Cost-Sensitive Label Embedding for Multi-Label Classification

Kuan-Hao Huang · Hsuan-Tien Lin

Abstract Label embedding (LE) is an important family of multi-label classification algorithms that digest the label information jointly for better performance. Different real-world applications evaluate performance by different cost functions of interest. Current LE algorithms often aim to optimize one specific cost function, but they can suffer from bad performance with respect to other cost functions. In this paper, we resolve the performance issue by proposing a novel cost-sensitive LE algorithm that takes the cost function of interest into account. The proposed algorithm, cost-sensitive label embedding with multidimensional scaling (CLEMS), approximates the cost information with the distances of the embedded vectors by using the classic multidimensional scaling approach for manifold learning. CLEMS is able to deal with both symmetric and asymmetric cost functions, and effectively makes cost-sensitive decisions by nearest-neighbor decoding within the embedded vectors. We derive theoretical results that justify how CLEMS achieves the desired cost-sensitivity. Furthermore, extensive experimental results demonstrate that CLEMS is significantly better than a wide spectrum of existing LE algorithms and state-of-the-art cost-sensitive algorithms across different cost functions.

1 Introduction

The multi-label classification problem (MLC), which allows multiple labels to be associated with each example, is an extension of the multi-class classification problem. The MLC problem satisfies the demands of many real-world applications (Carneiro et al, 2007; Trohidis et al, 2008; Barutçuoglu et al, 2006). Different applications usually need different criteria to evaluate the prediction performance of MLC algorithms. Some popular criteria are Hamming loss, Rank loss, F1 score, and Accuracy score (Tsoumakas et al, 2010; Madjarov et al, 2012).

Kuan-Hao Huang, Hsuan-Tien Lin
CSIE Department, National Taiwan University, Taipei, Taiwan
E-mail: {r03922062, htlin}@csie.ntu.edu.tw

Label embedding (LE) is an important family of MLC algorithms that jointly extract the information of all labels to improve the prediction performance. LE algorithms automatically transform the original labels to an embedded space, which represents the hidden structure of the labels. After conducting learning within the embedded space, LE algorithms make more accurate predictions with the help of the hidden structure.

Existing LE algorithms can be grouped into two categories based on the dimension of the embedded space: label space dimension reduction (LSDR) and label space dimension expansion (LSDE). LSDR algorithms (Hsu et al, 2009; Kapoor et al, 2012; Tai and Lin, 2012; Sun et al, 2011; Chen and Lin, 2012; Yu et al, 2014; Lin et al, 2014; Balasubramanian and Lebanon, 2012; Bi and Kwok, 2013; Bhatia et al, 2015; Yeh et al, 2017) consider a low-dimensional embedded space for digesting the information between labels and conduct more effective learning. In contrast to LSDR algorithms, LSDE algorithms (Zhang and Schneider, 2011; Ferng and Lin, 2013; Tsoumakas et al, 2011a) focus on a high-dimensional embedded space. The additional dimensions can then be used to represent different angles of joint information between the labels to reach better performance.

While LE algorithms have become major tools for tackling the MLC problem, most existing LE algorithms are designed to optimize only one or few specific criteria. The algorithms may then suffer from bad performance with respect to other criteria. Given that different applications demand different criteria, it is thus important to achieve cost (criterion) sensitivity to make MLC algorithms more realistic. Cost-sensitive MLC (CSMLC) algorithms consider the criterion as an additional input, and take it into account either in the training or the predicting stage. The additional input can then be used to guide the algorithm towards more realistic predictions. CSMLC algorithms are attracting research attention in recent years (Lo et al, 2011, 2014; Dembczynski et al, 2010, 2011; Li and Lin, 2014), but to the best of our knowledge, there is no work on cost-sensitive label embedding (CSLE) algorithms yet.

In this paper, we study the design of CSLE algorithms, which take the intended criterion into account in the training stage to locate a cost-sensitive hidden structure in the embedded space. The cost-sensitive hidden structure can then be used for more effective learning and more accurate predictions with respect to the criterion of interest. Inspired by the finding that many of the existing LSDR algorithms can be viewed as linear manifold learning approaches, we propose to adopt manifold learning for CSLE. Nevertheless, to embed any general and possibly complicated criterion, linear manifold learning may not be sophisticated enough. We thus start with multidimensional scaling (MDS), one famous non-linear manifold learning approach, to propose a novel CSLE algorithm. The proposed cost-sensitive label embedding with multidimensional scaling (CLEMS) algorithm embeds the cost information within the distance measure of the embedded space. We further design a *mirroring trick* for CLEMS to properly embed the possibly asymmetric criterion information within the symmetric distance measure. We also design an efficient procedure that conquers the difficulty of making predictions through the non-linear cost-sensitive hidden structure. Theoretical results justify that CLEMS achieves cost-sensitivity through learning in the MDS-embedded space. Extensive empirical results demonstrate that CLEMS usually reaches better performance than leading LE algorithms across different criteria. In addition, CLEMS also performs better than the state-of-the-art CSMLC algorithms (Li and Lin, 2014;
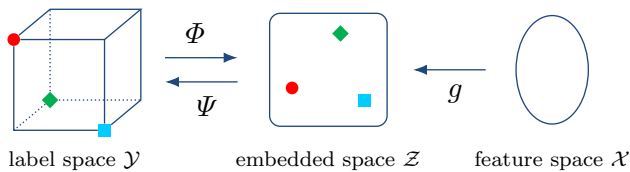
Fig. 1 Flow of label embedding

Dembczynski et al, 2010, 2011). The results suggest that CLEMS is a promising algorithm for CSMLC.

This paper is organized as follows. Section 2 formalizes the CSLE problem and Section 3 illustrates the proposed algorithm along with theoretical justifications. We discuss the experimental results in Section 4 and conclude in Section 5.

2 Cost-sensitive label embedding

In multi-label classification (MLC), we denote the feature vector of an instance by $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ and denote the label vector by $\mathbf{y} \in \mathcal{Y} \subseteq \{0,1\}^K$ where $\mathbf{y}[i] = 1$ if and only if the instance is associated with the i-th label. Given the training instances $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$, the goal of MLC algorithms is to train a predictor $h \colon \mathcal{X} \to \mathcal{Y}$ from $\mathcal{D}$ in the training stage, with the expectation that for any unseen testing instance $(\mathbf{x}, \mathbf{y})$, the prediction $\tilde{\mathbf{y}} = h(\mathbf{x})$ can be close to the ground truth $\mathbf{y}$.

A simple criterion for evaluating the closeness between $\mathbf{y}$ and $\tilde{\mathbf{y}}$ is *Hamming* $loss(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{K} \sum_{i=1}^K [\![\mathbf{y}[i] \neq \tilde{\mathbf{y}}[i]]\!]$. It is worth noting that Hamming loss separately evaluates each label component of $\tilde{\mathbf{y}}$. There are other criteria that jointly evaluate all the label components of $\tilde{\mathbf{y}}$, such as F1 score, Rank loss, 0/1 loss, and Accuracy score (Tsoumakas et al, 2010; Madjarov et al, 2012).

Arguably the simplest algorithm for MLC is binary relevance (BR) (Tsoumakas and Katakis, 2007). BR separately trains a binary classifier for each label without considering the information of other labels. In contrast to BR, label embedding (LE) is an important family of MLC algorithms that *jointly* use the information of all labels to achieve better prediction performance. LE algorithms try to identify the hidden structure behind the labels. In the training stage, instead of training a predictor h directly, LE algorithms first embed each K-dimensional label vector $\mathbf{y}^{(n)}$ as an M-dimensional embedded vector $\mathbf{z}^{(n)} \in \mathcal{Z} \subseteq \mathbb{R}^M$ by an embedding function $\Phi \colon \mathcal{Y} \to \mathcal{Z}$. The embedded vector $\mathbf{z}^{(n)}$ can be viewed as the hidden structure that contains the information pertaining to all labels. Then, the algorithms train a internal predictor $g \colon \mathcal{X} \to \mathcal{Z}$ from $\{(\mathbf{x}^{(n)}, \mathbf{z}^{(n)})\}_{n=1}^N$. In the predicting stage, for the testing instance $\mathbf{x}$, LE algorithms obtain the predicted embedded vector $\tilde{\mathbf{z}} = g(\mathbf{x})$ and use a decoding function $\Psi \colon \mathcal{Z} \to \mathcal{Y}$ to get the prediction $\tilde{\mathbf{y}}$. In other words, LE algorithms learn the predictor by $h = \Psi \circ g$. Figure 1 illustrates the flow of LE algorithms.

Existing LE algorithms can be grouped into two categories based on M (the dimension of $\mathcal{Z}$) and K (the dimension of $\mathcal{Y}$). LE algorithms that work with $M \leq K$ are termed as label space dimension reduction (LSDR) algorithms. They consider a low-dimensional embedded space for digesting the information between labels and utilize different pairs of (Φ, Ψ) to conduct more effective learning. Compressed

sensing (Hsu et al, 2009) and Bayesian compressed sensing (Kapoor et al, 2012) consider a random projection as Φ and obtain Ψ by solving an optimization problem per test instance. Principal label space transformation (Tai and Lin, 2012) considers Φ calculated from an optimal linear projection of the label vectors and derives Ψ accordingly. Some other works also consider optimal linear projections as Φ but take feature vectors into account in the optimality criterion, including canonical-correlation-analysis methods (Sun et al, 2011), conditional principal label space transformation (Chen and Lin, 2012), low-rank empirical risk minimization for multi-label learning (Yu et al, 2014), and feature-aware implicit label space encoding (Lin et al, 2014). Canonical-correlated autoencoder (Yeh et al, 2017) extends the linear projection works by using neural networks instead. Landmark selection method (Balasubramanian and Lebanon, 2012) and column subset selection (Bi and Kwok, 2013) design Φ to select a subset of labels as embedded vectors and derive the corresponding Ψ. Sparse local embeddings for extreme classification (Bhatia et al, 2015) trains a locally-linear projection as Φ and constructs Ψ by nearest neighbors. The smaller M in LSDR algorithms allows the internal predictor g to be learned more efficiently and effectively.

