Author manuscript, published in "ECAI'08: Eighteenth European Conference on Artificial Intelligence, Patras, Greece : "

lirmm-00315916, version 2 - 15 Mar 2009

Online Rule Learning via Weighted Model Counting

Frédéric Koriche[1]

Abstract. Online multiplicative weight-update learning algorithms, such as Winnow, have proven to behave remarkably for learning simple disjunctions with few relevant attributes. The purpose of this paper is to extend the Winnow algorithm to more expressive target concepts characterized by DNF formulas with few relevant rules. For such problems, the convergence of Winnow is still fast, since the number of mistakes increases only linearly with the number of attributes. Yet, the learner is confronted with an important computational barrier: during any prediction, it must evaluate a weighted sum of an exponential number of rules. To circumvent this issue, we convert the prediction problem into a Weighted Model Counting problem. The resulting algorithm, *SharpNow*, is an exact simulation of Winnow equipped with backtracking, caching, and decomposition techniques. Experiments on static and drifting problems demonstrate the performance of the algorithm in terms of accuracy and speed.

1 INTRODUCTION

A recurrent theme in Machine Learning is the development of online mistake-driven learning algorithms [4]. Such algorithms are "anytime learners" that can be interrupted at each instant to provide a prediction whose correctness is related to the number of mistakes that have been made so far. Basically, the underlying model takes place in a sequence of trials. At any time step, the learner is presented an observation and it is asked to predict its associated class. If the prediction is incorrect, we charge it one mistake.

In a landmark paper, Littlestone [13] introduced the Winnow algorithm, which has rapidly become the blueprint of many efficient online learners. Winnow resembles the Perceptron algorithm in its simplicity, but uses multiplicative, rather than additive, weight updates on input features. Consequently, when the target concept is a k out of n variable disjunction, the number of mistakes grows as $k \log n$ instead of kn. The fact that the dependence on n is reduced to logarithmic, rather than linear, makes this algorithm applicable even if the number of features is enormous.

This remarkable property opens the door to learning problems characterized by high-dimensional feature spaces. One of them concerns the well-known problem of *rule learning* which consists in identifying, from a collection of examples, a small set of rules that explains all the positive examples and none of the negative ones [10]. In the paradigm of concept learning, any rule theory can be viewed as a DNF formula, that is, a disjunction of conjunctive features. Based on this notion, Winnow can naturally be extended to rule theories by projecting the data into an higher-dimensional feature space in which any conjunctive feature is viewed as a basic attribute. The enhanced algorithm inherits of an increased expressiveness while preserving a strong learning power. Indeed, if the observed examples are vectors of n attributes taking values over a domain of size d then, providing that the number of all conjunctive features is bounded by $(d+1)^n$, the performance of Winnow degrades only linearly with the input dimension. In fact, for any target concept, the number of mistakes grows essentially as $kn\log(d+1)$, where k is the minimum number of rules needed to represent the concept into a rule theory.

In its primal form, Winnow maintains a vector that assigns a weight to each distinct feature. For rule learning, such an implementation is computationally prohibitive, since the number of possible rules grows exponentially with the input dimension. Specifically, during any prediction, the learner is confronted with the problem of computing the weighted sum of an exponential number of features.

Kernel methods have emerged as a standard approach for solving counting problems that arise from high-dimensional feature spaces. The underlying idea is to start from the dual form of the learning algorithm and to use a kernel function that simulates the target feature space while working with original input data. Specifically, in the setting of Boolean DNF formulas, efficient kernel functions have been obtained for the Perceptron algorithm and its maximum margin variants [15]. Unfortunately, it seems impossible to find an analogous result for the Winnow algorithm. Indeed, as observed by Khardon et al. [11], the *Kernel Winnow Prediction* problem is #P-hard, even for the restricted class of monotone DNF formulas.

Such a computational barrier does not imply that, in practice, the sole option for the learner is a brute force enumeration of its feature space. To this very point, in the AI literature, a great deal of attention has been devoted to a related problem, referred to as *Weighted Model Counting* [5, 9, 17]. The problem is to evaluate the sum of weights of all the assignments satisfying a CNF formula. The basic building block for most model counting algorithms is the Davis-Putnam (DP) procedure that performs a backtracking search in the space of candidate models [3]. Based on this procedure, recent programs such as *Cachet* [16] and *SharpSat* [19] can handle large instances by combining formula caching and decomposition into connected components.

