國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

多標籤分類問題中考量特徵的標籤空間降維法

Feature-aware Label Space Dimension Reduction for Multi-label Classification Problem

陳耀男

Yao-Nan Chen

指導教授：林軒田 博士

Advisor: Hsuan-Tien Lin, Ph.D.

中華民國 101 年 6 月

June, 2012

致謝

這兩年來， 很感謝那些協助完成碩士階段學習以及論文的人， 托大家的福， 此碩士畢業論文逐漸成長茁壯並迎來即將收成的時刻。

首先誠摯的感謝指導教授林軒田老師的指導， 讓我能夠一窺機器學習領域的奧秘。 在學習過程中老師給予了許多的學習機會和指導，並給予正確的研究方向以及研究態度。 除此之外，也能跟老師同披戰袍征戰大小比賽。 最後，也很感謝他能夠撥空修改論文並在寫作上給予許多寶貴的意見。

此外，感謝實驗室的成員， 能夠一起討論研究論文 並在研究和論文上提供很多意見以及協助。 也很感謝大家總能為實驗室帶來熱絡的氣氛。 同時也感謝大學時的同學以及朋友能夠適時地給予協助。

最後十分感謝父母以及家人在我成長的路上默默的付出以及幫助，有家人的支持才能使我在學習的道路上勇往直前。

回首過去，本人對在這追求知識並連接遙遠的過去與未來的道路上給予過協助、擔心或照顧的人獻上十二分的謝意。

中文摘要

標籤空間降維法（Label Space Dimension Reduction）在多標籤分類問題（Multi-lable Classification Problem）中是一個有效且有效率的方法。現存的標籤空間降維法（例如：壓縮感知（Compressive Sensing）與主要標籤空間變換（Principal Label Space Transformation））只利用資料中標籤部份的資訊。在本論文中，我們提出一個能夠同時考量特徵與標籤之資訊的標籤空間降維法。此稱為條件式主要標籤空間變換（Conditional Principal Label Space Transformation）之演算法的設計目的是最小化廣泛使用的漢明虧損的上界。此方法的最小化步驟能夠透過有效率的執行奇異值分解達成。此外，此方法能夠擴展至核函數方法，核函數方法允許使用更精密複雜的特徵組合去幫助標籤空間降維。實驗結果顯示此方法在標籤分類問題中確實比其他現存方法更有效。

關鍵詞：機器學習、多標籤分類、標籤空間、降維、核函數方法

Abstract

Label space dimension reduction (LSDR) is an efficient and effective paradigm for multi-label classification with many classes. Existing approaches to LSDR, such as compressive sensing and principal label space transformation, exploit only the label part of the dataset, but not the feature part. In this thesis, we propose a novel approach to LSDR that considers both the label and the feature parts. The approach, called conditional principal label space transformation, is based on minimizing an upper bound of the popular Hamming loss. The minimization step of the approach can be carried out efficiently by a simple use of singular value decomposition. In addition, the approach can be extended to a kernelized version that allows the use of sophisticated feature combinations to assist LSDR. The experimental results verify that the proposed approach is more effective than existing ones to LSDR across many real-world datasets. **Keywords**: Machine Learning, Multi-label Classification, Label Space, Dimension Reduction, Kernel Method.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The multi-label classification problem is an extension of the traditional multiclass classification problem. In contrast to the multiclass problem, which associates only a single label to each instance, the multi-label classification problem allows multiple labels for each instance. General solutions to this problem meet the demands of many real-world applications for classifying instances into multiple concepts, including categorization of text [1], scene [2], genes [3], music [4] and so on. Given the wide range of such applications, the multi-label classification problem has been attracting much attention of researchers in machine learning [5, 6].

Label space dimension reduction (LSDR) is a new paradigm in multi-label classification [7, 8]. By viewing the set of multiple labels as a high-dimensional vector in some label space, LSDR approaches use certain assumed or observed properties of the vectors to "compress" them. The compression step transforms the original multi-label classification problem (with many labels) to a small number of learning tasks. If the compression step, de-compression step, and learning steps can be executed efficiently and effectively, LSDR approaches can be useful for multi-label classification because of the appropriate use of joint information within the labels [8].

For instance, compressive sensing [CS; 7] is a representative LSDR approach. Under the assumption that only a few labels are associated with each example, CS transforms the multi-label classification problem into a small number of regression tasks. Another representative LSDR approach is the principal label space transformation [PLST; 8]. PLST takes advantage of the key linear correlations between labels and uses them to build a small number of regression tasks.

LSDR approaches are homologous to the well-studied feature space dimension reduction (FSDR) approaches in machine learning. LSDR and FSDR share similar advantages: they save computational power and storage without much loss of prediction accuracy as well as improve learning performance by removing irrelevant, redundant, or noisy information [9]. There are two types of FSDR approaches: unsupervised and supervised. Unsupervised FSDR considers only feature information during reduction, while supervised FSDR considers the additional label information. A typical instance of unsupervised FSDR is principal component analysis [PCA; 10]. PCA transforms the features into a small number of uncorrelated variables. On the other hand, the representative supervised FSDR approaches include supervised principal component analysis [11] and linear discriminant analysis [12]. In particular, for multi-label classification, a leading supervised FSDR approach is canonical correlation analysis [CCA; 13, 14] which is based on linear projections in both the feature space and the label space. In general, well-tuned supervised FSDR approaches can perform better than unsupervised ones because of the additional label information.

It has been argued that PLST can be viewed as the counterpart of PCA in the label space [8] and is feature-unaware. That is, it considers only the label information during reduction. Another LSDR approach, CS, is also a feature-unaware LSDR approach. Motivated by the superiority of supervised FSDR over unsupervised approaches, we are interested in studying feature-aware LSDR: LSDR that considers feature information.

In this thesis, we propose a novel feature-aware LSDR approach, conditional principal label space transformation (CPLST). CPLST combines the concepts of PLST (LSDR) and CCA (supervised FSDR). Notably, CPLST improves PLST through the addition of feature information. We derive CPLST by minimizing an upper bound of the popular Hamming loss and show that CPLST can be accomplished by a simple use of singular value decomposition (SVD). Moreover, CPLST can be flexibly extended by the kernel trick with suitable regularization, thereby allowing the use of sophisticated feature information to assist LSDR. The experimental results on real-world datasets confirm that CPLST can reduce the number of learning tasks without loss of prediction performance. In particular, CPLST is usually significantly better than PLST and other related LSDR approaches.

The rest of this thesis is organized as follows. In Chapter 2, we define the multi-label classification problem and review related works. Then, in Chapter 3, we derive the proposed CPLST approach. Finally, we present the experimental results in Chapter 4 and conclude our study in Chapter 5.

Chapter 2

Label Space Dimension Reduction

2.1 Multi-Label Classification Problem

The multi-label classification problem aims at finding a classifier from the input vector $\mathbf{x}$ to a label set $\mathcal{Y}$, where $\mathbf{x} \in \mathbb{R}^d$, $\mathcal{Y} \subseteq \{1, 2, \ldots, K\}$ and K is the number of classes. The label set $\mathcal{Y}$ is often conveniently represented as a label vector, $\mathbf{y} \in \{0, 1\}^K$, where $\mathbf{y}[k] = 1$ if and only if $k \in \mathcal{Y}$. Given a dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, which contains N training examples $(\mathbf{x}_n, \mathbf{y}_n)$, the multi-label classification algorithm uses $\mathcal{D}$ to find a classifier h: $\mathcal{X} \to \mathcal{Y}$ anticipating that h predicts $\mathbf{y}$ well on any future (unseen) test example $(\mathbf{x}, \mathbf{y})$.

2.1.1 Binary Relevance

There are many existing algorithms for solving multi-label classification problems. The simplest and most intuitive one is binary relevance [15]. Binary relevance decomposes the original dataset $\mathcal{D}$ into K binary classification datasets, $\mathcal{D}_k = \{(\mathbf{x}_n, \mathbf{y}_n[k])\}_{n=1}^N$, and learns K independent binary classifiers $h_1, h_2, \ldots, h_K$. Each classifier h_k is learned from $\mathcal{D}_k$ and is responsible for predicting whether the label set $\mathcal{Y}$ includes label k. In other words, binary relevance transforms any multi-label classification problem into K binary classification problems, one per label.

When K is small, binary relevance is an efficient and effective baseline algorithm for multi-label classification. However, when K is large, the algorithm can be computationally expensive, as discussed next.

2.2 Label Space Dimension Reduction

In several different domains, one can find multi-label classification problems with a large number of labels. For example, the `eurovoc` [16] dataset is a text collection of document about European Union law and contains 3,993 labels.

The large number of labels challenges multi-label classification algorithms in three major ways [15]. i) The computational cost of training a multi-label classifier may be significantly increased by the number of labels. For example, although binary relevance is a highly efficient algorithm, when used to train K binary classifiers, it becomes computationally expensive if K is large. ii) In the prediction phase, the computational cost of prediction is also strongly affected by the number of labels, K. When K is large, the algorithms may not be efficient enough for applications that require fast response time. iii) The amount of storage space needed to save multi-label classifiers for prediction increases with K. For example, binary relevance algorithms require sufficient space to save K binary classifiers for prediction.

Facing the above challenges, LSDR (Label Space Dimension Reduction) offers a potential solution to these issues by compressing the K-dimensional label space before learning. LSDR transforms $\mathcal{D}$ into M datasets, where $\mathcal{D}_m = \{(\mathbf{x}_n, \mathbf{t}_n[m])\}_{n=1}^{N}$, $m = 1, 2, \ldots, M$, and $M \ll K$ such that the multi-label classification problem can be tackled efficiently without significant loss of prediction performance. In particular, LSDR involves solving, predicting with, and storing the models for only M, instead of K, learning tasks. Then, many benefits are realized. i) With an efficient encoding method, the computational cost of training multi-label classifiers depends primarily on M rather than K. ii) In the prediction phase, with an efficient decoding method, the computational cost of prediction also depends mainly on M. iii) The storage space for saving the multi-label classifier can mainly depend on M if the storage space needed for encoding and decoding is small enough.

2.2.1 Compressive Sensing

Compressive sensing [CS; 7], a precursor of LSDR, is based on the assumption that the label set vector $\mathbf{y}$ is sparse (i.e., contains few ones). Then, $\mathbf{y}$ can be "compressed" to a shorter code vector $\mathbf{t}$ by projecting $\mathbf{y}$ on M random directions $\mathbf{v}_1, \cdots, \mathbf{v}_M$, where $M \ll K$ can be determined according to the assumed sparsity level. Setting $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \cdots \mathbf{v}_M]^T$, CS transforms the original multi-label classification problem, in which $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, into M regression tasks with $\mathcal{D}_m = \{(\mathbf{x}_n, \mathbf{t}_n[m])\}_{n=1}^N$, where $\mathbf{t}_n = \mathbf{V}\mathbf{y}_n$. After obtaining a multi-output regressor $\mathbf{r}(\mathbf{x})$ for predicting the code vector $\mathbf{t}$, CS decodes $\mathbf{r}(\mathbf{x})$ to the optimal label set vector by solving an optimization problem for each input $\mathbf{x}$ under the sparsity assumption. The prediction phase of CS may be inefficient because it must solve an optimization problem for each test instance.

2.2.2 Principal Label Space Transformation

Principal label space transformation [PLST; 8] is another approach to LSDR. PLST first shifts each label set vector $\mathbf{y}$ to $\mathbf{z} = \mathbf{y} - \bar{\mathbf{y}}$, where $\bar{\mathbf{y}} = \frac{1}{N}\sum_{n=1}^N \mathbf{y}_n$ is the estimated mean of the label set vectors. Similar to CS, PLST takes a matrix $\mathbf{V}$ that linearly maps $\mathbf{z}$ to the code vector $\mathbf{t}$ by $\mathbf{t} = \mathbf{V}\mathbf{z}$. However, unlike CS, PLST takes principal directions $\mathbf{v}_m$ (to be introduced next) rather than the random ones, and does not need to solve an optimization problem during decoding.

In particular, PLST considers only a matrix $\mathbf{V}$ with orthogonal rows, and decodes $\mathbf{r}(\mathbf{x})$ to the predicted labels by

$$\mathbf{h}(\mathbf{x}) = \text{round}\left(\mathbf{V}^T\mathbf{r}(\mathbf{x}) + \bar{\mathbf{y}}\right).$$

Such a decoding is called round-based decoding [8]. Tai and Lin [8] prove that when using round-based decoding and a linear transformation $\mathbf{V}$ that contains orthogonal rows, the common Hamming loss for evaluating multi-label classifiers [17] is bounded by

$$\text{Training Hamming Loss} \le c\left(\left\|\mathbf{r}(\mathbf{X}) - \mathbf{Z}\mathbf{V}^T\right\|_F^2 + \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2\right), \qquad (2.1)$$

where $\mathbf{r}(\mathbf{X})$ contains $\mathbf{r}(\mathbf{x}_n)^T$ as rows, $\mathbf{Z}$ contains $\mathbf{z}_n^T$ as rows and c is a constant that depends on K and N. The matrix $\mathbf{Z}\mathbf{V}^T$ then contains the code vector $\mathbf{t}_n^T$ as rows. The bound can be divided into two parts. The first part is $\left\|\mathbf{r}(\mathbf{X}) - \mathbf{Z}\mathbf{V}^T\right\|_F^2$, which represents the total prediction error from the regressor $\mathbf{r}(\mathbf{x}_n)$ to the desired code vectors $\mathbf{t}_n$. The second part is $\left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2$, which stands for the total encoding error for projecting $\mathbf{z}_n$ into the closest vector in $span\{\mathbf{v}_1, \cdots, \mathbf{v}_M\}$, which is $\mathbf{V}^T t_n$.

PLST is derived by minimizing the encoding error [8]. More specifically, PLST finds the M by K matrix $\mathbf{V}$ that solves

$$\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \quad \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2 .$$

The optimal $\mathbf{V}$ can be computed by applying the singular value decomposition on $\mathbf{Z}$ to generate $\mathbf{Z} = \mathbf{A}\mathbf{\Sigma}\mathbf{B}$, and taking as $\mathbf{V}$ the M rows of $\mathbf{B}$ that correspond to the M largest singular values. The M vectors $\mathbf{v}_1, \cdots, \mathbf{v}_M$ are called the principal directions for representing $\mathbf{z}_n$.

We note that PLST can be viewed as a linear case of the kernel dependency estimation (KDE) algorithm [18]. Nevertheless, the general nonlinear KDE algorithm must solve a computationally expensive pre-image problem for each test input $\mathbf{x}$ during the prediction phase. The linearity of PLST avoids the pre-image problem and enjoys efficient round-based decoding. In this thesis, we will focus on the linear case in order to design efficient algorithms for LSDR during both the training and prediction phases.

2.2.3 Canonical Correlation Analysis

A related technique that we will consider in this thesis is canonical correlation analysis [13]. CCA is a well-known statistical technique for analyzing the linear relationship between two multi-dimensional variables. More recently, it has been used for multi-label classification [14]. Traditionally, CCA is regarded as a feature space dimension reduction approach in multi-label classification. In this subsection, we discuss whether CCA can also be viewed as an LSDR approach.

