Multi-label Classification with Principal Label Space Transformation

Farbound Tai and Hsuan-Tien Lin

{b94901176, htlin}@ntu.edu.tw

Department of Computer Science, National Taiwan University

Keywords: multi-label classification, hypercube view, regression

Abstract

We consider a hypercube view to perceive the label space of multi-label classification problems geometrically. The view allows us to not only unify many existing multi-label classification approaches, but also design a novel algorithm, Principal Label Space Transformation (PLST), which captures key correlations between labels before learning. The simple and efficient PLST relies on only the singular value decomposition as the key step. We derive the theoretical guarantee of PLST and evaluate its empirical performance using real-world data sets. Experimental results demonstrate that PLST is faster than the traditional binary relevance approach and is superior to the modern compressive sensing approach in terms of both accuracy and efficiency.

1 Introduction

Multi-label classification problems naturally arise in domains such as text mining, vision, or bio-informatics. For instance, a document is usually associated with more than one category; a picture often includes many objects; a gene is usually multi-functional.

The problem generalizes the traditional multi-class classification problem—the former allows a set of labels to be associated with an instance while the latter allows only one. Because of the wide range of potential applications in genomics (Barutcuoglu et al., 2006; Vens et al., 2008), scene classification (Boutell et al., 2004), video segmentation (Snoek et al., 2006), music classification (Trohidis et al., 2008) and text categorization (Schapire and Singer, 2000), multi-label classification is attracting more and more research attention.

Existing multi-label classification approaches usually fall into one of two categories (Tsoumakas et al., 2010a): *Algorithm Adaptation* or *Problem Transformation*. As its name suggests, Algorithm Adaptation directly extends some specific algorithms to solve the multi-label classification problem. Typical members of Algorithm Adaptation include Adaboost.MH (Schapire and Singer, 2000), Multi-label C4.5 (Clare and King, 2001) and ML-KNN (Zhang and Zhou, 2007). Problem Transformation (sometimes also called reduction) approaches, on the other hand, transform the multi-label classification problem to one or more reduced tasks. Typical members of Problem Transformation include Label Power-set, Binary Relevance and Label Ranking (Fürnkranz et al., 2008; Elisseeff and Weston, 2002). Label Power-set reduces multi-label classification to multi-class classification by treating each distinct label set as a unique multi-class label. Binary Relevance, also known as one-versus-all, reduces multi-label classification to many different binary classification tasks, each for one of the labels. Label Ranking approaches transform the multi-label classification problem to the task of ranking all the labels by relevance and the task of determining a threshold of relevance. As can be seen from above, an advantage of Problem Transformation over Algorithm Adaptation is that any algorithm which deals with the reduced tasks can be easily extended to multi-label classification via the transformation.

In this paper, we discuss Problem Transformation approaches from a special perspective: the hypercube view. The view describes all possible label sets in the multi-label classification problem as the vertices of a high-dimensional hypercube. The view not only unifies Label Power-set, Binary Relevance and Label Ranking under the same framework, but also allows us to design better methods that make use of the geometric properties of those label-set vertices. We demonstrate the use of the hypercube view with a novel method, Principal Label Space Transformation (PLST), which captures the

key correlations between labels using a flat (a low-dimensional linear subspace) in the high-dimensional space. The method only uses a simple linear encoding of the vertices and a simple linear decoding of the predictions, both easily computed from the Singular Value Decomposition (SVD) of a matrix composed of the label-set vertices. Moreover, by keeping only the key correlations, PLST can dramatically decrease the number of reduced tasks to be solved without loss of prediction accuracy. Such a computational advantage is especially important for scaling up multi-label classification algorithms to a larger number of labels (Tsoumakas et al., 2010a).

Another recent work, multi-label prediction via Compressive Sensing (Hsu et al., 2009), also seeks to perform multi-label classification with a linear encoding of the label-sets vertices. Compressive Sensing operates under the assumption of sparsity in the label sets and thus can describe the label-set vertices with a small number of linear random projections as its encoding. Although the encoding component of Compressive Sensing is linear, the decoding component is not. In particular, for each incoming test instance, Compressive Sensing needs to solve an optimization problem with respect to its sparsity assumption. That is, Compressive Sensing can be time consuming during prediction. In our experiments, we will demonstrate that PLST is not only more efficient but also more accurate than Compressive Sensing.

As mentioned by Tsoumakas et al. (2010a) and Hsu et al. (2009), large-scale multi-label classification poses a computational challenge as even the efficient Binary Relevance can require thousands of classifiers. Other Problem Transformation methods such as Label Ranking or Label Power-set come with computational complexity that grows polynomially or exponentially with the number of labels and are thus not feasible for the challenge. PLST can be viewed as a linear dimension-reduction method in the label space for conquering both the training and the prediction parts of the challenge; Compressive Sensing solves the training part of the challenge, but not the prediction part. For dimension reduction in the label space, there are also methods that are based on non-linear dimension reduction. A representative method is to use the topic model to group labels into a small set of topics (Law et al., 2010). The method solves the prediction part of the challenge (predicting the few topics instead of the many labels) but training a topic model is a non-trivial task in computation. We thus focus our attention only on linear dimension reduction methods like PLST for efficiency of both training

and prediction.

When viewing multi-label classification as a special case of the structured output prediction problem, the kernel dependency estimation algorithm (KDE; Weston et al., 2002) could be applied to multi-label classification by designing appropriate kernels for the label space (Dembczynski et al., 2010a). Interestingly, PLST is equivalent to the linear form of KDE for the special case. The linear PLST avoids the computationally-expensive pre-image problem (Weston et al., 2002) in the general non-linear KDE. To the best of our knowledge, neither the linear form of KDE nor its application to multi-label classification has been seriously studied. Our work provides a solid understanding to the linear form of KDE with novel theoretical and empirical results.

Some other related methods come from works on multi-task learning, which takes multi-label classification as a simple special case. Ando and Zhang (2005) propose a multi-task learning method, SVD-based alternating structure optimization (SVD-ASO), that simultaneously optimizes a loss function of all the tasks (label predictions) and performs dimension reduction to learn a compact joint representation of the feature space. A similar formulation is taken for multi-label classification by Ji et al. (2010). While both our proposed PLST method and SVD-ASO uses SVD as a core component, they are different as the former performs dimension reduction on the label space while the latter focuses on the feature space.

The paper is organized as follows. In Section 2, we give a formal setup of the multi-label classification problem and introduce the hypercube view. Then, in Section 3, we unify Binary Relevance and CS under the same framework via the hypercube view and describe our proposed method: PLST. Finally, we present the experimental results in Section 4 and conclude in Section 5.

2 Hypercube View

In the multi-label classification problem, we seek for a multi-label classifier that maps the input vector $\mathbf{x} \in \mathbb{R}^d$ to a set of label $\mathcal{Y}$, where $\mathcal{Y} \subseteq \mathcal{L} = \{1, 2, \ldots, K\}$ with K being the number of classes. Consider a training set $\mathcal{S}$ that contains N training examples of the form $(\mathbf{x}_n, \mathcal{Y}_n)$. Multi-label classification aims at using $\mathcal{S}$ to find a multi-label classifier $g\colon \mathbb{R}^d \to 2^{\mathcal{L}}$ such that $g(\mathbf{x})$ predicts $\mathcal{Y}$ well on any future test example $(\mathbf{x}, \mathcal{Y})$.

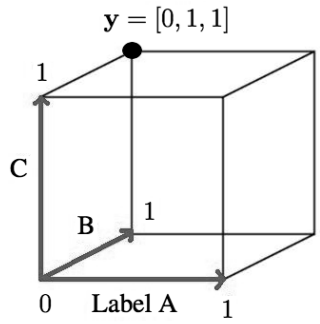


Figure 1: a hypercube with $K = 3$

The key of the *hypercube view* is to represent the label set $\mathcal{Y}$ by a vector $\mathbf{y} \in \{0, 1\}^K$, where the k-th component of $\mathbf{y}$ is 1 if and only if $k \in \mathcal{Y}$. Then, as shown in Fig. 1, we can visualize each $\mathcal{Y}$ as a vertex of a K-dimensional hypercube. The k-th component of $\mathbf{y}$ corresponds to an axis of the hypercube, which represents the presence or absence of a label k in $\mathcal{Y}$. We will use $\mathcal{Y}$ and its corresponding $\mathbf{y}$ interchangeably in this paper. The hypercube view allows us to unify many existing Problem Transformation approaches, as discussed below.

Hypercube View of Label Power-set: One of the simplest approaches to multi-label classification is Label Power-set, as shown in Algorithm 1.

Algorithm 1 Label Power-set

1. pre-processing: map each vertex $\mathbf{y}_n$ (or each label-set $\mathcal{Y}_n$) to a hyper-label $y_n \in \{1, 2, \cdots, 2^K\}$ with a bijection function B.
2. training: learn a multi-class classifier $g_c(\mathbf{x})$ from $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$.
3. predicting: for each $\mathbf{x}$, return $B^{-1}\big(g_c(\mathbf{x})\big)$.

In particular, Label Power-set simply treats each vertex of the hypercube as a different hyper-label and performs regular multi-class classification with the hyper-labels. That is, Label Power-set essentially breaks the structure of the hypercube and does not consider the relations (edges) between the vertices. The approach is often criticized for the large number of possible hyper-labels and the relatively few number of examples

per hyper-label, which may degrade the learning performance.

Hypercube View of Binary Relevance: Another straight-forward approach to multi-label classification is Binary Relevance. This approach decomposes the original multi-label problem into K isolated relevance-learning sub-tasks, as shown in Algorithm 2.

Algorithm 2 Binary Relevance

1. training: for $k = 1$ to K, learn a relevance function $r_k(\mathbf{x})$ from $\{\big(\mathbf{x}_n, \mathbf{y}_n[k]\big)\}_{n=1}^{N}$.
2. predicting: for each input vector $\mathbf{x}$, compute $\mathbf{r}(\mathbf{x}) \equiv \big[r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_K(\mathbf{x})\big]$. Then, return round$\Big(\mathbf{r}(\mathbf{x})\Big)$, where round$(\cdot)$ maps each component of the vector to the closest value in $\{0, 1\}$.

Using the hypercube view, the k-th iteration of Binary Relevance can be thought as projecting the vertices to the k-th dimension (axis) before training. In addition, the relevance vector $\mathbf{r}(\mathbf{x}) \equiv [r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_K(\mathbf{x})]$ can be viewed as a point in $\mathbb{R}^K$ and the round$(\cdot)$ operation maps the point to the closest vertex of the hypercube in terms of the ℓ_1-distance.

Despite its effectiveness, Binary Relevance is often criticized for neglecting the correlation between labels, which may carry useful information in multi-label classification tasks. Furthermore, the training complexity of Binary Relevance is linear to the number of labels K, which can still be expensive if K is too large. Recently, Hsu et al. (2009) attempted to address this problem through Compressive Sensing, which will be discussed later in this section.

Hypercube View of Label Ranking: As shown in Algorithm 3, Label Ranking approaches learn two components (jointly or separately) from the multi-label classification data set: the order of label relevance that is often represented by a scoring function on the labels, and the threshold for label presence. Note that Binary Relevance is a special case of Label Ranking when taking the relevance function $\mathbf{r}(\mathbf{x})$ as the scoring function and a naïve threshold at 0.5 per label.

Using the hypercube view, the ordering component in Label Ranking can be thought as learning a length-K path from $[0, 0, \cdots, 0]$ to $[1, 1, \cdots, 1]$ using the hypercube

Algorithm 3 Label Ranking

1. training: learn a scoring function $\mathbf{s}(\mathbf{x}) \equiv \left[s_1, s_2, \cdots, s_K\right]$ from $\{\left(\mathbf{x}_n, \mathbf{y}_n\right)\}_{n=1}^{N}$ that gives a score s_i to each label l_i in $\mathcal{L}$ and a threshold function $\mathbf{t}$ that converts scores above a certain threshold to 1 and the rest to 0.

2. predicting: for each $\mathbf{x}$, return $\mathbf{t}(\mathbf{s}(\mathbf{x}))$.

edges. Each vertex $\mathbf{y}_n$ in the training examples then represent multiple length-K edge-paths that go through $\mathbf{y}_n$, whose ℓ_1 norm indicates the desired thresholding level. An early representative of Label Ranking is Rank-SVM (Elisseeff and Weston, 2002), in which the scores are obtained via the relevance function in Binary Relevance and the thresholding function comes from estimating the number of relevant labels. Another popular approach is Calibrated Label Ranking (Fürnkranz et al., 2008), in which the scoring function is learned from a pairwise comparison of the labels and the thresholding function comes from the score of an additional virtual label that is added during training.

Hypercube View of Compressive Sensing: Under the assumption that the label sets $\mathcal{Y}$ are sparse (i.e. containing only a few elements), it is possible to compress the label sets and learn to predict the compressed labels instead. Such a possibility allows Compressive Sensing (CS; Hsu et al., 2009) to reduce the number of sub-tasks in Binary Relevance to be computationally feasible for data sets with a large K. In particular, each label set $\mathcal{Y}$ (vertex $\mathbf{y}$) can be taken as a K-dimensional signal. The theory of compressive sensing states that when the signals are sparse, one does not need to sample at the Nyquist rate in order to accurately recover the original signals. A vector is said to be *s-sparse* if it contains at most *s* nonzero entries. Thus, as the sketch of CS in Algorithm 4 shows, when all $\mathbf{y}$ contain only a few 1's, CS only needs to solve $M \ll K$ sub-tasks instead of K for multi-label classification.

