Proceedings of Machine Learning Research 129:241–256, 2020 ACML 2020

Data-Dependent Conversion to a Compact Integer-Weighted Representation of a Weighted Voting Classifier

Mitsuki Maekawa MAEMITSU@IST.HOKUDAI.AC.JP
Atsuyoshi Nakamura ATSU@IST.HOKUDAI.AC.JP
Mineichi Kudo MINE@IST.HOKUDAI.AC.JP
Hokkaido University, Sapporo, Japan

Editors: Sinno Jialin Pan and Masashi Sugiyama

Abstract

We propose a method of converting a real-weighted voting classifier to a compact integer-weighted voting classifier. Real-weighted voting classifiers like those trained using boosting are very popular and widely used due to their high prediction performance. Real numbers, however, are space-consuming and its floating-point arithmetic is slow compared to integer arithmetic, so compact integer weights are preferable for implementation on devices with small computational resources. Our conversion makes use of given feature vectors and solves an integer linear programming problem that minimizes the sum of integer weights under the constraint of keeping the classification result for the vectors unchanged. According to our experimental results using datasets of UCI Machine Learning Repository, the bit representation sizes are reduced to 5.2-33.4% within 3.7% test accuracy degrade in 7 of 8 datasets for the weighted voting classifiers of decision stumps learned using AdaBoost-SAMME.

Keywords: ensemble classifier, weighted voting, compact representation, integer linear programming

1. Introduction

Ensemble classifiers are popularly used when high classification performance is required. Ensemble classifiers integrate outputs of base classifiers, and a weighted voting is one of the most popular ways of integration. We can obtain a high-performance weighted voting classifier by boosting algorithms like AdaBoost (Freund and Schapire, 1997) from given labeled data.

Due to recent development of machine learning technology, machine learning algorithms and the predictors learned by them have become applied to various areas. In some applications such as edge computing, computational resource in a terminal device is limited and compact representation that enables fast hardware implementation is preferable. Thus, converting a space-consuming weighted voting classifier to compact representation may make the choice of the classifier possible in such application. Compressed representation also has various possibility. For example, a more accurate classifier can be loaded on the same sized memory, a less powerful device can be used for classification, and the number of classification tasks done parallelly or per unit time in a HPC machine can increase.

Real values are generally used as weights in a weighted voting classifier, and a real value requires 32 bit as a float type representation in C language. By converting real weights to small integer weights, weight arithmetic speed can be improved. Conversion from real weights to integer weights is possible by simple quantization, however, those integer weights might not be small under the constraint of keeping classifier accuracy. In this paper, we propose data-dependent conversion of real weights to compact integer weights using integer linear programming (ILP). Given a real-weighted voting classifier and a set of unlabeled data S, our method calculates non-negative integer weights that minimize the sum of them subject to weighted voting classification unchanged for all data in S. Integer linear programming is known to be NP-hard, but approximate solution can be obtained reasonably fast using mixed integer programming solvers such as commercial solvers Gurobi and CPLEX, and open source solvers CBC, SCIP and GLPK.

In our ILP formulation, the number of constraints systematically generated from a weighted voting classifier and a dataset, is $N(K-1)$, where N is the number of data and K is the number of classes. This number becomes large for large N, which may make ILP solvers slow. Many constraints, however, might be unnecessary, that is, they might be implied by other constraints. To efficiently eliminate such unnecessary constraints, we introduce a method of checking set-inclusion relation between sets of voters to the same label.

By using the training data of the given weighted voting classifier as a set of data inputted to our method, a kind of simplification of the weighted voting classifier can be achieved without changing its classification performance for the training data. Then, the converted integer-weighted voting classifier is expected to have generalized classification performance similar to the original real-weighted voting classifier. Since keeping classification results for wrongly predicted training data seems nonsense, we propose a way of applying our method in this situation that uses correctly predicted data only as a given set of unlabeled data S.

Effectiveness of our method is demonstrated through experimental results using several datasets in UCI machine learning repository. A set of real weights learned by AdaBoost-SAMME with 100 decision stumps and its training dataset are inputted to our method. On average, more than half weights become 0 for all the datasets except digits dataset, and 6 bits are enough for representing converted maximum integer weight. Assuming 32 bit representation of real number, bit representation size of an ensemble classifier is reduced to $5.2 \sim 33.4\%$ in 7 of 8 datasets. The generalized (test) classification performance of the classifier with the converted integer weights is almost the same as that of the classifier with the original real weights, and the performance degradation is at most 3.7%. Our proposed constraint reduction method works and makes the running time significantly shorter for some datasets: $60 \sim 96\%$ constraints are removed except for cancer and Parkinson datasets, which induces $44 \sim 89\%$ running time reduction. When a set of training data of given classifiers is inputted as a set of unlabeled data, using correctly predicted training data only was confirmed to be effective in our experiments. Effectiveness of our method was also demonstrated for not only real-weighted voting classifiers learned by boosting, but also simple voting classifiers learned by random forest; its bit representation size dropped to $1.6 \sim 31.9\%$ within 4.9% accuracy degradation.

This paper is organized as follows. Sec. 2 introduces notation about ensemble classifier and formulates the problem that we deal with in this study. Sec. 3 describes the details of our proposed method to handle the problem formulated in Sec. 2. Sec. 4 reports the experimental settings and results for demonstrating effectiveness of our proposed method and Sec. 5 discusses the interpretability and generalization performance of the ensemble classifiers converted by the proposed method based on our experimental results. In Sec 6, we summarize the contributions of our study and describe the perspectives for future work.

Related Work

There are many studies on efficient hardware implementation of ensemble classifiers such as random forests (Van Essen et al., 2012) and AdaBoost (Shi et al., 2008). Memory reduction of AdaBoost weights by converting floating point to fixed point was investigated in (Mitsunari and Yu, 2016), but the conversion was implemented by quantization and no optimization was done. In game theory, the minimum integer weighted binary voting representation has been studied for the weighted game (Freixas and Kurz, 2014). Reduction to integer linear programming problem is studied in the paper but the objective of their study is purely theoretical analysis and no practical use was considered.