Other LE algorithms work with $M > K$, which are called label space dimension expansion (LSDE) algorithms. Canonical-correlation-analysis output codes (Zhang and Schneider, 2011) design Φ based on canonical correlation analysis to generate additional output codes to enhance the performance. Error-correcting codes (ECC) algorithms (Ferng and Lin, 2013) utilize the encoding and decoding functions of standard error-correcting codes for communication as Φ and Ψ, respectively. Random k-labelsets (Tsoumakas et al, 2011a), a popular algorithm for MLC, can be considered as an ECC-based algorithm with the repetition code (Ferng and Lin, 2013). LSDE algorithms use additional dimensions to represent different angles of joint information between the labels to reach the better performance.

To the best of our knowledge, all the existing LE algorithms above are designed for one or few specific criteria and may suffer from bad performance with respect to other criteria. For example, the optimality criterion within principal label space transformation (Tai and Lin, 2012) is closely related to Hamming loss. For MLC data with very few non-zero $\mathbf{y}[i]$, which are commonly encountered in real-world applications, optimizing Hamming loss can easily results in all-zero predictions $\tilde{\mathbf{y}}[i]$, which suffer from bad F1 score.

MLC algorithms that take the evaluation criterion into account are called cost-sensitive MLC (CSMLC) algorithms and are attracting research attentions in recent years. CSMLC algorithms take the criterion as an additional input and consider it either in the training or the predicting stage. For any given criterion, CSMLC algorithms can ideally make cost-sensitive predictions with respect to the criterion without extra efforts in algorithm design. Generalized k-labelsets ensemble (Lo et al, 2011, 2014) is extended from random k-labelsets (Tsoumakas et al, 2011a) and digests the criterion by giving appropriate weights to labels. The ensemble algorithm performs well for any weighted Hamming loss but cannot tackle more general criteria that jointly evaluate all the label components, such as F1 score. Two CSMLC algorithms for arbitrary criterion are probabilistic classifier chain (PCC) (Dembczynski et al, 2010, 2011) and condensed filter tree (CFT) (Li and Lin, 2014). PCC is based on estimating the probability of each label and making a Bayes-optimal inference for the evaluation criterion. While PCC can in principle be used for any criterion, it may suffer from computational difficulty un-

less an efficient inference rule for the criterion is designed first. CFT is based on converting the criterion as weights when learning each label. CFT conducts the weight-assignment in a more sophisticated manner than generalized k-labelsets ensemble does, and can hence work with arbitrary criterion. Both PCC and CFT are extended from classifier chain (CC) (Read et al, 2011) and form a chain of labels to utilize the information of the earlier labels in the chain, but they cannot globally find the hidden structure of all labels like LE algorithms.

In this paper, we study the design of cost-sensitive label embedding (CSLE) algorithms that respect the criterion when calculating the embedding function Φ and the decoding function Ψ. We take an initiative of studying CSLE algorithms, with the hope of achieving cost-sensitivity and finding the hidden structure at the same time. More precisely, we take the following CSMLC setting (Li and Lin, 2014). Consider a cost function $c(\mathbf{y}, \tilde{\mathbf{y}})$ which represents the penalty when the ground truth is $\mathbf{y}$ and the prediction is $\tilde{\mathbf{y}}$. We naturally assume that $c(\mathbf{y}, \tilde{\mathbf{y}}) \geq 0$, with value 0 attained if and only if $\mathbf{y}$ and $\tilde{\mathbf{y}}$ are the same. Given training instances $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}$ and the cost function $c(\cdot, \cdot)$, CSLE algorithms learn an embedding function Φ, a decoding function Ψ, and an internal predictor g, based on both the training instance $\mathcal{D}$ and the cost function $c(\cdot, \cdot)$. The objective of CSLE algorithms is to minimize the expected cost $c(\mathbf{y}, h(\mathbf{x}))$ for any unseen testing instance $(\mathbf{x}, \mathbf{y})$, where $h = \Psi \circ g$.

3 Proposed algorithm

We first discuss the difficulties of directly extending state-of-the-art LE algorithms for CSLE. In particular, the decoding function Ψ of many existing algorithms, such as conditional principal label space transformation (Chen and Lin, 2012) and feature-aware implicit label space encoding (Lin et al, 2014), are derived from Φ and can be divided into two steps. The first step is using some $\psi \colon \mathcal{Z} \to \mathbb{R}^{K}$ that corresponds to Φ to decode the embedded vector $\mathbf{z}$ to a real-valued vector $\hat{\mathbf{y}} \in \mathbb{R}^{K}$; the second step is choosing a threshold to transform $\hat{\mathbf{y}}$ to $\tilde{\mathbf{y}} \in \{0, 1\}^{K}$. If the embedding function Φ is a linear function, the corresponding ψ can be efficiently computed by pseudo-inverse. However, for complicated cost functions, a linear function may not be sufficient to properly embed the cost information. On the other hand, if the embedding function Φ is a non-linear function, such as those within kernel principal component analysis (Schölkopf et al, 1998) and kernel dependency estimation (Weston et al, 2002), ψ is often difficult to derive or time-consuming in calculation, which then makes Ψ practically infeasible to compute.

To resolve the difficulties, we do not consider the two-step decoding function Ψ that depends on deriving ψ from Φ. Instead, we first fix a decent decoding function Ψ and then derive the corresponding embedding function Φ. We realize that the goal of Ψ is simply to locate the most probable label vector $\tilde{\mathbf{y}}$ from $\mathcal{Y}$, which is of a finite cardinality, based on the predicted embedded vector $\tilde{\mathbf{z}} = g(\mathbf{x}) \in \mathcal{Z}$. If all the embedded vectors are sufficiently far away from each other in $\mathcal{Z}$, one natural decoding function is to calculate the nearest neighbor $\mathbf{z}_q$ of $\tilde{\mathbf{z}}$ and return the corresponding $\mathbf{y}_q$ as $\tilde{\mathbf{y}}$. Such a nearest-neighbor decoding function Ψ is behind some ECC-based LSDE algorithms (Ferng and Lin, 2013) and will be adopted.

The nearest-neighbor decoding function Ψ is based on the distance measure of $\mathcal{Z}$, which matches our primary need of representing the cost information. In

particular, if $\mathbf{y}_i$ is a lower-cost prediction than $\mathbf{y}_j$ with respect to the ground truth $\mathbf{y}_t$, we hope that the corresponding embedded vector $\mathbf{z}_i$ would be closer to $\mathbf{z}_t$ than $\mathbf{z}_j$. Then, even if g makes a small error such that $\tilde{\mathbf{z}} = g(\mathbf{x})$ deviates from the desired $\mathbf{z}_t$, nearest-neighbor decoding function Ψ can decode to the lower-cost $\mathbf{y}_i$ as $\tilde{\mathbf{y}}$ instead of $\mathbf{y}_j$. In other words, for any two label vectors $\mathbf{y}_i, \mathbf{y}_j \in \mathcal{Y}$ and the corresponding embedded vectors $\mathbf{z}_i, \mathbf{z}_j \in \mathcal{Z}$, we want the Euclidean distance between $\mathbf{z}_i$ and $\mathbf{z}_j$, which is denoted by $d(\mathbf{z}_i, \mathbf{z}_j)$, to preserve the magnitude-relationship of the cost $c(\mathbf{y}_i, \mathbf{y}_j)$.

Based on this objective, the framework of the proposed algorithm is as follows. In the training stage, for each label vector $\mathbf{y}_i \in \mathcal{Y}$, the proposed algorithm determines an embedded vector $\mathbf{z}_i$ such that the distance between two embedded vectors $d(\mathbf{z}_i, \mathbf{z}_j)$ in $\mathcal{Z}$ approximates the transformed cost $\delta(c(\mathbf{y}_i, \mathbf{y}_j))$, where $\delta(\cdot)$ is a monotonic transform function to preserve the magnitude-relationship and will be discussed later. We let the embedding function Φ be the mapping $\mathbf{y}_i \to \mathbf{z}_i$ and use $\mathcal{Q}$ to represent the embedded vector set $\{\Phi(\mathbf{y}_i) \mid \mathbf{y}_i \in \mathcal{Y}\}$. Then the algorithm trains a regressor $g\colon \mathcal{X} \to \mathcal{Z}$ as the internal predictor.

In the predicting stage, when receiving a testing instance $\mathbf{x}$, the algorithm obtains the predicted embedded vector $\tilde{\mathbf{z}} = g(\mathbf{x})$. Given that the cost information is embedded in the distance, for each $\mathbf{z}_i \in \mathcal{Q}$, the distance $d(\mathbf{z}_i, \tilde{\mathbf{z}})$ can be viewed as the estimated cost if the underlying truth is $\mathbf{y}_i$. Hence the algorithm finds $\mathbf{z}_q \in \mathcal{Q}$ such that the distance $d(\mathbf{z}_q, \tilde{\mathbf{z}})$ is the smallest (the smallest estimated cost), and lets the corresponding $\mathbf{y}_q = \Phi^{-1}(\mathbf{z}_q) = \tilde{\mathbf{y}}$ be the final prediction for $\mathbf{x}$. In other words, we have a nearest-neighbor-based Ψ, with the first step being the determination of the nearest-neighbor of $\tilde{\mathbf{z}}$ and the second step being the utilization of Φ^{-1} to obtain the prediction $\tilde{\mathbf{y}}$.

Three key issues of the framework above are yet to be addressed. The first issue is the determination of the embedded vectors $\mathbf{z}_i$. The second issue is using the symmetric distance measure to embed the possibly asymmetric cost functions where $c(\mathbf{y}_i, \mathbf{y}_j) \neq c(\mathbf{y}_j, \mathbf{y}_i)$. The last issue is the choice of a proper monotonic transform function $\delta(\cdot)$. The issues will be discussed in the following sub-sections.

3.1 Calculating the embedded vectors by multidimensional scaling

As mentioned above, our objective is to determine embedded vectors $\mathbf{z}_i$ such that the distance $d(\mathbf{z}_i, \mathbf{z}_j)$ approximates the transformed cost $\delta(c(\mathbf{y}_i, \mathbf{y}_j))$. The objective can be formally defined as minimizing the *embedding error*

$$(d(\mathbf{z}_i, \mathbf{z}_j) - \delta(c(\mathbf{y}_i, \mathbf{y}_j)))^2.$$

We observe that the transformed cost $\delta(c(\mathbf{y}_i, \mathbf{y}_j))$ can be viewed as the dissimilarity between label vectors $\mathbf{y}_i$ and $\mathbf{y}_j$. Computing an embedding based on the dissimilarity information matches the task of manifold learning, which is able to preserve the information and discover the hidden structure. Based on our discussions above, any approach that solves the manifold learning task can then be taken to solve the CSLE problem. Nevertheless, for CSLE, different cost functions may need different M (the dimension of $\mathcal{Z}$) to achieve a decent embedding. We thus consider manifold learning approaches that can flexibly take M as the parameter, and adopt a classic manifold learning approach called multidimensional scaling (MDS) (Kruskal, 1964).