Using the hypercube view, the m-th iteration of CS can be thought as projecting the vertices to a random direction before training. Because $M \ll K$, the subspace explored by CS is much smaller than the space that the hypercube resides in. CS is able to work

Algorithm 4 Compressive Sensing

1. pre-processing: compress $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ to $\{(\mathbf{x}_n, \mathbf{h}_n)\}$, where $\mathbf{h} = \mathbf{P}_s \cdot \mathbf{y}$ using an M by K random projection matrix $\mathbf{P}_s$ with M determined by the assumed sparsity level s. Each label-set $\mathbf{y}_n$ is assumed to be s-sparse.

2. training: for $m = 1$ to M, learn a function $r_m(\mathbf{x})$ from $\{(\mathbf{x}_n, \mathbf{h}_n[m])\}_{n=1}^N$.

3. prediction: for each input vector $\mathbf{x}$, compute $\mathbf{r}(\mathbf{x}) = \big[r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_M(\mathbf{x})\big]$. Then, obtain a sparse vector $\hat{\mathbf{y}}$ such that $\mathbf{P}_s \cdot \hat{\mathbf{y}}$ is "closest" to $\mathbf{r}(\mathbf{x})$ using an optimization algorithm. Finally, return $\hat{\mathbf{y}}$.

on such a small subspace because of the label-set sparsity assumption, which implies that only a limited number of vertices in the hypercube are relevant for the multi-label classification task.

Although the random projection in the pre-processing step of CS is efficient, the prediction step requires solving an optimization problem for every coming input vector $\mathbf{x}$. Such a prediction step is very time consuming. In addition, the assumption on label-set sparsity puts a restriction on the practical use of the CS approach.

Hypercube View of Topic Modeling: Under the assumption that $P(\mathbf{y}|\mathbf{x})$, the probability of getting a particular label-set vector $\mathbf{y}$ given $\mathbf{x}$, can be modeled through a hidden random variable $z \in \{1, 2, \cdots, M\}$ called the "topic", topic modeling decomposes $P(\mathbf{y}|\mathbf{x})$ to

$$P(\mathbf{y}|\mathbf{x}) = \sum_{m=1}^{M} P(\mathbf{y}|z=m) \underbrace{P(z=m|\mathbf{x})}_{r_m(\mathbf{x})}.$$

In the original work of topic modeling (Law et al., 2010), the former term $P(\mathbf{y}|z=m)$ is learned with latent Dirichlet allocation (Blei et al., 2003); the latter term $P(z=m|\mathbf{x})$ is learned with the maximum entropy classifier (Csiszár, 1995). Note that a topic is essentially a cluster of label-set vertices $\mathbf{y}$. Thus, more generally, any probabilistic clustering algorithm can be used to get the former term $P(\mathbf{y}|z=m)$ and any probabilistic classification algorithm can be used to get the latter term $r_m(\mathbf{x}) = P(z=m|\mathbf{x})$, as shown in Algorithm 5.

The original work of topic modeling (Law et al., 2010) treats $P(\mathbf{y}|\mathbf{x})$ as a standalone

probabilistic classifier and does not discuss much about its deterministic decoding. One simple procedure of making a deterministic prediction, as illustrated in Algorithm 5, is to round from the expected value of $\mathbf{y}$ given $\mathbf{x}$. The procedure equivalently finds the best determistic prediction $\hat{\mathbf{y}}$ subject to $P(\mathbf{y}|\mathbf{x})$ in terms of the Hamming loss, a popular performance measure that will be discussed later in this section.

Algorithm 5 Generalized Topic Modeling

1. clustering: cluster $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ to $\{(\mathbf{x}_n, \mathbf{h}_n)\}$, where $\mathbf{h}_n[m]$ represents the probability of $\mathbf{y}_n$ residing in the m-th cluster characterized by $P(\mathbf{y}|z = m)$. Let $\mathbf{p}_m$ be the center of the cluster, i.e., the expected value of $P(\mathbf{y}|z = m)$.

2. training: for $m = 1$ to M, learn a probabilistic classifier $\mathbf{r}(\mathbf{x})$ from $\{(\mathbf{x}_n, \mathbf{h}_n[m])\}_{n=1}^{N}$, where the m-th component of $\mathbf{r}(\mathbf{x})$ indicates the probability of $\mathbf{x}$ being in cluster m.

3. prediction: for each input vector $\mathbf{x}$, compute $\mathbf{r}(\mathbf{x}) = \big[r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_M(\mathbf{x})\big]$. Then, compute
$$\tilde{\mathbf{y}} = \sum_{m=1}^{M} r_m(\mathbf{x}) \cdot \mathbf{p}_m$$
and return $\hat{\mathbf{y}} = \text{round}(\tilde{\mathbf{y}})$.

Let the vector $\mathbf{p}_m$ be the expected value of $\mathbf{y}$ given $z = m$, which is the center of the cluster. The vector is inside the hypercube and can be viewed as the representative point of the cluster. From the hypercube view, generalized topic modeling identifies the representative point of each cluster (that is able to capture the nearby vertices of the hypercube), and then adopts a probabilistic multi-class classifier to map the input vector $\mathbf{x}$ to a distribution of those representative points. An extreme case of topic modeling is thus Label Power-set (Algorithm 1), which takes each vertex as its own cluster and a deterministic classifier g as the probabilistic classifier $\mathbf{r}$. If the label-set vectors $\mathbf{y}$ form a small number of meaningful clusters in R^K, topic modeling can use the property to predict efficiently and effectively. From the geometric perspective, however, clustering in a K dimensional space is a non-trivial and a time-consuming task when K is large because of the curse of dimensionality.

Hypercube View of Kernel Dependency Estimation: Consider a (high-dimensional) transform function ϕ: $R^K \rightarrow \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space; let $K(\mathbf{y}, \mathbf{y}')$ embeds the inner product $\langle \phi(\mathbf{y}), \phi(\mathbf{y}') \rangle$ that represents the similarity between $\mathbf{y}$ and $\mathbf{y}'$. Under the assumption that $\phi(\mathbf{y})$ approximately resides in an M-dimensional flat (a linear subspace) within $\mathcal{H}$, the kernel dependency estimation approach (Weston et al., 2002) implicitly locates the reference point $\mathbf{o}$ and the basis vectors $\{\mathbf{u}_m\}_{m=1}^M$ of the flat using the kernel K. Then, the approach transforms $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ to $\{(\mathbf{x}_n, \mathbf{h}_n)\}$ by $\mathbf{h}_n[m] = \langle \phi(\mathbf{y}_n) - \mathbf{o}, \mathbf{u}_m \rangle$. For each $m = 1, 2, \cdots, M$, the approach then performs kernel ridge regression (Saunders et al., 1998) from $\mathbf{x}$ to $\mathbf{h}[m]$ to get a regressor $r_m(\mathbf{x})$. During prediction, the approach returns the best $\mathbf{y}$ such that each $\langle \phi(\mathbf{y}) - \mathbf{o}, \mathbf{u}_m \rangle \approx r_m(\mathbf{x})$, as shown in Algorithm 6.

Algorithm 6 Kernel Dependency Estimation

1. decomposition of output space: perform kernel principal component analysis on $\mathbf{y}$ with some kernel function that embeds the transformation ϕ; transform $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ to $\{(\mathbf{x}_n, \mathbf{h}_n)\}$, where $\mathbf{h}_n[m] = \langle \phi(\mathbf{y}_n) - \mathbf{o}, \mathbf{u}_m \rangle$ with $\mathbf{o}$ being the mean of $\phi(\mathbf{y}_n)$ and $\mathbf{u}_m$ being the m-th principal component.

2. training: for $m = 1$ to M, learn a function $r_m(\mathbf{x})$ from $\{(\mathbf{x}_n, \mathbf{h}_n[m])\}_{n=1}^N$ with kernel ridge regression (or more generally, any regression algorithm).

3. prediction: for each input vector $\mathbf{x}$, compute $\mathbf{r}(\mathbf{x}) = \left[r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_M(\mathbf{x})\right]$ Then, return

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \{0,1\}^K}{\operatorname{argmin}} \left\| \left[\langle \phi(\mathbf{y}) - \mathbf{o}, \mathbf{u}_1 \rangle, \langle \phi(\mathbf{y}) - \mathbf{o}, \mathbf{u}_2 \rangle, \cdots, \langle \phi(\mathbf{y}) - \mathbf{o}, \mathbf{u}_M \rangle \right] - \mathbf{r}(\mathbf{x}) \right\|. \quad (1)$$

Consider a point $\mathbf{y}' \in R^K$. If $\phi(\mathbf{y}')$ falls in the M-dimensional flat in $\mathcal{H}$,

$$\langle \phi(\mathbf{y}') - \mathbf{o}, \mathbf{u}_m \rangle = 0 \text{ for } m = 1, 2, \cdots, M.$$

That is, the points $\mathbf{y}'$ reside in a hypersurface defined from an intersection of M hypersurfaces $\langle \phi(\mathbf{y}') - \mathbf{o}, \mathbf{u}_m \rangle = 0$ in R^K. From the hypercube view, kernel dependency estimation assumes that the vertices $\mathbf{y}$ are close to a hypersurface that corresponds to some M-dimensional flat in $\mathcal{H}$, and then performs learning on the flat instead of in

the original space. Because the hypersurface is usually nonlinear, the prediction procedure (1) in Algorithm 6 is a challenging optimization task and can be time consuming.

The key geometric objects used for modeling multi-label classification in the representative PT approaches above are summarized in the Table 1.

Table 1: Geometric Objects behind PT Approaches

approach	geometric objects
label powerset	vertices
binary relevance	axes
label ranking	edge-paths
compressive sensing	flat that approximates close-to-origin vertices
topic modeling	cluster of vertices
kernel density estimation	hypersurface that approximates vertices

The hypercube view not only unifies the PT approaches above, but also offers a geometric interpretation for the Hamming loss, which is commonly used to evaluate multi-label classifiers (Dembczynski et al., 2010b). Assume that the target label-set vertex is $\mathbf{y}$ and the predicted vertex is $\hat{\mathbf{y}}$, Hamming loss is defined as

$$\Delta(\hat{\mathbf{y}}, \mathbf{y}) \equiv \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{y}}[k] \oplus \mathbf{y}[k].$$

An alternative way to look at Hamming loss is

$$\Delta(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{K} \|\hat{\mathbf{y}} - \mathbf{y}\|_1.$$

That is, Hamming loss is simply a scaled ℓ_1-distance between $\hat{\mathbf{y}}$ and $\mathbf{y}$. The distance also corresponds to the shortest edge-path to walk from $\hat{\mathbf{y}}$ to $\mathbf{y}$ on the hypercube. The hypercube view justifies that Hamming loss can be a suitable error measure for algorithms that operates with respect to the space or the structure of the hypercube, such as Binary Relevance, CS (decoding to the closest vertex) or Label Ranking (thresholding edge-paths).

3 Proposed Approach

From the hypercube view, CS relies on label-set sparsity to consider a small number of vertices of the hypercube. Our proposed approach stems from the same consideration, but without requiring the assumption on label-set sparsity. From the hypercube view, there are 2^K vertices of the hypercube, and each training example $(\mathbf{x}_n, \mathbf{y}_n)$ occupies only one vertex $\mathbf{y}_n$. In large multi-label classification data sets, it is typical for K to exceed hundreds or even thousands. Then, usually the number of training examples $N \ll 2^K$. In addition, not all 2^K vertices are needed for the multi-label classification problem because of the possible hierarchy, correlation or hidden relationship between the different labels. For instance, if classes labeled 1 and 2 are disjoint sub-classes of class 3, only 3 vertices out of the 8 candidates are needed: $[0, 0, 0]$, $[1, 0, 1]$, $[0, 1, 1]$. Thus, during training, relatively few vertices will be occupied by a decent number of examples. We call this phenomenon *hypercube sparsity* to distinguish it from the *label-set sparsity* that CS uses.

Note that label-set sparsity implies hypercube sparsity, but not vice versa. By definition, for a data set with label-set sparsity at s, all the hypercube vertices with more than s labels are unoccupied by training examples—the phenomenon of hypercube sparsity. For instance, if a data set is label-set sparse at $s = 2$, then such a data set is also hypercube sparse because the number of occupied vertices is at most $\binom{K}{2} + K + 1 \ll 2^K$.

On the other hand, hypercube sparsity does not necessarily imply label-set sparsity, because the few occupied label-set vertices may contain many labels. For instance, a data set with all label sets containing at least $(K-1)$ labels is hypercube sparse with the number of occupied vertices being at most $K + 1 \ll 2^K$, but is by no means label-set sparse.

Because of the hypercube sparsity, multi-label classification algorithms do not need to learn with the entire hypercube in $\mathbb{R}^K$ and can focus on some vertices of the hypercube (and their neighborhood area) instead. For instance, the Pruned Label Power-set approach (Read et al., 2008), which is a variant of the usual Labl Power-set, only considers vertices occupied by enough examples during training; topic modeling (Law et al., 2010) groups the occupied vertices as clusters; kernel dependency estimation (Weston et al., 2002) describes the occupied vertices by a (possibly non-linear) hypersurface. In other

words, hypercube sparsity allows dimensionality reduction in the label space without loss of prediction performance.

3.1 Linear Label Space Transformation

As shown in Algorithm 4, under the assumption label-set sparsity, CS is able to compress (reduce) the label space using an M by K random projection matrix $\mathbf{P}_s$. The random projection matrix defines a flat, which is a linear subspace of R^K with at most M dimensions. When taking the hypercube sparsity into account, could a flat also be helpful in modeling the occupied vertices in lower dimensions?