The power of these techniques raises the natural question of whether the Kernel Winnow Prediction problem can be solved in practice by translation to Weighted Model Counting. This paper provides initial evidence that the answer is affirmative: such a translation can be indeed effective for learning, with fast convergence and speed, "sparse" target concepts involving a small number of relevant rules. The resulting algorithm, called *SharpNow*, is an exact simulation of kernel Winnow, equipped with backtracking, caching, and decomposition techniques. For sparse target concepts, it can efficiently handle large spaces of conjunctive features with an accuracy superior to that of kernel Perceptron-like algorithms.

The background about online rule learning can be found in section 2. The translation method and the SharpNow algorithm are presented in section 3. Experiments on both static and drifting problems are reported in section 4. Finally, section 5 concludes the paper with a discussion about related work and perspectives of further research.

[1] LIRMM, Université Monptellier 2, France, Frederic.Koriche@lirmm.fr

lirmm-00315916, version 2 - 15 Mar 2009

2 ONLINE RULE LEARNING

In the online learning model, the algorithm is presented with a series of examples $\{\mathbf{x}_t, y_t\}$ labeled by a target concept. At each time step t, the algorithm first outputs a class prediction $\hat{y}_t$ for the observation $\mathbf{x}_t$, and then updates its hypothesis based on the true class y_t. We adopt the convention that $y_t \in \{-1, +1\}$.

The learners of particular interest in this study are the Perceptron [14] and Winnow [13] algorithms applied to rule theories.

2.1 Rule Theories

Let $\{x_1, \cdots, x_n\}$ be a set of attributes taking values in a discrete domain of size d. For convenience, we assume a standard way of naming values, as a list of natural numbers. A *concept* is a function f from d^n to $\{-1, +1\}$, and an *example* is a vector $\mathbf{x}$ in d^n. We say that a series of examples $\{\mathbf{x}_t, y_t\}$ is *labeled* by a target concept f if $f(\mathbf{x}_t) = y_t$ for each index t.

An *atom* is an expression $x_i = j$, also denoted x_i^j, where x_i is an attribute and j a value. A conjunctive feature, or *rule*, is a conjunction of atoms, and a *rule theory* is a set of rules. A rule r *covers* an example $\mathbf{x}$ if for all atoms x_i^j in r, the value of x_i in $\mathbf{x}$ is j. A rule theory R *covers* an example $\mathbf{x}$ if at least one rule in R covers $\mathbf{x}$. For instance, the following expression is a theory with three rules

$$\begin{cases} \mathit{sky} = \mathit{sunny} \wedge \mathit{humidity} = \mathit{normal} \\ \mathit{sky} = \mathit{cloudy} \wedge \mathit{temp} = \mathit{mild} \\ \mathit{sky} = \mathit{rain} \wedge \mathit{wind} = \mathit{weak} \end{cases}$$

The example $(\mathit{sunny}, \mathit{normal}, \mathit{mild}, \mathit{weak})$ is classified as positive by this theory, because it is covered by the first rule.

It is well-known that any concept can be represented by an equivalent rule theory. The *rule size* of a concept f, denoted $|f|$, is the minimal number of rules needed to represent f as a rule theory.

The feature space $\mathbf{R}_{n,d}$ of all rules generated from n attributes taking values over a domain of size d is represented by an indexed set $\{r_1, \cdots, r_N\}$, where $N = (d+1)^n$. Given an example $\mathbf{x}$, the *feature expansion* of $\mathbf{x}$ onto $\mathbf{R}_{n,d}$ is a vector $\boldsymbol{\phi}(\mathbf{x})$ in $\{0,1\}^N$ where $\phi_i(\mathbf{x}) = 1$ if and only if r_i covers $\mathbf{x}$.

2.2 Perceptron

For sake of clarity, we examine the zero-threshold version of the Perceptron algorithm. Throughout its execution, it maintains a vector $\mathbf{w}_t$ in $\mathbb{R}^N$ which is initialized to $\mathbf{w}_0 = \mathbf{0}$. Upon receiving an example $\mathbf{x}_t$, the algorithm predicts using the rule

$$\hat{y}_t = \mathrm{sign}(\mathbf{w}_t \cdot \boldsymbol{\phi}(\mathbf{x}_t)) \tag{1}$$

No change is made to $\mathbf{w}_t$ if the prediction is correct. In case of mistake, the algorithm uses the additive rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \boldsymbol{\phi}(\mathbf{x}_t) \tag{2}$$

