UNIVERSITY OF SOUTHAMPTON

Sparse Kernel Feature Extraction

by

Charanpal Dhanjal

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

November 2008

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Sparse Kernel Feature Extraction

by Charanpal Dhanjal

The presence of irrelevant features in training data is a significant obstacle for many machine learning tasks, since it can decrease accuracy, make it harder to understand the learned model and increase computational and memory requirements. One approach to this problem is to extract appropriate features. General approaches such as Principal Components Analysis (PCA) are successful for a variety of applications, however they can be improved upon by targeting feature extraction towards more specific problems. More recent work has been more focused and considers sparser formulations which potentially have improved generalisation. However, sparsity is not always efficiently implemented and frequently requires complex optimisation routines. Furthermore, one often does not have a direct control on the sparsity of the solution. In this thesis, we address some of these problems, first by proposing a general framework for feature extraction which possesses a number of useful properties. The framework is based on Partial Least Squares (PLS), and one can choose a user defined criterion to compute projection directions. It draws together a number of existing results and provides additional insights into several popular feature extraction methods. More specific feature extraction is considered for three objectives: matrix approximation, supervised feature extraction and learning the semantics of two-viewed data. Computational and memory efficiency is prioritised, as well as sparsity in a direct manner and simple implementations. For the matrix approximation case, an analysis of different orthogonalisation methods is presented in terms of the optimal choice of projection direction. The analysis results in a new derivation for Kernel Feature Analysis (KFA) and the formation of two novel matrix approximation methods based on PLS. In the supervised case, we apply the general feature extraction framework to derive two new methods based on maximising covariance and alignment respectively. Finally, we outline a novel sparse variant of Kernel Canonical Correlation Analysis (KCCA) which approximates a cardinality constrained optimisation. This method, as well as a variant which performs feature selection in one view, is applied to an enzyme function prediction case study.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Charanpal Dhanjal, declare that the thesis entitled "Sparse Kernel Feature Extraction" and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as:
    - C. Dhanjal, S. R. Gunn, and J. Shawe-Taylor. Sparse Feature Extraction using Generalised Partial Least Squares, In *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, pages 27-32, 2006.

Signed: ....

Date: ....

Acknowledgements

Primarily I would like to thank Prof. Steve Gunn for his advice, enthusiasm and support during the course of my PhD. I am also grateful to Prof. John Shawe-Taylor for his supervision during the first part of my degree. Thanks for the support and advice of my colleagues Dr. Craig Saunders and Dr. Sandor Szedmak. For their crucial part in formulating the data for the enzyme function prediction work, Katja Astikainen, Liisa Holm, Esa Pitkänen, and Juho Rousu are credited. Furthermore, I would like to thank Dr. Adam Prügel-Bennett for assisting with my foray into the world of research.

I am grateful for the useful and inspiring conversations with Alexei, the positivity of Jacob, and the cold realism of Shodhan. Lastly, and perhaps most importantly I am thankful for the patience, support and understanding of my parents.

Nomenclature

$\mathbf{A}, \ldots, \mathbf{Z}$	a bold uppercase letter represents a matrix
$\mathbf{a}, \ldots, \mathbf{z}$	a bold lowercase letter represents a vector
k	number of output features
ℓ	number of examples
m	number of features of $\mathbf{X}$
n	number of features of $\mathbf{Y}$
$\mathbf{A}_{ij}$	element at ith row and jth column of $\mathbf{A}$
$\mathbf{0}$	all zeros vector
$\boldsymbol{\alpha}, \boldsymbol{\beta}$	dual projection vectors
$\mathbf{e}_i$	ith standard unit vector
$\mathbf{j}$	all ones vector
$\mathbf{u}, \mathbf{v}, \mathbf{w}$	primal projection vectors
$\mathbf{x}$	an example
$\mathbf{y}$	label vector or example
$\mathbf{C}$	covariance matrix
$\mathbf{I}$	identity matrix
$\mathbf{K}$	kernel matrix
$\mathbf{X}$	data matrix with rows as examples
$\mathbf{Y}$	label matrix with rows as labels vectors
$\mathrm{corr}(\cdot, \cdot)$	correlation between two random variables
$\mathrm{cov}(\cdot, \cdot)$	covariance between two random variables
$\mathrm{var}(\cdot)$	variance of a random variable
$\mathbb{E}[\cdot]$	expectation of a random variable
$\hat{\mathbb{E}}[\cdot]$	empirical expectation of a random variable
$\mathcal{L}(\cdot)$	Lagrange objective function
$\phi(\cdot)$	function mapping from input to feature space
$\hat{\phi}(\cdot)$	function mapping from input to extracted feature space
$\kappa(\cdot, \cdot)$	kernel function
$\mathrm{card}(\cdot)$	number of non-zero elements of a vector
$\mathrm{diag}(\cdot)$	vector of diagonal elements of a matrix
$\mathrm{rank}(\cdot)$	matrix rank
$\mathrm{tr}(\cdot)$	matrix trace

$\mathbf{A}'$	matrix transpose of $\mathbf{A}$
$\mathbf{A}[I, J]$	submatrix of $\mathbf{A}$ defined by row indices I and column indices J
$\\| \cdot \\|_p$	p-norm, with default value 2

Chapter 1

Introduction

A crucial aspect of science is making observations for a particular phenomenon and then formulating inferences from the observations. Where the data is highly multidimensional and patterns are complicated, inferences can be difficult to discover manually. Hence, there is a requirement for automatic procedures for such tasks, which is precisely the aim of *machine learning* (Mitchell (1997); Bishop (2006)). To illustrate a typical application of machine learning, consider the prediction problem using the Iris flower dataset (Fisher (1936)). The dataset contains 150 observations of three types of plant characterised by four *attributes* or *features*: sepal length, sepal width, petal length and petal width. The aim of the prediction problem is to learn the mapping of the attributes to the plant type. One can then identify a new unseen plant using its attributes. This task is an example of a *classification* problem since the value one is trying to predict is selected from a finite number of classes. In the *regression* problem, one predicts a real valued number or vector, for example the plant height. Both classification and regression have been well-studied and several popular and effective algorithms include Support Vector Machines (SVMs, Boser et al. (1992)), Logistic Regression (Hosmer and Lemeshow (2000)) and Boosting (Freund and Schapire (1995)).

Although these algorithms often generalise well, it is not always clear which features are useful for prediction. Furthermore, certain application domains such as text classification, bioinformatics, and image retrieval are characterised by a high number of features, many of which are irrelevant for learning. The discovery of the useful features improves understanding of the data, reduces computational storage and processing requirements and can increase prediction accuracy (e.g. with SVMs in Weston et al. (2000); Bi et al. (2003)). The field of *feature selection*[1] (Guyon and Elisseeff (2003); Guyon (2008)) is devoted to finding a subset of the features which can improve learning performance or give insight into the data. In this thesis, we focus mainly on the more general *feature*

[1] An insightful discussion on the relevance of features can be found in Blum and Langley (1997).

extraction (Guyon et al. (2006)) problem which aims to reduce the dimensionality of the data, whilst retaining important information.

Before continuing, it is useful to formalise some of the concepts given in the preceding text, starting with the definition of two types of dataset.

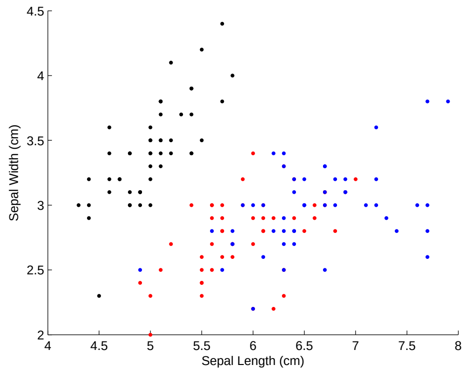
Definition 1.1 (Unlabelled Dataset)**.** An *unlabelled dataset* is denoted by the set of vectors $S = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$ where $\mathbf{x}_i \in \mathbb{R}^m$ is an *example* or *observation* with m features or *dimensions*. An example is a representation used for an object in a machine learning task, and a feature is a quantity used to describe the example.

Definition 1.2 (Labelled Dataset)**.** A *labelled dataset* is an unlabelled dataset with an additional *label* or *target* vector for each example, and is given by the set of pairs $S = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$, with $\mathbf{y}_i \in \mathbb{R}^n$. A label vector is composed of the set of quantities one wishes to predict for each example.

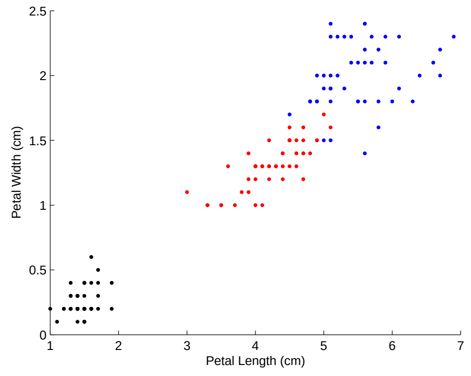
These definitions lead onto more precise statements describing feature selection and feature extraction.

Definition 1.3 (Feature Extraction)**.** Let S be an unlabelled dataset described as a matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$ which has its rows composed of the observations. *Unsupervised feature extraction* is the process of finding a new matrix $\hat{\mathbf{X}} \in \mathbb{R}^{\ell \times k}$, $k < m$, which is a low dimensional representation of the useful information in $\mathbf{X}$. *Supervised feature extraction* performs an equivalent operation on a labelled dataset.

Definition 1.4. (Feature selection) Feature selection is a special case of feature extraction, in which the new matrix $\hat{\mathbf{X}}$ is composed of a subset of the columns of $\mathbf{X}$.



(a) Sepal width versus sepal length



(b) Petal width versus petal length

FIGURE 1.1: Plot of the Iris dataset using different feature sets. The black points represent Iris setosa plants, the red points are Iris versicolour, and the blue ones are Iris virginica plants.

Figure 1.1 illustrates two plots using different features from the Iris dataset. In Figure 1.1(a) the sepal features are not useful for discriminating Iris versicolour from Iris virginica plants. In contrast, Figure 1.1(b) shows that petal lengths and widths can effectively discriminate all three types of plant. Furthermore, one could accurately classify the plant types using just a single feature: the ratio between petal width and length. This simple example illustrates a situation in which feature extraction can reduce the number of features from 4 to 1, whilst still retaining much of the discriminative information.

1.1 Challenges and Problem Statement

One of the important problems in feature extraction is defining a precise problem statement. For example, if one were to target features towards performing regression then the optimal choice depends on the regression algorithm as opposed to the regression problem itself. To illustrate this point, a polynomial mapping between features and labels would be appropriate for a polynomial regression method, as opposed to one which fits linear models. It follows that a general approach to feature extraction is useful since it allows one to specify the type of the features required without designing an algorithm from scratch. One such approach is given in Smola et al. (1999) which iteratively projects the examples into the space orthogonal to a projection direction. A disadvantage of this approach however is that the resulting features could potentially be correlated and hence redundant. One issue examined in this thesis is whether one can formulate a general feature extraction method which guarantees uncorrelated features.

Another key challenge in feature extraction occurs when one uses *kernel functions* (Aronszajn (1950)), which provide a flexible way of modelling non-linear relationships. With an algorithm that uses kernel functions, one no longer has access to the kernel features and hence performing feature selection becomes difficult or impossible. Furthermore, since learning often requires a kernel matrix which is quadratic in the number of examples, good computational and memory efficiency is difficult to achieve.

One way of improving efficiency is to introduce sparsity into the feature extraction formulation, which restricts learning to use only a subset of the examples. This can have the additional advantage of improving interpretability and generalisation, since examples which are considered as outliers can be omitted. To date, there have been several traditional feature extraction methods (such as Principal Components Analysis (PCA, Hotelling (1933))) which have been adapted to incorporate sparsity (e.g. sparse PCA (d'Aspremont et al. (2005))). However, sparsity is often implemented using complex optimisation routines, and few methods offer a direct control on the sparsity of the solution. In this thesis, we propose several feature extraction approaches which offer a direct control on sparsity in a simple and efficient manner.

This thesis tackles three important feature extraction problems. The first asks the question: How can one find a low dimensional approximation of a set of examples most effectively? In particular, given that orthogonalisation is a intuitive method to remove important aspects of the features we pose the problem of finding optimal directions for different orthogonalisation methods. The second problem considers the scenario where the labels are used to guide feature extraction, which is useful as a step before prediction for example. Finally, we examine the problem of finding the common semantics of a set of paired examples, $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$, where $\mathbf{y} \in \mathbb{R}^n$ is an alternative representation of $\mathbf{x}$.

1.2 Contributions

The main contributions of this thesis are summarised as follows:

- The formulation of a general framework for feature extraction which is a generalisation of Partial Least Squares (PLS, Wold (1966)) and its kernel variant. The framework draws together a number of existing results and provides additional insight into several popular feature extraction methods. It also serves as a tool to formulate new methods, which have the advantage of possessing many of the useful properties of PLS.
- The analysis of orthogonalisation procedures in the context of matrix approximation. Given a method of orthogonalising a data or kernel matrix, we compute optimal sparse projection directions. The analysis results in the formation of two novel matrix approximation algorithms based on PLS, which is the first instance it has been used in this setting. Furthermore, the investigation reveals an alternative derivation for an existing sparse approximation method.
- The formation of two new supervised feature extraction methods using the general feature extraction framework. Both a theoretical and empirical evaluation is conducted for these algorithms. A useful theoretical result gives insight into the statistical stability of the resulting features, and is general enough to be applied to PLS.
- An alternative sparse variant of the Kernel Canonical Correlation Analysis (KCCA) algorithm given in Bach and Jordan (2003), which finds the semantics of a set of paired examples. The formulation used potentially results in an increased level of sparsity by targeting sparsity towards optimising correlation. Furthermore, we derive a variant of our sparse approach which performs feature selection.
- The application of KCCA and our sparse variants to a real-world enzyme function prediction problem. This work is novel because it considers fine-grained prediction of enzyme function and also uses new feature representations for both enzymes and their reactions.

- The implementation of a significant volume of code, some of which is available at `http://users.ecs.soton.ac.uk/cd04r/code.php`.

This work has contributed to the following publications:

- C. Dhanjal, S. R. Gunn, and J. Shawe-Taylor. Sparse Feature Extraction using Generalised Partial Least Squares, In *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, pages 27-32, 2006.
- C. Dhanjal, S. R. Gunn, and J. Shawe-Taylor. Efficient Sparse Kernel Feature Extraction Based on Partial Least Squares, Accepted for publication in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

1.3 Thesis Outline

The thesis first reviews in Chapter 2 important work in feature extraction. The chapter begins with a description of two key elements of current feature extraction algorithms: kernel methods and eigenproblems. It then presents a literature review corresponding to each of the three feature extraction scenarios outlined in Section 1.1. Chapter 3 introduces a general framework for feature extraction and demonstrates its specialisations to several existing feature extraction methods. The next three chapters consider specific feature extraction scenarios. Chapter 4 presents a study of matrix approximation by examining a number of different orthogonalisation methods. Two new sparse approximation methods are derived and evaluated against several existing algorithms. Chapter 5 considers the supervised feature extraction problem by using the general framework for feature extraction with supervised projection directions. This is followed by Chapter 6 which explores sparse feature extraction using paired examples. Two sparse alternatives to KCCA are formulated, analysed, and then applied to an enzyme function prediction case study. This thesis concludes with Chapter 7 which provides a summary and review of the thesis, as well as interesting directions for future research.

Chapter 2

Feature Extraction

As previously described, feature extraction is a process which reduces the dimensionality of a set of examples by removing redundant or irrelevant information. The informative characteristics of the examples are defined by the problem under investigation. For example, in PCA the directions of minimal variance are considered as uninformative and hence removed, whereas in a supervised setting directions which are predictive of the labels are selected. In this chapter important feature extraction techniques which fall under several different problem settings are reviewed. The aim is to provide an overview of the field and also contextualise later work.

The feature extraction approaches covered in this chapter broadly can be categorised into linear and kernel methods. In the linear case, examples are projected onto the columns of a projection matrix $\mathbf{Z} = [\mathbf{z}_1, \dots \mathbf{z}_k]$, i.e. $\hat{\mathbf{X}} = \mathbf{X}\mathbf{Z}$, where $\mathbf{z}_1, \dots \mathbf{z}_k$ are called *projection vectors*. For kernel-based feature extraction, one operates in a high dimensional feature space. The new data is given by the projection $\hat{\mathbf{X}} = \mathbf{K}\mathbf{Q}$, where $\mathbf{K}$ is a *kernel matrix* and $\mathbf{Q}$ is a projection matrix whose columns are known as the *dual projection vectors*. One can see the equivalence to the primal case if the linear kernel is used, i.e. $\mathbf{K} = \mathbf{X}\mathbf{X}'$, and $\mathbf{Z} = \mathbf{X}'\mathbf{Q}$. If a non-linear kernel is used then the examples lie in a high dimensional feature space and $\hat{\mathbf{X}}$ is a linear projection of the examples in that space (illustrated in Figure 2.1).

Our overview begins with an introduction to eigenproblems[1] and kernel methods, which are key properties of many feature extraction algorithms. The following three sections detail important work in specific feature extraction scenarios which mirror the novel approaches developed in Chapters 4, 5 and 6. Section 2.4 details several unsupervised feature extraction methods, including the popular Principal Components Analysis and Gram-Schmidt Orthogonalisation approaches. Partial Least Squares is central to this thesis, and along with several supervised approaches, it is given an in depth coverage

[1] A good survey of eigenproblem based feature extraction methods is given in De Bie et al. (2004).

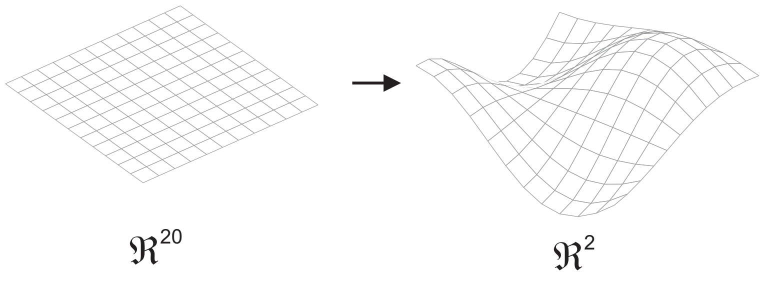


FIGURE 2.1: Mapping from a 20-dimensional kernel-defined feature space (left) to a 2-dimensional input space (right). A linear surface in the kernel-defined feature space is mapped in a non-linear manner into the input space.

in Section 2.5. In Section 2.6, methods which operate on paired data are reviewed, including Canonical Correlation Analysis (CCA, Hotelling (1936)) and variants.

2.1 A Toy Example

Consider a synthetic dataset composed of 150 examples and 200 features. The examples are generated using pseudo-random numbers drawn from a normal distribution with mean 0 and standard deviation 1. Each example $\mathbf{x}$ has a corresponding binary label $y \in \{-1, +1\}$ which is computed as follows

$$y = \text{sign}(\mathbf{x}'\mathbf{c} + n(0,1)),$$

where $\mathbf{c}$ is a vector of regression coefficients, $n(\mu, \sigma^2)$ is a small Gaussian noise component with mean μ and variance σ^2, and examples are generated using $\mathbf{x}_i = n(0,1)$, $i = 1, \dots, m$. In this case the number of non-zero elements of $\mathbf{c}$ is 10, hence only 10 features are used to formulate the labels.

We conduct a simple test as follows. The dataset is split into a training set of size 100 and a test set of size 50. A linear SVM is trained with values of the penalty parameter selected as $C \in \{2^{-4}, 2^{-3}, \dots, 2^{7}\}$ and the lowest error for the predictions on the test set is recorded. This process is repeated for varying numbers of irrelevant features from 0 to 190 in steps of 10.

The results are shown in Figure 2.2 and clearly there is a general increase in the classification error with more irrelevant features. The reason for the increase in classification error is that the 2-norm of the weight vector used in the SVM does not mind using many coefficients.

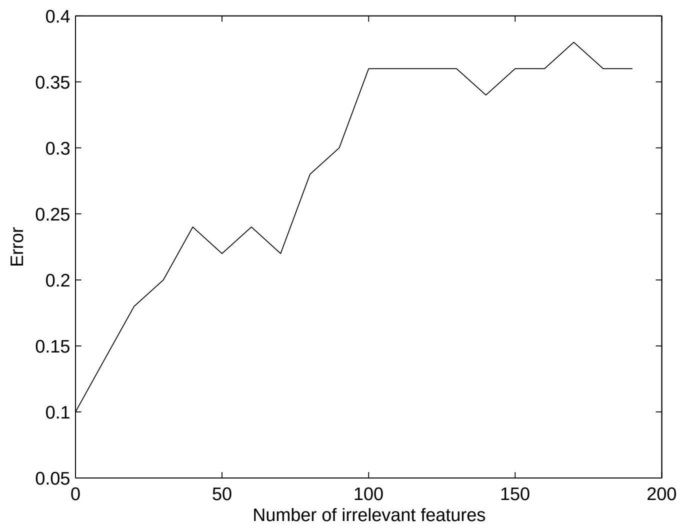


Figure 2.2: Test errors obtained on a toy dataset as the number of irrelevant features changes.

2.2 Eigenproblems

There are a number of feature extraction methods that rely on solving eigenproblems so these problems are introduced here and several of their important properties are detailed. First, the set of *eigenvectors* $\mathbf{v}$ of a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined as those non-zero vectors which when multiplied by $\mathbf{A}$ result in a scaling,

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}, \tag{2.1}$$

where λ is the corresponding *eigenvalue* of $\mathbf{A}$. The eigenvectors are invariant to scaling, however we assume unless otherwise stated, that they have unit norm. If $\mathbf{A}$ is a symmetric matrix, then the eigenvectors corresponding to distinct eigenvalues are orthogonal since

$$\begin{aligned} \lambda_j \mathbf{v}_j' \mathbf{v}_i &= \mathbf{v}_j' \mathbf{A}' \mathbf{v}_i \\ &= \mathbf{v}_j' \mathbf{A} \mathbf{v}_i \\ &= \lambda_i \mathbf{v}_j' \mathbf{v}_i, \end{aligned}$$

where $\lambda_j, \mathbf{v}_j$ and $\lambda_i, \mathbf{v}_i$ are eigenvalue-vector pairs of $\mathbf{A}$ and $\lambda_i \neq \lambda_j$. This property suggests that $\mathbf{A}$ can be written as

$$\mathbf{AV} = \mathbf{V\Lambda} \Rightarrow \mathbf{A} = \mathbf{V\Lambda V}',$$

where $\mathbf{V}$ has columns composed of the eigenvectors of $\mathbf{A}$ and $\mathbf{\Lambda}$ is a diagonal matrix of corresponding eigenvalues, $\mathbf{\Lambda}_{ii} = \lambda_i$. This representation is known as the *eigenvalue decomposition* of a symmetric matrix $\mathbf{A}$.

Next consider the *generalised eigenvalue problem*, which has the form

$$\mathbf{Av} = \lambda \mathbf{Bv}, \tag{2.2}$$

where $\mathbf{A}, \mathbf{B}$ are symmetric and $\mathbf{B} \in \mathbb{R}^{n \times n}$ is *positive definite*, i.e. $\mathbf{z}'\mathbf{Bz} > 0$ for non-zero $\mathbf{z}$. Clearly $\mathbf{B}$ is invertible hence one can write $\mathbf{B}^{-1}\mathbf{Av} = \lambda\mathbf{v}$ which brings us back to the initial eigenvalue problem. Furthermore, the positive definiteness of $\mathbf{B}$ implies that it can be written as $\mathbf{B} = \mathbf{B}^{1/2}\mathbf{B}^{1/2}$, hence defining $\mathbf{w} = \mathbf{B}^{1/2}\mathbf{v}$ results in the standard eigenvalue problem in terms of $\mathbf{w}$,

$$\mathbf{B}^{-1/2}\mathbf{AB}^{-1/2}\mathbf{w} = \lambda\mathbf{w}. \tag{2.3}$$

Since $\mathbf{B}^{-1/2}\mathbf{AB}^{-1/2}$ is symmetric it follows that $\mathbf{w}_i'\mathbf{w}_j = \mathbf{v}_i'\mathbf{Bv}_j = 0$, $\lambda_i \neq \lambda_j$, which implies that the eigenvectors $\mathbf{v}$ are orthogonal in the metric defined by $\mathbf{B}$. This in turn implies a simple operation on $\mathbf{A}_1 = \mathbf{A}$ which yields an iterative solution to the above eigenproblem,

$$\mathbf{A}_{j+1} = \mathbf{A}_j - \lambda_j \mathbf{Bv}_j\mathbf{v}_j'\mathbf{B}', \tag{2.4}$$

where $\mathbf{v}_j'\mathbf{Bv}_j = 1$, and $(\lambda_j, \mathbf{v}_j)$ is the dominant eigenvalue-vector pair of $\mathbf{A}_j$, $j = 1, \ldots, n$. Note that $\mathbf{A}_{j+1}\mathbf{v}_j = \mathbf{A}_j\mathbf{v}_j - \lambda_j\mathbf{Bv}_j\mathbf{v}_j'\mathbf{B}'\mathbf{v}_j = \mathbf{A}_j\mathbf{v}_j - \mathbf{A}_j\mathbf{v}_j = \mathbf{0}$, implying that the above step sets the dominant eigenvalue to zero. Also observe that for $i \geq j$, $\mathbf{A}_{j+1}\mathbf{v}_i = \mathbf{A}_j\mathbf{v}_i - \lambda_j\mathbf{Bv}_j\mathbf{v}_j'\mathbf{B}'\mathbf{v}_i = \mathbf{A}_j\mathbf{v}_i$, due to the orthogonality of the eigenvectors. It follows that Equation 2.4 sets the dominant eigenvalue of $\mathbf{A}_j$ to zero whilst leaving all other eigenvalues and eigenvectors intact.

Finally, we consider the case when $\mathbf{A}$ is not a square matrix hence by definition does not have eigenvalues and eigenvectors. One can instead consider the eigenvalue problem using the matrix $\mathbf{A}'\mathbf{A}$,

$$\mathbf{A}'\mathbf{Av} = \lambda\mathbf{v}. \tag{2.5}$$

Premultiplying both sides by $\mathbf{A}$ and letting $\sigma_u\mathbf{u} = \mathbf{A}\mathbf{v}$, where σ_u is a scaling factor, results in

$$\mathbf{A}\mathbf{A}'\mathbf{u} = \lambda\mathbf{u}, \tag{2.6}$$

which in turn implies $\sigma_v\mathbf{v} = \mathbf{A}'\mathbf{u}$, for some σ_v, using a similar process. It remains only to compute the scaling factors σ_u and σ_v. If Equations 2.5 and 2.6 are premultiplied by $\mathbf{v}'$ and $\mathbf{u}'$ respectively then

$$\lambda = \frac{\mathbf{v}'\mathbf{A}'\mathbf{A}\mathbf{v}}{\mathbf{v}'\mathbf{v}} = \frac{\mathbf{u}'\mathbf{A}\mathbf{A}'\mathbf{u}}{\mathbf{u}'\mathbf{u}},$$

and solving for σ_u and σ_v gives

$$\sigma_v^2 = \frac{\mathbf{v}'\mathbf{A}'\mathbf{A}\mathbf{v}}{\mathbf{v}'\mathbf{v}} = \lambda = \frac{\mathbf{u}'\mathbf{A}\mathbf{A}'\mathbf{u}}{\mathbf{u}'\mathbf{u}} = \sigma_u^2,$$

hence $\mathbf{A}'\mathbf{u} = \sigma\mathbf{v}$ and $\mathbf{A}\mathbf{v} = \sigma\mathbf{u}$, with $\sigma = \sigma_u = \sigma_v$. This is the *singular value decomposition* of $\mathbf{A}$ where $\mathbf{u}$ and $\mathbf{v}$ are the left and right *singular vectors* and σ is the corresponding *singular value* of $\mathbf{A}$. Due to the orthogonality of the singular vectors, $\mathbf{A}$ can be written as

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}',$$

where $\mathbf{U}$ and $\mathbf{V}$ are matrices with columns composed of the left and right singular vectors respectively and $\boldsymbol{\Sigma}$ is a diagonal matrix with $\boldsymbol{\Sigma}_{ii} = \sigma_i$.

2.3 Kernel Methods

Kernel functions are another key element of many feature extraction methods. They allow one to operate in a high dimensional feature space without explicitly computing feature representations in that space. They were popularised by the Support Vector Machine, however Aronszajn (1950) was one of the first to employ the kernel method.

Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ denote a dataset with $\mathbf{x} \in \mathcal{X}$, and $\phi(\mathbf{x}) \in \mathcal{F}$ be an alternative high dimensional representation for $\mathbf{x}$. Instead of explicitly computing $\phi(\mathbf{x})$, one evaluates inner products using a *kernel function* $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$,

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle.$$

All pairwise kernel evaluations are often represented as a kernel matrix, defined by $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, and provided a learning algorithm only requires inner product calculations, this is sufficient between examples. Notice that the kernel matrix is independent of the size of the feature representation $\phi(\mathbf{x})$, which implies computational storage and processing advantages for large feature vectors. Additionally, evaluating a kernel function is often less computationally expensive than explicitly computing the corresponding inner product. An important advantage of kernel learning algorithms is that one does not need not be concerned with the choice of kernel function when designing them.

The requirement that a kernel learning algorithm should only use the inner products between the examples, implies that it needs the norms of the examples and their relative angles. The inner product between two vectors can be written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \|\phi(\mathbf{x})\| \|\phi(\mathbf{z})\| \cos \theta,$$

where θ is the angle between $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$. Clearly one can compute θ using a kernel matrix by noting $\|\phi(\mathbf{x})\| = \sqrt{\langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle}$. Thus a kernel matrix is invariant to a rotation of the examples in the kernel feature space.

There are a few constraints required to formulate a kernel function. It should be *symmetric*, i.e. $\kappa(\mathbf{x}, \mathbf{z}) = \kappa(\mathbf{z}, \mathbf{x})$, and positive semi-definite so that

$$\sum_{i=1}^{\ell} \sum_{j=1}^{\ell} c_i c_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \geq 0,$$

for any $\ell > 0$, $\mathbf{x}_1, \ldots, \mathbf{x}_\ell \in \mathcal{X}$, and any choice of numbers $c_1, \ldots, c_\ell \in \mathbb{R}$. It follows that any mapping $\phi : \mathcal{X} \to \mathbb{R}^m$ for $m \geq 0$ results in a valid kernel. One need not restrict themselves to real vector spaces, since kernels can be defined for any vector space that has a dot product and is complete[2], known as a *Hilbert Space*. This is formalised by the following theorem.

Theorem 2.1. *For any kernel κ on a space $\mathcal{X}$, there exists a Hilbert space $\mathcal{F}$ and a mapping $\phi : \mathcal{X} \to \mathcal{F}$ such that*

$$\kappa(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle, \quad \textit{for any } \boldsymbol{x}, \boldsymbol{z} \in \mathcal{X},$$

where $\langle \cdot, \cdot \rangle$ represents the dot product in the Hilbert space between two points in $\mathcal{F}$.

[2]Completeness (Kreyszig (1978)) means that every Cauchy sequence of points in the space has a limit that is also in the space. For a Cauchy sequence, after a finite number of steps starting from the first term, any pair of elements chosen from the remaining terms will be within distance ε of each other, for a fixed $\varepsilon > 0$.

An intuitive way of considering kernel functions is as a similarity measure between two examples, i.e. $\kappa(\mathbf{x}, \mathbf{z})$ is large if $\mathbf{x}$ and $\mathbf{z}$ are similar. This notion can be useful when choosing or designing a kernel for a particular application domain.

There are a large number of existing kernel functions, and we briefly introduce some well-known ones. The simplest is the *linear kernel* defined as

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle,$$

with $\phi(\mathbf{x}) = \mathbf{x}$ in this case. The *polynomial kernel* is computed using

$$\kappa(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + b)^d, \tag{2.7}$$

where b is the *bias*, and d is called the *degree*. In the case that $b = 0$, $d = 2$ and with input points in $\mathbb{R}^2$, one maps the original data using

$$\phi(\mathbf{x}) : (x_1, x_2)' \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2)',$$

and the connection to Equation 2.7 can be seen as follows

$$\begin{aligned} \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= \langle \mathbf{x}, \mathbf{z} \rangle^2. \end{aligned}$$

To conclude this section on kernel methods, note that a kernel-defined feature space can also be of infinite dimension which is clearly an impossibility if one were to use explicit feature representations. One example of a kernel with infinite dimension is the Radial Basis Function (RBF) kernel, which is a decreasing function of the distance between two points. It is defined as

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right),$$

where σ is known as the *kernel width*. The RBF kernel can be shown to be infinite dimensional by using the Taylor expansion of the exponential function (Shawe-Taylor and Cristianini (2004)).

2.4 Unsupervised Approaches

This section introduces several unsupervised feature extraction approaches such as the popular PCA and Gram-Schmidt Orthonormalistion methods, and more recent techniques. These approaches are particularly relevant for the work presented in Chapter 4 on matrix approximation.

2.4.1 Principal Components Analysis

PCA has successfully been applied to image compression (Jain (1989)) and face recognition (Turk and Pentland (1991b)). It projects examples onto a subspace defined by a set of orthogonal vectors that maximise the variance of the data. Figure 2.3 shows the PCA projection directions for an example 2-dimensional dataset. The first direction follows that of maximum variance and the second is orthogonal to the initial one. Often the majority of the variance of the data can be captured using a much smaller dimensionality than that of the original space, hence the projections of the examples into the PCA subspace provide a good approximation. The residual variance can be seen as noise in the data, and is not useful for learning.

