$, then*

$$\xi_d(1_{1:n}; a_{1:n}) = \prod_{i=1}^{d} \left(\frac{1}{2} + \frac{1}{2} \left[\!\!\left[\bigwedge_{t=1}^{n} a_t^i \right]\!\!\right] \right).$$

Proof. Consider what happens when the expression

$$\prod_{i=1}^{d} \left(\frac{1}{2} + \frac{1}{2} \left[\!\!\left[\bigwedge_{t=1}^{n} a_t^i \right]\!\!\right] \right)$$

is expanded. We get a sum containing 2^d terms, that can be rewritten as

$$\begin{aligned} \sum_{\mathcal{S} \in \mathcal{P}_d} \frac{1}{2^d} \left[\!\!\left[\bigwedge_{i \in \mathcal{S}} \bigwedge_{t=1}^{n} a_t^i \right]\!\!\right] &= \sum_{\mathcal{S} \in \mathcal{P}_d} \frac{1}{2^d} [\![h_{\mathcal{S}}(a_{1:n}) = 1_{1:n}]\!] \\ &= \xi_d(1_{1:n}; a_{1:n}). \end{aligned}$$

where the second equality follows from Equation 1 and the first from

$$\begin{aligned} \nu_{\mathcal{S}}(1_{1:n} | a_{1:n}) &= [\![h_{\mathcal{S}}(a_{1:n}) = 1_{1:n}]\!] \\ &= \bigwedge_{t=1}^{n} h_{\mathcal{S}}(a_t) = \bigwedge_{t=1}^{n} \bigwedge_{i \in \mathcal{S}} a_t^i = \bigwedge_{i \in \mathcal{S}} \bigwedge_{t=1}^{n} a_t^i. \end{aligned}$$

On MAP model selection from positive examples. If we further parametrize the right hand side of Proposition 6 by introducing a hyper-parameter $\alpha \in (0, 1)$ to give

$$\xi_d^{\alpha}(1_{1:n}; a_{1:n}) := \prod_{i=1}^{d} \left((1 - \alpha) + \alpha \left[\!\!\left[\bigwedge_{t=1}^{n} a_t^i \right]\!\!\right] \right), \quad (3)$$

we get a family of tractable Bayesian algorithms for learning monotone conjunctions from positive examples. The

α parameter controls the bias toward smaller or larger formulas; smaller formulas are favored if $\alpha < \frac{1}{2}$, while larger formulas are favored if $\alpha > \frac{1}{2}$, with the expected formula length being αd. If we denote the prior over $\mathcal{S}$ by $w_\alpha(\mathcal{S}) := \alpha^{|\mathcal{S}|}(1-\alpha)^{d-|\mathcal{S}|}$, we get the mixture $\xi_d^\alpha(x_{1:n}; a_{1:n}) = \sum_{\mathcal{S}\in\mathcal{P}_d} w_\alpha(\mathcal{S})\nu_{\mathcal{S}}(x_{1:n}; a_{1:n})$. From this we can read off the maximum a posteriori (MAP) model

$$\begin{aligned}\mathcal{S}'_n &:= \arg\max_{\mathcal{S}\in\mathcal{P}_d} w_\alpha(\mathcal{S} \mid x_{1:n}; a_{1:n}) \\ &= \arg\max_{\mathcal{S}\in\mathcal{P}_d} w_\alpha(\mathcal{S})\ \nu_{\mathcal{S}}(x_{1:n}|a_{1:n})\end{aligned}$$

under various choices of α. For positive examples (i.e. $x_{1:n} = 1_{1:n}$), this can be rewritten as

$$\mathcal{S}'_n = \arg\max_{\mathcal{S}\in\mathcal{P}_d} w_\alpha(\mathcal{S}) \left[\!\!\left[\bigwedge_{i\in\mathcal{S}} \bigwedge_{t=1}^{n} a_t^i \right]\!\!\right].$$

For $\alpha > \frac{1}{2}$, the MAP model $\mathcal{S}'_n$ at time n is unique, and is given by

$$\mathcal{S}'_n = \left\{ i \in \{1,\ldots,d\} : \bigwedge_{t=1}^{n} a_t^i \right\}. \tag{4}$$

For $\alpha = \frac{1}{2}$, a MAP model is any subset of $\mathcal{S}'_n$ as defined by Equation 4. For $\alpha < \frac{1}{2}$, the MAP model is $\{\}$. Finally, we remark that the above results allow for a Bayesian interpretation of Valiant's algorithm for PAC learning monotone conjunctions. His method, described in Section 5 of [Valiant, 1984], after seeing n positive examples, outputs the concept $\bigwedge_{i\in\mathcal{S}'_n} x_i$; in other words, his method can be interpreted as doing MAP model selection using a prior belonging to the above family when $\alpha > \frac{1}{2}$.

