LEARNING DECISION LISTS AND PARITY FUNCTIONS

GLEB POSOBIN

Abstract

In this paper we examine two learning problems: learning decision lists and learning parity functions. We describe several algorithms for both of these problems and using them demonstrate several basic concepts in computational learning theory and connections between these concepts. We also look at learning parities in memory-bound setting and present a family of algorithms for learning 1-parities.

Keywords: computational learning theory, learning parities, decision lists, online learning, PAC learning.

1 INTRODUCTION

Computational learning theory examines various properties of machine learning algorithms and machine learning problems. In this paper we survey various results, both old and recent, in computational learning theory, and construct a family of algorithms for learning in memory-bound setting.

We begin with two simple algorithms for learning decision lists and prove an important fact that algorithms that find some hypothesis function that agrees with all the seen data (so-called Occam algorithms) imply PAC learnability for certain classes of functions. Next we describe how to create a faster algorithm for learning decision lists of length k from [KS06].

After that we turn to look at parity functions and describe three algorithms for learning them. The first one is a simple Gaussian elimination. It uses $\mathcal{O}(n)$ samples and runs in time $\mathcal{O}(n^3)$. The second one is used for learning k-parities with small k and it is just a Gaussian elimination after removal of several randomly choosen variables from consideration. With a rather simple analysis we show that it uses $\mathcal{O}(n^{1-1/k} \log n)$ examples and its runtime is $\mathcal{O}(n^4)$. The third algorithm runs in time $\mathcal{O}(n^{k/2})$ and uses $\mathcal{O}(k \log n)$ examples.

Finally, we take a glance at the memory consumption of learning algorithms and look at the problem of learning 1-parities. Here we present a family of algorithms for learning 1-parities (and, by means of a simple construction, k-parities) in memory-bound setting.

2 BASIC DEFINITIONS

Let X be a set called the *instance space*. In our case, X will always be $\{0,1\}^n$.

We call $c \subset X$ a *concept over* X. Equivalently, c could be thought of as a mapping $c : X \to \{0,1\}$.

A *concept space* $\mathcal{C}$ over X is a set of concepts over X.

In our model, a learning algorithm will have access to positive and negative examples of an unknown *target concept* c, chosen from a known concept class $\mathcal{C}$.

Let $\mathcal{D}$ be any probability distribution over the instance space X.

If h is some concept over X, then $\mathcal{D}$ provides a natural measure of error between h and target concept c:

$$error(h) = \mathbf{Pr}_{x \in \mathcal{D}}[c(x) \neq h(x)]$$

Let $EX(c, \mathcal{D})$ be a procedure that runs in unit time, and on each call returns a labelled example $\langle x, c(x)\rangle$, where x is drawn randomly according to $\mathcal{D}$. Learning algorithm will have access to this oracle when learning the target concept $c \in \mathcal{C}$.

2.1 *PAC model*

An important concept in learning theory is that of a probably approximately correct (PAC) learning, introduced by Valiant [Val84]:

Definition 2.1. Let $\mathcal{C}$ be a concept class over $X = \{0,1\}^n$. We say that $\mathcal{C}$ is *PAC learnable*, if there exists an algorithm L with the following property: for every concept $c \in \mathcal{C}$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$, if L is given access to $EX(c, \mathcal{D})$ and inputs ϵ and δ, then with probability at least $1 - \delta$, L outputs a hypothesis concept $h \in \mathcal{C}$, satisfying $error(h) < \epsilon$. This probability is taken over the random examples drawn by calls to $EX(c, \mathcal{D})$, and any internal randomization of L.

If L runs in time polynomial in n, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, we say that $\mathcal{C}$ is efficiently PAC learnable.

We say that PAC learning algorithm L is *attribute-efficient* if number of used examples and its running time are bounded by a polynomial in $size(c)$, where $size(c)$ is length of c under some reasonable encoding.

2.2 *Mistake bound model*

Informally, mistake bound model can be described as follows: on each iteration the learning algorithm is presented with an unlabelled example x and has to predict the value of target function $c(x)$, and after that receives the real value. We would like this algorithm to do only a small (polynomial) number of mistakes. More formally:

Definition 2.2. We say that algorithm $\mathcal{A}$ *learns* $\mathcal{C}$ *in mistake bound model* if for any concept $c \in \mathcal{C}$, and for any ordering of examples consistent with c, the total number of mistakes ever made by $\mathcal{A}$ is bounded by polynomial in n and $size(c)$.