For a target dimension M, MDS attempts to discover the hidden structure of L_{MDS} objects by embedding their dissimilarities in an M-dimensional target space. The dissimilarity is represented by a symmetric, non-negative, and zero-diagonal dissimilarity matrix $\boldsymbol{\Delta}$, which is an $L_{MDS} \times L_{MDS}$ matrix with $\boldsymbol{\Delta}_{i,j}$ being the dissimilarity between the i-th object and the j-th object. The objective of MDS is to determine target vectors $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{L_{MDS}}$ in the target space to minimize the *stress*, which is defined as $\sum_{i,j} \mathbf{W}_{i,j}(d(\mathbf{u}_i, \mathbf{u}_j) - \boldsymbol{\Delta}_{i,j})^2$, where d denotes the Euclidean distance in the target space, and $\mathbf{W}$ is a symmetric, non-negative, and zero-diagonal matrix that carries the weight $\mathbf{W}_{i,j}$ of each object pair. There are several algorithms available in the literature for solving MDS. A representative algorithm is Scaling by MAjorizing a COmplicated Function (SMACOF) (De Leeuw, 1977), which can iteratively minimize *stress*. The complexity of SMACOF is generally $\mathcal{O}((L_{MDS})^3)$, but there is often room for speeding up with special weight matrices $\mathbf{W}$.

The *embedding error* $(d(\mathbf{z}_i, \mathbf{z}_j) - \delta(c(\mathbf{y}_i, \mathbf{y}_j)))^2$ and the *stress* $(d(\mathbf{u}_i, \mathbf{u}_j) - \boldsymbol{\Delta}_{i,j})^2$ are of very similar form. Therefore, we can view the transformed costs as the dissimilarities of embedded vectors and feed MDS with specific values of $\boldsymbol{\Delta}$ and $\mathbf{W}$ to calculate the embedded vectors to reduce the *embedding error*. Specifically, the relation between MDS and our objective can be described as follows.

i-th object	dissimilarity $\boldsymbol{\Delta}_{i,j}$	target vector $\mathbf{u}_i$	*stress* $(d(\mathbf{u}_i, \mathbf{u}_j) - \boldsymbol{\Delta}_{i,j})^2$
label vector $\mathbf{y}_i$	transformed cost $\delta(c(\mathbf{y}_i, \mathbf{y}_j))$	embedded vector $\mathbf{z}_i$	*embedding error* $(d(\mathbf{z}_i, \mathbf{z}_j) - \delta(c(\mathbf{y}_i, \mathbf{y}_j)))^2$

The most complete embedding would convert all label vectors $\mathbf{y} \in \mathcal{Y} \subseteq \{0, 1\}^K$ to the embedded vectors. Nevertheless, the number of all label vectors is 2^K, which makes solving MDS infeasible. Therefore, we do not consider embedding the entire $\mathcal{Y}$. Instead, we select some representative label vectors as a candidate set $\mathcal{S} \subseteq \mathcal{Y}$, and only embed the label vectors in $\mathcal{S}$. While the use of $\mathcal{S}$ instead of $\mathcal{Y}$ restricts the nearest-neighbor decoding function to only predict from $\mathcal{S}$, it can reduce the computational burden. One reasonable choice of $\mathcal{S}$ is the set of label vectors that appear in the training instances $\mathcal{D}$, which is denoted as $\mathcal{S}_{tr}$. We will show that using $\mathcal{S}_{tr}$ as $\mathcal{S}$ readily leads to promising performance and discuss more about the choice of the candidate set in Section 4.

After choosing $\mathcal{S}$, we can construct $\boldsymbol{\Delta}$ and $\mathbf{W}$ for solving MDS. Let L denote the number of elements in $\mathcal{S}$ and let $\mathbf{C}(\mathcal{S})$ be the transformed cost matrix of $\mathcal{S}$, which is an $L \times L$ matrix with $\mathbf{C}(\mathcal{S})_{i,j} = \delta(c(\mathbf{y}_i, \mathbf{y}_j))$ for $\mathbf{y}_i, \mathbf{y}_j \in \mathcal{S}$. Unfortunately, $\mathbf{C}(\mathcal{S})$ cannot be directly used as the symmetric dissimilarity matrix $\boldsymbol{\Delta}$ because the cost function $c(\cdot, \cdot)$ may be asymmetric ($c(\mathbf{y}_i, \mathbf{y}_j) \neq c(\mathbf{y}_j, \mathbf{y}_i)$). To resolve this difficulty, we propose a *mirroring trick* to construct a symmetric $\boldsymbol{\Delta}$ from $\mathbf{C}(\mathcal{S})$.

3.2 Mirroring trick for asymmetric cost function

The asymmetric cost function implies that each label vector $\mathbf{y}_i$ serves two roles: as the ground truth, or as the prediction. When $\mathbf{y}_i$ serves as the ground truth, we should use $c(\mathbf{y}_i, \cdot)$ to describe the cost behavior. When $\mathbf{y}_i$ serves as the prediction,
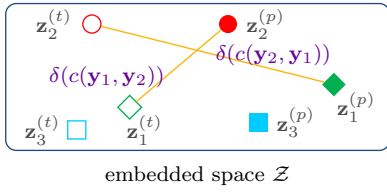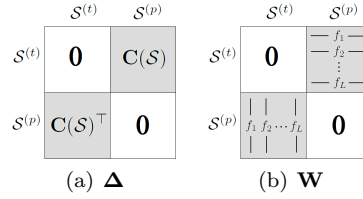
Fig. 2 Embedding cost in distance

(a) $\boldsymbol{\Delta}$ (b) $\mathbf{W}$

Fig. 3 Constructions of $\boldsymbol{\Delta}$ and $\mathbf{W}$

we should use $c(\cdot, \mathbf{y}_i)$ to describe the cost behavior. This motivates us to view these two roles separately.

For each $\mathbf{y}_i \in \mathcal{S}$, we mirror it as $\mathbf{y}_i^{(t)}$ and $\mathbf{y}_i^{(p)}$ to denote viewing $\mathbf{y}_i$ as the ground truth and the prediction, respectively. Note that the two mirrored label vectors $\mathbf{y}_i^{(t)}$ and $\mathbf{y}_i^{(p)}$ are in fact the same, but carry different meanings. Now, we have two roles of the candidate sets $\mathcal{S}^{(t)} = \{\mathbf{y}_i^{(t)}\}_{i=1}^L$ and $\mathcal{S}^{(p)} = \{\mathbf{y}_i^{(p)}\}_{i=1}^L$. Then, as illustrated by Figure 2, $\delta(c(\mathbf{y}_i, \mathbf{y}_j))$, the transformed cost when $\mathbf{y}_i$ is ground truth and $\mathbf{y}_j$ is the prediction, can be viewed as the dissimilarity between the ground truth role $\mathbf{y}_i^{(t)}$ and the prediction role $\mathbf{y}_j^{(p)}$, which is symmetric for them. Similarly, $\delta(c(\mathbf{y}_j, \mathbf{y}_i))$ can be viewed as the dissimilarity between prediction role $\mathbf{y}_j^{(p)}$ and ground truth role $\mathbf{y}_i^{(t)}$. That is, all the asymmetric transformed costs can be viewed as the dissimilarities between the label vectors in $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(p)}$.

Based on this view, instead of embedding $\mathcal{S}$ by MDS, we embed both $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(p)}$ by considering $2L$ objects, the first L objects being the elements in $\mathcal{S}^{(t)}$ and the last L objects being the elements in $\mathcal{S}^{(p)}$. Following the mirroring step above, we construct symmetric $\boldsymbol{\Delta}$ and $\mathbf{W}$ as $2L \times 2L$ matrices by the following equations and illustrate the constructions by Figure 3.

$$\boldsymbol{\Delta}_{i,j} = \begin{cases} \delta(c(\mathbf{y}_i, \mathbf{y}_{j-L})) & \text{if } (i,j) \text{ in top-right part} \\ \delta(c(\mathbf{y}_{i-L}, \mathbf{y}_j)) & \text{if } (i,j) \text{ in bottom-left part} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\mathbf{W}_{i,j} = \begin{cases} f_i & \text{if } (i,j) \text{ in top-right part} \\ f_j & \text{if } (i,j) \text{ in bottom-left part} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We explain the constructions and the new notations f_i as follows. Given that we are concerned only about the dissimilarities between the elements in $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(p)}$, we set the top-left and the bottom-right parts of $\mathbf{W}$ to zeros (and set the corresponding parts of $\boldsymbol{\Delta}$ conveniently to zeros as well). Then, we set the top-right part and the bottom-left part of $\boldsymbol{\Delta}$ to be the transformed costs to reflect the dissimilarities. The top-right part and the bottom-left part of $\boldsymbol{\Delta}$ are in fact $\mathbf{C}(\mathcal{S})$ and $\mathbf{C}(\mathcal{S})^\top$ respectively, as illustrated by Figure 3. Considering that every label vector may have different importance, to reflect this difference, we set the top-right part of weight $\mathbf{W}_{i,j}$ to be f_i, the frequency of $\mathbf{y}_i$ in $\mathcal{D}$, and set the bottom-left part of weight $\mathbf{W}_{i,j}$ to be f_j.