A Heuristic Predictor. Next we discuss a heuristic prediction method that incorporates Proposition 6 to efficiently perform Bayesian learning on only the positive examples. Consider the probabilistic predictor ξ_d^+ defined by

$$\xi_d^+(x_n|x_{<n}; a_{1:n}) := \frac{\xi_d(x_{<n}^+ x_n; a_{<n}^+ a_n)}{\xi_d(x_{<n}^+; a_{<n}^+)}, \tag{5}$$

where we denote by $a_{<n}^+$ the subsequence of $a_{<n}$ formed by deleting the a_k where $x_k = 0$, for $1 \le k \le n-1$. Similarly, $x_{<n}^+$ denotes to the subsequence formed from $x_{<n}$ by deleting the x_k where $x_k = 0$. Note that since $\xi_d(x_{<n}^+ 0; a_{<n}^+ a_n) = \xi_d(x_{<n}^+; a_{<n}^+)\left(1 - \xi_d(1|x_{<n}^+; a_{<n}^+ a_n)\right)$, Equation 3 can be used to efficiently compute Equation 5. To further save computation, the values of the $\bigwedge_{t=1}^n a_t^i$ terms can be incrementally maintained using $O(d)$ space. Using these techniques, each prediction can be made in $O(d)$ time. Of course the main limitation with this approach is that it ignores all of the information contained within the negative instances. It is easy to see that this has disastrous implications for the loss. For example, consider what happens if a sequence of n identical negative instances are supplied. Since no learning will ever occur, a positive constant loss will be suffered at every timestep, leading to a loss that grows linearly in n. This suggests that some form of memorization of negative examples is necessary.

Discussion. There are many noteworthy model classes (for example, see the work of Willems, Shamir, Erven, Koolen, Gyorgi, Veness et al 1995; 1996; 1997; 1999; 2007; 2011; 2012a; 2012; 2013) in which it is possible to efficiently perform exact Bayesian inference over large discrete spaces. The common theme amongst these techniques is the careful design of priors that allow for the application of either the generalized distributive law [Aji and McEliece, 2000] and/or dynamic programming to avoid the combinatorial explosion caused by naively averaging the output of many models.

4 Three efficient, low loss algorithms

We now apply the ideas from the previous sections to construct an efficient online algorithm whose loss is bounded by $O(d^2)$. The main idea is to extend the heuristic predictor so that it simultaneously memorizes negative instances while also favoring predictions of 0 in cases where the Bayesian learning component of the model is unsure. The intuition is that by adding memory, there can be at most 2^d times where a positive loss is suffered. Moving the α parameter closer towards 1 causes the Bayesian component to more heavily weigh the predictions of the longer Boolean expressions consistent with the data, which has the effect of biasing the predictions more towards 0 when the model is unsure. Although this causes the loss suffered on positive instances to increase, we can show that this effect is relatively minor. Our main contribution is to show that by setting $\alpha = 2^{-d/2^d}$, the loss suffered on both positive and negative instances is balanced in the sense that the loss can now be upper bounded by $O(d^2)$.

Algorithm. The algorithm works very similarly to the previously defined heuristic predictor, with the following two modifications: firstly, the set of all negative instances is incrementally maintained within a set $\mathcal{A}$, with 0 being predicted deterministically if the current negative instance has been seen before; secondly, the ξ_d terms in Equation 5 are replaced with ξ_d^α, with $\alpha = 2^{-d/2^d}$. More formally,

$$\zeta_d(x_t|x_{<t}; a_{1:t}) := \begin{cases} 1 - x_t & \text{if } a_t \in \mathcal{A}; \\ \frac{\xi_d^\alpha(x_{<t}^+ x_t; a_{<t}^+ a_t)}{\xi_d^\alpha(x_{<t}^+; a_{<t}^+)} & \text{otherwise.} \end{cases} \tag{6}$$

Pseudocode is given in Algorithm 1. The algorithm begins by initializing the weights and the set of negative instances $\mathcal{A}$. Next, at each time step t, a distribution $p_t(\cdot; a_t)$ over $\{0,1\}$ is computed. If a_t has previously been seen as a negative example, the algorithm predicts 0 deterministically. Otherwise it makes its prediction using the previously defined Bayesian predictor (with $\alpha = 2^{-d/2^d}$) that is trained from only positive examples. The justification for Line 7 is as follows: First note that w_i is always equal to the conjunction of the ith component of the inputs corresponding to the positive examples occurring before time t, or more formally

$$w_i = \bigwedge_{\tau=1:a_\tau\notin\mathcal{A}}^{t-1} a_\tau,$$

which by Equation 3 implies

$$\xi_d^\alpha(x_{<t}^+; a_{<t}^+) = \prod_{i=1}^{d} [(1-\alpha) + \alpha w_i].$$

Algorithm 1 $\zeta_d(x_{1:n}; a_{1:n})$

1: $w_i \leftarrow 1$ for $1 \le i \le d$
2: $\mathcal{A} \leftarrow \{\}$; $\alpha \leftarrow 2^{-d/2^d}$; $r \leftarrow 1$
3: **for** $t = 1$ to n **do**
4: Observe a_t
5: **if** $a_t \in \mathcal{A}$ **then**
6: $p_t(1; a_t) \leftarrow 0$; $p_t(0; a_t) \leftarrow 1$
7: **else** $p_t(1; a_t) \leftarrow \prod_{i=1}^{d} \frac{(1-\alpha)+\alpha w_i a_t^i}{(1-\alpha)+\alpha w_i}$
8: $p_t(0; a_t) \leftarrow 1 - p_t(1; a_t)$
9: **endif**
10: Observe x_t and suffer a loss of $-\log p_t(x_t; a_t)$
11: **if** $x_t = 1$ **then**
12: **for** $i = 1$ to d **do** $w_i \leftarrow w_i\, a_t^i$ **end for**
13: **else** $\mathcal{A} \leftarrow \mathcal{A} \cup \{a_t\}$
14: **endif**
15: $r \leftarrow p_t(x_t; a_t) r$
16: **end for**
17: **return** r

