Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

A Simple Cost-sensitive Multiclass Classification Algorithm Using One-versus-one Comparisons

Hsuan-Tien Lin

Abstract Many real-world applications require varying costs for different types of mis-classification errors. Such a cost-sensitive classification setup can be very different from the regular classification one, especially in the multiclass case. Thus, traditional meta-algorithms for regular multiclass classification, such as the popular one-versus-one approach, may not always work well under the cost-sensitive classification setup. In this paper, we extend the one-versus-one approach to the field of cost-sensitive classification. The extension is derived using a rigorous mathematical tool called the cost-transformation technique, and takes the original one-versus-one as a special case. Experimental results demonstrate that the proposed approach can achieve better performance in many cost-sensitive classification scenarios when compared with the original one-versus-one as well as existing cost-sensitive classification algorithms.

Keywords cost-sensitive classification, one-versus-one, meta-learning

1 Introduction

Many real-world applications of machine learning and data mining require evaluating the learned system with different costs for different types of mis-classification errors. For instance, a false-negative prediction for a spam classification system only takes the user an extra second to delete the email, while a false-positive prediction can mean a huge loss when the email actually carries important information. When recommending movies to a subscriber with preference "romance over action over horror", the cost of mis-predicting a romance movie as a horror one should be significantly higher than the cost of mis-predicting the movie as an action one. Such a need is also shared by applications like targeted marketing, information retrieval, medical decision making, object recognition and intrusion detection (Abe et al, 2004), and can be formalized as the *cost-sensitive classification* setup. In fact, cost-sensitive classification can be used to express any finite-choice and bounded-loss supervised learning setups (Beygelzimer et al, 2005). Thus, it has been attracting much research attention in recent years (Domingos, 1999; Margineantu, 2001; Abe et al, 2004; Beygelzimer et al, 2005; Langford and Beygelzimer, 2005; Beygelzimer et al, 2007).

H.-T. Lin
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.
E-mail: htlin@csie.ntu.edu.tw

Abe et al (2004) grouped existing research on cost-sensitive classification into three categories: making a particular classifier cost-sensitive, making the prediction procedure cost-sensitive, and making the training procedure cost-sensitive. The third category contains mostly meta-algorithms that reweight training examples before feeding them into the underlying learning algorithm. Such a meta-algorithm can be used to make any existing algorithm cost-sensitive. While a promising meta-algorithm exists and is well-understood for cost-sensitive binary classification (Zadrozny et al, 2003), the counterpart for multiclass classification remains an ongoing research issue (Abe et al, 2004; Langford and Beygelzimer, 2005; Zhou and Liu, 2006).

In this paper, we propose a general meta-algorithm that reduces cost-sensitive multiclass classification tasks to regular classification ones. The meta-algorithm is based on the *cost-transformation* technique, which converts one cost to another by not only reweighting the original training examples, but also *relabeling* them. We show that any cost can be transformed to the regular mis-classification one with the cost-transformation technique. As a consequence, general cost-sensitive classification and general regular classification tasks are equivalent in terms of hardness.

We further couple the meta-algorithm with another popular meta-algorithm in regular classification—the one-versus-one (OVO) decomposition from multiclass to binary. The resulting algorithm, which is called cost-sensitive one-versus-one (CSOVO), can perform cost-sensitive multiclass classification with any base binary classifier. Interestingly, CSOVO is algorithmically similar to an existing meta-algorithm for cost-sensitive classification: weighted all-pairs (WAP; Beygelzimer et al, 2005). Nevertheless, CSOVO is not only simpler but also more efficient than WAP. Our experimental results on real-world data sets demonstrate that CSOVO shares a similar performance over WAP, while both of them can be significantly better than OVO. Therefore, CSOVO is a preferable OVO-type cost-sensitive classification algorithm. Moreover, when compared with other meta-algorithms that reduce cost-sensitive classification to binary classification—namely, one-versus-all (Lin, 2008), error-correcting output code (Langford and Beygelzimer, 2005), tree (Beygelzimer et al, 2005), filter tree and all-pair filter tree (Beygelzimer et al, 2007) decompositions—we see that CSOVO can often achieve the best test performance. Those results further validate the usefulness of CSOVO.

The paper is organized as follows. In Section 2, we formalize the cost-sensitive classification setup. Then, we present the cost-transformation technique with its theoretical implications in Section 3, and derive our proposed CSOVO algorithm in Section 4. Finally, we compare CSOVO with other algorithms empirically in Section 5 and conclude in Section 6.

2 Problem Setup

We start by defining the setup that will be used in this paper.

Definition 1 (weighted classification) Assume that there is an unknown distribution $\mathcal{D}_w$ on $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^+$. where the input space $\mathcal{X} \subseteq \mathbb{R}^D$ and the label space $\mathcal{Y} = \{1, 2, \cdots, K\}$. A weighted example is a tuple $(\mathbf{x}, y, w) \in \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^+$, where the non-negative numbers $w \in \mathbb{R}^+$ are called the weights. In the weighted classification setup, we are given a set of i.i.d. weighted training examples $\mathcal{S}_w = \{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N \sim \mathcal{D}_w^N$. Use

$$E(g, \mathcal{D}) \equiv \mathop{\mathcal{E}}_{(\mathbf{x},y,w)\sim\mathcal{D}} \Big(w \cdot [\![y \neq g(\mathbf{x})]\!] \Big)$$

to denote the expected weighted classification error of any classifier $g\colon \mathcal{X} \to \mathcal{Y}$ with respect to some distribution $\mathcal{D}$. The goal of the weighted classification is to use $\mathcal{S}_w$ to find a classifier $\hat{g}$ such that $E(\hat{g}, \mathcal{D}_w)$ is small.

For $K = 2$, the setup is called (weighted) *binary classification*; for $K > 2$, the setup is called (weighted) *multiclass classification*. When the weights are constants (say, 1), weighted classification becomes a special case called *regular classification*, which has been widely and deeply studied for years (Beygelzimer et al, 2005). In general, weighted classification can be easily reduced to regular classification for both binary and multiclass cases using the famous COSTING reduction (Zadrozny et al, 2003). In addition, many of the existing regular classification algorithms can be easily extended to perform weighted classification. Thus, there are plenty of useful theoretical and algorithmic tools for both weighted classification and regular classification (Beygelzimer et al, 2005).

The main setup that we will study in this paper is cost-sensitive classification, which is more general than weighted classification.

Definition 2 (cost-sensitive classification) Assume that there is an unknown distribution $\mathcal{D}_c$ on $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^K$. A cost-sensitive example is a tuple $(\mathbf{x}, y, \mathbf{c}) \in \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^K$, where $\mathbf{c}[k]$ denotes the cost to be paid when $\mathbf{x}$ is predicted as category k. In the cost-sensitive classification setup, we are given a set of i.i.d. cost-sensitive training examples $\mathcal{S}_c = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N \sim \mathcal{D}_c^N$. We shall reuse

$$E(g, \mathcal{D}) \equiv \underset{(\mathbf{x},y,\mathbf{c})\sim\mathcal{D}}{\mathcal{E}} \mathbf{c}[g(\mathbf{x})]\,.$$

to denote the expected cost of any classifier $g\colon \mathcal{X} \to \mathcal{Y}$ with respect to some distribution $\mathcal{D}$. The goal of the cost-sensitive classification is to use $\mathcal{S}_c$ to find a classifier $\hat{g}$ such that $E(\hat{g}, \mathcal{D}_c)$ is small.

We make two remarks here. First, when looking at the definition of E, we see that the label y is actually not needed in evaluating the classifier g. We keep the label there to better illustrate the connection between cost-sensitive and regular/weighted classification. Naturally, we assume that $\mathcal{D}_c$ would only generate examples $(\mathbf{x}, y, \mathbf{c})$ such that $\mathbf{c}[y] = c_{\min} = \min_{1\le\ell\le K} \mathbf{c}[\ell]$.

Secondly, let us define the *classification cost vector* $\mathbf{c}_c^{(\ell)}[k] \equiv [\![\ell \neq k]\!]$. We see that weighted classification is a special case of cost-sensitive classification using $\mathbf{c} = w \cdot \mathbf{c}_c^{(y)}$ as the cost vector of $(\mathbf{x}, y, w)$, and regular classification is a special case of cost-sensitive classification using $\mathbf{c} = \mathbf{c}_c^{(y)}$ as the cost vector.

While both regular and weighted classification have been widely studied, cost-sensitive classification is theoretically well-understood only in the binary case (Zadrozny et al, 2003), in which weighted classification and cost-sensitive classification simply coincide. Next, we will introduce the cost-transformation technique, which allows us to tightly connect cost-sensitive classification with regular/weighted classification, and helps understand cost-sensitive classification better in the multiclass case.

3 Cost Transformation

Cost-transformation is a tool that connects a cost vector to other cost vectors. In particular, we hope to link any cost vector $\mathbf{c}$ with the classification cost vectors $C_c = \left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^K$, because the link allows us to reduce cost-sensitive classification (which deals with $\mathbf{c}$) to regular

classification (which deals with C_c). We start introducing the cost-transformation technique by making two definitions about the relations between cost vectors. The first definition relates two cost vectors $\tilde{\mathbf{c}}$ and $\mathbf{c}$.

Definition 3 (similar cost vectors) A cost vector $\tilde{\mathbf{c}}$ is similar to $\mathbf{c}$ by Δ if and only if $\tilde{\mathbf{c}}[\cdot] = \mathbf{c}[\cdot] + \Delta$ with some constant Δ.