CCA aims at finding two lists of basis vectors, one for each projected variable, such that the correlation of the projected variables is maximized. Formally, given an N by d matrix $\mathbf{X}$ with the n-th row being $\mathbf{x}_n^T$ (which is assumed to be zero mean) as well as an N by K matrix $\mathbf{Z}$ with the n-th row being $\mathbf{z}_n^T$ (which is assumed to be zero mean), the first step of CCA identifies two basis vectors $\mathbf{w}_x \in \mathcal{R}^d$ and $\mathbf{w}_z \in \mathcal{R}^K$ such that the correlation coefficient between the canonical variables $\mathbf{c}_x = \mathbf{X}\mathbf{w}_x$ and $\mathbf{c}_z = \mathbf{Z}\mathbf{w}_z$ is maximized. The goal can be formalized as the following optimization problem:

$$\begin{aligned} \max_{\mathbf{w}_x,\mathbf{w}_z} \quad & \mathbf{w}_x^T\mathbf{X}^T\mathbf{Z}\mathbf{w}_z \qquad (2.2)\\ \text{subject to} \quad & \mathbf{w}_x^T\mathbf{X}^T\mathbf{X}\mathbf{w}_x = \mathbf{w}_z^T\mathbf{Z}^T\mathbf{Z}\mathbf{w}_z = 1. \end{aligned}$$

After solving (2.2), CCA obtains the first pair of basis vectors $(\mathbf{w}_x^{(1)}, \mathbf{w}_z^{(1)})$ and two vectors of canonical variables $\mathbf{c}_x^{(1)} = \mathbf{X}\mathbf{w}_x^{(1)}$ and $\mathbf{c}_z^{(1)} = \mathbf{Z}\mathbf{w}_z^{(1)}$. Then, CCA iteratively finds additional basis vectors $\mathbf{w}_x^{(i)}$ and $\mathbf{w}_z^{(i)}$. In the i-th iteration, the i-th pair $(\mathbf{w}_x^{(i)}, \mathbf{w}_z^{(i)})$ is chosen such that the correlation between $\mathbf{c}_x^{(i)} = \mathbf{X}\mathbf{w}_x^{(i)}$ and $\mathbf{c}_z^{(i)} = \mathbf{Z}\mathbf{w}_z^{(i)}$ is maximized, under the constraint that $\mathbf{c}_x^{(i)}$ is uncorrelated to all other $\mathbf{c}_x^{(j)}$ and $\mathbf{c}_z^{(j)}$ for $1 \le j < i$. That is, CCA is equivalent to simultaneously solving the following constrained optimization problem:

$$\begin{aligned} \max_{\mathbf{W}_x,\mathbf{W}_z} \quad & \text{tr}\left(\mathbf{W}_x\mathbf{X}^T\mathbf{Z}\mathbf{W}_z^T\right) && (2.3)\\ \text{subject to} \quad & \mathbf{W}_x\mathbf{X}^T\mathbf{X}\mathbf{W}_x^T = \mathbf{W}_z\mathbf{Z}^T\mathbf{Z}\mathbf{W}_z^T = \mathbf{I}, && (2.4)\\ & \mathbf{W}_x\mathbf{X}^T\mathbf{Z}\mathbf{W}_z^T \text{ is a diagonal matrix.} && (2.5) \end{aligned}$$

Here, $\mathbf{W}_x$ is the matrix with the i-th row $\left(\mathbf{w}_x^{(i)}\right)^T$, and $\mathbf{W}_z$ is the matrix with the i-th row $\left(\mathbf{w}_z^{(i)}\right)^T$. Kettenring [19] showed that constraint (2.5) is implied by constraint (2.4). Therefore, (2.5) can be removed, and problem (2.3) becomes equivalent to

$$\begin{aligned} \min_{\mathbf{W}_x,\mathbf{W}_z} \quad & \left\|\mathbf{X}\mathbf{W}_x^T - \mathbf{Z}\mathbf{W}_z^T\right\|_F^2 \qquad (2.6)\\ \text{subject to} \quad & \mathbf{W}_x\mathbf{X}^T\mathbf{X}\mathbf{W}_x^T = \mathbf{W}_z\mathbf{Z}^T\mathbf{Z}\mathbf{W}_z^T = \mathbf{I}. \end{aligned}$$

When CCA is considered is the context of multi-label classification, $\mathbf{X}$ is the matrix that contains the mean-shifted $\mathbf{x}_n^T$ as rows and $\mathbf{Z}$ is the shifted label matrix that contains the mean-shifted $\mathbf{y}_n^T$ as rows. Traditionally, CCA is used as a supervised feature space dimension reduction approach that discards $\mathbf{W}_z$ and uses only $\mathbf{W}_x$ to project features onto a lower-dimension space before learning with binary relevance [14, 20].

On the other hand, due to the symmetry between $\mathbf{X}$ and $\mathbf{Z}$, we can also view CCA as an approach to feature-aware LSDR. In particular, CCA is equivalent to first seeking projection directions $\mathbf{W}_z$ of $\mathbf{Z}$, and then performing a multi-output linear regression from $\mathbf{x}_n$ to $\mathbf{W}_z\mathbf{z}_n$, under the constraints $\mathbf{W}_x\mathbf{X}^T\mathbf{X}\mathbf{W}_x^T = \mathbf{I}$, to obtain $\mathbf{W}_x$. However, the opportunity of using CCA for LSDR has not been seriously studied because $\mathbf{W}_z$ does not contain orthogonal rows. That is, unlike PLST, round-based decoding cannot be used and it is unclear how to design a suitable decoding scheme with CCA for LSDR.

Chapter 3

Proposed Algorithm

Inspired by CCA, we first design a variant that involves an appropriate decoding step. As suggested in Section 2.2.3, CCA is equivalent to finding a projection that minimizes the squared prediction error under the constraints $\mathbf{W}_x\mathbf{X}^T\mathbf{X}\mathbf{W}_x^T = \mathbf{W}_z\mathbf{Z}^T\mathbf{Z}\mathbf{W}_z^T = \mathbf{I}$. If we drop the constraint on $\mathbf{W}_x$ in order to further decrease the squared prediction error and change $\mathbf{W}_z\mathbf{Z}^T\mathbf{Z}\mathbf{W}_z^T = \mathbf{I}$ to $\mathbf{W}_z\mathbf{W}_z^T = \mathbf{I}$ in order to enable round-based decoding, we obtain

$$\begin{aligned} \min_{\mathbf{W}_x,\mathbf{W}_z} \quad & \left\|\mathbf{X}\mathbf{W}_x^T - \mathbf{Z}\mathbf{W}_z^T\right\|_F^2 \\ \text{subject to} \quad & \mathbf{W}_z\mathbf{W}_z^T = \mathbf{I} \end{aligned} \tag{3.1}$$

Problem (3.1) preserves the original objective function of CCA and specifies that $\mathbf{W}_z$ must contain orthogonal rows for applying round-based decoding. We call this algorithm orthogonally constrained CCA (OCCA). Then, using the Hamming loss bound (2.1), when $\mathbf{V} = \mathbf{W}_z$ and $\mathbf{r}(\mathbf{x}) = \mathbf{X}\mathbf{W}_z^T$, OCCA minimizes $\|\mathbf{r}(\mathbf{x}) - \mathbf{Z}\mathbf{W}_z^T\|$ in (2.1) with the hope that the Hamming loss is also minimized. In other words, OCCA is employed for the orthogonal directions $\mathbf{V}$ that are "easy to learn" (of low prediction error) in terms of linear regression.

For every fixed $\mathbf{W}_z = \mathbf{V}$ in (3.1), the optimization problem for $\mathbf{W}_x$ is simply a linear regression from $\mathbf{X}$ to $\mathbf{Z}\mathbf{V}^T$. Then, the optimal $\mathbf{W}_x$ can be computed by a closed-form solution

$$\mathbf{W}_x^T = \mathbf{X}^\dagger \mathbf{Z}\mathbf{V}^T,$$

where $\mathbf{X}^\dagger$ is the pseudo inverse of $\mathbf{X}$. When the optimal $\mathbf{W}_x$ is inserted back into (3.1), the optimization problem becomes

$$\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \quad \left\|\mathbf{X}\mathbf{X}^\dagger\mathbf{Z}\mathbf{V}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2 \tag{3.2}$$

which is equivalent to

$$\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \text{tr}\left(\mathbf{V}\mathbf{Z}^T\left(\mathbf{I}-\mathbf{H}\right)\mathbf{Z}\mathbf{V}^T\right). \tag{3.3}$$

The matrix $\mathbf{H} = \mathbf{X}\mathbf{X}^\dagger$ is called the hat matrix for linear regression [21]. Similar to PLST, by using Eckart-Young theorem [22], we can solve problem (3.3) by considering the eigenvectors that correspond to the largest eigenvalues of $\mathbf{Z}^T(\mathbf{H}-\mathbf{I})\mathbf{Z}$. Comparing OCCA to PLST, we see that they are similarly based on computing the singular value (eigenvalue) decomposition of a matrix. However, OCCA computes the pseudo inverse and a few matrix multiplications, and hence is somewhat slower. Unlike PLST, however, OCCA is feature-aware and captures the input-output relation during label space dimension reduction.

3.1 Conditional Principal Label Space Transformation

From the previous discussions, OCCA has been shown to capture the input-output relation in multi-label classification and to minimize the squared prediction error in bound (2.1) with the "easy" directions. In contrast, PLST minimizes the encoding error in bound (2.1) with the "principal" directions. In this section, we combine the benefits of the two algorithms, and minimize the two error terms simultaneously with the "conditional principal" directions. We begin by continuing our derivation of OCCA, which obtains $\mathbf{r}(\mathbf{x})$ by a linear regression from $\mathbf{X}$ to $\mathbf{Z}\mathbf{V}^T$. If we minimize both terms in (2.1) together with such a linear regression, the optimization problem becomes

$$\min_{\mathbf{W},\mathbf{V}\mathbf{V}^T=\mathbf{I}} c\left(\left\|\mathbf{X}\mathbf{W}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2 + \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2\right)$$

$$\Rightarrow \quad \min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \operatorname{tr}\left(\mathbf{V}\mathbf{Z}^T\left(\mathbf{I}-\mathbf{H}\right)\mathbf{Z}\mathbf{V}^T + \mathbf{Z}^T\mathbf{Z} - \mathbf{V}^T\mathbf{V}\mathbf{Z}^T\mathbf{Z} - \mathbf{Z}^T\mathbf{Z}\mathbf{V}^T\mathbf{V} + \mathbf{V}^T\mathbf{V}\mathbf{Z}^T\mathbf{Z}\mathbf{V}^T\mathbf{V}\right) \tag{3.4}$$

$$\Rightarrow \quad \min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \operatorname{tr}\left(\mathbf{V}\mathbf{Z}^T\left(\mathbf{I}-\mathbf{H}\right)\mathbf{Z}\mathbf{V}^T - \mathbf{V}\mathbf{Z}^T\mathbf{Z}\mathbf{V}^T\right) \tag{3.5}$$

$$\Rightarrow \quad \max_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} \operatorname{tr}\left(\mathbf{V}\mathbf{Z}^T\mathbf{H}\mathbf{Z}\mathbf{V}^T\right) \tag{3.6}$$

Equation (3.5) can be derived by cyclic permutation to eliminate the pair $\mathbf{V}$ and $\mathbf{V}^T$ and combine the last three terms of (3.4).

Problem (3.6) can again be solved by taking the eigenvectors with the largest eigenvalues of $\mathbf{Z}^T\mathbf{H}\mathbf{Z}$ as the rows of $\mathbf{V}$. Such a matrix $\mathbf{V}$ minimizes the prediction error term and the encoding error term simultaneously. The resulting algorithm is called conditional principal label space transformation (CPLST), as shown in Algorithm 1.

Algorithm 1 Conditional Principal Label Space Transformation

```
1: Let $\mathbf{Z} = [\mathbf{z}_1 \ldots \mathbf{z}_N]^T$ with $\mathbf{z}_n = \mathbf{y}_n - \bar{\mathbf{y}}$.
2: Preform SVD on $\mathbf{Z}^T\mathbf{H}\mathbf{Z}$ to obtain $\mathbf{Z}^T\mathbf{H}\mathbf{Z} = \mathbf{A}\mathbf{\Sigma}\mathbf{B}$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N$. Let
   $\mathbf{V}_M$ contain the top $M$ rows of $\mathbf{B}$.
3: Encode $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ to $\{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$, where $\mathbf{t_n} = \mathbf{V}_M\mathbf{z}_n$.
4: Learn a multi-dimension regressor $\mathbf{r}(\mathbf{x})$ from $\{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$.
5: Predict the label-set of an instance $\mathbf{x}$ by $\mathbf{h}(\mathbf{x}) = \text{round}\left(\mathbf{V}_M^T\mathbf{r}(\mathbf{x}) + \bar{\mathbf{y}}\right)$.
```

CPLST balances the squared prediction error with the encoding error and is closely related with bound (2.1). Moreover, in contrast with PLST, which uses the key unconditional correlations, CPLST is feature-aware and allows the capture of conditional correlations [23].

We summarizes the three algorithms in Table 3.1, and we will compare them empirically in Chapter 4. We can observe that the three algorithms are similar. They all operate with an SVD on a K by K matrix. PLST focuses on the encoding error and does not consider the features during LSDR, i.e. it is feature-unaware. On the other hand, CPLST and OCCA are feature-aware approaches, which consider features during label space dimension reduction. When using linear regression as the multi-output regressor, CPLST simultaneously minimizes the two terms in bound (2.1), while OCCA minimizes only one term of the bound.

Table 3.1: Summary of three LSDR algorithms

Algorithm	Matrix for SVD	LSDR	Relation to bound (2.1)
PLST	$\mathbf{Z}^T\mathbf{Z}$	feature-unaware	minimizes the encoding error (best encoding)
OCCA	$\mathbf{Z}^T(\mathbf{H}-\mathbf{I})\mathbf{Z}$	feature-aware	minimizes the squared prediction error (“easiest” regression tasks)
CPLST	$\mathbf{Z}^T\mathbf{H}\mathbf{Z}$	feature-aware	minimizes both

In contrast to PLST, the two feature-aware approaches OCCA and CPLST must calculate the matrix $\mathbf{H}$. This may result in slower performance than PLST during LSDR if the dimension d of the input space $\mathbf{X}$ is extremely large.

3.2 Kernelization and Regularization

Nonlinear transformation of the feature space is an important technique in machine learning [24]. This technique is utilized to capture sophisticated relationships between features and labels by transforming the features into a higher-dimensional feature space before performing learning algorithms. When the learning algorithms use linear models, the nonlinear feature transformation can often be conducted implicitly through the kernel trick – representing the inner product after transformation as a special function called the kernel [24]. With a proper choice of the kernel, kernelization allows transforming into a feature space with very high or even infinite number of dimensions. If we are going to use a feature space with very high dimensions, however, there is a clear danger of overfitting. Hence, we need to regularize the power of the learning algorithm. In this subsection, we show that kernelization and regularization can be applied to OCCA and CPLST.

In Section 3.1, we derive OCCA and CPLST by using linear regression as the underlying multi-output regression algorithm. Next, we replace linear regression by its kernelized form with ℓ_2 regularization, kernel ridge regression [25], as the underlying regression algorithm. Kernel ridge regression(KRR) considers a feature mapping $\Phi : \mathcal{X} \to \mathcal{F}$ before performing regularized linear regression. According to the feature mapping Φ, the kernel function $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$ is defined as an inner product of $\mathbf{x}$ and $\mathbf{x}'$ after mapped into the space $\mathcal{F}$. On the other hand, kernel ridge regression also adds a ℓ_2 regularization term $\frac{1}{2}\lambda\|\mathbf{w}\|^2$, where λ is the regularization parameter, into the objective function for regularizing the power of the high-dimensional linear regression.

That is, when given the dataset $\{(\mathbf{x}_n, r_n)\}_{n=1}^N$, where r_n is the target value of the input vector, the task of kernel ridge regression is to find an optimal $\mathbf{w}$ in the space $\mathcal{F}$ such that $\mathbf{w}$ minimizes

$$\sum_{n=1}^{N} (\mathbf{w}^T \Phi(\mathbf{x}_n) - r_n)^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2. \tag{3.7}$$

In particular, if $\Phi(\mathbf{x})$ can be explicitly computed, it is known that the closed-form solution of (3.7) is [25]

$$\mathbf{w} = \mathbf{\Phi}^T \left(\lambda \mathbf{I} + \mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1} \mathbf{r} = \mathbf{\Phi}^T \left(\lambda \mathbf{I} + \mathbf{K}\right)^{-1} \mathbf{r}, \tag{3.8}$$

where $\mathbf{r}$ is an N by 1 vector whose n-th element is r_n, $\mathbf{\Phi}$ is the matrix containing $\Phi(\mathbf{x}_n)^T$ as rows, and $\mathbf{K}$ is the matrix with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$. That is, $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$ and is called the kernel matrix of $\mathbf{X}$.

Now, we derive kernel-CPLST by inserting the optimal $\mathbf{W}$ into the Hamming loss bound (2.1). Kernel-CPLST uses KRR as underlying regression problem to map from $\mathbf{X}$ to $\mathbf{Z}\mathbf{V}^T$ and the optimal $\mathbf{W}$ would be

$$\mathbf{W}^T = \mathbf{\Phi}^T \left(\lambda \mathbf{I} + \mathbf{K}\right)^{-1} \mathbf{Z}\mathbf{V}^T. \tag{3.9}$$

When substituting (3.9) into minimizing the loss bound (2.1) and letting $\mathbf{Q} = (\lambda\mathbf{I} + \mathbf{K})^{-1}$, the optimization problem becomes

$$
\begin{aligned}
&\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} c \left(\left\|\mathbf{\Phi}\mathbf{W}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2 + \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2\right) \\
\Rightarrow &\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} c \left(\left\|\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{Q}\mathbf{Z}\mathbf{V}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2 + \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2\right) \\
\Rightarrow &\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} c \left(\left\|\mathbf{K}\mathbf{Q}\mathbf{Z}\mathbf{V}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2 + \left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2\right) \\
\Rightarrow &\min_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} c \operatorname{tr}\left(\mathbf{V}\mathbf{Z}^T\left(-2\mathbf{K}\mathbf{Q} + \mathbf{Q}\mathbf{K}\mathbf{K}\mathbf{Q} + \mathbf{I}\right)\mathbf{Z}\mathbf{V}^T - \mathbf{V}\mathbf{Z}^T\mathbf{Z}\mathbf{V}^T\right) \\
\Rightarrow &\max_{\mathbf{V}\mathbf{V}^T=\mathbf{I}} c \operatorname{tr}\left(\mathbf{V}\mathbf{Z}^T\left(2\mathbf{K}\mathbf{Q} - \mathbf{Q}\mathbf{K}\mathbf{K}\mathbf{Q} - \mathbf{I}\right)\mathbf{Z}\mathbf{V}^{\mathbf{T}} + \mathbf{V}\mathbf{Z}^T\mathbf{Z}\mathbf{V}^T\right)
\end{aligned}
\tag{3.10}
$$

Notice that in equation (3.10), kernel-CPLST do not need to explicitly compute the matrix $\mathbf{\Phi}$ and only needs the kernel matrix $\mathbf{K}$ (that can be computed through the kernel function $\mathbf{k}$). Therefore, a high or even an infinite dimensional feature transform can be used to assist label space dimension reduction in kernel-CPLST through a suitable kernel function, which is known as the kernel trick [26].

