Boosting over non-deterministic ZDDs

Fujita, Takahiro
Department of Informatics, Kyushu Univerisity

Hatano, Kohei
Faculty of Arts and Sciences, Kyushu University

Takimoto, Eiji
Department of Informatics, Kyushu University

https://hdl.handle.net/2324/2231502

出版情報：Theoretical computer science, 2018-12-04. Elsevier
バージョン：
権利関係：

Boosting over non-deterministic ZDDs

Takahiro Fujita[a], Kohei Hatano[b], Eiji Takimoto[a]

[a] *Department of Informatics, Kyushu University*
[b] *Faculty of Arts and Science, Kyushu University / RIKEN AIP*

Abstract

We propose a new approach to large-scale machine learning, *learning over compressed data*: First compress the training data somehow and then employ various machine learning algorithms on the compressed data, with the hope that the computation time is significantly reduced when the training data is well compressed. As a first step toward this approach, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data, which is a generalization of the ZDD by incorporating non-determinism. For the learning algorithm to be employed, we consider a boosting algorithm called AdaBoost$*$ and its precursor AdaBoost. In this paper, we give efficient implementations of the boosting algorithms whose running times (per iteration) are linear in the size of the given ZDD.

Keywords: data compression, optimization, machine learning, boosting, ZDD

1. Introduction

Most tasks in machine learning are formulated as optimization problems of various types. Recently, the amount of data to be treated is growing enormously large, and so the demands on scalable optimization methods are increasing. Probabilistic approach such as stochastic gradient descent methods [1] is now widely employed as standard techniques for large scale machine learning. Obviously, these methods require the time and/or the space complexity to be proportional to the size of given data.

In this paper, we propose a new approach: *learning over compressed data*. That is, we first compress given data somehow, and then employ various machine learning algorithms on the compressed data without explicitly re-constructing the original data. To be more precise, for any target machine learning algorithm to be employed, we apply an efficient algorithm running over the compressed data, which simulates the behavior of the target algorithm running over the original data, with the hope that the time and space complexity are significantly reduced when the data is well compressed. Although we need extra computation for compressing a given sample data set, we can expect great improvement of time and space complexity, especially when high compression ratio is achieved.

The methodology of working over compressed data has gained much attention in the areas of database and data mining, where various methods have been developed for string search from a compressed string and frequent word extraction from compressed texts [2, 3, 4]. But, as far as the authors are aware, most methods developed so far are limited to simple tasks such as search and counting, and few results are known for more complex tasks such as optimization in machine learning. Notable exceptions contain the results of Nishino et al. [5] and Tabei et al. [6], where they apply string compression techniques to matrix-based computations so as to reduce space complexity, rather than to reduce time complexity.

As a first step toward establishing a general methodology of learning from compressed data, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data. The ZDD is a general data structure for representing a family of sets [7, 8], and is appropriate for our purpose. Many results reported that the ZDD indeed has ability of compactly representing various data in various domains [9, 10, 11].

In this paper, we slightly generalize the ZDD by incorporating non-determinism and propose a new data structure called the non-deterministic ZDD (NZDD, for short). The NZDD has more flexibility for representing data because of the non-determinism. Also, the NZDD naturally fit to the boosting algorithms to be simulated. On the other hand, it seems to be NP-hard to construct an NZDD of minimal size from a given training data. An efficient construction method of succinct NZDDs is left as future work. Practically, the ZDD and the DFA may provide good alternatives since they are easy to construct.

For the learning algorithm to be employed over the NZDD representation of the training data, we consider a boosting algorithm called the AdaBoost$*$ [12]. The AdaBoost$*$ is a refined version of the seminal boosting algorithm AdaBoost [13] and is guaranteed to find a hyperplane that maximizes the margin.

In this paper, we give an algorithm running over the given NZDD that efficiently simulates the AdaBoost$*$. Its running time (per iteration) does not depend on the size of training data but is only linear in the size of the given NZDD. In addition, the implementation also works for the AdaBoost as well and a similar guarantee also holds. So, our method takes advantage when the size of the NZDD is significantly smaller than the size of the training data, provided that the time complexity of constructing the NZDD is moderately small.

We conduct preliminary experiments over artificial data, where we use ZDDs instead of NZDDs for representing the training data. It turns out that the construction time is negligible and the size of ZDDs constructed are indeed very small. Hence, the results show that our method outperforms the AdaBoost$*$ as expected.

2. Problem statement and AdaBoost$*$

First we describe the problem of 1-norm hard margin maximization and then briefly review the AdaBoost$*$ which is one of the boosting algorithms that solve the problem.

2.1. 1-norm hard margin maximization

Let X be a set called the instance space, and assume that we are given a finite set of *base hypotheses* $H = \{h_1, h_2, \ldots, h_n\} \subseteq \{h : X \to \{0, 1\}\}$. Note that the base hypotheses are usually assumed to take values in $\{-1, 1\}$, but since any function $g : X \to \{-1, 1\}$ can be represented as the difference of 0-1 valued functions (e.g., $g(x) = \mathbf{1}[g(x) = 1] - \mathbf{1}[g(x) = -1]$), we can assume 0-1 valued hypotheses without loss of generality. The base hypothesis class H defines a feature map, which maps any instance $x \in X$ to the feature vector $(h_1(x), h_2(x), \ldots, h_n(x))$ in the feature space $\{0, 1\}^n$. Later we will regard the feature vector for x as the set $H(x) = \{h_j \mid h_j(x) = 1\}$ and thanks to the assumption above, any base hypothesis $h_j \notin H(x)$ takes value 0 for x, which is a crucial property that makes our algorithm work.