For rule theories, implementing the Perceptron in its the primal form is computationally infeasible since we would need to maintain a weight vector of $(d+1)^n$ size. Yet, it is well-known that the dual form of the algorithm is a linear combination of inner products formed by the current observation $\mathbf{x}_t$ and the previous examples $\{(\mathbf{x}_s, y_s)\}$ on which mistakes where made. In the setting of $\mathbf{R}_{n,d}$, each inner product $\boldsymbol{\phi}(\mathbf{x}_s) \cdot \boldsymbol{\phi}(\mathbf{x}_t)$ can be simulated by the kernel function

$$K(\mathbf{x}_s, \mathbf{x}_t) = 2^{|\{i : x_{s,i} = x_{t,i}\}|}$$

In particular, the kernel obtained by Sadohara [15] can be derived from $d = 2$. Based on the so-called kernel trick, the prediction rule 1 can be replaced with

$$\hat{y}_t = \mathrm{sign}\left(\sum_{s=1}^{m} y_s K(\mathbf{x}_s, \mathbf{x}_t)\right) \tag{3}$$

Each trial of the kernel Perceptron algorithm can thus be executed in polynomial time. Unfortunately, the algorithm can provably require many updates even for very simple rule theories [11].

Theorem 1. *There exists a target concept of polynomial rule size and a sequence of examples labeled by it which causes the kernel Perceptron algorithm to make* $2^{\Omega(n)}$ *mistakes.*

Importantly, this result still holds for most Perceptron-like algorithms, including the version parameterized with a learning rate and a nonzero threshold, and the recent maximum margin variants [12].

2.3 Winnow

The algorithm has a very similar structure. It takes as input two parameters: a learning rate η and a threshold θ, and maintains a vector $\mathbf{w}_t$ which is initialized to $\mathbf{w}_0 = \mathbf{1}$. Upon receiving an example $\mathbf{x}_t$, the algorithm predicts according to the rule

$$\hat{y}_t = \mathrm{sign}(\mathbf{w}_t \cdot \boldsymbol{\phi}(\mathbf{x}_t) - \theta) \tag{4}$$

Again no change is made to $\mathbf{w}_t$ if the prediction is correct. In case of mistake, the hypothesis is updated with a multiplicative rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t \exp(\eta y_t \boldsymbol{\phi}(\mathbf{x}_t)) \tag{5}$$

According to these specifications, the following result can be deduced by a simple adaptation of Winnow's amortized analysis [1].

Theorem 2. *Let* $\theta = \frac{\eta}{2 \sinh \eta}(d+1)^n$*. Then, for any target concept* f*, the number* m *of mistakes made by the Winnow algorithm over any sequence of examples labeled by* f *satisfies*

$$m \leq \frac{e^\eta + 1}{\eta}\left[1 + |f| n \log(d+1)\right]$$

Thus, the Winnow algorithm has a polynomial mistake bound for learning polynomial-size rule theories. However, the key difficulty is to provide a computationally efficient simulation of the algorithm. Specifically, the *Kernel Winnow Prediction* problem is to infer the sign of $\mathbf{w}_t \cdot \boldsymbol{\phi}(\mathbf{x}_t) - \theta$ for the last example of a given sequence $\{(\mathbf{x}_t, y_t)\}$ of examples labeled by a target polynomial-size rule theory, after applying the prediction rule 4 and the update rule 5 on the weight vector $\mathbf{w}_t$ for the previous examples. As shown in [11] this problem is #P-hard, which implies that, unless #P = P, there is no general construction that will run Winnow using kernel functions.

In a nutshell, the most important message to be gleaned from online rule learning is that both additive and multiplicative update algorithms are, in theory, limited by either computational efficiency or convergence reasons. On the one hand, even if kernel Perceptron-like algorithms may be executed efficiently, they can provably require an exponential number of mistakes and, on the other hand, even if the kernel Winnow algorithm has a polynomial mistake bound, it seems impossible to simulate its execution in polynomial time.

lirmm-00315916, version 2 - 15 Mar 2009

3 WEIGHTED MODEL COUNTING

Despite the undoubtable importance of the aforementioned results, it remains to be seen whether, in practice, the execution of kernel Winnow can be simulated using efficient techniques that have been developed for solving real-world #P problems. The key motivation of this paper is to convert the Kernel Winnow Prediction problem into a Weighted Model Counting problem, for which general and efficient model counting techniques can be applied.