Similarly $\xi_d^\alpha(x^+_{<t}1; a^+_{<t}a_t) = \prod_{i=1}^d [(1-\alpha) + \alpha w_i a_t^i]$, which by Equation 6 for $a_t \notin \mathcal{A}$ implies

$$\zeta_d(x_t = 1 | x_{<t}; a_{1:t}) = \frac{\prod_{i=1}^d [(1-\alpha) + \alpha w_i a_t^i]}{\prod_{i=1}^d [(1-\alpha) + \alpha w_i]} = p_t(1; a_t).$$

Trivially $p_t(x_t; a_t) = 1 - x_t = \zeta_d(x_t | x_{<t}; a_{1:t})$ for $a_t \in \mathcal{A}$ from Line 6. After the label is revealed and a loss is suffered, the algorithm either updates $\mathcal{A}$ to remember the negative instance or updates its weights w_i, with the cycle continuing. Overall Algorithm 1 requires $O(nd)$ space and processes each example in $O(d)$ time.

Analysis. We now analyze the cumulative log-loss when using ζ_d in place of an arbitrary monotone conjunction corresponding to some $\mathcal{S}^* \in \mathcal{P}_d$.

Lemma 7. *For all $d \in \mathbb{N} \setminus \{1\}$, $-\log\left(1 - 2^{-d/2^d}\right) \le d$.*

Proof. We have that

$$\begin{aligned} -\ln(1 - \mathrm{e}^{-d/\mathrm{e}^d}) &\le -\ln\left(1 - \frac{1}{1 + d/\mathrm{e}^d}\right) \\ &= \ln \frac{1 + d/\mathrm{e}^d}{d/\mathrm{e}^d} = d + \ln\left(\frac{1}{d} + \frac{1}{\mathrm{e}^d}\right) \le d. \end{aligned} \quad (7)$$

The first bound follows from $\mathrm{e}^{-x} \le \frac{1}{1+x}$. The equalities are simple algebra. The last bound follows from $\frac{1}{d} + \frac{1}{\mathrm{e}^d} \le 1$ for $d \ge 2$. (A similar lower bound $-\ln(1 - \mathrm{e}^{-d/\mathrm{e}^d}) \ge -\ln(1 - (1 - d/\mathrm{e}^d)) = d - \ln d$ shows that the bound is rather tight for large d). Substituting $d \rightsquigarrow d \ln 2$ in (7) and dividing by $\ln 2$ proves the lemma. □

Theorem 8. *If $x_{1:n}$ is generated by a hypothesis $h_{\mathcal{S}^*}$ such that $\mathcal{S}^* \in \mathcal{P}_d$ then for all $n \in \mathbb{N}$, for all $d \in \mathbb{N} \setminus \{1\}$, for all $x_{1:n} \in \mathcal{B}^n$, for all $a_{1:n} \in \mathcal{B}^{n \times d}$, we have that $\mathcal{L}_n(\zeta_d) \le 2d^2$.*

Proof. We begin by decomposing the loss into two terms, one for the positive and one for the negative instances.

$$\begin{aligned} \mathcal{L}_n(\zeta_d) &= \sum_{t=1}^{n} -\log \zeta_d(x_t | x_{<t}; a_{1:t}) \\ &= \sum_{\substack{t \in [1,n] \\ \text{s.t. } x_t = 1}} -\log \zeta_d(x_t = 1 | x_{<t}; a_{1:t}) \\ &\qquad + \sum_{\substack{t \in [1,n] \\ \text{s.t. } x_t = 0}} -\log \zeta_d(x_t = 0 | x_{<t}; a_{1:t}) \\ &= \sum_{\substack{t \in [1,n] \\ \text{s.t. } x_t = 1}} -\log \frac{\xi_d^\alpha(x^+_{1:t}; a^+_{1:t})}{\xi_d^\alpha(x^+_{<t}; a^+_{<t})} + \sum_{\substack{t \in [1,n] \\ \text{s.t. } x_t = 0}} -\log \zeta_d(x_t = 0 | x_{<t}; a_{1:t}) \\ &= -\log \xi_d^\alpha(x^+_{1:n}; a^+_{1:n}) + \sum_{\substack{t \in [1,n] \\ \text{s.t. } x_t = 0}} -\log \zeta_d(x_t = 0 | x_{<t}; a_{1:t}), \end{aligned}$$

where we have used the notation $[1, d] := \{1, 2, \ldots, d\}$. The final step follows since the left summand telescopes. Next we will upper bound the left and right summands separately. For $\alpha \in (0.5, 1)$, we have for the left term that

$$\begin{aligned} -\log \xi_d^\alpha(x^+_{1:n}; a^+_{1:n}) &\le -\log\left(\alpha^{|\mathcal{S}^*|}(1-\alpha)^{d-|\mathcal{S}^*|}\right) \\ &\le -d \log(1-\alpha). \end{aligned} \quad (8)$$