Now we give the problem statement of 1-norm hard margin maximization. The input is a *sample* $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$, where x_i for $y_i = 1$ is called a positive instance and x_i for $y_i = -1$ a negative instance, and the output is a hyperplane in the feature space that separates the positive instances from the negative instances as much as possible. More precisely, the goal is to find

$$\boldsymbol{\alpha}^* = \arg \max_{\boldsymbol{\alpha} \in \{\mathbb{R}^n \mid \|\boldsymbol{\alpha}\|_1 = 1\}} \min_{1 \le i \le m} y_i \sum_{j=1}^{n} \alpha_j h_j(x_i). \tag{1}$$

We denote by $\boldsymbol{\alpha} \in \{\mathbb{R}^n \mid \|\boldsymbol{\alpha}\|_1 = 1\}$ the hyperplane whose normal vector is $\boldsymbol{\alpha}$, which also represents the convex combination of base hypotheses $f(x) = \sum_{j=1}^{n} \alpha_j h_j(x)$. (Here we extend the base hypotheses to include $\{-h_1, -h_2 \dots, -h_n\}$ as well, so that we allow the weights α_j to take negative values.) Note that since the 1-norm of $\boldsymbol{\alpha}$ is normalized, $|f(x)|$ denotes the distance of the feature vector $(h_1(x), \dots, h_n(x))$ to the hyperplane $\boldsymbol{\alpha}$ measured by ∞-norm. Thus, the signed distance $y_i f(x_i)$ (which is positive if and only if f correctly classifies x_i) is called the margin of the hyperplane $\boldsymbol{\alpha}$ with respect to the labeled instance (x_i, y_i). Let $\rho = \min_i y_i f(x_i)$ be the minimum margin of $\boldsymbol{\alpha}$ over all labeled instances in the sample. Note that $\boldsymbol{\alpha}^*$ is the hyperplane that maximizes ρ. It is well known that if $\rho > 0$, which means that the sample S is linearly separable, then the combined hypothesis f has a generalization error bound that is proportional to $1/\rho$ [14]. So, the goal of maximizing ρ is natural. Let $\rho^* = \min_i y_i \sum_j \alpha_j^* h_j(x_i)$ be the optimal margin.

In what follows, we assume without loss of generality that all labeled feature vectors $(h_1(x_i), \dots, h_n(x_i), y_i)$ are distinct.

2.2. AdaBoost∗

The optimization problem (1) can be formulated as a linear programming problem of size $O(nm)$ and hence efficiently solved by an LP solver. However, in many cases, the number n of base hypotheses is very large (sometimes infinite), and thus the problem is infeasible for LP solvers. In such cases, boosting may provide an alternative way. In particular, the AdaBoost∗ of Rätsch and Warmuth [12] provably converges to the maximum margin ρ^* within precision $\nu \in (0, 1]$ in $\frac{2 \log m}{\nu^2}$ iterations. Below we describe how the AdaBoost∗ behaves when applied to the base hypothesis class H. On each round $t = 1, 2, \dots, T$, it (i) computes a distribution d_t over the sample S, (ii) finds a base hypothesis $h_{j_t} \in H$ with the maximum *edge* (average margin) with respect to d_t, and (iii) updates the coefficient α_{j_t}. Finally, normalizing the coefficient $\boldsymbol{\alpha}$, it obtains a final hypothesis f. A pseudocode is given in Algorithm 1, where part (ii) above is implemented in a very naive manner:

Algorithm 1 AdaBoost∗

```
Input: $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$, $\nu \in (0, 1]$
Output: $f$

  1. Let $\alpha_j = 0$ for $j = 1, \ldots, n$
  2. Let $d_1(i) = 1/m$ for $i = 1, \ldots, m$
  3. For $t = 1, \ldots, T$
     (a) Compute the edges
         $\gamma_{t,j} = \sum_{i=1}^{m} d_t(i) y_i h_j(x_i)$ for $j = 1, \ldots, n$.
     (b) Let $j_t = \arg\max_{1 \le j \le n} |\gamma_{t,j}|$; $\gamma_t = \gamma_{t,j_t}$.
     (c) Set $\rho_t = \min_{r=1,\ldots,t} |\gamma_r| - \nu$;
     (d) Update coefficients $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t} - \frac{1}{2} \log \frac{1+\rho_t}{1-\rho_t}$
     (e) Update weights
         $d_{t+1}(i) = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i))/Z_t$
         for $i = 1, \ldots, m$, where
         $Z_t = \sum_{i=1}^{m} d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i))$
  4. Let $f(x) = \sum_{j=1}^{n} \frac{\alpha_j}{\|\boldsymbol{\alpha}\|_1} h_j(x)$
```

compute the edges of all base hypotheses (line 3-(a)) and then choose the maximum among them (line 3-(b)). So, this implementation is inefficient for a very large n. But, AdaBoost∗ (and any other boosting algorithm) has a considerable advantage over LP solvers when the hypothesis class H has an efficient implementation, called the base learner, for this part: to find a base hypothesis with the maximum edge from a given distribution over the sample. In this case, the two lines (3-(a) and 3-(b)) are replaced by the base learner. The next theorem shows a performance guarantee of the AdaBoost∗.