3.1 The Translation Method

Informally, any instance of the Weighted Model Counting problem consists in a set of weighted clauses; the task is to evaluate the sum of weights of assignments satisfying these clauses [17]. The intuitive idea behind the translation method is just to ascribe a weighted clause to each labeled example that has led to a mistake.

To this end, we need to introduce additional definitions. Consider again a set $\{x_1, \cdots, x_n\}$ of attributes taking values over a discrete domain of size d. In the following, any rule defined over this vocabulary is viewed as an *assignment*, that is a set of atoms or, equivalently, a map that assigns to each atom a value in $\{0, 1\}$. A *literal* is an atom x_i^j or its negation $\neg x_i^j$ and a *clause* is a disjunction of literals. Given a rule r and a literal x_i^j (resp. $\neg x_i^j$), we say that r *satisfies* x_i^j (resp. $\neg x_i^j$), if $x_i^j \in r$ (resp. $x_i^j \notin r$). Given a rule r and a clause c, we say that r *satisfies* c if r satisfies at least one literal occurring in c. Based on these notions, the *feature expansion* of a clause c under $\mathbf{R}_{n,d}$ is a map $\boldsymbol{\phi}(c)$ in $[0, 1]^N$ where $\phi_i(c) = 1$ if and only if r_i satisfies c.

A *weighted clause* is an expression of the form (c, w) where c is a clause and w is a value in $\mathbb{R}$. Intuitively, w reflects how strong a constraint it is: the higher the weight, the greater the difference in likelihood between a rule that satisfies the constraint and one that does not. In this setting, any "unweighted" clause c is treated as an abbreviation of $(c, 0)$: it denotes a hard constraint that restricts the space of possible rules. A *weighted knowledge base* KB is a finite set of weighted clauses. The *weight* of KB, denoted $\|KB\|$, is

$$\|KB\| = \sum_{i=1}^{N} \prod_{(c,w) \in KB} w^{1-\phi_i(c)} \tag{6}$$

As usual, we take the convention that $0^0 = 1$ and $0^z = 0$ for any real number $z > 0$. Thus, the weight of the knowledge base KB is just the sum of weights of the assignments that are models of KB.

We are now in position to present the translation method. The learning algorithm starts with the knowledge base $KB_0 = \varnothing$. On seeing an example $\mathbf{x}_t$, the algorithm predicts according to

$$\hat{y}_t = \text{sign}\left(\|KB_t \cup \{\neg x_i^j : x_{t,i} \neq j\}\| - \theta\right) \tag{7}$$

Recall that in the above expression, each unary clause $\neg x_i^j$ is treated as an abbreviation of $(\neg x_i^j, 0)$. In case of mistake, KB_t is simply expanded with a weighted clause that conveys, in a concise form, the information gathered from $\mathbf{x}_t$ and y_t. In formal terms

$$KB_{t+1} = KB_t \cup \left(\bigvee \{x_i^j : x_{t,i} \neq j\}, e^{\eta y_t}\right) \tag{8}$$

For instance, suppose that our learner makes a mistake on the positive example $(\mathit{sunny}, \mathit{normal}, \mathit{mild}, \mathit{weak})$. Then its knowledge base is expanded with the weighted clause (c, e^{η}), where c is the disjunction of all atoms $\mathit{sky} = \mathit{cloudy}, \mathit{sky} = \mathit{rain}, \ldots$ that are false in the example. In the next prediction, the weight of any rule that violates c will be multiplied by e^{η}.

Algorithm 1: $\textsc{Wmc}(KB, A)$

Input: a weighted knowledge base KB and a set of atoms A
Output: the sum of weights of all subsets of A according to KB

```
if InCache(KB) then return GetFromCache(KB)
if IsLeaf(KB) then return 2^|A| * ∏_{(c,w)∈KB} w
weight ← 1
for each connected component KB_i in KB do
    let A_i be the set of atoms occurring in KB_i
    choose an atom a in A_i
    KB_{i,1} ← {(c − {a}, w) : (c, w) ∈ KB_i, a ∉ c}
    weight_1 ← Wmc(KB_{i,1}, A_i − {a})
    KB_{i,2} ← {(c − {¬a}, w) : (c, w) ∈ KB_i, ¬a ∉ c}
    weight_2 ← Wmc(KB_{i,2}, A_i − {a})
    SetToCache(KB_i, weight_1 + weight_2)
    weight ← weight * (weight_1 + weight_2)
return weight
```

3.2 The SharpNow Algorithm

In the spirit of model counting algorithms such as *Cachet* [16] and *SharpSat* [19], we begin to develop a procedure for evaluating the weight of knowledge base that combines backtracking search with formula caching and decomposition into connected components.