Now, let $\mathcal{U} := \left\{t \in [1,n] : x_t = 0 \wedge \bigwedge_{i=1}^{t-1}(a_t \ne a_i)\right\}$ denote the set of time indices where a particular negative instance is seen for the first time and let

$$\mathcal{D}_t := \left\{ i \in [1,d] : \bigwedge_{\tau=1}^{t-1} \left(\neg x_\tau \vee a_\tau^i\right) \right\} \quad (9)$$

denote the indices of the variables not ruled out from the positive examples occurring before time t. Given these definitions, we have that

$$\begin{aligned} &\xi_d^\alpha(x^+_{<t}; a^+_{<t}) \\ &= \sum_{\mathcal{S} \in \mathcal{P}_d} \alpha^{|\mathcal{S}|}(1-\alpha)^{d-|\mathcal{S}|} \left[\!\left[h_{\mathcal{S}}(a^+_{<t}) = x^+_{<t}\right]\!\right] \\ &= \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{D}_t)} \alpha^{|\mathcal{S}|}(1-\alpha)^{d-|\mathcal{S}|} \\ &= (1-\alpha)^{d-|\mathcal{D}_t|} \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{D}_t)} \alpha^{|\mathcal{D}_t|}(1-\alpha)^{|\mathcal{D}_t|-|\mathcal{S}|} \\ &= (1-\alpha)^{d-|\mathcal{D}_t|}. \end{aligned} \quad (10)$$

and similarly for $t \in \mathcal{U}$

$$\begin{aligned} &\xi_d^\alpha(x^+_{<t}0; a^+_{<t}a^+) \\ &= \sum_{\mathcal{S} \in \mathcal{P}_d} \alpha^{|\mathcal{S}|}(1-\alpha)^{d-|\mathcal{S}|} \left[\!\left[h_{\mathcal{S}}(a^+_{<t}a_t) = x^+_{<t}0\right]\!\right] \\ &= \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{D}_t)} \alpha^{|\mathcal{S}|}(1-\alpha)^{d-|\mathcal{S}|} \left[\!\left[h_{\mathcal{S}}(a_t) = 0\right]\!\right] \\ &\ge \alpha^{|\mathcal{D}_t|}(1-\alpha)^{d-|\mathcal{D}_t|}. \end{aligned} \quad (11)$$

The last inequality follows by dropping all terms in the sum except for the term corresponding the maximally sized conjunction $\bigwedge_{t \in \mathcal{D}_t} x_t$, which must evaluate to 0 given a_t, since

$\mathcal{S}^* \subseteq \mathcal{D}_t$ and $t \in \mathcal{U}$. Using the above, we can now upper bound the right term

$$\sum_{\substack{t\in[1,n]\\ \text{s.t. } x_t=0}} -\log \zeta_d(x_t=0|x_{<t};a_{1:t})$$
$$\overset{(a)}{=} \sum_{t\in\mathcal{U}} -\log \frac{\xi_d^\alpha(x^+_{<t}0;a^+_{<t}a^+)}{\xi_d^\alpha(x^+_{<t};a^+_{<t})} \overset{(b)}{\leq} \sum_{t\in\mathcal{U}} -\log \alpha^{|\mathcal{D}_t|}$$
$$\overset{(c)}{\leq} \sum_{t\in\mathcal{U}} -d\log\alpha \overset{(d)}{\leq} -d\,2^d\log\alpha. \qquad (12)$$

Step (a) follows from the definition of ζ_d and $\mathcal{U}$ (recall that a positive loss occurs only the first time an input vector is seen). Step (b) follows from Equations 10 and 11. Step (c) follows since $|\mathcal{D}_t| \leq d$ by definition. Step (d) follows since there are at most 2^d distinct Boolean vectors of side information.

Now, by picking $\alpha = 2^{-d/2^d}$, we have from Equation 8 and Lemma 7 that $-\log \xi_d^\alpha(x^+_{1:n};a^+_{1:n}) \leq d^2$. Similarly, from Equation 12 we have that

$$\sum_{\substack{t\in[1,n]\\ \text{s.t. } x_t=0}} -\log \zeta_d(x_t=0|x_{<t};a_{1:t}) \leq -d\,2^d\log 2^{-d/2^d} = d^2.$$

Thus by summing our previous two upper bounds, we get

$$\mathcal{L}_n(\zeta_d) = -\log \xi_d^\alpha(x^+_{1:n};a^+_{1:n}) + \sum_{\substack{t\in[1,n]\\ \text{s.t. } x_t=0}} -\log \zeta_d(x_t=0|x_{<t};a_{1:t}) \leq 2d^2.$$

□

A better space/time complexity tradeoff. Although the loss of Algorithm 1 is no more than $2d^2$ (and independent of n), a significant practical drawback is its $O(nd)$ space complexity. We now present an alternative algorithm which reduces the space complexity to $O(d)$, at the small price of increasing the worst case loss to no more than $O(d\log n)$. The main intuition for our next algorithm follows from the loss analysis of Algorithm 1. Our proof of Theorem 8 led to a choice of $\alpha = 2^{-d/2^d}$, which essentially causes each probabilistic prediction to be largely determined by the prediction made by the longest conjunction consistent with the already seen positive examples. This observation led us to consider Algorithm 2, which uses a smoothed of this. More formally,

$$\pi_d(x_t|x_{<t};a_{1:t}) := \tfrac{t}{t+1}\left[\!\!\left[\bigwedge_{i\in\mathcal{D}_t} a_t^i = x_t\right]\!\!\right] + \tfrac{1}{t+1}\left[\!\!\left[\bigwedge_{i\in\mathcal{D}_t} a_t^i \neq x_t\right]\!\!\right],$$

where $\mathcal{D}_t$ denotes the indices of the variables not ruled out from the positive examples occurring before time n. Pseudocode for implementing this procedure in $O(d)$ time per iteration, using $O(d)$ space, is given in Algorithm 2. The set $\mathcal{D}$ incrementally maintains the set $\mathcal{D}_t$. Compared to Algorithm 1, the key computational advantage of this approach is that it doesn't need to remember the negative instances. We next upper bound the loss of Algorithm 2.