By using Eckart-Young theorem [22], problem (3.10) can again be solved by considering the eigenvectors with the largest eigenvalues of $\mathbf{Z}^T\left(2\mathbf{K}\mathbf{Q} - \mathbf{Q}\mathbf{K}\mathbf{K}\mathbf{Q}\right)\mathbf{Z}$ as the rows of $\mathbf{V}$. Similarly, we can extend OCCA to kernel-OCCA which takes the eigenvectors with the largest eigenvalues of $\mathbf{Z}^T\left(2\mathbf{K}\mathbf{Q} - \mathbf{Q}\mathbf{K}\mathbf{K}\mathbf{Q} - \mathbf{I}\right)\mathbf{Z}$ as the row of $\mathbf{V}$.

The kernelized formulations are computationally more expensive than linear ones when $N \gg d$, because CPLST and OCCA only need to calculate the pseudo inverse of the N by d matrix $\mathbf{X}$, while kernel-CPLST and kernel-OCCA need to calculate the inversion of the N by N kernel matrix. Somehow, kernel-CPLST and kernel-OCCA is more sophisticated than their linear counterparts and may be able to exploit more complicated interactions of features to assist label space dimension reduction.

Chapter 4

Experiment

In this chapter, we conduct experiments on eight real-world datasets to validate the performance of CPLST and other LSDR approaches. In Section 4.1, we compare the performance of different LSDR approaches coupled with linear regression. Then, in Section 4.2 and 4.3, we compare the LSDR approaches, including their kernelized version, when coupled with kernel ridge regression, with either a systematic parameter selection procedure (Section 4.2) or a fixed parameter setting (Section 4.3). In Section 4.4, we demonstrate the practical usefulness of the proposed CPLST by using the decision tree instead of the (kernel) ridge regression as the actual multi-output regression method. In Section 4.5, we compare CPLST with kernel-CPLST in detail with either linear regression or kernel ridge regression as the regression method. In Section 4.6, we show the optimal reduction size of the LSDR approaches. Then, we analysis the hamming loss bound and try to determine the optimal reduced size in Section 4.7. Finally, we give some discussion about the reason why CPLST would be better than other related FSDR algorithms in Section 4.8.

The datasets (Table 4.1) are all downloaded from Mulan [27] and cover a variety of domains. Because kernel ridge regression itself, kernel-OCCA and kernel-CPLST need to invert an N by N matrix, we can only afford to conduct a fair comparison using mid-sized ($N \leq 10000$) datasets. In each run of the experiment, we randomly sample 80% of the dataset for training and reserve the other 20% for testing. All the results are reported with the mean and the standard error over 100 different random runs.

Table 4.1 shows the datasets we use. Label cardinality of a dataset is the average number of labels per instance and label density of a dataset is cardinality divided by the number of labels K [15].

We include PLST, OCCA, CPLST, kernel-OCCA and kernel-CPLST in our comparison. We do not include Compressive Sensing [7] in the comparison because earlier work [8] has shown that the algorithm is more sophisticated while being inferior to PLST. In addition to those LSDR approaches, we also consider a simple baseline approach, partial binary relevance (PBR) [8]. PBR randomly selects M labels from the original label set during training and only learns those M binary classifiers for prediction. For those unselected labels, PBR directly predicts -1 without any training to match the sparsity assumption as exploited by Compressive Sensing [7].

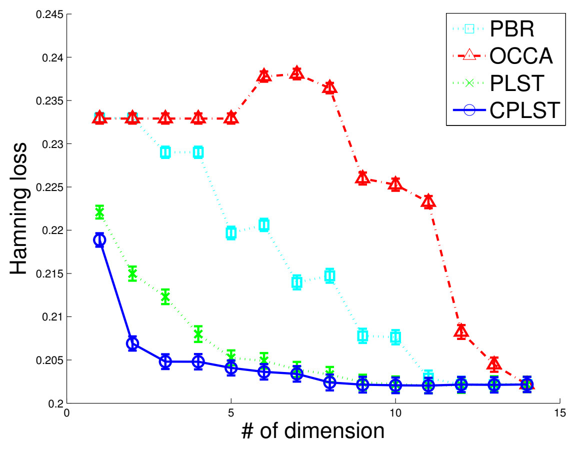
Table 4.1: Dataset Statistics

Dataset	Domain	# Instances	# Labels (K)	Cardinality	Density
`bibtex`	text	7395	159	2.402	0.015
`corel5k`	images	5000	374	3.522	0.009
`emotions`	music	593	6	1.869	0.311
`enron`	text	1702	53	3.378	0.064
`genbase`	biology	662	27	1.252	0.046
`medical`	video	978	45	1.245	0.028
`scene`	image	2407	6	1.074	0.179
`yeast`	biology	2417	14	4.237	0.303

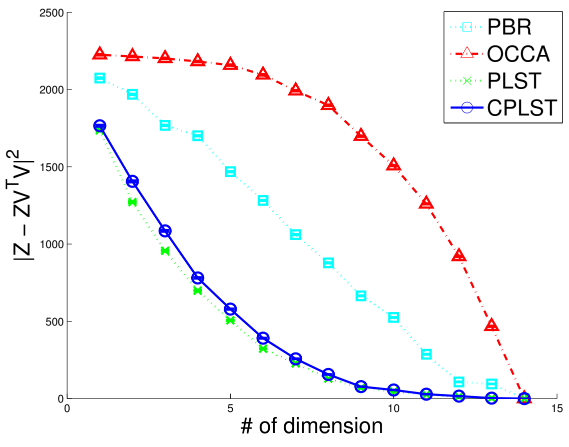
4.1 Label Space Dimension Reduction with Linear Regression

First, we couple PBR, OCCA, PLST and CPLST with linear regression and compare their test performance on the `yeast` dataset, which is of 14 classes.

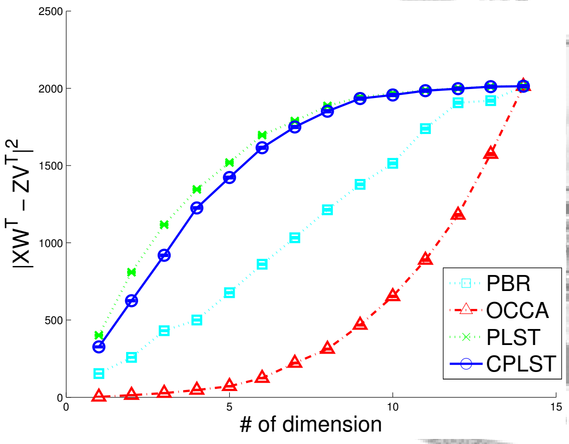
Figure 4.1(a) shows the test Hamming loss with respect to the possible M used. It is clear that CPLST is significantly better than the other three approaches. With only three dimensions, the test performance of CPLST is close to the full PBR (which is the regular binary relevance) using all 14 dimensions. PLST can reach similar performance to CPLST only at a larger M. The other two algorithms, OCCA and PBR, are both significantly worse than CPLST.



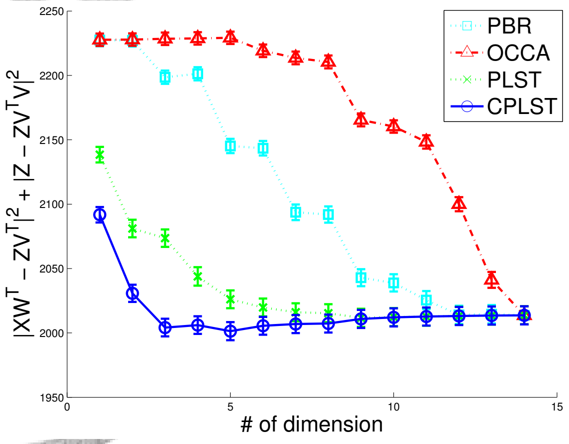
(a) Hamming loss



(b) encoding error



(c) prediction error



(d) loss bound

Figure 4.1: `yeast`: test results of label space dimension reduction algorithm when coupled with linear regression

To understand the cause of the different performance, we also plot the (test) encoding error $\left\|\mathbf{Z} - \mathbf{Z}\mathbf{V}^T\mathbf{V}\right\|_F^2$, the prediction error $\left\|\mathbf{X}\mathbf{W}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2$, and the loss bound (2.1) in Figure 4.1. Figure 4.1(b) shows the encoding error on the test set, which matches the design of PLST. Regardless of the approaches used, the encoding error decreases to 0 when all 14 dimensions are used because the $\{\mathbf{v}_m\}$'s are able to span the whole label space. As expected, PLST achieves the lowest encoding error across every number of dimensions. CPLST partially minimizes the encoding error in its objective function, and hence also achieves a decent encoding error. On the other hand, OCCA is blind to and hence worst at the encoding error. In particular, the encoding error of OCCA is even worse than the error of the baseline PBR. The result shows that OCCA pays a big price in encoding error by focusing on only the prediction error.

Figure 4.1(c) shows the prediction error $\left\|\mathbf{X}\mathbf{W}^T - \mathbf{Z}\mathbf{V}^T\right\|_F^2$ on the test set, which matches the design of OCCA. First, OCCA indeed achieves the lowest prediction error across all number of dimensions. PLST, which is blind to the prediction error, reaches the highest prediction error, and is even worse than PBR. The results further reveal the trade-off between the encoding error and the prediction error: more efficient encodings of the label space are harder to predict. PLST takes the more efficient encoding to the extreme, and results in worse prediction error; OCCA, on the other hand, is better in terms of the prediction error, but leads to the least efficient encoding.

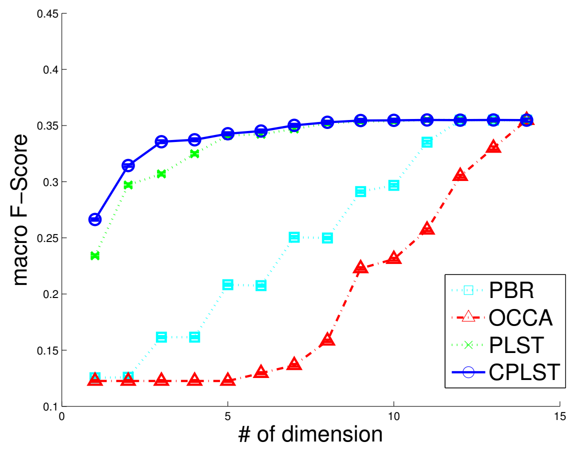
Figure 4.1(d) shows the scaled upper bound (2.1) of the Hamming loss, which equals the sum of the encoding error and the prediction error. CPLST is designed to knock down this bound, which explains its behavior in Figure 4.1(d) and echoes its superior performance in Figure 4.1(a). In fact, Figure 4.1(d) shows that the bound (2.1) is quite indicative of the performance differences in Figure 4.1(a). The results demonstrate that CPLST explores the trade-off between the encoding error and the prediction error in an optimal manner to reach the best performance for label space dimension reduction.

While CPLST is designed for minimizing the upper bound of the Hamming loss, we also show the results of four popular evaluation measures (the higher the better) for reference: macro-averaged F_1 score, micro-averaged F_1 score, the exact match ratio [15], and the area under curve (AUC) [28].

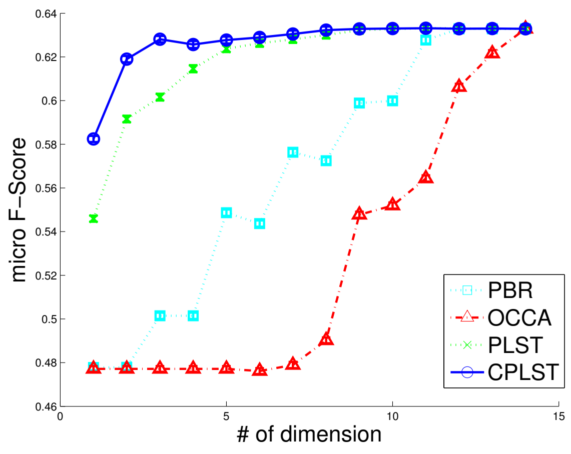
The F_1 measure is the harmonic mean of the precision and the recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

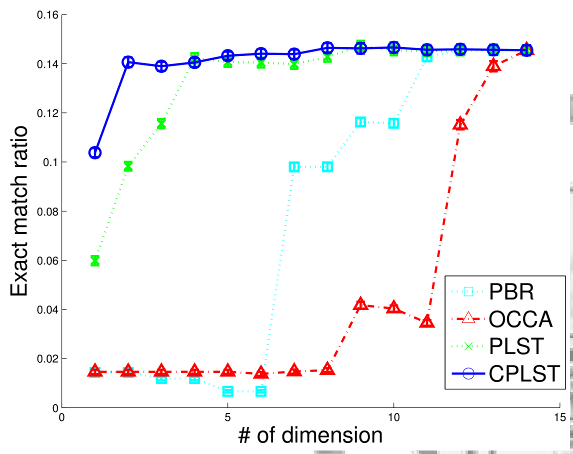
The macro-averaged F_1 score is the mean of F_1 scores of over all labels and thus treats each label equally. The micro-averaged F_1 score, on the other hand, mixes all the labels together for calculating the precision the recall, and thus favors the classes with more positive (+1) examples.



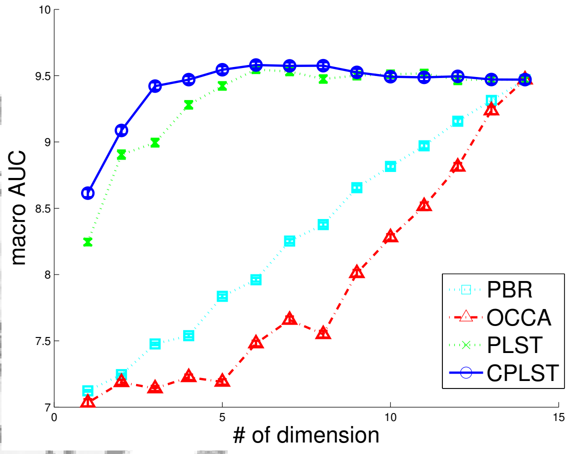
(a) macro F-score



(b) micro F-score



(c) exact match ratio



(d) macro AUC

Figure 4.2: `yeast`: test result of label space dimension reduction algorithms when coupled with linear regression

The exact match ratio for N examples is defined as:

$$\text{Exact Match Ratio } = \frac{1}{N}\sum_{i=1}^{N}[\![\mathbf{y}_i = \tilde{\mathbf{y}}_\mathbf{i}]\!].$$

Exact match ratio is a very strict measure since the predicted and the target label sets need to be exactly the same to count as a match.

The AUC is the area under receiver operating characteristic curve [28] and can illustrate the performance of a classifier as its discrimination threshold is varied. In particular, AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one [28].

The result of these four measures (the higher the better) are shown in Figure 4.2. Similar to the results of the Hamming loss, CPLST is the best of the four label space dimension reduction approaches, better than PLST in particular; while OCCA remains to be the worst.

The test Hamming loss achieved by PLST and CPLST on other datasets with different M are reported in Table 4.2. In most datasets, we can see that CPLST is at least as effective as PLST. Moreover, in the `bibtex`, `scene` and `yeast` datasets, CPLST performs significantly better than PLST.

Note that in the `medical` and `enron` datasets, all FSDR algorithms overfit when using many dimensions. That is, the performance of algorithms would be better when using fewer dimensions than the full binary relevance. These results demonstrate that LSDR approaches, like their feature space dimension reduction counterparts, can potentially help resolve the issue of overfitting.