The procedure, called $\textsc{Wmc}$, takes as input a weighted knowledge base KB and a set of atoms A. Basically, the procedure performs a depth-first search in the tree of partial assignments generated from A. Notably, a leaf of the tree is reached whenever every clause occurring in KB is empty. In this case, the resulting weight can be evaluated by simply taking the product of weights of these clauses. Following [2], the depth-first search procedure is enhanced by using decomposition into connected components. Namely, by identifying in linear time the connected components in the constraint graph of KB, the resulting weight can be determined by multiplying together the weight of each connected component. Finally, the $\textsc{Wmc}$ procedure is equipped with a caching technique that prevents it from recomputing the same component. Because the length of weighted clauses is in $O(nd)$, we employ the hybrid coding scheme suggested in [19] that concisely encodes a set of clauses KB as a vector of indices. Notice that the technique of component caching is particularly relevant in the setting of online learning when the algorithm is susceptible to recompute many identical subtrees from one prediction to the next.

Proposition 3. *Let KB be a weighted knowledge base, $\mathbf{x}_t$ an example, and A_t the set of all atoms that are true in $\mathbf{x}_t$. Let A be the set of atoms in A_t that occur in KB, and $\overline{A} = A_t - A$. Then*

$$\|KB \cup \{\neg x_i^j : x_{t,i} \neq j\}\| = 2^{|\overline{A}|} * \textsc{Wmc}(KB, A)$$

Proof. Based on the completeness of the DP backtracking search procedure for model counting [3], we know that

$$\textsc{Wmc}(KB, A) = \sum_{r \subseteq A} \prod \{w : (c, w) \in KB, r \cap c = \varnothing\}$$

Let $KB' = KB \cup \{\neg x_i^j : x_{t,i} \neq j\}$. From definition 6, we can infer

$$\begin{aligned} \|KB'\| &= \sum_{r \subseteq A_t} \prod \{w : (c, w) \in KB, r \cap c = \varnothing\} \\ &= 2^{|\overline{A}|} \textsc{Wmc}(KB, A) \end{aligned}$$

lirmm-00315916, version 2 - 15 Mar 2009

With these notions in hand, we can now present the algorithm *SharpNow*. As specified by the translation method, the algorithm starts with an empty knowledge base. During any trial, SharpNow predicts according to rule 7 and updates its knowledge base in light of rule 8. The prediction rule is implemented using the WMC procedure as specified in proposition 3. The following result claims that SharpNow and Winnow make exactly the same predictions on the same series of labeled examples. As an immediate corollary, the mistake bound of SharpNow is the same as the one derived for the original algorithm. This implies, among others, that the size of the knowledge base maintained by the learner is polynomial in the input dimension.

Theorem 4. *SharpNow is an exact simulation of kernel Winnow*

Proof. We consider that Winnow and SharpNow are run with the same parameters η and θ, and the same series of labeled examples $\{(\mathbf{x}_t, y_t)\}$. A sufficient condition for establishing the result is to prove that the following equation holds:

$$\mathbf{w}_t \cdot \boldsymbol{\phi}(\mathbf{x}_t) = \|KB_t \cup \{\neg x_i^j : x_{t,i} \neq j\}\| \tag{9}$$

First of all, consider an assignment r which is not an element of $\mathbf{R}_{n,d}$. Then, for any possible example $\mathbf{x}_t$, r violates at least one clause in the set $\{\neg x_i^j : x_{t,i} \neq j\}$. It follows that r is not a model of $KB_t \cup \{\neg x_i^j : x_{t,i} \neq j\}$. Thus, to prove 9, we only need to consider assignments that are elements of $\mathbf{R}_{n,d}$. So, let r_i be a rule in $\boldsymbol{R}_{n,d}$, $w_{t,i}$ the weight of r_i maintained by $\mathbf{w}_t$, and $w_{t,i}^{\text{KB}}$ the weight of r_i according to KB_t, i.e. $w_{t,i}^{\text{KB}} = \prod\{w : (c, w) \in KB_t, r_i \cap c = \varnothing\}$. We shall prove by induction on the number of trials that $w_{t,i} = w_{t,i}^{\text{KB}}$. Consider the first trial. We have $w_0 = 1$ and since $KB_0 = \varnothing$, we have $w_{0,i}^{\text{KB}} = 1$. Now consider an arbitrary trial and assume by induction hypothesis that $w_{t-1,i} = w_{t-1,i}^{\text{KB}}$ at the beginning of the trial. If no mistake has occurred during the trial, then $w_{t,i} = w_{t,i}^{\text{KB}}$ trivially holds. Thus, suppose that a mistake has occurred. If $\phi_i(\mathbf{x}_t) = 1$, then we know that r_i violates the clause $\bigvee\{x_i^j : x_{t,i} \neq j\}$. So $w_{t,i}^{\text{KB}} = e^{y_t \eta} w_{t-1,i}^{\text{KB}} = e^{y_t \eta} w_{t-1,i} = w_{t,i}$. As a similar strategy applies when $\phi_i(\mathbf{x}_t) = 0$. By applying the fact that $w_{t,i} = w_{t,i}^{\text{KB}}$ to each rule in $\mathbf{R}_{n,d}$, we therefore obtain the desired result.