Our weight conversion method makes the weights of unnecessary component classifiers be zero, that is, it has an effect of *ensemble pruning*. Ensemble pruning is a very popular study area and a lot of work have been done so far (Martínez-Muñoz et al., 2009; Tsoumakas et al., 2009). The task of ensemble pruning is to reduce the number of component classifiers, and does not address weight size reduction. This is a kind of combinatorial optimization problem with search space of exponential number of candidates. Using measures such as accuracy and diversity, various approaches are taken: top selection with the fixed (Margineantu and Dietterich, 1997) or statistically meaningful (Tsoumakas et al., 2005) number of component classifiers, greedy removing (Banfield et al., 2005), heuristic search by genetic algorithm (Hernández-Lobato et al., 2006), and formulation by quadratic integer programming (Zhang et al., 2006). In the last approach, the best fixed sized subset of component classifiers is searched. Recently, bi-objective formulation that also considers subset size minimization simultaneously has been studied (Qian et al., 2015).

Our proposed method is aimed at reducing the size without performance degradation, which is different from the above pruning studies that aim to improve the generalization performance. In addition to such difference in aim, our method is unique in the points that it does not need (1) class labels and (2) specifying the number of base classifiers to be selected. To our best knowledge, all the existing ensemble pruning methods use class labels or specify the number of base classifiers to be selected; some diversity-based pruning method like Kappa pruning (Margineantu and Dietterich, 1997) does not need class labels but must specify the number of base classifiers to be selected, and error minimization for validation data using genetic algorithm (Hernández-Lobato et al., 2006) does not need to specify the number of base classifiers to be selected but need class labels of validation data.

2. Problem Setting

For some natural numbers m and K, let $\mathbf{f}$ be a list of classification functions $(f_1, f_2, \ldots, f_m)$ from some domain X to $L = \{1, 2, \ldots, K\}$, and let $\mathbf{w}$ be a list of positive real numbers $(w_1, w_2, \ldots, w_m) \in (0, \infty)^m$. A *real-weighted voting classifier* $F[\mathbf{f}, \mathbf{w}]$ from X to L is defined as

$$F[\mathbf{f}, \mathbf{w}](x) = \arg\max_{\ell \in L} \sum_{f_i(x)=\ell} w_i \qquad \text{for } x \in X.$$

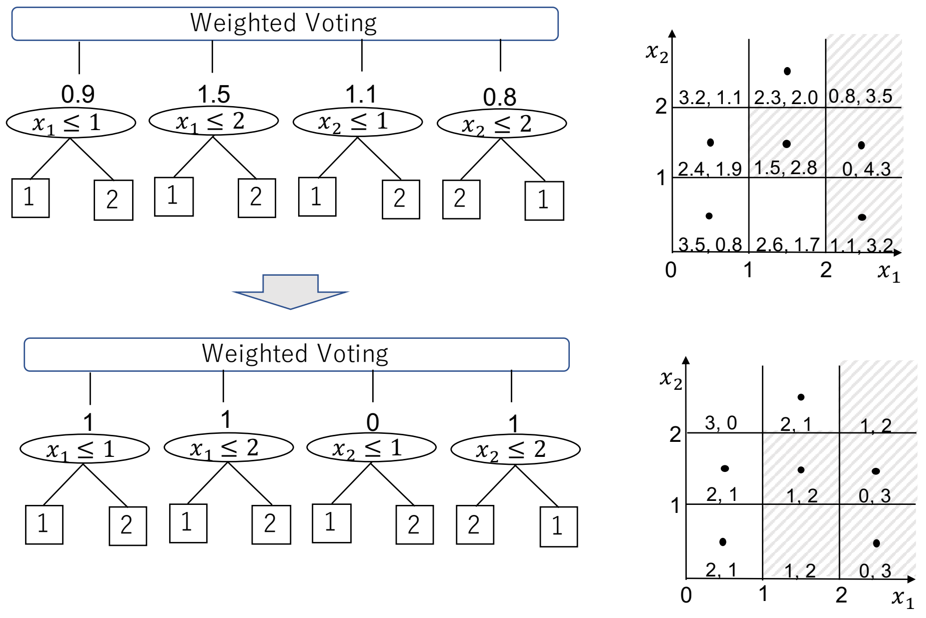

Figure 1: An example of Problem 1 (upper figures) and its solution (lower figures). The given list of classification functions (f_1, f_2, f_3, f_4) is composed of decision stumps from $\mathbb{R}^2$ to $L = \{1, 2\}$ and the given list of weights are $(0.9, 1.5, 1.1, 0.8)$. The given set of points S is $\{((0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 2.5), (2.5, 0.5), (2.5, 1.5)\}$. By the classification rules of the decision stumps, the domain $\mathbb{R}^2$ is partitioned into 9 regions, which are shown in the right figures. The first and second numbers written in each region of the figures are the weighted sums of votes to Class 1 and Class 2, respectively. The region with diagonal lines background is Class 2 region by the given weighted voting classifiers. The solution weighted voting classifier has integer weights and one of them is 0. The classification result for S does not change though the decision boundary changes.

The problem we address in this paper is stated as follows.

Problem 1 *Given a real-weighted voting classifier* $F[\mathbf{f}, \mathbf{w}]$ *and a set of points* $S \subseteq X$*, find a list of small non-negative integers* $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ *that satisfies* $F[\mathbf{f}, \mathbf{z}](x) = F[\mathbf{f}, \mathbf{w}](x)$ *for all* $x \in S$*.*

In the case that $F[\mathbf{f}, \mathbf{w}]$ is an ensemble classifier learned from a training data $T \subseteq X \times L$, we can use $\{x \mid (x, y) \in T\}$ as an input point set S of Problem 1, which enables a kind of simplification of a learned classifier.

Example 1 *An example of Problem 1 and its solution are shown in Fig. 1. In the example, the given weighted voting classifier is composed of four decision stumps from* $\mathbb{R}^2$ *to* $L = \{1, 2\}$ *whose weights are* $0.9, 1.5, 1.1$ *and* 0.8*. The set of six points shown in each right figure are given as* S*. The found small non-negative integer weights are* $1, 1, 0$ *and* 1 *with which* *the weighted voting classifier keeps classification result for S though the decision boundary changes. Since we may ignore* 0*-weighted component classifier, not only each weight size but also the number of component classifiers can be reduced when some of the solution integer weights are* 0 *like this example case.*

3. Proposed Method

In this section, we describe our proposed method for Problem 1.