For instance, $(4, 3, 2, 3)$ is similar to $(2, 1, 0, 1)$ by 2. We shall omit the "by Δ" part when it is clear from the context. Note that when $\tilde{\mathbf{c}}$ is similar to $\mathbf{c}$, using $\tilde{\mathbf{c}}$ for evaluating a prediction $g(\mathbf{x})$ is equivalent to using $\mathbf{c}$ plus a constant cost of Δ. The constant shifting from $\mathbf{c}$ to $\tilde{\mathbf{c}}$ does not change the relative cost difference between the prediction $g(\mathbf{x})$ and the best prediction y.

Next, we relate a cost vector $\mathbf{c}$ to a set of cost vectors C_b.

Definition 4 (decomposable cost vectors) A cost vector $\mathbf{c}$ is decomposable to a set of base cost vectors $C_b = \left\{\mathbf{c}_b^{(t)}\right\}_{t=1}^{T}$ if and only if there exists non-negative coefficients $\mathbf{q}[t]$ such that $\mathbf{c}[\cdot] = \sum_{t=1}^{T} \mathbf{q}[t] \cdot \mathbf{c}_b^{(t)}[\cdot]$.

That is, a cost vector $\mathbf{c}$ is decomposable to C_b if we can split $\mathbf{c}$ to a conic combination of the base cost vectors $\mathbf{c}_b^{(t)}$. Why is such a decomposition useful? Let us take a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ and choose the classification cost C_c as C_b. If $\mathbf{c}$ is decomposable to C_c, then for any classifier g,

$$\mathbf{c}[g(\mathbf{x})] = \sum_{\ell=1}^{K} \mathbf{q}[\ell] \cdot \mathbf{c}_c^{(\ell)}[g(\mathbf{x})] = \sum_{\ell=1}^{K} \mathbf{q}[\ell] \cdot [\![\ell \neq g(\mathbf{x})]\!] .$$

That is, if we randomly generate ℓ proportional to $\mathbf{q}[\ell]$ and relabel the cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ to a regular one $(\mathbf{x}, \ell, w = 1)$, then the cost that any classifier g needs to pay for its prediction on $\mathbf{x}$ is proportional to the expected classification error, where the expectation is taken with respect to the relabeling process. Thus, if a classifier g performs well for the "relabeled" (regular classification) task, it would also perform well for the original cost-sensitive classification task. The non-negativity of $\mathbf{q}[\ell]$ ensures that $\mathbf{q}$ can be normalized to form a probability distribution.[1]

Definition 4 is a key of the cost-transformation technique. It not only allows us to transform one cost vector $\mathbf{c}$ to an equivalent representation $\left\{(\mathbf{q}[t], \mathbf{c}_b^{(t)})\right\}$ for some general C_b, but more specifically also lets us relabel a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ to another (randomized) regular example $(\mathbf{x}, \ell, 1)$. However, is every cost vector $\mathbf{c}$ decomposable to the classification cost C_c? The short answer is no. For instance, the cost vector $\mathbf{c} = (6, 3, 0, 3)$ is not decomposable to C_c, because $\mathbf{c}$ yields a unique linear decomposition of C_c with some negative coefficients:

$$\mathbf{c} = -2 \cdot \mathbf{c}_c^{(1)} + 1 \cdot \mathbf{c}_c^{(2)} + 4 \cdot \mathbf{c}_c^{(3)} + 1 \cdot \mathbf{c}_c^{(4)}.$$

Although the cost vector $(6, 3, 0, 3)$ itself is not decomposable to C_c, we can easily see that its similar cost vector, $(12, 9, 6, 9)$, is decomposable to C_c. In fact, for any cost vector $\mathbf{c}$, there is an infinite number of its similar cost vectors $\tilde{\mathbf{c}}$ that are decomposable to C_c, as formalized below.

[1] We take a minor assumption that not all $\mathbf{q}[\ell]$ are zero. Otherwise $\mathbf{c} = \mathbf{0}$ and the example $(\mathbf{x}, y, \mathbf{c})$ can be simply discarded.

Theorem 1 (decomposition via a similar vector; Lin 2008) *Consider any cost vector* $\mathbf{c}$*. Assume that* $\tilde{\mathbf{c}}$ *is similar to* $\mathbf{c}$ *by* Δ*. Then,* $\tilde{\mathbf{c}}$ *is decomposable to* C_c *if and only if*

$$\Delta \geq (K-1)\, c_{\max} - \sum_{k=1}^{K} \mathbf{c}[k] \ ,$$

where $c_{\max} = \max_{1 \leq \ell \leq K} \mathbf{c}[\ell]$.

The proof (Lin, 2008) uses the fact that C_c is linearly independent while spanning $\mathbb{R}^K$, and the constant cost vector $(\Delta, \Delta, \cdots, \Delta)$ with a positive Δ is decomposable to C_c with $\mathbf{q}[\ell] = \frac{\Delta}{K-1}$.

From Theorem 1, there are infinitely many cost vectors $\tilde{\mathbf{c}}$ that we can use. The next question is, which is more preferable? Recall that with a given $\tilde{\mathbf{q}}$, the relabeling probability distribution is $\tilde{\mathbf{p}}[\ell] = \tilde{\mathbf{q}}[\ell] \Big/ \sum_{k=1}^{K} \tilde{\mathbf{q}}[k]$. To reduce the variance with respect to the relabeling process, one possibility is to require the discrete probability distribution $\tilde{\mathbf{p}}[\cdot]$ to be of the least entropy. That is, we want to solve the following optimization problem.

$$\begin{aligned} \min_{\tilde{\mathbf{p}}, \tilde{\mathbf{q}}, \Delta} \ & \sum_{\ell=1}^{K} \tilde{\mathbf{p}}[\ell] \log \frac{1}{\tilde{\mathbf{p}}[\ell]} \ , \qquad (1) \\ \text{subject to} \ \ & \mathbf{c}[\cdot] = \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot \mathbf{c}_c^{(\ell)}[\cdot] - \Delta, \quad \tilde{\mathbf{p}}[\cdot] = \tilde{\mathbf{q}}[\cdot] \Big/ \sum_{k=1}^{K} \tilde{\mathbf{q}}[k] \ ; \\ & \Delta \geq (K-1)\, c_{\max} - \sum_{k=1}^{K} \mathbf{c}[k] \ . \end{aligned}$$

Theorem 2 (decomposition with minimum-entropy; Lin 2008) *If not all* $\mathbf{c}[\ell]$ *are equal, the unique optimal solution to* (1) *is*

$$\tilde{\mathbf{q}}[\ell] = c_{\max} - \mathbf{c}[\ell] \,, \qquad (2)$$

$$\Delta = (K-1)\, c_{\max} - \sum_{k=1}^{K} \mathbf{c}[k] \,. \qquad (3)$$

Note that the resulting Δ is the smallest one that makes $\tilde{\mathbf{c}}$ decomposable to C_c. The details of the proof can be found in the previous work (Lin, 2008). Using Theorem 2, we can then define the following distribution $\mathcal{D}_r(\mathbf{x}, \ell, w)$ from $\mathcal{D}_c(\mathbf{x}, y, \mathbf{c})$.

$$\mathcal{D}_r(\mathbf{x}, \ell, w) = [\![w = 1]\!] \cdot \Lambda_1^{-1} \cdot \int_{y, \mathbf{c}} \tilde{\mathbf{q}}[\ell] \cdot \mathcal{D}_c(\mathbf{x}, y, \mathbf{c}),$$

where $\tilde{\mathbf{q}}[\ell]$ is computed from $\mathbf{c}$ using (2) and

$$\Lambda_1 = \int_{x, y, \mathbf{c}} \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot \mathcal{D}_c(\mathbf{x}, y, \mathbf{c}).$$

is a normalization constant.[2] Then, we can derive the following theorem.

[2] Even when all $\mathbf{c}[\ell]$ are equal, equation (2) can still be used to get $\tilde{\mathbf{q}}[\ell] = 0$ for all ℓ, which means the example $(\mathbf{x}, y, \mathbf{c})$ can be dropped instead of relabeled.

Theorem 3 (cost-transformation) *For any classifier g,*

$$E(g, \mathcal{D}_c) = \Lambda_1 \cdot E(g, \mathcal{D}_r) - \Lambda_2 \ ,$$

where Λ_2 is a constant that can be computed by integrating over the Δ term associated with each $\mathbf{c} \sim \mathcal{D}_c$ from (3)*.*

Theorem 3 can then be used to further prove the following regret equivalence theorem.

Theorem 4 (regret equivalence) *Consider $\mathcal{D}_c$ and its associated $\mathcal{D}_r$. If g_* is the optimal classifier under $\mathcal{D}_c$, and $\tilde{g_*}$ is the optimal classifier under the associated $\mathcal{D}_r$. Then, for any classifier g,*

$$E(g, \mathcal{D}_c) - E(g_*, \mathcal{D}_c) = \Lambda_1 \cdot \Big(E(g, \mathcal{D}_r) - E(\tilde{g_*}, \mathcal{D}_r) \Big).$$

That is, if a regular classification algorithm $\mathcal{A}_r$ can return some $\hat{g}$ that is close to $\tilde{g_*}$ under $\mathcal{D}_r$, the very same $\hat{g}$ would be close to the optimal classifier g_* for the original cost-sensitive classification task.

Theoretically, Theorem 4 indicates an equivalence in terms of hardness between general cost-sensitive classification tasks and general regular classification tasks. Then, we can reduce cost-sensitive classification to regular classification using Algorithm 1.