4 EXPERIMENTS

To provide empirical support for SharpNow, we evaluated it on several learning problems where the target concept is characterized by a small set of rules. A comparison with other standard online mistake-driven algorithms relies on their ability to achieve fast convergence to the optimal hyperplane in the conjunctive feature space $\mathbf{R}_{n,d}$.

The experiments were conducted on a 3.00 GHz Intel Xeon 5160 with 4 GB RAM running Windows XP. All algorithms were written in C++. Notably, the SharpNow algorithm was run using a cache of size 1 GB, and a learning rate $\eta = 1.278$ for which the term $\frac{\eta}{2\sinh\eta}$ in theorem 2 is minimized.

4.1 Static Problems

We begin with experiments on several UCI datasets, with no (known) concept drift, aiming at evaluating the performance of SharpNow relative to the kernel Perceptron and kernel Passive-Aggressive (PA) algorithms. Basically, the PA algorithm [7] is a maximum margin variant of Perceptron that forces the learner to achieve a unit margin on the most recent example while remaining as close as possible to the previous hypothesis. Both algorithms were implemented using the kernel prediction rule 3, and the PA algorithm was run with the update rule (PA-I) and a slack variable C fixed to 100.

data set	Perceptron		PA		SharpNow	
	error	ms	error	ms	error	ms
tic-tac-toe	0.058	0.02	0.073	0.02	**0.009**	0.21
kr-vs-kp	0.042	0.09	0.073	0.16	**0.028**	713
nothing	0.032	0.89	0.020	0.70	0.021	16.5
one pair	0.116	2.01	0.113	2.09	**0**	2.44
two pairs	0.026	0.78	0.042	0.53	**0**	5.17
three of a kind	0.004	0.12	0.007	0.22	**0**	4.41
straight	0.167	0.87	0.158	0.70	**0.107**	119
flush	0.012	0.06	0.011	0.03	0.010	0.03
full house	0.070	0.14	0.069	0.13	**0**	5.84
four of a kind	0.019	0.03	0.029	0.03	**0**	2.34
straight flush	0.010	0.03	0.009	0.01	**0.007**	0.01
royal flush	0.003	0.01	0.003	0.01	0.003	0.01

Table 1. Results for the tic-tac-toe, kr-vs-kp, and poker hand datasets

Experiments were conducted with the "tic-tac-toe" dataset (958 instances, 9 attributes, 27 atoms), the "kr-vs-kp" dataset (3196 instances, 36 attributes, 73 atoms), and the "poker-hand" dataset (1,025,010 instances, 10 attributes, 85 atoms). The last dataset was divided into 10 binary problems. Each problem consists in finding a particular class of poker hand, where all examples of this class are considered as positive, and all other examples are negative. Each card is described using two attributes (*suit* and *rank*); to compare pairs of cards, we introduced four additional binary attributes, *same-suit*, *same-rank*, *before-rank* and *next-rank*. The total number of atoms is thus 165. For all experiments, the accuracy results have been obtained using only one epoch of the training set. For the tic-tac-toe and kr-vs-kp datasets, we employed a standard 10-fold cross-validation. For the poker-hand dataset, we used the training set of 25,010 instances and a subset of 5,000 test instances in the pool of $1,000,000$ test instances. The test set was filled with up to 2,500 positives and the rest as negatives, all examples been chosen at random in the pool.