Let us first consider the case when there are two labels $\mathbf{y}[1]$ and $\mathbf{y}[2]$ and they are always the same for every example—a fully-correlated and equivalent relationship. The equivalence causes hypercube sparsity and hence only vertices $[0, 0]$ and $[1, 1]$ are needed to model the multi-label classification problem. Intuitively, for the particular problem, it suffices to predict only $\mathbf{y}[1]$ and then replicate the prediction for $\mathbf{y}[2]$. The intuition corresponds to making predictions by projecting $\mathbf{y}$ to the line (a 1-dimensional flat) $[0, 0] + \alpha_1[1, 1]$ and back—a form of linear dimension reduction.

We can extend the case to a multi-label classification problem that occupies three vertices: $\{[0, 0], [0, 1], [1, 1]\}$. In other words, examples with $\mathbf{y}[1] = 1$ is a subset of examples with $\mathbf{y}[2] = 1$, i.e., an inclusive relationship. Consider a line

$$\left[\tfrac{1}{2}, 0\right] + \alpha\,[1, 1]$$

The vertex $[0, 0]$ projects to a point $[\frac{1}{4}, -\frac{1}{4}]$, which is of $\alpha = -\frac{1}{4}$; the vertex $[0, 1]$ projects to a point $[\frac{1}{4}, \frac{3}{4}]$, of $\alpha = \frac{1}{4}$; the vertex $[1, 1]$ projects to a point $[\frac{5}{4}, \frac{3}{4}]$, of $\alpha = \frac{3}{4}$. Then, a simple dimension reduction procedure that performs regression from $\mathbf{x}$ to α on the flat with a low-error regressor $\mathbf{r}(\mathbf{x})$ and decodes the regression result by[1]

$$\text{round}(\left[\tfrac{1}{2}, 0\right] + \mathbf{r}(\mathbf{x})\,[1, 1])$$

would not incur any loss of information. That is, when the 2-dimensional hypercube is occupied by 3 out of the 4 vertices, there exists a 1-dimensional flat that describes the occupied vertices well.

[1] The detailed procedure would be discussed later.

Other types of vertex relations, which cause different patterns of hypercube sparsity, can also be captured by a flat. For instance, if two labels $\mathbf{y}[1]$ and $\mathbf{y}[2]$ are not fully correlated but just highly correlated with a positive correlation, a line

$$[0, 0] + \alpha[1, 1]$$

can be used to capture the key correlation and hence reaching satisfactory performance. Another type of vertex relation that can be captured is hirarchical. For instance, we have discussed that when classes labeled 1 and 2 are disjoint sub-classes of class 3, only 3 vertices of the hypercube would be occupied: $(0, 0, 0)$, $(1, 0, 1)$, $(0, 1, 1)$. Intuitively, the two dimensional flat

$$[0, 0, 0] + \alpha[1, 0, 1] + \beta[0, 1, 1]$$

perfectly describes the three vertices in the 3-dimensional hypercube. In fact, even when sub-classes 1 and 2 are not disjoint, for which 4 vertices on the hypercube would be occupied: $[0, 0, 0]$, $[1, 0, 1]$, $[0, 1, 1]$, $[1, 1, 1]$, it can be shown that encoding the 4 vertices by a 2-dimensional flat

$$[0.5, 0.5, 0.5] + \alpha[0.55, 0.4, 0.8] + \beta[-0.6, 0.8, 0]$$

and decoding by rounding would not incure any loss of information when using low-error regressors.

Next, we study a simple framework that focuses on a linear subspace instead of the whole hypercube in $\mathbb{R}^K$. The framework takes an M-flat as the subspace and encodes each vertex $\mathbf{y}$ of the hypercube to a vector $\mathbf{h}$ under the coordinate system of the M-flat by projection. Then, the original multi-label classification problem with $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ becomes a multi-dimensional regression problem with $\{(\mathbf{x}_n, \mathbf{h}_n)\}_{n=1}^N$. After obtaining a multi-dimensional regressor $\mathbf{r}(\mathbf{x})$ that predicts $\mathbf{h}$ well, the framework will then map $\mathbf{r}(\mathbf{x})$ back to a vertex of the hypercube in $\mathbb{R}^K$ using some decoder D. As discussed earlier, Hamming loss is effectively the scaled ℓ_1 distance in the hypercube. The new regression problem minimizes the ℓ_2 distance in the hypercube, which upperbounds the scaled Hamming loss. The framework will be named *Linear Label Space Transformation*, as shown in Algorithm 7.

As discussed, CS seeks to reduce the number of regressors by considering a flat with $M \ll K$. Its projection matrix $\mathbf{P}$ is chosen randomly from an appropriate distribution

Algorithm 7 Linear Label Space Transformation

1. pre-processing: consider an M-flat described by a reference point $\mathbf{o}$ and an M by K projection matrix $\mathbf{P}$. Then, encode $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ to $\{(\mathbf{x}_n, \mathbf{h}_n)\}$, where $\mathbf{h}_n = \mathbf{P}(\mathbf{y}_n - \mathbf{o})$ corresponds to a vector under the coordinate system of the flat.
2. training: for $m = 1$ to M, learn a function $r_m(\mathbf{x})$ from $\{(\mathbf{x}_n, \mathbf{h}_n[m])\}_{n=1}^N$.
3. prediction: for each input vector $\mathbf{x}$, compute $\mathbf{r}(\mathbf{x}) = \big[r_1(\mathbf{x}), r_2(\mathbf{x}), \cdots, r_M(\mathbf{x})\big]$. Then, return $D(\mathbf{r}(\mathbf{x}))$ where $D\colon \mathbb{R}^M \to \{0,1\}^K$ is a decoding function from the M-flat to the vertices of the hypercube.

(such as Gaussian, Bernoulli, or Hadamard) and the reference point $\mathbf{o}$ of the flat is simply $\mathbf{0}$, the origin of $\mathbb{R}^K$ as well as the most label-set-sparse vertex. The decoding algorithm D corresponds to the reconstruction algorithm in the terminology of CS, and requires solving an optimization problem for each different $\mathbf{x}$.

3.2 Linear Label Space Transformation with Round-based Decoding

As discussed above, CS may suffer from its slow decoding algorithm, while the round-based decoding in Binary Relevance can be more efficient. Next, we study a special form of Linear Label Space Transformation that is coupled with an efficient round-based decoding scheme. In particular, the decoding scheme first maps a prediction vector $\mathbf{r}(\mathbf{x})$ under the coordinate system of the M-flat back to a corresponding point $\tilde{\mathbf{y}}$ in $\mathbb{R}^K$. Then, the scheme rounds $\tilde{\mathbf{y}}$ to the closest vertex $\hat{\mathbf{y}}$ of the hypercube in terms of the ℓ_1 distance. The resulting approach, as shown in Algorithm 8, is called *Linear Label Space Transformation with Round-based Decoding*, which works directly with the geometry between the M-flat and the hypercube, and can be viewed as a simple and efficient form of the general Linear Label Space Transformation.

Note that Binary Relevance is a special case of Linear Label Space Transformation. For Binary Relevance, we can set $\mathbf{P} = \mathbf{I}$ with an arbitrary reference point $\mathbf{o}$. The usual Binary Relevance operates with $M = K$, which means that many regressors r_m are needed when K is large. To reduce the number of regressors, we can also use

Algorithm 8 Linear Label Space Transformation with Round-based Decoding

1. pre-processing: consider an M-flat described by a reference point $\mathbf{o}$ and an orthonormal basis $\{\mathbf{p}_m\}_{m=1}^M$. Use $\mathbf{p}_m^T$ as the rows of an M by K projection matrix $\mathbf{P}$ in Linear Label Space Transformation.

2. training: simply run Linear Label Space Transformation.

3. prediction: after getting $\mathbf{r}(\mathbf{x})$ from Linear Label Space Transformation, compute

$$\tilde{\mathbf{y}} = \mathbf{o} + \sum_{m=1}^{M} r_m(\mathbf{x}) \cdot \mathbf{p}_m = \mathbf{o} + \mathbf{P}^T \cdot \mathbf{r}(\mathbf{x}).$$

   Then, return $\hat{\mathbf{y}} = \text{round}(\tilde{\mathbf{y}})$.

Binary Relevance with $M < K$ by taking only M rows of $\mathbf{I}$ as the projection matrix $\mathbf{P}$. One simple heuristic that exploits the label-set sparsity is to choose the M rows that correspond to the M-most frequent labels (i.e. with more 1) in the training data; other rows would simply be decoded by the corresponding components of $\mathbf{o}$ without using any information from the regressor. The approach with this heuristic will be called Partial Binary Relevance (PBR). PBR equivalently discards some of the labels when learning the regressors and hence the test performance may not be satisfactory. We will use PBR as a baseline approach that respects the label-set sparsity within the simple heuristic, and compare it with PLST and CS experimentally in Section 4.

Next, we analyze the performance of Algorithm 8. Note that the round-based decoder equivalently works by[2]

$$\hat{\mathbf{y}}[k] = \left[\!\!\left[\tilde{\mathbf{y}}[k] \geq \frac{1}{2} \right]\!\!\right]. \tag{2}$$

If round-based decoding is used, we can simply prove that Hamming loss between $\hat{\mathbf{y}}$ and the desired $\mathbf{y}$ is upper-bounded by a scaled squared distance between $\tilde{\mathbf{y}}$ and $\mathbf{y}$, as formalized below.

Lemma 1. *For the round-based decoder in* (2)*,*

$$\Delta(\hat{\mathbf{y}}, \mathbf{y}) \leq \frac{4}{K} \|\tilde{\mathbf{y}} - \mathbf{y}\|^2.$$

[2] $[\![\cdot]\!]$ is 1 if the inner condition is true and 0 otherwise.

Proof. For any given k,

$$\begin{aligned} & \hat{\mathbf{y}}[k] \oplus \mathbf{y}[k] \\ = \; & \left[\!\left[\tilde{\mathbf{y}}[k] \geq \frac{1}{2}\right]\!\right] \left[\!\left[\mathbf{y}[k] = 0\right]\!\right] + \left[\!\left[\tilde{\mathbf{y}}[k] < \frac{1}{2}\right]\!\right] \left[\!\left[\mathbf{y}[k] = 1\right]\!\right] \\ \leq \; & 4\left(\tilde{\mathbf{y}}[k] - \mathbf{y}[k]\right)^2 \left[\!\left[\mathbf{y}[k] = 0\right]\!\right] + 4\left(\tilde{\mathbf{y}}[k] - \mathbf{y}[k]\right)^2 \left[\!\left[\mathbf{y}[k] = 1\right]\!\right] \\ = \; & 4\left(\tilde{\mathbf{y}}[k] - \mathbf{y}[k]\right)^2. \end{aligned}$$

The desired result can be proved by averaging over all $k = 1, 2, \cdots, K$. □

Thus, if the error $\|\tilde{\mathbf{y}} - \mathbf{y}\|^2$ is small, the corresponding Hamming loss $\Delta(\hat{\mathbf{y}}, \mathbf{y})$ would also be small. That is, we could replace $\Delta(\hat{\mathbf{y}}, \mathbf{y})$ with a proxy error function $\|\tilde{\mathbf{y}} - \mathbf{y}\|^2$ when using the round-based decoder. Then, we can prove an upper bound on the per-example Hamming loss of Algorithm 8.

Theorem 1. *Consider any example* $(\mathbf{x}, \mathbf{y})$ *given at the prediction step of Algorithm 8. Then,*

$$\Delta(\hat{\mathbf{y}}, \mathbf{y}) \;\; \leq \;\; \frac{4}{K}\left(\|\mathbf{r}(\mathbf{x}) - \mathbf{h}\|^2 + \|\mathbf{y} - \mathbf{o} - \mathbf{P}^T\mathbf{h}\|^2\right), \tag{3}$$

where $\mathbf{h} \equiv \mathbf{P}(\mathbf{y} - \mathbf{o})$.

Proof. Using the fact that $\{\mathbf{p}_m\}_{m=1}^M$ forms an orthonormal basis, we can uniquely decompose $\mathbf{y} = (\mathbf{o} + \mathbf{P}^T\mathbf{h} + \mathbf{p}_\perp)$, where

$$\mathbf{p}_\perp = \mathbf{y} - \mathbf{o} - \mathbf{P}^T\mathbf{h} = (\mathbf{I} - \mathbf{P}^T\mathbf{P})(\mathbf{y} - \mathbf{o})$$

is orthogonal to every $\mathbf{p}_m$.