Algorithm 1 Reduction with Relabeling

1. Obtain N' independent regular training examples $\mathcal{S}_r = \{(\mathbf{x}_n, \ell_n, 1)\}_{n=1}^{N'}$ from $\mathcal{D}_r$:
   (a) Transform each $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ by (2).
   (b) Apply the COSTING reduction (Zadrozny et al, 2003) and accept the multi-labeled example $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ with probability proportional to $\sum_{\ell=1}^{K} \tilde{\mathbf{q}}_n[\ell]$.
   (c) For those $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ that survive from COSTING, randomly assign its label ℓ_n with probability proportional to $\tilde{\mathbf{q}}_n[\ell]$.
2. Use a regular classification algorithm $\mathcal{A}_r$ on $\mathcal{S}_r$ to obtain a classifier $\hat{g}_r$ that ideally yields a small $E(\hat{g}_r, \mathcal{D}_r)$.
3. Return $\hat{g} \equiv \hat{g}_r$.

From Algorithm 1, any good regular classification algorithm $\mathcal{A}_r$ can be turned into a good cost-sensitive classification algorithm $\mathcal{A}_c$, and trivially vice versa. That is, cost-sensitive classification is as hard as (or as easy as) regular classification. Note that the "hard" part of the arguments is quite important, as illustrated below.

While the cost-transformation steps above are supported with theoretical guarantees from Theorems 3 and 4, they may not work well in practice. For instance, if we look at an example $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ with $y_n = 1$ and $\mathbf{c}_n = (0, 1, 1, 334)$, the resulting $\tilde{\mathbf{q}}_n = (334, 333, 333, 0)$. Because of the large value in $\mathbf{c}_n[4]$, the example looks almost like a uniform mixture of labels $\{1, 2, 3\}$, with only 0.334 of probability to keep its original label. In other words, for the purpose of encoding some large components in a cost vector, the relabeling process could pay a huge variance (like noise) and relabel (or mislabel) the example more often than not. Then, the regular classification algorithm $\mathcal{A}_r$ may receive some $\mathcal{S}_r$ that contains lots of misleading labels, making it hard for the algorithm to return a decent $\hat{g}_r$.

The observation above indicates that cost-transformation can introduce noise to the learning process through relabeling. In other words, it reduces the original cost-sensitive classification task to a possibly more noisy regular classification task. The noise makes the

learning process less stable, and hence the returned $\hat{g}_r$ may not be good. One small improvement that aims at decreasing the relabeling variance is to use Algorithm 2, called training set expansion and weighting (TSEW), instead of relabeling. Note that the algorithm reduces cost-sensitive classification to weighted classification rather than regular classification.

Algorithm 2 Training Set Expansion and Weighting

1. Obtain NK training examples $\mathcal{S}_w = \{(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell})\}$:
   (a) Transform each $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ by (2).
   (b) For every ℓ, let $(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell}) = (\mathbf{x}_n, \ell, \tilde{\mathbf{q}}_n[\ell])$
   (c) Add $(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell})$ to $\mathcal{S}_w$.
2. Use a weighted classification algorithm $\mathcal{A}_w$ on $\mathcal{S}_w$ to obtain a classifier $\hat{g}_w$.
3. Return $\hat{g} \equiv \hat{g}_w$.

In Algorithm 2, it is not hard to show that $\mathcal{D}_r(x, \ell, 1) \propto w \cdot \mathcal{D}_w(x, \ell, w)$ for some $\mathcal{D}_w$, and $\mathcal{S}_w$ contains (dependent) examples generated from $\mathcal{D}_w$. We can think of $\mathcal{S}_w$, which trades independence for smaller variance, as a more stable version of $\mathcal{S}_r$. The expanded training set $\mathcal{S}_w$ contains all possible ℓ, and hence always includes the correct label y_n along with the largest weight $w_{ny_n} = \tilde{\mathbf{q}}_n[y_n]$.

Note that the $\mathcal{A}_w$ in TSEW can also be performed by a regular classification algorithm $\mathcal{A}_c$ using the COSTING reduction (Zadrozny et al, 2003). Then, Algorithm 1 is simply a special (and possibly less stable) case of TSEW.

The TSEW algorithm is a good representative of our proposed cost-transformation technique. Note that TSEW is actually the same as the data space expansion (DSE) algorithm proposed by Abe et al (2004). Nevertheless, our derivation from the minimum entropy perspective is novel, and our theoretical results on the out-of-sample cost $E(g, \mathcal{D}_c)$ are more general than the in-sample cost analysis by Abe et al (2004). Xia et al (2007) also proposed an algorithm similar to TSEW using LogitBoost as $\mathcal{A}_w$ based on a restricted version of Theorem 3. It should be noted that the results discussed in this section are partially influenced by the work of Abe et al (2004) but are independent from the work of Xia et al (2007).

From the experimental results in literature, a direct use of TSEW (DSE) does not perform well in practice (Abe et al, 2004). A possible explanation is that although $\mathcal{S}_w$ does not contain relabeling noise, it still carries multi-labeling ambiguities. That is, the same input vector $\mathbf{x}_n$ can come with many different labels (with possibly different weights) in $\mathcal{S}_w$. Thus, common $\mathcal{A}_w$ can find $\mathcal{S}_w$ too difficult to digest (Xia et al, 2007). One could improve the basic TSEW algorithm by using (or designing) an $\mathcal{A}_w$ that is robust with multi-labeled training feature vectors. We shall present one such algorithm in the next section.

4 Cost-Sensitive One-Versus-One

In this section, we propose a novel cost-sensitive classification algorithms by coupling the cost-transformation technique with the popular and robust one-versus-one (OVO) algorithm for regular classification. Before we get into our proposed cost-sensitive one-versus-one (CSOVO) algorithm, we shall introduce the original OVO first.

4.1 Original One-versus-one

We shall present a weighted version of OVO here. As shown in Algorithm 3, OVO decomposes the multiclass classification task into $\frac{K(K-1)}{2}$ binary classification subtasks. Because of the $O(K^2)$ growth in the number of subtasks, OVO is usually more suited when K is not too large (Hsu and Lin, 2002).

Algorithm 3 One-versus-one (Hsu and Lin, 2002)

1. For each i, j that $1 \leq i < j \leq K$,
   (a) Take the original $\mathcal{S}_w = \{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$ and construct a binary training set $\mathcal{S}_b^{(i,j)} = \{(\mathbf{x}_n, y_n, w_n) \colon y_n = i \text{ or } j\}$.
   (b) Use a weighted binary classification algorithm $\mathcal{A}_b$ on $\mathcal{S}_b^{(i,j)}$ to get a binary classifier $\hat{g}_b^{(i,j)}$.
2. Return $\hat{g}(\mathbf{x}) = \underset{1 \leq \ell \leq K}{\operatorname{argmax}} \sum_{i<j} \left[\!\!\left[\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\!\right]$.

In short, each binary classification subtask consists of comparing examples from two categories only. That is, each $\hat{g}_b^{(i,j)}(\mathbf{x})$ intends to predict whether $\mathbf{x}$ "prefers" category i or category j, and $\hat{g}$ predicts with the preference votes gathered from those $\hat{g}_b^{(i,j)}$. The goal of $\mathcal{A}_b$ is to locate binary classifiers $\hat{g}_b^{(i,j)}$ with a small $E\Big(\hat{g}_b^{(i,j)}, \mathcal{D}_{\mathsf{OVO}}^{(i,j)}\Big)$, where

$$\mathcal{D}_{\mathsf{OVO}}^{(i,j)}\Big(\mathbf{x}, y, u\Big) = \Big[\!\!\Big[u = [\![y = i \text{ or } j]\!] \Big]\!\!\Big] \int_w \mathcal{D}_r\Big(\mathbf{x}, y, w\Big).$$

In particular, it has been proved (Beygelzimer et al, 2005) that

$$E(\hat{g}, \mathcal{D}_r) \leq 2 \sum_{i<j} E\Big(\hat{g}_b^{(i,j)}, \mathcal{D}_{\mathsf{OVO}}^{(i,j)}\Big).$$

That is, if $E\Big(\hat{g}_b^{(i,j)}, \mathcal{D}_{\mathsf{OVO}}^{(i,j)}\Big)$ are all small, then $E(\hat{g}, \mathcal{D}_r)$ should also be small.

4.2 Cost-sensitive One-versus-one

By coupling OVO with the cost-transformation technique (TSEW in Algorithm 2), we can easily get a preliminary version of CSOVO in Algorithm 4.

One thing to notice in Algorithm 4 is that each training example $(\mathbf{x}_n, y_n)$ may be split to two examples $\Big(\mathbf{x}_n, i, w_n^{(i)}\Big)$ and $\Big(\mathbf{x}_n, j, w_n^{(j)}\Big)$ for each $\mathcal{S}_b^{(i,j)}$. That is, the example is ambiguously presented for each binary classification subtask. We can take a simple trick to eliminate the ambiguity before training. In particular, we keep only the label (say, i) that comes with a larger weight, and adjust its weight to $\Big|w_n^{(i)} - w_n^{(j)}\Big|$. The trick follows from the same principle as shifting the cost vectors to a similar one. Then, we can eliminate one unnecessary example and remove the multi-labeling ambiguity in the binary classification subtask.

