Revising Projective DNF in the presence of noise

Robert H. Sloan

Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, USA.

Balázs Szőrényi

Hungarian Academy of Sciences and University of Szeged, Research Group on Artificial Intelligence, H-6720 Szeged, Aradi vérétanúk tere 1., Hungary

Abstract

A noise tolerant revision algorithm is given for the class of projective DNFs, a special class of DNF introduced by Valiant in [13].

1 Introduction

Machine learning in general is concerned with using computers to simulate the learning done by biological entities, especially human beings. Learning theory in particular is a research field that uses the tools of mathematics to study the design and analysis of algorithms to make predictions about the future based on past experiences. Many problems in learning theory have the following broad form (learning from examples): A fixed universe of interest $U$, called a domain, is specified (e.g., $\{0,1\}^n$). A learner is given some sort of access to examples of an unknown 0-1 function $f$ on $U$, for instance a pair $(x_t, f(x_t))$ might be given at each time step $t = 0, 1, 2, \ldots$. The goal of the learner is to determine $f$ (or a “good approximation” to $f$). Of course, if we require any kind of efficiency, (e.g., limited number of example-label pairs, computational efficiency), then there must be some a priori constraints on

---

1 Partially supported by NSF grant CCR-0100336.

Email addresses: sloan@uic.edu (Robert H. Sloan), szorenyi@rgai.hu (Balázs Szőrényi).
In computational learning theory, the usual constraint has been that \( f \) is known to come from some particular class of functions.

This paper focuses on one particular problem in the learning theory of logical formulas: revising projective disjunctive normal forms (projective DNFs) in the presence of noise. This problem is chosen because, while it may sound quite specific, in fact, it ties together three central issues in machine learning. These issues are biological plausibility, revision, and noise. We address each in turn as we explain what our problem is in slightly more detail.

Our discussion of biological plausibility borrows heavily from Valiant [12–14], though these issues have been discussed in a number of other places. One very significant difference between human learning and typical current machine learning (see, e.g., Mitchell [9]) is that most machine learning requires some specific process to do feature extraction when the data has high dimensionality, whereas humans seem to be able to directly process data of very high dimensionality.

Humans learn from quite few examples, in spite of having an internal representation that is potentially extremely large—perhaps in the general vicinity of \( 10^{10} \) neurons with each neuron having up to about \( 10^4 \) parameters (synapses). This means first that the things humans learn must be “small” by some measure, or else learning from a small number of examples would be impossible on dimensionality grounds. Furthermore, human senses are very rich—that is, the input (i.e., training data) that humans receive has very high dimensionality.

Therefore the underlying algorithms must have a special property: attribute efficiency [2, 7]. In attribute-efficient learning, one is learning a (typically Boolean) function that depends on only \( k \) variables in a universe of \( n \) variables, where \( k << n \). An attribute-efficient algorithm’s learning complexity (e.g., amount of training data required) is permitted to be polynomial in \( k \), but is required to be much smaller, typically either logarithmic or polylogarithmic, in \( n \). Littlestone’s Winnow [7] is a justly famous algorithm for the attribute efficient learning of Boolean disjunctions and conjunctions, which also has some other biologically appealing properties.

The purpose of projection learning [13] is to extend the class of Boolean functions for which we can exhibit a learning algorithm that is attribute efficient and otherwise biologically appealing. Projective classes are ones where one can hope to obtain simple, natural, two-level compositions of attribute-efficient algorithms such as Winnow. Perhaps the simplest is projective DNF, which we define formally in Section 2. Valiant gave a learning algorithm for this class [13]; Sloan et al. extended this to a revision algorithm [10]. In this paper, we will extend this to a revision algorithm that tolerates noise. Next we briefly discuss revision, and then noise.
The area of *theory revision* in machine learning is concerned with the revision, or correction of an initial theory (in this paper, a propositional logic formula used for classification) that is assumed to be roughly correct. A typical application area is refining a classifier obtained from a human expert in expert systems work. More broadly, it seems reasonable that in most cases learning starting from something that is close but not exactly right should be much easier than learning from scratch. Indeed, in the cases of human learning of natural language and of particular human faces, it may be that humans are born hardwired with a rough approximation, and in fact revise that, rather than learning from scratch.

The usual assumption of theory revision is that the correct theory can be obtained from the initial one by a small number of syntactic modifications, such as the deletion or the addition of a literal. An efficient revision algorithm is required to be polynomial in the number of literals that need to be modified and polylogarithmic in the total number of literals. Thus, theory revision is roughly an extension of attribute efficient learning—attribute efficient learning can be viewed as the special case of revising the null initial theory. Wrobel surveys the complete theory revision literature [15, 16]; efficient revision in the framework of learning propositional formulas with queries is discussed in detail in [3, 4]. Sloan et al. [10] show how to efficiently revise projective DNFs in the mistake bounded model of learning, where the learner first predicts the classification of a new instance presented by Nature, and then is informed of the correct classification.

However, the algorithm in Sloan et al. may fail badly if any wrong classifications are given to the learner. This is unacceptable, either for biological plausibility, or for application in any practical machine learning system. All real-world training data contains noise. Thus, in this paper, we complete the creation of a biologically plausible, potentially applicable learning of projective DNFs, by showing how to revise them in the presence of noise.