Then, from Algorithm 8, consider the point $\tilde{\mathbf{y}} = \mathbf{o} + \mathbf{P}^T\mathbf{r}(\mathbf{x})$. From Lemma 1,

$$\begin{aligned} \Delta(\hat{\mathbf{y}}, \mathbf{y}) \;\; & \leq \;\; \frac{4}{K}\|\tilde{\mathbf{y}} - \mathbf{y}\|^2 \\ & = \;\; \frac{4}{K}\left\|\mathbf{o} + \mathbf{P}^T\mathbf{r}(\mathbf{x}) - \mathbf{y}\right\|^2 \\ & = \;\; \frac{4}{K}\left(\left\|\mathbf{P}^T(\mathbf{r}(\mathbf{x}) - \mathbf{h})\right\|^2 + \|\mathbf{p}_\perp\|^2\right) \qquad (4) \\ & = \;\; \frac{4}{K}\left(\|\mathbf{r}(\mathbf{x}) - \mathbf{h}\|^2 + \|\mathbf{p}_\perp\|^2\right). \qquad (5) \end{aligned}$$

Here (4) comes from the fact that $\mathbf{p}_\perp$ is orthogonal to every $\mathbf{p}_m$; (5) is because $\{\mathbf{p}_m\}_{m=1}^M$ forms an orthonormal basis. □ □

We can take a closer look at the two terms in the right-hand-side of the bound (3). The first term describes the squared prediction error between $\mathbf{h}$ and $\mathbf{r}(\mathbf{x})$, two vectors represented under the coordinate system of the M-flat. The second term describes an encoding error for projecting $\mathbf{y}$ to the closest point on the M-flat. The training step of Linear Label Space Transformation aims at reducing the first term by learning from $\{(\mathbf{x}_n, \mathbf{h}_n)\}_{n=1}^N$.

The second term, on the other hand, does not depend on $\mathbf{x}$ and denotes a trade-off on the choice of M. In particular, the second term generally decreases when M increases, at the expense of more computational cost for learning the functions $\{r_m\}_{m=1}^M$. For instance, in PBR, if we take the origin as $\mathbf{o}$, the second term is upper bounded by $\frac{K-M}{K}$ (while the actual value depends on how sparse $\mathbf{y}$ is). When using the full Binary Relevance, there is no encoding error but many regressors are needed; when using PBR with $M \ll K$, we can use fewer regressors but the resulting Hamming loss may be large because of the large encoding error.

3.3 Principal Label Space Transformation

For a fixed value of M, the analysis of Algorithm 8 indicates that it is important to use an M-flat that makes the encoding error as small as possible. Next, we propose an approach that focuses on finding such an M-flat. In particular, the proposed *Principal Label Space Transformation* (PLST) approach is a special case of Algorithm 8 that seeks for a reference point $\mathbf{o} \in \mathbb{R}^K$ and an M by K matrix $\mathbf{P}$ by solving

$$\begin{aligned} \min_{\mathbf{o},\mathbf{P}} \quad & \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{o} - \mathbf{P}^T\mathbf{P}(\mathbf{y}_n - \mathbf{o})\|^2 \\ \text{such that} \quad & \mathbf{P}\mathbf{P}^T = \mathbf{I}. \end{aligned} \tag{6}$$

The objective function of (6) is the empirical average of the encoding error on the training set $\mathcal{S}$. Because PLST makes an optimal use of the budget on the $M \ll K$ basis functions, we can take advantage from the hypercube sparsity to reduce the computational cost in multi-label classification.

Similar to the traditional analysis of Principal Component Analysis (Hastie et al.,

2001), it can be proved that one optimal solution of (6) satisfies

$$\mathbf{o} \;=\; \frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n.$$

Then, the corresponding optimal $\mathbf{P}$ can be computed from the Singular Value Decomposition (SVD), as described below.

Consider a matrix $\mathbf{Z}$ with each column being $\mathbf{y}_n - \mathbf{o}$, a shifted version of the occupied vertices. Then, we perform SVD on the K by N matrix $\mathbf{Z}$ to obtain three matrices (Datta, 1995)

$$\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{7}$$

Here $\mathbf{U}$ is a K by K unitary matrix, $\mathbf{\Sigma}$ is a K by N diagonal matrix, and $\mathbf{V}$ is a N by N unitary matrix. Through SVD, each $\mathbf{y}_n - \mathbf{o}$ can be represented as a linear combination of the singular vectors $\mathbf{u}_m$ in the columns of $\mathbf{U}$. The vectors form a basis of a flat that passes through $\mathbf{o}$ as well as all the $\mathbf{y}_n$. The matrix $\mathbf{\Sigma}$ is a diagonal matrix that contains the singular value σ_m of each singular vector $\mathbf{u}_m$. We shall assume that the singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_K$.

Note that (7) can be rewritten as

$$\mathbf{U}^T\mathbf{Z} = \mathbf{\Sigma}\mathbf{V}^T$$

where the orthogonal basis $\mathbf{U}^T$ can be seen as a projection matrix of $\mathbf{Z}$ that maps each $\mathbf{y}_n - \mathbf{o}$ to a different coordinate system. Since the largest M singular values correspond to the principal directions for reconstructing $\mathbf{Z}$, we could discard the rest of the singular values and their associated basis vectors in $\mathbf{U}^T$ to obtain a smaller projection matrix $\mathbf{U}_M^T = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_M]^T$. The optimal $\mathbf{P}$ that solves (6) is indeed $\mathbf{U}_M^T$ (Hastie et al., 2001), which leads to the total empirical encoding error of $\sum_{m=M+1}^{K}\sigma_m^2$.

In summary, PLST solves an SVD problem to minimize an empirical version of the encoding error. Then, PLST calls for a good regression algorithm to reduce the squared error between $\mathbf{r}(\mathbf{x})$ and $\mathbf{h}$. According to Theorem 1, when both terms are small, the resulting Hamming loss would also be small.

Unlike PBR, for which $\mathbf{P}$ corresponds to the original axis, nor CS, for which $\mathbf{P}$ is formed randomly, the PLST projection matrix using the principal directions $\mathbf{u}_m$ captures the correlations in multi-label classification. Thus, PLST is able to exploit the

Algorithm 9 Principal Label Space Transformation

1. With a given parameter M, perform SVD on $\mathbf{Z}$ and obtain $\mathbf{U}_M^T = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_M]$.
2. Run Algorithm 8 using $\mathbf{o} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{y}_n$ and $\mathbf{P} = \mathbf{U}_M^T$.

hypercube sparsity to make an effective use of the $M \ll K$ basis functions while keeping the encoding error small.

4 Experiments

Next, we conduct experiments on five real-world data sets to compare the three algorithms within Linear Label Space Transformation: PBR, CS and our proposed PLST. The data sets are downloaded from Mulan (Tsoumakas et al., 2010b) and cover a variety of domains, sizes and characteristics, as shown in Table 2. We include data sets with a particularly large number of labels such as `delicious`, `corel5k` and `mediamill` to test the effectiveness of CS and PLST in reducing the dimension of the label space.

The *cardinality* column of Table 2 is defined as the average number of labels per example. The *distinct* column of Table 2 shows the number of distinct label sets, or using the hypercube view, the number of vertices occupied by examples. Dividing the value of distinct by 2^K in Table 2, we see that hypercube sparsity indeed exists in every data set.

On the other hand, the *nonzero* column of Table 2 shows the maximum number of non-zero entries in $\mathbf{y}_n$. Comparing the value of nonzero to K in Table 2, we see that most data sets come with a strong label-set sparsity except `yeast` and `emotions`.

In all experiments, we randomly partition each data set into 90% for training and 10% for testing. We record the mean and the standard error of the test Hamming loss over 20 different random partitions.

We test CS, PBR and PLST with Ridge Linear Regression (RLR; Hastie et al., 2001) and M5P Decision Tree (M5P; Wang and Witten, 1997) as the underlying regression algorithm. We implement Ridge Linear Regression with $\lambda = 0.01$ in MATLAB, and take the M5P Decision Tree from WEKA (Hall et al., 2009) with its default settings. For

Table 2: Data Set Statistics

data set	domain	N	K	cardinality	distinct	hypercube sparsity	nonzero
delicious	text	16105	983	19.02	15806	1.93×10^{-292}	25
corel5k	text	5000	374	3.52	3175	8.25×10^{-110}	5
mediamill	video	43507	101	4.38	6555	2.59×10^{-27}	18
yeast	biology	2417	14	4.24	198	1.21×10^{-2}	11
emotions	music	593	6	1.87	27	4.22×10^{-1}	3

CS, We follow the recommendation from Hsu et al. (2009) to use the Hadamard matrix as the projection matrix $\mathbf{P}$. Then, we take the best-performing reconstruction algorithm in their work, CoSaMP, as the decoding function D and set the sparsity parameter for the reconstruction algorithm to the *nonzero* column in Table 2. For PBR, we simply take the origin (the most label-set-sparse vertex of the hypercube) as the reference point $\mathbf{o}$. In other words, the discarded labels in PBR would be reconstructed with 0 (see Subsection 3.2). Other variants of CS and PBR would be explored in Subsection 4.4.

4.1 Comparison on Hamming Loss

Fig. 2 and Fig. 3 show the test Hamming loss of PBR, PLST and CS at different sizes of the reduced sub-tasks. In Fig. 2, the approaches are coupled with a linear regressor: RLR; in Fig. 3, the approaches are coupled with a non-linear regressor: M5P. First of all, we see that regardless of the type of the regressor used, PLST is always capable to reach reasonable performance while reducing the label space to a lower dimensional M-flat. In particular, PLST with $M \ll K$ regressors is capable of achieving the same or better Hamming loss than the full Binary Relevance (without dimension reduction) on all data sets.

When using RLR as the regressor, the Hamming loss curve of PLST is always below the curve of PBR across all M in all data sets, which demonstrates that PLST is the more effective choice in the family of Linear Label Space Transformation with round-based decoding (Algorithm 8). In addition, for data sets without a strong label-

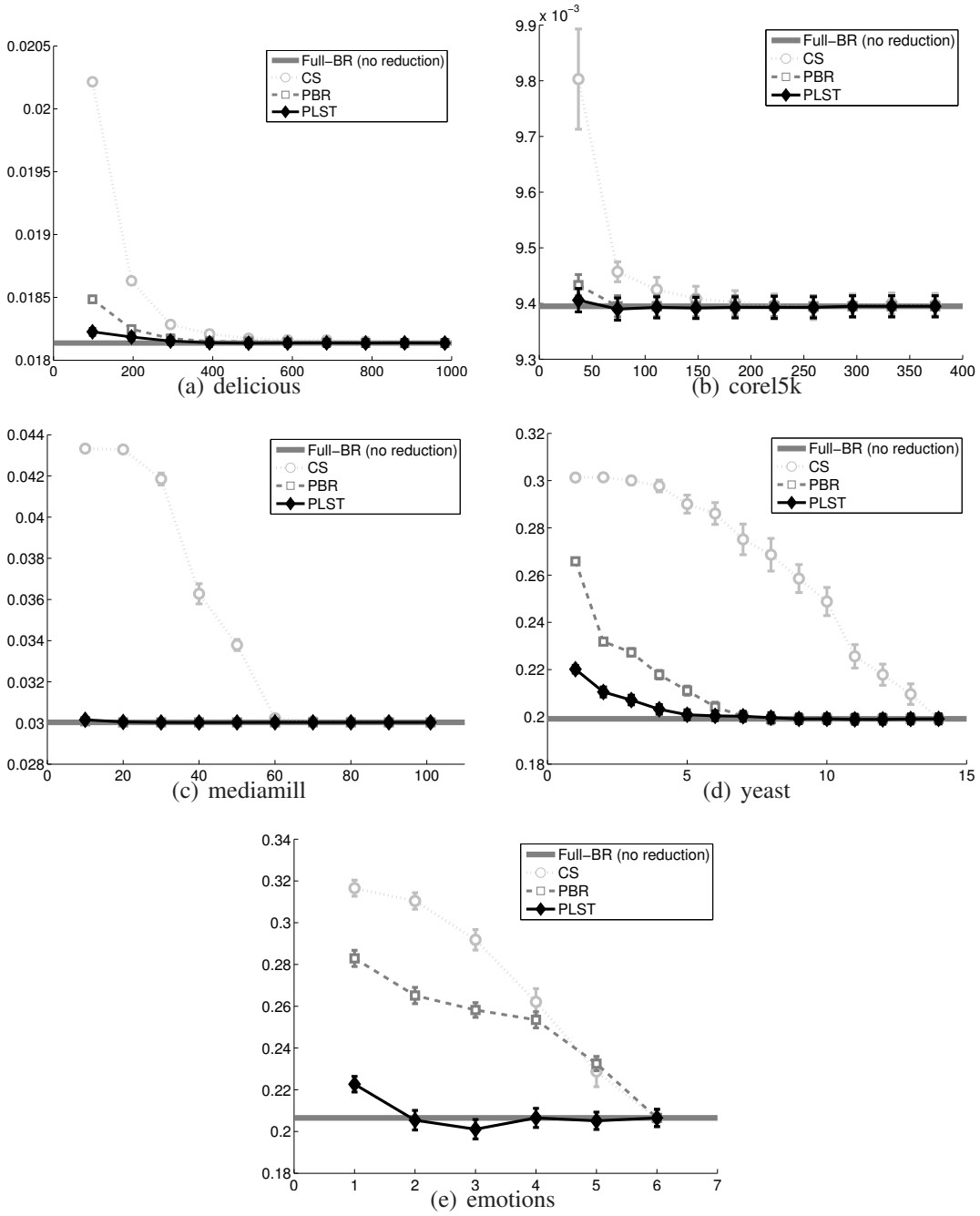


Figure 2: Test Hamming Loss of Linear Label Space Transformation Algorithms with RLR