Theorem 9. *If $x_{1:n}$ is generated by a hypothesis $h_{\mathcal{S}^*}$ such that $\mathcal{S}^* \in \mathcal{P}_d$, then for all $n \in \mathbb{N}$, for all $d \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{B}^n$, for all $a_{1:n} \in \mathcal{B}^{n\times d}$, we have that $\mathcal{L}_n(\pi_d) \leq (d+1)\log(n+1)$.*

Algorithm 2 $\pi_d(x_{1:n};a_{1:n})$

1: $\mathcal{D} \leftarrow \{1,2,\ldots,d\}$; $r \leftarrow 1$
2: **for** $t = 1$ to n **do**
3: Observe a_t
4: **if** $\prod_{i\in\mathcal{D}} a_t^i = 1$
5: **then** $p_t(1;a_t) := t/(t+1)$; $p_t(0;a_t) := 1/(t+1)$
6: **else** $p_t(1;a_t) := 1/(t+1)$; $p_t(0;a_t) := t/(t+1)$
7: **endif**
8: Observe x_t and suffer a loss of $-\log p_t(x_t;a_t)$.
9: **if** $x_t = 1$ **then** $\mathcal{D} \leftarrow \mathcal{D} \setminus \{i \in \{1,\ldots,d\} : a_t^i = 0\}$
endif
10: $r \leftarrow p_t(x_t;a_t)\, r$
11: **end for**
12: **return** r

Proof. As $x_{1:n}$ is generated by some $h_{\mathcal{S}^*}$ where $\mathcal{S}^* \in \mathcal{P}_d$, we have that $\mathcal{L}_n(\pi_d) = -\log \pi_d(x_{1:n};a_{1:n})$. We break the analysis of this term into two cases. At any time $1 \leq t \leq n$, we have either: Case (i): $\bigwedge_{i\in\mathcal{D}} a_t^i = \top$, which implies $h_{\mathcal{S}}(a_t) = 1$ for all $\mathcal{S} \subseteq \mathcal{D} = \mathcal{D}_t$. As the data is generated by some h_{S^*}, we must have $\mathcal{S}^* \subseteq \mathcal{D}_t$ and therefore $x_t = 1$, so a loss of $-\log \frac{t}{t+1}$ is suffered. Case (ii): $\bigwedge_{i\in\mathcal{D}} a_t^i = \bot$, where one of two situations occur: a) if $x_t = 0$ we suffer a loss of $-\log \frac{t}{t+1}$; otherwise b) we suffer a loss of $-\log(1/(t+1)) = \log(t+1)$ and at least one element in $\mathcal{D}$ gets removed. Notice that as the set $\mathcal{D}$ is initialized with d elements, case b) can only occur at most d times given any sequence of data.

Finally, notice that Case (ii b) contributes at most at d times $\log(n+1)$ to the loss. On the other hand, $\log \frac{t+1}{t}$ is suffered for each t of case (i) and (ii a), which can be upper bounded by $\sum_{t=1}^n \log \frac{t+1}{t} = \log(n+1)$. Together they give the desired upper bound $(d+1)\log(n+1)$. □

We also remark that Algorithm 2 could have been defined so that $p_t(1;a_t) = 1$ whenever $\bigwedge_{i\in\mathcal{A}} a_t^i = 1$. The reason we instead predicted 1 with probability $t/(t+1)$ is that it allows Algorithm 2 to avoid suffering an infinite loss if the data is not generated by some monotone conjunction. If however we are prepared to assume the realizable case, one can modify Algorithm 2 so as to also have finite cumulative loss for any possible *infinite* sequence of examples. First define

$$\mathcal{T}_n^- := \left\{ t \in [1,n] : x_t = 0 \ \wedge \left(\prod_{i\in\mathcal{D}} a_t^i = 0\right)\right\}$$

and

$$\mathcal{T}_n^+ := \left\{ t \in [1,n] : x_t = 1 \ \wedge \left(\prod_{i\in\mathcal{D}} a_t^i = 0\right)\right\}.$$

Next we change the probabilities assigned in Line 5 to

$$p_t(1;a_t) = 1, \quad p_t(0;a_t) = 0$$

and in Line 6 to

$$p_t(1;a_t) = \frac{1}{(n_t^-+1)^2}, \quad p_t(0;a_t) = \frac{n_t^-(n_t^-+2)}{(n_t^-+1)^2}$$

where $n_t^- := |\mathcal{T}_{t-1}^-| + 1$. If we denote this variant by π'_d, we can show the following theorem.