4.2 Label Space Dimension Reduction with Kernel Ridge Regression

In this Section, we conduct experiments for demonstrating the performance of kernelization and regularization. For kernel-CPLST, we use the Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ during label space dimension reduction and take kernel ridge regression with the same kernel and the same regularization parameter as the underlying multi-output regression method. We also couple PLST with kernel ridge regression for a fair comparison. The choice of the Gaussian kernel parameter γ and the regularization parameter λ is done by a 2-stage procedure. The goal of the first stage is to quickly identify the range of $(\log_2 \lambda, \log_2 \gamma)$ to be searched further. For each dataset, we take one particular realization of the random split, and conduct 5-fold cross validation using the sum of the Hamming loss across all dimension (the area under the curve in Figure 4.1(a)) as the criteria. We examine the choices of some ranges to make sure that the best parameter combinations of all approaches are not at the boundary of the range. The range

Table 4.2: Test Hamming loss of PLST and CPLST with linear regression

Dataset	Algorithm	20%	40%	60%	80%	100%
`bibtex`	PBR	0.0146 ± 0.0001	0.0140 ± 0.0001	0.0134 ± 0.0001	0.0129 ± 0.0000	**0.0123 ± 0.0000**
	OCCA	0.0140 ± 0.0000	0.0137 ± 0.0000	0.0135 ± 0.0000	0.0129 ± 0.0000	**0.0123 ± 0.0000**
	PLST	0.0129 ± 0.0000	0.0125 ± 0.0000	0.0124 ± 0.0000	**0.0123 ± 0.0000**	**0.0123 ± 0.0000**
	CPLST	**0.0127 ± 0.0000**	**0.0124 ± 0.0000**	**0.0123 ± 0.0000**	**0.0123 ± 0.0000**	**0.0123 ± 0.0000**
`corel5k`	PBR	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**
	OCCA	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**
	PLST	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**
	CPLST	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**	**0.0094 ± 0.0000**
`emotions`	PBR	0.3077 ± 0.0017	0.2738 ± 0.0017	0.2490 ± 0.0015	0.2397 ± 0.0016	**0.2040 ± 0.0022**
	OCCA	0.3125 ± 0.0014	0.2998 ± 0.0014	0.2953 ± 0.0015	0.2984 ± 0.0017	**0.2040 ± 0.0022**
	PLST	**0.2207 ± 0.0020**	**0.2064 ± 0.0023**	**0.1982 ± 0.0022**	**0.2013 ± 0.0020**	**0.2040 ± 0.0022**
	CPLST	**0.2189 ± 0.0019**	**0.2059 ± 0.0022**	**0.1990 ± 0.0022**	**0.2015 ± 0.0021**	**0.2040 ± 0.0022**
`enron`	PBR	**0.0645 ± 0.0003**	0.0769 ± 0.0004	0.0829 ± 0.0005	**0.0937 ± 0.0006**	**0.1028 ± 0.0007**
	OCCA	0.0656 ± 0.0002	**0.0722 ± 0.0003**	**0.0820 ± 0.0005**	0.0987 ± 0.0006	**0.1028 ± 0.0007**
	PLST	0.0728 ± 0.0004	0.0860 ± 0.0005	0.0946 ± 0.0006	0.1006 ± 0.0007	**0.1028 ± 0.0007**
	CPLST	0.0729 ± 0.0004	0.0864 ± 0.0005	0.0943 ± 0.0006	0.1006 ± 0.0007	**0.1028 ± 0.0007**
`genbase`	PBR	0.0313 ± 0.0006	0.0224 ± 0.0004	0.0144 ± 0.0003	0.0093 ± 0.0003	**0.0007 ± 0.0001**
	OCCA	0.0455 ± 0.0003	0.0376 ± 0.0005	0.0180 ± 0.0003	0.0165 ± 0.0004	**0.0007 ± 0.0001**
	PLST	**0.0169 ± 0.0004**	**0.0040 ± 0.0002**	**0.0012 ± 0.0001**	**0.0009 ± 0.0001**	**0.0007 ± 0.0001**
	CPLST	**0.0168 ± 0.0004**	**0.0041 ± 0.0002**	**0.0012 ± 0.0001**	**0.0008 ± 0.0001**	**0.0007 ± 0.0001**
`medical`	PBR	0.0645 ± 0.0003	0.0769 ± 0.0004	0.0829 ± 0.0005	0.0937 ± 0.0006	**0.1028 ± 0.0007**
	OCCA	0.0656 ± 0.0002	0.0722 ± 0.0003	0.0820 ± 0.0005	0.0987 ± 0.0006	**0.1028 ± 0.0007**
	PLST	**0.0346 ± 0.0004**	**0.0407 ± 0.0005**	**0.0472 ± 0.0005**	**0.0490 ± 0.0005**	**0.0497 ± 0.0006**
	CPLST	**0.0346 ± 0.0004**	**0.0406 ± 0.0005**	**0.0471 ± 0.0005**	**0.0490 ± 0.0005**	**0.0497 ± 0.0006**
`scene`	PBR	0.1619 ± 0.0004	0.1557 ± 0.0005	0.1496 ± 0.0006	0.1315 ± 0.0007	**0.1106 ± 0.0008**
	OCCA	0.1789 ± 0.0003	0.1665 ± 0.0005	0.1520 ± 0.0005	0.1328 ± 0.0005	**0.1106 ± 0.0008**
	PLST	0.1809 ± 0.0004	0.1718 ± 0.0006	0.1566 ± 0.0007	0.1321 ± 0.0008	**0.1106 ± 0.0008**
	CPLST	**0.1744 ± 0.0004**	**0.1532 ± 0.0005**	**0.1349 ± 0.0005**	**0.1209 ± 0.0007**	**0.1106 ± 0.0008**
`yeast`	PBR	0.2330 ± 0.0006	0.2197 ± 0.0007	0.2147 ± 0.0008	0.2029 ± 0.0009	**0.2022 ± 0.0009**
	OCCA	0.2329 ± 0.0006	0.2329 ± 0.0006	0.2364 ± 0.0006	0.2233 ± 0.0007	**0.2022 ± 0.0009**
	PLST	0.2150 ± 0.0008	0.2052 ± 0.0009	0.2033 ± 0.0009	**0.2020 ± 0.0009**	**0.2022 ± 0.0009**
	CPLST	**0.2069 ± 0.0008**	**0.2041 ± 0.0009**	**0.2024 ± 0.0009**	**0.2020 ± 0.0009**	**0.2022 ± 0.0009**

(those within one standard error of the lower one are in bold)

of $(\log_2 \lambda, \log_2 \gamma)$ of all datasets in second stage is reported in Table 4.3. The second stage zooms into the smaller range of $(\log_2 \lambda, \log_2 \gamma)$ and conduct parameter selection with 5-fold cross validation on the actual training set, which is different across the 100 runs. After the second stage of the parameter selection procedure, the 80% training set as well as the selected parameter are fed to the FSDR approaches and the resulting multi-label classifier is evaluated on the corresponding test set.

When coupled with kernel ridge regression, the comparison between PLST and kernel-CPLST in terms of the Hamming loss is shown in Table 4.4. Table 4.4 shows that kernel-CPLST performs well for LSDR and outperforms the feature-unaware PLST in most cases. In particular, in five out of the eight datasets, kernel-CPLST is significantly better than PLST regardless of the number of dimensions used. Moreover, kernel-CPLST with full dimension performs better than other PLST with full dimension in most cases.

Table 4.3: The parameter range used in each dataset

Dataset	$\log_2 \lambda$	$\log_2 \gamma$
bibtex	$\{-11, -9, \ldots, -3\}$	$\{-11, -9, \ldots, -1\}$
corel5k	$\{-17, -15, \ldots, -3\}$	$\{-11, -9, \ldots, 5\}$
emotions	$\{-23, -21, \ldots, -11\}$	$\{-25, -23, \ldots, -11\}$
enron	$\{-11, -9, \ldots, 5\}$	$\{-17, -15, \ldots, -1\}$
genbase	$\{-45, -43, \ldots, -11\}$	$\{-23, -21, \ldots, -3\}$
medical	$\{-19, -17, \ldots, -1\}$	$\{-15, -13, \ldots, -1\}$
scene	$\{-21, -19, \ldots, 3\}$	$\{-9, -7, \ldots, 3\}$
yeast	$\{-19, -17, \ldots, 1\}$	$\{-5, -3, \ldots, 7\}$

Table 4.4: Test Hamming Loss of LSDR algorithm with Kernel Ridge Regression

Dataset	Algorithm	20%	40%	60%	80%	100%
bibtex	PBR	0.0147 ± 0.0001	0.0141 ± 0.0001	0.0135 ± 0.0001	0.0130 ± 0.0000	0.0125 ± 0.0000
	kernel-OCCA	0.0137 ± 0.0000	0.0132 ± 0.0000	0.0125 ± 0.0000	0.0124 ± 0.0000	**0.0120 ± 0.0000**
	PLST	0.0151 ± 0.0000	0.0151 ± 0.0000	0.0151 ± 0.0000	0.0151 ± 0.0000	0.0151 ± 0.0000
	kernel-CPLST	**0.0127 ± 0.0000**	**0.0123 ± 0.0000**	**0.0121 ± 0.0000**	**0.0120 ± 0.0000**	**0.0120 ± 0.0000**
corel5k	PBR	0.0094 ± 0.0000	0.0095 ± 0.0000	0.0095 ± 0.0000	0.0095 ± 0.0000	0.0095 ± 0.0000
	kernel-OCCA	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0093 ± 0.0000
	PLST	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000
	kernel-CPLST	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**
emotions	PBR	0.3091 ± 0.0018	0.2747 ± 0.0019	0.2501 ± 0.0019	0.2401 ± 0.0020	0.2065 ± 0.0026
	kernel-OCCA	0.3112 ± 0.0014	0.2997 ± 0.0014	0.3026 ± 0.0016	0.2985 ± 0.0018	**0.1983 ± 0.0025**
	PLST	**0.2218 ± 0.0020**	0.2074 ± 0.0023	**0.1983 ± 0.0026**	0.2000 ± 0.0025	**0.2002 ± 0.0025**
	kernel-CPLST	**0.2231 ± 0.0020**	**0.2071 ± 0.0024**	**0.1981 ± 0.0025**	**0.1973 ± 0.0027**	**0.1988 ± 0.0027**
enron	PBR	0.0537 ± 0.0002	0.0517 ± 0.0002	0.0467 ± 0.0003	0.0471 ± 0.0003	0.0470 ± 0.0003
	kernel-OCCA	0.0636 ± 0.0002	0.0641 ± 0.0002	0.0640 ± 0.0002	0.0607 ± 0.0002	0.0499 ± 0.0003
	PLST	0.0460 ± 0.0002	0.0462 ± 0.0002	0.0466 ± 0.0002	0.0468 ± 0.0002	0.0469 ± 0.0002
	kernel-CPLST	**0.0453 ± 0.0002**	**0.0454 ± 0.0002**	**0.0455 ± 0.0002**	**0.0455 ± 0.0002**	**0.0456 ± 0.0002**
genbase	PBR	0.0313 ± 0.0006	0.0225 ± 0.0004	0.0145 ± 0.0004	0.0094 ± 0.0003	**0.0008 ± 0.0001**
	kernel-OCCA	0.0312 ± 0.0006	0.0231 ± 0.0004	0.0195 ± 0.0003	0.0163 ± 0.0003	**0.0007 ± 0.0001**
	PLST	**0.0169 ± 0.0004**	**0.0039 ± 0.0002**	0.0014 ± 0.0001	0.0010 ± 0.0001	**0.0008 ± 0.0001**
	kernel-CPLST	**0.0170 ± 0.0004**	**0.0040 ± 0.0002**	**0.0013 ± 0.0001**	**0.0009 ± 0.0001**	**0.0008 ± 0.0001**
medical	PBR	0.0255 ± 0.0001	0.0192 ± 0.0002	0.0155 ± 0.0002	0.0144 ± 0.0002	0.0139 ± 0.0002
	kernel-OCCA	0.0272 ± 0.0001	0.0262 ± 0.0002	0.0246 ± 0.0002	0.0210 ± 0.0002	0.0104 ± 0.0002
	PLST	0.0136 ± 0.0002	0.0106 ± 0.0002	0.0103 ± 0.0002	0.0102 ± 0.0002	0.0102 ± 0.0002
	kernel-CPLST	**0.0131 ± 0.0002**	**0.0098 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**
scene	PBR	**0.1578 ± 0.0005**	0.1494 ± 0.0006	0.1412 ± 0.0008	0.1214 ± 0.0008	0.0849 ± 0.0009
	kernel-OCCA	0.1659 ± 0.0007	**0.1440 ± 0.0008**	0.1273 ± 0.0007	0.1093 ± 0.0006	0.0751 ± 0.0007
	PLST	0.1713 ± 0.0004	0.1468 ± 0.0006	**0.1173 ± 0.0008**	0.0932 ± 0.0011	0.0731 ± 0.0007
	kernel-CPLST	0.1733 ± 0.0004	0.1470 ± 0.0006	**0.1179 ± 0.0007**	**0.0905 ± 0.0007**	**0.0717 ± 0.0007**
yeast	PBR	0.2320 ± 0.0006	0.2150 ± 0.0007	0.2080 ± 0.0008	0.1913 ± 0.0009	0.1882 ± 0.0009
	kernel-OCCA	0.2308 ± 0.0007	0.2280 ± 0.0010	0.2233 ± 0.0011	0.2052 ± 0.0009	0.1882 ± 0.0009
	PLST	0.2030 ± 0.0008	0.1913 ± 0.0009	0.1892 ± 0.0009	0.1882 ± 0.0009	0.1881 ± 0.0009
	kernel-CPLST	**0.2018 ± 0.0008**	**0.1904 ± 0.0009**	**0.1875 ± 0.0009**	**0.1869 ± 0.0009**	**0.1868 ± 0.0009**

(those within one standard error of the lower one are in bold)

In addition, in the medical and enron datasets, we can see that the overfitting problem is eliminated with regularization (and parameter selection), and hence kernel-CPLST performs ot only better than PLST with kernel ridge regression, but also better than (unregularized) CPLST with linear regression results in Table 4.2.

4.3 Kernel Ridge Regression with a Fixed Parameter Combination

The experimental results in Section 4.2 show that kernel-CPLST outperforms PLST under a systematic parameter selection procedure. However, in the real world, the computation constraints may prevent executing such a procedure. In this section, we conduct experiments using a fixed parameter combination to compare kernel-CPLST and other LSDR approaches more realistically.

The fixed parameter combination that we use is $\lambda = 10^{-3}$ and $\gamma = 10^{-4}$, which is generally within the reasonable range in Table 4.3. The results of the LSDR approaches coupled with kernel ridge regression is reported in Table 4.5. We can see that under this setting, the performances of kernel-CPLST and CPLST are similar, while both of them dominate the other LSDR approaches coupled with kernel ridge regression in four out of the eight datasets (`bibtex`, `emotions`, `scene`, and `yeast`). On the other hand, in remaining datasets (`corel5k`, `enron`, `genbase`, and `medical`), the results of kernel-CPLST, CPLST and PLST are similar and dominate other LSDR approaches. In summary, both kernel-CPLST and CPLST keep performing better than other LSDR approaches when coupled when a fixed-parameter kernel ridge regression.