In Section 2 we will give some notations and technical definitions. Then in Section 3 we present our algorithm and its analysis. We conclude briefly in Section 4.

## 2 Preliminaries

To avoid confusion, vectors from $\{0, 1\}^n$ will always be written in bold face (e.g., $w$ or $w_t$), and a component of a vector will be written in normal font (e.g., $w_i$ or $w_{t,i}$).

Projection learning [13] is a tool for learning classes of Boolean functions...
that have a special representation as disjunctions with a particular special structure. Perhaps the most basic class is \( k \)-projective DNF (\( k \)-PDNF). A DNF formula \( \varphi \) is a \( k \)-projective DNF, or \( k \)-PDNF if it is of the form

\[
\varphi = \rho_1 c_1 \lor \cdots \lor \rho_\ell c_\ell,
\]

where every \( \rho_i \), referred to as a \( k \)-conjunction, is a conjunction of exactly \( k \) literals, \( c_i \) is a conjunction and for every \( i \) it holds that

\[
\rho_i \varphi \equiv \rho_i c_i.
\]

Each \( \rho_i \) is called a projection, and \( c_i \) is called the subterm for projection \( \rho_i \).

It is slightly easier to obtain and explain our results in this paper if we do the formal work with the negation of \( k \)-PDNFs, so we extend the notion of projection to its dual. A CNF formula

\[
\varphi = (\rho_1 \lor c_1) \land \cdots \land (\rho_\ell \lor c_\ell)
\]

is called \( k \)-PCNF if each \( \rho_i \) is a disjunction (or clause) of size exactly \( k \); each \( c_i \) is a disjunction and

\[
\rho_i \lor \varphi \equiv \rho_i \lor c_i,
\]

for \( i = 1, \ldots, \ell \). The latter condition means that \( \varphi(\mathbf{x}) = c_i(\mathbf{x}) \) for any vector \( \mathbf{x} \) that falsifies \( \rho_i \). Again, each \( \rho_i \) is called a projection, and \( c_i \) is called the subclause for projection \( \rho_i \). It is easy to see that the negation of a \( k \)-PCNF is a \( k \)-PDNF and vice versa; thus any revision algorithm for one class can be easily transformed into a revision algorithm for the other. Furthermore, as is also true of PDNFs, PCNFs have the special feature that evaluating them on an input can be distributed: first local decisions are made on the subcubes determined by the projections (in the case of PCNFs \( c_i \) makes a decision on region \( \{ \mathbf{x} : \rho_i(\mathbf{x}) = 0 \} \) ), and then a global decision is made based only on the local decisions and on the origin of the input (i.e., which region is it from).

To discuss revising a formula \( \varphi_0 \) to a formula \( \varphi \), we need to define a measure of (syntactic) distance between two formulas. The general definition is, roughly, the minimum number of edit operation to transform the initial formula into the target formula, where the allowed edit operations are fixing an occurrence of a variable to either 0 or 1, or adding a variable. The general definition is discussed in detail in [3, 4].

Here we give in detail the definition particularized to the cases of \( k \)-PDNF and \( k \)-PCNF. The distance between \( t \) and \( t^* \) where \( t \) and \( t^* \) are either both
terms or both clauses is $|t \oplus t^*|$, the number of literals occurring in exactly one of the two.

The revision distance from an initial $k$-PDNF formula

$$\varphi_0 = \rho_1 t_1 \lor \cdots \lor \rho_\ell t_\ell \lor \rho_{\ell+1} t_{\ell+1} \lor \cdots \lor \rho_{\ell+d} t_{\ell+d}$$

to a target $k$-PDNF formula $\varphi = \rho_1 t'_1 \lor \cdots \lor \rho_\ell t'_\ell \lor \rho_{\ell+1} t'_{\ell+1} \lor \cdots \lor \rho_{\ell+d} t'_{\ell+d}$ is

$$\text{dist}(\varphi_0, \varphi) = d + \sum_{i=1}^{\ell} |t_i \oplus t_i^*| + \sum_{i=1}^{a} \max(|t'_i|, 1).$$

The distance is not symmetric, and this reflects the fact that we are interested in the number of edit operations required to transform $\varphi_0$ to $\varphi$. These edit operations are

- the deletion of a term (i.e., both the projection and its subterm),
- the addition of a new projection (with empty subterm), and
- the addition or deletion of a literal to/from a subterm.

For example, the $d$ term in the definition of $\text{dist}$ corresponds to the deletion of the $d$ terms $\rho_{\ell+1} t_{\ell+1}, \ldots, \rho_{\ell+d} t_{\ell+d}$.

We use the standard model of mistake bounded learning of Boolean functions [7]. In this model, the learning of an unknown Boolean function $f(x)$ proceeds in a sequence of trials. In each trial $t$ the learner receives an instance $x_t$, and produces a prediction $\hat{y}_t$. Then the algorithm receives a label $y_t$. If $y_t$ is the correct classification of $x_t$ (i.e., $y_t = f(x_t)$), then we say that $y_t$ is correct, otherwise it is false. Intuitively, a false label is noisy or wrong feedback to the learner.