Theorem 1 (Rätsch and Warmuth [12]). *If $T \geq \frac{2 \log m}{\nu^2}$, then AdaBoost∗ (Algorithm 1) outputs a combined hypothesis f such that* $\min_{1 \le i \le m} y_i f(x_i) \geq \rho^* - \nu$.

In this paper, we consider the situation where the size n of H is small but the sample size m is very large, as is often the case, and both the direct applications of LP solvers and the AdaBoost∗ may be useless.

2.3. AdaBoost

The AdaBoost, proposed by Freund and Schapire [13], is a precursor of the AdaBoost∗. The algorithm, unlike the AdaBoost∗, is not shown to

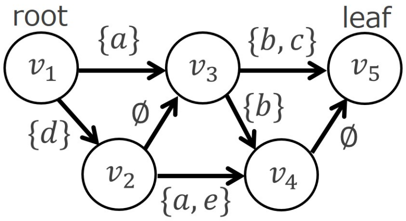

Figure 1: An NZDD representation for $\{\{a, b\}, \{a, b, c\}, \{a, d, e\}, \{b, c, d\}, \{b, d\}\}$.

provably maximize the hard margin. However, it is shown that it achieves at least half of the maximum hard margin asymptotically under weak technical conditions [15, 12]. Besides, the AdaBoost is much more popular because of its simplicity and the empirical performances. The behavior of the AdaBoost is almost the same as the AdaBoost∗. More precisely, instead of 3-(c) and 3-(d) in Algorithm 1, the AdaBoost updates the coefficient as $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t}$. Therefore, the theoretical results we will show also are applicable to the AdaBoost.

3. A dag representation for samples

As a data structure for storing the sample, we propose a dag representation for a family of sets called the non-deterministic ZDD (NZDD, for short). It can be seen as a generalization of the ZDD by incorporating non-determinism.

3.1. Non-deterministic ZDD (NZDD)

An NZDD is specified by a 4-tuple $G = (V, E, \Sigma, \Phi)$, where (V, E) is a directed acyclic graph with a single root and a single leaf, Σ is a ground set, and $\Phi : E \to 2^\Sigma$ is a function that assigns to each edge e a subset $\Phi(e)$ of Σ. Note that $\Phi(e)$ can be the empty set $\emptyset$. Furthermore we require the additional properties as described below. Let $\mathcal{P}_G$ be the set of all paths from the root to the leaf in G, where a path P in $\mathcal{P}_G$ is specified by the set of edges in P, i.e., $P \subseteq E$.

1. Every path $P \in \mathcal{P}_G$ represents a subset $\Psi(P) \subseteq \Sigma$ defined as $\Psi(P) = \bigcup_{e \in P} \Phi(e)$. Thus, the NZDD G defines a subset family as $L(G) = \{\Psi(P) \mid P \in \mathcal{P}_G\} \subseteq 2^\Sigma$.

2. For every path $P \in \mathcal{P}_G$, $\Phi(e) \cap \Phi(e') = \emptyset$ for any $e, e' \in P$ with $e \neq e'$.
3. For every pair of paths $P, P' \in \mathcal{P}_G$, $\Psi(P) \neq \Psi(P')$ if $P \neq P'$.

The second property says that every element $a \in \Sigma$ appears at most once in every path $P \in \mathcal{P}_G$. That is, letting $E(a) = \{e \in E \mid a \in \Phi(e)\}$, we have $|E(a) \cap P| \leq 1$ for every $P \in \mathcal{P}_G$. The third property says that there exists a one-to-one correspondence between the set of paths $\mathcal{P}_G$ and the subset family $L(G)$. In particular, we have $|\mathcal{P}_G| = |L(G)|$. Note that while the second property is crucial for our algorithm to work, the third property does not need to be strictly satisfied. More precisely, the duplication of a subset does not matter for our algorithm, except that the number of duplications affects the convergence rate.

Finally, we define the size of G as $|G| = \sum_{e \in E} |\Phi(e)|$. Note that the size $|G|$ can be significantly small as compared with the number of paths $|\mathcal{P}_G|$. In other words, the NZDD G is a compact representation for the subset family $L(G)$. As an example, we give in Fig. 1 an NZDD that represents a subset family.

3.2. NZDD representation for the sample

Now we describe how we represent the sample S as an NZDD.

Recall that $H(x) = \{h_j \in H \mid h_j(x) = 1\}$ for each instance $x \in X$. Let $Z^+ = \{H(x_i) \mid (x_i, 1) \in S\}$ and $Z^- = \{H(x_i) \mid (x_i, -1) \in S\}$ be the subset families with the ground set $\Sigma = H$, which correspond to the positive and the negative instances in the sample S, respectively. Let G^+ and G^- be NZDDs for the families Z^+ and Z^-, respectively. That is, $L(G^+) = Z^+$ and $L(G^-) = Z^-$. Finally, the NZDD G for the sample S is obtained by (i) putting an additional node as the global root with two outgoing edges labeled with $\emptyset$, where one edge is connected to the root of G^+ and the other is to the root of G^-, and (ii) merging the leaves of G^+ and G^- to a single leaf (See Fig. 2 for example). Note that G is not necessarily a minimal NZDD even if G^+ and G^- are minimal, because G may be further simplified by merging a node in G^+ and a node in G^-. But, we define G in this way, so that any path in G^+ and any path in G^- are disjoint.