3.1. Weight Conversion by Integer Linear Programming

In the definition of Problem 1,

$$F[\mathbf{f},\mathbf{z}](x) = F[\mathbf{f},\mathbf{w}](x) \quad \text{for } x \in S$$

must hold for $\mathbf{z}$. This condition is rewritten as

$$\sum_{i:f_i(x)=F[\mathbf{f},\mathbf{w}](x)} z_i > \sum_{i:f_i(x)=\ell} z_i \quad \text{for } \ell \neq F[\mathbf{f},\mathbf{w}](x), x \in S.$$

To make $\mathbf{z}$ be composed of small non-negative integers, we try to minimize $\sum_{i=1}^{m} z_i$. Then, Problem 1 becomes a constrained minimization problem defined as follows.

Problem 2 *Given a real-weighted voting classifier $F[\mathbf{f},\mathbf{w}]$ and a set of points $S \subseteq X$, solve the following integer linear programming (ILP) problem:*

$$\begin{aligned} &\textit{minimize} \sum_{i=1}^{m} z_i \\ &\textit{subject to} \sum_{i:f_i(x)=F[\mathbf{f},\mathbf{w}](x)} z_i > \sum_{i:f_i(x)=\ell} z_i \quad \textit{for } \ell \neq F[\mathbf{f},\mathbf{w}](x), x \in S \textit{ and} \\ &\qquad\qquad z_i \in \mathbb{N} \quad \textit{for } i = 1, 2, \ldots, m, \end{aligned}$$

where $\mathbb{N}$ is the set of non-negative integers.

Note that the formulated constraints are strict inequalities, so the above ILP problem is not a standard form. From the condition of $z_i \in \mathbb{N}$, the constraints of Problem 2 is equivalent to

$$\sum_{i:f_i(x)=F[\mathbf{f},\mathbf{w}](x)} z_i - \sum_{i:f_i(x)=\ell} z_i \geq 1 \quad \text{for } \ell \neq F[\mathbf{f},\mathbf{w}](x), x \in S$$
$$\text{and } z_i \in \mathbb{N} \quad \text{for } i = 1, 2, \ldots, m.$$

Therefore, Problem 2 can be solved as a standard form ILP problem. ILP problem is known to be NP-hard but able to be solved approximately and practically fast using mixed integer programming solvers such as commercial solvers Gurobi and CPLEX, and open source solvers CBC, SCIP and GLPK.

Example 2 *The problem of Example 1 can be converted to the following ILP problem.*

$$\begin{array}{llllllllllllll}
\textit{minimize} & \multicolumn{13}{l}{z_1 + z_2 + z_3 + z_4} \\
\textit{subject to} & z_1 & + & z_2 & + & z_3 & & & > & & & & & z_4 \\
& z_1 & + & z_2 & & & & & > & & & z_3 & + & z_4 \\
& z_1 & & & + & z_3 & + & z_4 & > & & z_2 & & & \\
& & & z_2 & & & + & z_4 & > & z_1 & & + & z_3 & \\
& z_1 & + & z_2 & & & + & z_4 & > & & & z_3 & & \\
& z_1 & + & z_2 & + & z_3 & + & z_4 & > & 0 & & & & \\
& \multicolumn{13}{l}{z_1, z_2, z_3, z_4 \in \mathbb{N}.}
\end{array}$$

3.2. Consideration of Effective Constraints

In the ILP problem defined in Problem 2, the number of constraints is $N(K-1)$ for the number of points $N = |S|$ and the number of classes $K = |L|$, where $|\cdot|$ is the number of elements in set '$\cdot$'. Therefore, Problem 2 seems to become difficult to solve as N increases. Some constraints, however, have no effect on the feasible solution space and can be eliminated without changing the space, so the number of effective constraints may be much smaller than $N(K-1)$ in practice. Reducing the number of constraints is considered to be effective to decrease the time and space complexity of our ILP problem. In this section, we propose an efficient method for deleting unnecessary constraints specific to the ILP problem defined in Problem 2.

Let's consider which constraint is unnecessary. For each $x \in S$ and $\ell \neq F[\mathbf{f}, \mathbf{w}](x)$, constraint

$$\sum_{i:f_i(x)=F[\mathbf{f},\mathbf{w}](x)} z_i > \sum_{i:f_i(x)=\ell} z_i$$

must be satisfied. Define $V_x(\ell)$ and ℓ_x as $V_x(\ell) = \{i \mid f_i(x) = \ell\}$ and $\ell_x = F[\mathbf{f}, \mathbf{w}](x)$, respectively. Then, the above constraint is rewritten as

$$\sum_{i \in V_x(\ell_x)} z_i > \sum_{i \in V_x(\ell)} z_i.$$

This constraint is unnecessary if there exist $x' \in S$ and $\ell' \neq \ell_{x'}$ such that

$$V_x(\ell_x) \supseteq V_{x'}(\ell_{x'}) \text{ and } V_{x'}(\ell') \supseteq V_x(\ell)$$

because this implies

$$\sum_{i \in V_x(\ell_x)} z_i \geq \sum_{i \in V_{x'}(\ell_{x'})} z_i \geq \sum_{i \in V_{x'}(\ell')} z_i \geq \sum_{i \in V_x(\ell)} z_i$$

due to non-negativeness of z_i $(i = 1, \ldots, m)$.

Consider a set pair $(V_x(\ell_x), V_x(\ell))$ for each constraint $\sum_{i \in V_x(\ell_x)} z_i > \sum_{i \in V_x(\ell)} z_i$. Then, from the above consideration, the problem of deleting unnecessary conditions is reduced to the following problem for set pairs $(V_x(\ell_x), V_x(\ell))$ $(x \in S, \ell \neq \ell_x)$.