Again, we say that algorithm $\mathcal{A}$ is *attribute-efficient* in mistake bound model if number of its mistakes is bounded by polynomial in $size(c)$.

Note that mistake bound model is probability-agnostic, which sometimes makes it more attractive than PAC model. Also, the notion of mistake bound learnability is a stronger one than that of PAC learnability, as we will show in section 3.4.

2.3 *Functions that we examine*

In this paper we will look at two concept classes: decision lists and parity functions.

Decision list L of length k over Boolean variables $x_1, \ldots, x_n$ is a list and a bit $(\ell_1, b_1), (\ell_2, b_2), \ldots, (\ell_k, b_k), b_{k+1}$, where each ℓ_i is a literal and each b_i is either 1 or -1. Given any $x \in \{0,1\}^n$, the value of $L(x)$ is the value returned by the following program:

$$\textbf{if } \ell_1 \textbf{ then } b_1 \textbf{ else if } \ell_2 \textbf{ then } b_2 \ldots \textbf{else if } \ell_k \textbf{ then } b_k \textbf{ else } b_{k+1}.$$

Parity function χ_S of length k is defined by a set $S \subset \{x_1, \ldots, x_n\}$ such that $|S| = k$. The parity function takes value 1 (-1) on inputs which set an even (odd) number of variables in S to 1.

Finally, we need two more definitions, that we will use for learning decision lists.

If for function $f : \{0,1\}^n \to \{1,-1\}$ there exist such $\theta \in \mathbb{R}$ and $w \in \mathbb{R}^n$ that $f(x) = sgn(x \cdot w - \theta)$ for all $x \in \{0,1\}^n$, then we call f a *linear threshold function*.

If for function $f : \{0,1\}^n \to \{1,-1\}$ there exists such a polynomial p in n variables with integer coefficients that $f(x) = sgn(p(x))$ for all $x \in \{0,1\}^n$, then we say that p is a *polynomial threshold function* for f.

3 LEARNING DECISION LISTS

3.1 *Simple algorithm for learning decision lists*

We begin with a simple algorithm for learning decision lists, which will just find a decision list that is consistent with the sample it has seen. Despite its simplicity, it is an efficient PAC algorithm. It proceeds as follows: at each iteration we maintain a decision list L and a set of labelled examples S. Initially, L is an empty decision list and S are all the examples. At each iteration we check if all examples in S have same answers. If it is so, we set the output bit of L to this answer and we are done. Otherwise, we find such a literal ℓ that all examples in S that set ℓ to 1 have the same answer b. Then, we append pair (ℓ, b) to L, delete all examples from S that set ℓ to 1 and repeat the process.

Firstly, we need to prove that this algorithm succeeds in completing the list eventually. It is not hard to see, since at each step we are choosing the literal, that 'explains' some of the data. Choosing one possible literal over another one does not prevent us from choosing the latter on the later iteration.

Now we need to prove that this procedure is, in fact, an effective PAC learning algorithm.

Lemma 3.1. *Given a concept $c \in \mathcal{C}$ and a sample S of c of size m drawn according to $\mathcal{D}$, the probability that there exists a concept $h \in \mathcal{C}$ such that h is consistent with S and has $error(h) \geq \epsilon$ is at most*

$$|\mathcal{C}|(1-\epsilon)^m$$

Proof. Since probability that $h(x) = c(x)$ on the random example x drawn according to $\mathcal{D}$ is $1 - error(h) \leq 1 - \epsilon$, probability that h is consistent with randomly drawn sample S of size m is less than $(1-\epsilon)^m$. As there are $|\mathcal{C}|$ members in $\mathcal{C}$, probability that there exists such concept $h \in \mathcal{C}$ is at most $|\mathcal{C}|(1-\epsilon)^m$. □

If we choose m to satisfy

$$m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{C}| + \ln\frac{1}{\delta}\right) \tag{1}$$

it would imply that

$$\delta \geq |\mathcal{C}|(1-\epsilon)^m \geq Pr(error(h) \geq \epsilon) \tag{2}$$

where h is a hypothesis that is consistent with S. Since any decision list on n variables could be encoded in $\mathcal{O}(n \log n)$ bits, $\ln|\mathcal{C}|$ is polynomial in n and thus m is also polynomial in n. Since the algorithm itself runs in time polynomial in m, concept class of decision lists is efficiently PAC learnable. This algorithm is here mainly for the sake of lemma 3.1, which we will use several times throughout this paper.