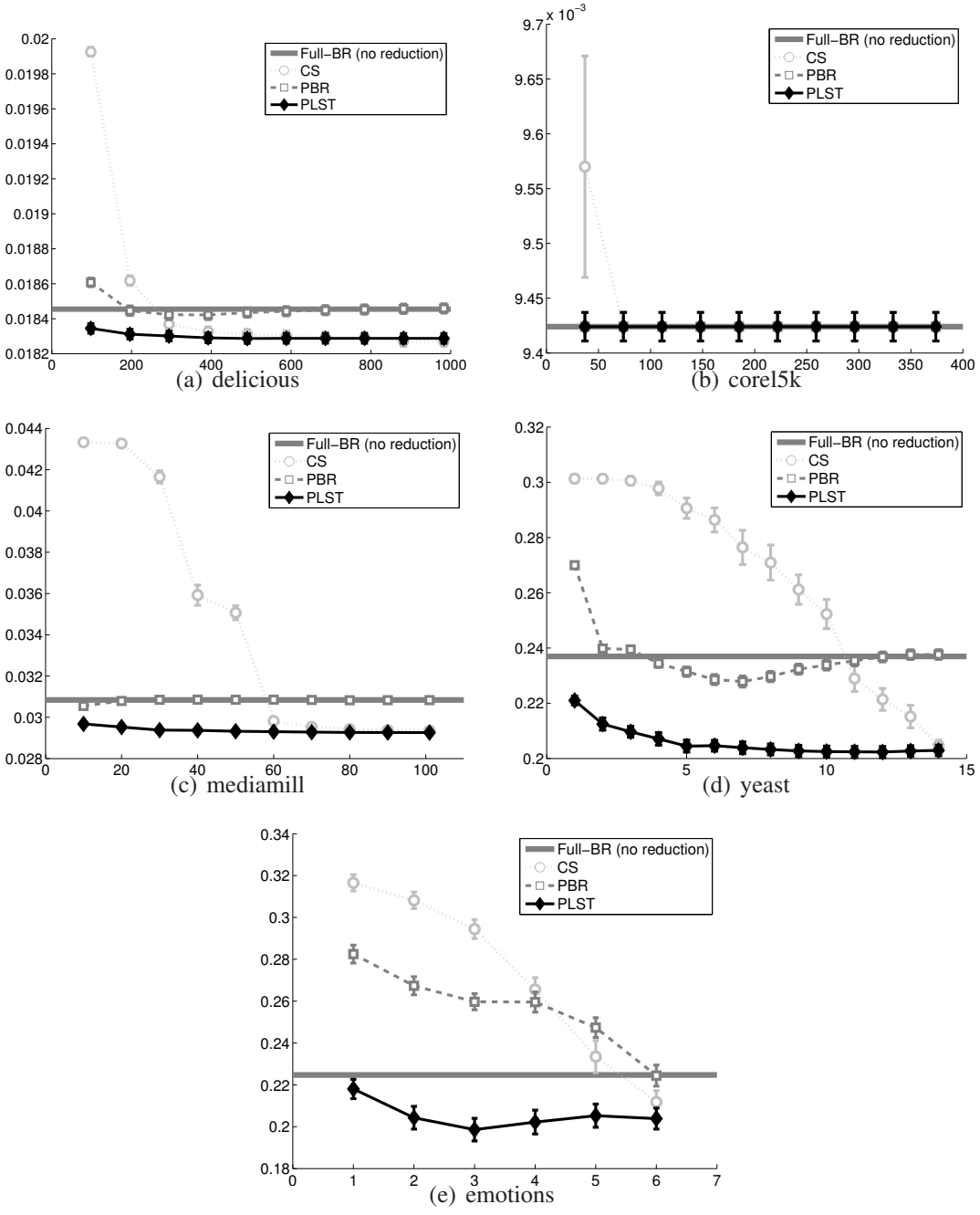


Figure 3: Test Hamming Loss of Linear Label Space Transformation Algorithms with M5P

set sparsity (yeast and emotions), as expected, PLST outperforms CS for both small and large M because CS cannot rely on label-set sparsity to make any compressions. On the other hand, for data sets with a strong label-set sparsity (delicious, corel5k and mediamill), the Hamming loss curve of PLST is below the curve of CS for small M, and comparable to the curve of CS for large M. That is, CS is able to decrease Hamming loss significantly after M is large enough by exploiting label-set sparsity. Nevertheless, PLST can always do even better by achieving the same Hamming loss with a much smaller M using hypercube sparsity. Thus, for data sets with or without label-set sparsity, PLST should be preferred over CS when RLR is taken as the regressor. One thing to notice is that on emotions, there is a decrease on Hamming loss for PLST with a small M, which demonstrates an additional potential advantage of focusing on the principal directions.

As shown on Fig. 3, the relationship between PLST, CS and PBR stays mostly unchanged when the non-linear regressor M5P is employed instead of RLR, especially when M is small. In addition, PLST is able to perform significantly better than a full BR in most of the data sets. The results justify that PLST as the leading approach for Linear Label Space Transformation—better than CS in particular—across the two different regressors tested.

Table 3 records the Hamming loss of PBR, PLST and CS at the optimal reduced sub-task size M^* and Table 4 records their respective training time. The results are obtained with RLR since it generally outperforms M5P when coupled with Linear Label Space Transformation algorithms. Here M^* is defined as the minimum dimension at which the Hamming loss difference between Binary Relevance and PLST is within their respective standard errors. In other words, this can be seen as the reduced dimension at which no performance loss is incurred. All the timing experiments were performed on the AMD Opteron Quad Core 2378 2.4 GHz Processor with 512 KB of cache. The programming environment was in MATLAB version 7.5.0.338 (R2007b). For most of the data sets with large amount of labels, PLST is able to drastically reduce the learning and inference time compared to the full Binary Relevance. This is less obvious in small data sets like yeast and emotions since their number of labels is already small before the transformation. In addition, PLST usually outperforms CS at M^* in terms of Hamming loss, training time in regression and training time in encoding/decoding.

Table 3: Test Hamming Loss of Full Binary Relevance (BR) versus PLST and CS at the Optimal Reduction Size of PLST

data set	M^*/K	BR (K)	PLST (M^*)	CS (M^*)
deli.	129/983 = 13%	0.01813 ± 0.00003	**0.01819** $\pm$ **0.00003**	0.01954 ± 0.00003
core.	16 /374 = 4%	0.00940 ± 0.00002	**0.00944** $\pm$ **0.00002**	0.01021 ± 0.00002
medi.	11 /101 = 11%	0.03003 ± 0.00006	**0.03015** $\pm$ **0.00006**	0.04332 ± 0.00007
yeas.	4 / 14 = 29%	0.19916 ± 0.00211	**0.20320** $\pm$ **0.00204**	0.29390 ± 0.00189
emot.	2 / 6 = 33%	0.20653 ± 0.00412	**0.20542** $\pm$ **0.00467**	0.31042 ± 0.00393

(the more accurate result between PLST and CS is marked in bold)

Table 4: Time of Full Binary Relevance (BR) versus PLST and CS at the Optimal Reduction Size of PLST

	BR (K)	PLST (M^*)		CS (M^*)	
data set	regression (sec)	regression (sec)	encode + decode (sec)	regression (sec)	encode + decode (sec)
deli.	4417.90	**577.38**	**154.38**	579.71	886.41
core.	560.11	**23.96**	7.76	24.13	**2.97**
medi.	105.96	11.55	**8.39**	**11.43**	57.14
yeas.	0.70	0.20	**0.02**	**0.18**	1.03
emot.	0.06	**0.02**	**0.00**	**0.02**	0.15

(the faster result between the corresponding columns of PLST/CS is marked in bold)

From Fig. 2, Fig. 3 and Table 3, it is clear that PLST is highly effective at reducing the number of sub-tasks solved for multi-label classification. Large data sets like delicious, corel5k and mediamill can be reduced to only 13%, 4%, and 11% of their original computational effort respectively with no sacrifice in performance. Note that we can further reduce the computational effort by tolerating a slight increase in Hamming loss, as can be seen in Fig. 2 and Fig. 3. These results demonstrate that PLST can take advantage of the hypercube sparsity to efficiently solve multi-label classification problems.

4.2 Comparison on Other Performance Measures

To further understand the benefits of PLST, we conduct more comparisons on three other popular performance measures: the average per-example ranking loss, the macro-averaged (per-label-averaged) area-under-the-ROC-curve (AUC), and the macro-averaged F1-score (Tsoumakas et al., 2010a). Although PLST is not particularly designed with respect to those measures, we shall demonstrate that PLST remains to be the most effective choice over CS and PBR in terms of the ranking loss and AUC. We only list the results with RLR here as it generally outperforms M5P, while similar findings have also been observed across most of the data sets when using M5P.

Fig. 4 shows the comparison of ranking loss using RLR. For each example, the ranking loss takes the soft prediction $\tilde{\mathbf{y}}$ as an order of the labels, and compares the predicted order to the desired order $\mathbf{y}$:

$$\text{ranking loss}(\tilde{\mathbf{y}}, \mathbf{y}) = \text{average}_{\mathbf{y}[k]<\mathbf{y}[\ell]} \left(\left[\!\left[\tilde{\mathbf{y}}[k] > \tilde{\mathbf{y}}[\ell] \right]\!\right] + \frac{1}{2} \left[\!\left[\tilde{\mathbf{y}}[k] = \tilde{\mathbf{y}}[\ell] \right]\!\right] \right)$$

CS cannot perform well on the ranking loss because it only outputs hard predictions that contain 0 or 1, which could introduce more loss on ties $[\![\tilde{\mathbf{y}}[k] = \tilde{\mathbf{y}}[\ell]]\!]$. Similarly, PBR cannot perform well on the ranking loss because many of its predictions $\tilde{\mathbf{y}}$ contains a constant 0 introduced by the reference point $\mathbf{o}$, which also leads to loss on ties. On the other hand, PLST is highly effective in capturing the ranking preferences with the principal directions in the label space. On larger data sets like `delicious`, PLST is able to achieve decent ranking loss using only 10% of the original dimensions. The results justify the usefulnes of PLST on the ranking loss.

The promising ranking performance of PLST makes it interesting to compare PLST with the label ranking approach, which is shown in Fig. 5. In particular, we take the state-of-the-art calibrated label ranking (Fürnkranz et al., 2008) approach and couple it with RLR for a fair comparison. Because label ranking takes pairwise comparison of the labels, we can only afford to run the experiments on `emotions`, `yeast` and `mediamill`. In terms of the ranking loss (the left-hand-side), PLST can be much worse than calibrated label ranking, which is expected because PLST does not include any pairwise information in its design for dimension reduction. In terms of the Hamming loss (the right-hand-side), however, PLST is quite competitive with calibrated label ranking. Note that calibrated label ranking pays for the pairwise comparisons and

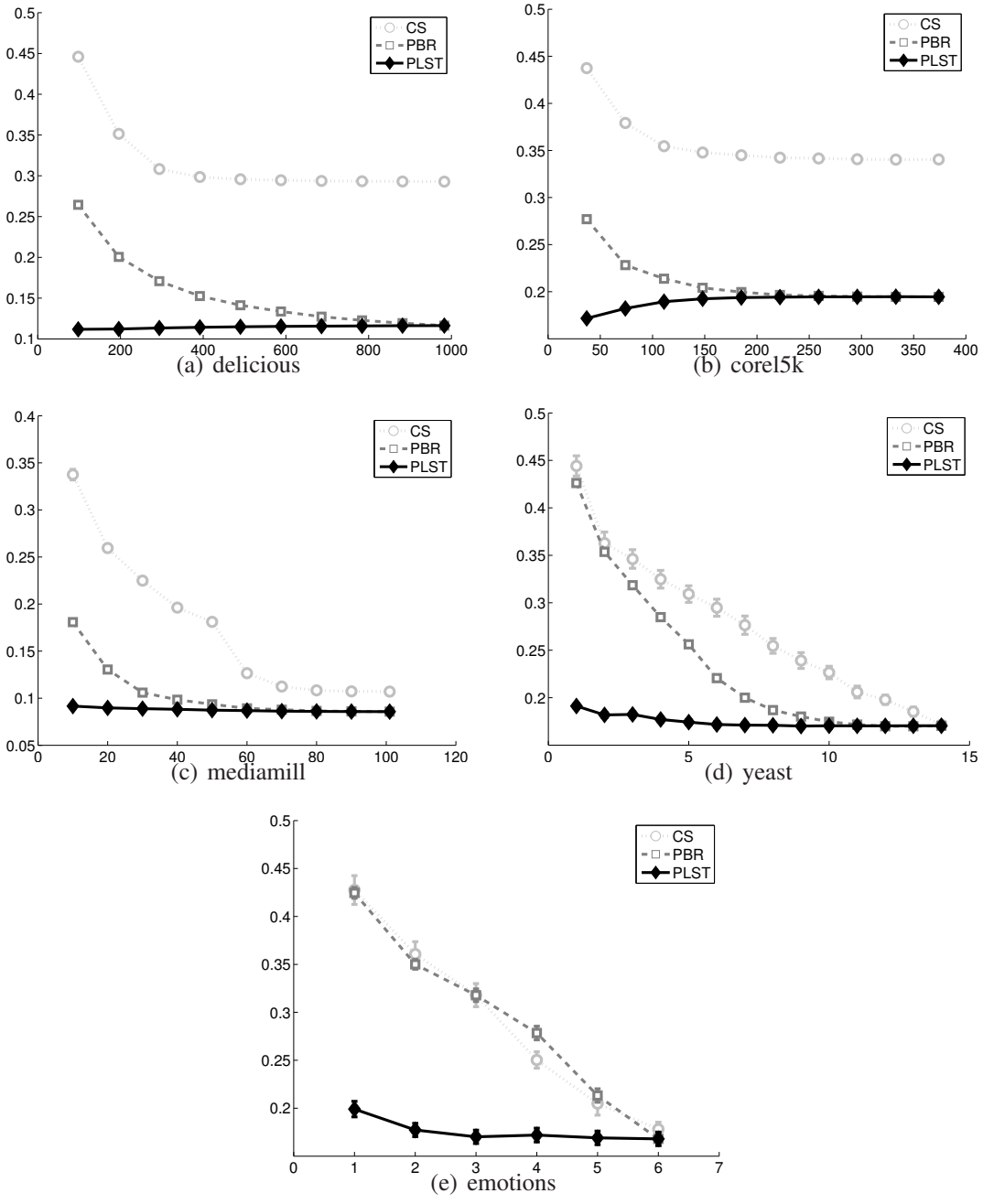


Figure 4: Test Ranking Loss of Linear Label Space Transformation Algorithms Using RLR

can hardly be scaled up for data sets with lots of labels, while PLST is much more efficient. The results reveal an important future research direction: designing a dimension reduction approach that is more efficient than calibrated label ranking while maintaining the same level of ranking performance.