Algorithm 4 TSEW-OVO

1. For each i, j that $1 \leq i < j \leq K$,
   (a) Transform each cost-sensitive example $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ by (2).
   (b) Use all the $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ to construct a binary classification training set

$$\mathcal{S}_b^{(i,j)} = \left\{ \left(\mathbf{x}_n, i, w_n^{(i)} \right) \right\} \cup \left\{ \left(\mathbf{x}_n, j, w_n^{(j)} \right) \right\},$$

   where $w_n^{(\ell)} = \tilde{\mathbf{q}}_n[\ell]$.
   (c) Use a weighted binary classification algorithm $\mathcal{A}_b$ on $\mathcal{S}_b^{(i,j)}$ to get a binary classifier $\hat{g}_b^{(i,j)}$.
2. Return $\hat{g}(\mathbf{x}) = \underset{1 \leq \ell \leq K}{\operatorname{argmax}} \sum_{i<j} [\![\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell]\!]$.

Recall that $(\mathbf{x}_n, i, w_n^{(i)})$ would be of weight $w_n^{(i)} = \tilde{\mathbf{q}}_n[i]$ and $(\mathbf{x}_n, j, w_n^{(j)})$ would be of weight $w_n^{(j)} = \tilde{\mathbf{q}}_n[j]$. By the discussion above, the simplified $\mathcal{S}_b^{(i,j)}$ is

$$\begin{aligned} & \left\{ \left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmax}} \, \tilde{\mathbf{q}}_n[\ell], \left| \tilde{\mathbf{q}}_n[i] - \tilde{\mathbf{q}}_n[j] \right| \right) \right\} \\ = & \left\{ \left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \, \mathbf{c}_n[\ell], \left| \mathbf{c}_n[i] - \mathbf{c}_n[j] \right| \right) \right\}. \end{aligned} \tag{4}$$

Then, we get our proposed CSOVO algorithm.

Algorithm 5 Cost-sensitive One-versus-one

1. For each i, j that $1 \leq i < j \leq K$,
   (a) Take the original $\mathcal{S}_c = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ and construct $\mathcal{S}_b^{(i,j)}$ by (4).
   (b) Use a weighted binary classification algorithm $\mathcal{A}_b$ on $\mathcal{S}_b^{(i,j)}$ to get a binary classifier $\hat{g}_b^{(i,j)}$.
2. Return $\hat{g}(\mathbf{x}) = \underset{1 \leq \ell \leq K}{\operatorname{argmax}} \sum_{i<j} [\![\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell]\!]$.

An intuitive explanation is that CSOVO asks each binary classifier $\hat{g}_b^{(i,j)}$ to answer the question "is $\mathbf{c}[i]$ or $\mathbf{c}[j]$ smaller for this $\mathbf{x}$?" We can easily see that CSOVO (Algorithm 5) takes OVO (Algorithm 3) as a special case when using only the weighted classification cost vectors $(w \cdot \mathbf{c}_c^{(\ell)})$.

4.3 Theoretical Guarantee

Next, we analyze the theoretical guarantee of Algorithm 5. Note that each created example

$$\left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \, \mathbf{c}_n[\ell], \left| \mathbf{c}_n[i] - \mathbf{c}_n[j] \right| \right)$$

can be thought as if coming from a distribution

$$\begin{aligned} & \mathcal{D}_{\text{CSOVO}}^{(i,j)}(\mathbf{x}, k, u) \\ = & \int_{y,\mathbf{c}} [\![k = \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \, \mathbf{c}[\ell]]\!] \, [\![u = \left| \mathbf{c}[i] - \mathbf{c}[j] \right|]\!] \, \mathcal{D}_c(\mathbf{x}, y, \mathbf{c}). \end{aligned}$$

We then get the following theorem:

Theorem 5 *Consider any family of binary classifiers*

$$\left\{g_b^{(i,j)} \colon \mathcal{X} \to \{i,j\}\right\}_{1\le i<j\le K} \quad .$$

Let $g(\mathbf{x}) = \underset{1\le\ell\le K}{\textit{argmax}} \sum_{i<j} \left[\!\left[g_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$ *. Then,*

$$E(g,\mathcal{D}_c) - \underset{(\mathbf{x},y,\mathbf{c})\sim\mathcal{D}_c}{\mathcal{E}} c_{\min} \le 2\sum_{i<j} E\left(g_b^{(i,j)}, \mathcal{D}_{\textit{CSOVO}}^{(i,j)}\right). \tag{5}$$

Proof For each $(\mathbf{x}, y, \mathbf{c})$ generated from $\mathcal{D}_c$, if $\mathbf{c}[g(\mathbf{x})] = \mathbf{c}[y] = c_{\min}$, its contribution on the left-hand side is 0, which is trivially less than its contribution on the right-hand side.

Without loss of generality (by sorting the elements of the cost vector $\mathbf{c}$ and shuffling the labels $y \in \mathcal{Y}$), consider an example $(\mathbf{x}, y, \mathbf{c})$ such that

$$c_{\min} = \mathbf{c}[1] \le \mathbf{c}[2] \le \ldots \le \mathbf{c}[K] = c_{\max}.$$

From the results of Beygelzimer et al (2005), suppose $g(\mathbf{x}) = k$, then for each $1 \le \ell \le k-1$, there are at least $\lceil k/2 \rceil$ pairs (i, j), where $i \le k < j$, and

$$g_b^{(i,j)}(\mathbf{x}) \neq \underset{\ell=i \text{ or } j}{\text{argmin}}\ \mathbf{c}_n[\ell]\,.$$

Therefore, the contribution of $(\mathbf{x}, y, \mathbf{c})$ on the right-hand side is no less than

$$\begin{aligned}
\sum_{\ell=1}^{k-1} (\mathbf{c}[\ell+1] - \mathbf{c}[\ell]) \left\lceil \ell \Big/ 2 \right\rceil &\ge \frac{1}{2} \sum_{\ell=1}^{k-1} \ell\,(\mathbf{c}[\ell+1] - \mathbf{c}[\ell]) \\
&= \frac{1}{2} \sum_{\ell=1}^{k-1} (\mathbf{c}[k] - \mathbf{c}[\ell]) \\
&\ge \frac{1}{2} (\mathbf{c}[k] - c_{\min})\ ,
\end{aligned}$$

and the left-hand-side contribution is $(\mathbf{c}[k] - c_{\min})$. The desired result can be proved by integrating over all $\mathcal{D}_c$. □

Thus, similar to the original OVO algorithm, if $E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\text{CSOVO}}^{(i,j)}\right)$ are all small, then the resulting $E(\hat{g}, \mathcal{D}_c)$ should also be small.

4.4 A Sibling Algorithm: Weighted All-pairs

Note that a similar theoretical proof of Theorem 5 was made by Beygelzimer et al (2005) to analyze another algorithm called weighted all-pairs (WAP). As illustrated in Algorithm 6, the WAP algorithm shares many similar algorithmic structures with CSOVO. In particular, we see that except the difference between equations (4) and (6), WAP is exactly the same as CSOVO.

Algorithm 6 A Special Version of WAP (Beygelzimer et al, 2005)

Run CSOVO, while replacing (4) in step 1(a) with

$$\mathcal{S}_b^{(i,j)} = \left\{ \left(\mathbf{x}_n, \operatorname*{argmin}_{\ell=i \text{ or } j} \mathbf{c}_n[\ell], \left| \mathbf{v}_n[i] - \mathbf{v}_n[j] \right| \right) \right\} \tag{6}$$

$$\text{where} \quad \mathbf{v}_n[i] = \int_{c_{\min}}^{\mathbf{c}_n[i]} \frac{1}{|\{k\colon \mathbf{c}_n[k] \le t\}|} dt$$

Define

$$\begin{aligned} &\mathcal{D}_{\mathsf{WAP}}^{(i,j)}(\mathbf{x}, k, u) \\ &= \int_{y,\mathbf{c}} \left[\!\!\left[k = \operatorname*{argmin}_{\ell=i \text{ or } j} \mathbf{c}[\ell] \right]\!\!\right] \left[\!\!\left[u = \left| \mathbf{v}[i] - \mathbf{v}[j] \right| \right]\!\!\right] \mathcal{D}_c(\mathbf{x}, y, \mathbf{c}), \end{aligned}$$

where $\mathbf{v}$ is computed from $\mathbf{c}$ using a similar definition as the one in Algorithm 6. The cost bound of WAP is then (Beygelzimer et al, 2005)

$$E(g, \mathcal{D}_c) - \mathop{\mathcal{E}}_{(\mathbf{x},y,\mathbf{c})\sim\mathcal{D}_c} c_{\min} \le 2 \sum_{i<j} E\left(g_b^{(i,j)}, \mathcal{D}_{\mathsf{WAP}}^{(i,j)} \right). \tag{7}$$

Note that we can let $\mathbf{v}'_n[i] \equiv \mathbf{c}_n[i] - c_{\min} = \int_{c_{\min}}^{\mathbf{c}_n[i]} (1)\, dt$. Then, CSOVO equivalently uses $\left| \mathbf{v}'_n[i] - \mathbf{v}'_n[j] \right|$ as the underlying example weight. It is not hard to see that for any given example $(\mathbf{x}, y, \mathbf{c})$, the associated

$$\left| \mathbf{v}'_n[i] - \mathbf{v}'_n[j] \right| \ge \left| \mathbf{v}_n[i] - \mathbf{v}_n[j] \right|.$$

Thus, the WAP cost bound (7) is tighter than the CSOVO one (5) when using the same binary classifiers $\left\{ \hat{g}_b^{(i,j)} \right\}$. In particular, while the right-hand-side of (5) and (7) look similar, the total weight that CSOVO takes in $\mathcal{D}_{\mathsf{CSOVO}}^{(i,j)}$ is larger than the total weight that WAP takes in $\mathcal{D}_{\mathsf{WAP}}^{(i,j)}$. The difference allows WAP to have an $O(K)$ regret transform (Beygelzimer et al, 2005) instead of the $O(K^2)$ one of CSOVO (Theorem 5). Thus, for binary classifiers $\left\{ \hat{g}_b^{(i,j)} \right\}$ with the same error rate, it appears that WAP is better than CSOVO because of the tighter upper bound. However, CSOVO enjoys the advantage of efficiency and simplicity in implementation, because equation (6) would require a complete sorting of each cost vector (of size K) to compute while (4) only needs a simple subtraction. In the next section, we shall study whether the tighter cost bound with the additional complexity (WAP) leads to better empirical performance than the other way around (CSOVO).