3.3. Relations to ZDDs and NFAs

We show that the ZDD representation is a special case of the NZDD representation. To see this, we consider the class of NZDDs of the following form:

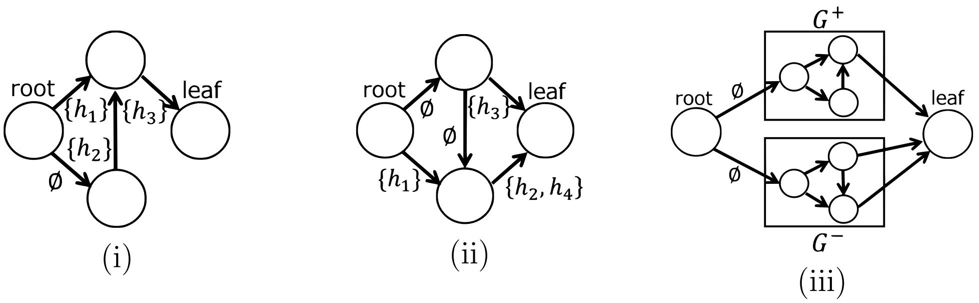

Figure 2: (i) An NZDD G^+ for $Z^+ = \{\{h_1, h_3\}, \{h_2, h_3\}\}$; (ii) An NZDD G^- for $Z^- = \{\{h_1, h_2, h_4\}, \{h_2, h_4\}, \{h_3\}\}$; (iii) An NZDD for the sample consisting of positive instances Z^+ and negative instances Z^-

1. Each edge e is labeled with either a singleton or the empty set. That is, $|\Phi(e)| \leq 1$.
2. Each internal node has one or two outgoing edges. If it has two outgoing edges, one of them is labeled with the empty set.
3. There exists a fixed ordering over Σ such that for any pair of edges e and e' labeled with singletons $\{a\}$ and $\{a'\}$, respectively, if e is an ancestor of e', then a precedes a' in this ordering.

It is easy to see that any ZDD can be seen as an NZDD in this form, by letting each 0-edge be associated with the empty set and 1-edge with a singleton $\{a\}$ where a is the node label of the origin of the 1-edge. This characterization gives an alternative definition of ZDDs.

NZDDs are more flexible than ZDDs even if we restrict them to be ordered. For example, for each pair of edges e and e' outgoing from a common node, we allow non-determinism in the sense that $\Phi(e) \cap \Phi(e') \neq \emptyset$. Another example is that when taking the union of two (disjoint) subset families $L(G_1)$ and $L(G_2)$ represented by NZDDs G_1 and G_2, respectively, we can easily construct an NZDD that represents $L(G_1) \cup L(G_2)$ in the following way: just put an additional node as the global root with two outgoing edges labeled with $\emptyset$, where one edge is connected to the root of G_1 and the other is to the root of G_2. This construction ensures that the NZDD size for the union is bounded by the sum $|G_1|$ and $|G_2|$. This contrasts with the fact that, for the ZDD case, the size for the union is bounded by the product $|G_1||G_2|$ even if G_1 and G_2 are ZDDs.

Next we consider the relation of ordered NZDDs to NFAs. Under the ordering over Σ, we can identify a subset $\{a_1, a_2, \ldots, a_k\} \subseteq \Sigma$ with a string $a_1a_2 \cdots a_k \in \Sigma^*$ over the alphabet Σ, where $a_1 < a_2 < \cdots < a_k$ under the ordering $<$. Note that the empty set corresponds to the empty string ϵ. In this way, a subset family can be seen as a language. From this viewpoint, we can regard an NZDD G as an NFA that recognizes the language $L(G)$, with the root identified with the start state and the leaf with the unique accepting state. The difference is that, in the NZDD representation, we have only a single accepting path for each string in the language. This implies that any DFA for such a language can be converted to an NZDD in an obvious way. Note that in order to make the accepting state unique, we may need to put an additional leaf and connecting every accepting state to the leaf by an additional edge labeled with the empty set (ϵ-transition).

3.4. Complexity of constructing NZDDs

When given a subset family $L \subseteq 2^\Sigma$, we want to compute a minimal NZDD G with $L(G) = L$. So far, the time complexity of the problem is unknown, but it seems to be NP-hard because so are the closely related problems, namely, construction of a minimal ZDD (over all ordering) [8] and construction of a minimal NFA [16]. On the other hand, we have a very efficient algorithm for constructing a minimal ZDD when given an ordering [17] and a linear time algorithm for constructing a minimal DFA for a finite language [18]. So, practically, we can use these algorithms for constructing an ordered NZDD of small size.

4. Simulating AdaBoost$*$ over an NZDD representation for the sample

In this section, we give an algorithm that efficiently simulates the AdaBoost$*$ over an NZDD G that represents a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, without explicitly reconstructing the sample S from G. In particular, the running time (per iteration) of our algorithm does not depend on the sample size m but is linear in the size of G. First we state the main theorem.