Fig. 6 shows the comparison of the macro AUC when using RLR. For each label $k = 1, 2, \cdots, K$, AUC takes the soft predictions $\tilde{\mathbf{y}}[k]$ of all the test examples to construct the ROC curve, and computes the area under the curve. The ROC curve reveal the trade-off between the precision and the recall of the soft predictions, and larger AUC indicates better performance. One simple view of the macro AUC is that it measures the ranking performance *per label*, while the ranking loss discussed above measures the ranking performance *per example*. Similar to the ranking loss, CS cannot perform well on AUC because it is only able to output a hard prediction; PBR also cannot perform well because its constant predictions in some labels. Thus, PLST remains to be the most effective choice for achieving decent AUC while performing linear dimension reduction.

Fig. 7 compares the macro F1 score using RLR. Instead of exploring the full trade-off of the precision and the recall with macro AUC, for each label $k = 1, 2, \cdots, K$, by the hard predictions $\hat{\mathbf{y}}[k]$. One interesting finding is that PLST is not always better than PBR or CS when using the macro F1 score. That is, while PLST achieves a decent trade-off between precision and recall using the soft predictions, its hard predictions (using rounding at 0.5) leave some room for improvements. The results echo earlier findings by Fan and Lin (2007) on improving the F1 score by tuning the rounding thresholds.

To understand the cause of the different performance in the F1 score, we show the macro precision and the macro recall on `delicious` in Fig. 8. We see that CS is more aggressive in finding the present labels, which leads to better recall when M is around 300 and explains its better F1 score. On the other hand, the precision of CS is not satisfactory. PLST is less aggressive, which results in a much better precision but worse recall when compared with CS. PBR is even less agressive and thus achieves the worst recall of all three algorithms.

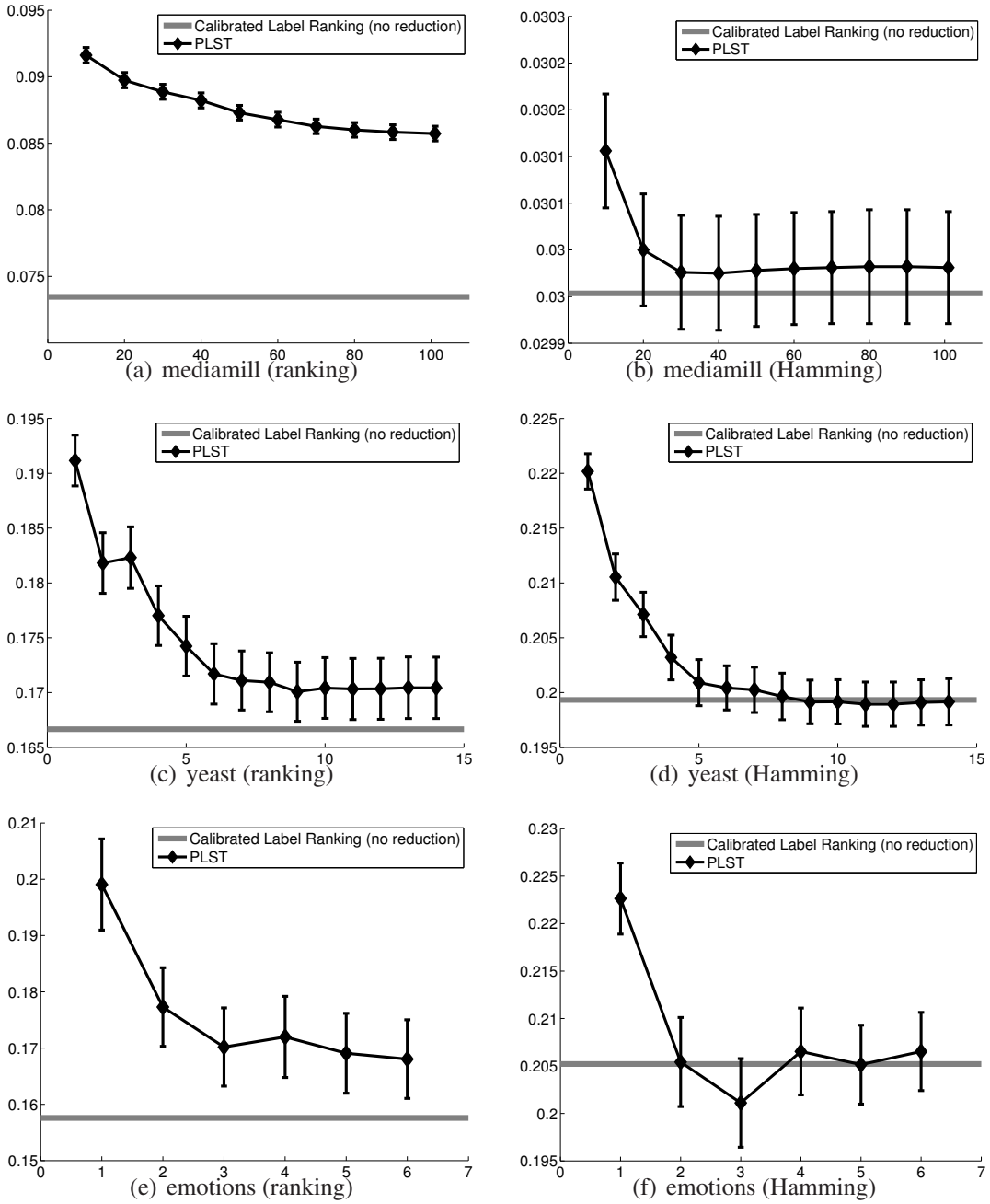
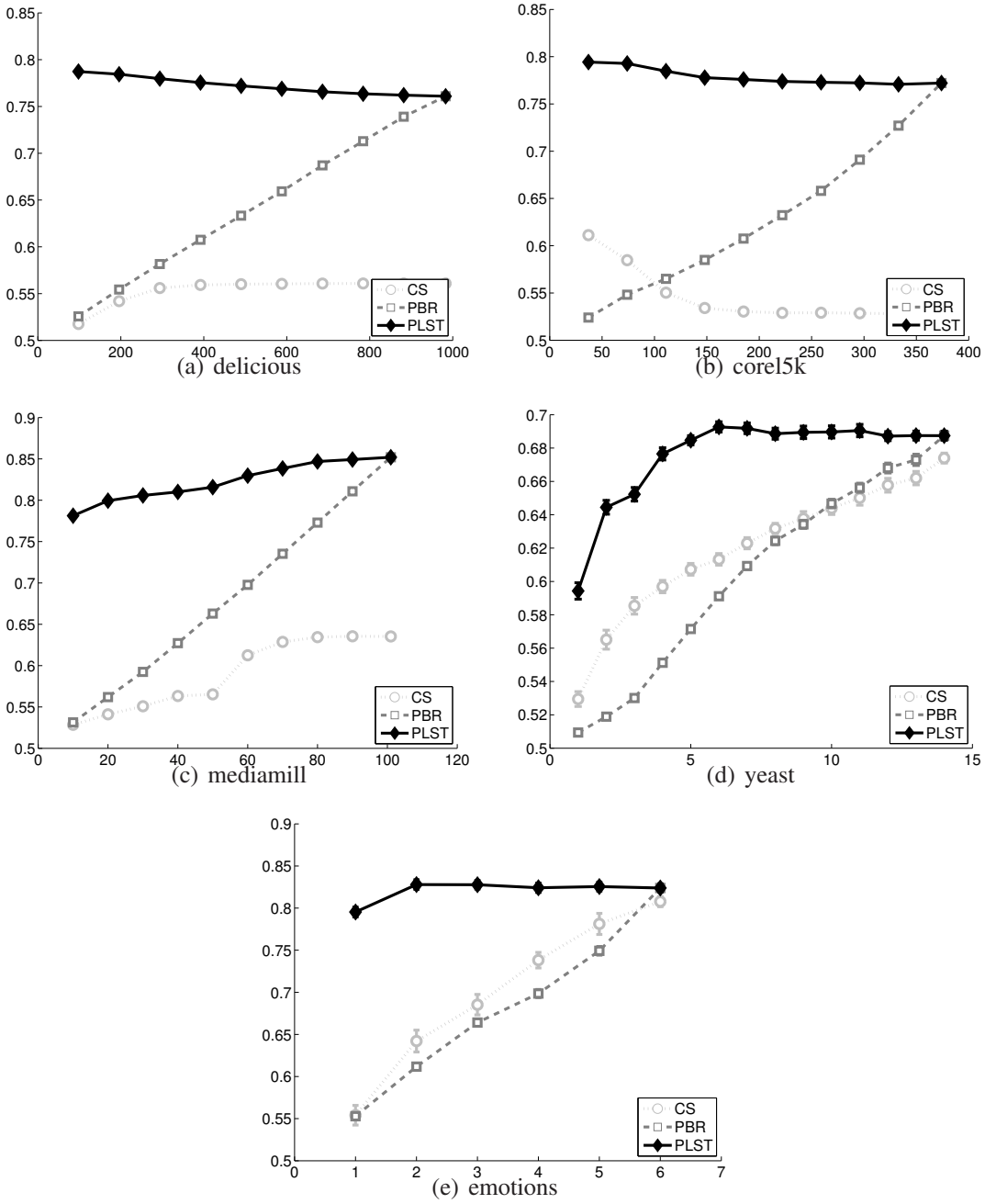


Figure 5: Comparison between Linear Label Space Transformation Algorithms and Calibrated Label Ranking Using RLR



(a) delicious (b) corel5k

(c) mediamill (d) yeast

(e) emotions

Figure 6: Test AUC of Linear Label Space Transformation Algorithms Using RLR

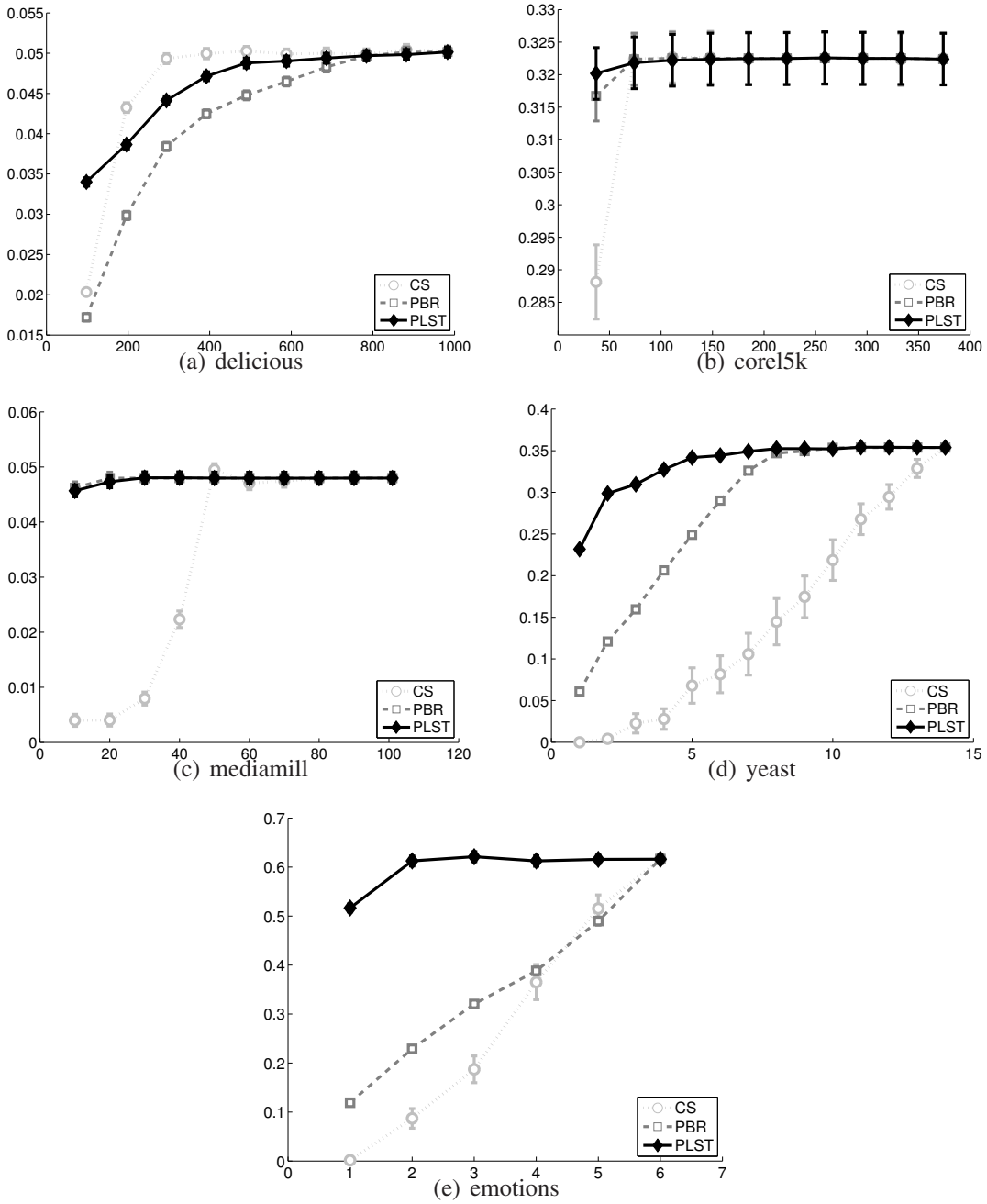


Figure 7: Test F1 Score of Linear Label Space Transformation Algorithms Using RLR