4.5 Another Sibling Algorithm: All-pair Filter Tree

Another sibling algorithm of CSOVO is called the all-pair filter tree (APFT; Beygelzimer et al, 2007). APFT designs a elimination-based tournament in order to find the label with the lowest cost. During training, if an example $(\mathbf{x}_n, y_n)$ as well as the classifiers in the lower

levels of the tree allow labels $\{i, j\}$ to meet in one game of the tournament, a weighted example

$$\left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \mathbf{c}_n[\ell], \left|\mathbf{c}_n[i] - \mathbf{c}_n[j]\right|\right)$$

is added to the training set $\mathcal{S}_b^{(i,j)}$ for learning a classifier $\hat{g}_b^{(i,j)}$. The goal of $\hat{g}_b^{(i,j)}$ is to achieve a small $E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\mathsf{APFT}}^{(i,j)}\right)$, where

$$\begin{aligned}&\mathcal{D}_{\mathsf{APFT}}^{(i,j)}(\mathbf{x}, k, u)\\ &= \int_{y,\mathbf{c}} \left[\!\!\left[i, j \text{ attends the tournament}\right]\!\!\right] \left[\!\!\left[k = \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \mathbf{c}[\ell]\right]\!\!\right] \left[\!\!\left[u = \left|\mathbf{c}[i] - \mathbf{c}[j]\right|\right]\!\!\right] \mathcal{D}_c(\mathbf{x}, y, \mathbf{c}).\end{aligned}$$

Note that the condition $\left[\!\!\left[i, j \text{ attends the tournament}\right]\!\!\right]$ depends on lower-level classifiers that "filter" the distribution for higher-level training. During prediction, the results from $\left\{\hat{g}_b^{(i,j)}\right\}$ are decoded using the same tournament design rather than voting.

Let $\left\{g_b^{(i,j)}\right\}$ be a set of binary classifiers and g_{APFT} be the resulting classifier after decoding the predictions of $\left\{g_b^{(i,j)}\right\}$ from the tournament. It can be shown (Beygelzimer et al, 2007) that

$$E(g_{\mathsf{APFT}}, \mathcal{D}_c) - \underset{(\mathbf{x},y,\mathbf{c})\sim\mathcal{D}_c}{\mathcal{E}} c_{\min} \leq \sum_{i<j} E\left(g_b^{(i,j)}, \mathcal{D}_{\mathsf{APFT}}^{(i,j)}\right). \tag{8}$$

Comparing CSOVO with APFT, we see one similarity: the weighted examples included in each $\mathcal{S}_b^{(i,j)}$. Nevertheless, note that APFT uses fewer examples than CSOVO—the former uses only pairs of labels that are met in the tournament and the latter uses all possible pairs of labels. The difference allows for a tighter error bound for APFT by conditioning on the tournament results. Thus, when using binary classifiers of the same error rate, it appears that APFT is better than CSOVO because of the tighter upper bound. Furthermore, by restricting to a specific tournament, APFT results in an $O(K)$ predicting scheme instead of the $O(K^2)$ one that CSOVO needs to take. Nevertheless, APFT essentially breaks the symmetry between classes by restricting to a specific tournament, and the reduced number of examples in each $\mathcal{S}_b^{(i,j)}$ could degrade the practical learning performance. In the next section, we shall also study whether the tighter cost bound with the tournament restriction (APFT) leads to better empirical performance than the other way around (CSOVO).

5 Experiments

We will first compare CSOVO with the original OVO on various real-world data sets. Then, we will compare CSOVO with WAP (Beygelzimer et al, 2005) and APFT (Beygelzimer et al, 2007). All four algorithms are of OVO-type. That is, they obtain a multiclass classifier $\hat{g}$ by calling a weighted binary classification algorithm $\mathcal{A}_b$ for $\frac{K(K-1)}{2}$ times. During prediction, CSOVO, OVO and WAP requires gathering votes from $\frac{K(K-1)}{2}$ binary classifiers, while APFT determines the label by using $K-1$ of those classifiers in the tournament. In addition, we will compare CSOVO with other existing algorithms that also reduce cost-sensitive classification to weighted binary classification.

Table 1 classification data sets

data set	# examples	# categories (K)	# features
zoo	101	7	16
glass	214	6	9
vehicle	846	4	18
vowel	990	11	10
yeast	1484	10	8
segment	2310	7	19
dna	3186	3	180
pageblock	5473	5	10
satimage	6435	6	36
usps	9298	10	256

We take the support vector machine (SVM) with the perceptron kernel (Lin and Li, 2008) as $\mathcal{A}_b$ in all the experiments and use LIBSVM (Chang and Lin, 2001) as our SVM solver. Note that SVM with the perceptron kernel is known as a strong classification algorithm (Lin and Li, 2008) and can be naturally adopted to perform weighted binary classification (Zadrozny et al, 2003). We will also take a weaker classifier, namely SVM with the linear kernel, in Subsection 5.6.

We use ten classification data sets: zoo, glass, vehicle, vowel, yeast, segment, dna, pageblock, satimage, usps (Table 1).[3] The first nine come from the UCI machine learning repository (Hettich et al, 1998) and the last one is from Hull (1994).

Note that the ten data sets were originally gathered as regular classification tasks. We shall first adopt the randomized proportional (RP) cost-generation procedure that was used by Beygelzimer et al (2005). In particular, we generate the cost vectors from a cost matrix $C(y,k)$ that does not depends on $\mathbf{x}$. The diagonal entries $C(y,y)$ are set as 0 and each of the other entries $C(y,k)$ is a random variable sampled uniformly from $\left[0, 2000\frac{|\{n:\, y_n=k\}|}{|\{n:\, y_n=y\}|}\right]$. Then, for a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$, we simply take $\mathbf{c}[k] = C(y,k)$. We acknowledge that the RP procedure may not fully reflect realistic application needs. Nevertheless, we still take the procedure as it is a longstanding benchmark for comparing general-purpose cost-sensitive classification algorithms. We will take another cost-generating procedure in Subsection 5.4.

We randomly choose 75% of the examples in each data set for training and leave the other 25% of the examples as the test set. Then, each feature in the training set is linearly scaled to $[-1, 1]$, and the feature in the test set is scaled accordingly. The results reported are all averaged over 20 trials of different training/test splits, along with the standard error. In the coming tables, those entries within one standard error of the lowest one are marked in bold.

SVM with the perceptron kernel takes a regularization parameter (Lin and Li, 2008), which is chosen within $\left\{2^{-17}, 2^{-15}, \ldots, 2^{3}\right\}$ with a 5-fold cross-validation (CV) procedure on only the training set (Hsu et al, 2003). For the OVO algorithm, the CV procedure selects the parameter that results in the smallest cross-validation classification error. For CSOVO and other cost-sensitive classification algorithms, the CV procedure selects the parameter that results in the smallest cross-validation cost. We then re-run each algorithm on the whole training set with the chosen parameter to get the classifier $\hat{g}$. Finally, we evaluate the average performance of $\hat{g}$ with the test set.

[3] All data sets except zoo, glass, yeast and pageblock are actually downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

Table 2 test cost of CSOVO/OVO

data set	CSOVO	OVO
zoo	$\mathbf{36.79 \pm 9.09}$	105.56 ± 33.45
glass	$\mathbf{210.06 \pm 15.88}$	492.99 ± 40.77
vehicle	$\mathbf{155.14 \pm 20.63}$	185.38 ± 17.23
vowel	20.05 ± 1.95	$\mathbf{11.90 \pm 1.96}$
yeast	$\mathbf{52.45 \pm 2.97}$	5823.21 ± 1290.65
segment	$\mathbf{25.27 \pm 2.25}$	$\mathbf{25.15 \pm 2.11}$
dna	53.18 ± 4.25	$\mathbf{48.15 \pm 3.33}$
pageblock	$\mathbf{24.98 \pm 4.94}$	501.57 ± 74.98
satimage	$\mathbf{66.57 \pm 4.77}$	94.07 ± 5.49
usps	$\mathbf{20.51 \pm 1.17}$	23.62 ± 0.66

Table 3 test classification error of CSOVO/OVO

data set	CSOVO	OVO
zoo	0.131 ± 0.015	$\mathbf{0.060 \pm 0.010}$
glass	0.605 ± 0.034	$\mathbf{0.304 \pm 0.010}$
vehicle	0.283 ± 0.020	$\mathbf{0.185 \pm 0.005}$
vowel	0.059 ± 0.010	$\mathbf{0.011 \pm 0.002}$
yeast	0.767 ± 0.007	$\mathbf{0.398 \pm 0.006}$
segment	0.051 ± 0.008	$\mathbf{0.024 \pm 0.001}$
dna	0.115 ± 0.017	$\mathbf{0.043 \pm 0.002}$
pageblock	0.776 ± 0.064	$\mathbf{0.033 \pm 0.001}$
satimage	0.168 ± 0.012	$\mathbf{0.072 \pm 0.002}$
usps	0.077 ± 0.030	$\mathbf{0.023 \pm 0.000}$

5.1 CSOVO versus OVO

Table 2 compares the test cost of CSOVO and the original cost-insensitive OVO. We can see that on 7 out of the 10 data sets, CSOVO is significantly better than OVO, which justifies that it can be useful to include the cost information into the training process. The t-test results, which will be shown in Table 8, suggest the same finding. The big difference on yeast and pageblock is because they are highly unbalanced and hence the components in $\mathbf{c}$ can be huge. Then, not using (or discarding) cost information (as OVO does) would intuitively lead to worse performance.