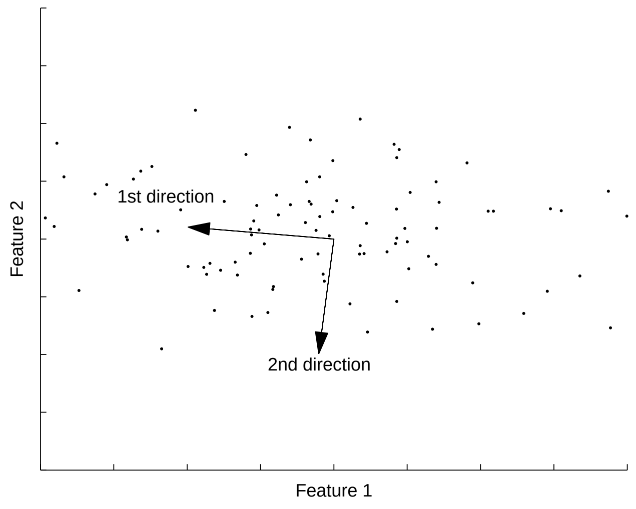


FIGURE 2.3: Plot of the PCA projection vectors for an example 2-dimensional dataset.

Since PCA maximises variance, we begin with the definition of the variance[3] of a zero mean random variable x

$$\operatorname{var}(x) = \mathbb{E}[(x - \mathbb{E}[x])^2] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \mathbb{E}[x^2],$$

[3]The term *input space variance* is used to refer to the variance of the original examples.

where the rightmost expression follows from $\mathbb{E}[x] = 0$. Now consider the variance of the projection of a zero mean multivariate random variable $\mathbf{x} \in \mathbb{R}^m$ projected onto a unit vector $\mathbf{u} \in \mathbb{R}^m$, $\mathbb{E}[\mathbf{u}'\mathbf{x}\mathbf{x}'\mathbf{u}]$. The corresponding empirical variance with respect to a sample of size ℓ is

$$\hat{\mathbb{E}}[\mathbf{u}'\mathbf{x}\mathbf{x}'\mathbf{u}] = \frac{1}{\ell}\sum_{i=1}^{\ell} \mathbf{u}'\mathbf{x}_i\mathbf{x}_i'\mathbf{u} = \frac{1}{\ell}\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u},$$

where $\mathbf{X}$ is a matrix with rows $\mathbf{x}_i$, $i = 1, \dots, \ell$. PCA finds the unit projection direction $\mathbf{u}$ to maximise the above quantity, i.e. it solves

$$\begin{array}{ll} \max & \mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u} \\ \text{s.t.} & \|\mathbf{u}\| = 1. \end{array}$$

Now shown, using the Lagrangian method, is that the solution to the above is the eigenvector of $\mathbf{X}'\mathbf{X}$ corresponding to the largest eigenvalue. The objective function of the above optimisation is

$$\mathcal{L}(\mathbf{u}) = \mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u} - \lambda\mathbf{u}'\mathbf{u},$$

where λ is a Lagrange multiplier. Differentiating and equating the derivative to zero yields the eigenvalue equation

$$\mathbf{X}'\mathbf{X}\mathbf{u} = \lambda\mathbf{u}. \tag{2.8}$$

Notice that $\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u} = \lambda\mathbf{u}'\mathbf{u} = \lambda$ hence λ/ℓ is the empirical variance of the data. It follows that to maximise the variance one must choose $\mathbf{u}$ to be the eigenvector of $\mathbf{X}'\mathbf{X}$ corresponding to the largest eigenvalue. In order to compute k directions, one must find eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ corresponding to the k largest eigenvalues $\lambda_1, \dots, \lambda_k$, $\lambda_1 \geq \lambda_2 \dots \geq \lambda_k$. The projections of the data onto these directions, given by $\mathbf{X}\mathbf{u}_1, \dots, \mathbf{X}\mathbf{u}_k$, are called the *Principal Components*. Furthermore, the projection of a new test point $\phi(\mathbf{x})$ is given by

$$\hat{\phi}(\mathbf{x})' = \phi(\mathbf{x})'\mathbf{U},$$

where $\mathbf{U}$ is a matrix whose columns are $\mathbf{u}_1, \dots, \mathbf{u}_k$. Notice that the projection directions are orthogonal since they are the eigenvectors of a symmetric matrix. Additionally, the principal components are orthogonal since $\mathbf{u}_i'\mathbf{X}'\mathbf{X}\mathbf{u}_j = \lambda_i\mathbf{u}_i'\mathbf{u}_j = 0$ and $\lambda_i \neq 0$.

A direct method of computing the projection matrix for PCA is through the eigen-decomposition of $\mathbf{X}'\mathbf{X}$. However, there is also an iterative approach to finding the PCA projection directions. Let $\mathbf{X}_1 = \mathbf{X}$, then at iteration j, $\mathbf{u}_j$ is the eigenvector corresponding to the largest eigenvalue of $\mathbf{X}_j'\mathbf{X}_j$, where $\mathbf{X}_j$ is called the jth *residual matrix*. The residual matrices are computed using a process known as *deflation*, which is an orthogonal projection into a subspace. As a general point on terminology, a residual matrix or example is that obtained by projection into an orthogonal subspace. In the PCA case, the rows of $\mathbf{X}_j$ are projected into the subspace orthogonal to $\mathbf{u}_j$,

$$\mathbf{X}_{j+1} = \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j} \right), \tag{2.9}$$

and the process of selecting a direction and deflating is repeated for the desired number of iterations (see Algorithm 1). As one is extracting the features iteratively, techniques such as the Power method (Strang (2003)) can be used to efficiently extract the first eigenvector at each iteration.

Algorithm 1 Pseudo code for iterative PCA.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, dimension $k$
Process:
  1. $\mathbf{X}_1 = \mathbf{X}$
  2. For $j = 1, \ldots, k$
     (a) Select $\mathbf{u}_j$ as the first eigenvector of $\mathbf{X}_j'\mathbf{X}_j$
     (b) $\mathbf{X}_{j+1} = \mathbf{X}_j \left( \mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j} \right)$
  3. End
Output: Projection directions $\mathbf{u}_j$ and features $\mathbf{X}\mathbf{u}_j$, $j = 1, \ldots, k$
```

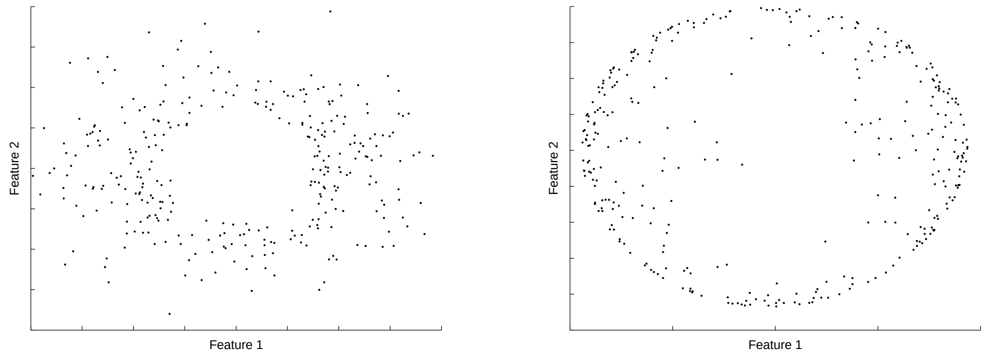
It may not be immediately clear that Algorithm 1 finds the first k eigenvectors of $\mathbf{X}'\mathbf{X}$. However, the effect of the deflation of Equation 2.9 is to set the dominant eigenvalue of $\mathbf{X}_{j+1}'\mathbf{X}_{j+1}$ to zero,

$$\begin{aligned}
\mathbf{X}_{j+1}'\mathbf{X}_{j+1} &= \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j} \right) \mathbf{X}_j'\mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j} \right) \\
&= \mathbf{X}_j'\mathbf{X}_j - \frac{\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j}{\mathbf{u}_j'\mathbf{u}_j} - \frac{\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j} + \frac{\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{(\mathbf{u}_j'\mathbf{u}_j)^2} \\
&= \mathbf{X}_j'\mathbf{X}_j - \frac{\lambda_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j},
\end{aligned}$$

since $\mathbf{X}'_{j+1}\mathbf{X}_{j+1}\mathbf{u}_j = \mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j - \lambda_j\mathbf{u}_j = \mathbf{0}$ and $\mathbf{u}_j$ is a non-zero vector, it follows that the eigenvalue of $\mathbf{X}'_{j+1}\mathbf{X}_{j+1}$ corresponding to $\mathbf{u}_j$ is zero. The remaining eigenvalues are left intact since for $i \neq j$, $\mathbf{X}'_{j+1}\mathbf{X}_{j+1}\mathbf{u}_i = \mathbf{X}'_j\mathbf{X}_j\mathbf{u}_i$ due to the orthogonality of the eigenvectors.

2.4.1.1 Dual Form

Kernel PCA (KPCA) was introduced in Schölkopf et al. (1998) to address the problem of finding non-linear principal components. Its application to a toy dataset is shown in Figure 2.4. Figure 2.4(a) displays the original data, and Figure 2.4(b) shows the features extracted by KPCA in an RBF feature space with $\sigma = 1.2$. One can see that the variance of the data has been reduced by projecting the points onto an ellipse. It is not always possible to discover this line as the projections directions for KPCA exist in a kernel-defined feature space and their representation in the input space may not exist.



(a) Original features (b) Extracted features

FIGURE 2.4: The application of KPCA to a toy dataset using the RBF kernel.

The derivation of KPCA relies on the observation that in the primal case the eigenvectors of the covariance matrix $\mathbf{X}'\mathbf{X}$ lie in the row space of $\mathbf{X}$ since $\mathbf{u} = (1/\lambda)\mathbf{X}'(\mathbf{X}\mathbf{u})$. Hence the primal projection vector can be written in the form $\mathbf{u} = \mathbf{X}'\boldsymbol{\alpha}$ for some dual projection vector $\boldsymbol{\alpha}$. It follows from Equation 2.8 that

$$\mathbf{X}'\mathbf{X}\mathbf{X}'\boldsymbol{\alpha} = \lambda\mathbf{X}'\boldsymbol{\alpha} \tag{2.10}$$

$$\Rightarrow \mathbf{K}\mathbf{K}\boldsymbol{\alpha} = \lambda\mathbf{K}\boldsymbol{\alpha}, \tag{2.11}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ is the kernel matrix. Schölkopf et al. (1998) shows that this is equivalent to finding the solutions of

$$\mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha} \tag{2.12}$$

for non-zero eigenvalues, and their argument is summarised here. Let the orthogonal eigenvectors of $\mathbf{K}$ be denoted by $\boldsymbol{\beta}_i$, with corresponding eigenvalues γ_i, $i = 1, \ldots, \ell$. Suppose $\boldsymbol{\alpha}$, λ satisfies Equation 2.11, then $\boldsymbol{\alpha}$ can be written as $\boldsymbol{\alpha} = \sum_{i=1}^{\ell} a_i \boldsymbol{\beta}_i$ (since $\boldsymbol{\alpha}$ exists in the space of $\mathbf{K}$ spanned by its eigenvectors) and Equation 2.11 becomes

$$\sum_{i=1}^{\ell} \gamma_i^2 a_i \boldsymbol{\beta}_i = \lambda \sum_{i=1}^{\ell} \gamma_i a_i \boldsymbol{\beta}_i,$$

which implies

$$\gamma_i^2 a_i = \lambda \gamma_i a_i,$$

for $i = 1, \ldots, \ell$. Hence, one of the following conditions must be true to satisfy the above equality: $\gamma_i = \lambda$ or $a_i = 0$ or $\gamma_i = 0$. Now assume that $\boldsymbol{\alpha}$ and λ satisfy Equation 2.12. Using a similar argument,

$$\sum_{i=1}^{\ell} \gamma_i a_i \boldsymbol{\beta}_i = \lambda \sum_{i=1}^{\ell} a_i \boldsymbol{\beta}_i,$$

which implies

$$\gamma_i a_i = \lambda a_i.$$

To satisfy this equality either $a_i = 0$ or $\gamma_i = \lambda$. Hence all solutions to Equation 2.12 are also solutions of Equation 2.11, and the additional solutions have zero eigenvalues. Since the eigenvalues of $\mathbf{K}$ correspond to the input svariance, eigenvectors with zero eigenvalues have no relevance and Equation 2.12 supplies all of the required directions.

Recall from the primal case that the extracted features are given by the projections of the examples onto the directions, hence $\mathbf{X}\mathbf{u}_j = \mathbf{X}\mathbf{X}'\boldsymbol{\alpha}_j = \mathbf{K}\boldsymbol{\alpha}_j$, $j = 1, \ldots, k$. Furthermore, since $\mathbf{u}_j$ is a unit vector, $\boldsymbol{\alpha}_j$ must be scaled using

$$\boldsymbol{\alpha}_j \leftarrow \frac{\boldsymbol{\alpha}_j}{\sqrt{\boldsymbol{\alpha}_j'\mathbf{K}\boldsymbol{\alpha}_j}} = \frac{\boldsymbol{\alpha}_j}{\sqrt{\lambda_j}}.$$

It follows that the expression for the projection of a new test point is

$$\hat{\phi}(\mathbf{x})' = \mathbf{k}'\mathbf{A}\Lambda^{-1/2},$$

where $\mathbf{A}$ is a matrix whose columns are $\boldsymbol{\alpha}_j$, $j = 1, \ldots, k$, and $\mathbf{k}$ is a vector of inner products between the test and training examples.

In the primal case one can compute the eigenvectors of the covariance matrix using deflation and the same technique can be applied for KPCA. By using Equation 2.9 with $\mathbf{K}_j = \mathbf{X}_j\mathbf{X}_j'$, one obtains

$$\begin{aligned}
\mathbf{K}_{j+1} &= \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j}\right)^2 \mathbf{X}_j' && (2.13)\\
&= \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j}\right) \mathbf{X}_j' && (2.14)\\
&= \mathbf{X}_j\mathbf{X}_j' - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'}{\mathbf{u}_j'\mathbf{u}_j} && (2.15)\\
&= \mathbf{K}_j - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j} && (2.16)\\
&= \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j, && (2.17)
\end{aligned}$$

where $\boldsymbol{\alpha}_j$ is the dominant eigenvector of $\mathbf{K}_j$. It is then simple to show that this deflation sets the dominant eigenvalue in $\mathbf{K}_j$ to zero since,

$$\begin{aligned}
\mathbf{K}_{j+1} &= \mathbf{K}_j - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\\
&= \mathbf{K}_j - \frac{\lambda_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\boldsymbol{\alpha}_j}.
\end{aligned}$$

The other eigenvectors and eigenvalues are left intact since the eigenvectors corresponding to distinct eigenvalues are orthogonal. The iterative KPCA algorithm is shown in Algorithm 2.

2.4.2 Gram-Schmidt Orthogonalisation

We now introduce the well-known Gram-Schmidt method (Poole (2003)) which orthogonalises a set of vectors. In machine learning it is often used to orthogonalise a set of examples, but it can also approximate a data matrix. Gram-Schmidt is an iterative

Algorithm 2 Pseudo code for iterative KPCA.

Inputs: Kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell \times \ell}$, dimension k

Process:

```
1. $\mathbf{K}_1 = \mathbf{K}$
2. For $j = 1, \ldots, k$
    (a) Select $\boldsymbol{\alpha}_j$, $\lambda_j$, the first eigenvector-eigenvalue pair of $\mathbf{K}_j$
    (b) $\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{K}_j \boldsymbol{\alpha}_j \boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j' \mathbf{K}_j \boldsymbol{\alpha}_j}\right) \mathbf{K}_j$
3. End
```

Output: Dual projection directions $\boldsymbol{\alpha}_j$ and features $\mathbf{K}\boldsymbol{\alpha}_j/\sqrt{\lambda_j}$, $j = 1, \ldots, k$

method, and at the jth iteration the rows of the residual matrices are projected into the space orthogonal to $\mathbf{u}_j$, i.e. one uses the PCA deflation (Equation 2.9). In this case $\mathbf{u}_j = \mathbf{X}_j' \boldsymbol{\alpha}_j$ with

$$\boldsymbol{\alpha}_j = \frac{\mathbf{e}_j}{\sqrt{\mathbf{e}_j' \mathbf{X}_j \mathbf{X}_j' \mathbf{e}_j}}, \tag{2.18}$$

where $\mathbf{e}_j$ is the jth standard vector (which has a 1 at the jth position and zeros elsewhere) hence $\mathbf{u}_j$ is a scalar multiple of the jth deflated example. Notice that the PCA deflation removes from the rows of $\mathbf{X}_j$ the component in the direction of $\mathbf{u}_j$, which implies that $\mathbf{u}_j' \mathbf{u}_i = 0$, $i \neq j$. The orthogonality of the projection directions in conjunction with the PCA deflation can be seen by writing

$$\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{X}_j \mathbf{u}_j \boldsymbol{\alpha}_j'}{\mathbf{u}_j' \mathbf{u}_j}\right) \mathbf{X}_j,$$

and hence, for $i < j$,

$$\begin{aligned}
\mathbf{u}_j' \mathbf{u}_i &= \boldsymbol{\alpha}_j' \mathbf{X}_j \mathbf{u}_i \\
&= \boldsymbol{\alpha}_j' \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \boldsymbol{\alpha}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{u}_{j-1}}\right) \cdots \left(\mathbf{I} - \frac{\mathbf{X}_i \mathbf{u}_i \boldsymbol{\alpha}_i'}{\mathbf{u}_i' \mathbf{u}_i}\right) \mathbf{X}_i \mathbf{u}_i \\
&= \boldsymbol{\alpha}_j' \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \boldsymbol{\alpha}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{u}_{j-1}}\right) \cdots \left(\mathbf{X}_i \mathbf{u}_i - \frac{\mathbf{X}_i \mathbf{u}_i \mathbf{u}_i' \mathbf{u}_i}{\mathbf{u}_i' \mathbf{u}_i}\right) \\
&= 0.
\end{aligned}$$

It follows that after k iterations the first k residual examples are orthogonalised. In addition, one can use the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ as projection directions in which case Gram-Schmidt functions as a matrix approximation method.

Algorithm 3 Pseudo code for Gram-Schmidt Orthogonalisation.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, dimension $k$
Process:
  1. $\mathbf{X}_1 = \mathbf{X}$
  2. For $j = 1, \ldots, k$
      (a) Let $\mathbf{u}_j = \mathbf{X}_j'\mathbf{e}_i / \sqrt{\mathbf{e}_i'\mathbf{X}_j\mathbf{X}_j'\mathbf{e}_i}$ where $i$ is index of example with maximum norm
      (b) $\mathbf{X}_{j+1} = \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j}\right)$
  3. End
Output: Orthogonalised examples $\mathbf{u}_j$ and features $\mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$
```

In machine learning, a common strategy is to use the example with the maximum norm as the projection direction which requires a search through ℓ examples. Given that each deflation costs $O(\ell m)$, it follows that the training complexity of k iterations of this approach to Gram-Schmidt is $O(\ell m k)$. The complete Gram-Schmidt method is shown in Algorithm 3.

2.4.2.1 Dual form

It is useful to be able to compute the projection of the examples in a kernel-defined feature space, which is precisely the aim of Kernel Gram-Schmidt (KGS, Cristianini et al. (2002)). Let $\mathbf{K}_1 = \mathbf{K}$, then from Equation 2.9 it follows that the kernel matrix should be deflated in an identical manner to that of KPCA (Equation 2.17). Furthermore, the dual projection direction is a multiple of the ith standard vector where i is the index of the maximum diagonal entry of the residual kernel matrix. Hence from Equation 2.18,

$$\boldsymbol{\alpha}_j = \frac{\mathbf{e}_i}{\sqrt{\mathbf{e}_i'\mathbf{K}_j\mathbf{e}_i}}. \tag{2.19}$$

In the primal case, one computes the projections of the examples onto $\mathbf{u}_j$, $j = 1, \ldots, k$, however these vectors are not available in this case. Let $\mathbf{A}$ be the matrix with columns

$$\mathbf{a}_j = \prod_{i=1}^{j-1} \left(\mathbf{I} - \frac{\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\boldsymbol{\alpha}_j,$$

where the product is left to right, then $\mathbf{u}_j = \mathbf{X}'\mathbf{a}_j$, $j = 1, \ldots, k$. This implies $\mathbf{U} = \mathbf{X}'\mathbf{A}$ and the projection of a new test point is given by

$$\hat{\phi}(\mathbf{x})' = \mathbf{k}'\mathbf{A}.$$

Notice that $\mathbf{a}_j$ is computed using only those examples which have previously been used for deflation, hence it contains j non-zero entries. This implies that the computation of $\hat{\phi}(\mathbf{x})$ requires k kernel evaluations. Furthermore, given that the deflation of the kernel matrix is $O(\ell^2)$, KGS has a training complexity of $O(k\ell^2)$.

Algorithm 4 Pseudo code for Kernel Gram-Schmidt Orthogonalisation.

```
Inputs: Kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell \times \ell}$, dimension $k$
Process:
  1. $\mathbf{K}_1 = \mathbf{K}$
  2. For $j = 1, \ldots, k$
      (a) Let $\boldsymbol{\alpha}_j = \mathbf{e}_i / \sqrt{\mathbf{e}_i'\mathbf{K}_j\mathbf{e}_i}$ where $i$ is index of maximum diagonal entry of $\mathbf{K}_j$
      (b) $\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j$
  3. End
Output: Dual directions $\boldsymbol{\alpha}_j$ and features $\mathbf{K}_j\boldsymbol{\alpha}_j$, $j = 1, \ldots, k$
```

2.4.2.2 Cholesky Decomposition

Another way of viewing Gram-Schmidt Orthogonalisation is as the QR-decomposition of $\mathbf{X}$, i.e. $\mathbf{X}' = \mathbf{QR}$, where $\mathbf{Q}$ has columns $\mathbf{u}_j$, $j = 1, \ldots, k$, k is the rank of $\mathbf{X}$ and $\mathbf{R}$ is an upper triangular matrix. The matrix $\mathbf{R}$ is a representation of each example in the basis defined by $\mathbf{u}_1, \ldots, \mathbf{u}_k$. It follows that $\mathbf{R}$ can be found by projecting $\mathbf{X}$ onto $\mathbf{Q}$, since $\mathbf{XQ} = \mathbf{R}'\mathbf{Q}'\mathbf{Q} = \mathbf{R}'$. Furthermore, if one considers the kernel matrix given by $\mathbf{K} = \mathbf{XX}'$, then $\mathbf{K} = \mathbf{R}'\mathbf{Q}'\mathbf{QR} = \mathbf{R}'\mathbf{R}$, which is the Cholesky decomposition of a positive semidefinite matrix, see Shawe-Taylor and Cristianini (2004) for further details.

The complete Cholesky decomposition of a kernel matrix provides an exact representation of the examples. One can also obtain an approximation of the kernel matrix by projecting into $\mathbf{u}_1, \ldots, \mathbf{u}_j$, $j < k$, using the Incomplete Cholesky decomposition (ICD, Bach and Jordan (2003)). ICD has the decomposition $\mathbf{K} \approx \mathbf{R}_j\mathbf{R}_j'$ where $\mathbf{R}_j \in \mathbb{R}^{\ell \times j}$ and j is less than the rank of $\mathbf{K}$. The algorithm picks a column of $\mathbf{K}$ at a time, greedily maximising a lower bound on the reduction in the error of the approximation. It can be efficiently implemented at $O(\ell j^2)$ complexity since it uses a comparison of the diagonal elements of $\mathbf{K} - \mathbf{R}_i\mathbf{R}_i'$ at the ith iteration to pick a kernel matrix column. A variation of

ICD is given in Bach and Jordan (2005), which at the same computational complexity, approximates a kernel matrix using the labels as guidance.

2.4.3 Recent Advances

KPCA is an effective approach for capturing the variance of a set of examples, however the resulting solution is not sparse in the sense that the projection of a test point requires ℓ kernel evaluations. Using more examples in KPCA implies better approximations, however this also results in reduced efficiency in computing projections. Several authors have suggested sparse variants of KPCA to overcome this limitation. In Tipping (2001) a sparse KPCA algorithm is derived by approximating the covariance matrix with a reduced number of examples using a maximum likelihood approach. Jolliffe et al. (2003) proposes ScoTLASS which uses an upper bound on the 1-norm of the projection directions, leading to a non-convex problem. SPCA (Zou et al. (2006)) is another 1-norm penalised algorithm, which targets directions towards regression. It can be solved efficiently using a method known as least angle regression.

All of d'Aspremont et al. (2005), Moghaddam et al. (2006b) and Sriperumbudur et al. (2007) consider the PCA optimisation with a stricter zero-norm[4] constraint on the projection vectors. In d'Aspremont et al. (2005), this optimisation is relaxed to give a convex semidefinite programming formulation. Moghaddam et al. (2006b) provides both greedy and exact methods for solving the cardinality constrained optimisation using insights between the eigenvalues of a positive definite covariance matrix and the eigenvalues of its submatrices. Sriperumbudur et al. (2007) approximates the zero-norm using $\|\mathbf{u}\|_0 \approx \sum_i \log(\epsilon + |\mathbf{u}_i|)$ where $0 \leq \epsilon \ll 1$ avoids problems when $\mathbf{u}_i$ is zero. The resulting optimisation is framed as a difference of convex functions program and solved as a sequence of locally convex programs. Additional directions are found by deflating the covariance matrix.

Williams and Seeger (2000b) outlines an approach to approximate the kernel matrix based on the Nyström method. Let $\mathbf{A}[I_r, I_c]$ be the submatrix of $\mathbf{A}$ composed of the rows indexed by I_r and columns indexed by I_c, and $\mathbf{A}[I_r,]$ and $\mathbf{A}[, I_c]$ be the rows and columns of $\mathbf{A}$ indexed by I_r and I_c respectively. The Nyström approximation is given by

$$\tilde{\mathbf{K}}_k = \mathbf{K}[, I]\mathbf{K}[I, I]^{-1}\mathbf{K}[I,],$$

where $I \in [\ell]^k$ is a set of k random indices. This approximation can be computed in $O(k^2\ell)$ operations, and follows from an analysis of the eigenfunctions of the kernel operator.

[4]The zero-norm, though not a true norm, is the number of non-zero elements in a vector.

An orthogonalisation-based variant of KPCA is presented in Smola et al. (1999), and in Chapter 4 it will become clear how it is related to our approach for feature extraction. Essentially the method finds a projection direction which maximises variance and then orthogonalises the examples by projecting them into the space orthogonal to the direction, i.e. using Equation 2.17. Each direction is chosen to be a scalar multiple of a single deflated example, hence is sparse. A related paper, Smola and Schölkopf (2000), is concerned with sparse greedy matrix approximation methods. A compact approximation of the kernel matrix is computed using a subset of kernel matrix columns or basis functions (kernel evaluations of the form $\kappa(\mathbf{x}, \cdot)$ for a fixed $\mathbf{x}$). With the column space approximation method, the kernel matrix is deflated in the following way

$$\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{K}_j \boldsymbol{\alpha}_j \boldsymbol{\alpha}_j' \mathbf{K}_j'}{\boldsymbol{\alpha}_j' \mathbf{K}_j' \mathbf{K}_j \boldsymbol{\alpha}_j} \right) \mathbf{K}_j,$$

so that the columns of $\mathbf{K}_{j+1}$ are orthogonal to $\mathbf{K}_j \boldsymbol{\alpha}_j$.

Franc and Hlavác (2006) derives a different sparse KPCA approach called Greedy KPCA (GKPCA). Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ be the training examples then GKPCA tries to find a subset S' of S which can be used to compute an approximation of the data. This approximation is given by $\hat{\mathbf{X}} = \mathbf{X}\mathbf{U}$ where the columns of $\mathbf{U}$ are linear combinations of the examples in S'. As implied by its name, GKPCA uses a greedy approach to select the examples in S'. One simply chooses the example which minimises the residual approximation error, deflates using the PCA deflation method, and repeats until the desired number of examples are selected. On a selection of benchmark datasets, GKPCA is shown to approximate the data with significantly larger error than KPCA, although the difference between the two methods decreases as the number of iterations approaches ℓ as one might expect.

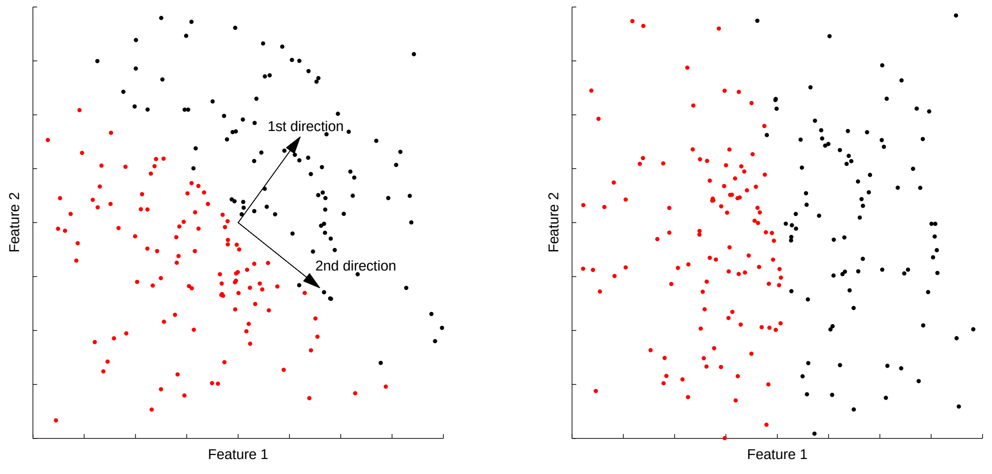
2.5 Supervised Approaches

A common use of feature extraction is as a step before classification or regression. Many practitioners use PCA in this way, however one can in general improve upon PCA if the labels are utilised. This section provides an introduction to supervised feature extraction which covers several important techniques. In particular, an in depth coverage of PLS is given since it is one of the core components of this thesis.

2.5.1 Partial Least Squares

PLS has enjoyed success in chemometrics where high dimensional and correlated representations are common, and has recently gained favour within the machine learning community. It iteratively extracts features which are most covariant with the labels and then performs least squares regression on the extracted features. Since its appearance, PLS has since seen many variants, for example PLS Mode A (Wold (1975)), PLS-SB (Sampson et al. (1989)) and in Barker and Rayens (2003) which tailors PLS for classification. General surveys of PLS variants are presented in Rosipal and Kramer (2006); Wegelin (2000). Furthermore, Bennett and Embrechts (2003) gives an optimisation perspective on PLS and its kernel variant. The approach we outline here is often called PLS2 or PLS regression, however we will simply refer to it as PLS.

The application of PLS feature extraction to a simple 2-dimensional, binary labelled dataset is shown in Figure 2.5. The red points represent positively labelled examples and the black ones are negatively labelled examples. The directions in Figure 2.5(a) are those of maximum covariance and from Figure 2.5(b) one can see that the first of the extracted features is able to discriminate the positive and negative examples almost perfectly.



(a) Original features (b) Extracted features

FIGURE 2.5: Plot of the PLS projection vectors for an example 2-dimensional binary labelled dataset.

We start our coverage of PLS with a definition of the covariance of two zero-mean random variables x and y,

$$\begin{aligned} \mathrm{cov}(x, y) &= \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] \\ &= \mathbb{E}[xy], \end{aligned}$$

which follows from $\mathbb{E}[x] = \mathbb{E}[y] = 0$. In PLS, one projects an observation $\mathbf{x} \in \mathbb{R}^m$ onto direction $\mathbf{u}$, and the corresponding vector of labels $\mathbf{y} \in \mathbb{R}^n$ onto $\mathbf{v}$. The covariance of these projections is

$$\mathrm{cov}(\mathbf{x}'\mathbf{u}, \mathbf{y}'\mathbf{v}) = \mathbb{E}[\mathbf{x}'\mathbf{u}\mathbf{y}'\mathbf{v}].$$

Now consider the empirical covariance of a set of observations $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$,

$$\hat{\mathbb{E}}[\mathbf{x}'\mathbf{u}\mathbf{y}'\mathbf{v}] = \frac{1}{\ell}\sum_{i=1}^{\ell} \mathbf{u}'\mathbf{x}_i\mathbf{y}_i'\mathbf{v} = \frac{1}{\ell}\mathbf{u}'\mathbf{X}'\mathbf{Y}\mathbf{v},$$

where $\mathbf{Y}$ has rows[5] $\mathbf{y}_i$, $i = 1\ldots,\ell$. This leads us to the PLS optimisation, which maximises the above quantity subject to unit norm projection vectors,

$$\begin{array}{ll} \max & \mathbf{u}'\mathbf{X}'\mathbf{Y}\mathbf{v} \\ \text{s. t.} & \|\mathbf{u}\| = 1 \\ & \|\mathbf{v}\| = 1. \end{array} \tag{2.20}$$

The unit norm constraints on $\mathbf{u}$ and $\mathbf{v}$ are required to avoid trivial solutions. The solutions for $\mathbf{u}$ and $\mathbf{v}$ are the left and right singular vectors corresponding to the largest singular value of the covariance matrix $\mathbf{X}'\mathbf{Y}$. This can be seen using the Lagrangian approach,

$$\mathcal{L}(\mathbf{u}, \mathbf{v}) = \mathbf{u}'\mathbf{X}'\mathbf{Y}\mathbf{v} - \frac{1}{2}\sigma_{\mathbf{u}}(\mathbf{u}'\mathbf{u} - 1) - \frac{1}{2}\sigma_{\mathbf{v}}(\mathbf{v}'\mathbf{v} - 1),$$

where $\sigma_{\mathbf{u}}$ and $\sigma_{\mathbf{v}}$ are Lagrange multipliers. Differentiating with respect to $\mathbf{u}$ and $\mathbf{v}$ and equating to zero gives,

$$\begin{aligned} \mathbf{X}'\mathbf{Y}\mathbf{v} &= \sigma_{\mathbf{u}}\mathbf{u} \\ \mathbf{Y}'\mathbf{X}\mathbf{u} &= \sigma_{\mathbf{v}}\mathbf{v}. \end{aligned}$$

Notice that premultiplying the above pair of equations by $\mathbf{u}$ and $\mathbf{v}$ respectively results in $\mathbf{u}'\mathbf{X}'\mathbf{Y}\mathbf{v} = \sigma_{\mathbf{u}}\mathbf{u}'\mathbf{u}$ and $\mathbf{v}'\mathbf{Y}'\mathbf{X}\mathbf{u} = \sigma_{\mathbf{v}}\mathbf{v}'\mathbf{v}$, hence $\sigma_{\mathbf{u}} = \sigma_{\mathbf{v}} = \sigma$. Combining this information with the above equations,

[5]For the classification variant of PLS, $\mathbf{Y}$ is an indicator matrix and is substituted with $\tilde{\mathbf{Y}} = \mathbf{Y}(\mathbf{Y}'\mathbf{Y})^{-1/2}$ so that $\tilde{\mathbf{Y}}'\tilde{\mathbf{Y}} = \mathbf{I}$.

$$\begin{aligned} \mathbf{X}'\mathbf{Y}\mathbf{Y}'\mathbf{X}\mathbf{u} &= \sigma^2\mathbf{u} && (2.21) \\ \mathbf{Y}'\mathbf{X}\mathbf{X}'\mathbf{Y}\mathbf{v} &= \sigma^2\mathbf{v}, && (2.22) \end{aligned}$$

which implies from Equations 2.5 and 2.6 that $\mathbf{u}$ and $\mathbf{v}$ are the left and right singular vectors of $\mathbf{X}'\mathbf{Y}$. Since σ corresponds to the quantity being maximised, it follows that one must choose those singular vectors that are paired with the largest singular value of $\mathbf{X}'\mathbf{Y}$.