We define $\text{False}(t)$ to be a 0–1 indicator variable for $y_t$ being a false label. We denote by $\text{NBADLABELS}$ the number of false labels during a particular run, and by $\text{NBADNEG}$ the number of 0 labels that are false—that is, $\text{NBADLABELS} = \sum_t \text{False}(t)$ and $\text{NBADNEG} = \sum_{t:y_t=0} \text{False}(t)$. Of course it is sensible to assume that the number of false labels is relatively small compared to the number of trials, if we expect the learning to be successful. If $\hat{y}_t \neq y_t$ then the learner made a mistake. The mistake bound of the learning algorithm is the maximal number of mistakes, taken over all possible runs, that is, sequences of instances.

The goal of theory revision is that given an initial formula $\varphi_0$, learn to simulate the unknown target formula $\varphi$ making a number of mistakes that is bounded from above by a polynomial of $\text{dist}(\varphi_0, \varphi)$, the number of false labels, and the logarithm of the number of variables in the universe under consideration.
3 Algorithm and analysis

In this section, we present our main results—our algorithm and its analysis and proof of correctness. The overall algorithm has a two-level structure, with many instances of a revision version of Winnow on the lower level feeding their outputs to one instance of a revision version of Winnow on the top level. In Section 3.1 we describe a key subroutine, Algorithm RevWinn, and then in Section 3.2 the overall algorithm.

3.1 Revising disjunctions—Algorithm RevWinn

Algorithm RevWinn (pseudocode displayed as Algorithm 1) revises a monotone disjunction in an attribute-efficient manner. It can be applied to revise an arbitrary disjunction by introducing extra variables for the negated literals, and this in turn can be used to revise arbitrary conjunctions by applying the De Morgan rules. This version of Winnow is similar to one used in [10], although several details are quite different. We now present RevWinn; we will later assume without further discussion that we have versions available for arbitrary disjunctions and for conjunctions.

Algorithm RevWinn revises an initial disjunction \( \varphi \) over a universe that contains a total of \( n \) variables. Like most versions of Winnow, RevWinn also needs parameters for tuning the learning process. In our case, these are \( 0 < \delta < 1 \) and \( \alpha \geq 0 \).

Algorithm RevWinn maintains a weight vector \( w \) of length \( n \), which determines the current hypothesis, and it is updated each time a mistake is made. We use \( w_t \) to denote its value after trial \( t \). Accordingly \( w_0 \) denotes the initial weight vector. To avoid confusion, in general for components we will use indices \( i \) and \( j \), and for trials we will always use \( t \)—accordingly \( w_{t,i} \) denotes the value of the \( i \)-th component of vector \( w \) in trial \( t \).

The algorithm consists of three main parts: initialization of \( w \) (which initializes the hypothesis), prediction (the hypothesis part), and the update part. Formally, we break out each as a subroutine to make later discussion easier.

Let us now describe these three parts of RevWinn. First the weight vector \( w \) is initialized according to the initial hypothesis (caluse) \( \varphi_0 \) by setting it to \( \text{INIT}(n, \varphi_0, \delta) \). The function \( \text{INIT}(n, \varphi_0, \delta) \) returns an \( n \)-vector \( w \) with

\[
 w_i = \begin{cases} 
 1 & \text{if variable } x_i \text{ appears in } \varphi_0 \\
 \delta & \text{otherwise}
\end{cases}
\]
Algorithm 1 Algorithm RevWinn\((n, \varphi_0, \delta, \alpha)\). (For “standard version”, set \(\delta = 1/2n\) and \(\alpha = 1\).)

\[
\text{\textbf{w := INIT}(n, \varphi_0, \delta) \{initialize the weight vector\}}
\]

\textbf{for} trial \(t = 1, 2, \ldots\) \textbf{do}

\hspace{1em} On input \(x_t\) evaluate hypothesis

\[
\hat{y}_t := h(x_t, w) = \begin{cases} 0 & \text{if } w \cdot x_t < 1/2 \\ 1 & \text{otherwise} \end{cases}
\]

\hspace{1em} Predict \(\hat{y}_t\)

\hspace{1em} if the current label, \(y_t\) differs from \(\hat{y}_t\) \textbf{then}

\hspace{1em} \hspace{1em} \textbf{w := UPDATE}(y_t, x_t, \alpha, w)

\hspace{1em} \textbf{end if}

\hspace{1em} \textbf{end for}

\textbf{for} \(i = 1, \ldots, n\).

The hypothesis function on input instance \((n\text{-dimensional vector}) x\) is

\[
h(x, w) = \begin{cases} 0 & \text{if } w \cdot x < 1/2 \\ 1 & \text{otherwise} \end{cases}
\]

where \(w \cdot x = \sum_{i=1}^{n} w_i x_i\) is the dot product of \(w\) and \(x\). The hypothesis is used to make predictions; in trial \(t\) we predict that the label of \(x_t\) is \(\hat{y}_t = h(x_t, w_{t-1})\).

Finally the function \textbf{UPDATE}(\(y, x, \alpha, w\)), returns a vector \(w'\), a modification of the weight vector \(w\):

\[
w'_i = w_i \left(1 + \frac{\alpha}{4} \frac{x_i}{x \cdot w} (y - \hat{y})\right),
\]

where \(\hat{y}\) is the output of the hypothesis function on \(x\) (i.e., \(\hat{y} = h(x, w)\)). This function does nothing and need not even be called if there is not mistake; that is, if \(\hat{y} = y\).