4.4 Coupling Label Space Dimension Reduction with the M5P Decision Tree

CPLST or kernel-CPLST are designed by assuming a specific regression method of either linear regression or kernel ridge regression. Next, we demonstrate that the input-output relationship captured by CPLST (and other feature-aware FSDR approaches) is not restricted for coupling with linear regression, but can be effective for other regression methods in the learning stage (step 4 of Algorithm 1). We do so by coupling the LSDR approaches with the M5P decision tree [29]. M5P decision tree is a non-linear regression method (different from linear regression) and is quite different from kernel ridge regres-

Table 4.5: Test Hamming Loss of LSDR algorithm with KRR using a specific parameter combination

Dataset	Algorithm	20%	40%	60%	80%	100%
`bibtex`	PBR	0.0146 ± 0.0001	0.0140 ± 0.0001	0.0133 ± 0.0001	0.0128 ± 0.0000	**0.0122 ± 0.0000**
	OCCA	0.0140 ± 0.0000	0.0138 ± 0.0000	0.0135 ± 0.0000	0.0129 ± 0.0000	**0.0122 ± 0.0000**
	PLST	0.0129 ± 0.0000	0.0125 ± 0.0000	0.0124 ± 0.0000	**0.0122 ± 0.0000**	**0.0122 ± 0.0000**
	CPLST	**0.0127 ± 0.0000**	**0.0124 ± 0.0000**	**0.0122 ± 0.0000**	**0.0122 ± 0.0000**	**0.0122 ± 0.0000**
	kernel-OCCA	0.0139 ± 0.0000	0.0138 ± 0.0000	0.0135 ± 0.0000	0.0129 ± 0.0000	**0.0122 ± 0.0000**
	kernel-CPLST	**0.0127 ± 0.0000**	**0.0124 ± 0.0000**	0.0123 ± 0.0000	**0.0122 ± 0.0000**	**0.0122 ± 0.0000**
`corel5k`	PBR	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**
	OCCA	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	**0.0093 ± 0.0000**
	PLST	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**
	CPLST	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**
	kernel-OCCA	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	**0.0093 ± 0.0000**
	kernel-CPLST	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**	**0.0093 ± 0.0000**
`emotions`	PBR	0.2919 ± 0.0020	0.2723 ± 0.0024	0.2528 ± 0.0025	0.2344 ± 0.0024	**0.1980 ± 0.0026**
	OCCA	0.3113 ± 0.0014	0.3000 ± 0.0014	0.2951 ± 0.0017	0.2937 ± 0.0018	**0.1980 ± 0.0026**
	PLST	**0.2217 ± 0.0023**	0.2080 ± 0.0023	**0.1961 ± 0.0025**	**0.1982 ± 0.0023**	**0.1980 ± 0.0026**
	CPLST	**0.2215 ± 0.0021**	**0.2049 ± 0.0024**	**0.1975 ± 0.0025**	**0.1976 ± 0.0025**	**0.1980 ± 0.0026**
	kernel-OCCA	0.3115 ± 0.0015	0.3060 ± 0.0014	0.3023 ± 0.0017	0.2981 ± 0.0016	**0.1980 ± 0.0026**
	kernel-CPLST	**0.2205 ± 0.0021**	**0.2051 ± 0.0023**	**0.1981 ± 0.0025**	**0.1968 ± 0.0025**	**0.1980 ± 0.0026**
`enron`	PBR	0.0613 ± 0.0004	0.0593 ± 0.0005	0.0572 ± 0.0005	0.0552 ± 0.0004	**0.0533 ± 0.0003**
	OCCA	0.0636 ± 0.0002	0.0627 ± 0.0002	0.0637 ± 0.0002	0.0643 ± 0.0002	**0.0533 ± 0.0003**
	PLST	0.0507 ± 0.0002	**0.0524 ± 0.0003**	**0.0530 ± 0.0003**	**0.0533 ± 0.0003**	**0.0533 ± 0.0003**
	CPLST	0.0505 ± 0.0002	**0.0524 ± 0.0003**	**0.0530 ± 0.0003**	**0.0532 ± 0.0003**	**0.0533 ± 0.0003**
	kernel-OCCA	0.0635 ± 0.0002	0.0623 ± 0.0002	0.0634 ± 0.0002	0.0635 ± 0.0002	**0.0533 ± 0.0003**
	kernel-CPLST	**0.0502 ± 0.0002**	**0.0524 ± 0.0003**	**0.0530 ± 0.0003**	**0.0532 ± 0.0003**	**0.0533 ± 0.0003**
`genbase`	PBR	0.0378 ± 0.0005	0.0287 ± 0.0006	0.0189 ± 0.0006	0.0108 ± 0.0005	**0.0016 ± 0.0001**
	OCCA	0.0465 ± 0.0003	0.0402 ± 0.0005	0.0185 ± 0.0003	0.0169 ± 0.0004	**0.0016 ± 0.0001**
	PLST	**0.0175 ± 0.0004**	**0.0045 ± 0.0002**	**0.0020 ± 0.0001**	**0.0017 ± 0.0001**	**0.0016 ± 0.0001**
	CPLST	**0.0174 ± 0.0004**	**0.0046 ± 0.0002**	**0.0020 ± 0.0001**	**0.0017 ± 0.0001**	**0.0016 ± 0.0001**
	kernel-OCCA	0.0415 ± 0.0004	0.0281 ± 0.0004	0.0118 ± 0.0004	0.0075 ± 0.0002	**0.0016 ± 0.0001**
	kernel-CPLST	**0.0175 ± 0.0004**	**0.0046 ± 0.0002**	**0.0019 ± 0.0001**	**0.0017 ± 0.0001**	**0.0016 ± 0.0001**
`medical`	PBR	0.0239 ± 0.0003	0.0207 ± 0.0003	0.0173 ± 0.0003	0.0138 ± 0.0003	**0.0101 ± 0.0002**
	OCCA	0.0275 ± 0.0001	0.0260 ± 0.0001	0.0243 ± 0.0001	0.0226 ± 0.0002	**0.0101 ± 0.0002**
	PLST	**0.0132 ± 0.0002**	**0.0102 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**
	CPLST	**0.0132 ± 0.0002**	**0.0102 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**
	kernel-OCCA	0.0275 ± 0.0001	0.0265 ± 0.0001	0.0256 ± 0.0001	0.0237 ± 0.0001	**0.0101 ± 0.0002**
	kernel-CPLST	**0.0132 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**	**0.0101 ± 0.0002**
`scene`	PBR	0.1665 ± 0.0007	0.1540 ± 0.0008	0.1412 ± 0.0008	0.1283 ± 0.0008	**0.1021 ± 0.0008**
	OCCA	0.1789 ± 0.0003	0.1657 ± 0.0005	0.1494 ± 0.0005	0.1290 ± 0.0005	**0.1021 ± 0.0008**
	PLST	0.1794 ± 0.0004	0.1689 ± 0.0005	0.1519 ± 0.0006	0.1251 ± 0.0009	**0.1021 ± 0.0008**
	CPLST	**0.1737 ± 0.0004**	**0.1506 ± 0.0005**	**0.1302 ± 0.0006**	**0.1155 ± 0.0006**	**0.1021 ± 0.0008**
	kernel-OCCA	0.1789 ± 0.0003	0.1659 ± 0.0005	0.1494 ± 0.0005	0.1291 ± 0.0005	**0.1021 ± 0.0008**
	kernel-CPLST	**0.1740 ± 0.0004**	**0.1510 ± 0.0005**	**0.1301 ± 0.0006**	**0.1157 ± 0.0006**	**0.1021 ± 0.0008**
`yeast`	PBR	0.2277 ± 0.0008	0.2205 ± 0.0009	0.2147 ± 0.0010	0.2070 ± 0.0009	**0.2003 ± 0.0008**
	OCCA	0.2329 ± 0.0006	0.2329 ± 0.0006	0.2365 ± 0.0006	0.2225 ± 0.0007	**0.2003 ± 0.0008**
	PLST	0.2142 ± 0.0008	0.2037 ± 0.0008	0.2018 ± 0.0008	**0.2005 ± 0.0008**	**0.2003 ± 0.0008**
	CPLST	**0.2059 ± 0.0008**	**0.2019 ± 0.0008**	**0.2008 ± 0.0008**	**0.2005 ± 0.0008**	**0.2003 ± 0.0008**
	kernel-OCCA	0.2329 ± 0.0006	0.2329 ± 0.0006	0.2367 ± 0.0006	0.2228 ± 0.0007	**0.2003 ± 0.0008**
	kernel-CPLST	**0.2059 ± 0.0008**	**0.2018 ± 0.0008**	**0.2008 ± 0.0008**	**0.2005 ± 0.0008**	**0.2003 ± 0.0008**

(those within one standard error of the lowest one are in bold)

sion. We take the implementation from WEKA [30] for M5P with the default parameter setting. For kernel-CPLST and kernel-OCCA, we use the best parameters (λ, γ) selected in Section 4.2.

The experimental results are shown in Table 4.6. The relations between PBR, PLST, OCCA and CPLST when coupled with M5P are similar to the ones when coupled with

Table 4.6: Test Hamming loss of LSDR algorithms with M5P

Dataset	Algorithm	20%	40%	60%	80%	100%
`bibtex`	PBR	0.0152 ± 0.0001	0.0152 ± 0.0001	0.0151 ± 0.0001	0.0150 ± 0.0001	0.0149 ± 0.0001
	OCCA	0.0140 ± 0.0001	0.0138 ± 0.0001	0.0137 ± 0.0001	0.0132 ± 0.0001	0.0128 ± 0.0001
	PLST	**0.0130 ± 0.0001**	**0.0128 ± 0.0001***	**0.0128 ± 0.0001**	**0.0127 ± 0.0001***	**0.0127 ± 0.0001***
	CPLST	**0.0129 ± 0.0001***	**0.0128 ± 0.0001***	**0.0127 ± 0.0001***	**0.0127 ± 0.0001***	**0.0127 ± 0.0001***
	kernel-OCCA	0.0140 ± 0.0000	0.0137 ± 0.0000	0.0133 ± 0.0000	0.0132 ± 0.0000	**0.0127 ± 0.0000***
	kernel-CPLST	**0.0130 ± 0.0000**	**0.0128 ± 0.0000***	**0.0128 ± 0.0000**	**0.0127 ± 0.0000***	**0.0127 ± 0.0000***
`corel5k`	PBR	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
	OCCA	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
	PLST	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
	CPLST	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
	kernel-OCCA	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
	kernel-CPLST	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***	**0.0094 ± 0.0000***
`emotions`	PBR	0.3118 ± 0.0036	0.2832 ± 0.0034	0.2600 ± 0.0032	0.2512 ± 0.0032	0.2239 ± 0.0033
	OCCA	0.3103 ± 0.0034	0.2974 ± 0.0033	0.2932 ± 0.0033	0.2995 ± 0.0035	0.2073 ± 0.0031
	PLST	**0.2213 ± 0.0030**	**0.2109 ± 0.0030**	0.2039 ± 0.0029	0.2051 ± 0.0029	**0.2063 ± 0.0030**
	CPLST	**0.2209 ± 0.0031***	**0.2085 ± 0.0032***	**0.2004 ± 0.0031***	**0.2020 ± 0.0031**	**0.2046 ± 0.0031***
	kernel-OCCA	0.3111 ± 0.0034	0.3008 ± 0.0033	0.3005 ± 0.0034	0.3011 ± 0.0034	0.2080 ± 0.0031
	kernel-CPLST	**0.2212 ± 0.0032**	**0.2103 ± 0.0032**	**0.2004 ± 0.0030***	**0.2016 ± 0.0031***	**0.2047 ± 0.0030**
`enron`	PBR	0.0561 ± 0.0002	0.0552 ± 0.0002	0.0504 ± 0.0002	0.0514 ± 0.0003	0.0516 ± 0.0003
	OCCA	0.0637 ± 0.0002	0.0628 ± 0.0002	0.0635 ± 0.0002	0.0626 ± 0.0002	**0.0489 ± 0.0002**
	PLST	**0.0490 ± 0.0002**	**0.0488 ± 0.0002***	**0.0489 ± 0.0002***	**0.0490 ± 0.0002***	**0.0490 ± 0.0002***
	CPLST	**0.0489 ± 0.0003**	**0.0489 ± 0.0003**	**0.0490 ± 0.0003**	**0.0490 ± 0.0003***	**0.0490 ± 0.0003***
	kernel-OCCA	0.0635 ± 0.0002	0.0630 ± 0.0002	0.0629 ± 0.0002	0.0623 ± 0.0002	**0.0489 ± 0.0002**
	kernel-CPLST	**0.0488 ± 0.0003***	**0.0489 ± 0.0003**	**0.0490 ± 0.0003**	**0.0490 ± 0.0003***	**0.0491 ± 0.0003**
`genbase`	PBR	0.0359 ± 0.0009	0.0315 ± 0.0006	0.0255 ± 0.0004	0.0243 ± 0.0004	0.0215 ± 0.0004
	OCCA	0.0465 ± 0.0003	0.0448 ± 0.0004	0.0327 ± 0.0004	0.0311 ± 0.0004	0.0212 ± 0.0004
	PLST	**0.0215 ± 0.0004***	**0.0202 ± 0.0004***	**0.0195 ± 0.0003***	**0.0194 ± 0.0003***	**0.0194 ± 0.0003***
	CPLST	**0.0215 ± 0.0004***	**0.0202 ± 0.0004***	**0.0195 ± 0.0003***	**0.0195 ± 0.0003**	**0.0195 ± 0.0003**
	kernel-OCCA	0.0462 ± 0.0004	0.0450 ± 0.0005	0.0427 ± 0.0006	0.0362 ± 0.0007	0.0204 ± 0.0004
	kernel-CPLST	**0.0215 ± 0.0004***	**0.0202 ± 0.0004***	**0.0195 ± 0.0003***	**0.0195 ± 0.0003**	**0.0195 ± 0.0003**
`medical`	PBR	0.0244 ± 0.0001	0.0168 ± 0.0002	0.0132 ± 0.0002	0.0120 ± 0.0002	0.0108 ± 0.0002
	OCCA	0.0275 ± 0.0001	0.0260 ± 0.0001	0.0245 ± 0.0001	0.0225 ± 0.0002	0.0101 ± 0.0002
	PLST	**0.0127 ± 0.0002**	**0.0099 ± 0.0002**	**0.0097 ± 0.0002**	**0.0097 ± 0.0002**	**0.0097 ± 0.0002**
	CPLST	**0.0126 ± 0.0002***	**0.0099 ± 0.0002**	**0.0096 ± 0.0002***	**0.0096 ± 0.0002***	**0.0096 ± 0.0002***
	kernel-OCCA	0.0274 ± 0.0001	0.0258 ± 0.0001	0.0251 ± 0.0001	0.0226 ± 0.0002	0.0100 ± 0.0002
	kernel-CPLST	**0.0126 ± 0.0002***	**0.0098 ± 0.0002***	**0.0097 ± 0.0002**	**0.0097 ± 0.0002**	**0.0097 ± 0.0002**
`scene`	PBR	**0.1665 ± 0.0005***	0.1650 ± 0.0006	0.1649 ± 0.0008	0.1565 ± 0.0008	0.1369 ± 0.0009
	OCCA	0.1789 ± 0.0003	0.1658 ± 0.0005	0.1501 ± 0.0006	0.1437 ± 0.0006	0.1296 ± 0.0009
	PLST	0.1802 ± 0.0005	0.1688 ± 0.0007	0.1540 ± 0.0008	0.1396 ± 0.0011	0.1281 ± 0.0008
	CPLST	0.1674 ± 0.0005	**0.1538 ± 0.0006***	**0.1428 ± 0.0007***	**0.1289 ± 0.0007***	**0.1268 ± 0.0008***
	kernel-OCCA	0.1789 ± 0.0003	0.1656 ± 0.0005	0.1525 ± 0.0006	0.1437 ± 0.0007	0.1306 ± 0.0008
	kernel-CPLST	0.1837 ± 0.0005	0.1667 ± 0.0007	0.1525 ± 0.0009	0.1370 ± 0.0008	0.1285 ± 0.0009
`yeast`	PBR	0.2379 ± 0.0006	0.2328 ± 0.0006	0.2381 ± 0.0008	0.2351 ± 0.0008	0.2384 ± 0.0008
	OCCA	0.2329 ± 0.0006	0.2330 ± 0.0006	0.2339 ± 0.0006	0.2225 ± 0.0008	**0.2055 ± 0.0009**
	PLST	0.2162 ± 0.0008	0.2082 ± 0.0009	**0.2071 ± 0.0009**	**0.2064 ± 0.0009***	0.2067 ± 0.0009
	CPLST	**0.2083 ± 0.0009***	**0.2064 ± 0.0009***	**0.2063 ± 0.0009***	**0.2064 ± 0.0009***	0.2066 ± 0.0009
	kernel-OCCA	0.2329 ± 0.0006	0.2329 ± 0.0006	0.2319 ± 0.0006	0.2215 ± 0.0007	**0.2050 ± 0.0009***
	kernel-CPLST	0.2162 ± 0.0008	0.2080 ± 0.0009	**0.2070 ± 0.0009**	**0.2064 ± 0.0009***	0.2066 ± 0.0009

(those with the lowest mean are marked with *; those within one standard error of the lowest one are in bold)

linear regression. In the `enron`, `genbase`, and `medical` datasets, the results of CPLST and kernel-CPLST are similar to the result of PLST and all of the three dominate other LSDR algorithms. In particular, in the `yeast`, `scene`, and `emotions` datasets, the result of CPLST outperforms the result of other LSDR algorithms. This result demonstrates that the captured input-output relation is also effective for regression methods other than linear regression.

Another thing need to notice is that in the `scene` and `yeast` datasets, the performance of kernel-CPLST is usually between CPLST and PLST and not as good as CPLST because the Gaussian kernel is a sophisticated kernel mapping and is hard to be fitted by M5P (which is very different from kernel ridge regression); therefore, M5P cannot fit the captured relation between the input and the output by kernel-CPLST. That is, when coupled with other regression methods which are different from kernel ridge regression, such as M5P, using CPLST as the FSDR approach is more promising. An interesting further work may be to study the kernel (may be not so complicated as the Gaussian kernel) that can capture input-output relation to be reused by various underlying regression methods.

One more thing is that in all datasets, the performance of these FSDR approaches is better than a full binary relevance. This result demonstrates the benefit of LSDR for improving performance through capturing key information.