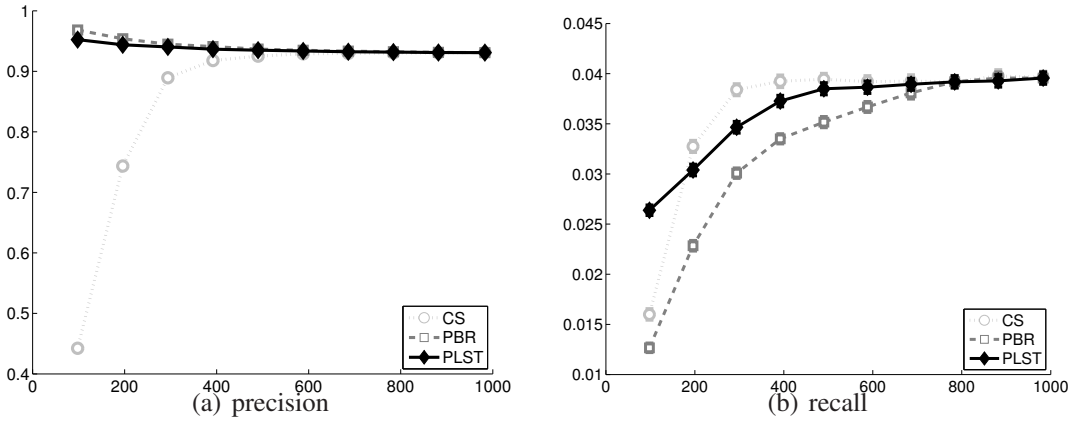


Figure 8: Test Precision and Recall of Linear Label Space Transformation Algorithms Using RLR

4.3 Comparison Using the Ideal Learner

Next, we analyze the reasons behind the success of PLST. We do so by including an ideal (non-realistic) learner as the underlying regression algorithm. The ideal learner is allowed to look at all the test labels and can always achieve $\|\mathbf{r}(\mathbf{x}) - \mathbf{h}\|^2 = 0$ for every $(\mathbf{x}, \mathbf{y})$. Fig. 9 shows the results of using the ideal learner. When using the ideal learner, the Hamming loss curve for PLST is still always below that of PBR's. Thus, using the principal directions, as expected, is a superior encoding scheme than the original axis. On the other hand, CS outperforms PLST when M is large enough when using an ideal learner on `delicious`, `corel5k` and `mediamill`. That is, in the ideal case, the decoder of CS is capable of working accurately. However, the realistic regressors are of course not ideal, which explains the inferior performance of CS over PLST. In particular, because CS projects the vertex to a random direction, the resulting regression tasks can be difficult to learn for realistic regressors. Thus, CS achieves lower Hamming loss than PLST only "ideally." Note that for data sets that do not come with strong sparsity, such as `yeast` and `emotions`, CS performs similarly to or worse than PLST even with the ideal learner.

Note that when taking the ideal learner in Algorithm 8, the only factor that accounts for the Hamming loss is the encoding error—the second term in the right-hand-side of (3). Fig. 10 shows the average encoding error evaluated on the test set. When comparing Fig. 10 with Fig. 9, we see that the encoding error is a loose but indicative upper bound of the Hamming loss. In particular, the difference of performance between PLST

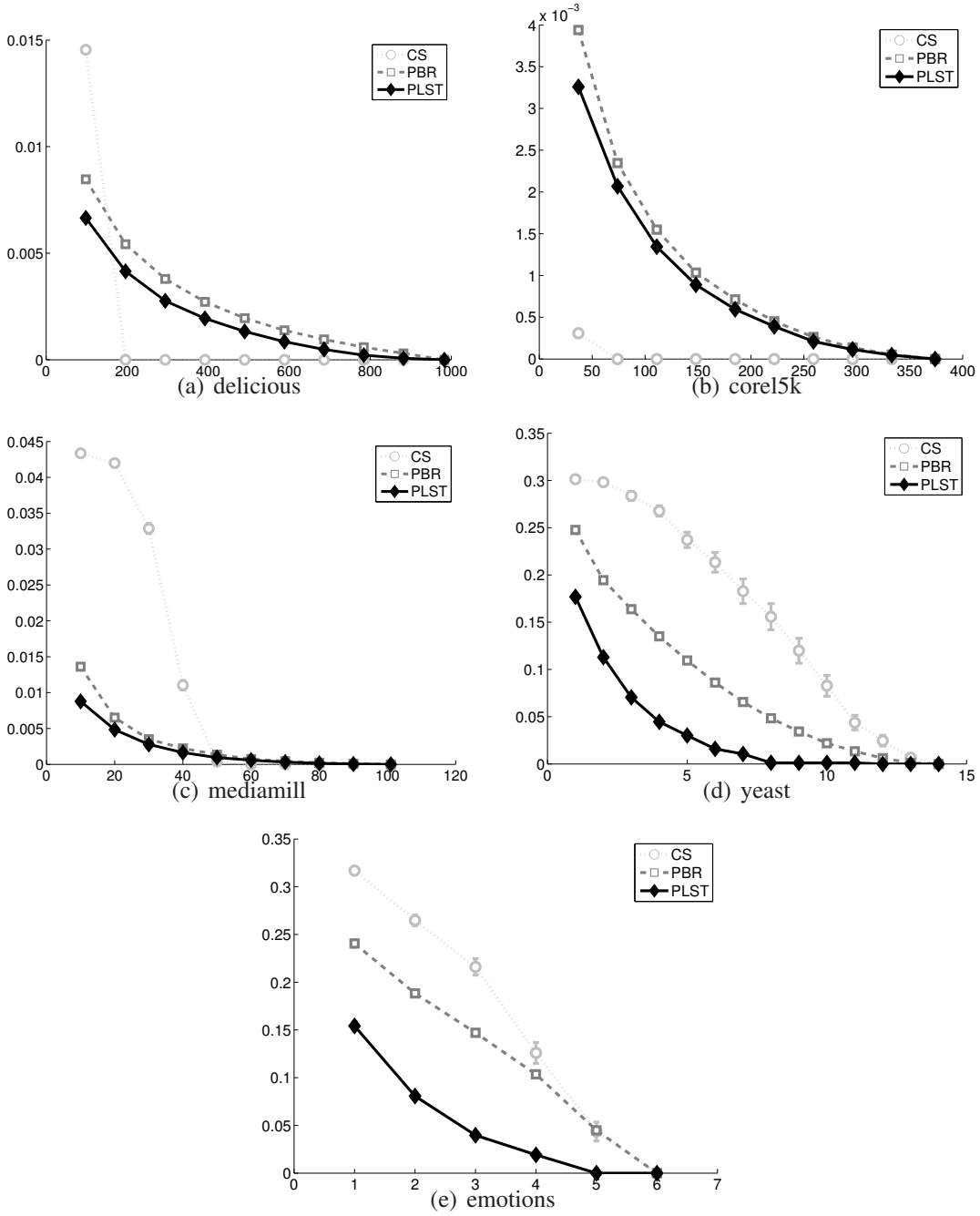


Figure 9: Test Hamming Loss of Linear Label Space Transformation Algorithms Using the Ideal Learner

and PBR can be well-explained with the encoding error. The results further verify that the promising performance of PLST over PBR can be attributed to its effective use of the M basis functions for keeping the encoding error small.

4.4 Importance of Reference Points

One particular advantage of PLST appears to be using the average $\mathbf{y}_n$ as the reference point $\mathbf{o}$, which readily captures an estimate of the ratio of presence per label. PBR uses the origin as the reference point and hence the advantge on changing the reference point has not been explored. The flexibility of CS for choosing the reference point (to make the signal vectors more sparse) has not been explored, either.

To understand whether the choice of reference point is an important factor behind the success of PLST, we conduct some additional experiments using three more algorithms:

1. PLST-Origin: a crippled PLST that takes the origin $\mathbf{0}$ as $\mathbf{o}$ and extracts a different set of principal directions that passes the origin. The usual PLST is then renamed PLST-Mean for clarity.

2. PBR-Mean: an improved PBR that takes the average $\mathbf{y}_n$ as $\mathbf{o}$. The usual PBR is renamed PBR-Origin.

3. CS-BestVertex: an improved CS that transforms $\mathbf{y}[k]$ to

$$\mathbf{y}[k] \oplus \text{majority}\{\mathbf{y}_n[k] \colon n = 1, 2, \cdots, N\}$$

   before (and after) training. In other words, the improved CS computes the best vertex to strengthen label-set sparsity prior to training. The usual CS is renamed CS-Origin.

Fig. 11 shows the Hamming loss comparison between the usual PLST, PBR, CS and their variants on `delicious` and `yeast` using RLR as the regressor. For `delicious`, in which there is strong label-set sparsity, we see that the change of reference point does not lead to much difference for each pair of algorithms. In particular, the origin is readily the best vertex for CS and quite close to the mean for PBR and PLST. On the other

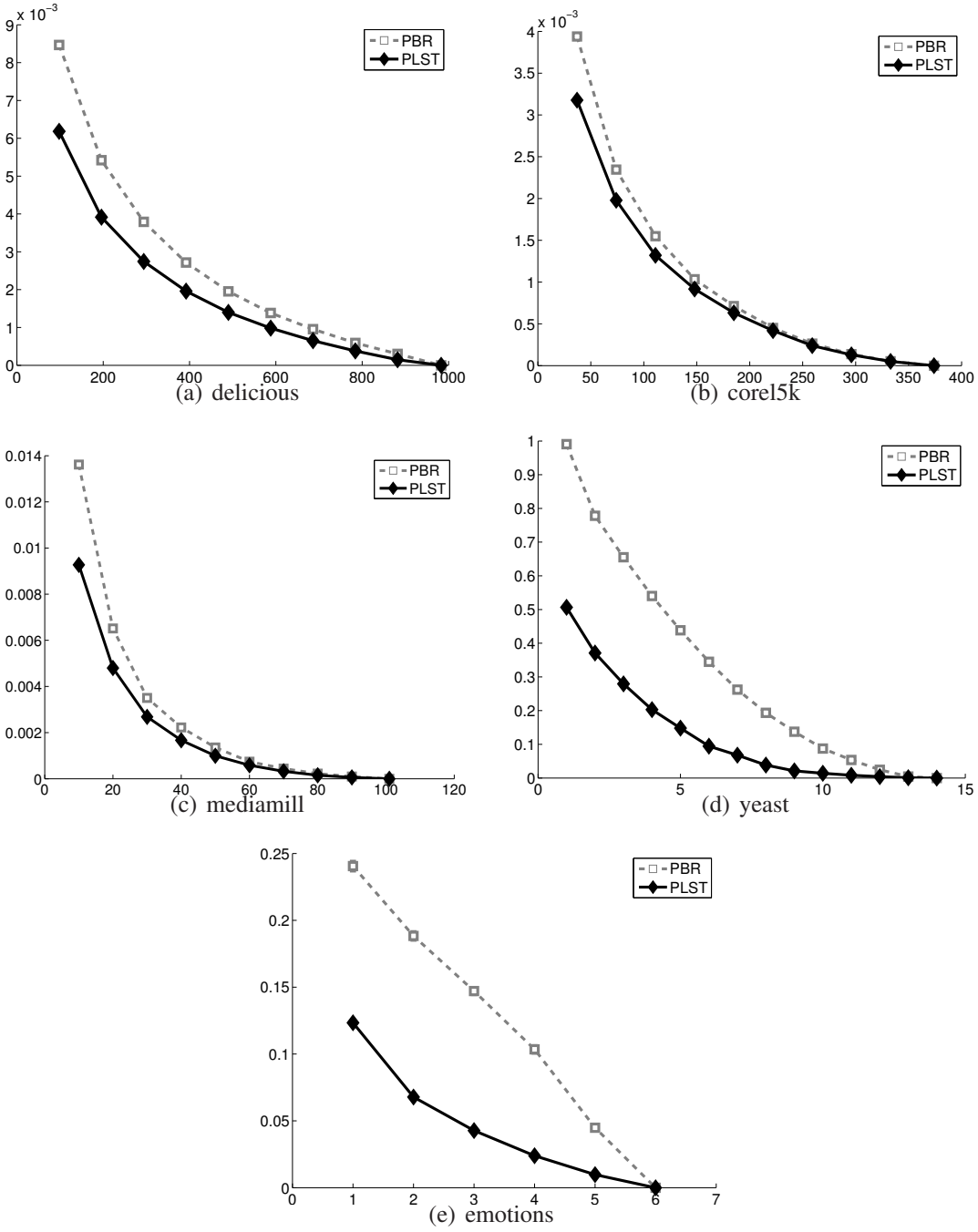


Figure 10: Test Encoding Error of PBR and PLST

hand, for `yeast`, in which there is no strong label-set sparsity, the change of reference point not only is important when M is small in PBR and PLST but also leads to significant improvements in CS.