The two parameters \(\delta\) and \(\alpha\) define a particular version of RevWinn. For example, to obtain the version of RevWinn that can be applied simply to revise a disjunction \(\varphi_0\), put \(\delta = 1/2n\) and \(\alpha = 1\). We call this the \textit{standard} version of RevWinn. We give the mistake bound for this standard version in Corollary 3 below.

Let RevWinnC denote the variant of RevWinn that revises conjunctions—as explained at the beginning of this section—and let INITC, hC, and UPDATEC denote its main functions.
3.2 Algorithm RevPCNF

For revising PCNFs, we use a two-level algorithm (see Algorithm 2) based on Valiant’s construction ([13] and also in [10]). On the lower level, for each $k$-projection $\rho$, we run an instance of RevWinn to find the appropriate clause for that projection. We call this the $\rho$ instance of RevWinn, and we denote its weight vector by $w^\rho$. The input and the label for each of these instances are $x_t$ and $y_t$. An update is applied to the $\rho$ instance of RevWinn only when $\rho(x_t) = 0$ (and additionally the top-level algorithm’s prediction of the label was wrong and agreed with the prediction of the $\rho$-instance of RevWinn), because in this case, by Equation (1) if $\rho$ appears in the target formula with subclause $c$, then the output of the target formula agrees with $c$—and this is the key to the whole algorithm. Intuitively, we hope that for each clause of the form $\rho \lor c$ in the target formula, where $\rho$ is a $k$-projection, that the hypothesis of the $\rho$ instance of RevWinn will converge to $c$. The prediction of the $\rho$ instance of RevWinn is denoted by $\hat{y}_t^\rho$ and $\hat{y}_t^\rho = h(x_t, w_{t-1}^\rho)$.

On the top level, an instance of RevWinnC is run. In the rest of this paper, $w$ is used to denote the weight vector of this top-level algorithm (and, if we want to emphasize the trial, $w_t$ denotes its value after trial $t$). We also introduce a new Boolean variable $v_\rho$ for each $k$-projection $\rho$. We denote by $v$ the vector formed by all these variables, and by $v_t$ its value in trial $t$. The vector $v_t$ is defined by

$$v_{t,\rho} := \rho(x_t) \lor h(x_t, w_{t-1}^\rho).$$

Also define $\hat{y}_t = hC(v_t, w_{t-1})$. The top-level RevWinnC algorithm learns a conjunction over variables $v_\rho$, which would ideally consist of exactly those variables that are indexed by projections appearing in the target formula.

If one simply uses two levels of Littlestone’s Winnow for PCNF revision [10], then noise may be a serious problem: even one or two single false labels may cause undesired weight updates in so many projections’ corresponding weight vectors that very many mistakes are forced in the rest of the run in order to gain back (resp. lose) all that lost (resp. gained) weight. So we need to limit somehow the amount of weight update. But this also has its hazards: if the change of weight during an update is too small, than again we might be forced to make too many mistakes to finally reach a sufficiently big weight.

Our solution to the problem, which is applied in algorithm RevPCNF (see algorithm 2) is the following. Let the initial formula be $\varphi_0 = (\rho_1 \lor c_1) \land \cdots \land (\rho_s \lor c_s)$, and in trial $t$ let

$$m_t = |\{i : \rho_i(x_t) = 0 \text{ and } \hat{y}_t^{\rho_i} \neq y_t, 1 \leq i \leq s\}|$$
Algorithm 2 The procedure RevPCNF(ϕ₀)

1: {ϕ₀ = (ϕ₁ ∨ c₁) ∧ ⋯ ∧ (ϕₙ ∨ cₙ) is the k-PCNF to be revised}
2: w := INITC (2⁵(n \choose k), vₚ₁, ⋯, vₚₙ; \frac{1}{2k+1(\frac{n}{k})})
3: for each k-projection ρ do
4:   if ρ = ρᵢ for an i ∈ {1, ⋯, s} then
5:     wᵣ := INIT(n, cᵢ, \frac{1}{2n(s)})
6:   else
7:     wᵣ := INIT(n, empty-disjunction, \frac{1}{2n})
8:   end if
9: end for
10: for each trial t with input xₜ do
11:   evaluate v by vₚ := ρ(xₜ) ∨ h(xₜ; wᵣ)
12:   In trial t predict ŷₜ = hC(vₚ, w)
13: if yₜ ≠ ŷₜ then {Update only if wrong top-level prediction}
14:   w := UPDATEC(yₜ, v, 1, w)
15:   m := |{i : ρᵢ(xₜ) = 0 and h(xₜ; wᵣᵢ) ≠ yₜ, 1 ≤ i ≤ s}| 
16:   for each ρ with ρ(xₜ) = 0 and vᵣ ≠ yₜ do
17:     if ρ = ρᵢ for an i ∈ {1, ⋯, s} then
18:       wᵣ := UPDATE(yₜ, xₜ, m; wᵣ)
19:     else
20:       wᵣ := UPDATE(yₜ, xₜ, 1; wᵣ)
21:     end if
22:   end for
23: end if
24: end for

be the number of projections of the initial formula whose weight is updated in that trial, assuming that there was a mistake, (i.e., mₜ is the value of m \text{ in Line 15 of Algorithm 2 in trial } t). For the s projections corresponding to the initial formula, we use an update rule with an update value of

\[ α = 1/mₜ. \]  \tag{2} \]

Thus the change in the sum of the weights of a projection ρ ∈ {ρ₁, ⋯, ρₙ} is

\[ \sum_{i=1}^{n}(w^ρ_{t,i} - w^ρ_{t-1,i}) = \frac{1}{4mₜ} \sum_{i=1}^{n} xₜi wₜ-1,i (\hat{y}_t - y_t) = \frac{1}{4mₜ} (y_t - \hat{y}_t). \]  \tag{3} \]

Now summing the change in the weights over all the projections of the initial formula gives exactly ±1/4. Note that if mₜ = 0 then by definition, we do not update the weight vectors of these projections; thus we don’t have to worry that Equation (2) might be undefined.