Theorem 2. *There exists an algorithm that, when given an NZDD G that represents a sample S, exactly simulates AdaBoost$*$ whose running time is $O(|G|)$ per iteration.*

So, if the sample is significantly compressed in the NZDD representation, our algorithm runs much faster than the direct application of the AdaBoost$*$

when the computation time of constructing G from S is negligible. More specifically, if we use a linear time algorithm for constructing an NZDD from a minimal DFA as described in the previous section, then the total running time of our algorithm is $O(nm + T|G|)$, whereas the total running time of the direct application of the AdaBoost$*$ is $O(nmT)$. So, if $|G| \ll nm$, then our algorithm would be faster[1].

Further, since the AdaBoost is almost identical to the AdaBoost$*$ in an algorithmic sense, we have the following corollary as well.

Corollary 1. *There exists an algorithm that, when given an NZDD G representing S, simulates AdaBoost whose running time is $O(|G|)$ per iteration.*

Below we describe a basic idea of the algorithm. Obviously, we cannot explicitly maintain the distribution d_t over the sample S. Instead, we maintain one weight $w_{t,e}$ for each edge e of G, so that the edge weights w_t implicitly represents d_t. The same idea is used in [19] to efficiently simulate online prediction algorithms with multiplicative update rules, where the decision space is the set of paths of a given directed acyclic graph.

To describe the idea formally, we need some additional notations. Recall that there exists a one-to-one correspondence between the sample S and the set of all root-to-leaf paths $\mathcal{P}_G$ in G. So, we identify a labeled instance $(x_i, y_i) \in S$ with a path $P \in \mathcal{P}_G$, and we will denote the weight for the instance by $d_t(P)$ instead of $d_t(i)$. Furthermore, let $\mathcal{P}_G^+$ and $\mathcal{P}_G^-$ denote the set of paths that pass through G^+ and the set of paths that pass through G^-, respectively.

Now we give the two conditions C1 and C2 that the edge weights w_t need to satisfy, so as to represent the path distribution d_t.

C1. The edge weights w_t need to satisfy

$$d_t(P) = \prod_{e \in P} w_{t,e}$$

for every path (labeled instance) $P \in \mathcal{P}_G$.

C2. The outflow from each internal node should be one. That is, w_t need to satisfy

$$\sum_{a:(u,a) \in E(G)} w_{t,(u,a)} = 1$$

[1] Note that it always holds that $|G| \leq nm$.

Algorithm 2 Initializing the path distribution

```
1. Let $w'_e = 1$ for all edges in $G$.
2. Apply the Weight Pushing algorithm to $w'$ and get $w_1$.
```

for every internal node u, where $E(G)$ denotes the set of edges of G.

What we need to show is how to simulate AdaBoost$*$ efficiently by using the edge weights w_t. More precisely, we need to simulate the two parts of AdaBoost$*$:

(a) updating the path distributions d_t (corresponding to Line 2 and Line 3-(e) of Algorithm 1), and

(b) computing the edges $\gamma_{t,j}$ (corresponding to Line 3-(a)).

In the following subsections, we give algorithms that simulate the two parts.

4.1. Updating the path distributions d_t

To simulate this part, we use the Weight Pushing algorithm developed by [20], which rearranges the edge weights so that relative weights on the path remain unchanged but again satisfy the two conditions. More precisely, the Weight Pushing algorithm has the following property.

Proposition 1 (Mohri [20])**.** *When given arbitrary edge weights $w'_e \geq 0$, the Weight Pushing algorithm produces edge weights w_e in time $O(|E|)$ such that w_e satisfies condition C2 and*

$$\prod_{e \in P} w_e = \frac{\prod_{e \in P} w'_e}{\sum_{P \in \mathcal{P}_G} \prod_{e \in P} w'_e}$$

for every path $P \in \mathcal{P}_G$.

The initialization of the path weights ($d_1(P) = 1/m$) of Line 2 of Algorithm 1 can be realized by the two steps as described in Algorithm 2. It is justified by Proposition 1 which implies

$$\prod_{e \in P} w_{1,e} = \frac{1}{|\mathcal{P}_G|} = 1/m = d_1(P).$$

Moreover, the running time of Algorithm 2 is $O(|E|)$.

Algorithm 3 Updating the path distribution

```
1. Forall $e \in E(G)$, let $w'_e = w_{t,e}$
2. Forall $e \in E(G^+)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(-\alpha_{j_t})$
3. Forall $e \in E(G^-)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(\alpha_{j_t})$
4. Apply the Weight Pushing algorithm to $w'$ and get $w_{t+1}$.
```

The update of path distributions of Line 3-(e) of Algorithm 1 can be realized by multiplying the weights of the edges e such that $h_{j_t} \in \Phi(e)$, and applying the Weight Pushing algorithm. See Algorithm 3 for more details.

Below we give a justification of Algorithm 3.

Lemma 1. *Algorithm 3 exactly simulates Line 3-(e) of Algorithm 1 in time $O(|E|)$.*

Proof. Let P be a path in $\mathcal{P}_G$ that corresponds to a labeled instance (x_i, y_i) and examine the quantity $\prod_{e \in P} w'_e$. Recall that

$$\bigcup_{e \in P} \Phi(e) = \{h_j \in H \mid h_j(x_i) = 1\}$$

by the definition of the NZDD construction for S.