The only data set on which CSOVO is much worse than OVO is vowel. One may wonder why including the accurate cost information does not improve performance. The reason lies in Table 3, in which we compare the test classification error rate of CSOVO and OVO. Not surprisingly, OVO can always achieve lower test error than CSOVO. In fact, on yeast and pageblock, in which some of the cost components are large, we clearly see that CSOVO is willing to trade a significant amount of classification accuracy for a lower cost. For vowel, note that OVO can achieve a very low test error (1.1%), which readily leads to a low test cost. Then the caveat of using CSOVO, or more generally the cost-transformation technique, arises. In particular, cost transformation reduces the original "easy" task for OVO to a more difficult one. The change in hardness degrades the learning performance, and thus CSOVO results in relatively higher test cost.

Recall that CSOVO comes from coupling the cost-transformation technique with OVO, and we discussed in Section 3 that cost transformation inevitably introduces multi-labeling ambiguity into the learning process. The ambiguity acts like noise, and generally makes the learning tasks more difficult. On the vowel data set, OVO readily achieves low test error and hence low test cost, while CSOVO suffers from the difficult learning tasks and hence

Table 4 test cost of CSOVO/WAP/APFT

data set	CSOVO	WAP	APFT
zoo	**36.79±9.09**	49.09±16.67	56.92±15.41
glass	**210.06±15.88**	**220.29±18.56**	**215.95±17.36**
vehicle	**155.14±20.63**	**148.63±19.74**	**158.60±20.35**
vowel	**20.05±1.95**	**19.36±1.81**	27.59±2.94
yeast	**52.45±2.97**	**52.71±3.58**	63.53±8.11
segment	**25.27±2.25**	**24.40±1.96**	28.51±2.55
dna	**53.18±4.25**	**51.13±4.37**	**53.37±5.49**
pageblock	24.98±4.94	**20.68±2.52**	25.60±4.98
satimage	**66.57±4.77**	72.05±5.13	80.70±5.98
usps	**20.51±1.17**	**21.04±1.19**	29.75±1.75

Table 5 test cost bound of CSOVO/WAP/APFT

data set	CSOVO with (5)	WAP with (7)	APFT with (8)
zoo	465.53±124.48	**147.56±30.32**	**143.54±40.97**
glass	2138.52±260.90	811.30±69.02	**471.66±50.41**
vehicle	499.53±55.49	295.87±30.25	**245.15±27.54**
vowel	475.25±35.18	126.60±9.51	**111.26±10.87**
yeast	4820.96±301.43	736.31±26.70	**401.92±55.94**
segment	245.86±21.83	94.37±7.50	**77.65±7.04**
dna	95.77±6.76	**71.94±5.77**	**69.18±8.35**
pageblock	533.36±89.50	260.73±43.80	**112.51±28.93**
satimage	477.45±22.52	**193.25±10.88**	**183.24±12.73**
usps	415.36±23.40	112.22±6.08	**98.34±5.78**

gets high test cost. Similar situations happen on segment, dna, usps, in which the test cost of CSOVO and OVO are quite close. That is, it is worth trading the cost information for easier learning tasks. On the other hand, when OVO cannot achieve low test error (like on vehicle) or when the cost information is extremely important (like on pageblock), it is worth trading the easy learning tasks for knowing the accurate cost information, and thus CSOVO performs better.

5.2 CSOVO versus WAP and APFT

Next, we compare CSOVO with WAP and APFT in terms of the average test cost in Table 4. We see that CSOVO and WAP are comparable in performance, with WAP being slightly worse on satimage and CSOVO being slightly worse on pageblock. The similarity is natural because CSOVO and WAP are only different in the weights given to the underlying binary examples. On the other hand, Table 4 shows that APFT usually performs slightly worse than CSOVO and WAP. Thus, CSOVO and WAP should be better choices, unless the $O(K)$ prediction time (and the shorter $O(K^2)$ training time because of the conditioning on the tournament) of APFT is needed. Again, the t-test results in Table 8 lead to the same conclusion.

We mentioned in Subsections 4.4 and 4.5 that the cost bounds of WAP and APFT are both tighter than the one of CSOVO. To understand whether the cost bounds can explain the results in Table 4, we compute the bounds using the test set and list them in Table 5. We see that both WAP and APFT can indeed reach lower cost bounds. Nevertheless, the bounds are quite loose when compared to the actual test cost values in Table 4. Because of

Table 6 comparison of meta-algorithms that reduce cost-sensitive to binary classification

	CSOVO	WAP	APFT	FT/TREE	SECOC	CSOVA
# of binary classifiers	$\frac{K(K-1)}{2}$	$\frac{K(K-1)}{2}$	$K-1$	$\frac{K(K-1)}{2}$	$12 \cdot 2^{\lceil \log_2 K \rceil}$	K
prediction time	$O(K^2)$	$O(K^2)$	$O(K)$	$O(\log_2 K)$	$O(K)$	$O(K)$
theoretical guarantee	yes	yes	yes	yes	yes	partial

the looseness, using WAP for the tighter bound (or APFT for the tighter bound) does not lead to much gain in performance.

In summary, CSOVO performs better than APFT; CSOVO performs similarly to WAP but enjoys a simpler and more efficient implementation (see Subsection 4.4). Thus, CSOVO should be preferred over both WAP and APFT in practice.

5.3 CSOVO versus Others

Next, we compare CSOVO with four other existing algorithms, namely TREE (Beygelzimer et al, 2005), Filter Tree (FT; Beygelzimer et al, 2007), Sensitive Error Correcting Output Codes (SECOC; Langford and Beygelzimer, 2005), and Cost-sensitive One-versus-all (CSOVA; Lin, 2008). The algorithms cover commonly-used decompositions from multiclass classification to binary classification. Note that TREE, FT and SECOC, like CSOVO and WAP, come with sound theoretical guarantee, which assures that a good binary classifier can be cast as a good cost-sensitive one. CSOVA, on the other hand, follows a heuristic step in its derivation and hence does not carry a strong theoretical guarantee. A quick comparison about the properties of the four meta-algorithms (as well as CSOVO, WAP and APFT) are shown in Table 6.

Table 7 compares the average test RP cost of CSOVO, TREE, FT, SECOC and CSOVA; Table 8 lists the paired t-test results with significance level 0.05. SECOC is the worst of the five, which is because it contains a thresholding (quantization) step that can lead to an inaccurate representation of the cost information.

FT performs slightly worse than CSOVO, which demonstrates that a full pairwise comparison (CSOVO/WAP) can be more stable than an elimination-based tournament (FT). TREE performs even worse than FT, which complies with the finding in the original FT paper (Beygelzimer et al, 2007) that a regret-based reduction (FT) can be more robust than an error-based reduction (TREE).

Interestingly, when comparing Table 7 with Table 4, we see that FT performs better than APFT. Such a result suggests that decomposing the decision of each game to pairwise classifiers (APFT) is not better than directly predicting the outcome of each game in the tournament (FT). This is possibly because the reduced number of examples in each $\mathcal{S}_b^{(i,j)}$ of APFT could degrade the practical learning performance (see Subsection 4.5), especially when using a highly nonlinear model like SVM with the perceptron kernel. We will discuss more about this issue in Subsection 5.6.

CSOVO and CSOVA are quite similar in performance on many data sets. Nevertheless, recall that CSOVO comes with a stronger theoretical guarantee than CSOVA. Thus, when K is relatively small (like on our data sets) and training $\frac{K(K-1)}{2}$ binary classifiers is affordable, CSOVO is the best meta-algorithm for reducing multiclass cost-sensitive classification to binary classification in terms of both accuracy and efficiency.