For all the other projections that do not appear in the initial formula, a stan-
standard RevWinn version is used. A standard RevWINNC algorithm is also used for the top-level algorithm.

**Theorem 1** Suppose that the initial formula $\varphi_0$ and the target formula $\varphi$ are the $k$-PCNF formulas

$$
\varphi_0 = (\rho_1 \lor c_1) \land \cdots \land (\rho_\ell \lor c_\ell) \land (\rho_{\ell+1} \lor c_{\ell+1}) \land \cdots \land (\rho_{\ell+d} \lor c_{\ell+d})
$$

$$
\varphi = (\rho_1 \lor c^*_1) \land \cdots \land (\rho_\ell \lor c^*_\ell) \land (\rho'_1 \lor c'_1) \land \cdots \land (\rho'_a \lor c'_a),
$$

and let $e = \text{dist}(\varphi_0, \varphi)$. Then algorithm RevPCNF makes $O \left((\text{NBADLABELS}(e + \text{NBADLABELS})k \log n)^2\right)$ mistakes.

The proof is based on the following technical lemma. (We give the lemma and its proof, then state some corollaries of the lemma, and then return to the proof of Theorem 1.)

**Lemma 2** The number of trials $t$ for which at least one of $w^{\rho_1}_t, \ldots, w^{\rho_\ell}_t$ is changed by an update is $O(e \text{NBADLABELS}(e + \text{NBADLABELS})k \log n)$.

**Proof:** There are three cases for a trial $t$: no change is made to any of the weights $w^{\rho_1}, \ldots, w^{\rho_\ell}$, one or more of those $w^{\rho_1}$ is decreased (a demotion step), or one or more of those $w^{\rho_1}$ is increased (a promotion step). (Since the same value of $y_t$ is passed to all the lower-level RevWinn instances, it cannot be that some weights are increased while other weights are decreased.)

Now consider any run of the algorithm of length $T$. Let $M^-$ denote the total number of demotions and $M^+$ the total number of promotions. We prove this lemma by using a potential function argument to obtain a bound on the sum $(M^- + M^+)$. Let $P = \{\rho_1, \ldots, \rho_\ell\}$. Let $P_t \subseteq P$ be the set of projections $\rho$ that appear in both the initial and the target formula such that the $\rho$ instance of RevWinn updates its weight in trial $t$, that is

$$
P_t = \begin{cases} 
\emptyset & \text{if } \hat{y}_t = y_t \\
\{\rho_i : \rho_i(x_i) = 0 \text{ and } \hat{y}_t^{\rho_i} \neq y_t, 1 \leq i \leq \ell\} & \text{if } \hat{y}_t \neq y_t
\end{cases}
$$

Also let $p_t = |P_t|$ (thus $m_t \geq p_t$). For $i = 1, \ldots, \ell$ let $I^{\rho_i}$ be the set of indices $j$ of variables that appear in both $c_i$ and $c^*_{\ell}$, and such that there is at least one trial $t$ with $w^{\rho_i}_{t,j} < 1/2$. For $i = 1, \ldots, \ell$ let $J^{\rho_i}$ be the set of indices of variables that appear in $c^*_{\ell}$ but not in $c_i$. Let us also introduce the notation $I^{\rho} \cup J^{\rho} = \{1, \ldots, n\} \setminus (I^{\rho} \cup J^{\rho})$ for $\rho \in P$. When no confusion arises, we will
sometimes refer to a variable \( x_i \) belonging to one of these sets when we really should say that the variable’s index belongs to the set.

We use a potential function \( \Phi(w^{\rho_1}, \ldots, w^{\rho_\ell}) = \sum_{i=1}^{\ell} \sum_{i=1}^{n} \Phi_{\rho_i}(w^{\rho_i}) \), where

\[
\Phi_{\rho_i}(w) = \begin{cases} 
  w_i - 1 + \gamma \ln \frac{1}{w_i} & \text{if } i \in I^\rho \cup J^\rho \\
  \text{otherwise} 
\end{cases}
\]

where \( \gamma > 1 \) is an appropriate constant which satisfies \( \gamma \frac{\ln(1+x)}{x} \geq 1 \) for any \( x \in (0, 0.5] \)—note that such \( \gamma \) exists.\(^2\) It can be verified that \( \Phi_{\rho_i}(w) \geq 0 \) for any \( w \in (0, 1]^n \) after noticing that \( \Phi_{\rho_i}(w) \) depends only on \( w_i \) and is monotone in \( w_i \) for all \( i \).\(^3\) Note that the potential function depends only on the weights corresponding to the projections that appear both in the initial and the target formula. Also note that it is defined with respect to a specific run of the algorithm—that is, not only its values, but even the function itself may vary in different runs.