First consider the case where $h_{j_t}(x_i) = 0$. In this case, there is no edge $e \in P$ such that $h_{j_t} \in \Phi(e)$. Therefore, $w'_e = w_{t,e}$ for all edges e in P. Thus,

$$\begin{aligned}\prod_{e \in P} w'_e = \prod_{e \in P} w_{t,e} &= d_t(P) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

Next consider the case where $y_i = 1$ (i.e., P passes through G^+) and $h_{j_t}(x_i) = 1$. In this case, there exists a unique edge $e \in P$ such that $h_{j_t} \in \Phi(e)$. The uniqueness comes from Property 2 of the NZDD. So, $w'_e = w_{t,e} \exp(-\alpha_{j_t})$ for the edge e. Since $h_{j_t} \notin \Phi(e')$ for any other edge $e' \in P$, we have

$$\begin{aligned}\prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e}\right) \exp(-\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

For the last case where $y_i = -1$ and $h_{j_t}(x_i) = 1$, a similar argument to the case above gives

$$\begin{aligned}\prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e}\right) \exp(\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

Hence for all paths P, we have

$$\prod_{e \in P} w'_e = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).$$

Therefore, Proposition 1 ensures that w_{t+1} represents the path distribution d_{t+1} as desired. □

4.2. Computing the edges $\gamma_{t,j}$

To compute $\gamma_{t,j}$, we first compute the following quantity

$$f_e = \sum_{P \in \mathcal{P}_G : e \in P} d_t(P)$$

for all edges e, which can be interpreted as the probability flow of edge e, i.e., the probability that the path P goes through edge e when P is chosen according to the distribution d_t. Since G is a directed acyclic graph, we can compute f_e for all edges e by dynamic programming (e.g., the forward-backward algorithm) in linear time. Then, it is not hard to see that $\gamma_{t,j}$ can be computed by

$$\gamma_{t,j} = \sum_{e \in E(G^+) : h_j \in \Phi(e)} f_e - \sum_{e \in E(G^-) : h_j \in \Phi(e)} f_e.$$

We summarize the result as in the following lemma.

Lemma 2. *There exists an algorithm that exactly simulates Line 3-(a) of Algorithm 1 in time $O(|G|)$.*

Theorem 2 follows from Lemma 1 and Lemma 2.

4.3. On Property 3 of the NZDD

Finally, we remark on the necessity of Property 3 of the NZDD.

Assume that an NZDD G representing a sample given by Z^+ and Z^- does not satisfy Property 3. That is, some instances $H(x_i)$ are represented by two or more paths in G. In this case, letting the sample be redefined as multisets $\tilde{Z}^+ = \bigcup_{P \in \mathcal{P}^+} \Psi(P)$ and $\tilde{Z}^- = \bigcup_{P \in \mathcal{P}^-} \Psi(P)$, we can see that our algorithm simulates AdaBoost$*$ running over the new sample $(\tilde{Z}^+, \tilde{Z}^-)$, where the sample size is now $\tilde{m} = |\mathcal{P}_G|$, which is greater than m. Fortunately, the optimization problem (1) remains unchanged with some labeled instances (x_i, y_i) duplicated, the AdaBoost$*$ works and hence our algorithm works. The difference is that the convergence rate becomes worse, which is $2(\log \tilde{m})/\nu^2$. But the dependence on the sample size is only logarithmically, the negative effect seems not very serious.

5. Experiments

In this section, we show some preliminary experimental results on artificial data comparing our method with the direct application of the AdaBoost$*$. The experiments were performed on a server with four cores of Intel Xeon CPU X5560 2.80GHz and 198 GB memory. We implemented algorithms in C and C++. For a practical reason, we consider ZDDs instead of NZDDs. For an algorithm to compress input data Z^+ and Z^- into ZDDs G^+ and G^-, we used the zcomp of Toda [17] [2], which is a very fast algorithm for constructing a ZDD of minimal size from a given subset family. More precisely, we simplified each of the ZDDs $G = (V, E, \Sigma, \Phi)$ obtained by zcomp by replacing every isolated paths (a sequence of edges $(e_1, e_2, \ldots, e_k)$ of outdegree 1) with a single edge e' with edge label $\Phi'(e') = \bigcup_{i=1}^{k} \Phi(e)$. The resultant NZDD $G' = (V, E', \Sigma, \Phi')$ is of the same size as G, but may have much less number of edges than G, that is, $|E'| \ll |E|$.

For the AdaBoost$*$ and our implementation, we set $T = 100$ and $\nu = 0.01$. Note that $T = 100$ rounds may not be large enough for the margin of the combined hypothesis to converge to within $\rho^* - \nu$. But the purpose of the experiment is to see how fast our method runs relative to the AdaBoost$*$, and so we could choose T large enough, so that the time for constructing

[2] http://www.sd.is.uec.ac.jp/toda/code/zcomp.html

NZDDs are negligible. Indeed, the zcomp is very fast and it produces all the ZDDs in the experiments in less than 10% of the total computation time.