By solving MDS with the above-mentioned $\boldsymbol{\Delta}$ and $\mathbf{W}$, we can obtain the target vector $\mathbf{u}_i^{(t)}$ and $\mathbf{u}_i^{(p)}$ corresponding to $\mathbf{y}_i^{(t)}$ and $\mathbf{y}_i^{(p)}$. We take $\mathcal{U}^{(t)}$ and
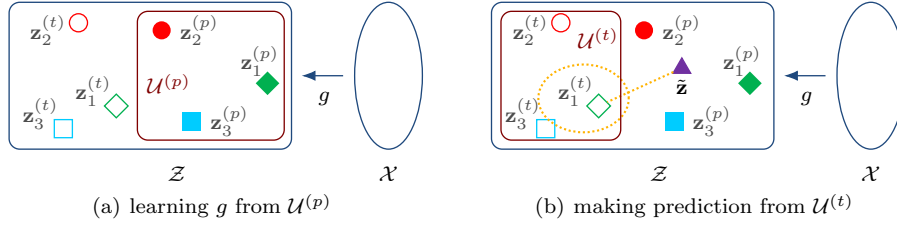
(a) learning g from $\mathcal{U}^{(p)}$ (b) making prediction from $\mathcal{U}^{(t)}$

Fig. 4 Different use of two roles of embedded vectors

Algorithm 1 Training process of CLEMS

1: Given $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}$, cost function c, and embedded dimension M
2: Decide the candidate set $\mathcal{S}$, and calculate $\mathbf{\Delta}$ and $\mathbf{W}$ by (1) and (2)
3: Solve MDS with $\mathbf{\Delta}$ and $\mathbf{W}$, and obtain the two roles of embedding vectors $\mathcal{U}^{(t)}$ and $\mathcal{U}^{(p)}$
4: Set embedding function $\Phi: \mathcal{S} \to \mathcal{U}^{(p)}$ and embedded vector set $\mathcal{Q} = \mathcal{U}^{(t)}$
5: Train a regressor g from $\{(\mathbf{x}^{(n)}, \Phi(\mathbf{y}^{(n)}))\}_{n=1}^{N}$

Algorithm 2 Predicting process of CLEMS

1: Given a testing example $\mathbf{x}$
2: Obtain the predicted embedded vector $\tilde{\mathbf{z}} = g(\mathbf{x})$
3: Find $\mathbf{z}_q \in \mathcal{Q}$ such that $d(\mathbf{z}_q, \tilde{\mathbf{z}})$ is the smallest
4: Make prediction $\tilde{\mathbf{y}} = \Phi^{-1}(\mathbf{z}_q)$

$\mathcal{U}^{(p)}$ to denote the target vector sets $\{\mathbf{u}_i^{(t)}\}_{i=1}^{L}$ and $\{\mathbf{u}_i^{(p)}\}_{i=1}^{L}$, respectively. Those target vectors minimize $\sum_{i,j} \mathbf{W}_{i,j}(d(\mathbf{u}_i^{(t)}, \mathbf{u}_j^{(p)}) - \delta(c(\mathbf{y}_i, \mathbf{y}_j)))^2$. That is, the cost information is embedded in the distances between the elements in $\mathcal{U}^{(t)}$ and $\mathcal{U}^{(p)}$.

Since we mirror each label vector $\mathbf{y}_i$ as two roles ($\mathbf{y}_i^{(t)}$ and $\mathbf{y}_i^{(p)}$), we need to decide which target vector ($\mathbf{u}_i^{(t)}$ and $\mathbf{u}_i^{(p)}$) is the embedded vector $\mathbf{z}_i$ of $\mathbf{y}_i$. Recall that the goal of the embedded vectors is to train a internal predictor g and obtain $\tilde{\mathbf{z}}$, the "predicted" embedded vector. Therefore, we take the elements in $\mathcal{U}^{(p)}$, which serve the role of the prediction, as the embedded vectors of the elements in $\mathcal{S}$, as illustrated by Figure 4(a). Accordingly, the nearest embedded vector $\mathbf{z}_q$ should be the role of the ground truth because the cost information is embedded in the distance between these two roles of target vectors. Hence, we take $\mathcal{U}^{(t)}$ as $\mathcal{Q}$, the embedded vector set in the first step of nearest-neighbor decoding, and find the nearest embedded vector $\mathbf{z}_q$ from $\mathcal{Q}$, as illustrated by Figure 4(b). The final cost-sensitive prediction $\tilde{\mathbf{y}} = \mathbf{y}_q$ is the corresponding label vector to $\mathbf{z}_q$, which carries the cost information through nearest-neighbor decoding.

With the embedding function Φ using $\mathcal{U}^{(p)}$ and the nearest-neighbor decoding function Ψ using $\mathcal{Q} = \mathcal{U}^{(t)}$, we have now designed a novel CSLE algorithm. We name it cost-sensitive label embedding with multidimensional scaling (CLEMS). Algorithm 1 and Algorithm 2 respectively list the training process and the predicting process of CLEMS.

3.3 Theoretical guarantee and monotonic function

The last issue is how to choose the monotonic transform function $\delta(\cdot)$. We suggest a proper monotonic function $\delta(\cdot)$ based on the following theoretical results.

Theorem 1 *For any instance* $(\mathbf{x}, \mathbf{y})$, *let* $\mathbf{z}$ *be the embedded vector of* $\mathbf{y}$, $\tilde{\mathbf{z}} = g(\mathbf{x})$ *be the predicted embedded vector,* $\mathbf{z}_q$ *be the nearest embedded vector of* $\tilde{\mathbf{z}}$, *and* $\mathbf{y}_q$ *be the corresponding label vector of* $\mathbf{z}_q$. *In other words,* $\mathbf{y}_q$ *is the outcome of the nearest-neighbor decoding function* Ψ. *Then,*

$$\delta(c(\mathbf{y}, \mathbf{y}_q))^2 \leq 5\Big(\underbrace{(d(\mathbf{z}, \mathbf{z}_q) - \delta(c(\mathbf{y}, \mathbf{y}_q)))^2}_{embedding\ error} + \underbrace{d(\mathbf{z}, \tilde{\mathbf{z}})^2}_{regression\ error}\Big).$$

Proof Since $\mathbf{z}_q$ is the nearest neighbor of $\tilde{\mathbf{z}}$, we have $d(\mathbf{z}, \tilde{\mathbf{z}}) \geq \frac{1}{2}d(\mathbf{z}, \mathbf{z}_q)$. Hence,

$$\begin{aligned}
embedding\ error + regression\ error &= (d(\mathbf{z}, \mathbf{z}_q) - \delta(c(\mathbf{y}, \mathbf{y}_q)))^2 + d(\mathbf{z}, \tilde{\mathbf{z}})^2 \\
&\geq (d(\mathbf{z}, \mathbf{z}_q) - \delta(c(\mathbf{y}, \mathbf{y}_q)))^2 + \frac{1}{4}d(\mathbf{z}, \mathbf{z}_q)^2 \\
&= \frac{5}{4}(d(\mathbf{z}, \mathbf{z}_q) - \frac{4}{5}\delta(c(\mathbf{y}, \mathbf{y}_q)))^2 + \frac{1}{5}\delta(c(\mathbf{y}, \mathbf{y}_q))^2 \\
&\geq \frac{1}{5}\delta(c(\mathbf{y}, \mathbf{y}_q))^2.
\end{aligned}$$

This implies the theorem.

Theorem 1 implies that the cost of the prediction can be bounded by *embedding error* and *regression error*. In our framework, the *embedding error* can be reduced by multidimensional scaling and the *regression error* can be reduced by learning a good regressor g. Theorem 1 provides a theoretical explanation of how our framework achieves cost-sensitivity.

In general, any monotonic function $\delta(\cdot)$ can be used in the proposed framework. Based on Theorem 1, we suggest $\delta(\cdot) = (\cdot)^{1/2}$ to directly bound the cost by $c(\mathbf{y}, \mathbf{y}_q) \leq 5(embedding\ error + regression\ error)$. We will show that the suggested monotonic function leads to promising practical performance in Section 4.

4 Experiments

We conduct the experiments on nine real-world datasets (Tsoumakas et al, 2011b; Read et al, 2016) to validate the proposed algorithm, CLEMS. The details of the datasets are shown by Table 1. We evaluate the algorithms in our cost-sensitive setting with three commonly-used evaluation criteria, namely *F1 score*$(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{2\|\mathbf{y} \cap \tilde{\mathbf{y}}\|_1}{\|\mathbf{y}\|_1 + \|\tilde{\mathbf{y}}\|_1}$, *Accuracy score*$(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{\|\mathbf{y} \cap \tilde{\mathbf{y}}\|_1}{\|\mathbf{y} \cup \tilde{\mathbf{y}}\|_1}$, and *Rank loss*$(\mathbf{y}, \tilde{\mathbf{y}}) = \sum_{\mathbf{y}[i] > \mathbf{y}[j]} ([\![\tilde{\mathbf{y}}[i] < \tilde{\mathbf{y}}[j]]\!] + \frac{1}{2}[\![\tilde{\mathbf{y}}[i] = \tilde{\mathbf{y}}[j]]\!])$. Note that F1 score and Accuracy score are symmetric while Rank loss is asymmetric. For CLEMS, the input cost function is set as the corresponding evaluation criterion.

All the following experimental results are averaged over 20 runs of experiments. In each run, we randomly split 50%, 25%, and 25% of the dataset for training, validation, and testing. We use the validation part to select the best parameters for all

Table 1 Properties of datasets

Dataset	# of instance N	# of feature d	# of labels K	# of distinct labels
CAL500	502	68	174	502
emotions	593	72	6	27
birds	645	260	19	133
medical	978	1449	45	94
enron	1702	1001	53	753
scene	2407	294	6	15
yeast	2417	103	14	198
slashdot	3279	1079	22	156
EUR-Lex(dc)	19348	5000	412	1615

the algorithms and report the corresponding testing results. For all the algorithms, the internal predictors are set as random forest (Breiman, 2001) implemented by scikit-learn (Pedregosa et al, 2011) and the maximum depth of the trees is selected from $\{5, 10, ..., 35\}$. For CLEMS, we use the implementation of scikit-learn for solving SMACOF algorithm to obtain the MDS-based embedding and the parameters of SMACOF algorithm are set as default values by scikit-learn. For the other algorithms, the rest parameters are set as the default values suggested by their original papers. In the following figures and tables, we use the notation $\uparrow (\downarrow)$ to highlight whether a higher (lower) value indicates better performance for the evaluation criterion.