In summary, the experimental results justify that CPLST is the most effective approach to LSDR even when coupled with M5P decision tree.

4.5 Comparison CPLST and kernel-CPLST

In this section, we conduct experiments for comparing CPLST and kernel-CPLST in more detail. Table 4.7 shows the test Hamming loss of four algorithm combinations, CPLST coupled with linear regression, kernel-CPLST coupled with linear regression, CPLST coupled kernel ridge regression, and kernel-CPLST coupled with kernel ridge regression, and Table 4.8 shows the training Hamming loss of those four algorithm combinations. For kernel-CPLST coupled with linear regression and kernel-CPLST coupled with kernel ridge regression, we use the best parameter (λ, γ) selected in Section 4.2. For CPLST coupled with kernel ridge regression, we follow the procedure in Section 4.2 to select the best parameters (λ, γ) for itself.

In Table 4.7, the performance of these FSDR approaches coupled with kernel ridge regression is better than the ones coupled with linear regression in most datasets. The results demonstrate that the underlying regression method is still important.

Table 4.7: Test Hamming Loss

Dataset	Algorithm	20%	40%	60%	80%	100%
`bibtex`	CPLST + LR	0.0127 ± 0.0000	0.0124 ± 0.0000	0.0123 ± 0.0000	0.0123 ± 0.0000	0.0123 ± 0.0000
	kernel-CPLST + LR	0.0128 ± 0.0000	0.0125 ± 0.0000	0.0124 ± 0.0000	0.0123 ± 0.0000	0.0123 ± 0.0000
	CPLST + KRR	**0.0125 ± 0.0000**	**0.0122 ± 0.0000**	**0.0120 ± 0.0000**	**0.0120 ± 0.0000**	**0.0119 ± 0.0000**
	kernel-CPLST + KRR	0.0127 ± 0.0000	0.0123 ± 0.0000	0.0121 ± 0.0000	**0.0120 ± 0.0000**	0.0120 ± 0.0000
`corel5k`	CPLST + LR	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000
	kernel-CPLST + LR	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000	0.0094 ± 0.0000
	CPLST + KRR	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**
	kernel-CPLST + KRR	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**	**0.0092 ± 0.0000**
`emotions`	CPLST + LR	**0.2189 ± 0.0019**	**0.2059 ± 0.0022**	**0.1990 ± 0.0022**	0.2015 ± 0.0021	0.2040 ± 0.0022
	kernel-CPLST + LR	**0.2184 ± 0.0020**	**0.2067 ± 0.0022**	**0.1988 ± 0.0023**	0.1988 ± 0.0023	0.2040 ± 0.0022
	CPLST + KRR	0.2238 ± 0.0020	**0.2069 ± 0.0024**	**0.1981 ± 0.0024**	**0.1979 ± 0.0026**	**0.1986 ± 0.0026**
	kernel-CPLST + KRR	0.2231 ± 0.0020	**0.2071 ± 0.0024**	**0.1981 ± 0.0025**	**0.1973 ± 0.0027**	**0.1988 ± 0.0027**
`enron`	CPLST + LR	0.0729 ± 0.0004	0.0864 ± 0.0005	0.0943 ± 0.0006	0.1006 ± 0.0007	0.1028 ± 0.0007
	kernel-CPLST + LR	0.0718 ± 0.0004	0.0862 ± 0.0005	0.0939 ± 0.0006	0.1004 ± 0.0007	0.1028 ± 0.0007
	CPLST + KRR	**0.0454 ± 0.0002**	**0.0453 ± 0.0002**	**0.0454 ± 0.0002**	**0.0455 ± 0.0002**	**0.0456 ± 0.0002**
	kernel-CPLST + KRR	**0.0453 ± 0.0002**	**0.0454 ± 0.0002**	**0.0455 ± 0.0002**	**0.0455 ± 0.0002**	**0.0456 ± 0.0002**
`genbase`	CPLST + LR	**0.0168 ± 0.0004**	**0.0041 ± 0.0002**	**0.0012 ± 0.0001**	**0.0008 ± 0.0001**	**0.0007 ± 0.0001**
	kernel-CPLST + LR	**0.0168 ± 0.0004**	**0.0040 ± 0.0002**	**0.0013 ± 0.0001**	**0.0008 ± 0.0001**	**0.0007 ± 0.0001**
	CPLST + KRR	**0.0168 ± 0.0004**	**0.0041 ± 0.0002**	**0.0013 ± 0.0001**	**0.0009 ± 0.0001**	**0.0008 ± 0.0001**
	kernel-CPLST + KRR	**0.0172 ± 0.0004**	**0.0040 ± 0.0002**	**0.0013 ± 0.0001**	**0.0009 ± 0.0001**	**0.0008 ± 0.0001**
`medical`	CPLST + LR	0.0346 ± 0.0004	0.0406 ± 0.0005	0.0471 ± 0.0005	0.0490 ± 0.0005	0.0497 ± 0.0006
	kernel-CPLST + LR	0.0345 ± 0.0004	0.0403 ± 0.0005	0.0471 ± 0.0005	0.0481 ± 0.0006	0.0497 ± 0.0006
	CPLST + KRR	**0.0130 ± 0.0002**	**0.0098 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**
	kernel-CPLST + KRR	**0.0131 ± 0.0002**	**0.0098 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**	**0.0096 ± 0.0002**
`scene`	CPLST + LR	0.1744 ± 0.0004	0.1532 ± 0.0005	0.1349 ± 0.0005	0.1209 ± 0.0007	0.1106 ± 0.0008
	kernel-CPLST + LR	0.1776 ± 0.0004	0.1674 ± 0.0006	0.1466 ± 0.0008	0.1279 ± 0.0007	0.1106 ± 0.0008
	CPLST + KRR	**0.1720 ± 0.0004**	**0.1364 ± 0.0006**	**0.1111 ± 0.0006**	**0.0846 ± 0.0006**	**0.0722 ± 0.0007**
	kernel-CPLST + KRR	0.1733 ± 0.0004	0.1470 ± 0.0006	0.1179 ± 0.0007	0.0905 ± 0.0007	**0.0717 ± 0.0007**
`yeast`	CPLST + LR	0.2069 ± 0.0008	0.2041 ± 0.0009	0.2024 ± 0.0009	0.2020 ± 0.0009	0.2022 ± 0.0009
	kernel-CPLST + LR	0.2139 ± 0.0008	0.2052 ± 0.0009	0.2032 ± 0.0009	0.2020 ± 0.0009	0.2022 ± 0.0009
	CPLST + KRR	**0.1968 ± 0.0008**	**0.1888 ± 0.0009**	**0.1871 ± 0.0009**	**0.1871 ± 0.0009**	**0.1870 ± 0.0009**
	kernel-CPLST + KRR	0.2018 ± 0.0008	0.1904 ± 0.0009	**0.1875 ± 0.0009**	**0.1869 ± 0.0009**	**0.1868 ± 0.0009**

(those within one standard error of the lowest one are in bold)

On the other hand, in terms of the LSDR approaches, the performance of CPLST is usually at least as effective as kernel-CPLST. In particular, in `scene` and `yeast`, CPLST is better than kernel-CPLST regardless of whether it is coupled with linear regression or kernel ridge regression. If we look the training performance (Table 4.8), it shows that the training Hamming loss of kernel-CPLST is better than the one of CPLST in some cases including the `bibtex` and `yeast`, but can be overfitting (worse test Hamming loss). The reason may be because kernel-CPLST is so complicated and thus it incur more risk of overfitting. In other words, for simple datasets, using kernel-CPLST may be risky; for more complicated ones; however, kernel-CPSLT can still be useful.

On the basis of Section 4.4 and this section, when we want to choose the suitable one from these LSDR approaches, CPLST may be the first choice because the captured input-output relations is more easily utilized by various underlying regression methods

Table 4.8: Training Hamming Loss

Dataset	Algorithm	20%	40%	60%	80%	100%
`bibtex`	CPLST + LR	0.0114 ± 0.0000	0.0100 ± 0.0000	0.0091 ± 0.0000	0.0085 ± 0.0000	0.0080 ± 0.0000
	kernel-CPLST + LR	0.0114 ± 0.0000	0.0100 ± 0.0000	0.0093 ± 0.0000	0.0086 ± 0.0000	0.0080 ± 0.0000
	CPLST + KRR	0.0094 ± 0.0000	0.0061 ± 0.0000	**0.0038 ± 0.0000**	**0.0021 ± 0.0000**	**0.0015 ± 0.0000**
	kernel-CPLST + KRR	**0.0090 ± 0.0000**	**0.0059 ± 0.0000**	0.0039 ± 0.0000	0.0022 ± 0.0000	**0.0015 ± 0.0000**
`corel5k`	CPLST + LR	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000
	kernel-CPLST + LR	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000	0.0089 ± 0.0000
	CPLST + KRR	**0.0023 ± 0.0000**	**0.0010 ± 0.0000**	**0.0004 ± 0.0000**	**0.0001 ± 0.0000**	**0.0000 ± 0.0000**
	kernel-CPLST + KRR	**0.0021 ± 0.0001**	**0.0009 ± 0.0001**	**0.0004 ± 0.0002**	**0.0001 ± 0.0002**	**0.0000 ± 0.0000**
`emotions`	CPLST + LR	0.2019 ± 0.0003	0.1728 ± 0.0003	**0.1574 ± 0.0003**	**0.1508 ± 0.0004**	**0.1444 ± 0.0003**
	kernel-CPLST + LR	**0.2016 ± 0.0003**	**0.1721 ± 0.0003**	**0.1575 ± 0.0003**	**0.1508 ± 0.0003**	**0.1444 ± 0.0003**
	CPLST + KRR	0.2082 ± 0.0006	0.1820 ± 0.0006	0.1670 ± 0.0007	0.1624 ± 0.0008	0.1577 ± 0.0009
	kernel-CPLST + KRR	0.2083 ± 0.0002	0.1811 ± 0.0005	0.1667 ± 0.0007	0.1622 ± 0.0008	0.1578 ± 0.0013
`enron`	CPLST + LR	**0.0203 ± 0.0000**	**0.0121 ± 0.0000**	**0.0097 ± 0.0000**	**0.0081 ± 0.0000**	**0.0076 ± 0.0000**
	kernel-CPLST + LR	0.0208 ± 0.0000	0.0122 ± 0.0000	**0.0097 ± 0.0000**	**0.0081 ± 0.0000**	**0.0076 ± 0.0000**
	CPLST + KRR	0.0282 ± 0.0002	0.0237 ± 0.0002	0.0223 ± 0.0002	0.0217 ± 0.0002	0.0215 ± 0.0002
	kernel-CPLST + KRR	0.0283 ± 0.0001	0.0239 ± 0.0001	0.0225 ± 0.0002	0.0220 ± 0.0006	0.0218 ± 0.0006
`genbase`	CPLST + LR	**0.0155 ± 0.0001**	**0.0033 ± 0.0000**	**0.0009 ± 0.0000**	**0.0003 ± 0.0000**	0.0002 ± 0.0000
	kernel-CPLST + LR	**0.0155 ± 0.0001**	**0.0033 ± 0.0000**	**0.0009 ± 0.0000**	**0.0003 ± 0.0000**	0.0002 ± 0.0000
	CPLST + KRR	**0.0155 ± 0.0001**	**0.0032 ± 0.0000**	**0.0009 ± 0.0000**	**0.0003 ± 0.0000**	0.0002 ± 0.0000
	kernel-CPLST + KRR	0.0158 ± 0.0002	**0.0032 ± 0.0001**	**0.0008 ± 0.0000**	**0.0003 ± 0.0000**	**0.0001 ± 0.0000**
`medical`	CPLST + LR	0.0071 ± 0.0000	**0.0022 ± 0.0000**	**0.0008 ± 0.0000**	**0.0004 ± 0.0000**	**0.0002 ± 0.0000**
	kernel-CPLST + LR	**0.0070 ± 0.0000**	**0.0022 ± 0.0000**	**0.0008 ± 0.0000**	**0.0004 ± 0.0000**	**0.0002 ± 0.0000**
	CPLST + KRR	0.0074 ± 0.0000	0.0026 ± 0.0000	0.0013 ± 0.0000	0.0009 ± 0.0001	0.0007 ± 0.0001
	kernel-CPLST + KRR	0.0073 ± 0.0001	0.0026 ± 0.0002	0.0013 ± 0.0003	0.0009 ± 0.0003	0.0007 ± 0.0000
`scene`	CPLST + LR	0.1732 ± 0.0002	0.1457 ± 0.0001	0.1198 ± 0.0001	0.0970 ± 0.0001	0.0758 ± 0.0001
	kernel-CPLST + LR	0.1739 ± 0.0002	0.1577 ± 0.0002	0.1283 ± 0.0003	0.1003 ± 0.0002	0.0758 ± 0.0001
	CPLST + KRR	**0.1585 ± 0.0004**	**0.1142 ± 0.0007**	0.0861 ± 0.0006	**0.0301 ± 0.0009**	**0.0058 ± 0.0005**
	kernel-CPLST + KRR	0.1643 ± 0.0011	0.1204 ± 0.0012	**0.0822 ± 0.0012**	0.0408 ± 0.0007	0.0150 ± 0.0000
`yeast`	CPLST + LR	0.1970 ± 0.0001	0.1886 ± 0.0001	0.1855 ± 0.0001	0.1848 ± 0.0001	0.1848 ± 0.0001
	kernel-CPLST + LR	0.2034 ± 0.0001	0.1895 ± 0.0001	0.1859 ± 0.0001	0.1848 ± 0.0001	0.1848 ± 0.0001
	CPLST + KRR	0.1270 ± 0.0006	0.0422 ± 0.0004	0.0114 ± 0.0002	**0.0016 ± 0.0000**	**0.0002 ± 0.0000**
	kernel-CPLST + KRR	**0.1147 ± 0.0017**	**0.0368 ± 0.0036**	**0.0062 ± 0.0043**	**0.0016 ± 0.0026**	**0.0002 ± 0.0000**

(those within one standard error of the lowest one are in bold)

than kernel-CPLST. Even when coupling with linear regression or kernel ridge regression as underlying regression method, kernel-CPLST takes more risk to overfit. However, if we want to improve the performance further and the major constraints is on the time of prediction or the storage of model (Section 2.2), checking the performance of kernel-CPLST would be a good option.

4.6 Optimal Reduced Size

In this section, we conduct experiment to check the optimal reduced size M^*. The optimal reduced size M^* is defined as the minimum number of dimension at which the Hamming loss difference between full binary relevance and the LSDR approach is within its respective standard errors. In other words, this shows the size of needed label space dimension at which the performance does not significantly degrade. The optimal reduced size of label by LSDR algorithm coupled with linear regression and kernel ridge regression are reported in Table 4.9 and Table 4.10 respectively.