Table 7 test cost of meta-algorithms that reduce cost-sensitive to binary classification

data set	CSOVO	FT	TREE	SECOC	CSOVA
zoo	$\mathbf{36.79 \pm 9.09}$	74.01 ± 27.65	57.07 ± 17.40	179.02 ± 30.52	79.70 ± 25.31
glass	$\mathbf{210.06 \pm 15.88}$	$\mathbf{212.76 \pm 19.38}$	264.02 ± 24.49	347.77 ± 37.87	231.77 ± 18.51
vehicle	$\mathbf{155.14 \pm 20.63}$	$\mathbf{156.01 \pm 20.14}$	$\mathbf{156.72 \pm 20.12}$	$\mathbf{167.60 \pm 20.97}$	$\mathbf{156.91 \pm 19.91}$
vowel	20.05 ± 1.95	24.66 ± 2.92	28.42 ± 3.02	95.25 ± 7.35	$\mathbf{15.06 \pm 1.73}$
yeast	$\mathbf{52.45 \pm 2.97}$	$\mathbf{53.73 \pm 3.14}$	66.49 ± 6.09	277.14 ± 49.41	86.80 ± 9.78
segment	$\mathbf{25.27 \pm 2.25}$	$\mathbf{27.06 \pm 2.32}$	27.76 ± 2.26	68.08 ± 4.02	$\mathbf{25.50 \pm 2.23}$
dna	53.18 ± 4.25	53.76 ± 4.23	55.32 ± 4.39	65.90 ± 5.66	$\mathbf{39.67 \pm 2.41}$
pageblock	$\mathbf{24.98 \pm 4.94}$	29.93 ± 6.16	$\mathbf{23.00 \pm 3.28}$	249.28 ± 67.31	42.99 ± 8.17
satimage	$\mathbf{66.57 \pm 4.77}$	74.01 ± 4.57	78.13 ± 4.31	102.81 ± 5.41	77.26 ± 4.73
usps	$\mathbf{20.51 \pm 1.17}$	27.07 ± 1.52	25.74 ± 1.55	86.59 ± 9.45	$\mathbf{21.49 \pm 1.23}$

Table 8 t-test for comparing CSOVO with other meta-algorithms using RP cost

data set	OVO	WAP	APFT	FT	TREE	SECOC	CSOVA
zoo	$\circ$	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\sim$
glass	$\circ$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$	$\sim$
vehicle	$\circ$	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\sim$
vowel	$\times$	$\sim$	$\circ$	$\circ$	$\circ$	$\circ$	$\times$
yeast	$\circ$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$	$\circ$
segment	$\sim$	$\sim$	$\circ$	$\circ$	$\circ$	$\circ$	$\sim$
dna	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\times$
pageblock	$\circ$	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$
satimage	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
usps	$\circ$	$\sim$	$\circ$	$\circ$	$\circ$	$\circ$	$\sim$

$\circ$: CSOVO significantly better; $\times$: CSOVO significantly worse; $\sim$: similar

5.4 Comparison with Emphasizing Cost

Next, we take another cost-generating procedure to compare cost-sensitive classification algorithms. We consider a practical task in which one wishes to mark some of the classes as "important." Traditionally, the task is tackled with the weighted classification approach: putting a larger weight w on those classes. That is, consider a cost matrix $C_c(y, k)$ where the y-th row of C_c is the classification cost vector $\mathbf{c}_c^{(y)}$. The approach above corresponds to scaling up some rows of C_c by w, indicating that examples that come from those classes are more important.

Another way of saying that class ℓ is important is to prevent mis-classifying examples of other classes as ℓ. For instance, ℓ can be the "non-cancerous" class and it is certainly bad to predict patients carrying any kind of cancer as non-cancerous. Thus, we need a cost matrix that comes from scaling up some *columns* of C_c. Weighted classification cannot deal with such a cost matrix, but cost-sensitive classification can.

In our experiments, we generate costs by first picking a random $\lfloor K/2 \rfloor$ of the columns, and decide whether to scale each of them by 100 with a fair coin flip. Thus, in expectation, $\frac{\lfloor K/2 \rfloor}{2}$ columns of C_c are scaled to form a cost matrix C. Then, for a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$, we take $\mathbf{c}[k] = C(y, k)$. The resulting cost C would be called *emphasizing cost*.

Table 9 compares the average test emphasizing cost of CSOVO with WAP, APFT, FT, TREE, SECOC and CSOVA; Table 10 lists the paired t-test results with significance level 0.05. Similar to the findings when using the RP procedure, the leading algorithms in performance are CSOVO, WAP and CSOVA. They are mostly able to achieve decent performance under the emphasizing cost. In particular, the average cost is usually less than 1, indicat-

Table 9 test emphasizing cost of meta-algorithms that reduce cost-sensitive to binary classification

data set	CSOVO	WAP	APFT	FT	TREE	SECOC	CSOVA
zoo	**0.08±0.01**	**0.07±0.01**	0.10±0.02	0.09±0.01	**0.08±0.01**	0.95±0.19	0.85±0.58
glass	**0.57±0.09**	**0.59±0.09**	0.75±0.19	0.83±0.27	0.98±0.28	1.16±0.20	1.15±0.32
vehicle	0.49±0.03	0.48±0.03	0.50±0.03	0.48±0.03	0.48±0.03	0.71±0.01	**0.43±0.04**
vowel	0.12±0.02	0.11±0.02	0.15±0.03	0.12±0.03	0.17±0.05	0.87±0.02	**0.07±0.01**
yeast	**0.50±0.02**	0.51±0.02	0.51±0.02	**0.49±0.01**	0.60±0.05	0.91±0.01	0.51±0.02
segment	0.19±0.04	**0.16±0.03**	0.17±0.03	0.17±0.03	0.23±0.06	0.83±0.03	**0.14±0.02**
dna	0.27±0.02	0.27±0.02	0.27±0.02	0.28±0.02	0.29±0.02	0.61±0.01	**0.23±0.02**
pageblock	**0.23±0.04**	**0.27±0.06**	**0.24±0.04**	**0.24±0.04**	0.28±0.06	0.83±0.04	0.58±0.15
satimage	**0.24±0.02**	0.29±0.02	**0.25±0.02**	0.37±0.12	0.35±0.08	0.81±0.01	0.28±0.03
usps	**0.11±0.01**	**0.11±0.01**	0.13±0.01	0.13±0.01	0.12±0.01	0.87±0.01	0.12±0.02

Table 10 t-test for comparing CSOVO with other meta-algorithms with emphasizing cost

data set	WAP	APFT	FT	TREE	SECOC	CSOVA
zoo	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\sim$
glass	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$
vehicle	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\sim$
vowel	$\sim$	$\circ$	$\sim$	$\sim$	$\circ$	$\times$
yeast	$\sim$	$\circ$	$\sim$	$\circ$	$\circ$	$\sim$
segment	$\sim$	$\sim$	$\sim$	$\sim$	$\circ$	$\sim$
dna	$\sim$	$\sim$	$\circ$	$\circ$	$\circ$	$\times$
pageblock	$\circ$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$
satimage	$\circ$	$\sim$	$\sim$	$\sim$	$\circ$	$\circ$
usps	$\sim$	$\circ$	$\circ$	$\circ$	$\circ$	$\sim$

$\circ$: CSOVO significantly better; $\times$: CSOVO significantly worse; $\sim$: similar

ing that the serious mis-classification costs (the emphasized ones with cost 100) have been carefully prevented. On the other hand, SECOC often cannot reach the same level of performance; APFT, FT and TREE are also sometimes worse. The results again verify that CSOVO is a superior choice for reducing from cost-sensitive classification to binary classification.

5.5 Comparison on Ordinal Ranking Data Sets

We mentioned in Section 1 that cost-sensitive classification can express any finite-choice and bounded-loss supervised learning setups (Beygelzimer et al, 2005). Next, we demonstrate the usefulness of cost-sensitive classification with one such setup: ordinal ranking. Ordinal ranking can be viewed as a special case of cost-sensitive classification (Lin, 2008) that takes cost vectors of some specific forms. For instance, many existing works on ordinal ranking (Chu and Keerthi, 2007; Li and Lin, 2007) focus on the absolute cost vectors $\mathbf{c}_a^{(y)}[k] = |y - k|$. We adopt these cost vectors in our experiments and assign $\mathbf{c} = \mathbf{c}_a^{(y)}$ for each cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$. We then conduct experiments with eight benchmark ordinal ranking data sets: pyrimdines, machineCPU, boston, abalone, bank, computer, california, census, which were used by Chu and Keerthi (2007). Similar to their original procedure, we kept the same training/test split ratios, and averaged the results over 20 trials.

Table 11 compares the test performance of all the cost-sensitive algorithms on the ordinal ranking data sets; Table 12 lists the t-test results. We see that CSOVO is usually significantly better than the other cost-sensitive classification algorithms. In addition, CSOVO

Table 11 test absolute cost on ordinal ranking data sets (benchmark results from Chu and Keerthi, 2007)

data set	CSOVO	WAP	APFT	FT	TREE	SECOC	CSOVA	ben.
pyr.	1.34±0.05	1.40±0.06	1.52±0.07	1.47±0.05	1.51±0.05	1.62±0.06	1.61±0.07	**1.29±0.05**
mac.	**0.86±0.02**	**0.88±0.02**	0.96±0.03	0.95±0.03	0.97±0.03	1.21±0.02	0.97±0.02	0.99±0.03
bos.	0.80±0.01	0.81±0.01	0.87±0.01	0.90±0.02	0.93±0.02	1.14±0.02	0.95±0.02	**0.75±0.01**
aba.	1.42±0.00	1.48±0.01	1.51±0.01	1.54±0.01	1.57±0.02	1.70±0.00	1.68±0.01	**1.36±0.00**
ban.	1.41±0.00	1.45±0.00	1.55±0.01	1.67±0.02	1.65±0.03	1.74±0.00	1.81±0.00	**1.39±0.00**
com.	**0.58±0.00**	0.59±0.00	0.63±0.00	0.66±0.01	0.67±0.01	0.96±0.00	0.64±0.00	0.60±0.00
cal.	**0.95±0.00**	0.98±0.00	1.05±0.00	1.09±0.01	1.07±0.01	1.29±0.00	1.12±0.00	1.01±0.00
cen.	**1.13±0.00**	1.16±0.00	1.25±0.00	1.30±0.01	1.28±0.01	1.42±0.00	1.33±0.00	1.21±0.00

Table 12 t-test for comparing CSOVO with other meta-algorithms on ordinal ranking data sets

data set	WAP	APFT	FT	TREE	SECOC	CSOVA
pyr.	○	○	○	○	○	○
mac.	∼	○	○	○	○	○
bos.	∼	○	○	○	○	○
aba.	○	○	○	○	○	○
ban.	○	○	○	○	○	○
com.	○	○	○	○	○	○
cal.	○	○	○	○	○	○
cen.	○	○	○	○	○	○

○: CSOVO significantly better; ×: CSOVO significantly worse; ∼: similar

performs similarly to the benchmark SVOR-IMC (Chu and Keerthi, 2007) algorithm. The results further justify the validity and the usefulness of CSOVO.