Results are reported in Table 1. In term of accuracy, the performance of SharpNow is remarkable. Notably, for many problems in the poker hand dataset, we observed that the algorithm converges using less than 5,000 trials while kernel Perceptron and the kernel Passive-Aggressive show some difficulty in approaching the target concept. The running time is measured in milliseconds per trial. As expected, SharpNow is the slowest because it must solve a Weighted Model Counting problem during each prediction. Yet, with a speed of several milliseconds per trial, SharpNow can be used in real-time for medium-size datasets involving a sparse target function.

4.2 Drifting Problems

A natural application for online learning algorithms is to track concepts that are allowed to change over time. In this setting, we analyze the performance of SharpNow relative to the kernel Perceptron and kernel Forgetron algorithms. The Forgetron [8] is a shifting variant of Perceptron that gradually forgets the oldest supports in the set of examples on which mistakes were made; to this end, it uses a decaying rule that diminishes the contribution of old supports and a fixed memory budget B that removes the oldest support.

We conducted experiments with a variant of the Stagger Concepts, a standard benchmark for drifting problems. Each example is a scene involving objects $o_1, \cdots, o_k$ with attributes $color(o_i) \in \{green, blue, red\}$, $shape(o_i) \in \{triangle, circle, square\}$, and $size(o_i) \in \{small, medium, large\}$. In the original problem [18], each scene involves only one object. To analyze the performance of

lirmm-00315916, version 2 - 15 Mar 2009

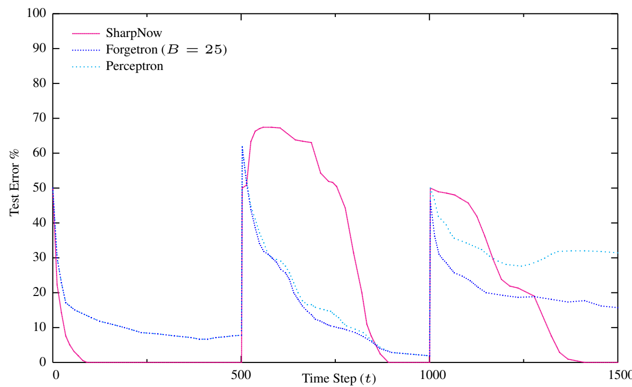

Figure 1. Stagger Concepts with 5 objets

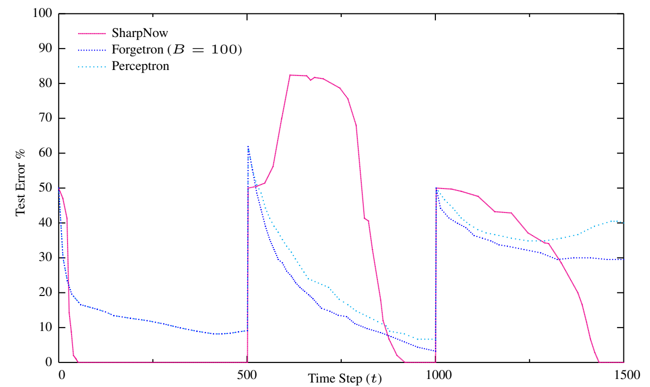

Figure 2. Stagger Concepts with 20 objets

algorithms with more complex scenes including multiple objects, we made 20 series of experiments ranging from $k = 1$ to $k = 20$. Each experiment lasts for 1500 trials, and consists of three rule theories that lasts for 500 time step each: (1) $\{color(o_k) = red \wedge size(o_k) = small\}$, (2) $\{color(o_k) = green, shape(o_k) = circle\}$, and (3) $\{size(o_k) = medium, size(o_k) = large\}$. At each trial, the learner is trained on one example, and tested on 250 positive and 250 negative examples generated randomly according to the current concept.

The accuracy results obtained with $k = 5$ and $k = 20$ are reported in Figures 1 and 2. We observe that SharpNow needs some time to readjust its weights but always converges toward the target concepts; by contrast the kernel Perceptron and kernel Forgetron algorithms show some difficulty in approaching the first and the last concepts. This phenomenon increases with k, revealing that SharpNow is quite robust to irrelevant features. The running time of SharpNow ranges from less than $\frac{1}{1000}$ seconds per trial with $k \leq 5$ to $\frac{1}{4}$ seconds per trial for $k = 20$ (180 atoms and 10^{28} conjunctive features).

5 CONCLUSIONS

We presented SharpNow, an online rule learning algorithm that aims at combining a multiplicative weight update strategy and a weighted model counting method. As an exact simulation of kernel Winnow, the mistake bound of SharpNow is linear in the input dimension. Preliminary experiments on static and drifting problems tend to confirm that SharpNow is particularly efficient for learning small rule theories in presence of many irrelevant conjunctive features.

