Cost-sensitive Label Embedding for Multi-label Classification

Kuan-Hao Huang
Hsuan-Tien Lin
Department of Computer Science and Information Engineering, National Taiwan University

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 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. Theoretical results justify that CLEMS achieves the cost-sensitivity and 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; Barutcuoglu 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).

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 transforms the original labels to an embedded space, which represents the hidden structure of the labels. After conduct 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 consider low-dimensional embedded space for digesting the information between labels and conducting more effective learning. Some representative LSDR algorithms are principal label space transformation (Tai & Lin, 2012), conditional principal label space transformation (Chen & Lin, 2012), feature-aware implicit label space encoding (Lin et al., 2014), and sparse local embeddings for extreme classification (Bhatia et al., 2015).

In contrast to LSDR algorithms, LSDE algorithms 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. The random $k$-labelsets approach (Tsoumakas et al., 2011a) and its extensions with error-correcting codes (Ferng & Lin, 2013) are some representative LSDE algorithms.

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 LE algorithms more realistic. Cost-sensitivity for MLC is attracting research attention in recent years (Dembczynski et al., 2010; 2011; Li & 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 conduct a pioneering study on 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 prediction with respect to the criterion of interest. Inspired by the finding that most of the existing LSDR algorithms can be viewed as linear manifold learning approaches, we propose to adopt manifold learning for CSLE.

Nevertheless, to embed 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 space embedding with multidimensional scaling (CLEMS) algorithm embeds the cost information within the distance measure of the embedded space. We propose a mirroring trick for CLEMS that can properly embed the possibly asymmetric criterion within the symmetric distance measure of the embedded space. We also design an efficient procedure that conquers the difficulty of making predictions through the cost-sensitive hidden structure. Theoretical results justify that CLEMS achieves cost-sensitivity through learning in the MDS-embedded space. 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 cost-sensitive MLC algorithm (Li & Lin, 2014). The results justify the superiority of the proposed algorithm.

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 \( x \in \mathcal{X} \subseteq \mathbb{R}^d \) and denote the label vector by \( y \in \mathcal{Y} \subseteq \{0, 1\}^K \) where \( y[i] = 1 \) iff the instance is associated with the \( i \)-th label. Given the training instances \( D = \{(x^{(n)}, y^{(n)})\}_{n=1}^N \), which are drawn from an unknown distribution \( \mathcal{P} \), the goal of MLC algorithms is to train a predictor \( h : \mathcal{X} \rightarrow \mathcal{Y} \) from \( D \) in the training stage, with the expectation that for any unseen test instance \((x, y)\) drawn from \( \mathcal{P} \), the prediction \( \hat{y} = h(x) \) can be close to the ground truth \( y \) in the prediction stage.

The simplest criteria to evaluate the closeness between \( y \) and \( \hat{y} \) is Hamming loss, which is represented by \( \text{Hamming}(y, \hat{y}) = \frac{1}{K} \sum_{i=1}^K |y[i] - \hat{y}[i]| \). It is worth noting that Hamming loss separately evaluates each label component of \( y \). There are other criteria that jointly evaluate all the label components of \( y \), such as 0/1 loss, Rank loss, F1 score, and Accuracy score (Tsoumakas et al., 2010).

A simple algorithm for MLC is binary relevance (BR) (Tsoumakas & Katakis, 2007). BR separately trains a binary classifier for each label without considering the information of other labels. Unlike BR, label embedding (LE) is a big and important family of MLC algorithms that jointly use the information of all labels to achieve better prediction performance. The LE algorithms try to identify the hidden structure behind the labels. In the training stage, instead of training a predictor \( h \) directly, the LE algorithms first embed each \( K \)-dimensional label vector \( y^{(n)} \) as an \( M \)-dimensional embedded vector \( z^{(n)} \in Z \subseteq \mathbb{R}^M \) by an embedding function \( \Phi : \mathcal{Y} \rightarrow Z \). The embedded vector \( z^{(n)} \) can be viewed as the hidden structure that contains the information pertaining to all labels. Then, the algorithms train a predictor \( g : \mathcal{X} \rightarrow Z \) from \( \{(x^{(n)}, z^{(n)})\}_{n=1}^N \). In the prediction stage, for any test instance \( x \), the LE algorithms predict the embedded vector \( \hat{z} = g(x) \) and use a decoding function \( \Psi : Z \rightarrow \mathcal{Y} \) to obtain the prediction \( \hat{y} \). In other words, the LE algorithms obtain the predictor by \( h = \Psi \circ g \).