4.1 Comparing CLEMS with LSDR algorithms

In the first experiment, we compare CLEMS with four LSDR algorithms introduced in Section 2: principal label space transformation (PLST) (Tai and Lin, 2012), conditional principal label space transformation (CPLST) (Chen and Lin, 2012), feature-aware implicit label space encoding (FaIE) (Lin et al, 2014), and sparse local embeddings for extreme classification (SLEEC) (Bhatia et al, 2015)

Since the prediction of SLEEC is a real-value vector rather than binary, we choose the best threshold for quantizing the vector according to the given criterion during training. Thus, our modified SLEEC can be viewed as "semi-cost-sensitive" algorithm that learns the threshold according to the criterion.

Figures 5 and Figure 6 show the results of F1 score and Accuracy score across different embedded dimensions M. As M increases, all the algorithms reach better performance because of the better preservation of label information. CLEMS outperforms the non-cost-sensitive algorithms (PLST, CPLST, and FaIE) in most of the cases, which verifies the importance of constructing a cost-sensitive embedding. CLEMS also exhibits considerably better performance over SLEEC in most of the datasets, which demonstrates the usefulness to consider the cost information during embedding (CLEMS) rather than after the embedding (SLEEC). The results of Rank loss are shown by Figure 7. CLEMS again reaches the best in most of the cases, which justifies its validity for asymmetric criteria through the mirroring trick.
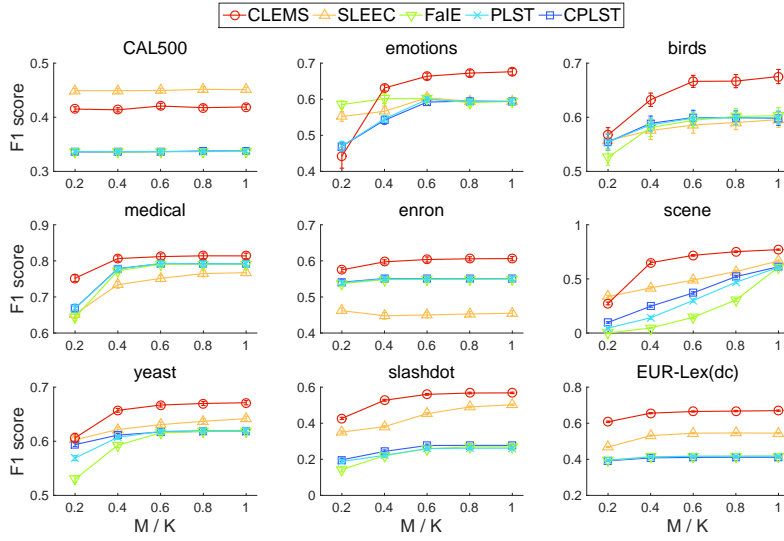
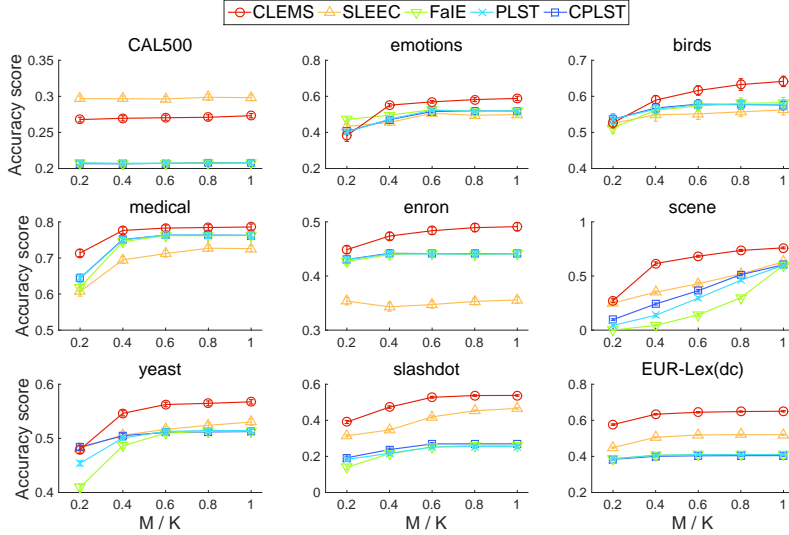**Fig. 5** F1 score (↑) with the 95% confidence interval of CLEMS and LSDR algorithms

Fig. 6 Accuracy score (↑) with the 95% confidence interval of CLEMS and LSDR algorithms

4.2 Comparing CLEMS with LSDE algorithms

We compare CLEMS with ECC-based LSDE algorithms (Ferng and Lin, 2013). We consider two promising error-correcting codes, *repetition code* (ECC-RREP) and *Hamming on repetition code* (ECC-HAMR) in the original work. The former is equivalent to the famous Random k-labelsets (RAkEL) algorithm (Tsoumakas et al, 2011a).
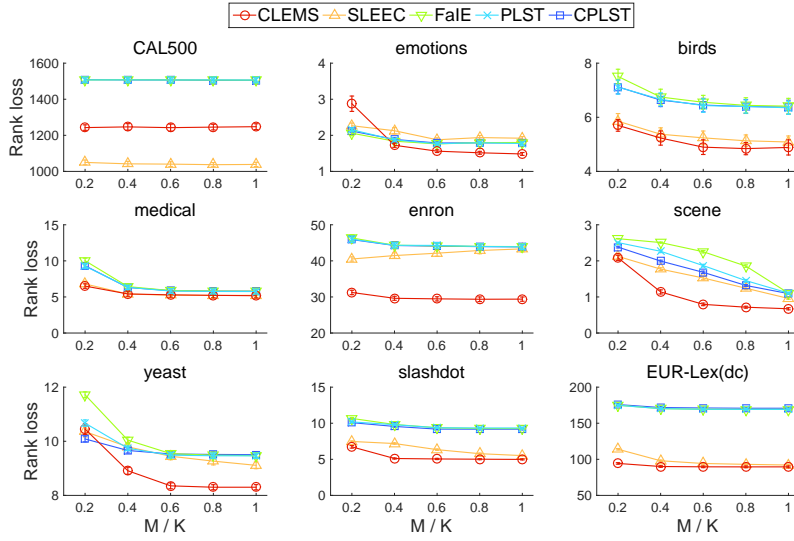
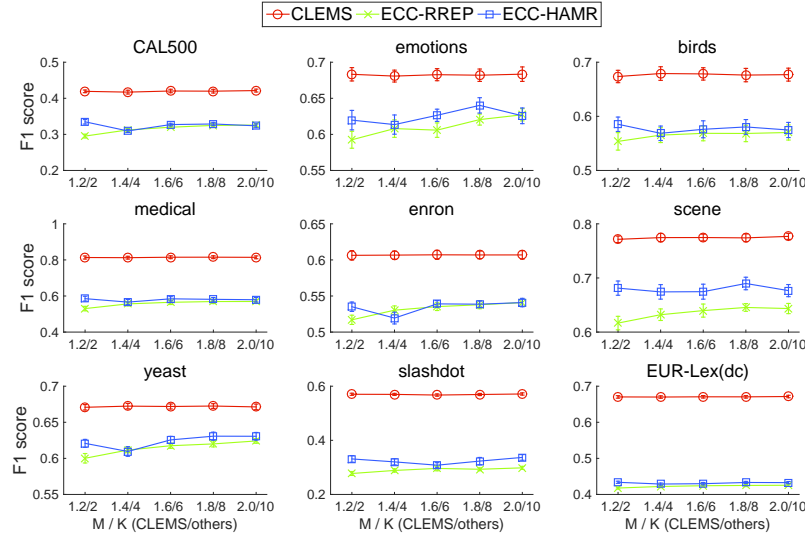**Fig. 7** Rank loss (↓) with the 95% confidence interval of CLEMS and LSDR algorithms

Fig. 8 F1 score (↑) with the 95% confidence interval of CLEMS and LSDE algorithms

Figure 8 shows the results of F1 score. Note that in the figure, the scales of M/K for CLEMS and other LSDE algorithms are different. The scale of CLEMS is $\{1.2, 1.4, 1.6, 1.8, 2.0\}$ while the scale of other LSDE algorithms is $\{2, 4, 6, 8, 10\}$. Although we give LSDE algorithms more dimensions to embed the label information, CLEMS is still superior to those LSDE algorithms in most of cases. Similar results happen for Accuracy score and the Rank loss (Figure 9 and Figure 10). The results again justify the superiority of CLEMS.
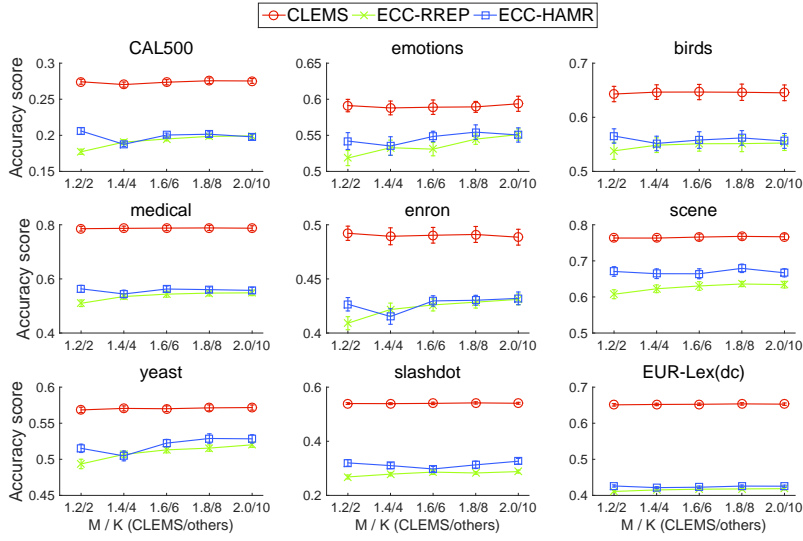
Fig. 9 Accuracy score (↑) with the 95% confidence interval of CLEMS and LSDE algorithms
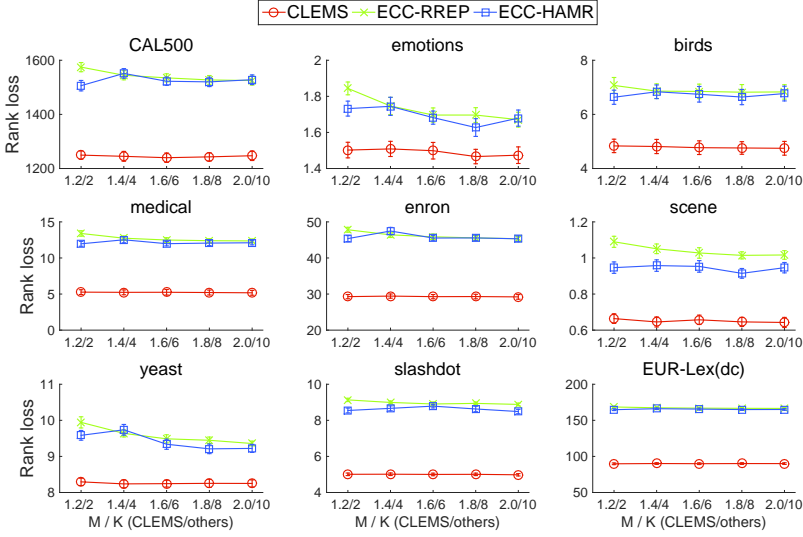