Theorem 10. *If $x_{1:\infty} \in \mathcal{B}^\infty$ is generated by a hypothesis $h_{\mathcal{S}^*}$ such that $\mathcal{S}^* \in \mathcal{P}_d$, then for all $d \in \mathbb{N}$, for all $a_{1:\infty} \in (\mathcal{B}^d)^\infty$, we have that $\mathcal{L}_\infty(\pi'_d) < \infty$.*

Proof. Exploiting the assumption that $x_{1:n}$ is generated by a hypothesis $h_{\mathcal{S}^*}$, the cumulative loss can be expressed as

$$\begin{aligned}\mathcal{L}_n(\pi'_d) &= -\log\left(\prod_{t\in\mathcal{T}_n^-}\frac{n_t^-(n_t^-+2)}{(n_t^-+1)^2}\prod_{t\in\mathcal{T}_n^+}\frac{1}{(n_t^-+1)^2}\right)\\ &= -\log\left(\prod_{i=1}^{|\mathcal{T}_n^-|}\frac{i(i+2)}{(i+1)^2}\right)+2\sum_{t\in\mathcal{T}_n^+}\log(n_t^-+1).\end{aligned} \tag{13}$$

Now we can bound the first summand in Equation 13 by

$$\begin{aligned}&-\log\left(\prod_{i=1}^{|\mathcal{T}_n^-|}\frac{i(i+2)}{(i+1)^2}\right)\\ &= -\sum_{i=1}^{|\mathcal{T}_n^-|}\log i-\sum_{i=1}^{|\mathcal{T}_n^-|}\log(i+2)+2\sum_{i=1}^{|\mathcal{T}_n^-|}\log(i+1)\\ &= -\sum_{i=1}^{|\mathcal{T}_n^-|}\log i+\sum_{i=2}^{|\mathcal{T}_n^-|+1}\log i\\ &\qquad -\sum_{i=2}^{|\mathcal{T}_n^-|+1}\log(i+1)+\sum_{i=1}^{|\mathcal{T}_n^-|}\log(i+1)\\ &= \log(|\mathcal{T}_n^-|+1)+1-\log(|\mathcal{T}_n^-|+2)\\ &\le 1.\end{aligned}$$

Thus we have

$$\begin{aligned}\mathcal{L}_n(\pi'_d) &\le 1+2\sum_{t\in\mathcal{T}^+}\log(n_t^-+1)\\ &\le 1+2d\max_{t\in\mathcal{T}_n^+}\log(n_t^-+1)\quad(\text{since } |\mathcal{T}_n^+|\le d)\\ &\le 1+2d\log\left(\max(\mathcal{T}_n^+)+1\right).\end{aligned}$$

Since $|\mathcal{T}_n^+| \le d$, the above bound implies $\mathcal{L}_\infty(\pi'_d) < \infty$. □

There are two main drawbacks with this approach. The first is that an infinite loss can be suffered in the non-realizable case. The second is that, in the case of finite sequences, the loss is upper bounded (see the proof of Theorem 10) by $2d\log(n+1)$ instead of $(d+1)\log(n+1)$ as per Theorem 9.

A method that exploits simplicity in the hypothesis space. The well-known WINNOW1 algorithm [Littlestone, 1988] can also be used to learn monotone conjunctions online under the 0/1 loss via the transformation given in Example 5 of the original paper. Provided a hypothesis $h_{\mathcal{S}^*}$ with $\mathcal{S}^* \in \mathcal{P}_d$ generates the data, the total number of mistakes this algorithm makes is known to be upper bounded by

$$\alpha|\mathcal{S}^*|\left(\log_\alpha\theta+1\right)+\frac{d}{\theta} \tag{14}$$

whenever the algorithm hyper-parameters satisfy both $\alpha > 1$ and $\theta \ge 1/\alpha$. For example, by setting $\theta = d/2$ and $\alpha = 2$, one can bound the number of mistakes by $2|\mathcal{S}^*|\log d + 2$. This particular parametrization is well suited to the case where $|\mathcal{S}^*| \ll d$; that is, whenever many features are irrelevant.

Algorithm 3 $\omega_d(x_{1:n}; a_{1:n})$

Require: $\alpha, \theta \in \mathbb{R}$ such that $\alpha > 1, \theta \ge 1/\alpha$

$w_i \leftarrow 1, \ \forall 1 \le i \le d$
for $t = 1$ to n **do**
  Observe a_t
  $y_t \leftarrow \left[\!\left[\sum_{i=1}^d w_i(1-a_t^i) \le \theta\right]\!\right]$
  $p_t(y_t; a_t) := t/(t+1)$
  $p_t(1-y_t; a_t) := 1/(t+1)$
  Observe x_t and suffer a loss of $-\log p_t(x_t; a_t)$.
  if $y_t \ne x_t$ **then**
    if $x_t = 1$ **then**
      $w_i \leftarrow 0$ if $a_t^i = 0, \ \forall 1 \le i \le d$
    else
      $w_i \leftarrow \alpha w_i$ if $a_t^i = 0, \ \forall 1 \le i \le d$
    end if
  end if
  $r \leftarrow p_t(x_t; a_t)\, r$
end for
return r

Here we describe an adaptation of this method for the logarithmic loss along with a worst-case analysis. Although the resultant algorithm will not enjoy as strong a loss guarantee as Algorithm 2 in general, one would prefer its guarantees in situations where $|\mathcal{S}^*| \ll d$. The main idea is to assign probability $t/(t+1)$ at time t to the class predicted by WINNOW1 (as applied to monotone conjunctions). Pseudocode for this procedure is given in Algorithm 3.

The following theorem bounds the cumulative loss. Compared with Theorem 9, here we see that a multiplicative dependence on $O(|\mathcal{S}^*|\log d)$ is introduced in place of the previous $O(d)$, which is preferred whenever $|\mathcal{S}^*| \ll d$.