Figure 1 illustrates the flow of the LE algorithms.

Compressed sensing (CS) (Hsu et al., 2009), principal label space transformation (PLST) (Tai & Lin, 2012), conditional principal label space transformation (CPLST) (Chen & 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) are some representative LE algorithms when \( M \) (the dimension of \( Z \)) is no more than \( K \) (the dimension of \( \mathcal{Y} \)). They utilize different pairs of \( (\Phi, \Psi) \) for embedding and decoding. CS considers random projection as \( \Phi \) and obtains \( \Psi \) by solving an optimization problem per test instance. PLST, CPLST, FaIE all consider \( \Phi \) calculated from some optimal linear projection of the label vectors and \( \Psi \) from the corresponding linear transform, where the latter two take feature vectors into account in the optimality criterion. SLEEC trains a locally-linear projection as \( \Phi \) and constructs \( \Psi \) by nearest neighbors. Given that these algorithms only consider the case \( M \leq K \), they are alternatively termed as label space dimension reduction (LSDR) algorithms. LSDR algorithms are attracting much attention recently because the smaller \( M \) allows \( g \) to be learned more efficiently and effectively.

Other LE algorithms work with \( M \geq K \), which are called
label space dimension expansion (LSDE) algorithms. A representative LSDE algorithm (Ferng & Lin, 2013) is based on error-correcting codes (ECC). The algorithm utilizes the encoding and decoding functions of standard error-correcting codes for communication as Φ and Ψ, respectively. Random k-labelsets (RA/EL) (Tsoumakas et al., 2011a), a popular algorithm for MLC, can be considered as an ECC-based LSDE algorithm with the repetition code (Ferng & Lin, 2013).

To the best of our knowledge, all the existing LE algorithms above are designed for one or only few specific criteria, but may not work well under other criteria. For example, the optimality criterion within PLST is closely related to Hamming loss. For MLC data with very few non-zero \( y[i] \), which are commonly encountered in real-world applications, optimization with respect to Hamming loss can easily results in all-zero predictions \( \hat{y}[i] \), which suffers from bad F1 score. The issue motivates us to study the design of cost-sensitive label embedding (CSLE) algorithms that respect the criteria when calculating the embedding function \( \Phi \) and the decoding function \( \Psi \).

MLC algorithms that take the evaluation criterion into account are called cost-sensitive MLC algorithms and are attracting research attentions in recent years. Two representative approaches are probabilistic classifier chain (PCC) (Dembczynski et al., 2010; 2011) and condensed filter tree (CFT) (Li & Lin, 2014). PCC is based on estimating the probability of each label and making a Bayes-optimal inference for the evaluation criterion. Thus, PCC is restricted by the need of designing an efficient inference rule for each criterion. CFT is based on converting the criterion as instance weights when learning each label and can handle arbitrary criterion. It has been reported that CFT is competitive with PCC on the criteria that PCC can efficiently handle, and better than PCC on other criteria (Li & Lin, 2014). Both algorithms 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 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 cost-sensitive setting. Consider a cost function \( c(y, \hat{y}) \) which represents the penalty when the ground truth is \( y \) and the prediction is \( \hat{y} \). We naturally assume that \( c(y, \hat{y}) \geq 0 \), with value 0 attained iff \( y \) and \( \hat{y} \) are the same. Given training instances \( \mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N} \) drawn from \( \mathcal{P} \) and the cost function \( c \), CSLE algorithms should learn an embedding function \( \Phi \), a decoding function \( \Psi \), and a predictor \( g \) based on not only the training instance \( \mathcal{D} \) but also the cost function \( c \). The objective of CSLE algorithms is to minimizes the expected cost \( c(y, h(x)) \) for any unseen test instance \((x, y)\) drawn from \( \mathcal{P} \), 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 \( \Psi \) of many existing algorithms, such as PLST and FaIE, are derived from \( \Phi \) and can be divided into two steps. The first step is using some \( \psi: \mathcal{Z} \rightarrow \mathbb{R}^{K} \) that corresponds to \( \Phi \) to decode the embedded vector \( z \) to a real-valued vector \( \hat{y} \in \mathbb{R}^{K} \); the second step is choosing a threshold to transform \( \hat{y} \) to \( y \in \{0, 1\}^{K} \). If the embedding function \( \Phi \) is a linear function such as PLST, the corresponding \( \psi \) 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 \( \Phi \) 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), \( \psi \) is often difficult to derive or time-consuming in calculation, which then makes \( \Psi \) practically infeasible to compute.