Algorithm 1 Algorithm for Problem 3

```
Input: $\mathcal{F} = \{(V_i, U_{i,\ell}) \mid V_i, U_{i,\ell} \subseteq \{1,\ldots,m\}, i = 1,\ldots,N, \ell = 1,\ldots,K-1\}$ that satisfies
    $V_i \cap U_{i,\ell} = \emptyset$ and $V_i \cup \bigcup_{\ell=1}^{K-1} U_{i,\ell} = \{1,\ldots,m\}$ for $(V_i, U_{i,1}),\ldots,(V_i, U_{i,K-1}) \in \mathcal{F}$
1: $V_{(1)},\ldots,V_{(N)} \leftarrow$ sorted list of $V_1,\ldots,V_N$ in descending order of #elements.
2: for $i = 1$ to $N$ do
3:   $L \leftarrow \{1,\ldots,K-1\}$
4:   for $i' = N$ to $i+1$ do
5:     if $V_{(i)} \supseteq V_{(i')}$ then
6:       for $\ell \in L$ do
7:         if $\exists \ell'$ s.t. $U_{(i'),\ell'} \supseteq U_{(i),\ell}$ then
8:           $\mathcal{F} \leftarrow \mathcal{F} \setminus \{(V_{(i)}, U_{(i),\ell})\}$, $L \leftarrow L \setminus \{\ell\}$
9:           if $L = \emptyset$ then break for-loop of Lines 4-9
```

Problem 3 *Given a family of set pairs* $\mathcal{F} = \{(V_i, U_{i,\ell}) \mid V_i, U_{i,\ell} \subseteq \{1,\ldots,m\}, i = 1,\ldots,N, \ell = 1,\ldots,K-1\}$ *that satisfies* $V_i \cap U_{i,\ell} = \emptyset$ *and* $V_i \cup \bigcup_{\ell=1}^{K-1} U_{i,\ell} = \{1,\ldots,m\}$ *for* $(V_i, U_{i,1}),\ldots,(V_i, U_{i,K-1}) \in \mathcal{F}$*, remove all the pairs* $(V_i, U_{i,\ell})$ *from* $\mathcal{F}$ *that have* $i' \neq i$ *and* $\ell' \in \{1,\ldots,K-1\}$ *such that* $V_i \supseteq V_{i'}$ *and* $U_{i',\ell'} \supseteq U_{i,\ell}$*.*

To solve Problem 3, $V_i \supseteq V_{i'}$ is checked for every $(i,i') \in \{1,\ldots,N\}^2$ with $i \neq i'$, so the number of checked (i,i') is $N(N-1) = O(N^2)$. For every (i,i') satisfying $V_i \supseteq V_{i'}$, $U_{i',\ell'} \supseteq U_{i,\ell}$ is checked for every $(\ell,\ell') \in \{1,\ldots,K-1\}^2$, so at most $(K-1)^2$ pairs are checked. Therefore, at most $O((K-1)^2N^2 + N^2)$ set inclusion checks are necessary. Each set inclusion check can be done by $O(m)$ time, thus Problem 3 can be solved in $O(mK^2N^2)$ time. Since the relation $A \supseteq B$ holds only when the number of elements in A is larger than that of B, we only have to check pairs (A,B) that satisfy $|A| \geq |B|$ for the checks of $A \supseteq B$. Furthermore, the larger the difference $|A|-|B|$ is, the larger the chance that $A \supseteq B$ holds becomes. A simple algorithm taking account of this fact is Algorithm 1, which sorts[1] $\{V_i \mid i = 1,\ldots,N\}$ by the number of elements, and then checks $V_i \supseteq V_{i'}$ for pairs (i,i') with $|V_i| \geq |V_{i'}|$ only. Checking $V_i \supseteq V_{i'}$ in the order of decreasing in $|V_i|$ and increasing in $|V_{i'}|$ can increase the chance of fast finding of elements to remove, which results in reduction of the number of set inclusion checks.

Example 3 *In the problem of Example 1, set* $\{(V_x(\ell_x), V_x(\ell)) \mid x \in S\}$ *is*

$$\{(\{1,2,3\},\{4\}),(\{1,2\},\{3,4\}),(\{1,3,4\},\{2\}),(\{2,4\},\{1,3\}),(\{1,2,4\},\{3\}),(\{1,2,3,4\},\emptyset)\}.$$

Then the answer of Problem 3 is $\{(\{1,2\},\{3,4\}),(\{1,3,4\},\{2\}),(\{2,4\},\{1,3\})\}$*. Thus, the ILP problem of Example 2 can be reduced to*

$$\begin{array}{lrcl}
\textit{minimize} & z_1+z_2+z_3+z_4 & & \\
\textit{subject to} & z_1+z_2 & > & z_3+z_4 \\
 & z_1+z_3+z_4 & > & z_2 \\
 & z_2+z_4 & > & z_1+z_3 \\
 & z_1,z_2,z_3,z_4 \in \mathbb{N}. & &
\end{array}$$

1. Note that time complexity of sorting ($O(N \log N)$) is smaller than the number of set inclusion checks (($O(N^2)$).

Table 1: UCI Machine Learning Repository datasets (Dua and Karra Taniskidou, 2017) used in our experiments

dataset	#data	#feature	#class	dataset name
Blood	748	4	2	Blood Transfusion Service Center (Yeh et al., 2009)
cancer	569	30	2	Breast Cancer Wisconsin (Diagnostic)
MAGIC	19020	10	2	MAGIC Gamma Telescope
MUSK	476	166	2	Musk (Version 1)
Parkinson	756	753	2	Parkinson's Disease Classification (Sakar et al., 2019)
spam	4601	57	2	Spambase
iris	150	4	3	Iris
digits	1797	64	10	Optical Recognition of Handwritten Digits (Testing)

3.3. Simple Exploitation of Class Labels

Our formalization of the problem does not need class labels of points in a given set S, but the most natural candidate of set S for a real-weighted voting classifier F is the training dataset T used to learn F. Points in training dataset T have class labels, so we can make use of them. The simplest way to exploit class labels is to remove points wrongly-predicted by F from $\{x \mid (x, y) \in T\}$, that is, to use S defined as $S = \{x \mid (x, y) \in T, F(x) = y\}$. In our problem formulation, the classification results of F need to be kept, but its incorrect classifications do not have to be kept. are unnecessary. Using the correctly-predicted points only, prediction performance can be also expected to be improved.

4. Experiments

In this section, we check effectiveness of our method by applying it to voting classifiers learned from several real-world datasets in UCI Machine Learning Repository (Dua and Karra Taniskidou, 2017). We empirically investigate effectiveness of the followings: (E1) application to real-weighted voting classifiers learned by AdaBoost, (E2) the proposed constraint reduction algorithm, (E3) using correctly predicted training data only, and (E4) application to simple voting classifiers learned by a random forest learning algorithm.