Table 4.9: Optimal reduced size M^* of LSDR algorithm when coupled with linear regression

Dataset	# Labels (K)	PLST	CPLST	kernel-CPLST
`bibtex`	159	**1 (0.0143 ± 0.0007)**	**1 (0.0143 ± 0.0007)**	**1 (0.0144 ± 0.0007)**
`corel5k`	374	33 (0.0094 ± 0.0000)	**17 (0.0094 ± 0.0000)**	33 (0.0094 ± 0.0000)
`emotions`	6	3 (0.1928 ± 0.0022)	**2 (0.2059 ± 0.0022)**	3 (0.1988 ± 0.0023)
`enron`	53	**1 (0.0639 ± 0.0003)**	**1 (0.0640 ± 0.0003)**	**1 (0.0637 ± 0.0003)**
`genbase`	27	23 (0.0007 ± 0.0001)	**22 (0.0007 ± 0.0001)**	**22 (0.0008 ± 0.0001)**
`medical`	45	**1 (0.0303 ± 0.0003)**	**1 (0.0303 ± 0.0003)**	**1 (0.0303 ± 0.0003)**
`scene`	6	**5 (0.1094 ± 0.0009)**	**5 (0.1097 ± 0.0009)**	**5 (0.1095 ± 0.0009)**
`yeast`	14	9 (0.2024 ± 0.0009)	**8 (0.2024 ± 0.0009)**	9 (0.2024 ± 0.0009)

Table 4.10: Optimal reduced size M^* of LSDR algorithm when coupled with kernel ridge regression

Dataset	# Labels (K)	PLST	CPLST	kernel-CPLST
`bibtex`	159	151 (0.0151 ± 0.0000)	**31 (0.0125 ± 0.0000)**	42 (0.0125 ± 0.0000)
`corel5k`	374	**1 (0.0094 ± 0.0000)**	**1 (0.0094 ± 0.0000)**	**1 (0.0094 ± 0.0000)**
`emotions`	6	**2 (0.2074 ± 0.0023)**	**2 (0.2071 ± 0.0024)**	**2 (0.2069 ± 0.0024)**
`enron`	53	4 (0.0471 ± 0.0002)	**3 (0.0473 ± 0.0002)**	**3 (0.0471 ± 0.0002)**
`genbase`	27	23 (0.0008 ± 0.0001)	**22 (0.0008 ± 0.0001)**	**22 (0.0008 ± 0.0001)**
`medical`	45	9 (0.0136 ± 0.0002)	**8 (0.0135 ± 0.0002)**	**8 (0.0135 ± 0.0002)**
`scene`	6	5 (0.0723 ± 0.0007)	**4 (0.0846 ± 0.0006)**	5 (0.0715 ± 0.0007)
`yeast`	14	9 (0.1885 ± 0.0009)	**5 (0.1888 ± 0.0009)**	7 (0.1884 ± 0.0009)

Table 4.11: Time of LSDR algorithm when coupled with kernel ridge regression at optimal reduced size (sec)

Dataset	PLST		CPLST		kernel-CPLST	
	FSDR	regression	FSDR	regression	FSDR	regression
`bibtex`	0.256	245797.61	14.321	**2089.68**	67.81	2930.69
`corel5k`	0.106	21.778	1.119	26.674	22.328	**21.538**
`emotions`	0.002	0.140	0.007	**0.129**	0.048	0.135
`enron`	0.020	4.091	1.147	2.765	1.032	**2.617**
`genbase`	0.001	3.074	0.287	2.667	0.119	**2.656**
`medical`	0.003	2.908	0.814	2.470	0.266	**2.463**
`scene`	0.002	10.376	0.157	**8.518**	1.688	10.022
`yeast`	0.002	17.209	0.033	**10.321**	1.701	13.756

Table 4.9 shows the optimal reduced size of PLST, CPLST, and kernel-CPLST when coupled with linear regression. In four out of the eight datasets (`corel5k`, `emotions`, `genbase`, and `yeast`), CPLST can use fewer dimension to get similar performance.

Table 4.10 shows the optimal reduced size of PLST, CPLST, and kernel-CPLST when coupled with kernel ridge regression and Table 4.11 shows the corresponding training time and FSDR time which contains encoding time and decoding time. The computational time are measured on Intel Xeon E5530 2.4G Processor with 8192KB cache size. All algorithms including FSDR algorithms and KRR are implemented by MATLAB version 7.14.0.739 (R2012a). We can found that CPLST can use fewer dimension to get similar performance and reduce the training time ("regression" column in table 4.11). In most of the dataset, CPLST and kernel-CPLST are better than PLST. Even in those datasets which the M^* of CPLST and kernel-CPLST are equal to the one of PLST, the Hamming loss of CPLST is still better than the one of PLST. The relation between CPLST and kernel-CPLST is still similar to the previous analysis 4.5. Kernel-CPLST is between CPLST and PLST and usually slight worse than CPLST. In conclusion, this result verifies that CPLST (including kernel-CPLST) is better than PLST and can further decrease the number of need dimension.

Table 4.12: Time of LSDR algorithm when coupled with kernel ridge regression at optimal reduced size (sec)

Dataset		PLST			CPLST			kernel-CPLST	
	FSDR	regression	Total	FSDR	regression	Total	FSDR	regression	Total
`bibtex`	0.256	1538.749	1539.005	14.321	78.882	93.203	74.561	0.907	**75.468**
`corel5k`	0.106	21.911	**22.016**	1.119	21.570	22.689	22.328	0.242	22.570
`emotions`	0.002	0.065	0.067	0.007	0.053	0.060	0.048	0.005	**0.053**
`enron`	0.020	1.140	1.160	1.147	0.900	2.047	1.032	0.051	**1.083**
`genbase`	0.001	0.152	0.153	0.287	0.159	0.446	0.119	0.023	**0.142**
`medical`	0.003	0.300	0.303	0.814	0.266	1.080	0.266	0.028	**0.294**
`scene`	0.002	1.745	1.747	0.157	1.660	1.817	1.688	0.050	**1.738**
`yeast`	0.002	1.713	**1.715**	0.033	1.699	1.732	1.701	0.025	1.726

When coupling with kernel ridge regression with same kernel and regularization parameter for all labels, kernel ridge regression can reuse the inversion of kernel matrix which is calculated by kernel-CPLST. Next, we compare the computational time when considering reusing the inversion of kernel matrix in Table 4.12. From this table, because the most computational expensive part is the inversion of kernel matrix, the consumed time of all FSDR methods are similar. In this case, CPLST and kenel-CPLST are still useful for saving computational time during prediction phase and reducing the storage space of models. Notice that reusing the inversion of kernel matrix only holds when coupling with kernel ridge regression with same kernel and regrularization parameter.

4.7 Determine Optimal Reduced Size M

For determining the optimal reduced size M, we check the eigenvalue of CPLST. The eigenvalues that CPLST obtains from solving the optimization problem (3.6) is related to the amount of training hamming loss bound (2.1). By these eigenvalues, the objective value of (3.6) can be easily calculated by $\|\mathbf{Z}\|_F^2 - \sum_i^M e_i$, where e_i is the i-th largest eigenvalue. In this section, we analyze the eigenvalue, hamming loss bound and hamming loss of CPLST.

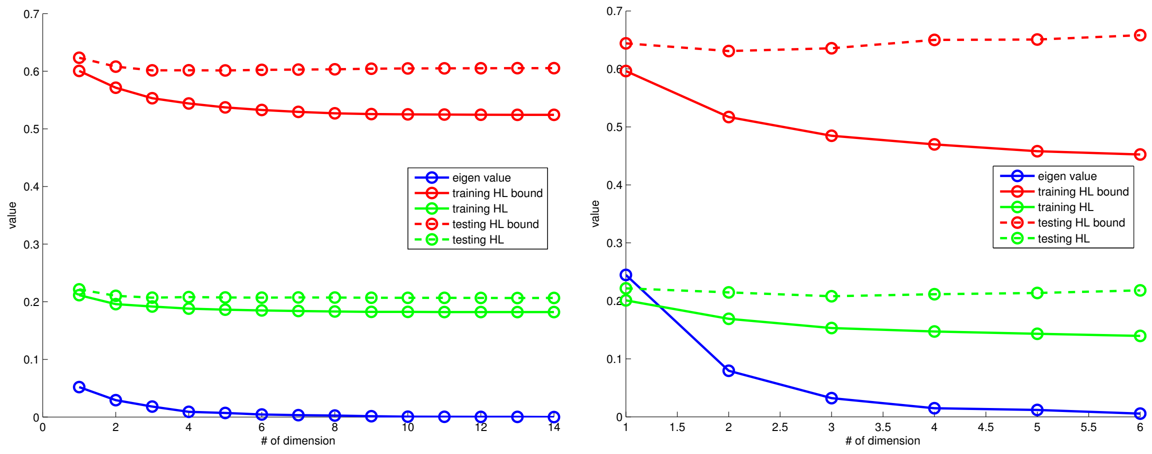
Figure 4.3 shows the eigenvalue, hamming loss bound and hamming loss of CPLST in `yeast` and `emotions` dataset when coupling with linear regression. The red lines mean the hamming loss and the green lines mean the hamming loss bound. The blue lines mean the eigenvalue. The solid lines mean the training result and the dashed lines mean the testing result.

From Figure 4.3, we can found that there is a gap between the hamming loss bound and the actual hamming loss. However, the hamming loss bound and hamming loss share similar trend. This verifies that we can minimize the hamming loss by minimizing the hamming loss bound though there is a gap between them.

In the `yeast` dataset, the training hamming loss bound and the testing hamming loss bound share similar trend. As the training hamming loss bound decreases, the testing hamming loss bound and the hamming loss also decrease. Then, we may use the eigenvalues for estimating the training hamming loss bound and decide how many dimensions we should use.

On the other hand, in the `emotions` dataset, the training hamming loss bound and the testing hamming loss bound are somewhat different in the trend. It is thus difficult to use the eigenvalues in this case.

In conclusion, from the figures, it is still difficult to determine the reduces size by the eigenvalues. One possible solution maybe using validation set or cross validation.



(a) yeast (b) emotions

Figure 4.3: Hamming loss bound and Hamming loss of CPLST in `yeast` and `emotions`

4.8 Discussions and Analysis

In this section, we discuss about the reasons that CPLST is better than other related algorithms in three way, dimension reduction, optimization, and output coding method.

From the view of dimension reduction, PLST can be regarded as a feature-unaware dimension reduction on label space. That is, like unsupervised dimension reduction on features space, PLST projects the original label into new lower dimension subspace without utilizing the feature information. The lack of awareness on the feature information makes PLST inferior. CPLST utilizes the feature information and captures the input-output relation with a first-order method: linear regression. CPLST echoes many previous works (in FSDR) for capturing input-output relationship with linear regression [31, 32, 33] and therefore achieves better performance.

From the view of optimization, when dealing with minimizing the bound of Hamming loss, PLST can be split into two stages. The first stage minimizes the encoding error term by singular value decomposition. Then, the second stage minimizes the squared prediction error by an underlying regression method. However, in terms of optimization, it can be found that this two stages minimization may not guarantee to minimize the whole bound (2.1). For example, after projection, it may become more "difficult" to learn multi-dimension regression and the squared prediction error cannot be reduced. On the other hand, CPLST minimizes the whole bound of Hamming Loss directly; therefore, CPLST certainly gets better result in the optimization problem.

From the view of output coding method, the previous work of Ferng and Lin [34] demonstrates the trade-off between the strength of Error Correcting Code (ECC) and the difficulty of base learning problems. Stronger ECC, it can correct more bit errors, but may leave more difficult learning problems. On the other hand, with weaker may only be able to correct few bit errors, but will have simpler learning problems. PLST can be regarded as a good output coding method that takes the more efficient encoding to the extreme; nevertheless, following ECC, it may need to face "difficult" regression problems in the learning step. On the other hand, OCCA manages to minimize the squared prediction error as mentioned in Section 2.2.3. That is, OCCA aims at finding the transformation whose projected result $\mathbf{ZV}$ is "easiest" to learn by linear regression, but it may lead to worse encoding method. CPLST can be view as a method to find the best balanced combination between the strength of encoding and the hardness of learning task. Through this balance situation, CPLST outperforms PLST which focuses on the strength of encoding and OCCA which focuses on the difficulty of learning task.

Chapter 5

Conclusions

In this thesis, we studied feature-aware label space dimension reduction (LSDR) approaches, which utilize the feature information during LSDR and can be viewed as the counterpart of supervised feature space dimension reduction. We reviewed the precursors of LSDR, Compressive Sensing and Principal Label Space Transformation (PLST), and showed that they can be viewed as feature-unaware LSDR approaches. Then, we demonstrated that canonical correlation analysis (CCA), which transitionally is used as a supervised FSDR approach can also be viewed as an approach to feature-aware LSDR. Base on CCA, we derived a preliminary algorithm, orthogonally constrained CCA (OCCA), which not only preserves the original objective function of CCA but also can apply the efficient round-based decoding to. Then, we proposed a novel feature-aware LSDR algorithm, conditional principal label space transformation (CPLST) which utilizes the key conditional correlations for dimension reduction. Like PLST, CPLST enjoys the theoretical guarantee in balancing between the prediction error and the encoding error in minimizing the Hamming loss bound. In addition, we extended CPLST to a kernelized version for capturing more sophisticated relations between features and labels. We conducted experiments for comparing CPLST and its kernelized version with other LSDR approaches. The experimental results demonstrated that CPLST is the best among the LSDR approaches when coupled with linear regression or kernel ridge regression. In particular, CPLST is better than its feature-unaware precursor, PLST. Moreover, the input-output relation captured by CPLST can be utilized by regression method other than linear regression.

Appendix A

Speed up Experiment of Kernel Ridge Regression

In usual repeated experiment setting of kernel ridge regression, we randomly split dataset into training part and testing part, perform cross validation for λ on each training part, and train kernel ridge regression on training part using the best parameter according to the result cross validation. In this appendix, we show the experiment of kernel ridge regression under this setting can be speed up by linear algebra.

A.1 Setup

As mentioned in section 3.2, kernel ridge regression considers a feature mapping $\Phi : \mathcal{X} \to \mathcal{F}$ before performing regularized linear regression. According to the feature mapping Φ, the kernel function $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T\Phi(\mathbf{x}')$ is defined as an inner product of $\mathbf{x}$ and $\mathbf{x}'$ after mapped into the space $\mathcal{F}$. On the other hand, kernel ridge regression also adds a ℓ_2 regularization term $\frac{1}{2}\lambda\|\mathbf{w}\|^2$, where λ is the regularization parameter, into the objective function for regularizing the power of the high-dimensional linear regression.

That is, when given the dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where y_n is the target value of the input vector, the task of kernel ridge regression is to find an optimal $\mathbf{w}$ in the space $\mathcal{F}$ such that w minimizes

$$\sum_{n=1}^{N}(\mathbf{w}^T\Phi(\mathbf{x}_n) - y_n)^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2. \tag{A.1}$$

Equation A.1 can be rewritten in matrix form as

$$\|\mathbf{\Phi}\mathbf{w} - \mathbf{y}\|^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2, \tag{A.2}$$

where $\mathbf{y}$ is an N by 1 vector whose n-th element is y_n, $\mathbf{\Phi}$ is the matrix containing $\Phi(\mathbf{x}_n)^T$ as rows.

In particular, if $\Phi(\mathbf{x})$ can be explicitly computed, it is known that the closed-form solution of (A.2) is [25]

$$\mathbf{w} = \mathbf{\Phi}^T\left(\lambda\mathbf{I} + \mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{y} = \mathbf{\Phi}^T\left(\lambda\mathbf{I} + \mathbf{K}\right)^{-1}\mathbf{y}, \tag{A.3}$$

where $\mathbf{K}$ is the matrix with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$. That is, $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$ and is called the kernel matrix of $\mathbf{X}$.

Equation A.3 can be rewritten as $\mathbf{w} = \sum_{n=1}^{N}\alpha_i\Phi(\mathbf{x}_i)$ with $\boldsymbol{\alpha} = \mathbf{y}^T\left(\lambda\mathbf{I} + \mathbf{K}\right)^{-1}$. By observing $\mathbf{w} = \sum_{n=1}^{N}\alpha_i\Phi(\mathbf{x}_i)$, the most computationally expensive part is calculating $\left(\lambda\mathbf{I} + \mathbf{K}\right)^{-1}$ which is to invert a N by N matrix. In particular, if we repeat random split many times and do cross validation on each training part, the consuming time can be very expensive. Motivated by this, we explores the potential possibility of speed up the experiment in this setting.

When considering cross validation on different λ, the matrices need to be inverted seem "similar". "Similar" means that the difference of the matrix is only at diagonal and it is always a constant value. For example, when cross validation on λ_1, the matrix need to be inverted is $\mathbf{A}_1 = (\lambda_1\mathbf{I} + \mathbf{K})$ and when cross validation on λ_2, the matrix is $\mathbf{A}_2 = (\lambda_2\mathbf{I} + \mathbf{K})$. The difference between $\mathbf{A}_1$ and $\mathbf{A}_2$ is only at diagonal and it is $\lambda_1 - \lambda_2$. Taking the advantage of this property may be able to speed up the flow of experiment of kernel ridge regression.

On the other hand, when considering random splitting data, we image there is a matrix $\tilde{\mathbf{A}} = (\lambda \mathbf{I} + \tilde{\mathbf{K}})$ where $\tilde{\mathbf{K}}$ is the kernel matrix of all data without partition. Then, after split, the training part is a subset of all data and the kernel matrix $\mathbf{K}$ of training part is $\tilde{\mathbf{K}}$ without test part. That is, the matrix need to be inverted for each random split data can be obtained by removing some rows and corresponding column according the random split from $\tilde{\mathbf{K}}$. Motivate by this, we explore whether the experiment of kernel ridge regression can be speed up as taking the advantage of this.

A.2 Cross Validation

In this section, we study the speed-up of cross validation when using different λ. The procedure of cross validation contains two procedure. The first procedure is randomly partitioning the training part into N subsamples, i.e. N-fold. A single subsample is retained as the validation data for testing the model performance and the remaining $N - 1$ subsamples are used as training data, and this is then repeated N times, with each of the N subsamples use exactly once as the validation data. The second procedure is using different λ to train model and validating the performance of model on the validation data. It is obvious that the procedure of N-fold is a special case of random splitting data; therefore, the N-fold part is leave to next section and we focus on using different λ_l to train kernel ridge regression in this section.