Theorem 11. *If $x_{1:n}$ is generated by a hypothesis $h_{\mathcal{S}^*}$ such that $\mathcal{S}^* \in \mathcal{P}_d$, then for all $n \in \mathbb{N}$, for all $d \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{B}^n$, for all $a_{1:n} \in \mathcal{B}^{n\times d}$, we have that*

$$\mathcal{L}_n(\omega_d) \le \left(\alpha|\mathcal{S}^*|\left(\log_\alpha\theta+1\right)+\frac{d}{\theta}+1\right)\log\left(n+1\right).$$

In particular, if $\theta = d/2$ and $\alpha = 2$ then

$$\mathcal{L}_n(\omega_d) \le \left(2|\mathcal{S}^*|\log d+2\right)\log\left(n+1\right).$$

Proof. Let $\mathcal{M}_n$ denote the set of times where the original WINNOW1 algorithm would make a mistaken prediction, i.e.

$$\mathcal{M}_n := \{t \in [1,n] : y_t \ne x_t\}.$$

Also let $\overline{\mathcal{M}}_n := [1,n] \setminus \mathcal{M}_n$. We can now bound the cumulative loss by

$$\mathcal{L}_n(\omega_d) = -\log\omega_d(x_{1:n}; a_{1:n})$$

$$
\begin{aligned}
&\overset{(a)}{=} -\log \prod_{t\in\mathcal{M}_n}\left(\frac{1}{t+1}\right)\prod_{t\in\overline{\mathcal{M}}_n}\left(\frac{t}{t+1}\right)\\
&= -\log\left(\prod_{t\in\mathcal{M}_n}\frac{1}{t+1}\right) - \log\left(\prod_{t\in\overline{\mathcal{M}}_n}\frac{t}{t+1}\right)\\
&\overset{(b)}{\leq} -\log\left(\frac{1}{n+1}\right)^{\alpha k(\log_\alpha\theta+1)+d/\theta}\\
&\quad -\log\left(\prod_{t=1}^{n-\lfloor\alpha k(\log_\alpha\theta+1)+d/\theta\rfloor}\frac{t}{t+1}\right)\\
&= [\alpha k(\log_\alpha\theta+1)+d/\theta]\log(n+1)\\
&\quad +\log\left(n-\lfloor\alpha k(\log_\alpha\theta+1)+d/\theta\rfloor+1\right)\\
&\leq [\alpha k(\log_\alpha\theta+1)+d/\theta+1]\log(n+1).
\end{aligned}
$$

Step (a) follows from definition of Algorithm 3 and $\mathcal{M}_n$. Step (b) applies both Equation 14 and that $1/(n+1) \leq 1/(t+1)$ for all $1 \leq t \leq n$. □

5 Handling k-CNF Boolean functions

Finally, we describe how our techniques can be used to probabilistically predict the output of an unknown k-CNF function. Given a set of d variables $\{x_1,\ldots,x_d\}$, a k-CNF Boolean function is a conjunction of clauses $c_1 \wedge c_2 \wedge \cdots \wedge c_m$, where for $1 \leq y \leq m$, each clause c_y is a disjunction of k literals, with each literal being an element from $\{x_1,\ldots,x_d,\neg x_1,\ldots,\neg x_d\}$. The number of syntactically distinct clauses is therefore $(2d)^k$. We will use the notation $\mathcal{C}_d^k$ to denote the class of k-CNF Boolean formulas that can be formed from d variables.

The task of probabilistically predicting a k-CNF Boolean function of d variables can be reduced to that of probabilistically predicting a monotone conjunction over a larger space of input variables. We can directly use the same reduction as used by Valiant [1984] to show that the class of k-CNF Boolean functions is PAC-learnable. The main idea is to first transform the given side information $a \in \mathcal{B}^d$ into a new Boolean vector $c \in \mathcal{B}^{(2d)^k}$, where each component of c corresponds to the truth value for each distinct k-literal clause formed from the set of input variables $\{a^i\}_{i=1}^d$, and then run either Algorithm 1 or Algorithm 2 on this transformed input. In the case of Algorithm 1, this results in an online algorithm where each iteration takes $O(d^k)$ time; given n examples, the algorithm runs in $O(nd^k)$ time and uses $O(nd^k)$ space. Furthermore, if we denote the above process using either Algorithm 1 or Algorithm 2 as $\textsc{Alg}1_d^k$ or $\textsc{Alg}2_d^k$ respectively, then Theorems 8 and 9 allows us to upper bound the loss of each approach.

Corollary 12. *For all $n \in \mathbb{N}$, for all $k \in \mathbb{N}$, for any sequence of side information $a_{1:n} \in \mathcal{B}^{n\times d}$, if $x_{1:n}$ is generated from a hypothesis $h^* \in \mathcal{C}_d^k$, the loss of $\textsc{Alg}1_d^k$ and $\textsc{Alg}2_d^k$ with respect to h^* satisfies the upper bounds $\mathcal{L}_n(\textsc{Alg}1) \leq 2^{2k+1}d^{2k}$ and $\mathcal{L}_n(\textsc{Alg}2) \leq \left(2^k d^k+1\right)\log(n+1)$ respectively.*

6 Closing Remarks

This paper has provided three efficient, low-loss online algorithms for probabilistically predicting targets generated by some unknown k-CNF Boolean function of d Boolean variables in time (for fixed k) polynomial in d. The construction of Algorithm 1 is technically interesting in the sense that it is a hybrid Bayesian technique, which performs full Bayesian inference only on the positive examples, with a prior carefully chosen so that the loss suffered on negative examples is kept small. This approach may be potentially useful for more generally applying the ideas behind Bayesian inference or exponential weighted averaging in settings where a direct application would be computationally intractable. The more practical Algorithm 2 is less interpretable, but has $O(d)$ space complexity and a per instance time complexity of $O(d)$, while enjoying a loss within a multiplicative $\log n$ factor of the intractable Bayesian predictor using a uniform prior. The final method, a derivative of WINNOW, has favorable regret properties when many of the input features are expected to be irrelevant.