Let us introduce the short notation \(^4\) \( \Phi_t = \Phi(w^{\rho_1}_t, \ldots, w^{\rho_\ell}_t) \), and let \( \Delta \Phi_t = \Phi_t - \Phi_{t-1} \) denote the negative of the change of the potential function during trial \( t \). We will derive both upper and lower bounds on \( \sum_{t=1}^{T} \Delta \Phi_t \) that will allow us to relate the sum \((M^- + M^+)\) to \( e, n \), and \( \text{NBadLabels} \).

First we derive an upper bound:

\[
\sum_{t=1}^{T} \Delta \Phi_t = \Phi_0 - \Phi_T \\
\leq \Phi_0 - \sum_{\rho \in P} \sum_{i \in I^\rho \cup J^\rho} \Phi_{\rho}(w_{T,i}) \\
= \sum_{\rho \in P} \sum_{i \in I^\rho} \Phi_{\rho}(w^\rho_0) + \sum_{\rho \in P} \sum_{j \in J^\rho} \Phi_{\rho}(w^\rho_0) + \sum_{\rho \in P} \sum_{i \in I^\rho \cup J^\rho} (w^\rho_{0,i} - w^\rho_{T,i}).
\]

For \( i \in I^\rho \) we initialized \( w^\rho_{0,i} = 1 \) so \( \Phi_{\rho}(w^\rho_0) = 0 \). Also, \( \sum_{\rho \in P} |J^\rho| \leq e \), and \( \Phi_{\rho}(w^\rho_0) = \gamma \ln(2n(\ell + d)) - \frac{2n(\ell + d) - 1}{2n(\ell + d)} < \gamma (k + 1) \ln(2n) \) for \( j \in J^\rho \), so the sum

\(^2\) For example \( \gamma = 1/\ln 2 \) suffices, noting that the function \( \ln(1+x) \) is strictly concave, \( x \ln 2 \) is linear, both are nonnegative on \((0, 1] \) and that they evaluate the same in 0 and in 1.

\(^3\) In particular, \( \Phi_{\rho_i}(w) \) is it is monotonically decreasing for those indices \( i \in I^\rho \cup J^\rho \), in which case \( \Phi_{\rho_i}(w) = 0 \) when \( w_i = 1 \). It is monotonically increasing for all other indices \( i \).

\(^4\) Note that using \( \Phi \) with one single lower index \( t \) denotes the value of the potential function in trial \( t \), meanwhile using it with both an upper index \( \rho \) and a lower index \( i \) denotes the sub-sum of the potential function that corresponds to projection \( \rho \) and component \( i \).
of the first two terms is at most $e\gamma(k + 1) \ln(2n)$. Now we need to bound the third term. The elements of the third term can be divided into three groups according to their indices $\rho_j$ and $i$:

- $i \notin c_j \cup c^*_j$. For these indices we have $w_{0,i}^\rho - w_{T,i}^\rho \leq w_{0,i}^\rho = \frac{1}{2n(t+\delta)}$, so altogether they can contribute at most 1/2 to the sum.
- $i \in c_j \setminus c^*_j$. There are at most $e$ of this kind, each at most 1, thus they contribute at most $e$ to the sum.
- $i \in c_j \cap c^*_j$. By definition, any decrease of the weights corresponding to these index pairs is due to a false negative label, thus their contribution to the sum is at most $\frac{N_{\text{BadNeg}}}{4}$ (by the argument before Theorem 1).

Thus we get

$$\sum_{t=1}^{T} \Delta \Phi_t \leq e\gamma(k + 1) \ln(2n) + e + \frac{1}{2} + \frac{1}{4} N_{\text{BadLabels}}.$$  

(4)

To get a lower bound on the sum, we begin by deriving a lower bound on the change in potential in one trial. Now, since $w_{t,i}^\rho = w_{t-1,i}^\rho$ for each $\rho \in P \setminus P_t$ and all $i$, and applying (3),

$$\Delta \Phi_t = \sum_{\rho \in P} \left( \sum_{i \in I_{\rho} \cup J_{\rho}} (w_{t-1,i}^\rho - w_{t,i}^\rho + \gamma \ln \frac{w_{t,i}^\rho}{w_{t-1,i}^\rho}) + \sum_{i \in I_{\rho} \cup J_{\rho}} (w_{t-1,i}^\rho - w_{t,i}^\rho) \right)$$

$$= \sum_{\rho \in P} \sum_{i=1}^{n} (w_{t-1,i}^\rho - w_{t,i}^\rho) + \sum_{\rho \in P_t} \sum_{i \in I_{\rho} \cup J_{\rho}} \gamma \ln \frac{w_{t,i}^\rho}{w_{t-1,i}^\rho}$$

$$= \frac{P_t(y_t - \hat{y}_t)}{4m_t} + \sum_{\rho \in P_t} \sum_{i \in I_{\rho} \cup J_{\rho}} \gamma \ln \frac{w_{t,i}^\rho}{w_{t-1,i}^\rho}. \quad (5)$$

Also note that for any $\rho \in P_t$

$$\ln \frac{w_{t,i}^\rho}{w_{t-1,i}^\rho} = x_{t,i} \ln \left( 1 \pm \frac{1}{4m_t} \frac{1}{x_t \cdot \mathbf{w}_{t-1}^\rho} \right).$$

Now we analyze the change of the potential function during trial $t$ depending on whether $t$ was a demotion step or a promotion step—obviously, when no update is done in trial $t$ (i.e., $y_t = \hat{y}_t$), then $\Delta \Phi_t = 0$.