To resolve the difficulties, we do not consider the two-step decoding function \( \Psi \) that depends on deriving \( \psi \) from \( \Phi \). Instead, we first fix a decent decoding function \( \psi \) and then derive the corresponding embedding function \( \Phi \). We realize that the goal of \( \Psi \) is simply to locate the most probable label vector \( \hat{y} \) from \( \mathcal{Y} \), which is of a finite cardinality, based on the predicted embedded vector \( z = g(x) \in \mathcal{Z} \). 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 \( z_{q} \) of \( z \) and return the corresponding \( y_{q} \) as \( \hat{y} \). Such a nearest-neighbor decoding function \( \Psi \) is actually behind some ECC-based LSDE algorithms (Ferng & Lin, 2013) and will be adopted here.

The nearest-neighbor decoding function \( \Psi \) is based on the distance measure of \( \mathcal{Z} \), which matches our primary need of representing the cost information. In particular, if \( y_{i} \) is a lower-cost prediction than \( y_{j} \) with respect to the ground truth \( y \), we hope that the corresponding embedded vector \( z_{i} \) would be closer to \( z \) than \( z_{j} \). Then, even if \( g \) makes a small error such that \( z = g(x) \) deviates from the desired \( z \), nearest-neighbor decoding function \( \Psi \) can decode to the lower-cost \( y_{i} \) as \( \hat{y} \) instead of \( y_{j} \). In other words, for any two label vectors \( y_{i}, y_{j} \in \mathcal{Y} \) and the corresponding embedded vectors \( z_{i}, z_{j} \in \mathcal{Z} \), we want the Euclidean distance between \( z_{i} \) and \( z_{j} \), which is denoted by \( d(z_{i}, z_{j}) \), to be isotonic to the cost \( c(y_{i}, y_{j}) \).

Based on this objective, the framework of the proposed algorithm is as follows. In the training stage, for each label vector \( y_{i} \in \mathcal{Y} \), the proposed algorithm determines an embedded vector \( z_{i} \) such that the distance between two em-
bedded vectors $d(z_i, z_j)$ in $Z$ approximates $\delta(c(y_i, y_j))$, where $\delta(\cdot)$ is an isotonic function and will be discussed later. We let the embedding function $\Phi$ be the mapping $y_i \rightarrow z_i$, and use $Q$ to represent the embedded vector set \{ $\Phi(y_i) \mid y_i \in \mathcal{Y}$ \}. Then the algorithm trains a multi-target regressor $g : \mathcal{X} \rightarrow \mathcal{Z}$ as the internal predictor.

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

Three key issues of the framework above are yet to be addressed. The first issue is to calculate the embedded vectors $z_i$. The second issue is to choose a proper isotonic function $\delta(\cdot)$. The last issue is to use the symmetric distance measure to embed the possibly asymmetric cost functions where $c(y_i, y_j) \neq c(y_j, y_i)$. Next, we start resolving the three issues.

**Calculating the embedded vectors by multidimensional scaling.** The isotonic cost $\delta(c(y_i, y_j))$ can be viewed as a dissimilarity measure between label vectors. 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 structures. Based on our discussions above, any approach that solves the manifold learning task can then be taken to solve the CSLLE problem. Nevertheless, for CSLLE, different cost functions may need different $M$ (the dimension of $Z$) to achieve a decent embedding. Thus, it is needed to consider manifold learning approaches that can flexibly take $M$ as the parameter. The need motivates us to adopt a classic manifold learning approach called multidimensional scaling (MDS) (Kruskal, 1964).