3.2 *Simple online algorithm for learning decision lists*

We will give an algorithm that learns decision lists in mistake bound model due to Blum [Blu98].

The hypothesis maintained by the algorithm will be a slight generalization of the decision list: instead of simple "if/then" rules, each item of the list will contain a set of such "if/then" rules.

1. Initialize hypothesis h with a one-level list, whose level contains all $4n + 2$ possible "if/then" rules in the decision list, including two possible terminating rules.
2. Given an example x, find the first set in h that contains a rule whose "if" statement is satisfied by x. Predict based on this rule, breaking ties arbitrarily.
3. If the prediction is mistaken, move all the rules in that set that predicted incorrectly to the next list in the set.
4. Return to step 2.

We can make several observations about this algorithm. With each mistake at least one term will be move one level lower in h. The very first rule in target concept L will never be moved, and, inductively, ith rule of L won't get moved below ith level of h. Therefore, each rule will move at most k times, where k is the length of target decision list L, and thus the algorithm makes at most $\mathcal{O}(nk)$ mistakes, meaning that the algorithm learns decision lists in mistake bound model. Its running time per stage is at most $\mathcal{O}(n)$.

3.3 *Faster online algorithm for learning decision lists*

In this section we sketch an algorithm due to Klivans and Servedio [KS06] which runs in mistake bound model and learns decision lists with at most $2^{\mathcal{O}(k^{1/3})} \log n$ mistakes and runs in time $n^{\mathcal{O}(k^{1/3})}$ per sample. It is faster than the previous one in a sense that when k is fixed, sample complexity is logarithmic in n, but running time is still polynomial in n.

3.3.1 *Winnow algorithm*

In order to describe the algorithm, we need to know how to learn polynomial threshold functions. It can be done with the Winnow algorithm (due to Littlestone [Lit88]) which learns linear threshold functions.

Winnow takes as input a vector $w \in \mathbb{R}_+^n$ and two real numbers: a promotion factor $\alpha > 1$ and a threshold $\theta > 0$.

Winnow runs in iterations and throughout the execution maintains a hypothesis linear threshold function, given by values of w and θ.

On each iteration it receives a vector x and predicts the value of target function to be $sgn(x \cdot w - \theta)$. If that prediction turns out to be correct, no adjustments to the stored hypothesis are made and we proceed with another iteration.

Otherwise, Winnow updates the vector w as follows:

- If $sgn(x \cdot w - \theta) = 1$, it means that we have overestimated the actual cross product, so we divide all the w_i for which $x_i = 1$ by α.
- If $sgn(x \cdot w - \theta) = -1$, we need to increase the weights, so we multiply all the w_i for which $x_i = 1$ by α.

Notice that signs of stored coefficients never change, so in order to learn linear threshold functions with negative weights, we add variables $1 - x_i$ and a constant 1 element to the sample vectors, thus making them $(2n + 1)$-dimensional, and run Winnow on them instead.

Littlestone [Lit88] proved the following theorem that bounds the number of mistakes made by the Winnow:

Theorem 3.1. *Let $f(x)$ be a linear threshold function $sgn(w \cdot x - \theta)$, where $w_1, \ldots, w_n$ and θ are integers. Let $W = \sum_{i=1}^{n} |w_i|$. Then Winnow learns $f(x)$ with mistake bound $\mathcal{O}(W^2 \log n)$ and uses $\mathcal{O}(n)$ time steps per example.*

For our purposes, we would need a modified version of Winnow (called Expanded-Winnow in [KS06]), that will learn polynomial threshold functions. Instead of running Winnow directly on the input x, we transform vector x into a $\sum_{i=1}^{d} \binom{n}{i}$-dimensional vector, in which each bit corresponds to the value of some monomial over $x_1, \ldots, x_n$, and give the expanded vector to Winnow. Then, the resulting algorithm will run in $n^{\mathcal{O}(d)}$ per example and will make at most $\mathcal{O}(W^2 d \log n)$ mistakes.