Fig. 10 Rank loss (↓) with the 95% confidence interval of CLEMS and LSDE algorithms

4.3 Candidate set and embedded dimension

Now, we discuss the influence of the candidate set $\mathcal{S}$. In Section 3, we proposed to embed $\mathcal{S}_{tr}$ instead of $\mathcal{Y}$. To verify the goodness of the choice, we compare CLEMS with different candidate sets. We consider the sets sub-sampled with different percentage from $\mathcal{S}_{tr}$ to evaluate the importance of label vectors in $\mathcal{S}_{tr}$. Furthermore, to know whether or not larger candidate set leads to better performance, we also
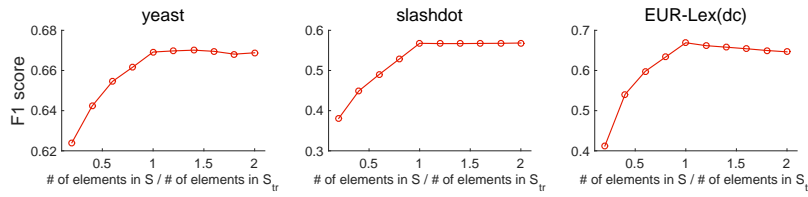
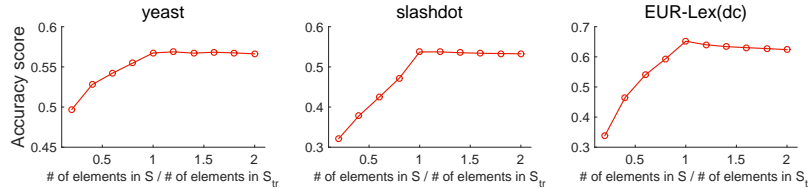**Fig. 11** F1 score (↑) of CLEMS with different size of candidate sets

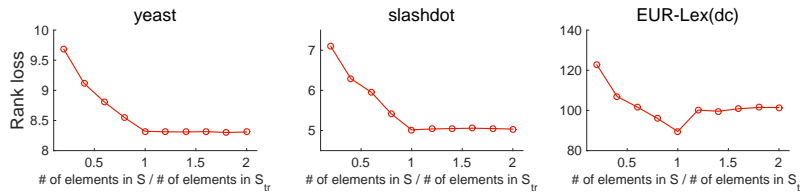**Fig. 12** Accuracy score (↑) of CLEMS with different size of candidate sets

Fig. 13 Rank loss (↓) of CLEMS with different size of candidate sets

randomly sample different percentage of additional label vectors from $\mathcal{Y} \setminus \mathcal{S}_{tr}$ and merge them with $\mathcal{S}_{tr}$ as the candidate sets. The results of three largest datasets are shown by Figures 11, 12, and 13. From the figures, we observe that sub-sampling from $\mathcal{S}_{tr}$ generally lead to worse performance; adding more candidates from $\mathcal{Y} \setminus \mathcal{S}_{tr}$, on the other hand, does not lead to significantly-better performance. The two findings suggest that using $\mathcal{S}_{tr}$ as the candidate set is necessary and sufficient for decent performance.

We conduct another experiment about the candidate set. Instead of random sampling, we consider $\mathcal{S}_{all}$, which denotes the set of label vectors that appear in the training instances and the testing instances, to estimate the benefit of "peeping" the testing label vectors and embedding them in advance. We show the results of CLEMS with $\mathcal{S}_{tr}$ (CLEMS-train) and $\mathcal{S}_{all}$ (CLEMS-all) versus different embedded dimensions by Figure 14, 15, and 16. From the figures, we see that the improvement of CLEMS-all over CLEMS-train is small and insignificant. The results imply again that $\mathcal{S}_{tr}$ readily allows nearest-neighbor decoding to make sufficiently good choices.

Now, we discuss about the embedded dimension M. From Figure 14, 15, and 16, CLEMS reaches better performance as M increases. For LSDR, M plays an important role since it decides how much information can be preserved in the embedded space. Nevertheless, For LSDE, the improvement becomes marginal when M increases. The results suggest that for LSDE, the influence of the additional dimension is not large, and setting the embedded dimension $M = K$ is sufficiently good
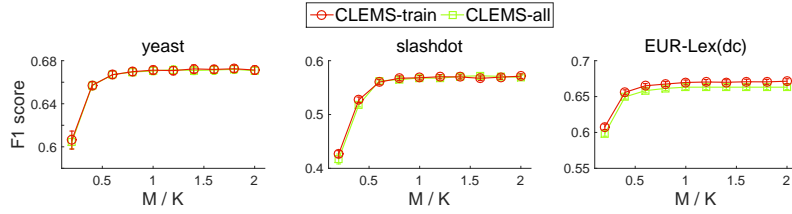
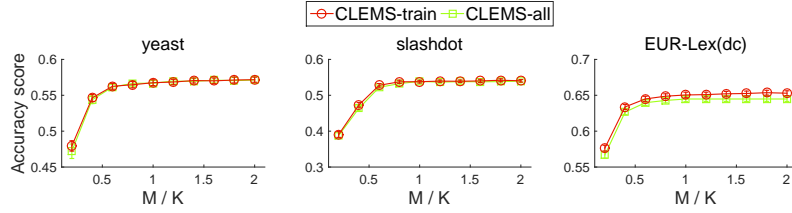**Fig. 14** F1 score (↑) with the 95% confidence interval of CLEMS-train and CLEMS-all

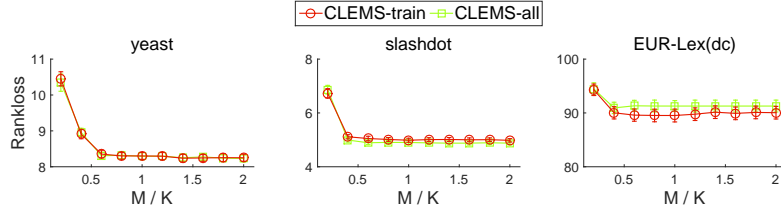**Fig. 15** Accuracy score (↑) with the 95% confidence interval of CLEMS-train and CLEMS-all

Fig. 16 Rank loss (↓) with the 95% confidence interval of CLEMS-train and CLEMS-all

in practice. One possible reason for the sufficiency is that the criteria of interest are generally not complicated enough and thus do not need more dimensions to preserve the cost information.

4.4 Comparing CLEMS with cost-sensitive algorithms

In this section, we compare CLEMS with two state-of-the-art cost-sensitive algorithms, probabilistic classifier chain (PCC) (Dembczynski et al, 2010, 2011) and condensed filter tree (CFT) (Li and Lin, 2014). Both CLEMS and CFT can handle arbitrary criteria while PCC can handle only those criteria with efficient inference rules. In addition, we also report the results of some baseline algorithms, such as binary relevance (BR) (Tsoumakas and Katakis, 2007) and classifier chain (CC) (Read et al, 2011). Similar to previous experiments, the internal predictors of all algorithms are set as random forest (Breiman, 2001) implemented by scikit-learn (Pedregosa et al, 2011) with the same parameter selection process.

Running Time. Figure 17 illustrates the average training, predicting, and total running time when taking F1 score as the intended criterion for the six largest
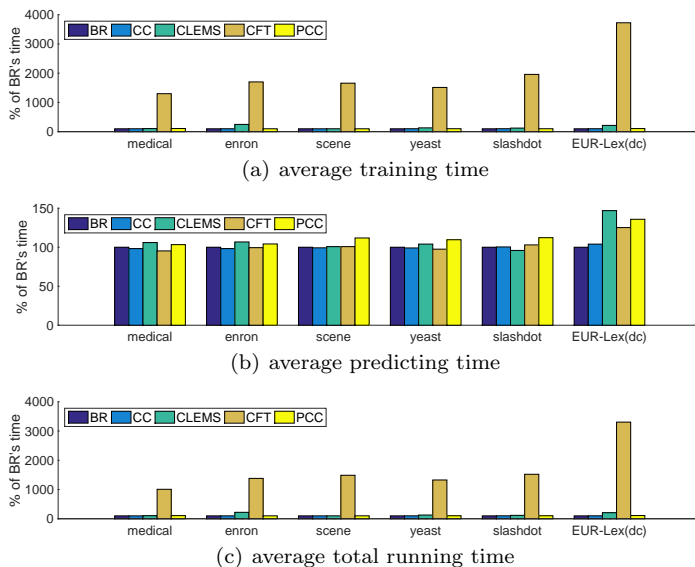
(a) average training time

(b) average predicting time

(c) average total running time

Fig. 17 Average running time when taking F1 score as cost function

datasets. The running time is normalized by the running time of BR. For training time, CFT is the slowest, because it needs to iteratively estimate the importance of each label and re-train internal predictors. CLEMS, which consumes time for MDS calculations, is intuitively slower than baseline algorithms and PCC during training, but still much faster than CFT. For prediction time, all algorithms, including PCC (using inference calculation) and CLEMS (using nearest-neighbor calculation) are similarly fast. The results suggest that for CSMLC, CLEMS is superior to CFT and competitive to PCC for the overall efficiency.

Performance. We compare the performance of CLEMS and other algorithms across different criteria. To demonstrate the full ability of CLEMS, in addition to F1 score, Accuracy score, and Rank loss, we further consider one additional criterion, *Composition loss* = $1 + 5 \times$ *Hamming loss* $-$ *F1 score*, as used by Li and Lin (2014). We also consider three more datasets (arts, flags, and language-log) that comes from other MLC works (Tsoumakas et al, 2011b; Read et al, 2016).