For a target dimension $M$, MDS attempts to discover the hidden structures of $L_{MDS}$ objects by embedding their dissimilarity in an $M$-dimensional target space. The dissimilarity is represented by a dissimilarity matrix $\Delta$, which is an $L_{MDS} \times L_{MDS}$ matrix with $\Delta_{i,j}$ being the dissimilarity between the $i$-th object and the $j$-th object. $\Delta$ should be symmetric, non-negative, and zero-diagonal. The objective of MDS is to determine target vectors $u_1, u_2, \ldots, u_{L_{MDS}}$ in the target space to minimize the stress, defined as $\sum_{i,j} W_{i,j}(\Delta_{i,j} - d(u_i, u_j))^2$, where $d$ denotes the Euclidean distance in the target space, and $W$ is a symmetric, non-negative, and zero-diagonal matrix that carries the weight $W_{i,j}$ of each object pair.

There are several algorithms available in 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 $O((L_{MDS})^3)$, but there is often room for speeding up with special weight matrices $W$. The objective of MDS is similar to our need of embedding $\delta(c(y_i, y_j))$. We will feed MDS with specific values of $\Delta$ and $W$ based on $\delta$ and $c$ to calculate the embedded vectors.

**Choosing a proper isotonic function using theoretical guarantee.** Before discussing the details of setting $\Delta$ and $W$, we first reveal the theoretical guarantee of the proposed framework. The theoretical results motivate the design of a concrete algorithm under the proposed framework.

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

$$
\delta(c(y, y_q))^2 \leq 5 \left( \frac{d(z, z_q) - \delta(c(y, y_q))}{\text{embedding error}} + \frac{d(z, \tilde{z})^2}{\text{regression error}} \right).
$$

The proof is in the Appendix. Based on Theorem 1, we choose $\delta(\cdot) = (\cdot)^{1/2}$. Then, $c(y, y_q) \leq 5 (\text{embedding error} + \text{regression error})$. The second term can be reduced by learning a good regressor $g$, and the first term can be reduced by MDS via minimizing stress, as shown next. Theorem 1 provides a theoretical explanation of how our framework achieves cost-sensitivity.

**Approximating the asymmetric cost function with MDS.** Next, we discuss the details of using MDS to approximate the cost function. Ideally, we should embed all label vectors $y \in \mathcal{Y} \subseteq \{0, 1\}^K$ in 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 choose a candidate set $S \subseteq \mathcal{Y}$, and only embed the label vectors in $S$. The use of $S$ instead of $\mathcal{Y}$ can reduce the computation burden, but restricts the nearest-neighbor decoding function to only predict from $S$. One reasonable choice of $S$ is the set of label vectors that appear in the training instances $D$, which is denoted as $S_{\text{train}}$. We will show that using $S_{\text{train}}$ as $S$ readily leads to promising performance and discuss more about the choice of the candidate set in Section 4.

After choosing $S$, we discuss the constructions of $\Delta$ and $W$ for solving MDS. Let $L$ denote the number of elements in $S$ and let $C(S)$ be the isotonic cost matrix of $S$, which is an $L \times L$ matrix with $C(S)_{i,j} = \delta(c(y_i, y_j))$ for $y_i, y_j \in$
S. Note that C cannot be directly used as Δ because c can be asymmetric. Next, we propose a mirroring trick to construct a symmetric Δ from C.

The asymmetric cost function implies that each label vector serves two roles: as the ground truth, or as the prediction. For each \( y_i \in S \), we mirror it as \( y_i^{(t)} \) and \( y_i^{(p)} \) to denote viewing \( y_i \) as the ground truth and the prediction, respectively. We then use \( S^{(t)} \) and \( S^{(p)} \) to denote the sets \( \{y_i^{(t)}\}_{i=1}^L \) and \( \{y_i^{(p)}\}_{i=1}^L \). The two mirrored label vectors \( y_i^{(t)} \) and \( y_i^{(p)} \) are in fact the same, but carry different meanings. Instead of embedding \( S \) and \( S \) respectively, we embed both \( S^{(t)} \) and \( S^{(p)} \) by considering \( 2L \) objects, the first \( L \) objects being the elements in \( S^{(t)} \) and the last \( L \) objects being the elements in \( S^{(p)} \). Then, \( \delta(c(y_i, y_j)) \) can be viewed as the dissimilarity between \( y_i^{(t)} \) and \( y_j^{(p)} \), and \( \delta(c(y_j, y_i)) \) can be viewed as the dissimilarity between \( y_j^{(t)} \) and \( y_i^{(p)} \).