3.3.2 *Constructing polynomial threshold functions for decision lists*

Now, we need to show that a low-weight polynomial threshold function exists for any decision list, then we will have an algorithm for learning decision lists with few mistakes just by applying Winnow.

For this we describe two constructions: outer and inner. We split a decision list $L = (\ell_1, b_1), (\ell_2, b_2), \ldots, (\ell_k, b_k), b_{k+1}$ into $\lceil k/h \rceil$ consecutive blocks of length at most h of form $(\ell_i, b_i), \ldots, (\ell_j, b_j), 0$, that is, an ordinary decision lists, except that it returns 0 if none of the literals in it were set to 1. Let f_i be a function that is computed by ith such block. It is easy to see that L is then computed by:

$$L(x) = sgn\left(\sum_{i=1}^{\lceil k/h \rceil} 3^{\lceil k/h \rceil - i + 1} f_i(x) + b_{k+1}\right) \tag{3}$$

This will be an outer construction.

Now we want to find low-weight and low-degree polynomials for each f_i. Let's begin with a simple polynomial for f_i. For $f_1 = (\ell_1, b_1), \ldots, (\ell_h, b_h), 0$ denote by ℓ_s^* x_i if ℓ_s is an unnegated variable x_i and $1 - x_i$ if ℓ_s is a negated variable $\overline{x_i}$. Then f_1 is computed by

$$f_1(x) = \ell_1^* b_1 + (1 - \ell_1^*)\ell_2^* b_2 + \cdots + (1 - \ell_1^*) \ldots (1 - \ell_{h-1}^*)\ell_h^* b_h$$

Since such polynomials have degree h and weight at most 2^{h+1}, when we put $f_1, \ldots, f_{\lceil k/h \rceil}$ into equation 3, we see that $L(x)$ is representable by a polynomial of degree h and of weight at most $3^{\lceil k/h \rceil - i + 1} \cdot 2^{h+1} = 2^{\mathcal{O}(k/h + h)}$.

To improve the result, we need to construct a polynomial that approximates f_i with low weight and low degree. This construction is rather technical, so we refer to [KS06], where they build such polynomials using Chebyshev polynomials and thus get that L can be computed by a polynomial threshold function of degree $\mathcal{O}(h^{1/2} \log h)$ and weight $2^{\mathcal{O}(k/h + h^{1/2} \log^2 h)}$. Taking $h = k^{2/3} / \log^{4/3} k$ they obtain

Theorem 3.2. *Any decision list L of length k over n variables is computed by a polynomial threshold function of degree $k^{1/3} \log^{1/3} k$ and weight $2^{\mathcal{O}(k^{1/3} \log^{4/3} k)}$.*

This theorem implies that Expanded-Winnow can learn decision lists with mistake bound $2^{\mathcal{O}(k^{1/3})} \log n$ and time $n^{\mathcal{O}(k^{1/3})}$.

3.4 *Relation between mistake bound model and PAC model*

Here we state the theorem that relates PAC learning with mistake bound learning.

Theorem 3.3. *If algorithm $\mathcal{A}$ learns a concept class $\mathcal{C}$ in mistake bound model with no more than M mistakes, then $\mathcal{A}$ also learns $\mathcal{C}$ in the probably approximately correct model.*

Proof. We need to transform algorithm $\mathcal{A}$ into a conservative one. A *conservative algorithm* is the one that updates its hypothesis only when it makes a mistake. Transformation is quite simple: before giving $\mathcal{A}$ an example, remember its state. Then, after we give the example to $\mathcal{A}$, if it predicts the result correctly, we just restore the saved image of $\mathcal{A}$. From $\mathcal{A}$'s point of view it would seem as though it received only examples on which it makes mistakes, but the number of mistakes is still bounded by M.

Now we have a conservative algorithm $\mathcal{A}'$. To make a PAC algorithm out of it, we run it on examples, halting if some hypothesis h survives for more than $K = \frac{1}{\epsilon}\ln(\frac{M}{\delta})$ successive examples. Then we return h as the final hypothesis.