5.6 Comparison with Linear SVM

Next, we take SVM with the linear kernel (linear SVM) as $\mathcal{A}_b$. Except for the change of kernel, all the other experiment procedures (including parameter selection) are kept the same. Because the dual problem solver in LIBSVM (Chang and Lin, 2001) can be slow when using the linear kernel, we only conduct the experiments with five smaller data sets for the RP/emphasizing costs and four smaller data sets for the absolute cost (ordinal ranking).

Table 13 compares the test performance of all the cost-sensitive classifications algorithms using the linear SVM; Table 14 lists the t-test results. We see that CSOVO remains to be a leading choice across different data sets and different cost-generating procedures. Nevertheless, unlike the results in the previous subsections, APFT becomes a tough competitor in RP and emphasizing costs, better than FT and similar to CSOVO and WAP. The results echo the finding in the original APFT paper (Beygelzimer et al, 2007) that APFT is a promising choice when using linear classifiers. A possible explanation is that APFT uses fewer examples per binary classifier, and thus may be able to sustain generalization when using weaker classifiers like linear SVM; when using nonlinear SVM, however, APFT may overfit the few examples and achieve worse performance.

When comparing the average test costs when using linear SVM (Table 13) with those when using SVM with the perceptron kernel (Tables 7, 9 and 11), we see that the latter usually leads to better performance. The results suggest that CSOVO plus SVM with the perceptron kernel can be a preferred first-hand choice for solving cost-sensitive multiclass classification tasks.

Table 13 test cost of meta-algorithms that reduce to linear SVM

data set	CSOVO	WAP	APFT	FT	TREE	SECOC	CSOVA
				RP cost			
zoo	**64.56±18.71**	81.15±25.64	79.21±21.79	**61.49±12.49**	**67.16±15.21**	154.71±19.83	76.65±20.69
glass	**221.96±19.96**	**218.95±20.48**	**236.29±22.18**	**216.16±21.32**	246.07±21.28	315.07±28.03	271.44±28.44
vehicle	**169.74±18.34**	**162.04±18.25**	**160.02±18.46**	180.01±19.65	182.42±19.60	195.52±20.66	**157.37±16.79**
vowel	432.18±12.80	415.66±15.08	**264.36±11.98**	429.45±14.24	513.13±23.00	504.16±14.80	384.79±13.16
yeast	**60.70±6.20**	**62.19±6.51**	**59.86±6.35**	**62.23±6.83**	74.29±9.45	304.65±77.78	101.52±15.09
				emphasizing cost			
zoo	**0.11±0.02**	**0.10±0.01**	0.70±0.56	0.88±0.58	0.29±0.19	1.16±0.25	0.48±0.37
glass	**0.64±0.09**	**0.64±0.09**	**0.63±0.11**	**0.65±0.09**	0.82±0.13	0.90±0.08	0.96±0.29
vehicle	**0.46±0.04**	**0.44±0.03**	0.56±0.06	0.48±0.06	0.50±0.05	0.79±0.05	0.50±0.07
vowel	0.34±0.02	0.35±0.02	**0.32±0.01**	0.60±0.01	0.63±0.02	0.89±0.00	0.58±0.01
yeast	**0.52±0.03**	**0.52±0.03**	**0.51±0.02**	0.54±0.03	0.58±0.04	0.92±0.01	0.60±0.01
				absolute cost			
pyrimdines	**1.44±0.04**	1.49±0.07	1.58±0.09	1.56±0.06	1.66±0.07	1.75±0.07	1.88±0.08
machineCPU	**0.91±0.02**	0.98±0.02	0.94±0.02	1.10±0.03	1.24±0.03	1.38±0.03	1.34±0.03
boston	**0.88±0.01**	0.90±0.01	0.94±0.01	1.11±0.02	1.15±0.03	1.37±0.02	1.25±0.02
abalone	**1.40±0.00**	1.45±0.00	1.42±0.00	1.54±0.01	1.62±0.02	1.73±0.00	1.86±0.01

Table 14 t-test for comparing CSOVO with other meta-algorithms coupled with Linear SVM

data set	WAP	APFT	FT	TREE	SECOC	CSOVA
			RP cost			
zoo	∼	∼	∼	∼	○	∼
glass	∼	∼	∼	∼	○	○
vehicle	×	×	○	○	○	∼
vowel	×	×	∼	○	○	×
yeast	∼	∼	∼	○	○	○
			emphasizing cost			
zoo	∼	∼	∼	∼	○	∼
glass	∼	∼	∼	○	○	∼
vehicle	∼	∼	∼	∼	○	∼
vowel	∼	×	○	○	○	○
yeast	∼	∼	○	○	○	○
			absolute cost			
pyrimdines	∼	○	○	○	○	○
machine	○	○	○	○	○	○
housing	○	○	○	○	○	○
abalone	○	○	○	○	○	○

○: CSOVO significantly better; ×: CSOVO significantly worse; ∼: similar

6 Conclusion

We presented the cost-transformation technique, which can transform any cost vector $\mathbf{c}$ to a similar one that is decomposable to the classification cost C_c with the minimum entropy. The technique allowed us to design the TSEW algorithm, which can be generally applied to make any regular classification algorithm cost-sensitive. We coupled TSEW with the popular OVO meta-algorithm, and obtained a novel CSOVO algorithm that can conquer cost-sensitive classification by reducing it to several binary classification tasks. Experimental results demonstrated that CSOVO can be significantly better than the original OVO for cost-sensitive classification, which justified the usefulness of CSOVO.

We also analyzed the theoretical guarantee of CSOVO, and discussed its similarity to the existing WAP algorithm. We conducted a thorough experimental study that compared CSOVO with not only WAP but also many major meta-algorithms that reduce cost-sensitive classification to binary classification. We empirically found that CSOVO is similar to WAP on general cost-sensitive classification data sets, and is usually better than WAP on ordinal ranking data sets. These experimental results, along with the relative simplicity and efficiency of CSOVO, make it the preferred OVO-type algorithm for cost-sensitive classification tasks. In addition, CSOVO performs better than other major meta-algorithms on many cost-sensitive classification and ordinal ranking data sets. Thus, when the number of classes is not too large, CSOVO is the best meta-algorithm from cost-sensitive to binary classification.

While CSOVO can perform well for cost-sensitive classification, it does not scale well with K, the number of classes. Applying the cost-transformation technique to design more efficient cost-sensitive classification algorithms will be an important future research direction.

Acknowledgments

The author thanks Yaser Abu-Mostafa, Amrit Pratap, Ling Li, Shou-de Lin, John Langford and Alina Beygelzimer for their valuable comments. The author is also grateful to the NTU Computer and Information Networking Center for the support of high-performance computing facilities. This work was partially supported by the National Science Council of Taiwan via NSC 98-2221-E-002-192.

References

Abe N, Zadrozny B, Langford J (2004) An iterative method for multi-class cost-sensitive learning. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 3–11

Beygelzimer A, Dani V, Hayes T, Langford J, Zadrozny B (2005) Error limiting reductions between classification tasks. In: Machine Learning: Proceedings of the 22rd International Conference, ACM, pp 49–56

Beygelzimer A, Langford J, Ravikumar P (2007) Multiclass classification with filter trees, downloaded from `http://hunch.net/~jl`

Chang CC, Lin CJ (2001) LIBSVM: A Library for Support Vector Machines. National Taiwan University, software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

Chu W, Keerthi SS (2007) Support vector ordinal regression. Neural Computation 19(3):792–815

Domingos P (1999) MetaCost: A general method for making classifiers cost-sensitive. In: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 155–164

Hettich S, Blake CL, Merz CJ (1998) UCI repository of machine learning databases. Downloadable at `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Hsu CW, Lin CJ (2002) A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks 13(2):415–425

Hsu CW, Chang CC, Lin CJ (2003) A practical guide to support vector classification. Tech. rep., National Taiwan University

Hull JJ (1994) A database for handwritten text recognition research. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(5):550–554

Langford J, Beygelzimer A (2005) Sensitive error correcting output codes. In: Learning Theory: 18th Annual Conference on Learning Theory, Springer-Verlag, pp 158–172

Li L, Lin HT (2007) Ordinal regression by extended binary classification. In: Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference, MIT Press, vol 19, pp 865–872

Lin HT (2008) From ordinal ranking to binary classification. PhD thesis, California Institute of Technology

Lin HT, Li L (2008) Support vector machinery for infinite ensemble learning. Journal of Machine Learning Research 9:285–312

Margineantu DD (2001) Methods for cost-sensitive learning. PhD thesis, Oregon State University

Xia F, Zhou L, Yang Y, Zhang W (2007) Ordinal regression as multiclass classification. International Journal of Intelligent Control and Systems 12(3):230–236

Zadrozny B, Langford J, Abe N (2003) Cost sensitive learning by cost-proportionate example weighting. In: Proceedings of the 3rd IEEE International Conference on Data Mining, IEEE Computer Society, pp 435–442

Zhou ZH, Liu XY (2006) On multi-class cost-sensitive learning. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI '06), pp 567–572