The problem of extending multiplicative weight-update algorithms to expressive concept classes has been a subject of ongoing research in Machine Learning. Yet, to the best of our knowledge, very few investigations have attempted to handle rule theories. A notable exception is the work by Chawla et. al [6] who suggested to explore Markov Chain Monte Carlo (MCMC) methods for approximating the Kernel Winnow Prediction problem. The main difference with our study is that MCMC methods cannot guarantee an exact simulation of kernel Winnow. Moreover, the running time of their resulting algorithm remains quite slow, taking days to learn a DNF formula over 20 variables. By contrast, the weighted model counting technique takes several milliseconds per trial for similar problems.

Several avenues of research naturally emerge from this study. One of them concerns the analysis of SharpNow under noisy environments using, for example, a discount factor suggested in kernel functions. An orthogonal direction is to extend our method to multi-class environments by simulating multiplicative voting algorithms [4].

REFERENCES

[1] P. Auer and M. K. Warmuth, 'Tracking the best disjunction', *Machine Learning*, **32**(2), 127–150, (1998).
[2] R. J. Bayardo and J. D. Pehoushek, 'Counting models using connected components', in *17th National Conference on Artificial Intelligence*, pp. 157–162, Austin, TX, (2000).
[3] E. Birnbaum and E. L. Lozinskii, 'The good old Davis-Putnam procedure helps counting models', *Journal of Artificial Intelligence Research*, **10**, 457–477, (1999).
[4] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, And Games*, Cambridge University Press, Cambridge, UK, 2006.
[5] M. Chavira and A. Darwiche, 'On probabilistic inference by weighted model counting', *Artificial Intelligence*, (2008). To appear.
[6] D. Chawla, L. Li, and S. Scott, 'On approximating weighted sums with exponentially many terms', *Journal of Computer and System Sciences*, **69**(2), 196–234, (2004).
[7] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, 'Online passive-aggressive algorithms', *Journal of Machine Learning Research*, **7**, 551–585, (2006).
[8] O. Dekel, S. Shalev-Shwartz, and Y. Singer, 'The Forgetron: A kernel-based Perceptron on a fixed budget', in *Advances in Neural Information Processing Systems 18*, Vancouver, Canada, (2005).
[9] C. Domshlak and J. Hoffmann, 'Probabilistic planning via heuristic forward search and weighted model counting', *Journal of Artificial Intelligence Research*, **30**, 565–620, (2007).
[10] J. Fürnkranz, 'Separate-and-conquer rule learning', *Artifical Intelligence Review*, **13**(1), 3–54, (1999).
[11] R. Khardon, D. Roth, and R. A. Servedio, 'Efficiency versus convergence of boolean kernels for on-line learning algorithms', *Journal of Artificial Intelligence Research*, **24**, 341–356, (2005).
[12] R. Khardon and R. A. Servedio, 'Maximum margin algorithms with boolean kernels', *J. of Machine Learning Res.*, **6**, 1405–1429, (2005).
[13] N. Littlestone, 'Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm', *Machine Learning*, **2**(4), 285–318, (1988).
[14] F. Rosenblatt, 'The Perceptron: a probabilistic model for information storage and organization in the brain', *Psych. Rev.*, **65**, 386–408, (1958).
[15] K. Sadohara, 'Learning of boolean functions using support vector machines.', in *12th Int. Conference on Algorithmic Learning Theory*, pp. 106–118, Washington, DC, (2001).
[16] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, 'Combining component caching and clause learning for effective model counting', in *7th Int. Conference on Theory and Applications of Satisfiability Testing*, Vancouver, BC, Canada, (2004).
[17] T. Sang, P. Beame, and H. A. Kautz, 'Performing Bayesian inference by weighted model counting', in *20th National Conference on Artificial Intelligence*, pp. 475–482, Pittsburgh, PA, (2005).
[18] J. C. Schlimmer and R. H. Granger, 'Beyond incremental processing: Tracking concept drift', in *5th National Conference on Artificial Intelligence*, pp. 502–507, Philadelphia, PA, (1986).
[19] M. Thurley, 'sharpSat - counting models with advanced component caching and implicit BCP', in *9th Int. Conference on Theory and Applications of Satisfiability Testing*, pp. 424–429, Seattle, WA, (2006).