The probability that h is accepted and $error(h) > \epsilon$ is at most $M(1-\epsilon)^K$, since we could have only chosen at most M "invalid" hypotheses, and each "invalid" hypothesis has $(1-\epsilon)^K$ probability that it would not fail on the K given examples.

$$M(1-\epsilon)^{\frac{1}{\epsilon}\ln(\frac{M}{\delta})} < Me^{-\frac{\epsilon}{\epsilon}\ln(\frac{M}{\delta})} = M\frac{\delta}{M} = \delta$$

That means that the presented algorithm is a PAC algorithm which takes $\frac{M}{\epsilon}\ln(\frac{M}{\delta})$ examples and has the same running time on each example as $\mathcal{A}$. □

Littlestone [Lit89] devised another algorithm for converting a mistake bound algorithm into a PAC learning one. It requires $\mathcal{O}(\frac{1}{\epsilon}(\ln(\frac{1}{\delta}) + M))$ examples and still runs in the same time as $\mathcal{A}$ per example, so it basically preserves all of the parameters of $\mathcal{A}$ while transforming it into a PAC algorithm.

4 LEARNING PARITIES

In this section we present three algorithms for learning parity functions.

4.1 *Gaussian elimination*

If we are given m labelled examples, the problem of determining the target parity function is that of linear algebra: we just need to find solutions of the system of linear equations over $GF(2)$ $Ax = b$, where A is a $m \times n$ matrix containing examples, and b is a vector with values of target parity function on these examples. The target parity function will always be among these solutions.

By corollary following from lemma 3.1, it is enough to take $m \geq 1/\epsilon(n + \ln(1/\delta))$ examples and return any of the solutions of this system. Since we can solve this problem in time $\mathcal{O}(m^3)$, we get an effective PAC algorithm for learning parities.

The problem with that algorithm is that it requires $\Omega(n)$ examples (in other words, its sample complexity is $\Omega(n)$). As it turns out, in case when length of the target parity function is known and is not too large, we can do better.

4.2 *Polynomial time algorithm for learning k-parities from sublinear number of examples*

The following theorem is due to Klivans and Servedio [KS06]:

Theorem 4.1. *The class of all parity functions of length at most k is PAC learnable in time $\mathcal{O}(n^4)$ using $\mathcal{O}(n^{1-1/k}\log n)$ examples. The hypothesis output by the learning algorithm is a parity function of length $\mathcal{O}(n^{1-1/k})$.*

Proof. We consider the case when $k = o(\log n)$, since in the other case we can just run the previous algorithm and get the claimed bounds.

Take $\ell = n^{1-1/k}$. Let H be all the parity functions of length at most ℓ. Request $m = \frac{1}{\epsilon}(\log|H| + \log(1/\delta))$ examples. Randomly choose a set

of $n - \ell$ variables and assign them the value 0. Now attempt to solve the resulting system of m linear equations in ℓ variables. Output the solution, or special value "Fail", in case when there are no solutions to the system.

Now we will show that the used system of linear equations has a solution with probability $\Omega(1/n)$. Since the expected number of repetitions of this algorithm before we find a solution is $\mathcal{O}(n)$, and we can solve the system of linear equations in $\mathcal{O}(n^3)$ time, the running time of the algorithm is $\mathcal{O}(n^4)$.

The probability that the system of linear equations has a solution is at least the probability that we have not set any of the k variables of the target parity to 0. The probability that we have not chosen any of the fixed k variables is:

$$\frac{\binom{n-k}{n-l}}{\binom{n}{n-l}} = \frac{\binom{n-k}{l-k}}{\binom{n}{l}} = \prod_{i=1}^{k} \frac{\ell - k + i}{n - k + i} > \left(\frac{\ell - k}{n - k}\right)^k = \left(\frac{\ell}{n}\right)^k \left(\frac{1 - \frac{k}{\ell}}{1 - \frac{k}{n}}\right)^k$$
$$> \frac{1}{n}\left(1 - \frac{k}{\ell}\right)^k > \frac{1}{n}\left(1 - \frac{k^2}{\ell}\right) > \frac{1}{2n}$$

□

4.3 *Algorithm with attribute-efficient sample complexity*

The following algorithm is due to Spielman [KS06], it learns parities on at most k variables in $\mathcal{O}(n^{k/2})$ time using $\mathcal{O}(k \log n)$ examples, while naïve exhaustive search takes $\mathcal{O}(n^k)$ time and uses the same number of examples.