After choosing a projection direction, one deflates the data matrix by projecting onto the space orthogonal to $\mathbf{X}_j\mathbf{u}_j$ at the jth iteration,

$$\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\mathbf{X}_j, \quad (2.23)$$

where $\mathbf{u}_j$ is the maximum left singular vector of $\mathbf{X}_j'\mathbf{Y}$. One need not deflate the $\mathbf{Y}$ matrix in a similar fashion since its deflation has no effect on the chosen directions, however in the dual formulation the deflation of $\mathbf{Y}$ is necessary. Note that in general the PLS directions do not correspond to the eigenvectors of Equation 2.21. However, Hoskuldsson (1988) shows that the first singular value of $\mathbf{X}_{j+1}'\mathbf{Y}$ is greater than or equal to the second singular value of $\mathbf{X}_j'\mathbf{Y}$.

The are several interesting properties of the PLS directions and projections. In Phatak and de Hoog (2002) the authors show that PLS is identical to the conjugate gradient method for solving a set of linear equations whose matrix is symmetric and positive definite. The same paper also demonstrates a connection between PLS and the Lanczos method for approximating the dominant eigenvalues of a symmetric matrix. One of the key properties of the PLS deflation is that $\mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$, are orthogonal since they are the projections of the residual matrices which have been previously deflated by $\mathbf{X}_i\mathbf{u}_i$, $i = 1, \ldots, j-1$. More formally,

$$\begin{aligned} \mathbf{X}_{j+1}'\mathbf{X}_i\mathbf{u}_i &= \left(\mathbf{I} - \frac{\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\mathbf{X}_j'\mathbf{X}_i\mathbf{u}_i \\ &= \left(\mathbf{I} - \frac{\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\cdots\left(\mathbf{I} - \frac{\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i\mathbf{u}_i'}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i}\right)\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i \\ &= \left(\mathbf{I} - \frac{\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\cdots\left(\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i - \frac{\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i(\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i)}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i}\right) \\ &= \mathbf{0}, \end{aligned}$$

for $i = 1, \ldots, j$. The first line is a rearrangement of the deflation of Equation 2.23 so that $\mathbf{X}_j$ is deflated by its rows. Note that the projections $\mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$, are also the final PLS features, and their orthogonality implies that they are uncorrelated.

Once PLS extracts features it uses them for least squares regression, whose optimisation is

$$\min \|\mathbf{XC} - \mathbf{Y}\|_F^2,$$

where $\mathbf{C} \in \mathbb{R}^{m \times n}$ is a matrix of regression coefficients and $\|\mathbf{A}\|_F = \sqrt{\mathrm{tr}(\mathbf{A}'\mathbf{A})}$ is the Frobenius norm. Solving by differentiating and equating to zero yields $\mathbf{C} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$ (assuming $\mathbf{X}'\mathbf{X}$ is invertible), but the inversion of $\mathbf{X}'\mathbf{X}$ is computationally expensive if $\mathbf{X}$ has many features. In the PLS case, the regression coefficients are $\hat{\mathbf{C}} = (\hat{\mathbf{X}}'\hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}'\mathbf{Y}$ where $\hat{\mathbf{X}} = [\mathbf{X}_1\mathbf{u}_1 \cdots \mathbf{X}_k\mathbf{u}_k]$ is the matrix of new features. This is more efficient than using the original features since $\hat{\mathbf{X}}$ has fewer columns than $\mathbf{X}$ and $\hat{\mathbf{X}}'\hat{\mathbf{X}}$ is a diagonal matrix hence is simple to invert. As well as gaining these efficiency improvements, using the PLS features can also be seen as a form of regularisation since the examples are projected into a low dimensional subspace. The PLS pseudo code is given in Algorithm 5.

Algorithm 5 Pseudo code for PLS regression.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, target vectors $\mathbf{Y} \in \mathbb{R}^{\ell \times n}$, dimension $k$
Process:
  1. $\mathbf{X}_1 = \mathbf{X}$
  2. For $j = 1, \ldots, k$
       (a) Select $\mathbf{u}_j$ as the first singular vector of $\mathbf{X}_j'\mathbf{Y}$
       (b) $\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\mathbf{X}_j$
  3. End
  4. Compute regression coefficients $\hat{\mathbf{C}} = (\hat{\mathbf{X}}'\hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}'\mathbf{Y}$ where $\hat{\mathbf{X}} = [\mathbf{X}_1\mathbf{u}_1 \cdots \mathbf{X}_k\mathbf{u}_k]$
Output: Directions $\mathbf{u}_j$, features $\mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$, and coefficients $\hat{\mathbf{C}}$
```

2.5.1.1 Projection of a Test Example

The PLS features are given in terms of the projections of the residual examples into a subspace, however it would be useful to discover the transformation on the original examples. This transformation, first derived in Manne (1987), allows one to find the projection of a new test point for example. A simple rearrangement of Equation 2.23 results in

$$\mathbf{X}_{j+1} = \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j \mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j} \right),$$

which can be applied to a single test point $\phi(\mathbf{x})$ for k iterations as follows

$$\phi(\mathbf{x})_{k+1}' = \phi(\mathbf{x})_k' \left(\mathbf{I} - \frac{\mathbf{u}_k \mathbf{u}_k' \mathbf{X}_k' \mathbf{X}_k}{\mathbf{u}_k' \mathbf{X}_k' \mathbf{X}_k \mathbf{u}_k} \right),$$

where $\phi(\mathbf{x})_1 = \phi(\mathbf{x})$. Let $\mathbf{p}_j = \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j / \mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j$, $j = 1, \ldots, k$, then the above expression can be written as

$$\begin{aligned} \phi(\mathbf{x})_{k+1}' &= \phi(\mathbf{x})_k' \left(\mathbf{I} - \mathbf{u}_k \mathbf{p}_k' \right) \\ &= \phi(\mathbf{x})_k' - \phi(\mathbf{x})_k' \mathbf{u}_k \mathbf{p}_k' \\ &= \phi(\mathbf{x})' - \sum_{j=1}^{k} \phi(\mathbf{x})_j' \mathbf{u}_j \mathbf{p}_j', \end{aligned}$$

where for the last line, we have substituted the left hand side of the equality into the first term on the right hand side. The final feature vector has components given by $\hat{\phi}(\mathbf{x}) = \left(\phi(\mathbf{x})_j' \mathbf{u}_j \right)_{j=1}^{k}$. Consider using the inner products between $\phi(\mathbf{x})$ and the matrix $\mathbf{U}$ which has columns $\mathbf{u}_j$, $j = 1, \ldots, k$,

$$\begin{aligned} \phi(\mathbf{x})_{k+1}' \mathbf{U} &= \phi(\mathbf{x})' \mathbf{U} - \sum_{j=1}^{k} \phi(\mathbf{x})_j' \mathbf{u}_j \mathbf{p}_j' \mathbf{U} && (2.24) \\ &= \phi(\mathbf{x})' \mathbf{U} - \hat{\phi}(\mathbf{x})' \mathbf{P}' \mathbf{U}, && (2.25) \end{aligned}$$

where $\mathbf{P}$ is the matrix with columns $\mathbf{p}_j$, $j = 1, \ldots, k$. Observe that $\mathbf{X}_j \mathbf{u}_i = \mathbf{0}$, $i < j$, since

$$\begin{aligned} \mathbf{X}_j \mathbf{u}_i &= \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \mathbf{u}_{j-1}' \mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{X}_{j-1}' \mathbf{X}_{j-1} \mathbf{u}_{j-1}} \right) \mathbf{X}_j \mathbf{u}_i \\ &= \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \mathbf{u}_{j-1}' \mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{X}_{j-1}' \mathbf{X}_{j-1} \mathbf{u}_{j-1}} \right) \cdots \left(\mathbf{I} - \frac{\mathbf{X}_i \mathbf{u}_i \mathbf{u}_i' \mathbf{X}_i'}{\mathbf{u}_i' \mathbf{X}_i' \mathbf{X}_i \mathbf{u}_i} \right) \mathbf{X}_i \mathbf{u}_i \\ &= \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \mathbf{u}_{j-1}' \mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{X}_{j-1}' \mathbf{X}_{j-1} \mathbf{u}_{j-1}} \right) \cdots \left(\mathbf{X}_i \mathbf{u}_i - \frac{\mathbf{X}_i \mathbf{u}_i (\mathbf{u}_i' \mathbf{X}_i' \mathbf{X}_i \mathbf{u}_i)}{\mathbf{u}_i' \mathbf{X}_i' \mathbf{X}_i \mathbf{u}_i} \right) \\ &= \mathbf{0}. \end{aligned}$$

This implies that $\phi(\mathbf{x})'_{k+1}\mathbf{U} = \mathbf{0}$, and substituting into Equation 2.25 and rearranging gives

$$\hat{\phi}(\mathbf{x})' = \phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1}. \tag{2.26}$$

This appears to require matrix inversion however $\mathbf{P}'\mathbf{U}$ is upper triangular (also tridiagonal) with constant diagonal 1, so the inversion involves the solution of k sets of k linear equations with an upper triangular matrix. To see that $\mathbf{P}'\mathbf{U}$ is upper triangular, note that the diagonal entries are $\mathbf{u}'_j\mathbf{p}_j = \mathbf{u}'_j\mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j/\mathbf{u}'_j\mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j = 1$. The lower diagonal entries are given by $\mathbf{u}'_i\mathbf{p}_j$, $i < j$. Since $\mathbf{X}_j\mathbf{u}_i = \mathbf{0}$, $\mathbf{u}'_i\mathbf{p}_j = \mathbf{u}'_i\mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j/\mathbf{u}'_j\mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j = 0$, and putting the pieces together gives the required result.

2.5.1.2 Continuum Regression

An interesting connection between PLS, Principal Components Regression (PCR, Massy (1965))[6], and Ordinary Least Squares (OLS) regression is demonstrated in Stone and Brooks (1990) under a general approach called Continuum Regression (CR). One of the key components of CR is the following criterion

$$T = (\mathbf{u}'\mathbf{X}'\mathbf{y})^2(\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u})^{\alpha/(1-\alpha)-1},$$

which is maximised to form the OLS ($\alpha = 0$), PLS ($\alpha = \frac{1}{2}$), and PCR ($\alpha = 1$) directions respectively. After selecting a direction, one uses the PLS deflation and repeats. For the OLS specialisation, the resulting projection direction is also the correlation coefficient vector and the algorithm stops at a single iteration.

2.5.1.3 Dual Form

Kernel PLS (KPLS) was introduced in Rosipal and Trejo (2001) and a closely related approach is shown to exhibit good performance with an SVM in Rosipal et al. (2003). KPLS follows naturally from the primal case, since from Equation 2.21 one can see the projection directions are in the row space of $\mathbf{X}_j$. We introduce a set of dual features $\boldsymbol{\tau}_j = \mathbf{X}_j\mathbf{u}_j = \mathbf{K}_j\boldsymbol{\alpha}_j$, where $\mathbf{K}_j = \mathbf{X}_j\mathbf{X}'_j$, and the kernel matrix is deflated in the following way

$$\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}'_j}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j}\right)\mathbf{K}_j\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}'_j}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j}\right), \tag{2.27}$$

[6]PCR is PCA feature extraction followed by least squares regression.

which only requires kernel matrices. The dual projections directions are computed from Equation 2.21,

$$\begin{aligned}\sigma^2\boldsymbol{\alpha} &= \mathbf{YY'Xu} && (2.28)\\ &= \mathbf{YY'XX'}\boldsymbol{\alpha} && (2.29)\\ &= \mathbf{YY'K}\boldsymbol{\alpha}, && (2.30)\end{aligned}$$

and scaled using $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/\sqrt{\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}}$ so that the primal direction has unit norm.

We stated earlier that the deflation of $\mathbf{Y}$ is unnecessary in the primal case. However, if $\mathbf{Y}$ is deflated is the same way as $\mathbf{X}$, i.e.

$$\mathbf{Y}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{Y}_j,$$

$\mathbf{Y}_1 = \mathbf{Y}$, then the covariance matrix $\mathbf{X}_j'\mathbf{Y}_j$ remains unchanged since,

$$\begin{aligned}\mathbf{X}_{j+1}'\mathbf{Y}_{j+1} &= \mathbf{X}_j'\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)^2\mathbf{Y}_j\\ &= \mathbf{X}_j'\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{Y}_j\\ &= \mathbf{X}_{j+1}'\mathbf{Y}_j\\ &= \mathbf{X}_{j+1}'\mathbf{Y}.\end{aligned}$$

In the dual case however, the deflation of $\mathbf{Y}$ is required to provide a closed form expression for the projection of a new test point. This expression starts from the primal representation, Equation 2.26. Since $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_j$ is in the column space of $\mathbf{Y}_j$, it follows, using an argument similar to that given above, that $\mathbf{u}_j = \mathbf{X}'\boldsymbol{\alpha}_j$. Similar reasoning also allows us to write $\mathbf{p}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j/\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = \mathbf{X}'\mathbf{X}_j\mathbf{u}_j/\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = \mathbf{X}'\boldsymbol{\tau}_j/\boldsymbol{\tau}_j'\boldsymbol{\tau}_j$. Hence $\mathbf{U} = \mathbf{X}'\mathbf{A}$ and $\mathbf{P} = \mathbf{X}'\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}$ where $\mathbf{A}$ has columns $\boldsymbol{\alpha}_j$ and $\mathbf{T}$ has columns $\boldsymbol{\tau}_j$, $j = 1, \ldots, k$. Assembling the parts gives,

$$\begin{aligned}\hat{\phi}(\mathbf{x})' &= \phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1} && (2.31)\\ &= \phi(\mathbf{x})'\mathbf{X}'\mathbf{A}((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{XX}'\mathbf{A})^{-1} && (2.32)\\ &= \mathbf{k}'\mathbf{A}((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}'\mathbf{A})^{-1}. && (2.33)\end{aligned}$$

The complete KPLS algorithm is given in Algorithm 6.

Algorithm 6 Pseudo code for KPLS regression.

```
Inputs: Kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell\times\ell}$, target vectors $\mathbf{Y} \in \mathbb{R}^{\ell\times n}$, dimension $k$
Process:
1. $\mathbf{K}_1 = \mathbf{K}$, $\mathbf{Y}_1 = \mathbf{Y}$
2. For $j = 1, \dots, k$
    (a) Let $\boldsymbol{\alpha}_j$ be the first eigenvector of $\mathbf{Y}_j\mathbf{Y}_j'\mathbf{K}_j$, $\boldsymbol{\alpha}_j \leftarrow \boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}$, and let $\boldsymbol{\tau}_j = \mathbf{K}_j\boldsymbol{\alpha}_j$
    (b) $\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{K}_j\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)$
    (c) $\mathbf{Y}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{Y}_j$
3. End
4. Compute regression coefficients $\hat{\mathbf{C}} = (\hat{\mathbf{X}}'\hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}'\mathbf{Y}$ where $\hat{\mathbf{X}} = [\boldsymbol{\tau}_1 \cdots \boldsymbol{\tau}_k]$
Output: Dual directions $\boldsymbol{\alpha}_j$, features $\boldsymbol{\tau}_j$, $j = 1, \dots, k$, and coefficients $\hat{\mathbf{C}}$
```

2.5.2 Kernel Boosting

The popularity of Boosting has motivated new approaches to many problems in machine learning. One such approach is Kernel Boosting (KB, Crammer et al. (2002)), which learns a kernel matrix from a set of "base kernels matrices". Essentially, it finds a weighted combination of kernels functions,

$$\hat{\kappa}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{k} \gamma_j \kappa_j(\mathbf{x}, \mathbf{y}),$$

where κ_j are known as *base kernel operators* and γ_j are weight coefficients, $j = 1, \dots, k$. The jth base kernel operator is defined as $\kappa_j(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{w}_j\mathbf{w}_j'\mathbf{z}$ where $\mathbf{w}_j$ is a projection direction. Hence, after k iterations each example is projected onto $\sqrt{\gamma_j}\mathbf{w}_j$, and it follows that

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{W}\mathbf{W}'\mathbf{z},$$

where $\mathbf{W}$ has columns $\sqrt{\gamma_j}\mathbf{w}_j$, $j = 1, \dots k$.

We now introduce a quantity known as *kernel alignment* (Cristianini et al. (2001)) which is key to the Kernel Boosting algorithm. Kernel alignment is a similarity measure between two kernel matrices, $\mathbf{K}_1$ and $\mathbf{K}_2$, and defined as

$$A(\mathbf{K}_1, \mathbf{K}_2) = \frac{\langle \mathbf{K}_1, \mathbf{K}_2 \rangle_F}{\sqrt{\langle \mathbf{K}_1, \mathbf{K}_1 \rangle_F \langle \mathbf{K}_2, \mathbf{K}_2 \rangle_F}}, \tag{2.34}$$

where $\langle \mathbf{A}, \mathbf{B} \rangle_F = \mathrm{tr}(\mathbf{A}'\mathbf{B})$ is the Frobenius inner product. The *kernel target alignment* is the kernel alignment between a kernel matrix $\mathbf{K}$ and the "ideal" kernel matrix $\mathbf{y}\mathbf{y}'$, with $\mathbf{y} \in \{-1, +1\}^\ell$. Clearly if $\mathbf{K} = \mathbf{y}\mathbf{y}'$ one can obtain perfect classification on the training examples using $f(\mathbf{x}) = \mathrm{sign}(y_i \kappa(\mathbf{x}_i, \mathbf{x}))$ for a fixed i.

Kernel Boosting operates over pairs of examples, which are said to "align" with their labels if $\mathrm{sign}(\kappa(\mathbf{x}_i, \mathbf{x}_j)) = y_i y_j$ for some i, j. One would like $\kappa(\mathbf{x}_i, \mathbf{x}_j) y_i y_j$ to be as large as possible, hence the following loss functions are introduced

$$\begin{aligned} l(\kappa(\mathbf{x}_i, \mathbf{x}_j), y_i y_j) &= \exp(-y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)) && \text{Exponential loss} \\ l(\kappa(\mathbf{x}_i, \mathbf{x}_j), y_i y_j) &= \log(1 + \exp(-y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j))) && \text{Log loss.} \end{aligned}$$

A distribution matrix is maintained over all pairs of examples and if the weight of a pair is high, the corresponding labels are not aligned with the examples. Those pairs of examples with high weights are emphasised, hence learning concentrates on harder examples.

Algorithm 7 Pseudo code for Kernel Boosting.

```
Inputs: Dataset $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \{-1, +1\}$, $1 \le i \le \ell$, and dimension $k$
Process:
  1. Kernel operator matrix $\mathbf{Q} \leftarrow \mathbf{0}$, initial kernel operator $\hat{\kappa}(\mathbf{x}, \mathbf{z}) = 0$
  2. For $j = 1, \ldots, k$
     (a) Calculate distribution over pairs of examples, $1 \le s, t \le \ell$,
         $\mathbf{D}_j(s,t) = \begin{cases} \exp(-y_s y_t \hat{\kappa}(\mathbf{x}_s, \mathbf{x}_t)) & \text{Exp. Loss} \\ \log(1 + \exp(-y_s y_t \hat{\kappa}(\mathbf{x}_s, \mathbf{x}_t))) & \text{Log Loss} \end{cases}$
     (b) Call $\mathbf{Q}_j = \texttt{baseKernelLearner}(\mathbf{D}_j, S)$. Let $\kappa_j(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{Q}_j\mathbf{z}$.
     (c) Calculate
         $P_j^+ = \{(s,t) | y_s y_t \kappa_j(\mathbf{x}_s, \mathbf{x}_t) > 0\}$    $P_j^- = \{(s,t) | y_s y_t \kappa_j(\mathbf{x}_s, \mathbf{x}_t) < 0\}$
         $w_j^+ = \sum_{(s,t) \in P_j^+} \mathbf{D}_j(s,t) |\kappa_j(\mathbf{x}_s, \mathbf{x}_t)|$    $w_j^- = \sum_{(s,t) \in P_j^-} \mathbf{D}_j(s,t) |\kappa_j(\mathbf{x}_s, \mathbf{x}_t)|$
     (d) Set $\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{Q}\mathbf{z}$ with $\mathbf{Q} \leftarrow \mathbf{Q} + \gamma_j \mathbf{Q}_j$ and $\gamma_j = \frac{1}{2} \log \left( \frac{w_j^+}{w_j^-} \right)$
  3. End
Output: Kernel operator $\hat{\kappa}(\mathbf{x}, \mathbf{z})$
```

Algorithm 7 shows the pseudo code[7] for the Kernel Boosting method. It differs slightly from that given in Crammer et al. (2002) since one can optionally use a set of unlabelled examples, and here we assume that this set is composed of the labelled examples without their labels. The first step of the loop computes a distribution matrix over pairs of examples using the loss functions given above. Following, one finds the kernel operator κ_j based on the distribution matrix using a "base kernel learner". Step 2c) evaluates how well the pairs of examples are aligned with their labels which is used to compute the weighting γ_j for the corresponding kernel operator in step 2d).

The definition of the base kernel learner in Algorithm 7 is left unspecified, however an implementation from Crammer et al. (2002) is shown in Algorithm 8. It finds the projection vector which maximises the alignment of the kernel matrix entries with the corresponding labels, subject to the weights given in the distribution matrix. Notice that the eigenvalue problem at step 2) can be written using the Rayleigh quotient formulation as

$$\max \frac{\mathbf{v}'\mathbf{X}\mathbf{X}'\mathbf{Y}\mathbf{D}\mathbf{Y}\mathbf{X}\mathbf{X}'\mathbf{v}}{\mathbf{v}'\mathbf{X}\mathbf{X}'\mathbf{v}},$$

or, in terms of $\mathbf{w} = \mathbf{X}'\mathbf{v}/\|\mathbf{X}'\mathbf{v}\|$ as

$$\max \frac{\mathbf{w}'\mathbf{X}'\mathbf{Y}\mathbf{D}\mathbf{Y}\mathbf{X}\mathbf{w}}{\|\mathbf{w}\|^2}. \tag{2.35}$$

The elements of the initial distribution matrix $\mathbf{D}_1$ are identical, so one can write $\mathbf{Y}\mathbf{D}_1\mathbf{Y} = c\mathbf{y}\mathbf{y}'$ for some constant c. Since Equation 2.35 is invariant under a scaling of $\mathbf{w}$ one can constrain $\mathbf{w}$ to have unit norm,

$$\begin{array}{ll} \max & (\mathbf{w}'\mathbf{X}'\mathbf{y})^2 \\ \text{s.t.} & \|\mathbf{w}\| = 1. \end{array}$$

Bearing in mind that the squared function is convex, this is the same as the PLS optimisation, Equation 2.20, in the single label case. Although the first kernel boosting direction is the same as the first PLS direction, additional directions are not computed in the same way and will in general be different.

2.5.2.1 Dual Form

The base kernel learner given in Algorithm 8 can easily be extended to operate in a kernel-defined feature space. Note that the norm squared of $\mathbf{w}$ is given by

[7] In a slight deviation from our standard notation, $\mathbf{A}(i,j)$ denotes the matrix element of $\mathbf{A}$ at row i and column j.

```
Algorithm 8 Kernel Boosting base kernel learner.
Input: Distribution matrix $\mathbf{D}$, dataset $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$
Process:
    1. Let
       $\mathbf{Y} = \begin{bmatrix} y_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & y_\ell \end{bmatrix}$
    2. Find $\mathbf{v} \in \mathbb{R}^\ell$ for which $\mathbf{XX'YDYXX'v} = \lambda \mathbf{XX'v}$ has the highest eigenvalue $\lambda$
    3. Set $\mathbf{w} = \mathbf{X'v}/\|\mathbf{X'v}\|$
Output: Kernel operator matrix $\mathbf{Q} = \mathbf{ww'}$
```

$$\|\mathbf{w}\|^2 = \mathbf{v'XX'v} = \mathbf{v'Kv}.$$

Furthermore, at iteration j of the kernel boosting algorithm,

$$\begin{aligned} \kappa_j(\mathbf{x}, \mathbf{z}) &= \mathbf{x}'\mathbf{w}_j\mathbf{w}_j'\mathbf{z} \\ &= \frac{\mathbf{k}_x'\mathbf{v}_j\mathbf{v}_j'\mathbf{k}_z}{\mathbf{v}_j'\mathbf{K}\mathbf{v}_j}, \end{aligned}$$

where $\mathbf{v}_j$ is the vector that solves the eigenvalue problem of Algorithm 8 and $\mathbf{k}_x$ and $\mathbf{k}_z$ are vectors of inner products between the training examples and $\mathbf{x}$ and $\mathbf{z}$ respectively.

2.5.3 Boosted Latent Features

Boosted Latent Features (BLF, Momma and Bennett (2005)) is another technique based on the boosting paradigm, however it is easier to think of it in terms of its relation to PLS. It is an iterative method which computes projection directions according to a user defined loss function and then deflates in the same manner as PLS. The resulting features are used to perform regression.

Name	Loss function $l(\mathbf{y}, \hat{\mathbf{y}})$
Least Squares	$\sum_{i=1}^{\ell}(y_i - \hat{y}_i)^2$
Least Absolute Deviation (LAD)/1-norm	$\sum_{i=1}^{\ell} \|y_i - \hat{y}_i\|$
Exponential	$\sum_{i=1}^{\ell} \exp(-y_i\hat{y}_i)$
Negative binomial log-likelihood/logistic	$\sum_{i=1}^{\ell} \log(1 + \exp(-2y_i\hat{y}_i))$

TABLE 2.1: Several example BLF loss functions.

The aim of BLF is to generate orthogonal linear hypotheses of the form $\mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$. A natural approach for obtaining orthogonality of these extracted features is to use the PLS deflation. Hence BLF computes projection directions which follow the gradient of a user defined loss function, and then deflates in the same way as PLS. The loss functions from Momma and Bennett (2005) are presented in Table 2.1. Note that with the least squares loss BLF specialises to PLS.

Let $l(\mathbf{y}, \hat{\mathbf{y}})$ be a loss function between labels $\mathbf{y}$ and predicted labels $\hat{\mathbf{y}}$ and let $\delta l(\mathbf{y}, \hat{\mathbf{y}})/\delta\hat{\mathbf{y}}$ be the gradient[8] of the loss function with respect to $\hat{\mathbf{y}}$. Then at the jth iteration, BLF solves

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{u}_j \\ \text{s. t.} & \|\mathbf{u}_j\| = 1, \end{array} \tag{2.36}$$

where $\boldsymbol{\alpha}_j = \delta l(\mathbf{y}, \hat{\mathbf{y}}_j)/\delta\hat{\mathbf{y}}_j$ is the gradient of the loss function and $\hat{\mathbf{y}}_j$ is the jth predicted label vector. This optimisation can be seen as finding the projection direction which maximises the inner product between the gradient of the loss function and the projected examples $\mathbf{X}_j\mathbf{u}_j$. The solution is given by $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{X}_j'\boldsymbol{\alpha}_j}$ using the Lagrangian approach. After computing this direction, one deflates the examples, calculates the predicted labels using the features $\mathbf{X}_i\mathbf{u}_i$, $i = 1, \ldots, j$, and finds the value of $\boldsymbol{\alpha}_{j+1}$.

Algorithm 9 Pseudo code for Boosted Latent Features.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, label vector $\mathbf{y} \in \mathbb{R}^{\ell}$, loss function $l$, dimension $k$
Process:
  1. $\mathbf{X}_1 = \left(\mathbf{I} - \frac{\mathbf{jj}'}{\ell}\right)\mathbf{X}$, $\mu_y = \arg\min_{\mu_y} l(\mathbf{y}, \mu_y\mathbf{j})$, $\boldsymbol{\alpha}_1 = -\delta l(\mathbf{y}, \mu_y\mathbf{j})/\delta(\mu_y\mathbf{j})$, $\boldsymbol{\mu}_X = \frac{1}{\ell}\mathbf{X}'\mathbf{j}$
  2. For $j = 1, \ldots, k$
     (a) Compute $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{X}_j'\boldsymbol{\alpha}_j}$, $\boldsymbol{\tau}_j = \mathbf{X}_j\mathbf{u}_j/\sqrt{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}$,
         $\mathbf{T} = [\boldsymbol{\tau}_1, \cdots \boldsymbol{\tau}_j]$
     (b) Deflate $\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{X}_j$
     (c) Compute regression coefficients $(\mu_y, \mathbf{c}) = \arg\min l(\mathbf{y}, \mu_y\mathbf{j} + \mathbf{Tc})$
     (d) Compute loss gradient $\boldsymbol{\alpha}_{j+1} = -\delta l(\mathbf{y}, \mu_y\mathbf{j} + \mathbf{Tc})/\delta(\mu_y\mathbf{j} + \mathbf{Tc})$
  3. End
  4. Compute projection matrix $\mathbf{Z} = \mathbf{U}(\mathbf{P}'\mathbf{U})^{-1}$ where $\mathbf{U} = [\mathbf{u}_1, \ldots \mathbf{u}_k]$, $\mathbf{P} = [\mathbf{p}_1, \ldots \mathbf{p}_k]$
     and $\mathbf{p}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j/\sqrt{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}$.
Output: Directions $\mathbf{Z}$, features $\mathbf{T}$, coefficients $\mathbf{c}$, bias $\mu_y$ and feature means $\boldsymbol{\mu}_X$
```

The complete BLF method is shown in Algorithm 9, and it starts by centering the features of $\mathbf{X}$. The initial feature is $\mu_y\mathbf{j}$ and the gradient of the loss function is computed

[8]One can also use functions that are sub-differentiable, i.e. those that do not have a derivative for all points.

with respect to this vector. In the for loop, the projection direction $\mathbf{u}_j$ is computed using Equation 2.36 and one then deflates the residual matrix $\mathbf{X}_j$. Steps 2c) and 2d) compute the regression coefficients and loss gradient using the features obtained so far (denoted by $\mathbf{T}$). The projection matrix for a new test point is computed at step 4) which is identical to that used in PLS (Equation 2.26). One can then make a prediction for a new test point using

$$f(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)'\mathbf{Z}\mathbf{c} + \mu_y, \tag{2.37}$$

where $\mathbf{Z}$ is the projection matrix for the centered examples, $\mathbf{c}$ is the regression coefficients, $\boldsymbol{\mu}_X$ is the means of original features and μ_y is a bias term.

2.5.3.1 Dual form

The kernel extension to BLF (KBLF) follows naturally from the primal version. Since we are working with residual kernel matrices, they are deflated in the same way as KPLS, i.e. using Equation 2.27. Many of the steps of KBLF mirror those of the primal algorithm. Furthermore, with the least squares loss KBLF reduces exactly to KPLS.

An important difference between KPLS and KBLF is in the way that the projections of a new test example are computed. For KBLF, one must deflate the dual directions as follows,

$$\tilde{\boldsymbol{\alpha}}_j = \left(\mathbf{I} - \sum_{i=1}^{j-1} \frac{\boldsymbol{\tau}_i\boldsymbol{\tau}_i'}{\boldsymbol{\tau}_i'\boldsymbol{\tau}_i}\right)\boldsymbol{\alpha}_j,$$

and the projection matrix for a new example is $\tilde{\mathbf{A}}(\mathbf{T}'\mathbf{K}_1\tilde{\mathbf{A}})^{-1}$ where $\tilde{\mathbf{A}}$ has columns $\tilde{\boldsymbol{\alpha}}_1, \ldots, \tilde{\boldsymbol{\alpha}}_k$. The function for the prediction of a new test point is

$$f(\mathbf{x}) = \mathbf{k}'\left(\mathbf{I} - \frac{\mathbf{j}\mathbf{j}'}{\ell}\right)\mathbf{Q}\mathbf{c} - \mu_K + \mu_y,$$

where μ_K is the mean of the predictions of the training examples and $\mathbf{Q} = \tilde{\mathbf{A}}(\mathbf{T}'\mathbf{K}_1\tilde{\mathbf{A}})^{-1}$.

2.5.4 Sparse KPLS

Sparse KPLS regression (Momma and Bennett (2003)) is an attempt to enforce sparse projection directions in an algorithm based on KPLS. The authors note that the steps in KPLS which require all of the training examples include centering of the data, computation of the features and deflation. These steps are therefore modified accordingly to create a sparse regression function.