Following the mirroring step above, we construct Δ and W as \( 2L \times 2L \) matrices by

\[
\Delta_{i,j} = \begin{cases} 
\delta(c(y_i, y_{j-L})) & \text{if } (i, j) \text{ in top-right part} \\
\delta(c(y_{i-L}, y_j)) & \text{if } (i, j) \text{ in bottom-left part} \\
0 & \text{otherwise}
\end{cases}
\]

and

\[
W_{i,j} = \begin{cases} 
f_i & \text{if } (i, j) \text{ in top-right part} \\
f_j & \text{if } (i, j) \text{ in bottom-left part} \\
0 & \text{otherwise}
\end{cases}
\]

We explain the construction and the new notations \( f_i \) as follows. Given that we are concerned only about the dissimilarities between the elements in \( S^{(t)} \) and the elements in \( S^{(p)} \), we set the top-left and the bottom-right parts of \( W \) to zeros (and the corresponding parts of \( \Delta \) conveniently to zeros as well). Then, we set the top-right part and the bottom-left part of \( \Delta \) to be the isotonic costs to reflect the dissimilarities. To echo Theorem 1, which bounds the cost for per instance, we set the top-right part of weight \( W_{i,j} \) to be \( f_i \), the frequency of \( y_i \) in \( \mathcal{D} \), and set the bottom-left part of weight \( W_{i,j} \) to be \( f_j \). The top-right and the bottom-left part of \( \Delta \) are in fact \( C(S) \) and \( C(S)^\top \) respectively, as illustrated by Figure 2.

Let \( u_i^{(t)} \) and \( u_i^{(p)} \) be the target vectors for \( y_i^{(t)} \) and \( y_i^{(p)} \), respectively, obtained by solving MDS with the above-mentioned \( \Delta \) and \( W \). Take \( U^{(t)} \) and \( U^{(p)} \) to denote the sets \( \{u_i^{(t)}\}_{i=1}^L \) and \( \{u_i^{(p)}\}_{i=1}^L \). Those target vectors minimize \( \sum_{i,j} (\delta(c(y_i, y_j)) - \delta(u_i^{(t)}, u_j^{(p)}))^2 \). That is, the cost information is embedded in the distances between the elements in \( U^{(t)} \) and \( U^{(p)} \). We can then take the elements in \( U^{(p)} \), which serve the role of the prediction, as the embedded vectors of the elements in \( S \), and take \( U^{(t)} \) as \( Q \), the embedded vector set in the first step of nearest-neighbor decoding. During prediction, \( \tilde{z} \) plays the role of a predicted embedded vector from \( y \), and then a minimum-embedded-cost target vector \( z_q \) in \( Q = U^{(t)} \) is decoded to reach a cost-sensitive prediction \( y_q \).

With the embedding function \( \Phi \) using \( U^{(p)} \) and the nearest-neighbor decoding function \( \Psi \) using \( Q = U^{(t)} \), as illustrated by Figure 3, we have now designed a novel CSLE algorithm. We name it cost-sensitive label space embedding with multidimensional scaling (CLEMS).

4. Experiments

In this section, we conduct experiments on eight real-world benchmark datasets downloaded from Mulan (Tsoumakas et al., 2011b). The details of the datasets are shown in the Appendix.

We evaluate all algorithms in our cost-sensitive setting with three commonly-used evaluation criteria, namely F1 score, Accuracy score, and Rank loss. They can be calculated as follows: \( F1(y, \hat{y}) = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|} \). \( Accuracy(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y| + |\hat{y}|} \), and \( Rank(y, \hat{y}) = \sum_{j} \left( |y[j] < \hat{y}[j]| + \frac{1}{2} |\hat{y}[j] = y[j]| \right) \). Note that F1 score and Accuracy score are symmetric while Rank loss is asymmetric.