The algorithm proceeds as follows: let's look at the example $(x_1, \ldots, x_n, y)$ as an $(n+1)$-dimensional vector $(x_1, \ldots, x_n, x_{n+1})$. Our goal is to find $s + 1 \leq k + 1$ indices $i_1, \ldots, i_s, i_{s+1}$, one of which is $n + 1$, such that $x_{i_1} + \cdots + x_{i_{s+1}}$ modulo 2 is zero for all examples.

Take some subset S of $\{1, \ldots, n, n+1\}$ and let ν_S be an m-dimensional vector whose ith coordinate equals value of χ_S on ith example.

Construct two lists of m-dimensional vectors. One consisting of vectors ν_S, where S ranges over $\left(\leq \frac{k}{2}\right)$-element subsets of $\{1, \ldots, n\}$, and the other one consisting of vectors $\nu_{S \cup \{n+1\}}$, where S again ranges over all $\left(\leq \frac{k}{2}\right)$-element subsets of $\{1, \ldots, n\}$.

Sort these two lists and find in them two equal vectors ν_{S_1} and $\nu_{S_2 \cup \{n+1\}}$. $\chi_{S_1 \cup S_2}$ is then a parity that is consistent with the given sample.

Thus, taking number of examples m to be $1/\epsilon(k \ln n + \ln 1/\delta)$, by lemma 3.1 we get that this is a PAC algorithm for k-parities.

4.4 *Learning parities with limited memory*

Previously we did not care about the memory consumption of the algorithms. Below we do not care about the running time instead.

Gaussian elimination, for example, needs $\frac{n^2}{4} + \mathcal{O}(n \log n)$ bits of memory, by keeping after seeing k examples matrix of size $k \times n$ which has an identity matrix in the first k columns. Can we learn parities with less memory but still looking only at a polynomial number of examples? Raz [Raz16] shows that if an algorithm uses less than $\frac{n^2}{25}$ bits of memory and the examples are uniformly distributed over $\{0,1\}^n$, then the number of used examples must be exponential.

Exponential algorithm that needs only n bits works as follows: iterate over all parities and for each of them ask for $\mathcal{O}(n)$ examples, halting if some parity is consistent with these examples. Its correctness follows from lemma 3.1.

If we look at k-parities, ideally we would like to be able to learn them with $poly(k \log n)$ examples and memory, since k-parity can be encoded with $k \log n$ bits. The simplest case is $k = 1$. Notice that in this case we have a

simple function: on input $(x_1, \ldots, x_n)$ it always outputs x_j for some fixed j, and we have to learn that index j.

We also note that being able to learn k-parities in PAC model with attribute-efficient sample complexity and memory is equivalent to being able to learn 1-parities attribute-efficiently, since for any example (x, y) we can construct new n^k-dimensional vector with values of various k-parities as coordinates and solve 1-parity problem for it.

4.4.1 *Algorithm for* 1*-parities with* $\log n$ *sample complexity*

We have to find such index i that $x_i = 1$ for all examples. We maintain n-bit vector $(\alpha_1, \ldots, \alpha_n)$, where $\alpha_j = 0$ indicates that $x_j \neq y$ in some example and otherwise $\alpha_j = 1$. After processing the whole sample S, algorithm returns any index i such that $\alpha_i = 1$. By lemma 3.1, if $|S| \geq 1/\epsilon(\ln n + \ln 1/\delta)$ then this is a PAC algorithm.

4.4.2 *Algorithm for* 1*-parities with* $\log n$ *memory*

Now we will maintain two variables: $left$ and $right$. Initially, $left = 1, right = n$. On each example if $x_{left} \neq y$, increment $left$. If $x_{right} \neq y$, decrement $right$. Return $left$ when whole sample is processed. We shall specify the size of the sample shortly.

4.4.3 *Generalization of two previous algorithms*

Let's break $\{1, \ldots, n\}$ into s continuous segments of length $\frac{n}{s}$ each. We will maintain $2s$ variables $left_i, right_i$, where i ranges from 1 to s. Variables $left_i, right_i$ will lie in ith segment, and initially will be equal to the corresponding segment's ends. Again, after looking at the example, for all i such that $x_{left_i} \neq y$ we increment $left_i$, and for all i such that $x_{right_i} \neq y$ we decrement $right_i$.