Algorithm 10 Pseudo code for sparse PLS.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, label vector $\mathbf{y} \in \mathbb{R}^{\ell}$, dimension $k$, sparsity $\nu$
Process:
  1. $\mathbf{X}_1 = \mathbf{X}$, $\mathbf{v}_1 = \mathbf{y}$
  2. For $j = 1, \ldots, k$
      (a) Compute $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$
      (b) Construct primal projection $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j / \sqrt{\boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{X}_j'\boldsymbol{\alpha}_j}$
      (c) Compute $\mathbf{s}_j = \mathbf{X}_j'\boldsymbol{\beta}_j$ and shift data, $\tilde{\mathbf{X}}_j = \mathbf{X}_j - \mathbf{j}\mathbf{s}_j'$
      (d) Compute $\boldsymbol{\tau}_j = \tilde{\mathbf{X}}_j\mathbf{u}_j / b_j$ with $b_j = \sqrt{\mathbf{u}_j'\tilde{\mathbf{X}}_j'\tilde{\mathbf{X}}_j\mathbf{u}_j}$, and let $\mathbf{T} = [\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_j]$
      (e) Deflate $\mathbf{X}_{j+1} = \tilde{\mathbf{X}}_j\left(\mathbf{I} - \mathbf{u}_j\mathbf{u}_j'\right)$
      (f) Compute regression function $(\mathbf{c}, \mu_y) = \arg\min \|\mathbf{y} - \mathbf{T}\mathbf{c} - \mu_y\mathbf{j}\|^2$
      (g) Compute residual $\mathbf{v}_{j+1} = \mathbf{y} - \mathbf{T}\mathbf{c} - \mu_y\mathbf{j}$
  3. End
  4. Compute projection matrix $\mathbf{Z} = \mathbf{U}\mathrm{diag}(\mathbf{b})^{-1}$, where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$ and diag($\mathbf{b}$)
     is the diagonal matrix with $b_j$ as its diagonal entries, $j = 1, \ldots, k$.
Output: Directions $\mathbf{Z}$, features $\mathbf{T}$, coefficients $\mathbf{c}$, bias $\mu_y$ and shift matrix $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_k]$
```

The primal sparse PLS method is shown in Algorithm 10. At the beginning of the for loop one computes a dual projection vector $\boldsymbol{\alpha}_j$ and the *dual shift vector* $\boldsymbol{\beta}_j$. Both of these vectors contain few non-zero entries which occur at identical positions. Step 2c) computes the primal shift direction $\mathbf{s}_j$ and performs a sparse centering process, known as *shifting*, on the examples. Shifting is sparse since $\mathbf{s}_j = \mathbf{X}_j'\boldsymbol{\beta}_j$ is a linear combination of only a few examples in $\mathbf{X}_j$. The projections of the shifted examples onto $\mathbf{u}_j$ are computed in step 2d), and one then deflates the data by its rows. Notice that this deflation is different from the standard PLS one, and the authors state that the modification is necessary since the PLS deflation requires all examples. It follows that the extracted features are no longer orthogonal. The final two steps of the loop compute regression coefficients using the features obtained so far, and deflate the residual label vector $\mathbf{v}_j$. The projection matrix for the shifted data (computed at step 4)), projects the examples onto $\mathbf{u}_j/b_j$, $j = 1, \ldots, k$. Note that this projection is not sparse since finding b_j at each iteration requires all of the training examples. One can perform regression on a new test example using

$$f(\mathbf{x}) = (\mathbf{x}' - \mathbf{j}'\mathbf{S}')\mathbf{Z}\mathbf{c} + \mu_y,$$

where $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_k]$ is a matrix of dual shift directions. The kernel variant of this algorithm follows from the primal one, see Momma and Bennett (2003) for details.

It remains to show how the dual directions $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$ are computed at step 2a). Let $\nu \in [0, 1]$ be the desired degree of sparsity, then one solves an optimisation based on the ϵ-insensitive loss function,

$$\begin{array}{ll} \min & \frac{1}{\nu\ell} \sum_{i=1}^{\ell} \xi_i + \epsilon \\ \text{s.t.} & \frac{1}{2}\|\mathbf{x}_i - \mathbf{s} - v_i \mathbf{u}\|^2 - \xi_i \leq \epsilon, \quad i = 1, \ldots, \ell \\ & \xi_i \geq 0, \quad i = 1, \ldots, \ell, \end{array}$$

where v_i is the ith element of residual label vector $\mathbf{v}$ and ξ_i, $i = 1, \ldots, \ell$, are *slack* variables. By using the Lagrangian approach one obtains a dual optimisation with solutions $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$. This optimisation can be solved using non-linear programming with linear constraints, with the number of non-zero elements in the dual directions upper bounded by $\nu\ell$.

One of the problems with sparse KPLS is that solving the above optimisation at each iteration is computationally expensive. This point is addressed in Momma (2005)[9] which proposes two heuristics for finding the dual projection vectors that do not require the full kernel matrix in memory and provide significant speed improvements. These approximations are shown to exhibit good performance on several UCI datasets (Newman et al. (1998)) in comparison with an SVM.

2.5.5 Further Advances

An alternative sparse PLS approach, known as Reduced Kernel Orthonormalised PLS (rKOPLS), is presented in Arenas-García et al. (2006). The authors develop rKOPLS from Orthonormalised Partial Least Squares (Worsley et al. (1997)), which minimises $\|\mathbf{Y} - \mathbf{X}\mathbf{U}\|_F^2$ for centered $\mathbf{X}$ and $\mathbf{Y}$. The dual form of this optimisation is equivalent to finding the dominant eigenvectors of

$$\mathbf{K}^x \mathbf{K}^y \mathbf{K}^x \boldsymbol{\alpha} = \lambda \mathbf{K}^x \mathbf{K}^x \boldsymbol{\alpha}, \tag{2.38}$$

where $\mathbf{K}^x = \mathbf{X}\mathbf{X}'$ and $\mathbf{K}^y = \mathbf{Y}\mathbf{Y}'$. A deflation based strategy can be applied to solve the above eigenproblem, which provides at most rank($\mathbf{Y}$) directions.

[9] Momma (2005) also details a sparse variant of KBLF but since we are interested in the general approach of the sparse method, only sparse KPLS is reviewed here.

To form rKOPLS the primal projection directions are selected from the space of r randomly selected examples. Let this set of examples form the rows of $\mathbf{X}_r$ and define $\mathbf{K}_r^x = \mathbf{X}_r\mathbf{X}'$, then rKOPLS solves

$$\mathbf{K}_r^x\mathbf{K}^y\mathbf{K}_r^{x\prime}\boldsymbol{\alpha} = \lambda\mathbf{K}_r^x\mathbf{K}_r^{x\prime}\boldsymbol{\alpha},$$

which is computationally cheaper than Equation 2.38. A related method is introduced in Hoegaerts et al. (2004) which uses the Nyström approximation to obtain a sparse KPLS solution. As in rKOPLS, the directions lie in the space of a subset of the training examples.

2.5.5.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA, Duda et al. (2000)) is another popular approach for supervised feature extraction. It specifically targets directions towards discriminative data, i.e. examples that are partitioned into a finite number of classes. Consider the binary label case, with negatively labelled examples $S_1 = \{\mathbf{x}_1^1, \ldots, \mathbf{x}_{\ell_1}^1\}$ and positively labelled ones $S_2 = \{\mathbf{x}_1^2, \ldots, \mathbf{x}_{\ell_2}^2\}$. LDA maximises the following quantity

$$J(\mathbf{u}) = \frac{\mathbf{u}'\mathbf{S}_b\mathbf{u}}{\mathbf{u}'\mathbf{S}_w\mathbf{u}},$$

where $\mathbf{S}_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'$, $\mathbf{S}_w = \sum_{i=1,2}\sum_{\mathbf{x}\in S_i}(\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)'$ and $\boldsymbol{\mu}_i$ is the mean of the examples in S_i, i.e. $\boldsymbol{\mu}_i = \frac{1}{\ell_i}\sum_{j=1}^{\ell_i}\mathbf{x}_j^i$. The matrix $\mathbf{S}_b$ represents the between-class scatter matrix, and $\mathbf{S}_w$ is the within-class scatter matrix. Hence, maximising $J(\mathbf{u})$ can be seen as finding the projection which maximises the distance between the class means whilst minimising the class variance. In Moghaddam et al. (2006a), LDA is adapted to use sparse directions, which generalises the previous work on cardinality constrained PCA formulations in Moghaddam et al. (2006b).

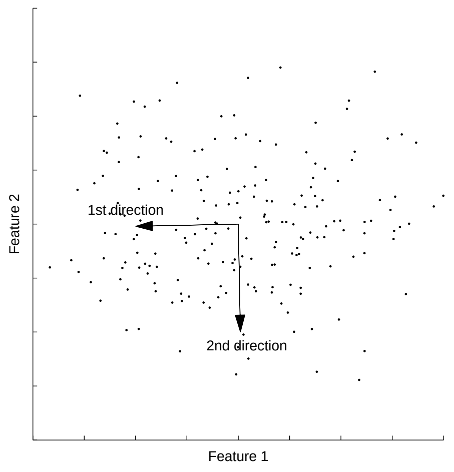
The kernel variant of LDA (Mika et al. (1999)) makes use of the fact that the projection directions are in the space of the examples. The resulting formulation, called Kernel Fisher Discriminant (KFD), is regularised by penalising the norm of the dual projection vector. KFD demonstrates good performance on a set of benchmark datasets when followed with SVM classification.

2.6 Feature Extraction Using Two-Viewed Data

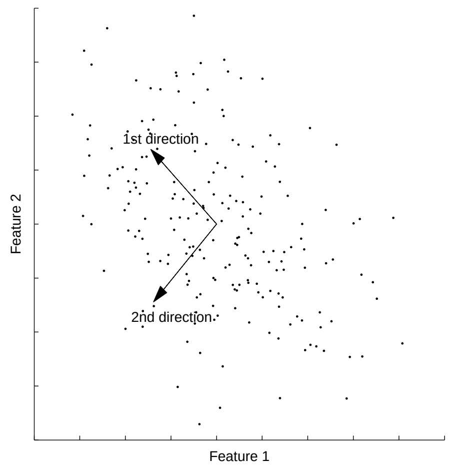
Consider a set of paired examples $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$, where $\mathbf{x}_i \in \mathcal{X}$ and $\mathbf{y}_i \in \mathcal{Y}$, $i = 1, \ldots, \ell$. Each pair of examples is an alternative pair of representations or *views* of a hidden variable $\mathbf{z} \in \mathcal{Z}$, and it is often useful to be able to recover $\mathbf{z}$, e.g. to discover the semantics in a set of English documents and their Japanese translations. Here we introduce several methods which have the aim of recovering the semantics from a paired dataset. These approaches tie in with the novel feature extractions techniques derived in Chapter 6, which are applied to an enzyme function prediction problem.

2.6.1 Canonical Correlation Analysis

CCA finds the hidden variables in a paired dataset by searching for directions which maximise the correlation. An example of the application of CCA to a simple toy dataset is shown in Figure 2.6. The examples are generated so that the features in the $\mathcal{X}$ view are rotated 45 degrees clockwise in order to form the $\mathcal{Y}$ view. CCA chooses a pair of directions in each view which undoes this rotation, hence one obtains maximum correlation. Notice that in general the CCA projection directions are not orthogonal, and further insight into the geometry of CCA and its dual variant is presented in Kuss and Graepel (2003).



(a) $\mathcal{X}$ view



(b) $\mathcal{Y}$ view

FIGURE 2.6: Plot of the CCA projection vectors for a set of paired 2-dimensional examples.

To derive CCA we start with the definition of the correlation of a pair of zero-mean random variables,

$$\operatorname{corr}(x, y) = \frac{\mathbb{E}[xy]}{\sqrt{\mathbb{E}[xx]\mathbb{E}[yy]}} = \frac{\operatorname{cov}(x, y)}{\sqrt{\operatorname{var}(x)\operatorname{var}(y)}}.$$

Again we consider the projections of the examples onto directions, given by $\mathbf{x}'\mathbf{u}$ and $\mathbf{y}'\mathbf{v}$ respectively. The empirical correlation is

$$\begin{aligned}
\frac{\hat{\mathbb{E}}[\mathbf{u}'\mathbf{x}\mathbf{y}'\mathbf{v}]}{\sqrt{\hat{\mathbb{E}}[\mathbf{u}'\mathbf{x}\mathbf{x}'\mathbf{u}]\hat{\mathbb{E}}[\mathbf{v}'\mathbf{y}\mathbf{y}'\mathbf{v}]}} &= \frac{\mathbf{u}'\mathbf{X}'\mathbf{Y}\mathbf{v}}{\sqrt{\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u}\mathbf{v}'\mathbf{Y}'\mathbf{Y}\mathbf{v}}} \\
&= \frac{\mathbf{u}'\mathbf{C}_{xy}\mathbf{v}}{\sqrt{\mathbf{u}'\mathbf{C}_{xx}\mathbf{u}\mathbf{v}'\mathbf{C}_{yy}\mathbf{v}}},
\end{aligned}$$

where $\mathbf{C}_{xy}$ is the covariance matrix between $\mathbf{X}$ and $\mathbf{Y}$ and similar definitions apply for $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$. Notice that in the above expression $\mathbf{u}$ and $\mathbf{v}$ are invariant to scaling, hence maximising correlation can be written as

$$\begin{aligned}
\max \quad & \mathbf{u}'\mathbf{C}_{xy}\mathbf{v} \\
\text{s.t.} \quad & \mathbf{u}'\mathbf{C}_{xx}\mathbf{u} = 1 \\
& \mathbf{v}'\mathbf{C}_{yy}\mathbf{v} = 1,
\end{aligned} \tag{2.39}$$

which is also equivalent to minimising the least squares error between $\mathbf{Xu}$ and $\mathbf{Yv}$, subject to the same constraints. Alternatively, one can see correlation as the cosine of the angle θ between $\mathbf{Xu}$ and $\mathbf{Yv}$, and hence maximising correlation is equivalent to minimising θ.

To solve Equation 2.39, we use the Lagrangian technique which leads to the following equations

$$\mathbf{C}_{xy}\mathbf{v} = \lambda_x \mathbf{C}_{xx}\mathbf{u} \tag{2.40}$$

$$\mathbf{C}_{yx}\mathbf{u} = \lambda_y \mathbf{C}_{yy}\mathbf{v}. \tag{2.41}$$

Notice that $\lambda_x = \lambda_y = \lambda$ since premultiplying Equations 2.40 and 2.41 by $\mathbf{u}$ and $\mathbf{v}$ respectively equates their left hand sides, and the value of λ is the correlation of the projections. Putting the two equations together,

$$\begin{pmatrix} \mathbf{0} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{C}_{xx} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{yy} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \tag{2.42}$$

which is a generalised eigenvalue problem. The CCA projection directions are those corresponding to the first k eigenvalues. The complete CCA algorithm is given in Algorithm 11.

There are several interesting properties of the above eigenproblem. The first is that for each eigenvalue-eigenvector pair $(\lambda, (\mathbf{u}' \; \mathbf{v}')')$, there is also a pair $(-\lambda, (\mathbf{u}' \; -\mathbf{v}')')$. However, since the eigenvalue corresponds to the correlation, one can ignore eigenvectors

```
Algorithm 11 Pseudo code for CCA feature extraction.
Inputs: Data matrices $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, $\mathbf{Y} \in \mathbb{R}^{\ell \times n}$, dimension $k$
Process:
    1. Find first $k$ eigenvectors of $\begin{pmatrix} \mathbf{0} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{C}_{xx} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{yy} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}$
Output: Directions $\mathbf{u}_j$, $\mathbf{v}_j$ and features $\mathbf{X}\mathbf{u}_j$, $\mathbf{Y}\mathbf{v}_j$, $j = 1, \ldots, k$
```

associated with negative eigenvalues. The second property, which follows from Equation 2.3, is that the projections vectors $\mathbf{u}$ and $\mathbf{v}$ are conjugate with respect to $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ respectively. Let $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$, $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ and the CCA features be $\hat{\mathbf{X}} = \mathbf{X}\mathbf{U}$ and $\hat{\mathbf{Y}} = \mathbf{Y}\mathbf{V}$. Then the columns of $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ are orthogonal, i.e. $\hat{\mathbf{X}}'\hat{\mathbf{X}} = \hat{\mathbf{Y}}'\hat{\mathbf{Y}} = \mathbf{I}$. Furthermore, $\hat{\mathbf{X}}'\hat{\mathbf{Y}}$ is a diagonal matrix with diagonal entries corresponding to the correlations of the features.

As a final remark on CCA, note that one can find a solution to Equation 2.42 using iterative deflation. The form of the deflation is given by Equation 2.4. In this case the result is not a simple transformation of the data matrices. Along with the conjugacy of the CCA directions, this highlights the tight coupling of the $\mathbf{X}$ and $\mathbf{Y}$ data matrices.

2.6.1.1 Dual form

Kernel CCA has successfully been applied to cross language information retrieval in Hardoon et al. (2004) and multimedia content-based retrieval in Vinokourov et al. (2003). The CCA projection directions lie in the row space of the examples, hence $\mathbf{u} = \mathbf{X}'\boldsymbol{\alpha}$ and $\mathbf{v} = \mathbf{Y}'\boldsymbol{\beta}$ for some $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. The optimisation from the primal case (Equation 2.39) can therefore be written in terms of kernel matrices

$$\begin{aligned} \max \quad & \boldsymbol{\alpha}'\mathbf{K}^x\mathbf{K}^y\boldsymbol{\beta} \\ \text{s.t.} \quad & \boldsymbol{\alpha}'\mathbf{K}^x\mathbf{K}^x\boldsymbol{\alpha} = 1 \\ & \boldsymbol{\beta}'\mathbf{K}^y\mathbf{K}^y\boldsymbol{\beta} = 1, \end{aligned} \tag{2.43}$$

where $\mathbf{K}^x = \mathbf{X}\mathbf{X}'$ and $\mathbf{K}^y = \mathbf{Y}\mathbf{Y}'$. Applying the Lagrangian method yields the following simultaneous equations

$$\mathbf{K}^x\mathbf{K}^y\boldsymbol{\beta} = \lambda_x\mathbf{K}^x\mathbf{K}^x\boldsymbol{\alpha} \tag{2.44}$$

$$\mathbf{K}^y\mathbf{K}^x\boldsymbol{\alpha} = \lambda_y\mathbf{K}^y\mathbf{K}^y\boldsymbol{\beta}, \tag{2.45}$$

and again we have $\lambda_x = \lambda_y = \lambda$ by premultiplying the first equation by $\boldsymbol{\alpha}$ and the second by $\boldsymbol{\beta}$. Using kernels often implies that examples are evaluated in a high dimensional feature space, and the likelihood of finding strong correlations increases. With the RBF kernel for example, the kernel matrix is always full rank provided examples are distinct. This poses a problem when measuring the correlation of the projected examples. If $\mathbf{K}^x$ and $\mathbf{K}^y$ are invertible,

$$\mathbf{K}^x \boldsymbol{\alpha} = \lambda \mathbf{K}^y \boldsymbol{\beta},$$

and substituting into Equation 2.44 gives

$$\mathbf{K}^x \mathbf{K}^y \boldsymbol{\beta} = \lambda^2 \mathbf{K}^x \mathbf{K}^y \boldsymbol{\beta} \Rightarrow \boldsymbol{\beta} = \lambda^2 \boldsymbol{\beta}.$$

Hence, $\lambda = \pm 1$, i.e. there is perfect correlation regardless of the choice of $\boldsymbol{\beta}$. This is not an ideal scenario and one would prefer to pick more stable directions. To move towards this aim we introduce a result from Hardoon and Shawe-Taylor (2007b) which bounds the correlation of the KCCA projections. The authors upper bound the expectation of $g(\mathbf{x}, \mathbf{y}) = \|\phi_x(\mathbf{x})'\mathbf{U} - \phi_y(\mathbf{y})'\mathbf{V}\|^2$ which is the norm squared of the difference between the projections of $\mathbf{x}$ and $\mathbf{y}$ in the feature space defined by mappings ϕ_x and ϕ_y respectively. Under the KCCA constraints, this function is simply $2(k - \rho)$ where ρ is the cumulative correlation.

Theorem 2.2 (Hardoon and Shawe-Taylor (2007b))**.** *Fix A and B in $\mathbb{R}^+$ and define $g(\boldsymbol{x}, \boldsymbol{y}) = \|\phi_x(\boldsymbol{x})'\boldsymbol{U} - \phi_y(\boldsymbol{y})'\boldsymbol{V}\|^2$ with $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k]$ and $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]$. Obtain features given by $\boldsymbol{u}_i$, $\boldsymbol{v}_i$, $i = 1, \ldots, k$, with $\|\boldsymbol{u}_i\|^2 \le A$ and $\|\boldsymbol{v}_i\|^2 \le B$ with correlations $\rho_i = \boldsymbol{u}_i'\boldsymbol{C}_{xy}\boldsymbol{v}_i$ and $\boldsymbol{u}_i'\boldsymbol{C}_{xx}\boldsymbol{u}_i = \boldsymbol{v}_i'\boldsymbol{C}_{yy}\boldsymbol{v}_i = 1$ on a paired training set S of size ℓ in the feature space defined by kernels κ_x and κ_y drawn i. i. d. according to a distribution $\mathcal{D}$. Then with probability greater than $1 - \delta$ over the generation of S, the expected value of g on new data is bounded by*

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[g(\boldsymbol{x}, \boldsymbol{y})] \;\le\;& \frac{1}{\ell}\sum_{i=1}^{k} 2(1 - \rho_i) + \frac{1}{\ell} 4(A + B)k\sqrt{\sum_{i=1}^{\ell} (\kappa_x(\boldsymbol{x}_i, \boldsymbol{x}_i) + \kappa_y(\boldsymbol{y}_i, \boldsymbol{y}_i))^2} \\ & + 3R(A + B)k\sqrt{\frac{\ln(2/\delta)}{2\ell}},\end{aligned}$$

where $R = \max_{\boldsymbol{x} \in supp(\mathcal{D})}(\kappa_x(\boldsymbol{x}, \boldsymbol{x}) + \kappa_y(\boldsymbol{y}, \boldsymbol{y}))$.

In order to keep $\mathbb{E}_{\mathcal{D}}[g(\mathbf{x}, \mathbf{y})]$ low, the values of R, A, B and the middle term must be small. There are two approaches one can take in order to improve statistical stability: preprocess the data or regularise the KCCA eigenproblem. To regularise KCCA, one penalises projection directions with large norms so that A and B are kept small. The value of R can be reduced by removing outliers (examples with larger than average deviation from the average norm). This step also decreases the value of the square root in the middle term. Note that $\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u}$ can be written as $\sum_{i=1}^{r} \lambda_i(\mathbf{w}_i'\mathbf{u})^2$ where $(\lambda_i, \mathbf{w}_i)$ is the ith eigenvalue-eigenvector pair of $\mathbf{X}'\mathbf{X}$ and r is the rank of $\mathbf{X}$. If $\mathbf{u}$ is entirely in the space of the last eigenvector, then we have $\lambda_r(\mathbf{w}_r'\mathbf{u})^2 = \lambda_r\mathbf{u}'\mathbf{u} = 1$ which implies $\mathbf{u}'\mathbf{u} = 1/\lambda_r$ in this case. It follows that to improve stability, one could also denoise the data using e.g. KPCA which would result in smaller final eigenvalues.

Motivated by the previous theorem, Equation 2.43 is regularised as follows

$$\begin{array}{ll} \max & \boldsymbol{\alpha}'\mathbf{K}^x\mathbf{K}^y\boldsymbol{\beta} \\ \text{s.t.} & (1-\tau)\boldsymbol{\alpha}'\mathbf{K}^x\mathbf{K}^x\boldsymbol{\alpha} + \tau\boldsymbol{\alpha}'\mathbf{K}^x\boldsymbol{\alpha} = 1 \\ & (1-\tau)\boldsymbol{\beta}'\mathbf{K}^y\mathbf{K}^y\boldsymbol{\beta} + \tau\boldsymbol{\beta}'\mathbf{K}^y\boldsymbol{\beta} = 1, \end{array} \tag{2.46}$$

where $0 \le \tau \le 1$ is a trade off parameter which interpolates between maximising correlation and covariance. In the case that $\tau = 1$ the above optimisation maximises the covariance hence the first directions will be identical to the PLS ones. Further directions will not in general be the same.

Equation 2.46 leads to the following pair of simultaneous equations,

$$\mathbf{K}^x\mathbf{K}^y\boldsymbol{\beta} = \lambda_x((1-\tau)\mathbf{K}^x\mathbf{K}^x - \tau\mathbf{K}^x)\boldsymbol{\alpha} \tag{2.47}$$

$$\mathbf{K}^y\mathbf{K}^x\boldsymbol{\alpha} = \lambda_y((1-\tau)\mathbf{K}^y\mathbf{K}^y - \tau\mathbf{K}^y)\boldsymbol{\beta}. \tag{2.48}$$

Again we observe that $\lambda = \lambda_x = \lambda_y$, however in this case λ is not the correlation and instead the "regularised correlation". Solving the above pair of equations is identical to finding the eigenvectors of

$$\begin{pmatrix} \mathbf{0} & \mathbf{K}^x\mathbf{K}^y \\ \mathbf{K}^y\mathbf{K}^x & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \lambda \begin{pmatrix} (1-\tau)(\mathbf{K}^x)^2 - \tau\mathbf{K}^x & \mathbf{0} \\ \mathbf{0} & (1-\tau)(\mathbf{K}^y)^2 - \tau\mathbf{K}^y \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix}. \tag{2.49}$$

The complete regularised KCCA algorithm is shown in Algorithm 12.

As a final point on KCCA, notice that the generalised eigenvalue problem involves matrices of size $2\ell \times 2\ell$ which is computationally prohibitive for large ℓ. To overcome this limitation one can use Incomplete Cholesky decompositions of the kernel matrices to create lower dimensional approximations of the data (Bach and Jordan (2003)).

```
Algorithm 12 Pseudo code for KCCA feature extraction.
Inputs: Kernel matrices $\mathbf{K}^x, \mathbf{K}^y \in \mathbb{R}^{\ell \times \ell}$, dimension $k$, regularisation parameter $\tau$
Process:
  1. Find first $k$ eigenvectors of
     $\begin{pmatrix} \mathbf{0} & \mathbf{K}^x\mathbf{K}^y \\ \mathbf{K}^y\mathbf{K}^x & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \lambda \begin{pmatrix} (1-\tau)(\mathbf{K}^x)^2 - \tau\mathbf{K}^x & \mathbf{0} \\ \mathbf{0} & (1-\tau)(\mathbf{K}^y)^2 - \tau\mathbf{K}^y \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix}$
Output: Directions $\boldsymbol{\alpha}_j$, $\boldsymbol{\beta}_j$ and features $\mathbf{K}^x\alpha_j$ and $\mathbf{K}^y\boldsymbol{\beta}_j$, $j = 1, \dots, k$
```

2.6.2 Recent Advances

A number of interesting CCA variants have been recently suggested, mainly focusing on efficient sparse solutions. One such method is given in Szedmak et al. (2007) which transforms the KCCA problem into a convex maximum margin problem. The resulting optimisation is analogous to the SVM optimisation albeit with vectorial target vectors.

Hardoon and Shawe-Taylor (2007a) introduces a method dubbed Sparse CCA (SCCA) which minimises the least square error between the projections of a paired dataset, with a 1-norm penalty on the projection directions. SCCA operates in the primal space for the $\mathcal{X}$ view and the dual space for the $\mathcal{Y}$ view, hence sparsity implies picking features in $\mathcal{X}$ and examples in $\mathcal{Y}$. The resulting SCCA optimisation is convex, and to prevent an all zeros solution one of the entries in the dual projection vector is fixed to be 1. After the computation of the primal and dual projection directions, one deflates using a modified (K)PLS deflation. However, the deflations used are costly as they operate on the entire data and kernel matrices respectively.

In Torres et al. (2007), the CCA eigenvalue problem is modified to have an additional sparsity constraint on the projection directions. First note that Equations 2.40 and 2.41 can be formulated into a single eigenvalue problem (assuming invertible covariance matrices),

$$\mathbf{C}_{xx}^{-1}\mathbf{C}_{xy}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}\mathbf{u} = \lambda^2\mathbf{u},$$

and a solution for $\mathbf{u}$ leads easily to one for $\mathbf{v}$. By constraining the zero-norm of $\mathbf{u}$, the resulting optimisation is NP-hard, hence an approximation for the zero-norm is used instead. In this case, one uses $\|\mathbf{u}\|_0 \approx \sum_i \log(\epsilon + |\mathbf{u}_i|)$ where $0 \leq \epsilon \ll 1$ avoids problems when $\mathbf{u}_i$ is zero. This is identical to the approximation used in Sriperumbudur et al. (2007). The resulting cardinality constrained CCA optimisation is

$$\begin{array}{ll} \max & \mathbf{u}'\mathbf{C}_{xx}^{-1}\mathbf{C}_{xy}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}\mathbf{u} - \sigma \sum_i \log(\epsilon + |\mathbf{u}_i|), \\ \text{s.t.} & \mathbf{u}'\mathbf{u} = 1 \end{array} \tag{2.50}$$

where σ is a penalty on the sparsity of the solution. Torres et al. (2007) states that this optimisation is difficult to solve because it involves maximising a convex objective function. However, local solutions can be found using gradient descent or by solving a sequence of linear approximations.

2.7 Summary

Feature extraction is a process for reducing the dimensionality of a set of examples to improve prediction performance, reduce computational requirements and gain a greater understanding of the data. Three categories of feature extraction techniques were identified: unsupervised, supervised and those that require paired examples. In these areas, PCA, PLS and CCA are popular choices, based on maximising variance, covariance and correlation respectively. All three algorithms result in eigenproblems, which can be solved in $O(n^3)$ where n is the problem size. More recent feature extraction has moved away from eigenproblem based formulations, and often used sparse projection directions. Furthermore, Boosting has also proven to a popular paradigm, motivating Kernel Boosting and BLF for example.

Chapter 3

A General Framework for Feature Extraction

Since the features one requires are often dependant on the particular learning task, this chapter introduces a general framework for feature extraction[1]. Within the framework, one can use a user defined criterion to select projection directions and the features are computed in the same manner as PLS. Consequently, many of the beneficial properties of PLS are preserved. In particular, the projection directions are conjugate with respect to the data (i.e. $\mathbf{W}'\mathbf{X}'\mathbf{X}\mathbf{W}$ is a diagonal matrix, where $\mathbf{W}$ has its columns composed of the projection directions for a new test point), extracted features are efficient to compute and one can operate in a kernel-defined feature space.

We shall see that the framework draws together a number of existing results and provides additional insight into several popular feature extraction methods. It is shown to specialise to PCA, PLS, BLF and their kernel variants.

3.1 Primal Feature Extraction

Our framework for feature extraction will start with an analysis of the primal representation, however it is easily extended to the dual case to allow the use of kernels. First of all, consider Algorithm 13 which shows what we refer to as the general feature extraction method. Essentially, the method operates iteratively, selecting a new feature direction $\mathbf{u}_j$ at iteration j and then deflating the data matrix $\mathbf{X}_j$ by projecting its columns into the space orthogonal to $\mathbf{X}_j\mathbf{u}_j$,

[1]The general framework can be classified as a Projection Pursuit technique (Friedman and Tukey (1974); Huber (1985)).

$$\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{X}_j \mathbf{u}_j \mathbf{u}_j' \mathbf{X}_j'}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j} \right) \mathbf{X}_j, \tag{3.1}$$

which is identical to the PLS deflation. Observe that the features $\mathbf{X}_j \mathbf{u}_j$, $j = 1, \ldots, \ell$, are orthogonal since they are a linear combination of the columns of $\mathbf{X}_j$ that have been repeatedly projected into the orthogonal complement of previous $\mathbf{X}_i \mathbf{u}_i$, for $i < j$.

Algorithm 13 Pseudo code for primal general feature extraction.