The results are shown by Table 2. Accuracy score and Composition loss for PCC are left blank since there is no efficient inference rules. The first finding is that cost-sensitive algorithms (CLEMS, PCC, and CFT) generally perform better than non-cost-sensitive algorithms (BR and CC) across different criteria. This validates the usefulness of cost-sensitivity for MLC algorithms.

For F1 score, Accuracy score, and Composition loss, CLEMS outperforms PCC and CFT in most cases. The reason is that these criteria evaluate all the labels jointly, and CLEMS can globally locate the hidden structure of labels to facilitate more effective learning, while PCC and CFT are chain-based algorithms and only locally discover the relation between labels. For Rank loss, PCC performs the best in most cases. One possible reason is that Rank loss can be expressed as a special weighted Hamming loss that does not require globally locating the hidden

Table 2 Performance across different criteria (mean ± ste (rank)) (best marked in bold)

Dataset	Alg.	F1 score (↑)	Acc. score (↑)	Rank loss (↓)	Compo. loss (↓)
flags	BR	0.703 ± 0.006 (5)	0.591 ± 0.007 (3)	3.011 ± 0.056 (4)	1.583 ± 0.028 (3)
	CC	0.704 ± 0.006 (4)	0.594 ± 0.008 (2)	2.998 ± 0.061 (3)	1.580 ± 0.028 (2)
	CLEMS	**0.731 ± 0.005 (1)**	**0.615 ± 0.008 (1)**	2.930 ± 0.061 (2)	**1.575 ± 0.026 (1)**
	CFT	0.692 ± 0.008 (3)	0.588 ± 0.009 (4)	3.075 ± 0.060 (5)	1.640 ± 0.033 (4)
	PCC	0.706 ± 0.006 (2)	–	**2.857 ± 0.051 (1)**	–
CAL.	BR	0.338 ± 0.002 (4)	0.208 ± 0.001 (3)	1504.8 ± 7.98 (4)	**1.366 ± 0.005 (1)**
	CC	0.328 ± 0.002 (5)	0.202 ± 0.002 (4)	1520.9 ± 9.04 (5)	1.371 ± 0.006 (2)
	CLEMS	**0.419 ± 0.002 (1)**	**0.273 ± 0.002 (1)**	1247.9 ± 8.21 (3)	1.426 ± 0.004 (4)
	CFT	0.371 ± 0.003 (3)	0.237 ± 0.002 (2)	1120.8 ± 8.46 (2)	1.378 ± 0.006 (3)
	PCC	0.391 ± 0.002 (2)	–	**993.6 ± 4.75 (1)**	–
birds	BR	0.569 ± 0.007 (5)	0.551 ± 0.007 (4)	6.845 ± 0.139 (5)	0.656 ± 0.011 (4)
	CC	0.570 ± 0.007 (4)	0.552 ± 0.007 (3)	6.825 ± 0.138 (4)	0.654 ± 0.011 (3)
	CLEMS	**0.677 ± 0.006 (1)**	**0.642 ± 0.007 (1)**	4.886 ± 0.142 (2)	**0.563 ± 0.012 (1)**
	CFT	0.601 ± 0.007 (3)	0.586 ± 0.007 (2)	4.908 ± 0.148 (3)	0.607 ± 0.012 (2)
	PCC	0.636 ± 0.007 (2)	–	**3.660 ± 0.103 (1)**	–
emot.	BR	0.596 ± 0.005 (5)	0.523 ± 0.004 (4)	1.764 ± 0.022 (5)	1.352 ± 0.012 (4)
	CC	0.615 ± 0.005 (4)	0.539 ± 0.004 (3)	1.715 ± 0.021 (4)	1.329 ± 0.013 (3)
	CLEMS	**0.676 ± 0.005 (1)**	**0.589 ± 0.006 (1)**	1.484 ± 0.020 (2)	**1.271 ± 0.013 (1)**
	CFT	0.640 ± 0.004 (3)	0.557 ± 0.004 (2)	1.563 ± 0.018 (3)	1.324 ± 0.016 (2)
	PCC	0.643 ± 0.005 (2)	–	**1.467 ± 0.018 (1)**	–
medic.	BR	0.517 ± 0.006 (5)	0.496 ± 0.006 (4)	13.784 ± 0.175 (5)	0.562 ± 0.006 (4)
	CC	0.533 ± 0.006 (4)	0.512 ± 0.006 (3)	13.328 ± 0.167 (4)	0.544 ± 0.007 (3)
	CLEMS	**0.814 ± 0.004 (1)**	**0.786 ± 0.004 (1)**	5.170 ± 0.159 (2)	**0.289 ± 0.005 (1)**
	CFT	0.635 ± 0.005 (2)	0.613 ± 0.005 (2)	5.811 ± 0.131 (3)	0.438 ± 0.007 (2)
	PCC	0.573 ± 0.006 (3)	–	**4.234 ± 0.109 (1)**	–
lang.	BR	0.160 ± 0.004 (5)	0.159 ± 0.004 (4)	42.46 ± 0.271 (5)	0.919 ± 0.004 (4)
	CC	0.161 ± 0.004 (4)	0.160 ± 0.004 (3)	42.42 ± 0.168 (4)	0.918 ± 0.004 (3)
	CLEMS	**0.375 ± 0.005 (1)**	**0.327 ± 0.005 (1)**	31.03 ± 0.383 (2)	**0.734 ± 0.007 (1)**
	CFT	0.168 ± 0.004 (3)	0.164 ± 0.004 (2)	34.16 ± 0.285 (3)	0.910 ± 0.005 (2)
	PCC	0.247 ± 0.004 (2)	–	**19.11 ± 0.211 (1)**	–
enron	BR	0.543 ± 0.003 (4)	0.433 ± 0.003 (4)	44.83 ± 0.376 (5)	0.688 ± 0.004 (4)
	CC	0.553 ± 0.003 (3)	0.443 ± 0.003 (3)	43.82 ± 0.429 (4)	0.678 ± 0.005 (3)
	CLEMS	**0.606 ± 0.003 (1)**	**0.491 ± 0.004 (1)**	29.40 ± 0.300 (3)	**0.659 ± 0.005 (1)**
	CFT	0.557 ± 0.004 (2)	0.448 ± 0.003 (2)	26.64 ± 0.311 (2)	0.677 ± 0.005 (2)
	PCC	0.542 ± 0.003 (5)	–	**25.11 ± 0.263 (1)**	–
scene	BR	0.577 ± 0.003 (5)	0.568 ± 0.004 (4)	1.169 ± 0.010 (5)	0.866 ± 0.007 (4)
	CC	0.598 ± 0.004 (4)	0.590 ± 0.004 (3)	1.122 ± 0.012 (4)	0.833 ± 0.009 (3)
	CLEMS	**0.770 ± 0.003 (1)**	**0.760 ± 0.004 (1)**	0.672 ± 0.015 (2)	**0.578 ± 0.009 (1)**
	CFT	0.703 ± 0.004 (3)	0.656 ± 0.004 (2)	0.723 ± 0.011 (3)	0.776 ± 0.009 (2)
	PCC	0.745 ± 0.003 (2)	–	**0.645 ± 0.005 (1)**	–
yeast	BR	0.611 ± 0.002 (5)	0.503 ± 0.002 (4)	9.673 ± 0.048 (5)	1.345 ± 0.006 (3)
	CC	0.612 ± 0.003 (4)	0.512 ± 0.003 (3)	9.530 ± 0.067 (4)	1.352 ± 0.009 (4)
	CLEMS	**0.671 ± 0.002 (1)**	**0.568 ± 0.002 (1)**	8.302 ± 0.049 (1)	**1.308 ± 0.006 (1)**
	CFT	0.649 ± 0.002 (2)	0.543 ± 0.002 (2)	8.566 ± 0.052 (3)	1.335 ± 0.007 (2)
	PCC	0.614 ± 0.002 (3)	–	8.469 ± 0.057 (2)	–
slash.	BR	0.215 ± 0.002 (5)	0.208 ± 0.002 (4)	9.819 ± 0.030 (5)	1.007 ± 0.003 (4)
	CC	0.230 ± 0.002 (4)	0.222 ± 0.002 (3)	9.662 ± 0.027 (4)	0.990 ± 0.003 (3)
	CLEMS	**0.568 ± 0.002 (1)**	**0.538 ± 0.002 (1)**	4.986 ± 0.038 (2)	**0.668 ± 0.003 (1)**
	CFT	0.429 ± 0.003 (3)	0.402 ± 0.003 (2)	5.677 ± 0.033 (3)	0.798 ± 0.003 (2)
	PCC	0.503 ± 0.003 (2)	–	**4.472 ± 0.029 (1)**	–
arts	BR	0.167 ± 0.002 (5)	0.156 ± 0.002 (4)	17.221 ± 0.064 (5)	1.117 ± 0.003 (4)
	CC	0.170 ± 0.002 (4)	0.160 ± 0.002 (3)	17.173 ± 0.064 (4)	1.113 ± 0.003 (3)
	CLEMS	**0.492 ± 0.002 (1)**	**0.451 ± 0.003 (1)**	9.865 ± 0.079 (2)	**0.815 ± 0.006 (1)**
	CFT	0.334 ± 0.002 (3)	0.281 ± 0.002 (2)	10.071 ± 0.060 (3)	1.001 ± 0.003 (2)
	PCC	0.349 ± 0.002 (2)	–	**8.467 ± 0.047 (1)**	–
EUR.	BR	0.417 ± 0.002 (4)	0.411 ± 0.001 (3)	168.38 ± 0.63 (4)	0.593 ± 0.002 (3)
	CC	0.416 ± 0.002 (5)	0.410 ± 0.001 (4)	168.57 ± 0.61 (5)	0.594 ± 0.002 (4)
	CLEMS	**0.670 ± 0.002 (1)**	**0.650 ± 0.002 (1)**	89.52 ± 0.61 (2)	**0.344 ± 0.002 (1)**
	CFT	0.456 ± 0.002 (3)	0.450 ± 0.002 (2)	129.53 ± 0.75 (3)	0.552 ± 0.002 (2)
	PCC	0.483 ± 0.002 (2)	–	**43.28 ± 0.22 (1)**	–

structure. Thus, chaining algorithms like PCC can still perform decently. Note, however, that CLEMS is often the second best for Rank loss as well.