4.1. Experimental Settings

Datasets used in our experiments are shown in Table 1. Note that the number of classes is more than 2 for 2 datasets, Iris and digits datasets. Performance was evaluated by 5-fold cross validation keeping class ratio. For each dataset, training set $T \subseteq X \times L$ was inputted to an ensemble learning algorithm to create a real-weighted voting classifier $F[\mathbf{f}, \mathbf{w}]$ with a list of 100 component classifiers $\mathbf{f}$. Then, for $F[\mathbf{f}, \mathbf{w}]$ and $S = \{x \in X \mid (x, \ell) \in T\}$, the ILP problem (Problem 2) was solved to obtain a list of integer weights $\mathbf{z}$ by using PULP_CBC_CMD solver[2]. Finally, performance for $F[\mathbf{f}, \mathbf{z}]$ was evaluated using the test set.

2. https://pythonhosted.org/PuLP/solvers.html

We use AdaBoost-SAMME (Zhu et al., 2009) as a learning algorithm for real-weighted voting classifiers with decision stump component classifiers in Experiments E1-3, and use random forest as that for simple voting classifiers in Experiment E4. AdaBoost-SAMME and random forest that we ran in our experiments are sklearn.ensemble.AdaBoostClassifier and sklearn.ensemble.RandomForestClassifier of scikit-learn machine learning library[3].

As a baseline comparison method, we adopt simple quantization that converts positive real weights to non-negative integers by simply partitioning the range between the minimum and maximum values into the intervals of equal length. Thus, in conversion from $\mathbf{w}$ to $\mathbf{z}$ of r bit non-negative integers, w_i is converted to $z_i = \max\{0, \lceil (w_i - \min_j w_j)/((\max_j w_j - \min_j w_j)/2^r)\rceil - 1)$ by simple quantization.

Compactness of weights $\mathbf{v}$ ($\mathbf{v} = \mathbf{w}, \mathbf{z}$) is measured by their total bit size, which is denoted by wsize($\mathbf{v}$). Then, effectiveness of conversion on weight compactness is evaluated by *bit size ratio of weights* $\mathbf{z}$ *to* $\mathbf{w}$ which is defined by wsize($\mathbf{z}$)/wsize($\mathbf{w}$). The total bit size of $\mathbf{z}$ is calculated as $\mathrm{wsize}(\mathbf{z}) = \lceil \log_2(\max_i z_i + 1)\rceil \times |\{z_i \mid z_i > 0\}|$. The bit size of a real number is assumed to be 32 bits, so the total bit size of real weights $\mathbf{w}$ of 100 component classifiers is calculated as $\mathrm{wsize}(\mathbf{w}) = 32 \times 100 = 3200$. Note that bit size of weights for a simple voting classifier is calculated as 0 though all the weights can be regarded as 1. We also evaluate compactness of whole ensemble classifier $F[\mathbf{f}, \mathbf{v}]$ ($\mathbf{v} = \mathbf{w}, \mathbf{z}$) by its total bit size, which is the sum of wsize($\mathbf{v}$) and the size of component classifiers $\mathbf{f}$, denoted by fsize($\mathbf{f}, \mathbf{v}$). In the case that component classifiers are decision trees, let nodes($\mathbf{f}, \mathbf{v}$) denote the sum of the number of nodes in all the component decision trees f_i with $v_i > 0$. Then, the number of bits needed for one node of component decision trees $\mathbf{f}$ with weights $\mathbf{v}$ is assumed to be the sum of feature id size $\lceil \log_2 d\rceil$ for d dimensional features, threshold size (32 bits) of a real number, the size of pointers to two child nodes $2\lceil \log_2 \mathrm{nodes}(\mathbf{f}, \mathbf{v})\rceil$ and class label size $\lceil \log_2 K\rceil$ needed for a leaf node. Therefore, the size of component classifiers $\mathbf{f}$ with weights $\mathbf{v}$ is calculated as $\mathrm{fsize}(\mathbf{f}, \mathbf{v}) = \mathrm{nodes}(\mathbf{f}, \mathbf{v}) \times (\lceil \log_2 d\rceil + 32 + 2\lceil \log_2 \mathrm{nodes}(\mathbf{f}, \mathbf{v})\rceil + \lceil \log_2 K\rceil)$. *Bit size ratio of classifier* $F(\mathbf{f}, \mathbf{z})$ *to* $F(\mathbf{f}, \mathbf{w})$ is defined by $(\mathrm{wsize}(\mathbf{z}) + \mathrm{fsize}(\mathbf{f}, \mathbf{z}))/(\mathrm{wsize}(\mathbf{w}) + \mathrm{fsize}(\mathbf{f}, \mathbf{w}))$.

All the experiments were done using iMac (27-inch, Late 2013) with macOS Catalina10.15.5, which has 3.2 GHz Intel Core i5 CPU and 32 GB 1600 MHz DDR3 memory.

4.2. Results

4.2.1. (E1) Effectiveness for Voting Classifiers Learned by AdaBoost

Table 2 shows Weight compactness and prediction accuracy for integer-weighted voting classifiers obtained by our method from real-weighted voting classifiers learned by AdaBoost.

The averaged number of nonzero weights becomes less than half except for digits datasets. The averaged maximum integer weights are at most 57.4, which can be represented by 6 bits. Assuming that a real weight is represented by 32 bits, the bit representation size is reduced to 3/16. By converting $\mathbf{w}$ to $\mathbf{z}$, bit size ratio of weights is reduced to $0.66 \sim 7.76\%$, and bit size ratio of a classifier is reduced to $5.23 \sim 33.41\%$ except digits dataset. Reducing the number of nonzero weights is more effective to reduce the total memory size needed for the ensemble classifier, and in digits dataset, more than 86% weights remain nonzero after conversion, which results in large bit size ratio of the classifier. Our conversion keeps

3. https://scikit-learn.org/stable/index.html

Table 2: Size and prediction performance for integer weight list $\mathbf{z}$ obtained from a $\mathbf{w}$-weighted voting classifier with 100 decision stumps learned by AdaBoost-SAMME. The results are averaged over 5 runs in 5-fold cross validation.