```
Inputs: Data matrix $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, target vectors $\mathbf{Y} \in \mathbb{R}^{\ell \times n}$, dimension $k$
Process:
  1. $\mathbf{X}_1 = \mathbf{X}$
  2. For $j = 1, \ldots, k$
       (a) Select $\mathbf{u}_j$ from the span of the rows of $\mathbf{X}$
       (b) $\mathbf{X}_{j+1} = \left( \mathbf{I} - \frac{\mathbf{X}_j \mathbf{u}_j \mathbf{u}_j' \mathbf{X}_j'}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j} \right) \mathbf{X}_j$
  3. End
Output: Directions $\mathbf{u}_j$ and features $\mathbf{X}_j \mathbf{u}_j$, $j = 1, \ldots, k$
```

There is one requirement that we impose on the choice of $\mathbf{u}_j$, that it should be in the row space of $\mathbf{X}$, i.e. for some $\boldsymbol{\alpha}_j$, $\mathbf{u}_j = \mathbf{X}' \boldsymbol{\alpha}_j$. This ensures a dual representation of the extracted features and later is shown to be important in the context of sparse projection directions. We have seen with PCA and PLS for example, that projection directions are in the row space of the residual matrices and our constraint is less strict since for some $\boldsymbol{\beta}_j$,

$$\begin{aligned}
\mathbf{X}_j' \boldsymbol{\beta}_j &= \mathbf{X}_{j-1}' \left(\mathbf{I} - \frac{\mathbf{X}_{j-1} \mathbf{u}_{j-1} \mathbf{u}_{j-1}' \mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}' \mathbf{X}_{j-1}' \mathbf{X}_{j-1} \mathbf{u}_{j-1}} \right) \boldsymbol{\beta}_j & (3.2) \\
&= \mathbf{X}' \prod_{i=1}^{j-1} \left(\mathbf{I} - \frac{\mathbf{X}_i \mathbf{u}_i \mathbf{u}_i' \mathbf{X}_i'}{\mathbf{u}_i' \mathbf{X}_i' \mathbf{X}_i \mathbf{u}_i} \right) \boldsymbol{\beta}_j, & (3.3)
\end{aligned}$$

where the product is from left to right. This implies that any vector in the row space of $\mathbf{X}_j$ is also in the row space of $\mathbf{X}$.

The apparent disadvantage with this simple description is that the features $\mathbf{X}_j \mathbf{u}_j$ are defined relative to the deflated matrices $\mathbf{X}_j$. We would, however, like to be able to compute the extracted features directly from the original feature vectors so that one can find features for a test set for example. The extracted features for a test point with feature vector $\phi(\mathbf{x})$ are given by

$$\hat{\phi}(\mathbf{x})' = \phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1},$$

where the matrices $\mathbf{U}$ and $\mathbf{P}$ have their columns composed of the vectors $\mathbf{u}_j$ and $\mathbf{p}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j/\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j$, $j = 1, \dots, k$, respectively. The derivation of this result is identical to that of PLS. Furthermore, since $\mathbf{P}'\mathbf{U}$ is an upper triangular matrix the inversion can be efficiently computed as the solution of k sets of k linear equations.

The vectors of feature values across the training set are orthogonal and can be written as $\mathbf{XU}(\mathbf{P}'\mathbf{U})^{-1}$, hence $(\mathbf{U}'\mathbf{P})^{-1}\mathbf{U}'\mathbf{X}'\mathbf{XU}(\mathbf{P}'\mathbf{U})^{-1}$ is a diagonal matrix. This conjugacy of the projection directions with respect to the examples ensures that the resulting features are as dissimilar as possible. It also guarantees that the extracted features are uncorrelated.

3.1.1 Specialisations to Existing Approaches

3.1.1.1 PCA

The connection between the PLS deflation and PCA was first identified in Stone and Brooks (1990) and here we give an equivalent result using iterative PCA. When solving PCA iteratively, one chooses $\mathbf{u}_j$ to be the first eigenvector of $\mathbf{X}_j'\mathbf{X}_j$,

$$\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = \lambda_j\mathbf{u}_j,$$

where λ_j is an eigenvalue. In this case, the deflation of $\mathbf{X}_j$ at each iteration is identical to the PCA deflation

$$\begin{aligned}
\mathbf{X}_{j+1} &= \left(\mathbf{I} - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\mathbf{X}_j \\
&= \left(\mathbf{X}_j - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right) \\
&= \left(\mathbf{X}_j - \frac{\lambda_j\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\lambda_j\mathbf{u}_j'\mathbf{u}_j}\right) \\
&= \mathbf{X}_j\left(\mathbf{I} - \frac{\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j}\right).
\end{aligned}$$

It follows that each deflation can be seen as shrinking the largest eigenvalue of $\mathbf{X}_j'\mathbf{X}_j$ to zero. Hence, the vectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ extracted from this process are exactly the first k eigenvectors of $\mathbf{X}'\mathbf{X}$, which are those used in PCA. The features extracted by PCA are given by $\mathbf{Xu}_1, \ldots, \mathbf{Xu}_k$, although those of the general feature extraction framework are the projection of the residual data matrices. However, these representations are equivalent since

$$\begin{aligned}\mathbf{X}_{j+1}\mathbf{u}_{j+1} &= \left(\mathbf{I} - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j'}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}\right)\mathbf{X}_j\mathbf{u}_{j+1} \\ &= \left(\mathbf{X}_j - \frac{\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'}{\mathbf{u}_j'\mathbf{u}_j}\right)\mathbf{u}_{j+1} \\ &= \mathbf{X}_j\mathbf{u}_{j+1} \\ &= \mathbf{Xu}_{j+1},\end{aligned}$$

where the third line follows from the orthogonality of $\mathbf{u}_j$, $j = 1, \ldots, k$.

We can thus see PCA in two different ways: it deflates the columns of $\mathbf{X}_j$ by the projection of the examples onto the vector of maximal variance, which is equivalent to the deflation of the rows of $\mathbf{X}_j$ by the same vector. Since deflation is a projection onto an orthogonal subspace, this demonstrates two geometric interpretations for the computation of the PCA directions.

3.1.1.2 PLS

If we consider the PLS algorithm, then this is also easily placed within the framework. In this case, $\mathbf{u}_j$ is chosen to be the first singular vector of $\mathbf{X}_j'\mathbf{Y}$. It should be clear that since we are deflating in the same way, the resulting projection vectors are identical, as are the final features. Furthermore, for the PLS case where one wishes to not just select features, but compute the overall regression coefficients for the primal PLS problem, they can be computed as

$$\hat{\mathbf{C}} = \mathbf{U}(\mathbf{P}'\mathbf{U})^{-1}\mathbf{C}',$$

where $\mathbf{C}$ is the matrix with columns $\mathbf{c}_j = \mathbf{Y}'\mathbf{X}_j\mathbf{u}_j/\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j$.

3.1.1.3 BLF

To position BLF within the general framework one must first center the data and compute the initial gradient of the user defined loss function, $\boldsymbol{\alpha}_1$. The jth projection direction is computed using $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{X}_j'\mathbf{X}_j\mathbf{X}_j'\boldsymbol{\alpha}_j}$, where the denominator ensures that the extracted features have unit norm. One then deflates as in Equation 3.1 and computes the next gradient of the loss function $\boldsymbol{\alpha}_{j+1}$ using the features extracted so far. Since the BLF deflation strategy is identical to that of the general framework, it follows that the resulting features are also the same. As well as outputting features, BLF additionally computes regression coefficients, details of which were given in Chapter 2.

3.2 Kernel Feature Extraction

Here we give a dual variable formulation of the framework, which allows feature extraction methods to be used in conjunction with kernels. Given a choice of dual variables $\boldsymbol{\alpha}_j$, let $\boldsymbol{\tau}_j = \mathbf{X}_j\mathbf{u}_j = \mathbf{K}_j\boldsymbol{\alpha}_j$. The deflation of $\mathbf{X}_j$ is given by Equation 3.1, hence the kernel matrix is deflated by

$$\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{K}_j, \tag{3.4}$$

which is computable without explicit feature vectors. Notice that this deflation is different to the dual-sided one used in KPLS, and in general results in non-symmetric residual kernel matrices. However, it is more useful for sparse feature extraction and we shall later see that it still allows for the specialisation to KPLS. The general kernel feature extraction method is given in Algorithm 14.

Algorithm 14 Pseudo code for general kernel feature extraction.

```
Input: Kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell\times\ell}$, target vectors $\mathbf{Y} \in \mathbb{R}^{\ell\times m}$, dimension $k$
Process:
  1. $\mathbf{K}_1 = \mathbf{K}$
  2. For $j = 1, \dots, k$
      (a) Choose dual vector $\boldsymbol{\alpha}_j \in \mathbb{R}^{\ell}$ and let $\boldsymbol{\tau}_j = \mathbf{K}_j\boldsymbol{\alpha}_j$
      (b) $\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{K}_j$
  3. End
Output: Dual vectors $\boldsymbol{\alpha}_j$ and features $\boldsymbol{\tau}_j, j = 1, \dots, k$
```

In the primal case, we were able to give a closed form expression for the projection of a new test point, and we would like to obtain a similar result for the dual variable case. The derivation of the projection of a new point starts with the primal representation and is identical to that of KPLS. Again, we have $\mathbf{U} = \mathbf{X}'\mathbf{A}$ and $\mathbf{P} = \mathbf{X}'\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}$ where $\mathbf{A}$ is the matrix with columns $\boldsymbol{\alpha}_j$ and $\mathbf{T}$ is the matrix with columns $\boldsymbol{\tau}_j$, $j = 1, \dots, k$. Hence, the final features are given by

$$\phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1} = \mathbf{k}'\mathbf{A}\left((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\mathbf{A}\right)^{-1}, \tag{3.5}$$

where $\mathbf{k}$ is a vector of inner products between the test and training examples.

3.2.1 Specialisations to Existing Approaches

3.2.1.1 KPCA

In kernel PCA one finds the most dominant eigenvector of the kernel matrix and then deflates using Equation 2.17. The resulting features are given by $\mathbf{K}\boldsymbol{\beta}_j/\sqrt{\lambda_j}$, $j = 1, \ldots, \ell$, where $(\lambda_j, \boldsymbol{\beta}_j)$ is the jth eigenvalue-eigenvector pair. By letting $\boldsymbol{\alpha}_j = \boldsymbol{\beta}_j/\sqrt{\lambda_j}$ it follows that kernel general framework deflation is equivalent to that of KPCA

$$\begin{aligned}
\mathbf{K}_{j+1} &= \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j \\
&= \left(\mathbf{I} - \frac{\lambda_j\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\lambda_j\boldsymbol{\alpha}_j'\mathbf{K}_j'\boldsymbol{\alpha}_j}\right)\mathbf{K}_j \\
&= \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j.
\end{aligned}$$

Hence, both the resulting residual kernel matrices and dual directions are the same as those produced by KPCA. The extracted features are also identical since

$$\begin{aligned}
\mathbf{K}_{j+1}\boldsymbol{\alpha}_{j+1} &= \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j\boldsymbol{\alpha}_{j+1} \\
&= \mathbf{K}_j\left(\mathbf{I} - \frac{\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\boldsymbol{\alpha}_{j+1} \\
&= \mathbf{K}_j\left(\mathbf{I} - \frac{\lambda_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\boldsymbol{\alpha}_{j+1} \\
&= \mathbf{K}_j\boldsymbol{\alpha}_{j+1} \\
&= \mathbf{K}\boldsymbol{\alpha}_{j+1},
\end{aligned}$$

where the fourth line follows from the orthogonality of the eigenvectors.

3.2.1.2 KPLS

At first glance, it may seem that the kernel variant of KPLS does not fit within the general framework for feature extraction since the deflations of the kernel matrices are different. However both deflations yield identical features when using the KPLS dual directions. First, note that Equation 3.3 can be written as

$$\begin{aligned}\mathbf{X}_{j+1} &= \prod_{i=1}^{j}\left(\mathbf{I} - \frac{\mathbf{X}_i\mathbf{u}_i\mathbf{u}_i'\mathbf{X}_i'}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i}\right)\mathbf{X} \\ &= \sum_{i=1}^{j}\left(\mathbf{I} - \frac{\mathbf{X}_i\mathbf{u}_i\mathbf{u}_i'\mathbf{X}_i'}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i}\right)\mathbf{X} \\ &= \left(\mathbf{I} - \mathbf{T}_j(\mathbf{T}_j'\mathbf{T}_j)^{-1}\mathbf{T}_j'\right)\mathbf{X},\end{aligned}$$

where $\mathbf{T}_j$ is the matrix composed of the first j columns of $\mathbf{T}$, and the second line follows from the orthogonality of $\mathbf{X}_i\mathbf{u}_i$, $i = 1, \ldots, j$. Let $\tilde{\mathbf{K}}_j = \mathbf{X}_j\mathbf{X}_j'$ be the jth residual kernel matrix for KPLS, then the following relationship exists with the corresponding matrices used for the kernel general framework

$$\begin{aligned}\tilde{\mathbf{K}}_j &= \mathbf{X}_j\mathbf{X}_j' \\ &= \mathbf{X}_j\mathbf{X}'\left(\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}_{j-1}'\mathbf{T}_{j-1})^{-1}\mathbf{T}_{j-1}'\right) \\ &= \mathbf{K}_j\left(\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}_{j-1}'\mathbf{T}_{j-1})^{-1}\mathbf{T}_{j-1}'\right),\end{aligned}$$

where we have assumed that the vectors $\boldsymbol{\tau}_i$, $i = 1, \ldots, j$, are identical under both deflations. To achieve the equivalence of the dual projections one could always choose dual directions that are deflated by the dual projections. However, this is not required in this case since $\boldsymbol{\alpha}_j$ is chosen to be the first eigenvector of $\mathbf{Y}_j\mathbf{Y}_j'\mathbf{K}_j$, and hence in the space orthogonal to $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{j-1}$.

We elaborate on the previous point. Recall that for KPLS, the deflation of $\mathbf{Y}$ is required in order to obtain the dual representations of the extracted features. It is deflated using the same deflation of the kernel general framework,

$$\begin{aligned}\mathbf{Y}_{j+1} &= \left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}_j'}{\boldsymbol{\tau}_j'\boldsymbol{\tau}_j}\right)\mathbf{Y}_j \\ &= \left(\mathbf{I} - \mathbf{T}_j(\mathbf{T}_j'\mathbf{T}_j)^{-1}\mathbf{T}_j'\right)\mathbf{Y}.\end{aligned}$$

Since $\boldsymbol{\alpha}_j$ is in the column space of $\mathbf{Y}_j$, it can be written as $\boldsymbol{\alpha}_j = \mathbf{Y}_j\boldsymbol{\beta}_j$ for some vector $\boldsymbol{\beta}_j$. Hence, $\boldsymbol{\alpha}_j$ is in the orthogonal space of $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{j-1}$ and

$$\begin{aligned}
\boldsymbol{\tau}_j &= \mathbf{K}_j\boldsymbol{\alpha}_j \\
&= \mathbf{K}_j\left(\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}'_{j-1}\mathbf{T}_{j-1})^{-1}\mathbf{T}'_{j-1}\right)\mathbf{Y}\boldsymbol{\beta}_j \\
&= \mathbf{K}_j\left(\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}'_{j-1}\mathbf{T}_{j-1})^{-1}\mathbf{T}'_{j-1}\right)^2\mathbf{Y}\boldsymbol{\beta}_j \\
&= \tilde{\mathbf{K}}_j\mathbf{Y}_j\boldsymbol{\beta}_j,
\end{aligned}$$

which shows the equivalence of $\boldsymbol{\tau}_j$ under both double and left-sided deflations of the kernel matrix.

To place KPLS within the general framework, we therefore need to modify Algorithm 14 as follows: at the start we let $\mathbf{Y}_1 = \mathbf{Y}$ and at each iteration we choose $\boldsymbol{\alpha}_j$ to be the first eigenvector of $\mathbf{Y}_j\mathbf{Y}'_j\mathbf{K}_j$, scaled so that $\boldsymbol{\alpha}_j \leftarrow \boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}'_j\tilde{\mathbf{K}}_j\boldsymbol{\alpha}_j} = \boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}'_j\mathbf{K}\boldsymbol{\alpha}_j}$; after deflating $\mathbf{K}_j$ we must also deflate $\mathbf{Y}_j$ by the process outlined above; the regression coefficients for KPLS are computed as $\mathbf{c}_j = \mathbf{Y}'_j\mathbf{X}_j\mathbf{u}_j/\mathbf{u}'_j\mathbf{X}'_j\mathbf{X}_j\mathbf{u}_j = \mathbf{Y}'\boldsymbol{\tau}_j/\boldsymbol{\tau}'_j\boldsymbol{\tau}_j$, making $\mathbf{C} = \mathbf{Y}'\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}$. Putting this together with Equation 3.5, the dual regression variables are given by $\mathbf{A}(\mathbf{T}'\mathbf{K}\mathbf{A})^{-1}\mathbf{T}'\mathbf{Y}$, which is identical to the expression given in Rosipal and Trejo (2001).

3.2.1.3 KBLF

Kernel BLF is closely related to the kernel general feature extraction framework. Its specialisation is formed by first centering the kernel matrix and computing the initial gradient of the loss function. Since the kernel matrix in KBLF is deflated on both sides, one computes the jth projection direction using

$$\begin{aligned}
\mathbf{u}_j &= \mathbf{X}'_j\boldsymbol{\beta}_j \\
&= \mathbf{X}'\left(\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}'_{j-1}\mathbf{T}_{j-1})^{-1}\mathbf{T}'_{j-1}\right)\boldsymbol{\beta}_j,
\end{aligned}$$

where $\boldsymbol{\beta}_j$ is the negative gradient of the loss function using the features extracted so far. Clearly we have $\boldsymbol{\alpha}_j = (\mathbf{I} - \mathbf{T}_{j-1}(\mathbf{T}'_{j-1}\mathbf{T}_{j-1})^{-1}\mathbf{T}'_{j-1})\boldsymbol{\beta}_j$ at each iteration, normalised with $\boldsymbol{\alpha}_j \rightarrow \boldsymbol{\alpha}_j/\sqrt{\boldsymbol{\alpha}'_j\mathbf{K}'_j\mathbf{K}_j\boldsymbol{\alpha}_j}$ so that $\boldsymbol{\tau}_j$ has unit norm, $j = 1, \ldots, k$.

Recall that in KBLF there is the requirement that one must deflate the dual projection directions in order to compute the projection matrix for a new test example. However, in this case the $\boldsymbol{\alpha}_j$ vectors are already deflated and the resulting features are computed as per Equation 3.5. Notice that $(\mathbf{T}'\mathbf{T})^{-1} = \mathbf{I}$, so that the projection of a new test example is $\hat{\phi}(\mathbf{x})' = \mathbf{k}'\mathbf{A}\left(\mathbf{T}'\mathbf{K}\mathbf{A}\right)^{-1}$.

3.3 Parallels with the (K)PCA Deflation

We have presented a framework based on deflating the columns of the data by the projections of the residual examples. A related deflation is that of PCA (used in a similar general setting in Smola et al. (1999)), which orthogonalises the rows of the examples. It is difficult to claim superiority of either of the deflations in a general setting. However, there are a number of parallels between these methods and here we compare and contrast them and their kernel variants.

Assume that under the PCA deflation the projection directions are in the row space of the residual matrices, i.e. $\mathbf{u}_j = \mathbf{X}_j'\boldsymbol{\alpha}_j$. Then in the same way that the vectors $\mathbf{X}_j\mathbf{u}_j$ are orthogonal whilst using the general framework, the PCA projection directions are mutually orthogonal as observed in Section 2.4.2. Furthermore, since the rows of $\mathbf{X}_j$ are projected into the space orthogonal to $\mathbf{u}_i$ for $i < j$, $\mathbf{X}_j\mathbf{u}_i = \mathbf{0}$. This is also the case with the general framework deflation,

$$\begin{aligned}
\mathbf{X}_j\mathbf{u}_i &= \left(\mathbf{I} - \frac{\mathbf{X}_{j-1}\mathbf{u}_{j-1}\mathbf{u}_{j-1}'\mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}'\mathbf{X}_{j-1}'\mathbf{X}_{j-1}\mathbf{u}_{j-1}}\right)\mathbf{X}_{j-1}\mathbf{u}_i \\
&= \left(\mathbf{I} - \frac{\mathbf{X}_{j-1}\mathbf{u}_{j-1}\mathbf{u}_{j-1}'\mathbf{X}_{j-1}'}{\mathbf{u}_{j-1}'\mathbf{X}_{j-1}'\mathbf{X}_{j-1}\mathbf{u}_{j-1}}\right)\cdots\left(\mathbf{I} - \frac{\mathbf{X}_i\mathbf{u}_i\mathbf{u}_i'\mathbf{X}_i'}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i}\right)\mathbf{X}_i\mathbf{u}_i \\
&= \mathbf{0},
\end{aligned}$$

however, this property does not imply in general an equivalence to the PCA deflation. Instead, the general framework can be seen as orthogonalising the examples with respect to the projection directions as well as applying an additional transformation on them. In the special case that the dual directions $\boldsymbol{\alpha}_j$ are in the space orthogonal to $\mathbf{X}_i\mathbf{u}_i$, $i < j$, the resulting projection directions are orthogonal

$$\begin{aligned}
\mathbf{u}_j'\mathbf{u}_i &= \boldsymbol{\alpha}_j'\mathbf{X}_j\mathbf{u}_i \\
&= 0,
\end{aligned}$$

which follows from the previous result, and applies to all of the general framework specialisations above.

With the KPCA deflation both the rows and columns of $\mathbf{K}_j$ are orthogonal to $\boldsymbol{\alpha}_i$, $i < j$, since

$$\begin{aligned}
\mathbf{K}_j\boldsymbol{\alpha}_i &= \left(\mathbf{I} - \frac{\mathbf{K}_{j-1}\boldsymbol{\alpha}_{j-1}\boldsymbol{\alpha}'_{j-1}}{\boldsymbol{\alpha}'_{j-1}\mathbf{K}_{j-1}\boldsymbol{\alpha}_{j-1}}\right)\mathbf{K}_{j-1}\boldsymbol{\alpha}_i \\
&= \left(\mathbf{I} - \frac{\mathbf{K}_{j-1}\boldsymbol{\alpha}_{j-1}\boldsymbol{\alpha}'_{j-1}}{\boldsymbol{\alpha}'_{j-1}\mathbf{K}_{j-1}\boldsymbol{\alpha}_{j-1}}\right)\cdots\left(\mathbf{I} - \frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}'_i}{\boldsymbol{\alpha}'_i\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\mathbf{K}_i\boldsymbol{\alpha}_i \\
&= \mathbf{0},
\end{aligned}$$

with a similar derivation showing that $\boldsymbol{\alpha}'_i\mathbf{K}_j = \mathbf{0}$. An identical property applies with the rows of the residual kernel matrices for the kernel general framework deflation, i.e. $\mathbf{K}_j\boldsymbol{\alpha}_i = \mathbf{0}$ for $i < j$.

3.4 A Note on Numerical Stability

One of the disadvantages of the deflation based strategies presented so far is that in practice computations are performed using floating point numbers of finite precision and errors can build up after a number of iterations. These errors are most apparent when the floating point numbers involved are close to the limit of their range. Note that the kernel general framework deflation never increases the norm of the kernel matrix columns. Consider the deflation of a single kernel matrix column $\mathbf{t}_j$ at the jth iteration,

$$\begin{aligned}
\|\mathbf{t}_{j+1}\|^2 &= \left\|\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}'_j}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j}\right)\mathbf{t}_j\right\|^2 \\
&= \mathbf{t}'_j\left(\mathbf{I} - \frac{\boldsymbol{\tau}_j\boldsymbol{\tau}'_j}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j}\right)\mathbf{t}_j \\
&= \mathbf{t}'_j\mathbf{t}_j - \frac{(\mathbf{t}'_j\boldsymbol{\tau}_j)^2}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j} \\
&= \|\mathbf{t}_j\|^2 - \frac{(\mathbf{t}'_j\boldsymbol{\tau}_j)^2}{\boldsymbol{\tau}'_j\boldsymbol{\tau}_j},
\end{aligned}$$

hence $\|\mathbf{t}_{j+1}\|^2$ is either reduced or stays the same, and any reduction is given by the second term in the last line. We are interested in the relative sizes of $\|\mathbf{t}_{j+1}\|^2$ and $\|\mathbf{t}_j\|^2$,

$$\frac{\|\mathbf{t}_{j+1}\|^2}{\|\mathbf{t}_j\|^2} = 1 - \frac{(\mathbf{t}'_j\boldsymbol{\tau}_j)^2}{\|\mathbf{t}_j\|^2\|\boldsymbol{\tau}_j\|^2}.$$

To prevent the columns of the kernel matrix from decreasing too quickly, the cosine of the angle between them and $\boldsymbol{\tau}_j$ must be small. Often the above quantity is small at the first iteration, since if the norm of the center of mass is large compared to the radius of the hypersphere enclosing the data then $(\mathbf{t}_j'\boldsymbol{\tau}_j)^2/\|\mathbf{t}_j\|^2$ is generally large. Hence, one can reduce numerical errors by centering the data, details of which are given in Appendix A.

3.5 Summary

This chapter has introduced a new general framework for feature extraction based on the PLS deflation scheme. Within this framework one chooses a projection direction using a user defined criterion, and then deflates. The framework brings together a number of existing results and supplies additional insights into several feature extraction methods. In particular, it unifies PCA, PLS, BLF and their kernel variants. Several useful properties from PLS are preserved in the framework, including conjugacy of the projection directions with respect to the data, efficient computation of the final features and the ability to operate in a kernel-defined feature space.

Chapter 4

Matrix Approximation for Machine Learning

Kernel methods require a kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell \times \ell}$, which implies a computational complexity of at least $O(\ell^2)$. If matrix inversion or eigenvalue decomposition is required then the complexity increases to $O(\ell^3)$. However, low-rank approximations of the kernel matrix can frequently improve efficiency. For example, the efficiency of SVMs changes from $O(\ell^3)$[1] to $O(\ell r)$ (where r is the rank of the approximation) in Fine and Scheinberg (2001), and the kernel k-means algorithm is shown to improve its efficiency with low-rank matrices in Kulis et al. (2005). Since kernel matrices often have rapidly decaying eigenvalues (Williams and Seeger (2000a)), good approximations can be obtained with a small rank. In this chapter, we provide some insights into matrix approximation using a general unsupervised technique which chooses projection directions by maximising the Frobenius norm difference between successive orthogonalised matrices.

Our analysis provides a number of interesting observations about KGS, KFA, (K)PCA and (K)PLS. It is extended to derive two new sparse kernel approximation algorithms called Greedy Single Deflated KPLS (GSD-KPLS) and Greedy Double Deflated KPLS (GDD-KPLS). These methods are based on the deflations of the general kernel feature extraction framework and KPLS respectively. The mechanism used to provide sparsity for GSD-KPLS and GDD-KPLS is simple to implement, and for GSD-KPLS results in a computationally efficient algorithm. The final part of this chapter compares the approximation methods on a selection of benchmark datasets.

[1] In practice, SVMs can be trained using the Sequential Minimal Optimisation (SMO, Platt (1999)) method which scales linearly to quadratically in ℓ for various test problems.

4.1 Aims and General Approach

Three requirements that are desirable in a matrix approximation method are a small approximation error using a low dimensionality, and computational and memory efficiency. The last two requirements are considered later in Section 4.4. The first leads to the question about how the quality of matrix approximations can be evaluated. This can vary depending on the use of the resulting matrices, for example in a supervised setting a good approximation should also be predictive for the labels. We contemplate a more general scenario and aim to find a low-rank approximation matrix $\hat{\mathbf{X}}$ such that the following residual error

$$\|\mathbf{X} - \hat{\mathbf{X}}\|_F^2 = \sum_{i,j} (\mathbf{X} - \hat{\mathbf{X}})_{ij}^2, \tag{4.1}$$

is small. One could also consider different error functions, for example the 1-norm loss between matrix elements. However, the Frobenius norm is simple to analyse and there is also a useful intuition behind it. Note that

$$\|\mathbf{X}\|_F^2 = \mathrm{tr}(\mathbf{X}'\mathbf{X}) = \sum_{i=1}^{m} \lambda_i,$$

where λ_i is the ith eigenvalue of $\mathbf{X}'\mathbf{X}$, hence the Frobenius norm of $\mathbf{X}$ is the cumulative variance of the columns of $\mathbf{X}$. It follows that Equation 4.1 is simply the residual variance of the approximation $\hat{\mathbf{X}}$.

A simple method for finding an approximation with least error is to project the examples onto a set of orthogonal basis vectors, denoted by the columns of $\mathbf{U}_k \in \mathbb{R}^{m \times k}$, and minimise Equation 4.1,

$$\begin{array}{ll} \min & \|\mathbf{X} - \mathbf{X}\mathbf{U}_k\mathbf{U}_k'\|_F^2 \\ \text{s.t.} & \mathbf{U}_k'\mathbf{U}_k = \mathbf{I} \\ & k < \mathrm{rank}(\mathbf{X}), \end{array}$$

which results in identical projection vectors to PCA. Often, one solves PCA using the eigen-decomposition of the covariance matrix $\mathbf{X}'\mathbf{X}$ however, earlier it was shown how it can also be solved using iterative deflation. Since several methods employing iterative deflation to extract features have been shown, it is natural to ask whether the alternative deflations also have merit in a matrix approximation context. A simple greedy approach for choosing projection directions is to maximise the Frobenius norm difference between successive deflated matrices,

$$\max \|\mathbf{X}_j - \mathbf{X}_{j+1}\|_F^2, \tag{4.2}$$

and this is equivalent to maximising the variance of the component removed from $\mathbf{X}_j$ through deflation. Observe that if we denote the approximation of $\mathbf{X}$ after k iterations as $\hat{\mathbf{X}} = \mathbf{X} - \mathbf{X}_{k+1}$, then the residual error given by substitution into Equation 4.1 is $\|\mathbf{X}_{k+1}\|_F^2$.

A similar criterion for choosing projection directions is used in Smola and Schölkopf (2000) to approximate the kernel matrix, and here we extend their analysis to a number of different deflation schemes. Furthermore, a full approximation of the kernel matrix is given as opposed to the one in Smola and Schölkopf (2000) which is simply a subset of the kernel matrix columns.

4.2 Data Matrix Approximation

The study of matrix approximation strategies begins with the PCA deflation, given by Equation 2.9. Solving Equation 4.2 leads to the PCA projection directions,

$$\|\mathbf{X}_j - \mathbf{X}_{j+1}\|_F^2 = \frac{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}{\mathbf{u}_j'\mathbf{u}_j},$$

and to maximise the above one can fix $\|\mathbf{u}_j\| = 1$, since $\mathbf{u}_j$ is invariant to scaling. The resulting projection directions can be used to approximate a test point $\phi(\mathbf{x})$ using

$$\hat{\phi}(\mathbf{x}) = \phi(\mathbf{x})'\mathbf{U}\mathbf{U}' = \sum_{j=1}^{k} \phi(\mathbf{x})'\mathbf{u}_j\mathbf{u}_j', \tag{4.3}$$

which is the projection of the example onto the basis defined by $\mathbf{u}_j$, $j = 1, \ldots, k$.

The PLS deflation of Equation 2.23 yields an identical result. In this case we have

$$\|\mathbf{X}_j - \mathbf{X}_{j+1}\|_F^2 = \frac{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}{\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j},$$

and fixing the denominator to be equal to 1 implies that maximising the above can be written as

$$\begin{array}{ll} \max & \mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j \\ \text{s.t.} & \mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = 1. \end{array}$$

The Lagrangian is denoted by

$$\mathcal{L}(\mathbf{u}_j, \lambda) = \mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j - \lambda(\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j - 1),$$

where λ is a Lagrange multiplier. Differentiating with respect to $\mathbf{u}_j$ and equating the derivative to zero results in the following eigenproblem

$$\mathbf{X}_j'\mathbf{X}_j\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = \lambda\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j.$$

Define $\mathbf{v}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j$, then $\mathbf{X}_j'\mathbf{X}_j\mathbf{v}_j = \lambda\mathbf{v}_j$ hence $\mathbf{v}_j$ is an eigenvector of the covariance matrix $\mathbf{X}_j'\mathbf{X}_j$. Clearly, we have $\mathbf{v}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = (1/\lambda)\mathbf{X}_j'\mathbf{X}_j\mathbf{v}_j$ which implies that $\mathbf{u}_j = (1/\lambda)\mathbf{v}_j$, assuming $\mathbf{X}_j'\mathbf{X}_j$ is invertible, hence $\mathbf{u}_j$ is computed identically to the corresponding PCA direction. One might expect directions after the first to be different, however Chapter 3 showed that picking directions in this way and using either the PCA or PLS deflations produces the same features.

One problem with the use of the PLS deflation is that the projection of a new test point given by Equation 2.26 is not orthonormal, hence unintuitive in the matrix approximation setting. Further insight into how the projections can be used to approximate a matrix can be gained by examining the PLS deflation

$$\begin{aligned} \mathbf{X}_{k+1} &= \left(\mathbf{I} - \frac{\mathbf{X}_k\mathbf{u}_k\mathbf{u}_k'\mathbf{X}_k'}{\mathbf{u}_k'\mathbf{X}_k'\mathbf{X}_k\mathbf{u}_k}\right)\mathbf{X}_k \\ &= \mathbf{X} - \sum_{i=1}^{k} \frac{\mathbf{X}_i\mathbf{u}_i\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i}{\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i} \\ &= \mathbf{X} - \sum_{i=1}^{k} \mathbf{X}_i\mathbf{u}_i\mathbf{p}_i', \end{aligned}$$

with $\mathbf{p}_i = \mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i/\mathbf{u}_i'\mathbf{X}_i'\mathbf{X}_i\mathbf{u}_i$. Rearranging gives

$$\begin{aligned} \mathbf{X} &= \mathbf{T}\mathbf{P}' + \mathbf{X}_{k+1} \\ &= \mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{X} + \mathbf{X}_{k+1}, \end{aligned}$$

where $\mathbf{T}$ has columns $\boldsymbol{\tau}_j = \mathbf{X}_j\mathbf{u}_j$, $j = 1, \ldots, k$. Hence the original data matrix $\mathbf{X}$ is the projection of the columns of $\mathbf{X}$ onto a basis defined by $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_k$ plus a residual term $\mathbf{X}_{k+1}$. The approximation of a new test example is therefore

$$\hat{\phi}(\mathbf{x}) = \phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1}\mathbf{P}'. \tag{4.4}$$

If one uses the PCA projection directions, $\mathbf{p}_j = \mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j/\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j = \mathbf{u}_j/\mathbf{u}_j'\mathbf{u}_j = \mathbf{u}_j$, hence the above expression is identical to Equation 4.3 in this case.

4.3 Kernel Matrix Approximation

A natural progression from approximating the data matrix is to do the same for the kernel matrix. In this case one needs to find the optimal directions $\boldsymbol{\alpha}_j$ from the residual matrices $\mathbf{K}_j$, $j = 1, \ldots, k$. In analogy to the previous section, these directions are found by maximising the norm of successively deflated kernel matrices,

$$\max \|\mathbf{K}_j - \mathbf{K}_{j+1}\|_F^2, \tag{4.5}$$

which in general is not equivalent to Equation 4.2.

The Frobenius norm of the kernel matrix is the sum of its squared eigenvalues since $\|\mathbf{K}\|_F^2 = \mathrm{tr}(\mathbf{V}\Lambda\mathbf{V}'\mathbf{V}\Lambda\mathbf{V}') = \mathrm{tr}(\Lambda^2)$, where $\mathbf{K} = \mathbf{V}\Lambda\mathbf{V}'$ is the eigen-decomposition of $\mathbf{K}$. The eigenvalues of the kernel matrix correspond to the input variance of the data, hence Equation 4.5 maximises the cumulative squared variance of the component removed from $\mathbf{K}_j$ through deflation, provided $\mathbf{K}_j$ is symmetric.

As before we solve the above optimisation for different deflation strategies, starting with that of KPCA (Equation 2.17). In this case, the dual projection directions are found by maximising

$$\|\mathbf{K}_j - \mathbf{K}_{j+1}\|_F^2 \quad = \quad \left(\frac{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)^2,$$

which is equivalent to solving

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j \\ \text{s.t.} & \boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1. \end{array}$$

The Lagrangian of this optimisation is

$$\mathcal{L}(\boldsymbol{\alpha}_j, \lambda) = \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j - \lambda(\boldsymbol{\alpha}_j'\mathbf{K}_j\boldsymbol{\alpha}_j - 1),$$

where λ is a Lagrange multiplier. After differentiating and equating to zero one obtains $\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = \lambda\mathbf{K}_j\boldsymbol{\alpha}_j$, which is the same as the KPCA problem given by Equation 2.11.

Next, consider the KPLS deflation of Equation 2.27. The value of $\boldsymbol{\alpha}_j$ is computed by maximising

$$\|\mathbf{K}_j - \mathbf{K}_{j+1}\|_F^2 = 2\frac{\boldsymbol{\alpha}_j'\mathbf{K}_j^4\boldsymbol{\alpha}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j^2\boldsymbol{\alpha}_j} - \left(\frac{\boldsymbol{\alpha}_j'\mathbf{K}_j^3\boldsymbol{\alpha}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j^2\boldsymbol{\alpha}_j}\right)^2. \tag{4.6}$$

Since directions are chosen differently to KPLS, the projection of a new test point is no longer computed in the same way. Instead, it is found using

$$\hat{\phi}(\mathbf{x})' = \mathbf{k}'\tilde{\mathbf{A}}\left((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\tilde{\mathbf{A}}\right)^{-1},$$

where $\tilde{\mathbf{A}}$ has columns

$$\tilde{\boldsymbol{\alpha}}_j = \sum_{i=1}^{j-1}\left(\mathbf{I} - \frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\boldsymbol{\alpha}_j, \quad j = 1, \ldots, k.$$

In the primal case Equation 4.4 gives the approximation for a new test point, and it follows that the corresponding approximate kernel evaluation is denoted by

$$\begin{aligned}\hat{\kappa}(\mathbf{x}, \mathbf{z}) &= \phi(\mathbf{x})'\mathbf{U}(\mathbf{P}'\mathbf{U})^{-1}\mathbf{P}'\mathbf{P}(\mathbf{U}'\mathbf{P})^{-1}\mathbf{U}'\phi(\mathbf{z}) && (4.7)\\ &= \hat{\phi}(\mathbf{x})'(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\hat{\phi}(\mathbf{z}) && (4.8)\\ &= \mathbf{k}_x'\tilde{\mathbf{A}}(\mathbf{T}'\mathbf{K}\tilde{\mathbf{A}})^{-1}\mathbf{T}'\mathbf{K}\mathbf{T}(\tilde{\mathbf{A}}'\mathbf{K}'\mathbf{T})^{-1}\tilde{\mathbf{A}}'\mathbf{k}_z, && (4.9)\end{aligned}$$

where $\mathbf{k}_x$ and $\mathbf{k}_z$ are vectors of inner products between the training examples and $\mathbf{x}$ and $\mathbf{z}$ respectively.

Finally, consider the deflation used in the kernel general feature extraction framework (Equation 3.4). The difference between successively deflated kernel matrices is

$$\|\mathbf{K}_j - \mathbf{K}_{j+1}\|_F^2 = \frac{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j},$$

hence, one needs to solve

$$\begin{aligned} \max \quad & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j \\ \text{s.t.} \quad & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1, \end{aligned} \tag{4.10}$$

which results in the eigenvalue equation

$$\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = \lambda\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j,$$

where λ is an eigenvalue. Let $\mathbf{v}_j = \mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j$ then $\boldsymbol{\alpha}_j = (1/\lambda)\mathbf{v}_j$ is a solution, assuming $\mathbf{K}_j'\mathbf{K}_j$ is invertible, which implies $\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = \lambda\boldsymbol{\alpha}_j$ for non-zero eigenvalues. This is identical to the way that KPCA chooses directions. Recall that in the primal case the optimal directions for the PLS deflation were the same as the PCA directions. In the same way and by using the result in Chapter 3, the optimal directions for the kernel general framework deflation are identical to the KPCA ones.

4.4 Sparsity

One drawback of some of the kernel-based approximation methods considered so far is that they are not sparse: to obtain projections for new test points, all training examples are often needed. Sparseness however is a desirable property, providing computational and efficiency benefits, and as such it is often the case that one is prepared to tolerate a small reduction in performance if a high degree of sparseness is achieved.

A direct approach for enforcing sparsity is to constrain the cardinality of the projection directions. For example, one can formulate a sparse variant of the PCA optimisation,

$$\begin{aligned} \max \quad & \mathbf{u}'\mathbf{A}\mathbf{u} \\ \text{s.t.} \quad & \mathbf{u}'\mathbf{u} = 1 \\ & \text{card}(\mathbf{u}) \leq p, \end{aligned} \tag{4.11}$$

where card is the cardinality of its input vector, p is an upper bound on the cardinality of $\mathbf{u}$ and $\mathbf{A} \in \mathbb{R}^{m\times m}$ is a positive semidefinite covariance matrix. Several authors have tackled this optimisation. In d'Aspremont et al. (2005) it is relaxed to form a semidefinite program which can be solved in $O(m^4\sqrt{\log(m)}\epsilon)$ complexity, where ϵ is the desired absolute accuracy. In contrast, Moghaddam et al. (2006b) proposes both greedy and exact methods for solving the above optimisation. The greedy method can use either forward or backward selection to find non-zero indices in $\mathbf{u}$, with complexities $O(m^3)$ and $O(m^4)$ respectively. The exact computation of Equation 4.11 is achieved with the branch and bound method, using a bound on the eigenvalues of the cardinality constrained eigenproblem. In Sriperumbudur et al. (2007), sparse eigenvalue problems are solved using a zero-norm penalty on the projection directions, where the zero-norm is estimated using the method presented in Weston et al. (2003). The resulting problem is formulated as a difference of convex functions program and solved using a sequence of locally convex programs at $O(\eta m^3)$ complexity, where η is the number of iterations before convergence. The authors also suggest relaxing the zero-norm constraint to a 1-norm constraint which can be solved using a sequence of sparse minimum eigenvalue programs.

Most of these sparse optimisations need complex optimisation procedures, however several authors have suggested simpler approaches to sparsity. For example, Arenas-García et al. (2006) implements a sparsity constraint for rKOPLS by projecting onto directions which are a linear combination of a random subset of the training examples. In Franc and Hlavác (2006), the residual approximation error on the training examples is minimised by using projection directions selected from a subset of the examples, which are in turn chosen using a simple greedy method.

A simple implementation is desirable since it is easy to test and analyse. Hence, we use the method adopted in Smola et al. (1999), which represents a trade off between the loss in the quality of the resulting directions, and the gains in computational efficiency. To achieve a sparse representation on the projection vectors, $\boldsymbol{\alpha}_j$ is chosen so that it has only one non-zero entry, hence after k iterations only k training examples will contribute to the directions. Furthermore, if $\boldsymbol{\alpha}_j$ has a non-zero entry at its ith position, then $\boldsymbol{\tau}_j = \mathbf{X}_j\mathbf{u}_j = \mathbf{K}_j\boldsymbol{\alpha}_j$ is a scalar multiple of the ith column of $\mathbf{K}_j$. One of the advantages of this approach to sparsity is that the computation of $\boldsymbol{\alpha}_j$ does not require the entire kernel matrix to be in memory. Furthermore, if an optimisation involving this cardinality constraint is independent of the scaling of $\boldsymbol{\alpha}_j$, then one need only iterate through at most ℓ non-zero entries to find a solution. However, if the examples are unevenly distributed and the directions of interest lie in the space not well covered by them, the resulting directions can be worse than those obtained without sparsity.

A further drawback of the above sparsity approach is that one still needs to run through the entire kernel matrix to find each direction, which implies an $O(\ell^2)$ complexity. This is more than we would like, and we propose a speed-up based on the method given in Smola and Schölkopf (2000). This speed-up uses a small random subset of the examples to select $\boldsymbol{\alpha}_j$, motivated by the following lemma.

Lemma 4.1 (Maximum of Random Variables (Smola and Schölkopf (2000))). *Denote by $\xi_1, \ldots, \xi_c$ identically distributed independent random variables with common cumulative distribution function $F(x)$. Then the cumulative distribution function of $\xi = \max(\xi_i)$ is $G(x) = F(x)^c$.*

Let $\xi_1, \ldots, \xi_c \in [0, 1]$ represent the values of a function $f : \mathbb{R}^\ell \to \mathbb{R}$ with c columns of $\mathbf{K}_j$, then $F(x)$ is the probability that each column has its ξ_i value less than or equal to

x. Clearly we have $P(\xi > x) = 1 - F(x)^c$ from the above lemma and it follows that this quantity rapidly becomes closer to 1 as c increases provided $F(x) \neq 1$. Hence an approximate method for finding the optimal dual sparse direction is to only consider a few columns of the residual kernel matrix.

4.5 Sparse Kernel Matrix Approximation

One therefore has the ingredients to formulate efficient sparse kernel matrix approximation methods. Each optimisation in Section 4.3 is solved subject to the constraint that the dual directions are chosen from scalar multiples of c standard unit vectors. One can then deflate in the appropriate manner. However, there are important properties for each case that are now outlined in detail.

4.5.1 Kernel Feature Analysis

The first method solves the KPCA optimisation subject to a sparsity constraint on the dual vectors,

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j' \mathbf{K}_j' \mathbf{K}_j \boldsymbol{\alpha}_j \\ \text{s.t.} & \boldsymbol{\alpha}_j' \mathbf{K}_j \boldsymbol{\alpha}_j = 1 \\ & \operatorname{card}(\boldsymbol{\alpha}_j) = 1, \end{array} \tag{4.12}$$

which is identical to the Kernel Feature Analysis (KFA) approach used in Smola et al. (1999). The authors also make the suggestion of randomly subsampling the kernel matrix columns in order to select optimal directions, to obtain an $O(\ell k^2 c)$ algorithm. One of the problems with the random subsampling approach is that the relative quality of the best column to the one selected is unknown. There could be one column that was much better than the rest so that being in the top 95% for example would not guarantee being close to this quality.

A simple result is now given that shows with high probability when being in the top percentile for the type of measures used here will also imply a quality close to the best. Consider a sample of data $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ (already projected into a feature space) and suppose that one wishes to find an index i such that $\mathbf{e}_i' \mathbf{K}^2 \mathbf{e}_i$ is large, where $\mathbf{K}$ is the corresponding kernel matrix. If we assume the vectors $\mathbf{x}_i$ have norm one, this corresponds to the optimisation considered in Equation 4.12. The largest possible value is upper bounded by the largest eigenvalue of $\mathbf{K}^2$ which is λ^2, where λ is the largest eigenvalue of $\mathbf{K} = \mathbf{X}\mathbf{X}'$ and also the largest eigenvalue of the matrix $\mathbf{C} = \mathbf{X}'\mathbf{X}$.

Now consider the eigenvalue decomposition of $\mathbf{C} = \mathbf{V}\Lambda\mathbf{V}'$. Let $\mathbf{M} = \mathbf{V}'\mathbf{X}'$, so that $\mathbf{X}' = \mathbf{V}\mathbf{M}$, making the columns $\mathbf{M}_i$ of $\mathbf{M}$ the coefficients of the data in the eigenbasis. It follows that $\|\mathbf{M}_i\| = 1$ and all entries in $\mathbf{M}$ are bounded by 1. Note that

$$\mathbf{M}\mathbf{M}' = \mathbf{V}'\mathbf{X}'\mathbf{X}\mathbf{V} = \Lambda,$$

implying that $\sum_{j=1}^{\ell} m_{1j}^2 = \lambda_1 = \lambda$. Consider a threshold $\theta = \lambda/2\ell$ for m_{1j}^2. At least t examples must have their m_{1j}^2 coordinate exceeding this threshold where

$$t.1 + (\ell - t)\theta = \lambda.$$

Here it is assumed that t examples have their m_{1j}^2 coordinate set to 1, and the remainder are set to θ. This implies that $t = \lambda\ell/(2\ell-\lambda)$ and a random sample of $\log(0.05)/\log((\ell-t)/(e\ell))$ examples will with probability 0.95 have an example j with $m_{1j}^2 \geq \theta$, where e is Euler's number. To derive the sample size c, notice that the probability of choosing c examples with their m_{1j}^2 coordinate less than θ should be less than 0.05,

$$\binom{\ell - t}{c} \Big/ \binom{\ell}{c} < 0.05,$$

where $\binom{\cdot}{\cdot}$ is the choose function. By substituting in bounds on the choose functions, $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ and $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$, the following inequality results

$$\left(\frac{\ell - t}{\ell e}\right)^c < 0.05,$$

and by taking logarithms and simplifying, c is at least $\log(0.05)/\log((\ell - t)/(e\ell))$. Returning to the quantity of interest,

$$\mathbf{e}_j'\mathbf{K}^2\mathbf{e}_j = \mathbf{e}_j\mathbf{X}\mathbf{X}'\mathbf{X}\mathbf{X}'\mathbf{e}_j = \mathbf{x}_j'\mathbf{C}\mathbf{x}_j = \mathbf{M}_j'\Lambda\mathbf{M}_j \geq \lambda\theta = \lambda^2/2\ell,$$

which for large λ is a significant fraction of the optimum, λ^2.

As an interesting geometrical interpretation of this result, observe that

$$\lambda = \mathbf{v}_1'\mathbf{X}'\mathbf{X}\mathbf{v}_1 = \sum_{i=1}^{\ell}(\mathbf{v}_1'\mathbf{x}_i)^2 = \sum_{i=1}^{\ell}\cos^2\gamma_i$$

where γ_i is the angle between the first eigenvector $\mathbf{v}_1$ and $\mathbf{x}_i$. Hence, $\lambda = \ell\hat{\mathbb{E}}[\cos^2\gamma]$, and when $\hat{\mathbb{E}}[\cos^2\gamma]$ is large, λ is close to ℓ. It follows that when the expectation of the angle between the examples and dominant eigenvector is small in general, then choosing one example from a random subset will give a variance close to the best example with high probability. One can also say that the resulting variance will be close to the variance obtained using the dominant eigenvector, hence little is lost through enforcing sparsity in the given way.

An additional observation about KFA (not covered in Smola et al. (1999)) is that in order to compute features at the end of the algorithm, one must deflate the dual directions as follows

$$\tilde{\boldsymbol{\alpha}}_j = \prod_{i=1}^{j-1} \left(\mathbf{I} - \frac{\boldsymbol{\alpha}_i \boldsymbol{\alpha}_i' \mathbf{K}_i'}{\boldsymbol{\alpha}_i' \mathbf{K}_i \boldsymbol{\alpha}_i} \right) \boldsymbol{\alpha}_j,$$

which implies $\mathbf{u}_j = \mathbf{X}'\tilde{\boldsymbol{\alpha}}$ hence $\mathbf{U} = \mathbf{X}'\tilde{\mathbf{A}}$ with $\tilde{\mathbf{A}} = [\tilde{\boldsymbol{\alpha}}_1, \dots, \tilde{\boldsymbol{\alpha}}_k]$. The projection of a new test point is therefore given by $\hat{\phi}(\mathbf{x})' = \mathbf{k}'\tilde{\mathbf{A}}$ and it follows that the corresponding kernel evaluation is

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{k}_x' \tilde{\mathbf{A}} \tilde{\mathbf{A}}' \mathbf{k}_z. \tag{4.13}$$

Notice that $\tilde{\boldsymbol{\alpha}}_j$ has exactly j non-zero entries, of which $j-1$ occur at identical indices to $\tilde{\boldsymbol{\alpha}}_{j-1}$, $j = 2, \dots, k$. Hence, for the above kernel approximation one need only compute k kernel evaluations with $\mathbf{x}$ and $\mathbf{z}$.

Observe that the Kernel Gram-Schmidt method is similar to KFA, since it uses the KPCA deflation and a single residual example per projection direction. Each direction is chosen to be a scalar multiple of the example corresponding to the largest diagonal element of the kernel matrix. It has been shown that this is not the ideal choice by considering the Frobenius norm difference between successively deflated matrices.

4.5.2 Greedy Double Deflated KPLS

The KPLS deflation is now used for sparse matrix approximation, formulating a method termed Greedy Double Deflated KPLS. The optimisation of Equation 4.6 is augmented with a cardinality constraint on the dual vector,

$$\begin{array}{ll} \max & 2\boldsymbol{\alpha}_j' \mathbf{K}_j^4 \boldsymbol{\alpha}_j - (\boldsymbol{\alpha}_j' \mathbf{K}_j^3 \boldsymbol{\alpha}_j)^2 \\ \text{s.t.} & \boldsymbol{\alpha}_j' \mathbf{K}_j^2 \boldsymbol{\alpha}_j = 1 \\ & \text{card}(\boldsymbol{\alpha}_j) = 1, \end{array}$$

which can be solved by computing the values of the objective function for all ℓ non-zero entries of $\boldsymbol{\alpha}_j$. One would like to use Lemma 4.1 to efficiently approximate this optimisation, however, there are cubic and quartic terms of the residual kernel matrix. The $\boldsymbol{\alpha}_j'\mathbf{K}_j^4\boldsymbol{\alpha}_j$ term requires the multiplication of two $\ell \times \ell$ matrices which is $O(\ell^3)$. If one also considers that the KPLS deflation requires $O(\ell^2)$ operations then it is clear that GDD-KPLS has a complexity of $O(k\ell^3)$. The complete GDD-KPLS method is shown in Algorithm 15.

Algorithm 15 Pseudo code for Greedy Double Deflated KPLS.