Notice that in case $s = 1$ this is the previous algorithm, and in case $s = n$ this is the first algorithm.

For this algorithm to correctly identify the target index, for all $i \in \{1, \ldots, s\} \setminus \{j\}$ $left_i$ must be greater than $right_i$, and $left_j$ must be equal to $right_j$. We will bound the probability that it happens.

We will look only at the case when examples are uniformly distributed over $\{0, 1\}^n$ (particularly, we will not prove that this is a PAC algorithm, since PAC learning algorithm must work for any probability distribution). Notice that $right_i - left_i$ decreases with probability at least $1/2$. Let $\xi_i = left_i - right_i + \frac{n}{s}$. Then by using Chernoff bounds probability that after m steps $\xi_i \leq \frac{n}{s}$ is:

$$Pr\left(\xi_i \leq \frac{m}{2} - \left(\frac{m}{2} - \frac{n}{s}\right)\right) \leq \exp\left(-\frac{2\left(\frac{m}{2} - \frac{n}{s}\right)^2}{m}\right) =$$
$$\exp\left(-\frac{m}{2} + 2\frac{n}{s} - \frac{2n^2}{s^2 m}\right) \leq \exp\left(-\frac{m}{2} + 2\frac{n}{s}\right)$$

The probability that all the ξ_i will increment at least $\frac{n}{s}$ times is at least:

$$\left(1 - \exp\left(-\frac{m}{2} + 2\frac{n}{s}\right)\right)^s \geq 1 - s\exp\left(-\frac{m}{2} + 2\frac{n}{s}\right) \geq 1 - \delta$$

where δ is from definition of PAC algorithm.

$$s\exp\left(-\frac{m}{2} + 2\frac{n}{s}\right) \leq \delta$$
$$-\frac{m}{2} \leq \ln\frac{\delta}{s} - 2\frac{n}{s}$$
$$m \geq 2\ln\frac{s}{\delta} + 4\frac{n}{s}$$

If we take $m = 2\ln\frac{s}{\delta} + 4\frac{n}{s}$, probability of a mistake would not exceed δ. Applying this bound to the previous algorithm, we see that it requires $\mathcal{O}(n)$ samples.

We get that for fixed s this algorithm uses $2s\log\frac{n}{s}$ bits of memory and $2\ln\frac{s}{\delta} + 4\frac{n}{s}$ examples.

In order to make this algorithm work for any distribution, we need to modify it slightly. Create $2s$ new variables $ageleft_i$ and $ageright_i$, which indicate for how long the corresponding variable ($left_i$ or $right_i$) has not changed. When any age variable exceeds $\frac{1}{\epsilon}(\ln n + \ln\frac{1}{\delta})$, halt and return corresponding index. For this algorithm total number of examples used is then not greater than $\frac{n}{2s\epsilon}(\ln n + \ln\frac{1}{\delta})$.

It is easy to notice that this bound does not yield an attribute-efficient algorithm for 1-parities for any value of s, so some other technique may be needed.

Also, methods from [Raz16] can't be applied here since their proof considerably relies on affine structure of the set of possible solutions.

Problem 1. Is there an algorithm that learns 1-parities with uniformly distributed examples with both polylogarithmic sample complexity and memory if we require the probability of mistake to be not greater than some fixed constant $\delta \in (0, 1)$?

Problem 2 (stronger version)**.** Is there an algorithm that PAC learns 1-parities (or, equivalently, k-parities) with both sample complexity and memory bounded by $poly(\log n, \frac{1}{\epsilon}, \frac{1}{\delta})$ (by $poly(k\log n, \frac{1}{\epsilon}, \frac{1}{\delta})$)?

REFERENCES

[Blu98] Avrim Blum. *On-line algorithms in machine learning*. Springer, 1998.

[KS06] Adam R Klivans and Rocco A Servedio. Toward attribute efficient learning of decision lists and parities. *The Journal of Machine Learning Research*, 7:587–602, 2006.

[KV94] Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

[Lit89] Nick Littlestone. From on-line to batch learning. In *Proceedings of the second annual workshop on Computational learning theory*, pages 269–284, 1989.

[Raz16] Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *arXiv preprint arXiv:1602.05161*, 2016.

[Riv87] Ronald L Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.