dataset	compactness				accuracy			
	#non-zero	$\max z_i$	bit size ratio (%) $\mathbf{z}$	bit size ratio (%) $F(\mathbf{f}, \mathbf{z})$	training	test $\mathbf{w}$①	test $\mathbf{z}$②	test ②/①
Blood	10.6	2.6	0.6625	7.6031	0.8095	0.7861	0.7861	1.000
cancer	14.8	1.4	0.9250	11.2480	1.0000	0.9574	0.9543	0.978
MAGIC	41.4	57.4	7.7625	33.4142	0.8409	0.8364	0.8365	1.000
MUSK	27.0	15.4	4.2188	21.9617	0.9485	0.8509	0.8424	0.990
Parkinson	40.04	2.2	2.5250	32.2182	1.0000	0.8837	0.8506	0.963
spam	35.0	13.8	4.3750	28.1034	0.9423	0.9341	0.9333	0.999
iris	7.2	3.4	0.6750	5.2330	0.9950	0.9467	0.9400	0.993
digits	86.4	15.8	13.50	75.3962	0.8304	0.8051	0.8013	0.995

accuracy for training datasets and you can see that test accuracy degrade is also within 3.7% from the table.

Comparison with the weights obtained by simple quantization is shown in Table 3. The number of bits r for simple quantization is set to $\lceil \max_i z_i + 1 \rceil$ for $\mathbf{z}$ obtained by proposed method so as to have the same number of bits per weight in calculation of bit size ratio. The proposed method reduces the number of nonzero weights more than quantization, as a result, it has smaller bit size ratio in 6 among 8 datasets. Accuracy of proposed method is also higher than quantization in 6 among the 8 datasets.

4.2.2. (E2) Effectiveness of Constraint Reduction

The results for the experiments on the effect by constraint reduction using[4] Algorithm 1 is shown in Table 4. Deleting unnecessary constraints reduces the number of constraints by $37 \sim 97\%$ in our datasets except Parkinson dataset. The total running time is also improved, and the improvements are significant for the datasets with large constraint reduction.

There is a tendency that, the smaller the number of features is, the larger the number of removed constraints is. The reason of the tendency is explained as follows. In this experiment, base classifiers are decision stumps $f[i, a, \ell]$, which classifies point x into class ℓ if and only if the value of the ith feature x_i is less than a, that is, $x_i < a$. Let $\mathcal{F}_{i,\ell} = \{f[i, a, \ell] \mid f[i, a, \ell] \text{ is a base classifier}\}$. Since $f[i, a, \ell](x) = \ell$ implies $f[i, b, \ell](x) = \ell$ if $a \leq b$, $\{f \in F_{i,\ell} \mid f(x) = \ell\} \supseteq \{f \in F_{i,\ell} \mid f(x') = \ell\}$ for all $x, x' \in S$ with $x_i \leq x'_i$. Thus, the possibility that set inclusion $V_x(\ell) \supseteq V_{x'}(\ell)$ holds, increases if, the number of base decision stumps with branching conditions of the same feature, increases, where $V_x(\ell) = \{i \mid f_i(x) = \ell\}$ defined in Sec. 3.2. The number of features in Parkinson dataset is significantly large

4. In our current implementation, $V_i \supseteq V_{i'}$ is checked in the order of decreasing in $|V_{i'}|$.

Table 3: Compactness and accuracy comparison of weights $\mathbf{z}$ obtained by proposed and quantization methods.

dataset	method	#nonzero	$\max z_i$	bit size ratio of $\mathbf{z}$ (%)	accuracy
Blood	proposed	10.6	2.6	0.6625	0.7861
	quantization	5.6	4.6	0.525	0.7607
cancer	proposed	14.8	1.4	0.9250	0.9543
	quantization	13.2	2.6	0.825	0.9366
MAGIC	proposed	41.4	57.4	7.7625	0.8365
	quantization	68.4	75.8	14.9625	0.8277
MUSK	proposed	27.0	15.4	4.2188	0.8424
	quantization	71.6	23.0	11.1875	0.8572
Parkinson	proposed	40.4	2.2	2.5250	0.8506
	quantization	53.4	5.4	5.0063	0.8344
spam	proposed	35.0	13.8	4.3750	0.9333
	quantization	70.6	24.6	11.0313	0.9324
iris	proposed	7.2	3.4	0.6750	0.9400
	quantization	57.8	4.6	2.8688	0.9800
digits	proposed	86.4	15.8	13.5000	0.8013
	quantization	98.6	27.8	15.4063	0.7677

(753), which causes the failure of constraint reduction. In fact, the number of distinct branching condition features in the ensemble of 100 decision stumps is 66.8 on average.

Our constraint reduction does not change the feasible solution space of our ILP problem, but ILP solver's approximate solution may change slightly. In fact, we observed slightly different ensemble classifiers whose accuracy for test datasets, the number of non-zero weights and maximum weight are almost similar.

4.2.3. (E3) Effectiveness of Using Correctly Predicted Training Data only

Results on effect by using correctly-predicted training data only is shown in Table 5. In the table, the results for 3 datasets are not shown because all or almost all the training data are predicted correctly for them. Compared to the results using all the training data, the number of nonzero weights decreases by $2 \sim 14\%$, $\max z_i$ decreases by $0 \sim 39\%$, as a result, bit size ratio decreases by $2 \sim 31\%$. Accuracy increases by $0 \sim 2\%$, and running time is reduced by $2 \sim 59\%$ except MAGIC dataset. Note that accuracy exceeds that of the original real-weighted ensemble classifiers for 4 among the 5 datasets, and 2% improvement

Table 4: Experimental results on effect by constraint reduction. Columns of "w/o CR" and "w/ CR" show the results without and with constraint reduction, respectively. The time column of "w/ CR" shows the total running times in addition to constraint reduction times first and ILP solving times second in the parentheses.

dataset	#constraint			time (sec)		
	w/o CR^{①}	w/ CR^{②}	②/①	w/o CR^{③}	w/ CR^{④} (CR,ILP)	④/③
Blood	598.4	16.6	0.0277	0.2535	0.0275(0.0088, 0.0186)	0.1085
cancer	455.2	286.2	0.6287	0.2507	0.1958(0.0278, 0.1680)	0.7814
MAGIC	15216	546.2	0.0359	9.5337	2.8492(2.5382, 0.3110)	0.2989
MUSK	380.8	152.0	0.3992	0.1984	0.1109(0.0123, 0.0986)	0.5590
Parkinson	604.8	590.4	0.9762	17.7434	14.2195(0.0469,14.1726)	0.8014
spam	3680.8	490.0	0.1332	3.2347	0.7086(0.4433, 0.2653)	0.2191
iris	240.0	28.6	0.1192	0.0927	0.0222(0.0018, 0.0204)	0.23.95
digits	12938.4	3343.2	0.2584	133.8033	31.2425(0.4626,30.7799)	0.2335