```
Inputs: Kernel $\mathbf{K} \in \mathbb{R}^{\ell\times\ell}$, dimension $k$
Process:
1) $\mathbf{K}_1 = \mathbf{K}$
2) For $j = 1, \ldots, k$
    (a) Solve
        $\max \quad 2\boldsymbol{\alpha}_j'\mathbf{K}_j^4\boldsymbol{\alpha}_j - (\boldsymbol{\alpha}_j'\mathbf{K}_j^3\boldsymbol{\alpha}_j)^2$
        $\text{s.t.} \quad \boldsymbol{\alpha}_j'\mathbf{K}_j^2\boldsymbol{\alpha}_j = 1$
        $\quad\quad \text{card}(\boldsymbol{\alpha}_j) = 1$
    (b) Deflate $\mathbf{K}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)\mathbf{K}_j\left(\mathbf{I} - \frac{\mathbf{K}_j\boldsymbol{\alpha}_j\boldsymbol{\alpha}_j'\mathbf{K}_j'}{\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j}\right)$
3) End
4) For $j = 1, \ldots, k$
    (a) $\tilde{\boldsymbol{\alpha}}_j = \sum_{i=1}^{j-1}\left(\mathbf{I} - \frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\boldsymbol{\alpha}_j$
5) End
6) Compute $\mathbf{Z} = (\mathbf{T}'\mathbf{K}\tilde{\mathbf{A}})^{-1}\mathbf{T}'\mathbf{K}\mathbf{T}(\tilde{\mathbf{A}}'\mathbf{K}'\mathbf{T})^{-1}$
Output: Directions $\boldsymbol{\alpha}_j$, projections $\mathbf{K}_j\boldsymbol{\alpha}_j$, $j = 1, \ldots, k$, and $\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{k}_x'\tilde{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{A}}'\mathbf{k}_z$
```

As well as having a method to find an approximate kernel evaluation, it is also useful to compute a set of primal features for GDD-KPLS. Notice that if we define $\mathbf{K}_{\mathcal{T}} = \mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'$ and substitute $\mathbf{K}_{\mathcal{T}}$ in place of $\mathbf{K}$ in Equation 4.9 the result is unaltered due to the orthogonality of $\mathbf{T}$. Hence, by using the Incomplete Cholesky decomposition to obtain $\mathbf{K}_{\mathcal{T}} = \mathbf{R}_k\mathbf{R}_k'$,

$$\hat{\phi}(\mathbf{x})' = \mathbf{k}'\tilde{\mathbf{A}}(\mathbf{T}'\mathbf{K}\tilde{\mathbf{A}})^{-1}\mathbf{T}'\mathbf{R}_k,$$

where $\mathbf{R}_k \in \mathbb{R}^{\ell\times k}$ since $\mathbf{K}_{\mathcal{T}}$ has rank k. It follows that the approximation of a test point $\hat{\phi}(\mathbf{x})$ has k features, and $\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \langle\hat{\phi}(\mathbf{x}), \hat{\phi}(\mathbf{z})\rangle$ as required.

4.5.3 Greedy Single Deflated KPLS

The final sparse approximation method, called Greedy Single Deflated KPLS, applies the deflation from the kernel general feature extraction framework. One solves the following optimisation at the jth iteration, based on Equation 4.10,

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j \\ \text{s.t.} & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1 \\ & \text{card}(\boldsymbol{\alpha}_j) = 1. \end{array} \tag{4.14}$$

This is equivalent to the expression used in Smola and Schölkopf (2000), which also applies the same deflation. Notice that by substituting $\mathbf{Q}_j = \mathbf{K}_j'\mathbf{K}_j$, which represents the covariance matrix of the columns of $\mathbf{K}_j$, Equation 4.14 becomes similar to the KFA optimisation of Equation 4.12. It follows that an application of Lemma 4.1 allows one to choose a direction close to the best with high probability when there is a rapid decay in the spectrum of $\mathbf{Q}_j$. The resulting value of the objective function is also likely to be close to that obtained without the cardinality constraint.

One of the problems with the optimisation of Equation 4.14 is that its complexity is $O(\ell^3)$ (as noted in Smola and Schölkopf (2000)). It would seem that since one requires the computation of $\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j$, Lemma 4.1 cannot be applied. However, observing the following relationship

$$\begin{aligned} \mathbf{K}_j'\mathbf{K}_j &= \mathbf{K}'\left(\mathbf{I} - \sum_{i=1}^{j-1}\frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)^2\mathbf{K} \\ &= \mathbf{K}'\left(\mathbf{I} - \sum_{i=1}^{j-1}\frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\mathbf{K} \\ &= \mathbf{K}_j'\mathbf{K}, \end{aligned}$$

where the second line follows from the orthogonality of the dual projections, implies that $\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}\mathbf{K}'\mathbf{K}_j\boldsymbol{\alpha}_j$. Furthermore, instead of using the full kernel matrix $\mathbf{K}$ one can substitute its Nyström approximation. Recall that this is given by

$$\tilde{\mathbf{K}}_k = \mathbf{K}[,I]\mathbf{K}[I,I]^{-1}\mathbf{K}[I,],$$

where $I \in [\ell]^k$ is a set of k random indices. The resulting optimisation can be solved in complexity $O(c^2\ell)$ if one uses c columns of $\mathbf{K}_j$ to select $\boldsymbol{\alpha}_j$, and $\tilde{\mathbf{K}}_c$ in place of $\mathbf{K}$. As with the KPCA deflation, the kernel general feature extraction framework deflates the columns of the kernel matrix independently. Hence, one only needs to deflate a subset of the kernel matrix columns at each iteration. To compute c columns of $\mathbf{K}_j$ in this way requires $O(cj\ell)$ operations, and it follows that k iterations of GSD-KPLS is $O(c^2k\ell + ck^2\ell)$.

After iterating the required number of times, an approximate kernel evaluation can be computed using a similar expression to that used for KPLS (Equation 4.9),

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{k}_x'\mathbf{AZA}'\mathbf{k}_z,$$

where $\mathbf{Z} = (\mathbf{T}'\mathbf{KA})^{-1}\mathbf{T}'\tilde{\mathbf{K}}_q\mathbf{T}(\mathbf{A}'\mathbf{K}'\mathbf{T})^{-1}$ and $q = \max(c, k)$. Notice that $\mathbf{A}$ is a sparse matrix with only one non-zero entry per column, hence one need only compute the k corresponding entries of $\mathbf{k}_x$ and $\mathbf{k}_z$. Furthermore, to compute a set of primal features one can use an identical method to that used for GDD-KPLS. The complete GSD-KPLS method is given in Algorithm 16.

Algorithm 16 Pseudo code for Greedy Single Deflated KPLS.

