Reduction from Cost-sensitive Multiclass Classification to One-versus-one Binary Classification

Hsuan-Tien Lin HTLIN@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University

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).

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 somewhat simpler to implement than WAP. Our experimental results on real-world data sets demonstrate that CSOVO shares a similar (and sometimes even better) 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, 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*

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

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* $\mathbf{c}_c^{(\ell)}[k] \equiv [\![\ell \ne k]\!]$. 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.

Theorem 5 (decomposition via a similar vector) *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*

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

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.

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

The proof is based on 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 5, 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.

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

Theorem 6 (decomposition with minimum-entropy) *If not all* $\mathbf{c}[\ell]$ *are equal, the unique optimal solution to* (1) *is*

$$\begin{aligned} \tilde{\mathbf{q}}[\ell] &= c_{\max} - \mathbf{c}[\ell]\,, \qquad (2) \\ \Delta &= (K-1)\, c_{\max} - \sum_{k=1}^{K} \mathbf{c}[k]\,. \qquad (3) \end{aligned}$$

The proof is listed in Appendix A. Note that the resulting Δ is the smallest one that makes $\tilde{\mathbf{c}}$ decomposable to C_c. Using Theorem 6, 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.

Theorem 7 (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).

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

Theorem 8 (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 8 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 (weighted) regular classification using Algorithm 1, called training set expansion and weighting (TSEW). The algorithm carries the information in $\tilde{\mathbf{q}}[\ell]$ as weights of multi-labelled examples.

Algorithm 1: 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 $\ell$, 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 $\hat{g} \equiv \hat{g}_w$.
```

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 7. 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 $\mathcal{S}_w$ 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 2, 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 2: One-versus-one (Hsu and Lin, 2002)

```
1. For each $i, j$ that $1 \le i < j \le K$,
    (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 $\hat{g}(\mathbf{x}) = \operatorname*{argmax}_{1 \le \ell \le K} \sum_{i<j} \left[\!\left[ \hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.
```

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 $E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\text{OVO}}^{(i,j)}\right)$, where

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

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

$$E(\hat{g}, \mathcal{D}_r) \le 2 \sum_{i<j} E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\text{OVO}}^{(i,j)}\right).$$

That is, if $E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\text{OVO}}^{(i,j)}\right)$ 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 1), we can easily get a preliminary version of CSOVO in Algorithm 3.

Algorithm 3: TSEW-OVO