Table 5: Size and prediction performance by the converted integer weighted voting classifier using correctly predicted training data only. Numbers in the parentheses are ratio compared to those using all training data. The percentage bit size ratio of $\mathbf{z}$ to $\mathbf{w}$ for each dataset is shown in the column of 'bit size ratio'.

dataset	tr err	#nonzero	$\max z_i$	bit size ratio	accuracy	time (sec)
Blood	0.1905	10.4(0.98)	2.6(1.00)	0.6500(0.98)	0.7874(1.00)	0.2089(0.82)
MAGIC	0.1591	40.6(0.98)	37.8(0.66)	7.6125(0.98)	0.8370(1.00)	9.8724(1.04)
MUSK	0.0515	25.6(0.95)	10.4(0.68)	3.2000(0.76)	0.8446(1.00)	0.1935(0.98)
spam	0.0577	32.4(0.93)	9.2(0.67)	4.0500(0.93)	0.9350(1.00)	1.9244(0.59)
digits	0.1696	74.4(0.86)	9.6(0.61)	9.3000(0.69)	0.8286(1.02)	55.1099(0.41)

was observed for digits dataset. Totally, we can conclude that using correctly predicted training data only is effective to some extent.

4.2.4. (E4) Effectiveness for Voting Classifiers Learned by Random Forest

Simple voting scheme is adopted in bagging-typed ensemble learners, and random forest is one of most popular bagging-typed ensemble learners. In this experiment, we check effectiveness of our weight conversion method in application to such simple voting classifiers. In Table 6, our experimental result on weight compactness and prediction performance are shown for integer-weighted voting classifiers converted from simple voting classifiers learned by random forest. The number of nonzero weights is reduced to $1/100 \sim$ about $1/4$, which leads to small bit size ratio ($1.6 \sim 31.9\%$) of a classifier. Accuracy degradation is at most 4.9%. Bit size ratio of a classifier is smaller than that for AdaBoost, but the bit size of a random forest is still larger than that of AdaBoost even after weight conversion because

Table 6: Size and prediction performance for integer weight list $\mathbf{z}$ obtained from a random forest classifier with 100 decision trees.

dataset	compactness			accuracy			
	#non-zero	$\max z_i$	bit size ratio of $F(\mathbf{f}, \mathbf{z})$ (%)	training	test		
					$\mathbf{w}^{①}$	$\mathbf{z}^{②}$	②/①
Blood	9.6	1.2	7.7369	0.9375	0.7313	0.7259	0.993
cancer	3.2	1.2	2.8008	1.0000	0.9631	0.9508	0.987
MAGIC	27.6	2.0	31.8809	1.0000	0.8789	0.8741	0.995
MUSK	5.0	1.2	5.3915	1.0000	0.9014	0.8570	0.951
Parkinson	7.0	1.4	6.2532	1.0000	0.8730	0.8373	0.959
spam	9.8	1.0	10.2264	0.9995	0.9526	0.9439	0.991
iris	1.0	1.0	1.6272	1.0000	0.9467	0.9467	1.000
digits	7.6	1.2	10.0264	1.0000	0.9772	0.9343	0.956

a component classifier of a random forest is a decision tree while that of AdaBoost in our experiment is a decision stump.

5. Discussion

5.1. Interpretability

In ensemble classifier, simple component classifiers such as decision stumps are often used, and most such simple classifiers are easy to interpret. However, even though interpretability of component classifiers are high, the total decision rule of the ensemble classifier is not easy to interpret due to large number of component classifiers and their real-weighted voting. Since our method reduces the number of component classifiers and converts their real weights to integer weights, its total decision rule becomes more interpretable.

Figure 2 shows one of the converted integer-weighted voting classifier and its decision boundary for iris dataset. This ensemble classifier is very simple; composed of four decision stumps only and refers to two features only. Integer weights are also very simple; three 1 and one 2. As a result, the total decision rule of this ensemble classifier looks easy to understand.

5.2. Margin Analysis of Original and Converted Ensemble Classifier

By our method, a real-weighted voting classifier can be simplified to an integer-weighted voting classifier with compact integer weights. Compact representation itself has merits such as enabling implementation on devices with small computational resource. In the past research, the main purpose of simplification for classifiers learned from a training data is, however, to improve generalization performance by correcting the overfitting. According to the experimental results in Sec. 4, our conversion keeps prediction accuracy to some extent but it is decreased slightly when all the training data are used as a given set of points S.

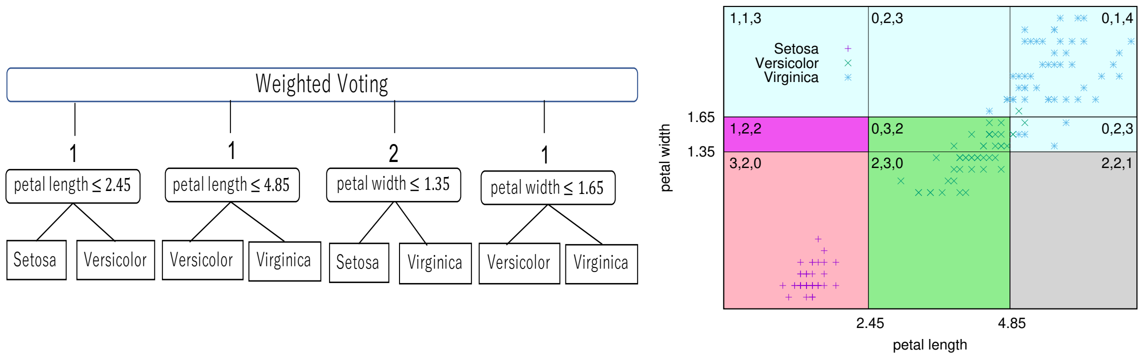

Figure 2: Example of the converted integer-weighted voting classifier (left) and its decision boundary (right) for iris dataset. The classifier is composed of four decision stumps. A split condition and a class label are assigned to each root and leaf node of each decision stump, respectively. In the right figure, $\mathbb{R}^2$ is partitioned into 9 regions by the decision boundaries of the decision stumps in the obtained voting classifier. The weighted sums of votes to Setosa, Versicolor and Virginica are shown in each region. The color of each region indicates the predicted class label by the obtained integer-weighted voting: pink – Setosa, green – Versicolor, cyan – Virginica, gray – Setosa or Versicolor, magenta – Versicolor or Virginica.