When considering cross validation with the parameter set $\lambda = \{\lambda_1, \lambda_2 \ldots \lambda_L\}$, we must calculate the inversion of $(\lambda_1 \mathbf{I} + \mathbf{K}), (\lambda_2 \mathbf{I} + \mathbf{K}) \ldots (\lambda_L \mathbf{I} + \mathbf{K})$. The key idea of speed-up is performing orthogonal diagonalization [35] on matrix $\mathbf{K}$ first. As $\mathbf{K}$ is symmetry and semi-positive definite, $\mathbf{K}$ can be decomposed to

$$\mathbf{K} = \mathbf{Q}^T \mathbf{D} \mathbf{Q},$$

where $\mathbf{Q}^T = \mathbf{Q}^{-1}$ and $\mathbf{D}$ is diagonal matrix.

After obtaining $\mathbf{Q}$ and $\mathbf{D}$, the inversion of $(\lambda_l \mathbf{I} + \mathbf{K})$ for λ_l can be easily obtained by

$$\begin{aligned}(\mathbf{K} + \lambda\mathbf{I})^{-1} &= (\mathbf{Q}^T\mathbf{D}\mathbf{Q} + \lambda\mathbf{I})^{-1} \\ &= (\mathbf{Q}^T\mathbf{D}\mathbf{Q} + \lambda\mathbf{Q}^T\mathbf{Q})^{-1} \\ &= (\mathbf{Q}^T(\mathbf{D} + \lambda\mathbf{I})\mathbf{Q})^{-1} \\ &= \mathbf{Q}^T(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{Q}\end{aligned}$$

Because $\mathbf{D}$ is diagonal matrix, the inversion of matrix $(\mathbf{D} + \lambda\mathbf{I})$, which is still a diagonal matrix, is extreme fast by calculating the reciprocal of each element at diagonal. However, because orthogonal diagonalization may be not as efficient as inversion, we conduct experiment about this and check it performance in in section A.4.

A.3 Different Split

In this section, we consider how to speed up when using different training part. When considering random splitting data, we build $\tilde{\mathbf{A}} = (\lambda\mathbf{I} + \tilde{\mathbf{K}})$ and its inversion $\tilde{\mathbf{A}}^{-1}$ where $\tilde{\mathbf{K}}$ is the kernel matrix of all data before partition. Then, after split, the training part is a subset of all data and thus the kernel matrix $\mathbf{K}$ of training part is $\tilde{\mathbf{K}}$ without test part. That is, $\mathbf{K}$ can be obtained by removing some rows and corresponding column from $\tilde{\mathbf{K}}$ and the matrix need to be inverted is $\mathbf{A} = (\lambda\mathbf{I} + \mathbf{K})$. Notice that $\mathbf{A}$ is also able to obtain by removing some rows and corresponding column from $\tilde{\mathbf{A}}$. Then, can we speed up the inversion of $\mathbf{A}$ after pre-calculating the inversion of $\tilde{\mathbf{A}}$? In formal, our goal is that given matrix $\tilde{\mathbf{A}}$ and its inversion $\tilde{\mathbf{A}}^{-1}$, can we calculate $\mathbf{A}^{-1}$ efficiently where $\mathbf{A}$ is obtained by removing rows and columns from $\tilde{\mathbf{A}}$? In particular, we define the set of the index of removed rows and columns is $R = (r_1, r_2, \ldots, r_{|R|}), r_1 < r_2 < \cdots < r_{|R|}$.

A.3.1 Permutation Matrix

When given a matrix $\tilde{\mathbf{A}}$, directly calculating $\mathbf{A}^{-1}$ is difficult because the index R which need to be removed is scattered over the matrix. Therefore, we try to permute the index R to the end first.

Lemma A.3.1 *Let a matrix* $\mathbf{A}$ *be symmetric and its inversion exists. Suppose* $\hat{\mathbf{A}}$ *is* $\mathbf{A}$ *after swapping row* (i, j) *and column* (i, j) *and called this operation swap* (i, j)*. Then* $\hat{\mathbf{A}}^{-1}$ *is* $\mathbf{A}^{-1}$ *after swap* (i, j)*.*

Lemma A.3.1 demonstrates that if we want to calculate $\mathbf{A}^{-1}$ which is $\tilde{\mathbf{A}}$ after removing R rows and columns, we can permute R rows and columns to $R' = (N-|R|+1, N-|R|+2, \ldots, N)$ rows and columns first and we call this operation permuting R to R' on $\tilde{\mathbf{A}}$. That is, when given a matrix $\tilde{\mathbf{A}}$ and the index R, we in order swap $(r_1, N-|R|+1)$, $(r_2, N-|R|+2)$, … , and $(r_{|R|}, N)$ on $\tilde{\mathbf{A}}$ to get $\hat{\mathbf{A}}$ and do the same thing on $\tilde{\mathbf{A}}^{-1}$ to get $\hat{\mathbf{A}}^{-1}$ first.

Then, the remaining problem is that given our goal is that given matrix $\tilde{\mathbf{A}}$ and its inversion $\tilde{\mathbf{A}}^{-1}$, can we calculate $\mathbf{A}^{-1}$ efficiently where $\mathbf{A}$ is obtained by removing $R = (N-|R|+1, N-|R|+2, \ldots, N)$ rows and columns from $\tilde{\mathbf{A}}$?

A.3.2 Blockwise Inversion

For solving the remaining task we need to first introduce a technique, block matrix inversion [36].

Lemma A.3.2 *Let matrix* $\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{C} \end{bmatrix}$*, where* $\tilde{\mathbf{A}}$ *and* $\mathbf{A}$ *are as previous definition. Then,*

$$\tilde{\mathbf{A}}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}^T\mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1} & -\mathbf{A}\mathbf{B}^T\mathbf{T}^{-1} \\ -\mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1} & \mathbf{T}^{-1} \end{bmatrix}$$

$$\mathbf{T} = \mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T$$

Notice that, Lemma A.3.2 is a special case of general block matrix inversion because we only need this special version in our derivation.

By this lemma, given $\tilde{\mathbf{A}}^{-1} = \begin{bmatrix} \mathbf{A}'_{11} & \mathbf{A}'^{T}_{21} \\ \mathbf{A}'_{21} & \mathbf{A}'_{22} \end{bmatrix}$, we can obtain a equation,

$$\mathbf{A}'_{11} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}^T\mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1}, \tag{A.4}$$

where matrix $\mathbf{A}'_{11}$, $\mathbf{A}$ and $\mathbf{B}$ are given and $\mathbf{A}^{-1}$ is our goal. In particular, $\mathbf{A}^{-1}$ in $\mathbf{A}^{-1}\mathbf{B}^T\mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1}$ can be combined and replaced by the known matrix.

$$\begin{align} \mathbf{A}^{-1}\mathbf{B}^T\mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1} &= \mathbf{A}^{-1}\mathbf{B}^T\mathbf{T}^{-1}(\mathbf{T}\mathbf{T}^{-1})\mathbf{B}\mathbf{A}^{-1} \tag{A.5} \\ &= \mathbf{A}'^{T}_{21}\mathbf{T}\mathbf{A}'_{21} \tag{A.6} \end{align}$$

Equation (A.6) can be obtained by substitute $\mathbf{A}'_{12} = \mathbf{T}^{-1}\mathbf{B}\mathbf{A}^{-1}$ into equation (A.5). After getting equation (A.6), we can substitute (A.6) into equation (A.4) and obtain

$$\begin{align} \mathbf{A}'_{11} =& \mathbf{A}^{-1} + \mathbf{A}'^{T}_{21}\mathbf{T}\mathbf{A}'_{21} \\ \Rightarrow \mathbf{A}^{-1} =& \mathbf{A}'_{11} - \mathbf{A}'^{T}_{21}\mathbf{T}\mathbf{A}'_{21}. \tag{A.7} \end{align}$$

In equation (A.7), $\mathbf{T}$ is unknown and $\mathbf{A}^{-1}$ is our target. The remaining problem is to calculate matrix $\mathbf{T}$. Observing that $\mathbf{T}$ is not only equal to $\mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T$ but also equal to $\mathbf{A}'^{-1}_{22}$, $\mathbf{T}$ can be obtained by calculating $\mathbf{A}'^{-1}_{22}$ and equation (A.7) becomes

$$\mathbf{A}^{-1} = \mathbf{A}'_{11} - \mathbf{A}'^{T}_{21}\mathbf{A}'^{-1}_{22}\mathbf{A}'_{21}. \tag{A.8}$$

By using equation (A.8), we can obtain two algorithms for calculating $\mathbf{A}^{-1}$ as shown in Algorithm 2 and Algorithm 3.

The first algorithm (Algorithm 2) is to calculate $\mathbf{T}$ by $\mathbf{A}^{-1}_{22}$ because no matter in cross validation or in split dataset, $\mathbf{A}_{22}$ is relative small with respect to $\mathbf{A}$ and thus $\mathbf{A}^{-1}_{22}$ can be calculated more efficiently than directly calculating $\mathbf{A}^{-1}$.

Algorithm 2 Inversion by remove rows and columns

```
1: Input: $\tilde{\mathbf{A}}^{-1}$ and $R$
2: Permuting $R$ to $R'$ on $\tilde{\mathbf{A}}^{-1}$.
3: Split $\tilde{\mathbf{A}}^{-1}$ into $\tilde{\mathbf{A}}_{11}$, $\tilde{\mathbf{A}}_{21}$, and $\tilde{\mathbf{A}}_{22}$.
4: Calculating $\mathbf{A}^{-1} = \tilde{\mathbf{A}}_{11} - \tilde{\mathbf{A}}_{21}^T \tilde{\mathbf{A}}_{22}^{-1} \tilde{\mathbf{A}}_{21}$.
5: Permuting $R'$ to $R$ on $\mathbf{A}^{-1}$.
6: Output: $\mathbf{A}^{-1}$
```

The second algorithm (Algorithm 3) is to remove all indexes in R one by one. By removing one by one, calculating the inversion of $\tilde{\mathbf{A}}_{22}$ can be extreme fast, because it is 1-by-1 matrix and can be obtained by reciprocal. However, the trade-off is this procedure must repeat $|R|$ times and need to perform plenty of matrix multiplications.

Algorithm 3 Inversion by remove rows and columns-one by one

```
1: Input: $\tilde{\mathbf{A}}^{-1}$ and $R$
2: Let $\mathbf{A}_0 = \tilde{\mathbf{A}}^{-1}$ .
3: for $i = 1 \to |R|$ do
4:     Run algorithm 2 on $\mathbf{A}_{i-1}$ and $\{r_i\}$ and get $\mathbf{A}_i$.
5: end for
6: Output: $\mathbf{A}_{|R|}$
```

For comparing these two algorithms, we conduct experiment for in section A.4.

A.4 Experiment

In this section, we conduct experiment for testing the performance of the method proposed in section A.2 and section A.3. We use a real world dataset `bibtex` which contains 7395 instances and 1836 features for comparison. The computational time are measured on Intel Xeon E5530 2.4G Processor with 8192KB cache size and these method are implemented by MATLAB version 7.14.0.739 (R2012a). In section A.4.1, we compare the time of calculating the inversion directly and calculating the inversion by using orthogonal diagonalization as mention in section A.2. And in section A.4.2, we conduct experiment about the time of calculating inversion under random split training part.

A.4.1 Experiment of Cross Validation

In this experiment, because the cross validation is usually performed on the training part; therefore, it use 5916 instances which is 80% from original data. The measured time is reported in table A.1.

	Inverse	Diagonalize	Inverse by Diagonalization
Time (s)	49.911741	111.854334	26.635818

Table A.1: Inversion vs Diagonalization vs Inverse by Diagonalization

In table A.1, we can see that the time for diagonalization is roughly as twice much as inversion. But after the preprocessing of diagonalization, the time of inversion only need a half of original time. If we check how many experiments need to do for offset the overhead of preprocessing, it need repeat

$$\Lambda = \frac{111.854334}{49.911741 - 26.635818} \approx 4.8056$$

times to offset the overhead. It shows that the experiment can be faster when we experiment for over 5 different λ value. And if the experiment for large number of different λ, it only need a half of original time.

A.4.2 Experiment of Different Split

The experiment in this subsection, we check the result of Algorithm 2 and Algorithm 3. The situation we experiment is partitioning whole dataset into 80% for training part and 20% for testing part. The measured time is reported in table A.2.

For both Algorithm 2 and Algorithm 3, it need to calculate the inversion of $\tilde{\mathbf{A}}$ which the original needn't to calculate. It can be found that the time of the inversion of $\tilde{\mathbf{A}}$ is as twice much as $\hat{\mathbf{A}}$ in table A.2, because the time complexity of inversion is worse than linear. Therefore, the inversion of $\tilde{\mathbf{A}}$ is the overhead for Algorithm 2 and Algorithm 3.

After preprocessing, when we calculate $\mathbf{T}$ by the inversion of $\tilde{\mathbf{A}}_{22}$ as Algorithm 2, the time compared with original one is extreme fast and is roughly a tenth of original time. On the other hand, the time using Algorithm 3 is extreme slow. It may be because the number of the multiplication of matrix we need is too many.

	Inverse $\tilde{\mathbf{A}}$	Inverse $\hat{\mathbf{A}}$	Inverse $\hat{\mathbf{A}}$ by Algorithm 2	Inverse $\hat{\mathbf{A}}$ by Algorithm 3
Time(s)	102.664412	50.405461	5.030477	6311.317599

Table A.2: Algorithm 2 vs Algorithm 3

Compared with directly inversing $\hat{\mathbf{A}}$, it need

$$\Delta = \frac{102.664412}{50.405461 - 5.030477} \approx 2.26258$$

times to offset the overhead. It shows tat the experiment can be faster when we repeat it over 3 times. If the experiment is repeated many times, it only need a tenth of original time.

Bibliography

[1] I. Katakis, G. Tsoumakas, and I. Vlahavas. Multilabel text classification for automated tag suggestion. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2008 Discovery Challenge*, 2008.

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

[3] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14*, 2001.

[4] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas. Multilabel classification of music into emotions. In *Proceedings of the 9th International Conference on Music Information Retrieval*, 2008.

[5] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, pages 254–269, 2011.

[6] G. Tsoumakas and I. Vlahavas. Random k-Labelsets: An ensemble method for multilabel classification. In *Machine Learning: the European Conference on Machine Learning*, volume 4701, pages 406–417. 2007.

[7] D. Hsu, S. Kakade, J. Langford, and T. Zhang. Multi-Label prediction via compressed sensing. In *Advances in Neural Information Processing Systems 22*. 2009.

[8] F. Tai and H.-T. Lin. Multi-Label classification with principal label space transformation. In *Neural Computation*, 2012.

[9] M.E. Wall, A. Rechtsteiner, and L.M. Rocha. Singular value decomposition and principal component analysis. *A Practical Approach to Microarray Data Analysis*, 2003.

[10] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, 2002.

[11] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Zolghadri Jahromi. Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds. *Pattern Recognition*, 44:1357–1371, 2011.

[12] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188, 1936.

[13] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28:321–377, 1936.

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

[15] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*. 2010.

[16] E. Loza Mencía and J. Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, 2008.

[17] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000.

[18] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In *Advances in Neural Information Processing Systems 15*, 2002.

[19] J. R. Kettenring. Canonical analysis of several sets of variables. *Biometrika*, 58: 433–451, 1971.

[20] S. Yu, K. Yu, V. Tresp, and H.-P. Kriegel. Multi-output regularized feature projection. *Knowledge and Data Engineering, IEEE Transactions on*, 18:1600 –1613, 2006.

[21] D. C. Hoaglin and R. E. Welsch. The hat matrix in regression and ANOVA. *The American Statistician*, 32:17–22, 1978.

[22] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[23] K. Dembczynski, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence in Multi-Label classification. In *Workshop Proceedings of Learning from Multi-Label Data*, 2010.

[24] B. Schölkopf and A. J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond.* The MIT Press, 1st edition, 2002.

[25] G. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

[26] A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

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

[28] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[29] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*, 1997.

[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11:10–18, 2009.

[31] F. Pereira and G. Gordon. The support vector decomposition machine. In *Proceedings of the 23rd International Conference on Machine learning*, 2006.

[32] I. Rish, G. Grabarnik, G. Cecchi, F. Pereira, and G. J. Gordon. Closed-form supervised dimensionality reduction with generalized linear models. In *Proceedings of the 25th International Conference on Machine learning*, 2008.

[33] Sajama and A. Orlitsky. Supervised dimensionality reduction using mixture models. In *Proceedings of the 22nd International Conference on Machine learning*, 2005.

[34] C.-S. Ferng and H.-T. Lin. Multi-label classification with error-correcting codes. *Journal of Machine Learning Research - Proceedings Track*, pages 281 –295, 2011.

[35] S. Lipschutz. *3,000 Solved Problems in Linear Algebra.* McGraw-Hill, 1989.

[36] S. E. Jo, S. W. Kim, and T. J. Park. *Equally constrained affine projection algorithm.* 2004.