```
1. For each $i, j$ that $1 \le i < j \le K$,
    (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 $\hat{g}(\mathbf{x}) = \underset{1 \le \ell \le K}{\operatorname{argmax}} \sum_{i<j} \left[\!\left[ \hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.
```

One thing to notice in Algorithm 3 is that each training example $(\mathbf{x}_n, y_n)$ may be split to two examples $\left(\mathbf{x}_n, i, w_n^{(i)}\right)$ and $\left(\mathbf{x}_n, j, w_n^{(j)}\right)$ 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 $\left| w_n^{(i)} - w_n^{(j)} \right|$. 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.

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.

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 4) takes OVO (Algorithm 2) 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 4. Note that each created example

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

Algorithm 4: Cost-sensitive One-versus-one

1. For each i, j that $1 \le i < j \le K$,
   (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 $\hat{g}(\mathbf{x}) = \operatorname*{argmax}_{1 \le \ell \le K} \sum_{i<j} \left[\!\left[\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.

can be thought as if coming from a distribution

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

We then get the following theorem:

Theorem 9 *Consider any family of binary classifiers*

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

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

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

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

$$g_b^{(i,j)}(\mathbf{x}) \ne \operatorname*{argmin}_{\ell = i \text{ or } j} \mathbf{c}_n[\ell] \,.$$

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

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

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 $E\left(\hat{g}_b^{(i,j)}, \mathcal{D}_{\mathsf{CSOVO}}^{(i,j)}\right)$ 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 9 was made by Beygelzimer et al. (2005) to analyze another algorithm called weighted all-pairs (WAP). As illustrated in Algorithm 5, 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 5: A Special Version of WAP (Beygelzimer et al., 2005)
Run CSOVO, while replacing (4) in step 1(a) with

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

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

Define

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

where $\mathbf{v}$ is computed from $\mathbf{c}$ using a similar definition as the one in Algorithm 5. 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 9). 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

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

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

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

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

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

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.

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 use ten classification data sets: zoo, glass, vehicle, vowel, yeast, segment, dna, pageblock, satimage, usps.[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 $\left[0, 2000\frac{|\{n\colon y_n=k\}|}{|\{n\colon y_n=y\}|}\right]$. 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 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

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

Table 1: test cost of CSOVO/OVO

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

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.

5.1. CSOVO versus OVO

Table 1 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 4, 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. We check and find that OVO achieves 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 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 2. We see that CSOVO and WAP are comparable in performance, with WAP being

Table 2: 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

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 2 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 4 lead to the same conclusion.

In summary, CSOVO performs better than APFT; CSOVO performs similarly to WAP but enjoys a simpler and 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), and Sensitive Error Correcting Output Codes (SECOC; Langford and Beygelzimer, 2005). The algorithms cover major types of decompositions from multiclass classification to binary classification.

Table 3 compares the average test RP cost of CSOVO, TREE, FT, and SECOC; Table 4 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). Overall, Table 3 and Table 4 suggest that CSOVO is the best meta-algorithm of the four.

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

Table 3: test cost of meta-algorithms that reduce cost-sensitive to binary classification

data set	CSOVO	FT	TREE	SECOC
zoo	**36.79±9.09**	74.01±27.65	57.07±17.40	179.02±30.52
glass	**210.06±15.88**	**212.76±19.38**	264.02±24.49	347.77±37.87
vehicle	**155.14±20.63**	**156.01±20.14**	**156.72±20.12**	**167.60±20.97**
vowel	20.05±1.95	24.66±2.92	28.42±3.02	95.25±7.35
yeast	**52.45±2.97**	**53.73±3.14**	66.49±6.09	277.14±49.41
segment	**25.27±2.25**	**27.06±2.32**	27.76±2.26	68.08±4.02
dna	53.18±4.25	53.76±4.23	55.32±4.39	65.90±5.66
pageblock	**24.98±4.94**	29.93±6.16	**23.00±3.28**	249.28±67.31
satimage	**66.57±4.77**	74.01±4.57	78.13±4.31	102.81±5.41
usps	**20.51±1.17**	27.07±1.52	25.74±1.55	86.59±9.45

Table 4: t-test for comparing CSOVO with other meta-algorithms using RP cost

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

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

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 but performs better than other major meta-algorithms on many cost-sensitive classification data sets. The results make CSOVO a promising 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.

References

Naoki Abe, Bianca Zadrozny, and John Langford. An iterative method for multi-class cost-sensitive learning. In *KDD*, pages 3–11. ACM, 2004.

Alina Beygelzimer, Varsha Dani, Tom Hayes, John Langford, and Bianca Zadrozny. Error limiting reductions between classification tasks. In *ICML*, pages 49–56. ACM, 2005.

Alina Beygelzimer, John Langford, and Pradeep Ravikumar. Multiclass classification with filter trees. Downloaded from http://hunch.net/~jl, 2007.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines*. National Taiwan University, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Pedro Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *KDD*, pages 155–164. ACM, 1999.

Seth Hettich, Catherine L. Blake, and Christopher J. Merz. UCI repository of machine learning databases, 1998. Downloadable at http://www.ics.uci.edu/~mlearn/MLRepository.html.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, National Taiwan University, 2003.

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

John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In *COLT*, pages 158–172. Springer-Verlag, 2005.

Hsuan-Tien Lin and Ling Li. Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research*, 9:285–312, 2008.

Dragos Dorin Margineantu. *Methods for Cost-Sensitive Learning*. PhD thesis, Oregon State University, 2001.

Fen Xia, Liang Zhou, Yanwu Yang, and Wensheng Zhang. Ordinal regression as multiclass classification. *International Journal of Intelligent Control and Systems*, 12(3):230–236, 2007.

Bianca Zadrozny, John Langford, and Naoki Abe. Cost sensitive learning by cost-proportionate example weighting. In *ICDM*, pages 435–442, 2003.

Zhi-Hua Zhou and Xu-Ying Liu. On multi-class cost-sensitive learning. In *AAAI*, pages 567–572, 2006.

Appendix A. Proof of Theorem 6

Proof If not all $\mathbf{c}[\ell]$ are equal, not all $\mathbf{q}[\ell]$ are equal. Now we substitute those $\tilde{\mathbf{p}}$ in the objective function by the right-hand sides of the equality constraints. Then, the objective function becomes

$$f(\Delta) = -\sum_{\ell=1}^{K} \frac{\mathbf{q}[\ell]+\Delta}{\sum_{k=1}^{K}\mathbf{q}[k]+K\Delta} \log \frac{\mathbf{q}[\ell]+\Delta}{\sum_{k=1}^{K}\mathbf{q}[k]+K\Delta} .$$

The constraint on Δ ensures that all the $p \log p$ operations above are well defined.[4] Now, let $\bar{q} \equiv \frac{1}{K}\sum_{k=1}^{K}\mathbf{q}[k]$. We get

$$\begin{aligned}
\frac{df}{d\Delta} &= -\frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\left(-\mathbf{q}[\ell]+\bar{q}\right)\cdot\left(\log\left(\frac{\mathbf{q}[\ell]+\Delta}{\bar{q}+\Delta}\right)-\log K+1\right)\\
&= -\frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\left(-\underbrace{(\mathbf{q}[\ell]+\Delta)}_{a_\ell}+\underbrace{(\bar{q}+\Delta)}_{b_\ell}\right)\cdot\log\frac{\mathbf{q}[\ell]+\Delta}{\bar{q}+\Delta}\\
&= \frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\left(a_\ell-b_\ell\right)\cdot\left(\log a_\ell-\log b_\ell\right).
\end{aligned}$$

When not all $\mathbf{q}[\ell]$ are equal, there exists at least one a_ℓ that is not equal to b_ℓ. Therefore, $\frac{df}{d\Delta}$ is strictly positive, and hence the unique minimum of $f(\Delta)$ happens when Δ is of the smallest possible value. That is, for the unique optimal solution,

$$\begin{cases}\Delta = \max\limits_{1\le\ell\le K}\left(-\mathbf{q}[\ell]\right) = c_{\max} - \left(\frac{1}{K-1}\sum_{k=1}^{K}\mathbf{c}[k]\right) \;; \\ \tilde{\mathbf{q}}[\ell] = c_{\max} - \mathbf{c}[\ell]\,,\ \tilde{\mathbf{p}}[\ell] = \frac{c_{\max}-\mathbf{c}[\ell]}{\sum_{k=1}^{K}(c_{\max}-\mathbf{c}[k])}.\end{cases}$$

■

4. We take the convention that $0\log 0 \equiv \lim_{\epsilon\to 0}\epsilon\log\epsilon = 0$.