We generate artificial data in the following way. The domain is $X = \{0,1\}^n$. We choose a set of m instances (without repetition) uniformly at random from $\{0,1\}^n$. To each instance $x \in \{0,1\}^n$, we assign a label $+1$ if the sum of the first k components of x is more than or equal to r and -1 otherwise. That is, the label for x is $+1$ if and only if $x_1 + x_2 + \cdots + x_k \geq r$, where x_i is the i-th component of x. We consider the hypothesis class $H = \{h_0, h_1, h_2, \ldots, h_n\}$, where h_0 is the constant function that maps any instance to 1, and h_j is a projection function defined as $h_j(x) = x_j$ for $1 \leq j \leq n$. Finally, we transform the sample S of size m obtained as described above into the subset families $Z^+ = \{H(x) \mid (x, 1) \in S\}$ and $Z^- = \{H(x) \mid (x, -1) \in S\}$, where $H(x) = \{j \mid h_j(x) = 1\}$. (Note that the hypothesis h_j is simply named j.) Then, we apply the zcomp to Z^+ and Z^- and obtain an NZDD representation G for the sample. In our experiments, we set $k = 10$, $r = 5$, and examine the performance of AdaBoost$*$ and our method for various dimensions n chosen from $\{20, 40, 60, 80, 100\}$ and sample sizes m from $\{100000, 200000, \ldots, 1000000\}$, respectively.

We first examined the compression ratio for each sample (Z^+, Z^-). The compression ratio is naturally defined as the ratio of the output NZDD size $|G|$ over the input size, where the input size, denoted by $|(Z^+, Z^-)|$, is the sum of $|H(x)|$ for all instances x in the sample. That is, $|(Z^+, Z^-)| = \sum_{H \in Z^+ \cup Z^-} |H|$. In Table 1 (second and fifth columns) we show the input size $|(Z^+, Z^-)|$ and the NZDD size $|G|$ for the case $n = 40$, and in Fig. 3 (left) we show the compression ratio when we increase the sample size m for several choices of n. As can be seen in the table and the figure, the samples are well compressed and the compression ratio becomes smaller as m increases, which is a desired property for our purpose of large scale machine learning.

Next we examined the computation time of our method and the AdaBoost$*$. Note that we implemented the AdaBoost$*$ in a standard way. More specifically, it receives (Z^+, Z^-), a sparse representation of the sample, and runs in time proportional to its size $|(Z^+, Z^-)|$ per iteration. Note also that the computation time of our method includes the time for constructing NZDDs. In Table 1 (last two columns) we summarize the running time of the Adaboost$*$ and our method for the case $n = 40$, and in Figure 3 (right), we plot their ratio, which is defined as the ratio of the computation time of our method over that of AdaBoost$*$. As can be seen in the table and the figure, our method

Table 1: Summary of the results for $n = 40$

Sample	Input size	NZDD size			Computation time (sec.)	
size m	$\lvert(Z^+, Z^-)\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert G\rvert$	AdaBoost∗	Our method
100000	2000505	157459	257457	594640	12.28	6.78
200000	3998893	312040	512038	996607	30.85	12.50
300000	5997874	462764	762762	1335236	47.70	18.26
400000	7999842	608798	1008796	1638201	64.06	23.68
500000	9998653	750227	1250225	1916730	79.43	29.13
600000	11997933	886971	1486969	2178909	95.70	34.20
700000	13998692	1019425	1719422	2429716	111.65	38.70
800000	15998475	1147999	1947996	2672657	127.30	43.41
900000	17999692	1272759	2172756	2909786	143.45	47.95
1000000	19999611	1394369	2394362	3141135	159.40	52.11

runs much faster than the standard implementation of the AdaBoost∗. Moreover, the behavior of the time ratio roughly reflects that of the compression ratio. On the other hand, as n increases, our method has less advantages over AdaBoost∗ both on time and space. This is probably because we need more instances to make our method effective as n grows.

Note that we can see in Table 1 that the NZDDs G obtained have the property that $|E|$ is much smaller than $|G|$. Since our algorithm simulates the initialization and update of distributions d_t in $O(|E|)$ time (see Lemma 1), the bottoleneck is to compute the edges of all hypotheses, which takes $O(|G|)$ time (see Lemma 2). Improving this part is left for a future work.

6. Conclusions

We proposed the data structure NZDDs, a variant of ZDDs for representing the training data succinctly and gave algorithms that, when given an NZDD representation of the sample, exactly simulate AdaBoost∗ (as well as AdaBoost) in time proportional to the size of the NZDD. This result suggests that some sort of linear programming problems can be efficiently solved over the NZDD-compressed data, rather than meaning that just boosting algorithms are simulated. As future work, the investigation of efficient con-