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. In the following figures, we use the notation ↑ (↓) to highlight whether a higher (lower) value indicates better performance for the evaluation criterion.

Comparison with LSDR algorithms. We first compare CLEMS with four LSDR algorithms introduced in Sec-
tion 2: PLST (Tai & Lin, 2012), CPLST (Chen & Lin, 2012), FaIE (Lin et al., 2014), and SLEEC (Bhatia et al., 2015). We do not include CS (Hsu et al., 2009) in the comparison because PLST is reportedly better than CS (Tai & Lin, 2012).

We couple CLEMS, PLST, CPLST, and FaIE with $M$ single-target random forest regression (Breiman, 2001). The maximum depth of the trees is selected from $\{5, 10, 15, ..., 35\}$ using the validation set. Because 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 criteria during training. That is, our modified SLEEC can be viewed as “semi-cost-sensitive” algorithm that learns the threshold according to the given criteria.

Figures 4 and 5 show the results of F1 score and Accuracy score. As the embedded dimension $M$ increases, all the algorithms reach better performance because of the better preservation of cost 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, except for CAL500, which demonstrates the usefulness to consider the cost information during embedding (CLEMS) rather than after the embedding (SLEEC). Note that CAL500 is a special dataset because every label vector appears only once ($f_i = 1$).

The results of Rank loss are shown in Figure 6. CLEMS again reaches the best performance in most of the cases, which justifies its validity for asymmetric criteria through the mirroring trick.

Comparison with LSDE algorithms. Next, we compare CLEMS with LSDE algorithms, including ECC-based algorithms (Ferng & Lin, 2013) and RA/EL (Tsoumakas et al., 2011a). For ECC-based algorithms, we consider two promising error-correcting codes, repetition code (ECC-RREP) and Hamming on repetition code (ECC-HAMR) in the original work (Ferng & Lin, 2013). Note that RA/EL is actually equivalent to ECC-RREP (Ferng & Lin, 2013). We couple ECC-RREP and ECC-HAMR with random forest classification (Breiman, 2001).

Figure 7 shows the results of F1 score. Note that $M$ for CLEMS and LSDE algorithms are different. 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 8 and 9). The results again justify the superiority of CLEMS.

Although for both LSDR and LSDE, CLEMS reaches better performance as $M$ increases, the improvement becomes small when $M \geq 100\%$ of $K$. The results suggest that LSDE is not necessary to CLEMS. The embedded dimension $M = 100\%$ of $K$ is sufficiently good for most of the cases.

Candidate set. Now, We discuss the influence of the candidate set. In Section 3, we proposed to embed $S_{\text{train}}$ instead of $\mathcal{Y}$. To verify the goodness of the choice, we compare CLEMS with $S_{\text{train}}$ (CLEMS-train) and $S_{\text{all}}$ (CLEMS-all) as candidate set, where $S_{\text{all}}$ denotes the set of label vectors that appear in the training instances and the test instances. We use CLEMS-all to estimate the benefit of “peeping” the test label vectors and embedding them in advance.

The results are shown in Figure 10, 11, and 12. From the figures, we see that the improvement of CLEMS-all over CLEMS-train is small and insignificant. The results imply that $S_{\text{train}}$ readily allows the nearest-neighbor decoding function to make sufficiently good choices.

### Table 1. Comparison with CFT

|                  | F1 score (%) | Accuracy score (%) | Rank loss (%) |
|------------------|--------------|--------------------|--------------|
|                  | CFT         | CLEMS              | CFT         | CLEMS        |
| enso             | 0.640       | 0.676              | 0.555       | 0.580        |
| scene            | 0.703       | 0.770              | 0.656       | 0.760        |
| yeast            | 0.649       | 0.671              | 0.543       | 0.568        |
| birds            | 0.601       | 0.674              | 0.586       | 0.642        |
| med              | 0.635       | 0.814              | 0.613       | 0.586        |
| enron            | 0.557       | 0.606              | 0.548       | 0.568        |
| CAL              | 0.371       | 0.419              | 0.237       | 0.273        |
| EUR              | 0.456       | 0.670              | 0.450       | 0.550        |
| EUR              | 0.670       | 0.450              | 0.450       | 0.550        |
| EUR              | 120.6         | 563.4              | 120.6         | 563.4        |
| EUR              | 120.6         | 563.4              | 120.6         | 563.4        |
| EUR              | 120.6         | 563.4              | 120.6         | 563.4        |