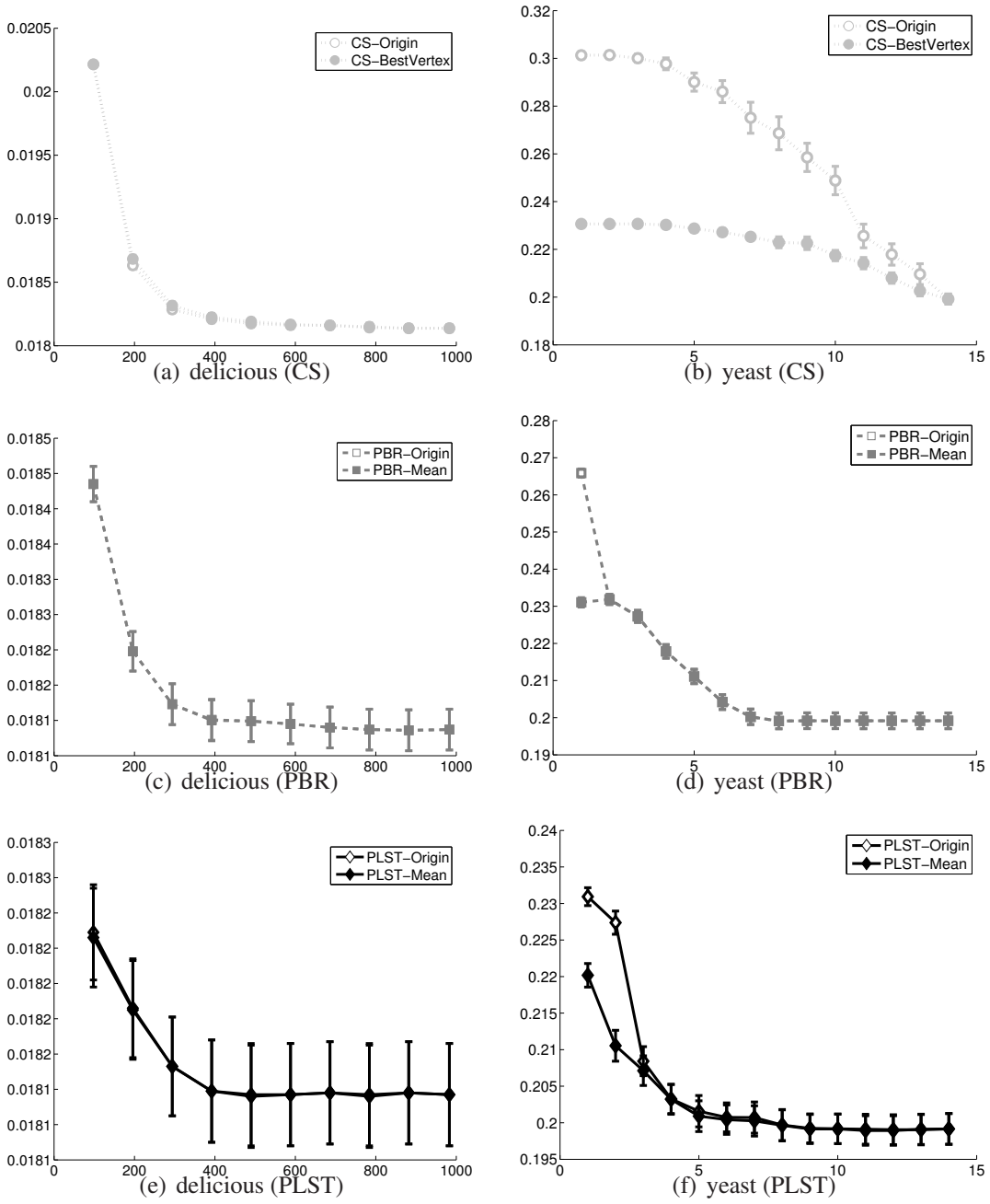
Fig. 12 compares PLST-Mean and PLST-Origin with PBR-Mean and CS-BestVertex on all data sets. Even with the improved PBR-Mean and CS-BestVertex, the findings in Fig. 12 remain the same. PLST with or without the mean shift is significantly better than the other two algorithms, with PLST-Mean being the better choice. The results suggest that the principal directions (by SVD) rather than the mean shifting are the key factors behind the success of PLST.

5 Conclusion

We presented the hypercube view for problem transformation approaches to multi-label classification. The view offers geometric interpretations to many existing algorithms including Binary Relevance, Label Power-set, Label Ranking, Compressive Sensing (CS), Topic Modeling and Kernel Dependency Estimation. Inspired by this view, we introduced the notion of hypercube sparsity and took it into account by Principal Linear Space Transformation (PLST). We derived the theoretical guarantee of PLST and conducted experiments to compare PLST with Binary Relevance and CS. Experimental results verified that PLST is successful in reducing the computational effort for multi-label classification, especially for data sets with large numbers of labels. Most importantly, when compared with CS, PLST not only enjoys a faster decoding scheme, but also reduces the multi-label classification problem to simpler and fewer regression sub-tasks. The advantages and the empirical superiority suggest that PLST should be a preferred choice over CS in practice.

As demonstrated through experiments, PLST was able to achieve similar performance with substantially less dimensions compared to the original label-space. An immediate future work is to conclude how to automatically and efficiently determine a reasonable parameter M for PLST.

We discussed in Section 1 that PLST can be viewed as a special case of the kernel dependency estimation (KDE) algorithm (Weston et al., 2002). To the best of our knowledge, we are the first to focus on KDE's linear form for multi-label classification,



(a) delicious (CS) (b) yeast (CS)

(c) delicious (PBR) (d) yeast (PBR)

(e) delicious (PLST) (f) yeast (PLST)

Figure 11: Change of Reference Points for Linear Label Space Transformation Algorithms

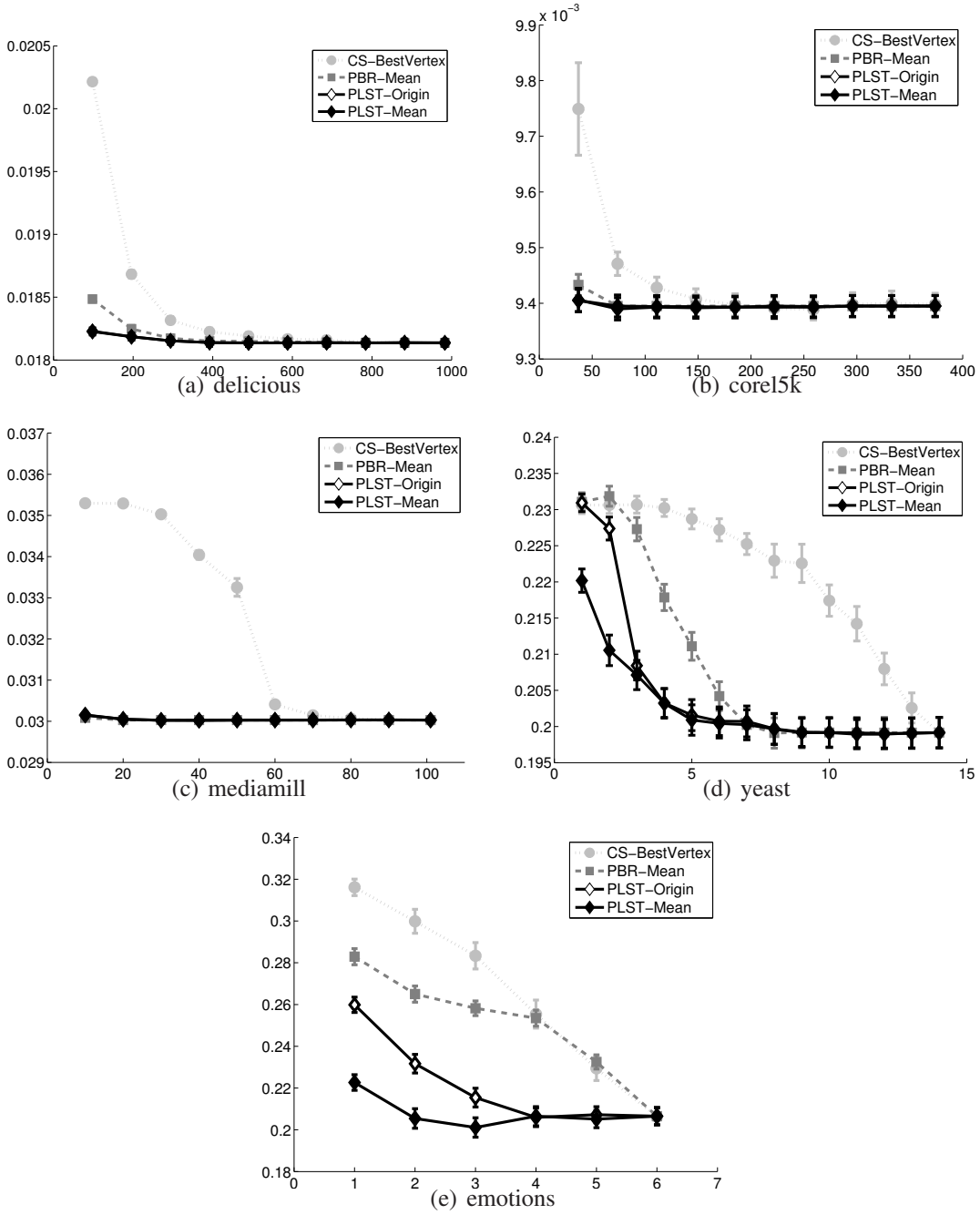


Figure 12: Test Hamming Loss of Improved PBR and CS versus PLST Variants with RLR

PLST, which readily leads to promising performance. A plausible future work is to carefully evaluate the usefulness of KDE's non-linear form for multi-label classification.

Acknowledgments

A preliminary version of this paper appeared in the Second International Workshop of Learning from Multi-label Data. We thank the reviewers of the workshop as well as reviewers for all versions of this paper for their many useful suggestions. We also thank Krzysztof Dembczynski, Weiwei Cheng, Eyke Hüllermeier and Willem Waegeman for valuable discussions. This research has been supported by the National Science Council of Taiwan via NSC 98-2221-E-002-192 and 100-2628-E-002-010.

References

Ando, R. and Zhang, T. (2005). A framework for learning predictative structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

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

Blei, D. M., Ng, A., and Jordan, M. (2003). Latent dirichlet allocation. *JMLR*, 3:993–1022.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771.

Clare, A. and King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53.

Csiszár, I. (1995). Maxent, mathematics, and information theory. In *Proceedings of the Fifteenth International Workshop on Maximum Entropy and Bayesian Methods*, pages 35–50.

Datta, B. N. (1995). *Numerical Linear Algebra and Applications*. Brooks/Cole Publishing.

Dembczynski, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2010a). On label dependencies in multi-label classification. In *Proceedings of the 2nd Workshop on Learning from Multi-label Data*.

Dembczynski, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2010b). Regret analysis for performance metrics in multi-label classication: The case of hamming and subset zero-one loss. In *European Conference on Machine Learning*.

Elisseeff, A. and Weston, J. (2002). A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14*, pages 681–688.

Fan, R.-E. and Lin, C.-J. (2007). A study on threshold selection for multi-label classification. Technical report, National Taiwan University.

Fürnkranz, J., Hüllermeier, E., Lozamencía, E., and Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. In *SIGKDD Explorations*, volume 11.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag.

Hsu, D., Kakade, S. M., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *Advances in Neural Information Processing Systems 22*, pages 772–780.

Ji, S., Tang, L., Yu, S., and Ye, J. (2010). A shared-subspace learning framework for multi-label classification. *ACM Transaction on Knowledge Discovery from Data*, 4(2):1–29.

Law, E., Settles, B., and Mitchell, T. (2010). Learning to tag using noisy labels. In *European Conference on Machine Learning*.

Read, J., Pfahringer, B., and Holmes, G. (2008). Multi-label classification using ensembles of pruned sets. In *ICDM'08*, pages 995–1000.

Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *In Proceedings of the 15th International Conference on Machine Learning*, pages 515–521. Morgan Kaufmann.

Schapire, R. E. and Singer, Y. (2000). Boostexter: a boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Snoek, C. G. M., Worring, M., van Gemert, J. C., Geusebroek, J. M., and Smeulders, A. W. M. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 421–430.

Trohidis, K., Tsoumakas, G., Kalliris, G., and Vlahavas, I. (2008). Multilabel classification of music into emotions. In *Proceedings of the 9th International Conference on Music Information Retrieval*, pages 325–330.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010a). *Mining Multi-label Data*. Data Mining and Knowledge Discovery Handbook, O. Maimon, L. Rokach (Ed.). Springer, 2nd edition.

Tsoumakas, G., Vilcek, J., and Xioufis, E. S. (2010b). Mulan: A java library for multi-label learning. `http://mulan.sourceforge.net/datasets.html`.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214.

Wang, Y. and Witten, I. H. (1997). Induction of model trees for predicting continuous classes. In *Proceedings of ECML '97*, pages 128–137. Springer.

Weston, J., Chapelle, O., Elisseeff, A., Schoelkopf, B., and Vapnik, V. (2002). Kernel dependency estimation. In *Advances in Neural Information Processing Systems 15*, pages 873–880.

Zhang, M. and Zhou, Z. (2007). ML-kNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048.