In a demotion step, $\hat{y}_t = 1$ and $y_t = 0$. If $\rho \in P$ updates its weights in this trial (i.e., $\rho \in P_t$) and $|{(I^\rho \cup J^\rho) \cap \mathbf{x}_t}| > 0$ then, by the definition of $I^\rho$ and $J^\rho$ and recalling that the target formula is PCNF, the label $y_t$ must be false, thus $|{(I^\rho \cup J^\rho) \cap \mathbf{x}_t}| = |{(I^\rho \cup J^\rho) \cap \mathbf{x}_t}| \text{FALSE}(t)$. Also, $\mathbf{x}_t \cdot \mathbf{w}_{t-1}^\rho \geq \frac{1}{2}$ for each $\rho \in P_t$, thus $1 - \frac{1}{4m_t} \frac{1}{x_t \cdot \mathbf{w}_{t-1}^\rho} \geq 1/2$. So, using (5),
\[ \Delta \Phi_t \geq \frac{p_t}{4m_t} + \sum_{\rho \in P_t} |(I^\rho \cup J^\rho) \cap x_t| \gamma \ln \left( \frac{1}{2} \right) \]
\[ \geq \frac{1}{4d} - \text{False}(t) \gamma (e + \sum_{\rho \in P} |I^\rho|) \ln 2 . \] (6)

In a promotion step, \( \hat{y}_t = 0 \) and \( y_t = 1 \), and \( x_t \cdot w_{t-1}^\rho < 1/2 \) for every \( \rho \in P_t \), thus \( 4m_t x_t \cdot w_{t-1}^\rho < 2m_t \). Also note that \(|(I^\rho \cup J^\rho) \cap x_t| \geq 1\) for every such \( \rho \), unless the label was false, thus

\[
\sum_{\rho \in P_t} \sum_{i \in I^\rho \cup J^\rho} \ln \left( 1 + \frac{1}{4m_t} \frac{x_{t,i}}{x_t \cdot w_{t-1}^\rho} \right) \geq \sum_{\rho \in P_t} |(I^\rho \cup J^\rho) \cap x_t| \ln \left( 1 + \frac{1}{2m_t} \right) \]
\[ \geq \sum_{\rho \in P_t} (1 - \text{False}(t)) \ln \left( 1 + \frac{1}{2m_t} \right) \]
\[ = p_t (1 - \text{False}(t)) \ln \left( 1 + \frac{1}{2m_t} \right) \]

consequently

\[ \Delta \Phi_t = -\frac{p_t}{4m_t} + \gamma \sum_{\rho \in P_t} \sum_{i \in I^\rho \cup J^\rho} \ln \left( 1 + \frac{1}{4m_t} \frac{x_{t,i}}{x_t \cdot w_{t-1}^\rho} \right) \]
\[ \geq -\frac{p_t}{4m_t} + \gamma p_t (1 - \text{False}(t)) \ln \left( 1 + \frac{1}{2m_t} \right) , \]

which, by the choice of \( \gamma \), gives

\[ \Delta \Phi_t \geq \frac{p_t}{4m_t} - \text{False}(t) \gamma \ln \left( 1 + \frac{1}{2m_t} \right) \geq \frac{1}{4d} - \text{False}(t) \gamma . \] (7)

Observe furthermore that for any \( \rho \in P \) and \( i \in I^\rho \) all decrease of the weight of variable \( x_i \) in \( \rho \)'s REVWNN instance is due to a false label. The sum of these weight losses is at least \( \frac{1}{2} \sum_{\rho \in P} |I^\rho| \) by the definition of \( I^\rho \). Noting again that any false label \( y_t \) changes \( \sum_{\rho \in P} \sum_{i=1}^n w_{t,i}^\rho \), with at most 1/4, we get

\[ \frac{1}{2} \sum_{\rho \in P} |I^\rho| \leq \text{NBADNeg}/4. \]

Then (6) and (7) give us

\[ \sum_{t=1}^T \Delta \Phi_t \geq \sum_{\text{demotion step } t} \left( \frac{1}{4d} - \text{False}(t) \gamma (e + \frac{1}{2} \text{NBADNeg}) \ln 2 \right) \]
\[ + \sum_{\text{promotion step } t} \left( \frac{1}{4d} - \text{False}(t) \gamma \right) \]
\[ \geq M^-/4d + M^+/4d - \gamma \text{NBADLabels}(\text{NBADLabels} + e) \ln 2 . \]
Combining this with (4) gives the desired mistake bound noting that $d \leq e$.  

Using a similar proof technique for the algorithm RevWinn applying the appropriate simplifications and other obvious modifications one derives the following result.

**Corollary 3** If at most $F$ false labels are received, then the standard RevWinn algorithm makes at most $O(F(s + F) \log n)$ mistakes in revising the empty disjunction to disjunction $\varphi$, where $s$ is the size of $\varphi$.

As a RevWinnC instance is actually a RevWinn instance with some extra negations (which are applied according to the De Morgan rules), the previous mistake bound also holds for RevWinnC.