Comparison with CFT. Finally, we compare CLEMS with CFT (Li & Lin, 2014), the state-of-the-art cost-sensitive MLC algorithm. We do not include PCC (Dembczynski et al., 2010; 2011) because there is no efficient rule for Accuracy score, and CFT is reportedly competitive to PCC (Li & Lin, 2014). Table 1 shows the average test results when $M = K$, and demonstrates that CLEMS outperforms CFT in most of the cases. The reason is that CLEMS globally locates the hidden structure of labels, while CFT is a chain-based algorithm and only locally discovers the relation between labels. The results suggest CLEMS to be a promising algorithm for cost-sensitive MLC.

5. Conclusion

We propose a novel cost-sensitive label embedding algorithm called cost-sensitive label space 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 can-
didate 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.
Figure 4. F1 score (↑) with the 95% confidence interval of CLEMS and LSDR algorithms

Figure 5. Accuracy (↑) with the 95% confidence interval of CLEMS and LSDR algorithms

Figure 6. Rank loss (↓) with the 95% confidence interval of CLEMS and LSDR algorithms
Cost-sensitive Label Embedding for Multi-label Classification

Figure 7. F1 score (↑) with the 95% confidence interval of CLEMS and LSDE algorithms

Figure 8. Accuracy (↑) with the 95% confidence interval of CLEMS and LSDE algorithms

Figure 9. Rank loss (↓) with the 95% confidence interval of CLEMS and LSDE algorithms
Figure 10. F1 score (↑) with the 95% confidence interval of CLEMS-train and CLEMS-all

Figure 11. Accuracy (↑) with the 95% confidence interval of CLEMS-train and CLEMS-all

Figure 12. Rank loss (↓) with the 95% confidence interval of CLEMS-train and CLEMS-all
Cost-sensitive Label Embedding for Multi-label Classification

References

Barutçuoğlu, Z., Schapire, R. E., and Troyanskaya, O. G. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

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

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

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

Chen, Y.-N. and Lin, H.-T. Feature-aware label space dimension reduction for multi-label classification. In *NIPS*, pp. 1538–1546, 2012.

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

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

Dembczynski, K., Waegeman, W., Cheng, W., and Hüllermeier, E. An exact algorithm for f-measure maximization. In *NIPS*, pp. 1404–1412, 2011.

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

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

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

Li, C.-L. and Lin, H.-T. Condensed filter tree for cost-sensitive multi-label classification. In *ICML*, pp. 423–431, 2014.

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

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

Tai, F. and Lin, H.-T. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

Trohidis, K., Tsoumakas, G., Kalliris, G., and Vlahavas, I. P. Multi-label classification of music into emotions. In *ISMIR*, pp. 325–330, 2008.

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

Tsoumakas, G., Katakis, I., and Vlahavas, I. P. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pp. 667–685, 2010.

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

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

Weston, J., Chapelle, O., Vapnik, V., Elisseeff, A., and Schölkopf, B. Kernel dependency estimation. In *NIPS*, pp. 873–880, 2002.
Appendix A. Proof of Theorem 1

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

\[
\delta(c(y, y_q))^2 \leq 5 \left( \frac{d(z, z_q) - \delta(c(y, y_q))}{d(z, \tilde{z})} + \right) \left( \frac{d(z, \tilde{z})}{5} \right)^2.
\]

*Proof.* Since \(z_q\) is the nearest neighbor of \(\tilde{z}\), we have

\[
d(z, \tilde{z}) \geq \frac{1}{2} d(z, z_q).
\]

Thus, the theorem.

This implies the theorem.

Appendix B. Datasets

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