In summary, we identify two merits of CLEMS. The first is that while PCC performs better on Rank loss, CLEMS is competitive for general cost-sensitivity and can be coupled with arbitrary criteria. The second is that although CFT also shoots for general cost-sensitivity, CLEMS outperforms CFT in most cases for all criteria. The results make CLEMS a decent first-hand-choice for general CSMLC.

Performance on other criteria. So far, we have justified the benefits of CLEMS for directly optimizing towards the criterion of interest. Next, we discuss about whether CLEMS can be used to *indirectly* optimize other criteria of interest, particularly when the criterion cannot be meaningfully expressed as the input to CLEMS. CLEMS follows the setting in Section 2 to accept *example-based* crite-

Table 3 Comparison for other criteria (best marked in bold)

Dataset	Criterion	BR	CC	CLEMS (F1)	CLEMS (Acc.)	CLEMS (Rank.)	PCC (F1)	PCC (Rank.)
emotions	Macro F1 (↑)	0.673	0.682	0.703	**0.711**	0.708	0.700	0.698
	Micro F1 (↑)	0.672	0.708	0.705	**0.717**	0.708	0.679	0.700
	Subset Acc. (↑)	0.282	0.317	0.248	**0.337**	0.262	0.277	0.248
medical	Macro F1 (↑)	0.345	0.364	0.395	**0.424**	0.408	0.361	0.188
	Micro F1 (↑)	0.692	0.699	0.757	**0.788**	0.657	0.586	0.316
	Subset Acc. (↑)	0.490	0.511	0.598	**0.673**	0.393	0.348	0.024
enron	Macro F1 (↑)	0.105	0.114	0.163	0.148	**0.223**	0.214	0.222
	Micro F1 (↑)	0.453	0.477	**0.559**	0.531	0.501	0.511	0.420
	Subset Acc. (↑)	0.033	0.036	**0.073**	**0.073**	0.007	0.029	0.000
scene	Macro F1 (↑)	0.693	0.682	0.766	**0.783**	0.766	0.748	0.698
	Micro F1 (↑)	0.690	0.705	0.736	**0.775**	0.733	0.727	0.700
	Subset Acc. (↑)	0.533	0.560	0.520	**0.704**	0.526	0.528	0.278
yeast	Macro F1 (↑)	0.450	0.458	0.496	0.492	**0.506**	0.464	0.474
	Micro F1 (↑)	0.641	0.644	0.674	**0.682**	0.672	0.605	0.665
	Subset. Acc. (↑)	0.164	**0.220**	0.150	0.214	0.127	0.121	0.132
Corel5k	Macro F1 (↑)	0.038	0.055	0.073	**0.092**	0.069	0.026	0.042
	Micro F1 (↑)	0.072	0.071	0.264	**0.272**	0.255	0.197	0.091
	Subset. Acc. (↑)	0.000	0.002	0.020	**0.034**	0.020	0.000	0.000

rion, which works on one label vector at a time. A more general type of criteria considers multiple or all the label vectors at the same time, called *label-based* criteria. Two representative *label-based* criteria are Micro F1 and Macro F1 (Madjarov et al, 2012), and will be studied next. The former calculates the F1 score over all the label components of testing examples, and the latter averages the per-label F1 score across examples. To the best of our knowledge, there is no cost-sensitive algorithms can handle arbitrary *label-based* criteria.

Another criterion that we will study is subset accuracy (Madjarov et al, 2012). It can be expressed as an *example-based* criterion with two possible values: whether the label vector is completely correct or not. The criterion is very strict and does not come with trade-off on big or small prediction errors. Thus, it is generally not meaningful to feed the criterion directly to CLEMS or other CSMLC algorithms.

Next, we demonstrate how CLEMS can indirectly optimize Micro/Macro F1 score and subset accuracy when fed with other criteria as inputs. We consider 6 pre-divided datasets (emotions, scene, yeast, medical, enron, and Corel5k) as used by Madjarov et al (2012). We consider two baseline algorithms (BR and CC), CLEMS with three different input criteria (F1 score, Accuracy score, Rank loss), and PCC with two different criteria (F1 score and Rank loss) that come with efficient inference rules. The results are shown in Table 3.

From the table, we observe that when selecting a proper criterion as the input of CSMLC algorithms (CLEMS or PCC), they can readily perform better than the baseline algorithms. The results justify the value of the CSMLC algorithms beyond handling *example-based* criteria. In particular, the cost input to CSMLC algorithms act as a tunable parameter towards optimizing other true criteria of interests. We also observe that CLEMS, especially CLEMS-Acc, performs better on the three criteria than PCC in the most datasets, which again validate the usefulness of CLEMS. An interesting future direction is whether CLEMS can be further extended to achieve cost-sensitivity for *label-based* criteria.

5 Conclusion

We propose a novel cost-sensitive label embedding algorithm called cost-sensitive label embedding with multidimensional scaling (CLEMS). CLEMS successfully embeds the label information and cost information into an arbitrary-dimensional

hidden structure by the classic multidimensional scaling approach for manifold learning, and handles asymmetric cost functions with our careful design of the mirroring trick. With the embedding, CLEMS can make cost-sensitive predictions efficiently and effectively by decoding to the nearest neighbor within a proper candidate set. The empirical results demonstrate that CLEMS is superior to state-of-the-art label embedding algorithms across different cost functions. To the best of our knowledge, CLEMS is the very first algorithm that achieves cost-sensitivity within label embedding, and opens a promising future research direction of designing cost-sensitive label embedding algorithms using manifold learning approaches.

References

Balasubramanian K, Lebanon G (2012) The landmark selection method for multiple output prediction. In: ICML

Barutçuoglu Z, Schapire RE, Troyanskaya OG (2006) Hierarchical multi-label prediction of gene function. Bioinformatics 22(7):830–836

Bhatia K, Jain H, Kar P, Varma M, Jain P (2015) Sparse local embeddings for extreme multi-label classification. In: NIPS, pp 730–738

Bi W, Kwok JT (2013) Efficient multi-label classification with many labels. In: ICML, pp 405–413

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Carneiro G, Chan AB, Moreno PJ, Vasconcelos N (2007) Supervised learning of semantic classes for image annotation and retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(3):394–410

Chen YN, Lin HT (2012) Feature-aware label space dimension reduction for multi-label classification. In: NIPS, pp 1538–1546

De Leeuw J (1977) Applications of convex analysis to multidimensional scaling. Recent Developments in Statistics pp 133–145

Dembczynski K, Cheng W, Hüllermeier E (2010) Bayes optimal multilabel classification via probabilistic classifier chains. In: ICML, pp 279–286

Dembczynski K, Waegeman W, Cheng W, Hüllermeier E (2011) An exact algorithm for F-measure maximization. In: NIPS, pp 1404–1412

Ferng CS, Lin HT (2013) Multilabel classification using error-correcting codes of hard or soft bits. IEEE Transactions on Neural Networks and Learning Systems 24(11):1888–1900

Hsu D, Kakade S, Langford J, Zhang T (2009) Multi-label prediction via compressed sensing. In: NIPS, pp 772–780

Kapoor A, Viswanathan R, Jain P (2012) Multilabel classification using bayesian compressed sensing. In: NIPS, pp 2654–2662

Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1):1–27

Li CL, Lin HT (2014) Condensed filter tree for cost-sensitive multi-label classification. In: ICML, pp 423–431

Lin Z, Ding G, Hu M, Wang J (2014) Multi-label classification via feature-aware implicit label space encoding. In: ICML, pp 325–333

Lo HY, Wang JC, Wang HM, Lin SD (2011) Cost-sensitive multi-label learning for audio tag annotation and retrieval. IEEE Transactions on Multimedia 13(3):518–529

Lo HY, Lin SD, Wang HM (2014) Generalized k-labelsets ensemble for multi-label and cost-sensitive classification. IEEE Transactions on Knowledge and Data Engineering 26(7):1679–1691

Madjarov G, Kocev D, Gjorgjevikj D, Dzeroski S (2012) An extensive experimental comparison of methods for multi-label learning. Pattern Recognition 45(9):3084–3104

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830

Read J, Pfahringer B, Holmes G, Frank E (2011) Classifier chains for multi-label classification. Machine Learning 85(3):333–359

Read J, Reutemann P, Pfahringer B, Holmes G (2016) MEKA: A multi-label/multi-target extension to Weka. Journal of Machine Learning Research 17(21):1–5

Schölkopf B, Smola A, Müller K (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural computation 10(5):1299–1319

Sun L, Ji S, Ye J (2011) Canonical correlation analysis for multilabel classification: A least-squares formulation, extensions, and analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 33(1):194–200

Tai F, Lin HT (2012) Multilabel classification with principal label space transformation. Neural Computation 24(9):2508–2542

Trohidis K, Tsoumakas G, Kalliris G, Vlahavas IP (2008) Multi-label classification of music into emotions. In: ISMIR, pp 325–330

Tsoumakas G, Katakis I (2007) Multi-label classification: An overview. International Journal of Data Warehousing and Mining 3(3):1–13

Tsoumakas G, Katakis I, Vlahavas IP (2010) Mining multi-label data. In: Data Mining and Knowledge Discovery Handbook, pp 667–685

Tsoumakas G, Katakis I, Vlahavas IP (2011a) Random k-labelsets for multi-label classification. IEEE Transactions on Knowledge and Data Engineering 23(7):1079–1089

Tsoumakas G, Spyromitros-Xioufis E, Vilcek J, Vlahavas IP (2011b) MULAN: A java library for multi-label learning. Journal of Machine Learning Research 12:2411–2414

Weston J, Chapelle O, Vapnik V, Elisseeff A, Schölkopf B (2002) Kernel dependency estimation. In: NIPS, pp 873–880

Yeh CK, Wu WC, Ko WJ, Wang YCF (2017) Learning deep latent space for multi-label classification. In: AAAI, pp 2838–2844

Yu HF, Jain P, Kar P, Dhillon IS (2014) Large-scale multi-label learning with missing labels. In: ICML, pp 593–601

Zhang Y, Schneider JG (2011) Multi-label output codes using canonical correlation analysis. In: AISTATS, pp 873–882