To further analyze the causes of prediction accuracy decrease by converted weights, we introduce *margin* which is used for evaluating the generalization performance of ensemble classifiers. The margin of a data point is defined as the distance between decision boundary and the data point. In multi-class ensemble classifier $F[\mathbf{f}, \mathbf{w}]$, the margin definition of some data point x with label y is formulated as follows (Breiman, 1999).

$$\mathrm{margin}(x) = \frac{\sum_{i:f_i(x)=y} w_i - \max_{\ell \neq y} \left(\sum_{i:f_i(x)=\ell} w_i \right)}{\sum_{i=1}^{m} w_i}$$

To evaluate the generalization performance of trained classifier, we calculated the minimum and mean margins among all training data points.

Table 7 shows the margins of weighted voting classifiers with original real weights $\mathbf{w}$ and those with the two lists of integer weights, weights obtained using all the training data $\mathbf{z}$(E1) and weights obtained using only correctly predicted training data $\mathbf{z}$(E3), as a given set of points S. From the result, we can know that our conversion to integer weights tends to increase mean margins but decrease minimum margins.

Mean margin increase seems good for generalization performance improvement, and minimum margin decrease seems not serious because minimum margin is easy to be affected by noise. Further investigation is necessary to analyze the reason why our simplification cannot improve generalization performance for the datasets.

Table 7: Margin of the ensemble classifier

dataset	minimum margin			mean margin		
	$\mathbf{w}$(original)	$\mathbf{z}$(E1)	$\mathbf{z}$(E3)	$\mathbf{w}$(original)	$\mathbf{z}$(E2)	$\mathbf{z}$(E3)
Blood	-0.4484	-0.6389	-0.6148	0.1710	0.2030	0.2149
cancer	0.1081	0.0639	–	0.3692	0.4074	–
MAGIC	-0.4515	-0.4760	-0.4825	0.1430	0.1505	0.1513
MUSK	-0.0634	-0.2352	-0.1223	0.1186	0.1494	0.1416
Parkinson	0.0149	0.0218	–	0.1797	0.1831	–
spam	-0.2687	-0.3462	-0.3561	0.2026	0.2354	0.2394
iris	0.0065	-0.0613	–	0.1306	0.2063	–
digits	-0.0573	-0.0580	-0.0747	0.0238	0.0235	0.0263

6. Conclusion

Our conversion method enables not only significant space reduction for weights but also sparse representation, and totally at least 66.5% bit size reduction has been observed for AdaBoost with decision stump base classifiers in 7 of 8 datasets of the UCI Machine Learning Repository. Given training data with a voting classifier trained from them, we obtained a simplified classifier with almost the same classification performance without using the label information of the training data. Simple exploitation of class labels is experimentally shown to be effective for improvement of generalized classification performance. We would like to develop more sophisticated way of exploiting them in the future.

Acknowledgments

We would like to thank Prof. Hiroki Arimura of Hokkaido University and anonymous reviewers for helpful comments to improve this research. This work was supported by JST CREST Grant Number JPMJCR18K3, Japan.

References

Robert E. Banfield, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1): 49–62, 2005.

Leo Breiman. Prediction games and arcing algorithms. *Neural Comput.*, 11(7):1493–1517, October 1999. ISSN 0899-7667.

Dheeru Dua and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Josep Freixas and Sascha Kurz. On minimum integer representations of weighted games. *Mathematical Social Sciences*, 67:9 – 22, 2014. ISSN 0165-4896.

Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

Daniel Hernández-Lobato, José Miguel Hernández-Lobato, Rubén Ruiz-Torrubiano, and Ángel Valle. Pruning adaptive boosting ensembles by means of a genetic algorithm. In *Intelligent Data Engineering and Automated Learning*, pages 322–329, 2006.

Dragos Margineantu and Thomas Dietterich. Pruning adaptive boosting. In *Proceedings of the Fourteenth International Conference on Machine Learning*, page 211–218, 1997.

Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(2):245–259, 2009.

Koichi Mitsunari and Jaehoon Yu. Influence of numerical precision on machine learning and embedded systems. In *International Workshop on Smart Info-Media Systems in Asia*, page 164–169, 2016.

Chao Qian, Yang Yu, and Zhi-Hua Zhou. Pareto ensemble pruning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, page 2935–2941, 2015.

C. Okan Sakar, Gorkem Serbes, Aysegul Gunduz, Hunkar C. Tunc, Hatice Nizam, Betul Erdogdu Sakar, Melih Tutuncu, Tarkan Aydin, M. Erdem Isenkul, and Hulya Apaydin. A comparative analysis of speech signal processing algorithms for parkinson's disease classification and the use of the tunable q-factor wavelet transform. *Applied Soft Computing*, 74:255–263, 2019. ISSN 1568-4946.

Yuehua Shi, Feng Zhao, and Zhong Zhang. Hardware implementation of adaboost algorithm and verification. In *International Conference on Advanced Information Networking and Applications Workshops*, 2008.

Grigorios Tsoumakas, Lefteris Angelis, and Ioannis Vlahavas. Selective fusion of heterogeneous classifiers. *Intelligent Data Analysis*, 9(6):511–525, 2005.

Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. *An Ensemble Pruning Primer*, pages 1–13. Springer Berlin Heidelberg, 2009.

Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239, 2012.

I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. Knowledge discovery on rfm model using bernoulli sequence. *Expert Syst. Appl.*, 36(3):5866–5871, 2009.

Yi Zhang, Samuel Burer, and W. Nick Street. Ensemble pruning via semi-definite programming. *J. Mach. Learn. Res.*, 7:1315–1338, 2006.

Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and Its Interface*, 2(3):349 – 360, 2009.