In terms of practical utility, we envision our techniques being most useful as component of a larger predictive ensemble. To give a concrete example, consider the statistical data compression setting, where the cumulative log-loss under some probabilistic model directly corresponds to the size of a file encoded using arithmetic encoding [Witten *et al.*, 1987]. Many strong statistical data compression techniques work by adaptively combining the outputs of many different probabilistic models. For example, the high performance PAQ compressor uses a technique known as geometric mixing [Mattern, 2013], to combine the outputs of many different contextual models in a principled fashion. Adding one of our techniques to such a predictive ensemble would give it the property that it could exploit k-CNF structure in places where it exists.

Acknowledgements. The authors would like to thank the following: Brendan McKay, for providing the proof of Theorem 4; Kee Siong Ng, for the suggestion to investigate the class of k-CNF formulas from an online, probabilistic perspective; Julien Cornebise, for some helpful comments and discussions; and finally to the anonymous reviewers for pointing out the connection to the WINNOW algorithm.

References

[Aji and McEliece, 2000] S.M. Aji and R.J. McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2):325–343, 2000.

[Bratko *et al.*, 2006] Andrej Bratko, Gordon V. Cormack, David R, Bogdan Filipi, Philip Chan, Thomas R. Lynam, and Thomas R. Lynam. Spam filtering using statistical data compression models. *Journal of Machine Learning Research (JMLR)*, 7:2673–2698, 2006.

[Cilibrasi and Vitányi, 2005] Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51:1523–1545, 2005.

[Frank *et al.*, 2000] Eibe Frank, Chang Chui, and Ian H. Witten. Text categorization using compression models. In *Proceedings of Data Compression Conference (DCC)*, pages 200–209. IEEE Computer Society Press, 2000.

[György *et al.*, 2011] A. György, T. Linder, and G. Lugosi. Efficient Tracking of Large Classes of Experts. *IEEE Transactions on Information Theory*, 58(11):6709–6725, 2011.

[Koolen *et al.*, 2012] Wouter M. Koolen, Dmitry Adamskiy, and Manfred K. Warmuth. Putting Bayes to sleep. In *Neural Information Processing Systems (NIPS)*, pages 135–143, 2012.

[Littlestone and Warmuth, 1994] Nick Littlestone and Manfred K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261, February 1994.

[Littlestone, 1988] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Machine Learning*, pages 285–318, 1988.

[Mattern, 2013] Christopher Mattern. Linear and Geometric Mixtures - Analysis. In *Proceedings of the 2013 Data Compression Conference*, DCC '13, pages 301–310. IEEE Computer Society, 2013.

[Shamir and Merhav, 1999] Gil I. Shamir and Neri Merhav. Low Complexity Sequential Lossless Coding for Piecewise Stationary Memoryless Sources. *IEEE Transactions on Information Theory*, 45:1498–1519, 1999.

[Vadhan, 2001] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[van Erven *et al.*, 2007] Tim van Erven, Peter Grünwald, and Steven de Rooij. Catching Up Faster in Bayesian Model Selection and Model Averaging. *Neural Information Processing Systems (NIPS)*, 2007.

[Veness *et al.*, 2011] Joel Veness, Kee Siong Ng, Marcus Hutter, William T. B. Uther, and David Silver. A monte-carlo AIXI approximation. *J. Artif. Intell. Res. (JAIR)*, 40:95–142, 2011.

[Veness *et al.*, 2012a] Joel Veness, Kee Siong Ng, Marcus Hutter, and Michael H. Bowling. Context Tree Switching. In *Data Compression Conference (DCC)*, pages 327–336, 2012.

[Veness *et al.*, 2012b] Joel Veness, Peter Sunehag, and Marcus Hutter. On Ensemble Techniques for AIXI Approximation. In *AGI*, pages 341–351, 2012.

[Veness *et al.*, 2013] J. Veness, M. White, M. Bowling, and A. Gyorgy. Partition Tree Weighting. In *Data Compression Conference (DCC), 2013*, pages 321–330, 2013.

[Veness *et al.*, 2015] Joel Veness, Marc G. Bellemare, Marcus Hutter, Alvin Chua, and Guillaume Desjardins. Compress and control. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3016–3023, 2015.

[Willems and Krom, 1997] F. Willems and M. Krom. Live-and-die coding for binary piecewise i.i.d. sources. In *IEEE International Symposium on Information Theory (ISIT)*, page 68, 1997.

[Willems *et al.*, 1995] Frans M.J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The Context Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.

[Willems, 1996] Frans M. J. Willems. Coding for a binary independent piecewise-identically-distributed source. *IEEE Transactions on Information Theory*, 42:2210–2217, 1996.

[Witten *et al.*, 1987] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30:520–540, June 1987.