**Corollary 4** Let the initial conjunction be $\varphi_0$ and the target conjunction be $\varphi$. If at most $F$ false labels are received, then algorithm RevWinnC makes at most $O(F(\text{dist}(\varphi_0, \varphi) + F) \log n)$ mistakes.

**Proof of Theorem 1:** The top-level RevWinnC of RevPDNF is revising a conjunction over the variables denoted $v_\rho$ in the overall algorithm. Corollary 4 tells us how to compute the mistake bound of that RevWinnC as a function of the revision distance and the number of false labels it receives in a trial.

The revision distance for RevWinnC is $d + a \leq e$. There are two sources of false labels from the point of view of RevWinnC. One is any trial where $y_t$ is a false label. The other are trials where one of the $v_\rho$ that is in the target takes on the wrong value because $\rho(x) = 0$ and the (lower-level) $\rho$ instance of RevWinn had the wrong output for $x$.

Let us restrict our attention to trials when the overall algorithm make a mistake (noting that in the rest of the cases no update is done, thus the bound we derive for this case will hold for the whole run as well).

By the projection property, for $1 \leq i \leq \ell$, when $\rho_i(x) = 0$ we know that $c_i^*(x)$ must have the same value as the entire target $k$-PCNF. The same holds for $\rho_j'$ and $c_j'$ for $1 \leq j \leq a$. Furthermore, the overall algorithm RevPCNF causes updates to a $\rho$ instance of RevWinn only when the overall algorithm makes a mistake (i.e., $y_t \neq \hat{y}_t$) and $\rho(x_i) = 0$. Therefore, whenever Update is called for either the $\rho_i$, for $1 \leq i \leq \ell$, or the $\rho_j'$, for $1 \leq j \leq a$, instance of RevWinn, then the $y_t$ passed down to that instance is the appropriate $y_t$ to which that instance’s output should be compared.

Excluding the trials where $y_t$ is a false label, the top-level RevWinnC receives the correct value for $v_{\rho, i} = \rho_i(x_i) \lor c_i^*(x_i)$ for $i = 1, \ldots, \ell$ anytime that the $\rho_i$ instance of RevWinn does not update its weight vector. This is so because in
these cases $v_{t,r_i}$ is always correct if $\rho_i(x_t) = 1$, and because of the projection property if $\rho_i(x_t) = 0$. Lemma 2 gives a bound on the number of times that these RevWinn instances update their weights, noting that we can use the value of NBADLABELS for the overall algorithm as an upper bound on the number of false labels that any $\rho_i$ instance of RevWinn will receive.

Corollary 3 similarly gives a bound on the number of wrong outputs that the $\rho'_j$ instance of RevWinn can pass up to RevWinnC as wrong $v_{\rho'_j}$ for $1 \leq j \leq a$.

Thus the overall mistake bound for the algorithm is the bound on RevWinnC from Corollary 4, where we use $e$ as the revision distance and the sum of NBADLABELS plus the sum of the two mistake bounds from Lemma 2 and Corollary 3 as an upper bound on $F$, noting that $e$ is an upper bound for $s$ in the mistake bound of Corollary 3.

Straightforward algebra gives the claimed bound. □

4 Concluding remarks and open problems

We have investigated, and have given positive learning results for, the revision of the special DNF class called projective DNFs (and for the dual projective CNFs) in the framework of mistake bounded learning. The theory revision model we use in this framework is the natural translation of the model introduced by Sloan and Turán in the query learning framework [11]. The most important contribution of this paper is handling noise in this revision model.

We should mention that our potential function argument (and also the potential function itself) was motivated by the proof techniques applied in another problem area called tracking (see, e.g., [1, 5, 6, 8]), and especially by the work of Aurer and Warmuth [1]. Tracking concerns a dynamic version of learning, where the target concept is allowed to change, or shift, from time to time. This suggests that between two shifts, the learner cannot afford to make many mistakes—which sounds like a requirement for theory revision. This naturally gives rise to the question, “What is the relationship between tracking and theory revision?” One might think that an algorithm for tracking would be also good for revision—but this is not the case. A tracking algorithm may make linearly many false prediction in the size of the “biggest” concept during tracking (furthermore our previous argument in section 3.2 also suggests that those versions of Winnow which are used for example in [1] can be forced to make this many mistakes). In fact it is essential to allow this linear dependence, because tracking, like learning, starts from scratch, in contrast to revision. In the opposite direction, an efficient revision algorithm might fail
badly at the tracking task: in the model that we use, a successful revision does not necessarily mean that the output of the algorithm is syntactically close to the target—although an incremental application of the revision algorithm, which would be necessary for tracking, would require this.

So the two problems, tracking and revision, do not imply each other—but, as well they are apparently related, can they at least be handled simultaneously? The fact that the basic tool of our algorithm—Winnow—is also the basic tool commonly used in tracking gives some hope that perhaps they can be handled simultaneously. Moreover, an appropriate version of Winnow might be just the right tool to achieve this. We can pose an even more ambitious goal based on the above facts: to modify algorithm RevPCNF so that the result is an efficient revision algorithm for projective CNFs (and DNFs) that can also be used for tracking for the same class. This would be interesting from another point of view as well: the previous tracking results—as far as we know—all consider linear functions; thus a tracking algorithm for a CNF class would be quite novel.

References