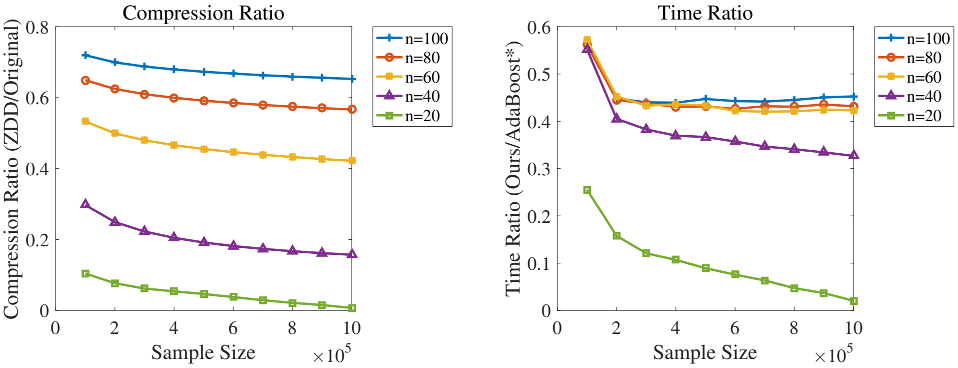

Figure 3: Compression ratio and time ratio for the artificial data. Here, the compression ratio is defined as $|G|/|(Z^+, Z^-)|$. The time ratio is the ratio of the computation time of our method over that of the AdaBoost$*$.

struction of small NZDDs is important. We also need to evaluate empirical performances of our method on real datasets. One of the open problems is to extend our method to the 1-norm *soft margin* maximization, which is very important from the machine learning perspective, because its solution has a guarantee of generalization error even when the sample is not linearly separable. Also, the problem of obtaining similar results for the 2-norm support vector machine remains open.

Acknowledgments

We thank Yasuo Tabei and Daisuke Kimura of RIKEN AIP for insightful discussion and pointing out relevant related work. This work is supported in part by JSPS KAKENHI Grant Number JP16J04621, JP16K00305 and JP15H02667, respectively.

[1] E. Hazan, Introduction to online convex optimization, Now Publishers Inc., 2016.

[2] Y. Lifshits, Processing compressed texts: a tractability border, in: CPM'07 Proceedings of the 18th annual conference on Combinatorial Pattern Matching, 2007, pp. 228–240.

[3] D. Hermelin, G. M. Landau, S. Landau, O. Weimann, A unified algorithm for accelerating edit-distance computation via text-compression,

in: 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), 2009.

[4] K. Goto, H. Bannai, S. Inenaga, M. Takeda, Fast q-gram mining on SLP compressed strings, Journal of Discrete Algorithms 18 (2013) 89–99.

[5] M. Nishino, N. Yasuda, S.-i. Minato, M. Nagata, Accelerating Graph Adjacency Matrix Multiplications with Adjacency Forest, in: Proceedings of the 2014 SIAM International Conference on Data Mining (SDM 14), 2014, pp. 1073–1081. doi:10.1137/1.9781611973440.122.

[6] Y. Tabei, H. Saigo, Y. Yamanishi, S. J. Puglisi, Scalable Partial Least Squares Regression on Grammar-Compressed Data Matrices, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), 2016, pp. 1875–1884. doi:10.1145/2939672.2939864.

[7] S. Minato, Zero-suppressed BDDs for set manipulation in combinatorial problems, in: DAC '93: Proceedings of the 30th international conference on Design automation, 1993.

[8] D. E. Knuth, Art of Computer Programming, Volume 4, Fascicle 1, The: Bitwise Tricks & Techniques; Binary Decision Diagrams, Addison-Wesley, 2009.

[9] S. Minato, T. Uno, H. Arimura, LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2008, pp. 234–246.

[10] S. Minato, T. Uno, Frequentness-transition queries for distinctive pattern mining from time-segmented databases, in: Proceedings of the 10th SIAM International Conference on Data Mining (SDM2010), 2010, pp. 339–349.

[11] T. Inoue, K. Takano, T. Watanabe, J. Kawahara, R. Yoshinaka, A. Kishimoto, K. Tsuda, S.-i. Minato, Y. Hayashi, Distribution Loss Minimization With Guaranteed Error Bound, IEEE Transactions on Smart Grid 5 (1) (2014) 102–111. doi:10.1109/TSG.2013.2288976.
URL http://ieeexplore.ieee.org/document/6693788/

[12] G. Rätsch, M. K. Warmuth, Efficient margin maximizing with boosting, Journal of Machine Learning Research 6 (2005) 2131–2152.

[13] Y. Freund, R. E. Schapire, A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.

[14] O. L. Mangasarian, Arbitrary-norm separating plane, Oper. Res. Lett. 24 (1-2) (1999) 15–23.

[15] G. Rätsch, Robust Boosting via Convex Optimization: Theory and Applications, Ph.D. thesis, University of Potsdam (2001).

[16] T. Jiang, B. Ravikumar, Minimal NFA problems are hard, SIAM Journal on Computing 22 (6) (1993) 1117–1141.

[17] T. Toda, Fast compression of large-scale hypergraphs for solving combinatorial problems, in: Discovery Science - 16th International Conference, DS 2013, Singapore, October 6-9, 2013. Proceedings, 2013, pp. 281–293.

[18] D. Revuz, Minimisation of acyclic deterministic automata in linear time, Theoretical Computer Science 92 (1992) 181–189.

[19] E. Takimoto, M. Warmuth, Path kernels and multiplicative updates, Journal of Machine Learning Research 4 (2003) 773–818.

[20] M. Mohri, General algebraic frameworks and algorithms for shortest-distance problems., Tech. rep., Technical Memorandum 981210-10TM, AT&T Labs- Research, 62 pages (1998).