```
Inputs: Kernel $\mathbf{K} \in \mathbb{R}^{\ell\times\ell}$, dimension $k$, sample size $c$
Process:
1) For $j = 1, \dots, k$
    (a) Randomly pick $\{i_1, \dots, i_c\} \in [\ell]$. Let $\mathbf{K}_1^{\ell,c} = \mathbf{KE}$, $\mathbf{E} = [\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_c}]$
    (b) For $i = 1, \dots, j-1$
        • Deflate $\mathbf{K}_{i+1}^{\ell,c} = \left(\mathbf{I} - \frac{\mathbf{K}_i\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}_i'}{\boldsymbol{\alpha}_i'\mathbf{K}_i'\mathbf{K}_i\boldsymbol{\alpha}_i}\right)\mathbf{K}_i^{\ell,c}$
    (c) End
    (d) Solve
            max   $\boldsymbol{\alpha}_j'\mathbf{K}_j'\tilde{\mathbf{K}}_c^2\mathbf{K}_j\boldsymbol{\alpha}_j$
            s.t.  $\boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1$
                  $\boldsymbol{\alpha}_j \in s \cdot \{\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_c}\}$
        for scalar $s$
2) End
3) Compute $\mathbf{Z} = (\mathbf{T}'\mathbf{KA})^{-1}\mathbf{T}'\tilde{\mathbf{K}}_q\mathbf{T}(\mathbf{A}'\mathbf{K}'\mathbf{T})^{-1}$, $q = \max(c, k)$
Output: Directions $\boldsymbol{\alpha}_j$, projections $\mathbf{K}_j\boldsymbol{\alpha}_j$, $j = 1, \dots, k$, and $\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \mathbf{k}_x'\mathbf{AZA}'\mathbf{k}_z$
```

4.5.4 A Stopping Condition

To conclude the discussion on sparse kernel approximation methods, we provide a stopping criterion which uses a lower bound $\epsilon \geq 0$ on the residual error, $\mathrm{tr}(\mathbf{K} - \hat{\mathbf{K}}) \leq \epsilon$. For GDD-KPLS this does not influence the complexity. With GSD-KPLS, the same situation is apparent using the following observation,

$$\begin{aligned}
\mathrm{tr}(\hat{\mathbf{K}}) &= \mathrm{tr}(\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\mathbf{T}(\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}') && (4.15)\\
&= \mathrm{tr}(\mathbf{T}'\mathbf{K}\mathbf{T}) && (4.16)\\
&= \sum_{i=1}^{k} \boldsymbol{\tau}_i'\mathbf{K}\boldsymbol{\tau}_i, && (4.17)
\end{aligned}$$

where $(\mathbf{T}'\mathbf{T}) = \mathbf{I}$ in this case. A substitution of $\tilde{\mathbf{K}}_c$ for $\mathbf{K}$ provides a method for approximating $\mathrm{tr}(\hat{\mathbf{K}})$, with no increase in the complexity at each iteration.

4.6 Computational Results

This final section is an empirical evaluation of the kernel approximation algorithms. We compare KPCA, KFA, KGS, GSD-KPLS and GDD-KPLS on a selection of benchmark datasets.

4.6.1 Residual Error

The first experiment compares the approximation methods by measuring the residual variance of the approximated kernel matrices,

$$\frac{1}{\ell}\mathrm{tr}(\mathbf{K} - \hat{\mathbf{K}}),$$

on the UCI Arrhythmia and Dermatology datasets. Some additional information about these datasets is shown in Table 4.1. Each dataset is first normalised and centered so that the features have unit norm and zero mean (Appendix A). Centering is required in order to reduce variance caused by a non-zero center of mass. To reduce bias caused by sample selection, the error is measured using 5-fold cross validation. All experimental code is implemented in Matlab.

Dataset	Examples	Features
Arrhythmia	452	279
Dermatology	366	34
Ionosphere	355	34
MUSK “Clean1”	476	166
SPECTF	267	44
WDBC	569	30

Table 4.1: Information about the UCI datasets.

Since one is interested in approximation errors across a range of output dimensionalities, each method is iterated from 5 to 25 times in steps of 5 for the Dermatology dataset. The same test is repeated using the RBF kernel, with $\sigma = 1$, using iterations from 5 to 205 in steps of 50. Notice that no attempt is made to select σ in a model selection framework, however it is intuitive to compare residual error using the same kernel matrix. With the Arrhythmia data we record residual errors for iterations 5 to 205 in steps of 40 with the linear kernel, and 5 to 255 in steps of 50 with the RBF kernel ($\sigma = 1$). In order to evaluate the effectiveness of Lemma 4.1 for KFA and GSD-KPLS, we vary the number of kernel matrix columns used to select the dual directions from the set $\{100, 200, \ell\}$.

	Projections				
	5	10	15	20	25
KPCA	**.0340** (.0007)	**.0203** (.0004)	**.0111** (.0004)	**.0053** (.0002)	**.0021** (.0001)
KGS	.0482 (.0019)	.0312 (.0016)	.0194 (.0011)	.0116 (.0010)	.0059 (.0004)
KFA $c = 100$	.0419 (.0014)	.0263 (.0006)	.0155 (.0002)	.0080 (.0002)	.0031 (.0002)
KFA $c = 200$	.0405 (.0012)	.0256 (.0005)	.0154 (.0004)	.0080 (.0002)	.0033 (.0001)
KFA $c = \ell$	.0403 (.0011)	.0255 (.0005)	.0151 (.0004)	.0078 (.0003)	.0031 (.0003)
GDD-KPLS	.0361 (.0007)	.0224 (.0005)	.0127 (.0005)	.0062 (.0003)	.0024 (.0002)
GSD-KPLS $c = 100$	.0361 (.0008)	.0229 (.0004)	.0131 (.0003)	.0066 (.0003)	.0027 (.0002)
GSD-KPLS $c = 200$	.0365 (.0007)	.0227 (.0004)	.0129 (.0002)	.0065 (.0002)	.0026 (.0002)
GSD-KPLS $c = \ell$	.0367 (.0006)	.0230 (.0007)	.0132 (.0004)	.0066 (.0002)	.0027 (.0003)
KPCA	.0380 (.0031)	.0272 (.0026)	**.0158** (.0017)	**.0076** (.0009)	**.0029** (.0006)
KGS	.0495 (.0030)	.0343 (.0021)	.0233 (.0025)	.0142 (.0011)	.0077 (.0006)
KFA $c = 100$	.0462 (.0027)	.0320 (.0039)	.0192 (.0020)	.0098 (.0015)	.0037 (.0008)
KFA $c = 200$	.0422 (.0040)	.0299 (.0032)	.0190 (.0016)	.0097 (.0005)	.0039 (.0008)
KFA $c = \ell$	.0426 (.0052)	.0299 (.0029)	.0187 (.0019)	.0095 (.0010)	.0039 (.0006)
GDD-KPLS	**.0373** (.0044)	.0276 (.0037)	.0170 (.0030)	.0084 (.0013)	**.0029** (.0006)
GSD-KPLS $c = 100$	.0382 (.0037)	**.0266** (.0030)	.0166 (.0027)	.0082 (.0009)	.0034 (.0009)
GSD-KPLS $c = 200$	.0402 (.0036)	.0274 (.0024)	.0174 (.0021)	.0088 (.0017)	.0033 (.0005)
GSD-KPLS $c = \ell$	.0381 (.0046)	.0267 (.0027)	.0162 (.0014)	.0084 (.0011)	.0037 (.0006)

Table 4.2: Residual errors of training (top) and test (bottom) kernels using the Dermatology dataset with the linear kernel. Best results are in bold, and standard deviations are in parentheses.

	Projections					
	5	45	85	125	165	205
KPCA	**.409** (.019)	**.118** (.009)	**.037** (.004)	**.010** (.001)	**.002** (.000)	**.000** (.000)
KGS	.467 (.020)	.178 (.011)	.068 (.005)	.024 (.003)	.007 (.001)	.001 (.000)
KFA $c = 100$	.461 (.019)	.164 (.010)	.064 (.006)	.022 (.002)	.007 (.001)	.001 (.000)
KFA $c = 200$	.457 (.019)	.163 (.011)	.064 (.006)	.022 (.002)	.007 (.001)	.001 (.000)
KFA $c = \ell$	.454 (.021)	.162 (.011)	.063 (.006)	.022 (.003)	.007 (.001)	.001 (.000)
GDD-KPLS	.424 (.019)	.135 (.009)	.045 (.005)	.012 (.002)	.003 (.000)	**.000** (.000)
GSD-KPLS $c = 100$	.429 (.020)	.138 (.010)	.047 (.005)	.013 (.002)	.003 (.001)	**.000** (.000)
GSD-KPLS $c = 200$	.428 (.022)	.138 (.011)	.047 (.005)	.013 (.002)	.003 (.001)	**.000** (.000)
GSD-KPLS $c = \ell$	.426 (.018)	.138 (.011)	.046 (.005)	.013 (.002)	.003 (.001)	**.000** (.000)
KPCA	.454 (.088)	.286 (.057)	.201 (.045)	.133 (.040)	.095 (.038)	.069 (.033)
KGS	.538 (.103)	.356 (.062)	.229 (.047)	.152 (.042)	.104 (.038)	.073 (.033)
KFA $c = 100$	.500 (.098)	.317 (.060)	.219 (.045)	.148 (.039)	.101 (.036)	.071 (.032)
KFA $c = 200$	.514 (.102)	.320 (.060)	.224 (.045)	.146 (.038)	.100 (.036)	.071 (.032)
KFA $c = \ell$	.513 (.102)	.320 (.060)	.222 (.048)	.147 (.040)	.101 (.036)	.071 (.033)
GDD-KPLS	**.443** (.086)	.280 (.053)	**.194** (.042)	**.123** (.039)	**.084** (.035)	**.063** (.033)
GSD-KPLS $c = 100$	.453 (.084)	.281 (.056)	.197 (.041)	.129 (.036)	.089 (.036)	.065 (.033)
GSD-KPLS $c = 200$	.445 (.085)	**.279** (.054)	.198 (.040)	.130 (.037)	.089 (.036)	.064 (.032)
GSD-KPLS $c = \ell$	.451 (.087)	.284 (.058)	.197 (.043)	.128 (.038)	.088 (.036)	.064 (.033)

Table 4.3: Residual errors of training (top) and test (bottom) kernels using the Arrhythmia dataset with the linear kernel.

Table 4.2 shows the results for the Dermatology dataset with the linear kernel. The kernel matrix of the training examples is referred to as a training kernel, and a test kernel

	Projections				
	5	55	105	155	205
KPCA	**.0371** (.0009)	**.0010** (.0000)	**.0003** (.0000)	**.0001** (.0000)	**.0000** (.0000)
KGS	.0614 (.0040)	.0028 (.0002)	.0008 (.0001)	.0002 (.0000)	.0001 (.0000)
KFA $c = 100$	.0494 (.0007)	.0024 (.0001)	.0007 (.0000)	.0003 (.0000)	.0001 (.0000)
KFA $c = 200$	.0478 (.0022)	.0023 (.0001)	.0007 (.0000)	.0002 (.0000)	.0001 (.0000)
KFA $c = \ell$	.0465 (.0012)	.0023 (.0001)	.0007 (.0000)	.0002 (.0000)	.0001 (.0000)
GDD-KPLS	.0389 (.0008)	.0011 (.0001)	.0004 (.0000)	**.0001** (.0000)	**.0000** (.0000)
GSD-KPLS $c = 100$	.0396 (.0010)	.0012 (.0001)	.0004 (.0000)	**.0001** (.0000)	.0001 (.0000)
GSD-KPLS $c = 200$	.0392 (.0012)	.0012 (.0000)	.0004 (.0000)	**.0001** (.0000)	.0001 (.0000)
GSD-KPLS $c = \ell$	.0397 (.0010)	.0012 (.0001)	.0004 (.0000)	**.0001** (.0000)	.0001 (.0000)
KPCA	.0411 (.0032)	**.0022** (.0003)	**.0015** (.0002)	.0012 (.0002)	**.0009** (.0002)
KGS	.0627 (.0024)	.0045 (.0004)	.0022 (.0002)	.0014 (.0002)	.0010 (.0002)
KFA $c = 100$	.0531 (.0047)	.0038 (.0003)	.0020 (.0003)	.0014 (.0002)	.0010 (.0002)
KFA $c = 200$	.0521 (.0046)	.0038 (.0004)	.0020 (.0003)	.0014 (.0002)	.0010 (.0002)
KFA $c = \ell$	.0500 (.0045)	.0038 (.0005)	.0020 (.0003)	.0014 (.0002)	.0010 (.0002)
GDD-KPLS	**.0394** (.0086)	.0023 (.0003)	**.0015** (.0002)	**.0011** (.0001)	**.0009** (.0001)
GSD-KPLS $c = 100$	.0427 (.0036)	.0023 (.0004)	**.0015** (.0002)	.0012 (.0002)	.0010 (.0002)
GSD-KPLS $c = 200$	.0431 (.0073)	.0023 (.0004)	.0016 (.0002)	.0012 (.0002)	.0010 (.0002)
GSD-KPLS $c = \ell$	.0457 (.0016)	.0023 (.0003)	**.0015** (.0002)	.0012 (.0002)	.0010 (.0002)

Table 4.4: Residual errors of training (top) and test (bottom) kernels using the Dermatology dataset with the RBF kernel.

	Projections					
	5	55	105	155	205	255
KPCA	**.273** (.010)	**.111** (.008)	**.045** (.004)	**.019** (.002)	**.008** (.001)	**.003** (.000)
KGS	.572 (.064)	.216 (.012)	.088 (.007)	.038 (.003)	.016 (.001)	.006 (.000)
KFA $c = 100$	.313 (.011)	.156 (.009)	.075 (.006)	.035 (.003)	.016 (.001)	.006 (.000)
KFA $c = 200$	.313 (.012)	.155 (.009)	.075 (.006)	.035 (.003)	.016 (.001)	.006 (.000)
KFA $c = \ell$	.311 (.010)	.155 (.009)	.074 (.005)	.035 (.003)	.015 (.001)	.006 (.000)
GDD-KPLS	.279 (.011)	.125 (.009)	.052 (.005)	.021 (.002)	.009 (.001)	.004 (.000)
GSD-KPLS $c = 100$	.283 (.010)	.127 (.008)	.055 (.005)	.024 (.002)	.010 (.001)	.004 (.000)
GSD-KPLS $c = 200$	.283 (.011)	.127 (.008)	.055 (.005)	.024 (.002)	.010 (.001)	.004 (.000)
GSD-KPLS $c = \ell$	.283 (.011)	.127 (.008)	.054 (.005)	.024 (.002)	.010 (.001)	.004 (.000)
KPCA	**.281** (.043)	.209 (.038)	**.177** (.037)	**.157** (.037)	**.147** (.037)	**.142** (.037)
KGS	.583 (.088)	.327 (.034)	.229 (.032)	.185 (.036)	.162 (.037)	.149 (.037)
KFA $c = 100$	.318 (.044)	.239 (.040)	.205 (.035)	.177 (.036)	.159 (.037)	.148 (.037)
KFA $c = 200$	.317 (.045)	.240 (.036)	.204 (.034)	.177 (.036)	.159 (.037)	.148 (.037)
KFA $c = \ell$	.313 (.045)	.241 (.037)	.206 (.034)	.177 (.036)	.159 (.037)	.148 (.037)
GDD-KPLS	.285 (.043)	**.208** (.038)	.178 (.035)	.158 (.036)	**.147** (.037)	**.142** (.037)
GSD-KPLS $c = 100$	.288 (.043)	.210 (.039)	.185 (.033)	.164 (.036)	.151 (.037)	.145 (.037)
GSD-KPLS $c = 200$	.289 (.038)	.209 (.039)	.184 (.035)	.163 (.036)	.151 (.037)	.144 (.038)
GSD-KPLS $c = \ell$	.291 (.041)	.210 (.038)	.186 (.036)	.163 (.037)	.151 (.038)	.144 (.038)

Table 4.5: Residual errors of training (top) and test (bottom) kernels using the Arrhythmia dataset with the RBF kernel.

is a kernel matrix of test examples. With the training kernels, the best approximations are always produced using KPCA. The other methods are sparse and one would expect their approximations of the training data to be worse than KPCA in general. Observe that KGS gives the highest approximation error in many cases, which correlates with the previous analysis which identified KGS as choosing non-optimal directions at each iteration. KFA fares better, as it selects directions that are optimal in the sense we defined earlier. Notice that for KFA and GSD-KPLS, setting c to 100 and 200 respectively results in residual errors that are only slightly worse than when $c = \ell$. This suggests that Lemma 4.1 is effective in this case. The training residuals with the KPLS based methods are only slightly worse than KPCA, however with the test kernels they are more competitive than KGS and KFA. Similar trends are evident with the Arrhythmia dataset in Table 4.3. Here, GSD-KPLS and GDD-KPLS give particularly good approximations on the test kernels, improving upon KPCA in every case.

The residual errors for the Dermatology dataset with the RBF kernel are shown in Table 4.4. In this case the rank of the original kernel matrices are often equivalent to the number of examples. Although initial approximation errors using 5 projection are higher than those of the linear kernel, the trends amongst the approximation methods are similar to the linear case. Further evidence for these trends is provided in Table 4.5 for the Arrhythmia dataset.

One might expect KFA to provide stable projections as they rely only on k examples for the projection directions. In contrast, the $\mathbf{Z}$ terms in the kernel evaluations of GSD-KPLS and GDD-KPLS involve the entire kernel matrix, and hence all training examples. Deflation however, is a process which acts upon all of the examples and selection of dual directions is influenced by perturbations in the data (as with PCA for example (Ng et al. (2001))). The property of interest is the variance covered by the dual projections, as opposed to the directions themselves. For KFA the variance is simply $\sum_{i=1}^{k} \boldsymbol{\alpha}_i' \mathbf{K}_i' \mathbf{K}_i \boldsymbol{\alpha}_i$, and each term in the sum is maximised at each iteration, subject to a cardinality constraint on the dual vector. In contrast, Equation 4.14 for GSD-KPLS can be written as

$$\begin{array}{ll} \max & \boldsymbol{\tau}_j' \mathbf{K}\mathbf{K} \boldsymbol{\tau}_j \\ \text{s.t.} & \boldsymbol{\tau}_j' \boldsymbol{\tau}_j = 1 \\ & \boldsymbol{\tau}_j = \mathbf{K}_j \boldsymbol{\alpha}_j \\ & \text{card}(\boldsymbol{\alpha}_j) = 1, \end{array}$$

and hence $\boldsymbol{\tau}_j$ maximises the variance of $\mathbf{K}\mathbf{K} = \mathbf{V}\Lambda^2\mathbf{V}'$, with $\mathbf{K} = \mathbf{V}\Lambda\mathbf{V}'$. Without the cardinality constraint on $\boldsymbol{\alpha}_j$, the solution to the above is simply the maximal eigenvector of $\mathbf{K}$. GSD-KPLS approximates kernel matrices better than KFA in many cases, since at the first iteration $\boldsymbol{\tau}_1 = \mathbf{K}_1\boldsymbol{\alpha}_1$ is a closer approximation of the first eigenvector of $\mathbf{K}$ than $\boldsymbol{\alpha}_1$.

4.6.2 UCI Classification Experiment

This second experiment uses the matrix approximation methods in conjunction with an SVM classifier. One would like to observe whether approximated kernel matrices provide an improvement in accuracy over the original ones. Hence in this case, matrix approximation acts as a noise reduction or regularisation method for the SVM. We use the Ionosphere, MUSK “Clean1”, SPECTF Heart and WDBC datasets, and each dataset is centered and normalised so that the features have unit norm and zero mean.

Furthermore, the LIBSVM package (Chang and Lin (2001)) is used for computing SVM models.

The general approach is to learn matrix approximations on a training set and then compute a test kernel approximation. The training kernel approximation is used to train an SVM and predictions are made using the corresponding test kernel approximation. Each approximation method is evaluated using 3-fold cross validation repeated twice with random permutations of the data. To select parameter values at each cross validation fold, an inner 3-fold cross validation loop is repeated 2 times with random permutations of the data. We also apply matrix approximation using RBF kernels, and for these tests the inner cross validation loop is run once only.

Parameters at the model selection stage are selected as follows. For each approximation method we vary the number of iterations from 1 to the rank of the original data, with a total of 20 steps within this range for the linear kernels and 10 steps for the RBF ones. For KFA and GSD-KPLS we fix $c = 300$, and the SVM penalty parameter is selected from $\{2^{-1}, 2^0, \ldots, 2^7\}$. With the RBF kernels, σ is selected from $\{2^{-4}, \ldots, 2^2\}$.

	SPECTF		MUSK		Ionosphere		WDBC	
	k	Error	k	Error	k	Error	k	Error
All features	44.0	**.202** (.021)	166.0	.182 (.014)	34.0	.155 (.008)	30.0	.037 (.008)
KPCA	43.0	.204 (.015)	136.3	.175 (.006)	19.7	.144 (.013)	19.0	.034 (.000)
KGS	43.0	.204 (.000)	115.0	.169 (.013)	31.0	.148 (.028)	18.7	.041 (.004)
KFA	43.0	**.202** (.012)	131.0	.173 (.011)	24.0	**.140** (.007)	19.0	.035 (.006)
GDD-KPLS	43.0	.204 (.015)	119.0	.170 (.006)	26.7	.144 (.012)	24.0	**.032** (.001)
GSD-KPLS	29.7	**.202** (.001)	119.0	**.162** (.015)	24.3	.145 (.009)	18.3	.038 (.006)
RBF features	44.0	.202 (.039)	166.0	.177 (.029)	34.0	.068 (.031)	30.0	.030 (.008)
KPCA	24.3	**.184** (.036)	115.0	**.086** (.057)	24.0	.071 (.027)	25.0	**.028** (.008)
KGS	43.0	.187 (.013)	152.3	.116 (.020)	33.0	.071 (.026)	24.0	.053 (.019)
KFA	40.3	.195 (.017)	163.0	.124 (.029)	15.0	**.051** (.009)	28.0	.037 (.019)
GDD-KPLS	40.3	.187 (.013)	163.0	.127 (.049)	14.0	.071 (.022)	24.0	.039 (.008)
GSD-KPLS	11.0	.191 (.051)	157.7	.133 (.042)	16.0	.066 (.025)	27.0	.042 (.014)

Table 4.6: Errors obtained by following kernel approximation with SVM classification. Top results use the linear kernel and bottom ones apply the RBF kernel.

The results of this experiment are shown in Table 4.6, and first we discuss the linear results. With the SPECTF dataset, GSD-KPLS is on par with KPCA and KFA, however it requires a smaller dimensionality to match their performance. On the MUSK data, the eigenspectrum decays rapidly and 92.8% of the variance can be captured with the first 30 eigenvectors, hence matrix approximation improves over the use of the original features. In general, the KPLS based methods are comparable to both KFA and KPCA and improve over KGS in many cases. It is also evident that KPCA provides little advantage over the sparse methods, despite using all of the training examples to compute directions.

With the RBF kernels KPCA is more effective than the other approximation methods, resulting in the lowest errors in 3 out of 4 datasets. The sparse methods are disadvantaged in this case since the examples may not cover the kernel space as effectively as in the linear case. One would expect the sparse methods to give better approximations if there was a larger number of examples. As a simple comparison to the linear case, we note that with the MUSK data and an RBF kernel ($\sigma = 0.25$), the first 30 eigenvectors capture just 34.1% of the total variance. However, as σ increases the first few eigenvectors capture a greater proportion of the total variance. In general, the sparse methods are not significantly worse than KPCA, except with the MUSK data, and broadly comparable to each other in terms of their resulting errors with the SVM.

It is worth reiterating that the features required for accurate prediction of the labels do not necessarily correspond to those which most effectively approximate the data. The results above show that approximation can improve accuracy over the use of the original features in conjunction with an SVM. However, one would expect better results if labels are used to guide feature extraction.

4.7 Summary

We have considered matrix approximation in conjunction with rank-one deflations, and analysed the directions which maximise the Frobenius norm difference between successively deflated matrices. The analysis has revealed that the PCA directions are optimal under both PCA and PLS deflations, and yield identical features. Analogous results were observed with the KPCA and left-sided KPLS deflations.

By enforcing sparsity using a single residual example per direction, three sparse kernel matrix approximation methods were formulated. The first, using the KPCA deflation, is identical to KFA. The remaining two were based on left and double sided KPLS deflations, resulting in the novel GSD-KPLS and GDD-KPLS methods respectively. Both KFA and GSD-KPLS have the advantage that they can be trained in linear complexity in the number of examples. Furthermore, when it comes to computing kernel evaluations for a new pair of test points, KFA and GSD-KPLS require only k kernel evaluations per example, where k is the output dimensionality.

The quality of the new sparse approximation methods was measured on a selection of real-world datasets. GSD-KPLS and GDD-KPLS are able to approximate test kernel matrices more effectively in many cases than KPCA and the other sparse approximation algorithms. Furthermore, when used in conjunction with an SVM they are in general comparable with several other approximation techniques.

Chapter 5

Supervised Feature Extraction

If feature extraction is followed by classification then one can often find a smaller and more relevant set of features, compared to the unsupervised case, by using the labels. As PLS is successful in this area, the general feature extraction framework introduced in Chapter 3 is applied to derive two new supervised algorithms, based on maximising kernel target alignment and the covariance with the labels respectively. As the derivation of the new sparse algorithms is relatively simple, the remainder of the chapter is devoted to their analysis.

The main aims of the new algorithms are to perform supervised feature extraction with a high degree of sparsity and in an efficient manner. Hence, we use the one example per projection direction method seen in Chapter 4. Note that few existing supervised algorithms meet the same criteria, with the exceptions of rKOPLS, and sparse KPLS which upper bounds the sparsity of the solution. Sparse LDA has a tight control on the sparsity of the solution, however, the training time scales cubically in the number of examples. Our chosen method of sparsity does come at a price, and the cost of using a single example per direction is compared to the use of multiple examples.

As well as the impact of sparsity, the sparse algorithms are analysed in a number of additional ways. The effectiveness of random subsampling for approximating the optimal example for each projection direction is considered. Furthermore, a worst case bound for the covariance of the resulting features is derived, which gives useful insight into the statistical stability of the features. The final section presents computational results comparing the new feature extraction methods to several other popular approaches. The comparison is performed first on several small datasets and then on larger text classification and face detection datasets, demonstrating the scalability of the methods.

5.1 Supervised Sparse Methods

5.1.1 Sparse Maximal Alignment

Our first algorithm is called Sparse Maximal Alignment (SMA), and it is based on the notion of kernel target alignment. This is a useful quantity to optimise since the alignment on a training set is likely to be similar to that of a test set. This result follows from the sharp concentration of alignment (see Cristianini et al. (2001)), which means that the probability of the empirical estimate of alignment deviating from its mean is an exponentially decaying function of that deviation.

The alignment between a kernel matrix and the label kernel matrix $\mathbf{yy}'$ is

$$A(\mathbf{K}, \mathbf{yy}') = \frac{\langle \mathbf{K}, \mathbf{yy}' \rangle_F}{\sqrt{\langle \mathbf{K}, \mathbf{K} \rangle_F \langle \mathbf{yy}', \mathbf{yy}' \rangle_F}},$$

which is the cosine of the angle between the rows of $\mathbf{K}$ concatenated together and labels $\mathbf{y}$. If the alignment is 1, the kernel matrix elements are proportional to the corresponding inner products between the labels, i.e. $\kappa(\mathbf{x}_i, \mathbf{x}_j) = cy_iy_j$, where c is a scaling factor. As noted earlier, this implies a simple method for making a perfect prediction for a test point.

SMA is derived by maximising the kernel target alignment of the kernel matrix given by projecting the residual examples, subject to the sparsity constraint described in Chapter 4. This kernel matrix is given by $\mathbf{K_u} = \mathbf{Xuu}'\mathbf{X}' = \mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}'$, and has an alignment of

$$\begin{aligned} A(\mathbf{K_u}, \mathbf{yy}') &= \frac{\langle \mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}', \mathbf{yy}' \rangle_F}{\sqrt{\langle \mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}', \mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}' \rangle_F \langle \mathbf{yy}', \mathbf{yy}' \rangle_F}} && (5.1) \\ &= \frac{\mathrm{tr}(\mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}'\mathbf{yy}')}{\sqrt{\mathrm{tr}(\mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}'\mathbf{K}\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}')\mathrm{tr}(\mathbf{yy}'\mathbf{yy}')}} && (5.2) \\ &= \frac{(\boldsymbol{\alpha}'\mathbf{K}'\mathbf{y})^2}{\boldsymbol{\alpha}'\mathbf{K}'\mathbf{K}\boldsymbol{\alpha}\ \mathbf{y}'\mathbf{y}}. && (5.3) \end{aligned}$$

Hence at the jth iteration of SMA, one solves

$$\begin{aligned} \max \quad & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} \quad & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1 \\ & \mathrm{card}(\boldsymbol{\alpha}_j) = 1, \end{aligned} \qquad (5.4)$$

for which the solution is found in an iterative manner by selecting each element of $\boldsymbol{\alpha}_j$ in turn as the non-zero entry and choosing the one which gives maximal alignment. This optimisation can be efficiently approximated by randomly subsampling the columns of the kernel matrix in order to find the non-zero elements of $\boldsymbol{\alpha}_j$. After finding $\boldsymbol{\alpha}_j$, one deflates using the kernel general framework deflation, and repeats for the desired number of iterations.

5.1.2 Sparse Maximal Covariance

Our second sparse algorithm, Sparse Maximal Covariance (SMC), maximises the empirical expectation of the covariance between the examples and their labels, subject to a sparsity constraint. Recall that the covariance for a pair of zero mean random variables x and y is

$$\mathrm{cov}(x, y) = \mathbb{E}[xy].$$

Consider the squared empirical covariance between projected examples and the labels, given by

$$\begin{aligned} \mathrm{C}(S) &= \hat{\mathbb{E}}[y\phi(\mathbf{x})'\mathbf{u}]^2 \\ &= \frac{1}{\ell^2}(\mathbf{u}'\mathbf{X}'\mathbf{y})^2 \\ &= \frac{1}{\ell^2}(\boldsymbol{\alpha}'\mathbf{K}'\mathbf{y})^2, \end{aligned}$$

where both examples and labels are centered. Maximising $\mathrm{C}(S)$ subject to $\|\mathbf{u}\| = 1$ results in an identical vector to that computed for PLS. SMC is therefore a variation on (K)PLS since it augments the optimisation with a cardinality constraint. At the jth iteration of SMC one solves

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} & \boldsymbol{\alpha}_j'\mathbf{K}\boldsymbol{\alpha}_j = 1 \\ & \mathrm{card}(\boldsymbol{\alpha}_j) = 1, \end{array} \tag{5.5}$$

where the first constraint follows from $\mathbf{u}_j'\mathbf{u}_j = \boldsymbol{\alpha}'\mathbf{X}\mathbf{X}'\boldsymbol{\alpha} = \boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}$. Observe that this optimisation is similar to that of Equation 5.4, however the directions are influenced by the input variance of the examples in this case. The pseudo code for both SMA and SMC is given by Algorithm 17.

Algorithm 17 Pseudo code for SMC and SMA.

Inputs: Kernel matrix $\mathbf{K} \in \mathbb{R}^{\ell \times \ell}$, labels $\mathbf{y} \in \mathbb{R}^{\ell}$, dimension k, sample size c, algorithm (i) SMA, (ii) SMC

Process:

```
1) For $j = 1, \ldots, k$
    (a) Randomly pick indices $\{i_1, \ldots, i_c\} \in [\ell]$ and let $\mathbf{K}_1^{\ell,c} = \mathbf{KE}$, $\mathbf{E} = [\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_c}]$
    (b) For $i = 1, \ldots, j-1$
        • Deflate $\mathbf{K}_{i+1}^{\ell,c} = \left(\mathbf{I} - \frac{\mathbf{K}_i \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i' \mathbf{K}_i'}{\boldsymbol{\alpha}_i' \mathbf{K}_i' \mathbf{K}_i \boldsymbol{\alpha}_i}\right) \mathbf{K}_i^{\ell,c}$
    (c) End
    (d) For scalar $s$, solve either (i)
            $\max \ \boldsymbol{\alpha}_j' \mathbf{K}_j' \mathbf{y}$
            $\text{s.t.} \ \boldsymbol{\alpha}_j' \mathbf{K}_j' \mathbf{K}_j \boldsymbol{\alpha}_j = 1$
            $\boldsymbol{\alpha}_j \in s \cdot \{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_c}\}$
        or (ii)
            $\max \ \boldsymbol{\alpha}_j' \mathbf{K}_j' \mathbf{y}$
            $\text{s.t.} \ \boldsymbol{\alpha}_j' \mathbf{K} \boldsymbol{\alpha}_j = 1$
            $\boldsymbol{\alpha}_j \in s \cdot \{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_c}\}$
2) End
3) Compute $\mathbf{Z} = ((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{KA})^{-1}$
```

Output: Directions $\boldsymbol{\alpha}_j$, projections $\mathbf{K}_j \boldsymbol{\alpha}_j$, $j = 1, \ldots, k$, and $\hat{\phi}(\mathbf{x}) = \mathbf{k}'\mathbf{AZ}$

The above definition of covariance assumes that the data is centered, however with sparse data the sparsity is often lost through centering. Centering the data reduces the sum of the eigenvalues of the kernel matrix, and removes irrelevant variance due to a shift in the center of mass. With uncentered data, the first SMC direction represents this variance and hence the initial deflation acts as an approximation to the centering operation.

5.2 Cost of Sparsity

Computing sparse projection directions has many advantages, however, sparsity does come at a price and one would like to know if the benefits justify the disadvantages. Here, we derive what we call p-sparse projection directions, which are those generated using at most p examples per direction, and compare them to 1-sparse directions.

First, consider finding a p-sparse projection direction which maximises covariance, i.e. one which solves

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} & \boldsymbol{\alpha}_j'\mathbf{K}\boldsymbol{\alpha}_j = 1 \\ & \operatorname{card}(\boldsymbol{\alpha}_j) \leq p. \end{array} \tag{5.6}$$

The solution can be found using a combinatorial approach. If one is already aware of which entries in $\boldsymbol{\alpha}_j$ are non-zero, and the corresponding indices are given by $\{i_1, \ldots, i_q\} \in [\ell]$, $q \leq p$, then the above problem can be rephrased by substituting $\boldsymbol{\alpha}_j = \mathbf{E}\boldsymbol{\beta}_j$ where $\mathbf{E} = [\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_q}]$. Hence, it can be reformulated as

$$\begin{array}{ll} \max & \boldsymbol{\beta}_j'\mathbf{E}'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} & \boldsymbol{\beta}_j'\mathbf{E}'\mathbf{K}\mathbf{E}\boldsymbol{\beta}_j = 1, \end{array} \tag{5.7}$$

where the cardinality constraint has been removed. Using the Lagrangian method, the solution to Equation 5.7 is given by

$$\boldsymbol{\beta}_j = \frac{1}{\lambda_j}(\mathbf{E}'\mathbf{K}\mathbf{E})^{-1}\mathbf{E}'\mathbf{K}_j'\mathbf{y},$$

where λ_j is a scaling factor and it is assumed $\mathbf{E}'\mathbf{K}\mathbf{E}$ is invertible. Finding the non-zero indices of $\boldsymbol{\alpha}_j$ in the first place is a costly procedure. One must search all subsets of the numbers in $[\ell]$ of size at most p. The complexity of computing a projection direction using this approach scales proportional to $\binom{\ell}{p}$, $p \leq \ell$, where $\binom{\cdot}{\cdot}$ is the choose function. However, this method suffices to compute p-sparse directions for small p.

In the case that one wishes to find a p-sparse direction which maximises kernel target alignment, it can be found using

$$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} & \boldsymbol{\alpha}_j'\mathbf{K}_j'\mathbf{K}_j\boldsymbol{\alpha}_j = 1 \\ & \operatorname{card}(\boldsymbol{\alpha}_j) \leq p. \end{array} \tag{5.8}$$

Applying the same technique by substituting $\boldsymbol{\alpha}_j = \mathbf{E}\boldsymbol{\beta}_j$ yields

$$\begin{array}{ll} \max & \boldsymbol{\beta}_j'\mathbf{E}'\mathbf{K}_j'\mathbf{y} \\ \text{s.t.} & \boldsymbol{\beta}_j'\mathbf{E}'\mathbf{K}_j'\mathbf{K}_j\mathbf{E}\boldsymbol{\beta}_j = 1, \end{array} \tag{5.9}$$

and via the Lagrangian approach the solution is

$$\boldsymbol{\beta}_j = \frac{1}{\lambda_j}(\mathbf{E}'\mathbf{K}_j'\mathbf{K}_j\mathbf{E})^{-1}\mathbf{E}'\mathbf{K}_j'\mathbf{y},$$

assuming $\mathbf{E}'\mathbf{K}_j'\mathbf{K}_j\mathbf{E}$ is invertible.

These sparse directions are now compared using a synthetic dataset consisting of 100 examples and 1000 features, with labels computed using a linear combination of 50 of the features plus a small uniform noise component. The labels are given by $y = \mathbf{x}'\mathbf{c} + u(0, 0.05)$ where $\mathbf{c}$ is a sparse vector of coefficients with 50 non-zero entries, $u(a, b)$ is a uniform random variable with range $[a, b]$ and $\mathbf{x} = u(0, 1)$. The data is first centered and normalised so that each feature has unit norm, and Equation 5.6 is solved for $p \in \{1, 2, 3\}$ on the resulting data. Note that p-sparse projection directions, $p > 3$, proved to be too computationally expensive on anything other than very small datasets. The extracted features are compared using the cumulative square covariance per kernel evaluation. For a sample S this is given by $B(S)/r$ where

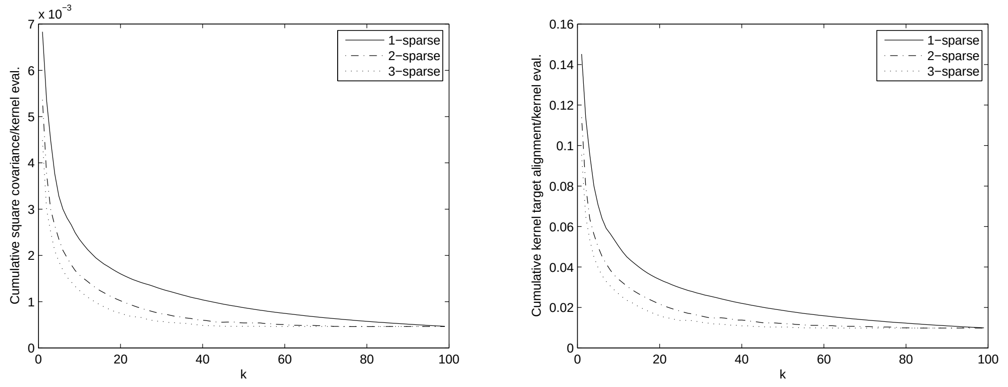
$$B(S) = \sum_{j=1}^{k} \hat{\mathbb{E}}[y\phi(\mathbf{x})_j'\mathbf{u}_j]^2 = \frac{1}{\ell^2}\sum_{j=1}^{k}(\mathbf{u}_j'\mathbf{X}_j'\mathbf{y})^2, \tag{5.10}$$

and r is the number of kernel evaluations used to project a new test point.

The same test is repeated using the solution to Equation 5.8, comparing the resulting features using the cumulative kernel target alignment per kernel evaluation. This quantity is given by $D(S)/r$ where

$$D(S) = \sum_{j=1}^{k} A(\mathbf{X}_j\mathbf{u}_j\mathbf{u}_j'\mathbf{X}_j', \mathbf{y}\mathbf{y}') = \sum_{j=1}^{k} \frac{(\mathbf{y}'\mathbf{X}_j\mathbf{u}_j)^2}{\mathbf{y}'\mathbf{y}\,\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j}, \tag{5.11}$$

which approaches 1 as k tends towards ℓ.



(a) Cumulative square covariance per kernel evaluation for p-sparse maximal covariance. (b) Cumulative kernel target alignment per kernel evaluation for p-sparse maximal alignment.

FIGURE 5.1: Effect of using different numbers of examples per sparse projection direction.

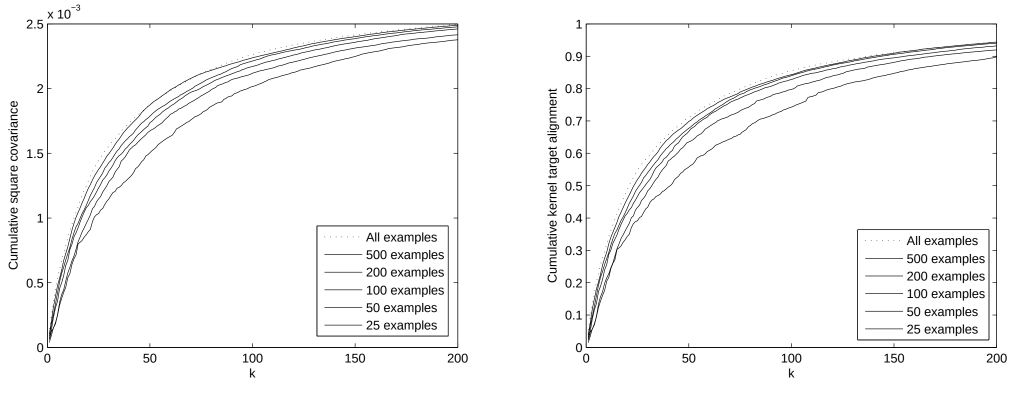
Figure 5.1(a) shows the results of this test using the directions which maximise covariance. The 1-sparse method has a greater covariance per example than the 2 and

3-sparse methods, although this advantage diminishes with increased dimensionality. Similar curves are observed in Figure 5.1(b). Again, 1-sparse projection directions provide more alignment per example than the 2 or 3-sparse ones. Furthermore, notice that the difference between the 1 and 2-sparse curves is generally larger than the difference between the 2 and 3-sparse curves. Hence, in this case 1-sparse directions appear to be optimal in terms of capturing covariance or alignment using sparse directions. One would generally expect this to be the case, although it is not intuitive that 1-sparse methods will always improve against the use of 2 or 3-sparse directions. If one considers the cumulative covariance or alignment, then any loss through enforcing a high level of sparsity can be compensated for by iterating further.

5.3 Computational Complexity

The efficiency of SMA and SMC is achieved by randomly subsampling the columns of the residual kernel matrices for the selection of the dual directions. In Chapter 4 this was motivated by Lemma 4.1 in the context of selecting dual directions for kernel matrix approximation and a similar argument can be applied in this case. Here a simple test is conducted which gives insight into how the best column differs from the best in a subset as the size of the subset varies.

A synthetic dataset is composed of 2000 examples and 1000 features, with labels computed as a linear combination of 50 features plus a small uniform noise component, i.e. labels are found in an identical manner to that of the previous test. The data is centered and normalised so that each feature has unit norm, and SMC and SMA are run using sets of kernel matrix columns of different sizes for the selection of the dual directions.



(a) Cumulative square covariance of SMC. (b) Cumulative kernel target alignment of SMA.

FIGURE 5.2: Effect of using different sized random subsets of the kernel matrix columns at each iteration for the selection of $\boldsymbol{\alpha}_j$. Curves are shown in order of legend.

Figure 5.2 shows how the cumulative square covariance of SMC and cumulative kernel target alignment of SMA vary using the subset selection strategy. It is clear that using

500 kernel matrix columns is nearly indistinguishable from using all 2000 in both the SMC and SMA case. Even with only 100 kernel matrix columns the results are close to optimal, hence Lemma 4.1 is useful in this case. In the general case, the selection of c depends on the distribution of qualities of the kernel matrix columns. If this distribution is tail ended for example, then one would need a larger proportion of kernel matrix columns to find one within the top few.

Using Lemma 4.1 for approximating the solutions to Equations 5.4 and 5.5 reduces the complexity from $O(\ell^2)$ to $O(c\ell)$. Furthermore, one need only deflate c columns of the original kernel matrix, which at the jth iteration requires $O(cj\ell + c\ell p)$ operations, where p is the cost of a kernel computation[1]. The final complexity of training both SMA and SMC is therefore $O(ck^2\ell + ck\ell p)$, which is linear in the number of examples and does not directly depend on the original dimensionality of the data. The projection of a new test example is computed in $O(kp + k^2)$ operations since $\mathbf{k}'\mathbf{A}$ requires k kernel evaluations and the resulting vector is then multiplied by the $k \times k$ matrix $((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{KA})^{-1}$. This compares favourably with the $O(\ell p + k\ell)$ operations required for the projection of a test point onto k non-sparse dual directions.

The complexities of SMA and SMC are compared with several other feature extraction methods in Table 5.1[2]. Training sparse KPLS is efficient at $O(\nu k\ell^2 p + k^4)$ complexity, as its dual projection directions have at most $\nu\ell$ non-zero elements and each iteration requires the projection of the residual kernel matrix onto these directions. The k^4 term arises from the evaluation of the regression coefficients at each iteration. To evaluate a new test point, one must project onto a set of deflated dual directions as the original directions are computed relative to the deflated kernel matrices. These deflated directions are often non-sparse, hence the projection of a new test point requires $O(\ell p + k\ell)$ operations.

	Primal		Dual	
Algorithm	Train	Test	Train	Test
PCA	$O(m^2\ell + m^3)$	$O(mk)$	$O(\ell^2 p + \ell^3)$	$O(\ell p + k\ell)$
PLS	$O(mk\ell)$	$O(mk)$	$O(\ell^2 p + k\ell^2)$	$O(\ell p + k\ell)$
KB	$O(k\ell^3)$	$O(mk)$	$O(\ell^2 p + k\ell^3)$	$O(\ell p + k\ell)$
BLF	$O(mk\ell + kq)$	$O(mk)$	$O(\ell^2 p + k\ell^2 + kr)$	$O(\ell p + k\ell)$
Sp. KPLS	$O(mk\ell + k^4)$	$O(mk)$	$O(\nu k\ell^2 p + k^4)$	$O(\ell p + k\ell)$
SMA/SMC	N/A	N/A	$O(ck\ell p + ck^2\ell)$	$O(kp + k^2)$

Table 5.1: The training and test complexities of some feature extraction algorithms.

To conclude, note that the efficiency of training SMA and SMC arises through the particular deflation mechanism used, sparsity and the use of randomisation for selecting

[1]We assume that kernel matrix elements are computed on demand, however if they are precomputed one can use $p = 1$ for example.

[2]We assume that the data is already centered which is an $O(m\ell)$ operation in the primal case and $O(\ell^2)$ for a kernel matrix. For BLF, the complexities of computing the loss function and gradient of the loss function are denoted by q and r in the primal and dual cases respectively. For PLS and KPLS we assume there is a single label per example.

dual directions. Some of the methods listed in Table 5.1 could be made more efficient by using sparsity and randomisation in a similar fashion. On the other hand, KPLS for example does not lend itself easily to efficiency improvements in this way since its deflation operates on the entire kernel matrix. This contrasts with the deflation of Equation 3.4 which can be applied to kernel matrix columns independently.

5.4 Statistical Stability

An important question about the general framework is how the quality of the generated subspace varies over different data sets from the same source. Such results have previously been derived for KPCA in Shawe-Taylor et al. (2005) and a similar result will be shown here. As PLS and the sparse algorithms extract features that are predictive towards the labels it is intuitive to measure the covariance of the extracted features with the labels. One would like to know in which circumstances a high covariance on the training set is maintained for a test set.

The value of interest is the expectation of the cumulative square covariance of the features with respect to the labels. Define the function $f(\mathbf{x}, y^2) = \sum_{i=1}^{k}(y\phi(\mathbf{x})'\mathbf{w}_i)^2 = y^2\phi(\mathbf{x})'\mathbf{W}\mathbf{W}'\phi(\mathbf{x})$, where $\mathbf{W}$ has its columns composed of the directions $\mathbf{w}_i$, $i = 1, \ldots, k$. Our aim is to provide a lower bound on $\mathbb{E}[f(\mathbf{x}, y^2)]$. In the case of the general framework, the projections of the training examples are given by $\mathbf{T}$, and the corresponding empirical estimate of the expectation is

$$\begin{aligned} \hat{\mathbb{E}}[f(\mathbf{x}, y^2)] &= \frac{1}{\ell}\sum_{i=1}^{\ell} y_i^2\phi(\mathbf{x}_i)'\mathbf{W}\mathbf{W}'\phi(\mathbf{x}_i) \\ &= \frac{1}{\ell}\mathrm{tr}(\mathbf{T}'\tilde{\mathbf{Y}}\mathbf{T}), \end{aligned}$$

where $\tilde{\mathbf{Y}}$ is a diagonal matrix with diagonal entries $\tilde{\mathbf{Y}}_{ii} = y_i^2$, $i = 1, \ldots, \ell$.

In order to derive a bound on the expected covariance, Rademacher theory (Ledoux and Talagrand (1991)) is applied, which is concerned with how functions from a certain function class are able to fit random data. The path taken starts with the definition of the *Rademacher complexity* of a real-valued function class, which is a measure its capacity. The function class of f is introduced as well as its corresponding Rademacher complexity. One can bound the expectation of f using its Rademacher complexity and empirical expectation.

Definition 5.1. For a sample $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ generated by a distribution $\mathcal{D}$ on a set $\mathcal{X}$ and a real-valued function class $\mathcal{F}$ with domain $\mathcal{X}$, the *empirical Rademacher complexity* of $\mathcal{F}$ is the random variable

$$\hat{R}_\ell(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f\in\mathcal{F}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(\mathbf{x}_i) \right| \middle| \mathbf{x}_1, \dots, \mathbf{x}_\ell \right],$$

where $\sigma = \{\sigma_1, \dots, \sigma_\ell\}$ are independent uniform $\{\pm 1\}$-valued (Rademacher) random variables. The *Rademacher complexity* of $\mathcal{F}$ is

$$R_\ell(\mathcal{F}) = \mathbb{E}_S \left[\hat{R}_\ell(\mathcal{F}) \right] = \mathbb{E}_{S\sigma} \left[\sup_{f\in\mathcal{F}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(\mathbf{x}_i) \right| \right],$$

where $\mathbb{E}_S[\cdot]$ is the expectation over all samples S generated by a distribution $\mathcal{D}$.

Notice that the term inside the sup is proportional to the covariance of the Rademacher variables and f. In other words, if f can be chosen to fit random data easily in general then the Rademacher complexity of $\mathcal{F}$ is high. It follows that there is a greater possibility of detecting a spurious pattern which will not generalise to a test set.

In our case, the class of functions of interest is linear with bounded norm

$$\begin{aligned} &\left\{ f_B(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i) : \mathbf{x}_i \in \mathcal{X}, \sum_{i,j} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \le B^2 \right\} \\ &\subseteq \{ f_B(\mathbf{x}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle : \|\mathbf{w}\| \le B \} = \mathcal{F}_B, \end{aligned}$$

where $\mathcal{X}$ is the domain of $\mathcal{F}_B$, α_i, $i = 1, \dots, \ell$, are dual variables and B is an upper bound on the norm of the functions in $\mathcal{F}_B$. Note that this definition does not depend on any particular training set.

We now introduce a bound on the empirical Rademacher complexity of $\mathcal{F}_B$ using the following theorem from Bartlett and Mendelson (2003).

Theorem 5.2 (Bartlett and Mendelson (2003))**.** *If $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel, and $S = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$ is a sample of points from $\mathcal{X}$, then the empirical Rademacher complexity of the class $\mathcal{F}_B$ satisfies*

$$\hat{R}_\ell(\mathcal{F}_B) \le \frac{2B}{\ell} \sqrt{\sum_{i=1}^{\ell} \kappa(\mathbf{x}_i, \mathbf{x}_i)} = \frac{2B}{\ell} \sqrt{tr(\mathbf{K})}.$$

It is intuitive that the Rademacher complexity of $\mathcal{F}_B$ depends on B. Less so is its dependence on the trace of the kernel matrix. However, the trace of $\mathbf{K}$ is the sum of its eigenvalues, and also the cumulative variance of the examples. It follows that since

$\mathbf{w}$ is chosen from the space of the training examples, the corresponding Rademacher complexity is dependent on this quantity.

The last ingredient for our bound is a theorem which lower bounds the expectation of a function in terms of its empirical expectation and the Rademacher complexity of its function class. It results from a small modification of a theorem from Shawe-Taylor and Cristianini (2004) which provides an upper bound on the expectation.

Theorem 5.3. *Fix $\delta \in (0,1)$ and let $\mathcal{F}$ be a class of functions mapping from X to $[0,1]$. Let $(\mathbf{x}_i)_{i=1}^{\ell}$ be drawn independently according to a probability distribution $\mathcal{D}$. Then with probability at least $1-\delta$ over random draws of samples of size ℓ, every $f \in \mathcal{F}$ satisfies*

$$\begin{aligned} \mathbb{E}_{\mathcal{D}}\left[f(\mathbf{x})\right] &\geq \hat{\mathbb{E}}\left[f(\mathbf{x})\right] - R_\ell(\mathcal{F}) - \sqrt{\frac{\ln(2/\delta)}{2\ell}} \\ &\geq \hat{\mathbb{E}}\left[f(\mathbf{x})\right] - \hat{R}_\ell(\mathcal{F}) - 3\sqrt{\frac{\ln(2/\delta)}{2\ell}}. \end{aligned}$$

To tie up loose ends we introduce Theorem 5.4 which provides some useful properties of Rademacher complexity.

Theorem 5.4 (Bartlett and Mendelson (2003))**.** *Let $\mathcal{F}$ and $\mathcal{G}$ be classes of real functions. Then:*

1. *If $\mathcal{F} \subseteq \mathcal{G}$, then $\hat{R}_\ell(\mathcal{F}) \leq \hat{R}_\ell(\mathcal{G})$;*
2. *For every $c \in \mathbb{R}$, $\hat{R}_\ell(c\mathcal{F}) = |c|\hat{R}_\ell(\mathcal{F})$.*

The previous definitions and theorems are now used to derive a theorem which lower bounds the expectation of our original function $f(\mathbf{x}, y^2)$.

Theorem 5.5. *Let $f(\boldsymbol{x}, y^2) = y^2\boldsymbol{k}'\boldsymbol{B}\boldsymbol{B}'\boldsymbol{k}$ be formulated by performing general feature extraction on a randomly drawn training set S of size ℓ in the feature space defined by a kernel $\kappa(\boldsymbol{x}, \boldsymbol{z})$ and projecting new data using the dual projection matrix $\boldsymbol{B} = \boldsymbol{A}((\boldsymbol{T}'\boldsymbol{T})^{-1}\boldsymbol{T}'\boldsymbol{K}\boldsymbol{A})^{-1}$. Fix c and let $f(\boldsymbol{x}, y^2)$ belong to a class of linear functions $\mathcal{F}_c$ with norm bounded by c. With probability greater than $1-\delta$ over the generation of the sample S, the expected value of f is bounded by*

$$\mathbb{E}_D[f(\boldsymbol{x}, y^2)] \geq \frac{1}{\ell}tr(\boldsymbol{T}'\tilde{\boldsymbol{Y}}\boldsymbol{T}) - \frac{2c}{\ell}\sqrt{tr(\hat{\boldsymbol{K}})} - 3cP^2\sqrt{\frac{\ln(2/\delta)}{2\ell}}, \tag{5.12}$$

where $\hat{\boldsymbol{K}}$ is the kernel matrix defined by $\hat{\kappa}(\boldsymbol{x}, \boldsymbol{z}) = y(\boldsymbol{x})^2 y(\boldsymbol{z})^2 \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z})\rangle^2$, $y(\boldsymbol{x})$ is the label corresponding to $\boldsymbol{x}$ and P is radius of the hypersphere enclosing the examples $y_i\phi(\boldsymbol{x}_i)$, $i = 1, \ldots, \ell$.

Proof. First consider the following

$$\begin{aligned} f(\mathbf{x}, y^2) &= y^2 \phi(x)' \mathbf{W}\mathbf{W}' \phi(\mathbf{x}) \\ &= y^2 \sum_{i,j} \phi(\mathbf{x})_i \phi(\mathbf{x})_j (\mathbf{W}\mathbf{W}')_{ij} \\ &= y^2 \langle \tilde{\phi}(\mathbf{x}), \tilde{\mathbf{W}} \rangle_F, \end{aligned}$$

where $\tilde{\phi}(\mathbf{x})_{ij} = \phi(\mathbf{x})_i \phi(\mathbf{x})_j$ and $\tilde{\mathbf{W}}_{ij} = (\mathbf{W}\mathbf{W}')_{ij}$. Hence, f can be considered as a linear function of its inputs with norm bounded by $c \geq \|\tilde{\mathbf{W}}\|_F$, provided $\mathbf{x}$ is mapped to the feature space defined by $\tilde{\phi}$. The kernel function corresponding to $\tilde{\phi}$ is

$$\begin{aligned} \langle \tilde{\phi}(\mathbf{x}), \tilde{\phi}(\mathbf{z}) \rangle_F &= \sum_{i,j} \phi(\mathbf{x})_i \phi(\mathbf{x})_j \phi(\mathbf{z})_i \phi(\mathbf{z})_j \\ &= \sum_i \phi(\mathbf{x})_i \phi(\mathbf{z})_i \sum_j \phi(\mathbf{x})_j \phi(\mathbf{z})_j \\ &= \kappa(\mathbf{x}, \mathbf{z})^2. \end{aligned}$$

An application of Theorem 5.2 provides a bound on the empirical Rademacher complexity of $\mathcal{F}_c$,

$$\hat{R}_\ell(\mathcal{F}_c) \leq \frac{2c}{\ell} \sqrt{\sum_i^\ell y_i^4 \kappa(\mathbf{x}_i, \mathbf{x}_i)^2}. \tag{5.13}$$

Define $h(\mathbf{x}, y^2) = f(\mathbf{x}, y^2)/cP^2$ which belongs to a class of linear functions $\mathcal{H}$ with bounded norm. Using Theorem 5.3,

$$\mathbb{E}_\mathcal{D}\left[h(\mathbf{x}, y^2)\right] \geq \hat{\mathbb{E}}[h(\mathbf{x}, y^2)] - \hat{R}_\ell(\mathcal{H}) - 3\sqrt{\frac{\log(2/\delta)}{2\ell}}. \tag{5.14}$$

The Rademacher complexity of $\mathcal{H}$ is

$$\begin{aligned} \hat{R}_\ell(\mathcal{H}) &= \frac{1}{cP^2} \hat{R}_\ell(\mathcal{F}) \\ &\leq \frac{2}{\ell P^2} \sqrt{\sum_{i=1}^\ell y_i^4 \kappa(\mathbf{x}_i, \mathbf{x}_i)^2}, \end{aligned}$$

which follows from an application of part 2 of Theorem 5.4. Substituting into Equation 5.14 and multiplying by cP^2 gives

$$\mathbb{E}_{\mathcal{D}}[f(\mathbf{x}, y^2)] \geq \hat{\mathbb{E}}[f(\mathbf{x}, y^2)] - \frac{2c}{\ell}\sqrt{\mathrm{tr}(\hat{\mathbf{K}})} - 3cP^2\sqrt{\frac{\ln(2/\delta)}{2\ell}}, \tag{5.15}$$

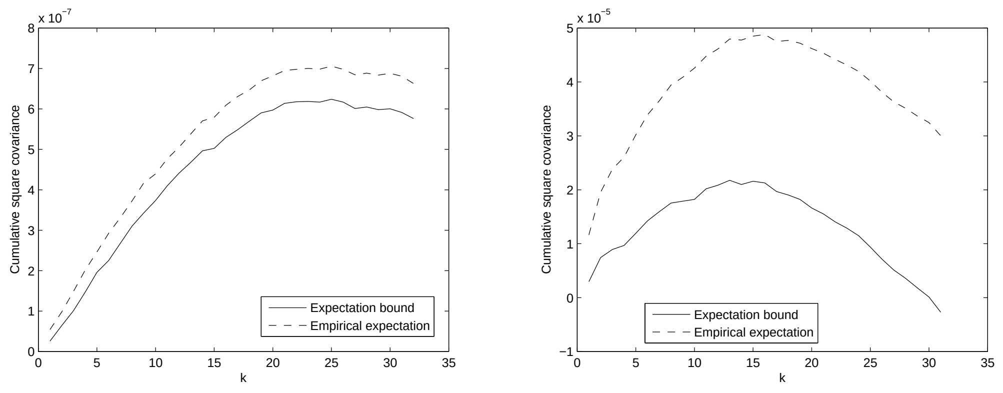
and then making a substitution $\hat{\mathbb{E}}[f(\mathbf{x}, y^2)] = \frac{1}{\ell}\mathrm{tr}(\mathbf{T}'\tilde{\mathbf{Y}}\mathbf{T})$ produces the required result.

□

This theorem indicates that the expected cumulative square covariance of the features produced under the general framework will be close to its empirical estimate provided the Rademacher and final terms are proportionately small. As one is working in kernel-defined feature spaces, the original dimensionality is unimportant and the final two terms grow inversely proportional to the root of the number of examples. For the middle term, the trace of $\hat{\mathbf{K}}$ can be understood as the cumulative variance of the examples given by $S' = \{\tilde{\phi}(\mathbf{x}_1)y_1^2, \ldots, \tilde{\phi}(\mathbf{x}_\ell)y_\ell^2\}$. Hence, to ensure stable patterns with high probability and allow a large covariance to be captured, one requires a rapid decay in the eigenvalues of $\hat{\mathbf{K}}$ and also a small value of $1/\sqrt{\ell}$.

To illustrate the effectiveness of Theorem 5.5 we consider a function $g(\mathbf{x}, y^2) = f(\mathbf{x}, y^2) - qk$ which introduces a cost q on the number of features k. This cost might represent extra computational or memory requirements and indicates a preference for low dimensional subspaces. We use two DELVE datasets for this evaluation. The first is one of the Pumadyn datasets composed of 8192 examples, 32 features, classified as "fairly linear" and with "high noise". The second is called "bank-32nm" from the bank family of datasets, and also has 8192 examples and 32 features. The original features are normalised, the value of c for Theorem 5.5 is estimated from the data, and we set $\delta = 0.1$. As a simple heuristic to select the value of q we use half the gradient of $\hat{\mathbb{E}}[f(\mathbf{x}, y^2)]$ at the first iteration. Using two thirds of the examples for training and the remaining for testing, we observe how the lower bound of $\mathbb{E}_D[g(\mathbf{x}, y^2)]$ varies with the number of iterations of SMC with $c = 500$. This bound is compared with the empirical expectation on the test examples.

Figure 5.3 shows the results of this test, and with the Pumadyn dataset the bound of $\mathbb{E}_D[g(\mathbf{x}, y^2)]$ is predictive of the empirical expectation of $g(\mathbf{x}, y^2)$ on the test set. Although the peaks of these curves are dependent on the manner in which q is selected, one could potentially use the bound as a method of selecting k in this case. Another possible use of Theorem 5.5 is for selecting the number of examples required for the lower bound of $\mathbb{E}_D[g(\mathbf{x}, y^2)]$ to be close to its empirical estimate. For these applications, Theorem 5.5 is most useful when the final two terms of Equation 5.12 are small relative to the empirical expectation. One can then say, with high probability, that the empirical



(a) Pumadyn dataset (b) Bank dataset

FIGURE 5.3: Plot of the lower bound of $\mathbb{E}_D[g(\mathbf{x}, y^2)]$ and empirical expectation of $g(\mathbf{x}, y^2)$ on a test set.

expectation of covariance of the features on a test set is close to the lower bound of $\mathbb{E}_D[g(\mathbf{x}, y^2)]$.

With the bank dataset, the bound is less close to the empirical expectation of $g(\mathbf{x}, y^2)$ on the test set. In this case a large value of P causes the final term of Equation 5.12 to be significant in comparison the empirical expectation of $f(\mathbf{x}, y^2)$. However, with this dataset most of the examples from S' lie in a hypersphere of small radius. Hence, to improve the statistical stability of the extracted features, one could remove the examples in S' with large norm which can be considered as outliers.

5.5 Computational Results

As SMA and SMC are supervised methods, the predictive performance of their features is compared with several other supervised feature extraction algorithms. This comparison is performed first on a few Bilkent University function approximation (Guvenir and Uysal (2000)) and UCI datasets and then on a large sample of the Reuters Corpus Volume 1 dataset (Rose et al. (2002)). This section is completed by applying SMA and SMC to an example face detection application, an area in which PCA has traditionally been a popular choice.

5.5.1 Bilkent Regression Experiment

This first experiment compares the regression performance of the features extracted by SMC and SMA to those generated by KPLS, sparse KPLS and rKOPLS. The datasets shown at the bottom of Table 5.2 are used, and each one has its examples and labels centered and normalised to have zero mean and unit norm. With large datasets only the first 1000 examples are used so that the tests can be run within a reasonable time frame.

Dataset	Examples	Features
Ionosphere	355	34
Sonar	208	60
SPECTF	267	44
WDBC	569	30
Ailerons	7154	40
Baseball	337	16
Pole Telecomm	9065	48

Table 5.2: Information about the UCI (top) and Bilkent University function approximation (bottom) datasets.

The methods are evaluated by following feature extraction with least squares regression, which in a kernel-defined feature space finds the minimum of $\|\mathbf{Kc}-\mathbf{y}\|^2$. The solution to this optimisation is $\mathbf{c} = \mathbf{K}^{-1}\mathbf{y}$, assuming $\mathbf{K}$ is full rank, which with the linear kernel is identical to OLS regression. After performing regression the root mean squared error is recorded, given by $\|f(\mathbf{X})-\mathbf{y}\|/\sqrt{\ell}$, where $f(\mathbf{X})$ is a vector of predicted labels of length ℓ. The error is measured using 5-fold cross validation, with an inner 5-fold cross validation loop for model selection.

The parameters for the feature extraction methods are selected using the following values. The number of extracted features is chosen from 1 to the rank of the data. The sparse KPLS sparsity parameter ν is selected from $\{0.125, 0.25, 0.5, 1.0\}$ and the heuristic used to compute dual projection directions is selected as either the Maximal Information (MI) criterion with a kernel cache size of 500 or the Maximum Residual (MR) criterion. For rKOPLS, the r parameter is chosen from $\{50, 100, 200, 400\}$. SMA and SMC are run using 500 kernel matrix columns for the selection of each dual projection direction. To test feature extraction in a dual space, each method is also used with the RBF kernel, with kernel width σ selected from $\{0.125, 0.25, \dots, 16\}$.

Method	Ailerons	Baseball	Pole Telecomm
All features	0.0168 (0.0021)	0.0329 (0.0043)	0.0232 (0.0007)
PLS	0.0160 (0.0005)	**0.0321** (0.0026)	**0.0231** (0.0005)
SMA	**0.0158** (0.0008)	0.0327 (0.0038)	0.0233 (0.0004)
SMC	0.0159 (0.0007)	0.0323 (0.0026)	**0.0231** (0.0005)
Sp. KPLS	0.0160 (0.0015)	0.0328 (0.0050)	0.0232 (0.0006)
rKOPLS	0.0315 (0.0035)	0.0453 (0.0122)	0.0316 (0.0010)
RBF Features	**0.0156** (0.0011)	0.0344 (0.0063)	0.0196 (0.0011)
RBF PLS	0.0158 (0.0012)	**0.0330** (0.0041)	**0.0094** (0.0014)
RBF SMA	0.0157 (0.0004)	0.0342 (0.0059)	0.0157 (0.0026)
RBF SMC	0.0162 (0.0019)	0.0343 (0.0059)	0.0124 (0.0014)
Sp. RBF KPLS	0.0161 (0.0015)	0.0353 (0.0036)	0.0117 (0.0005)
RBF rKOPLS	0.0170 (0.0020)	0.0371 (0.0057)	0.0114 (0.0020)

Table 5.3: Error rates of feature extraction followed by least squares regression.

The results, shown in Table 5.3, indicate that feature extraction often improves over the use of the least squares method with these datasets. Most notably, KPLS often results in the lowest error in both the linear and RBF spaces, although SMA and SMC are only slightly worse in general. Also observed was that the number of output features chosen for KPLS through the cross validation procedure is on average less than the corresponding number used for SMA and SMC. One might expect this to be the case since in contrast to KPLS, SMA and SMC have strict sparsity constraints. The rKOPLS method produces a single feature based on the projection onto a linear combination of a random subset of the examples. This clearly did not generalise well as rKOPLS results in the highest errors in most cases.

5.5.2 UCI Classification Experiment

Another common scenario is now studied, which is when feature extraction is followed by classification. In this case we apply k-Nearest Neighbour (KNN) and SVM classification, and use a selection of UCI datasets (listed at the top of Table 5.2). The KNN method is used since it classifies examples using the labels of nearby ones, and hence the features used are critical for achieving good accuracy. SVMs are more robust to irrelevant features, however one can still improve accuracy using feature extraction. The examples in the UCI datasets are preprocessed in an identical manner to that used in the previous experiment. We compare SMA and SMC to KPLS[3], sparse KPLS, Kernel Boosting and KBLF[4].

For most of these tests we use 5-fold cross validation repeated 3 times (with random permutations of the data) and 5-fold cross validation for model selection. Kernel Boosting, Least Absolute Deviation loss BLF and the RBF feature extraction algorithms are considerably slower than the other methods, hence, they are evaluated using 5-fold cross validation repeated 2 times with 3-fold cross validation for model selection.

The parameters for the feature extraction methods are selected using the same values used in the previous experiment. However, Kernel Boosting could iterate further than the rank of the data and was additionally allowed up to 1000 iterations in steps of 100. The SVM penalty parameter is selected from $\{0.125, 0.25, \ldots, 128\}$ and the number of neighbours k for the KNN method is selected from $\{1, 3, 5, 7, 9\}$. As a useful comparison to the RBF feature extraction methods we also apply the SVM and KNN classifiers to the original data using the RBF kernel, with σ selected from $\{0.125, 0.25, \ldots, 16\}$.

Tables 5.4 and 5.5 summarise the results of this experiment and we first discuss the primal results. A broad trend is that feature extraction results in larger improvements

[3]It may seem unusual to use regressive PLS in a classification task in light of the discriminative PLS method, however in the single label case the methods are identical.

[4]Sparse KBLF is not included in the comparison since Momma (2005) shows that it performs worse than an SVM.

	Ionosphere	Sonar	SPECTF	WDBC
All features	0.140 (0.02)	**0.150** (0.05)	0.244 (0.04)	0.034 (0.01)
PLS	0.110 (0.02)	0.179 (0.07)	0.230 (0.05)	0.032 (0.01)
SMA	0.106 (0.04)	0.215 (0.07)	0.253 (0.05)	0.047 (0.02)
SMC	0.105 (0.03)	0.203 (0.05)	0.235 (0.04)	0.045 (0.02)
Exp BLF	0.096 (0.03)	0.218 (0.07)	**0.206** (0.04)	0.049 (0.01)
Log BLF	0.087 (0.03)	0.215 (0.06)	**0.206** (0.06)	0.044 (0.02)
LAD BLF	0.099 (0.04)	0.241 (0.07)	0.219 (0.04)	0.043 (0.01)
Exp KB	0.134 (0.04)	0.237 (0.05)	0.238 (0.05)	0.046 (0.01)
Log KB	0.110 (0.03)	0.241 (0.07)	0.223 (0.04)	**0.027** (0.02)
Sp. KPLS	**0.084** (0.03)	0.180 (0.06)	0.236 (0.04)	0.044 (0.02)
RBF Features	0.141 (0.04)	0.133 (0.06)	0.235 (0.03)	0.033 (0.01)
RBF KPLS	**0.050** (0.02)	**0.107** (0.03)	0.223 (0.04)	**0.029** (0.01)
RBF SMA	0.053 (0.02)	0.168 (0.05)	**0.215** (0.07)	0.036 (0.02)
RBF SMC	0.057 (0.03)	0.173 (0.06)	0.243 (0.03)	0.052 (0.02)
Sp. RBF PLS	0.073 (0.04)	0.195 (0.07)	0.260 (0.04)	0.044 (0.02)

Table 5.4: Error rates of the extracted features with the KNN algorithm.

	Ionosphere	Sonar	SPECTF	WDBC
All features	0.134 (0.03)	0.231 (0.08)	0.205 (0.03)	0.025 (0.01)
PLS	0.132 (0.03)	0.224 (0.07)	0.201 (0.04)	0.028 (0.01)
SMA	0.133 (0.04)	0.224 (0.08)	0.230 (0.05)	0.028 (0.01)
SMC	**0.123** (0.03)	0.231 (0.07)	0.213 (0.03)	0.034 (0.02)
Exp BLF	0.138 (0.03)	0.234 (0.06)	0.224 (0.06)	0.040 (0.01)
Log BLF	0.133 (0.03)	0.228 (0.06)	0.223 (0.05)	0.032 (0.01)
LAD BLF	0.124 (0.04)	0.239 (0.05)	0.228 (0.04)	0.031 (0.01)
Exp KB	0.131 (0.03)	**0.205** (0.03)	**0.200** (0.04)	0.033 (0.01)
Log KB	0.129 (0.03)	0.217 (0.05)	0.206 (0.03)	0.023 (0.01)
Sp. KPLS	0.136 (0.04)	0.246 (0.06)	0.209 (0.04)	**0.021** (0.01)
RBF Features	**0.056** (0.03)	0.145 (0.07)	0.200 (0.03)	0.031 (0.01)
RBF KPLS	0.069 (0.02)	**0.122** (0.03)	0.219 (0.03)	**0.027** (0.01)
RBF SMA	0.057 (0.03)	0.146 (0.06)	**0.194** (0.03)	0.030 (0.02)
RBF SMC	0.057 (0.03)	0.141 (0.04)	0.202 (0.03)	0.031 (0.01)
Sp. RBF PLS	**0.056** (0.03)	0.137 (0.09)	0.291 (0.10)	**0.027** (0.01)

Table 5.5: Error rates of the extracted features with the SVM algorithm.

in the error with the KNN than with the SVM. This can be explained by the good generalisation implied by maximising the margin for the SVM (Vapnik (1998)), whereas KNN is more sensitive to noise. As with the regression results, SMC and SMA have comparable errors to many of the other feature extraction methods despite using only a single example for each projection vector. They are particularly effective on the Ionosphere dataset, improving the error obtained with the KNN method from 0.140 to 0.106 and 0.105 respectively, and SMC provides the lowest error with the linear SVM. Comparing SMC to PLS shows that the addition of the sparsity constraint does not have a large impact on the error. A possible explanation is that the distribution of examples in these datasets allows a single example to result in a covariance close to that obtained using a linear combination of the examples. Notice that the features produced by Kernel Boosting often lead to low error rates, however this is frequently at the expense of a higher dimensionality than that of the original data.

When considering the RBF features spaces, the low errors obtained on the Ionosphere and Sonar datasets with both the KNN and SVM imply a non-linear relationship between the features and labels. The RBF KPLS approach improves over using the plain RBF features with the KNN, and also for the Sonar and WDBC datasets with the SVM. KPLS also frequently has the lowest error rate when compared to the other RBF feature extraction methods. One possible explanation is that since the RBF feature space is infinite dimensional, examples are more spread out in this space. Therefore, enforcing sparsity on the projection directions has more of a detrimental effect on the quality of the resulting features. Having noted this however, SMA and SMC are only slightly worse than KPLS, and also comparable to sparse KPLS.

5.5.3 Text Retrieval

We demonstrate the scalability of the sparse algorithms by running them on the Reuters Corpus Volume 1 news database. The full database consists of about 800,000 news articles (the period of a whole year), but only 20,000 examples are considered from first three months of the Economics branch. As a preprocessing step, the Porter stemmer has been used to reduce all words to their stems which yields 136,469 distinct terms. Labels are assigned according to whether the articles are about the topic "Government Finance", with approximately 37% of the examples positively labelled. Features are extracted on this dataset by sparse KPLS, SMC and SMA and then used to train a linear SVM. In this case, we do not generate results using PLS, Kernel Boosting and BLF since the computational and memory requirements needed to run these methods are prohibitively high.

The process used to conduct this experiment is similar to that of the previous one. In this case the data is not centered since the examples are sparse and centering in general removes sparsity. The tests are run using a single repetition of 3-fold cross validation with an inner 3-fold cross validation loop, used on 2000 randomly sampled training examples, for model selection. For each method we record the average precision since it is a standard measure in information retrieval. It is defined as the cumulative precision after each relevant document is retrieved divided by the number of relevant documents, where precision is the proportion of relevant documents to all the documents retrieved. Average precision emphasises returning more relevant documents high up in the ranking.

At the model selection stage, the parameters are chosen as follows. Each feature extraction algorithm is run from 100 to 400 iterations in steps of 100. Sparse KPLS is run using both the MI heuristic with a kernel cache size of 300 and the MR heuristic, with

ν chosen from $\{0.2, 0.4, 0.8\}$. SMA and SMC are run using 500 kernel columns for the selection of the dual vector. The SVM is trained using a linear kernel and its penalty parameter is chosen from $\{0.125, 0.25, \ldots, 128\}$. Since the class labels are imbalanced, LIBSVM is parametrised so as to weight different classes with different values. The weight for each class is fixed as the percentage of examples of the opposite class.

	Average Precision	Projections	Sparsity	SVs
SVM	**0.847** (0.010)	-	-	8864
SMA	**0.848** (0.007)	366	366	9964
SMC	0.823 (0.009)	400	400	9953
Sparse KPLS	0.776 (0.032)	400	13334	6690

Table 5.6: Average precisions on the Reuters dataset. Sparsity is the number of kernel evaluations required for the projection of a new test example and SVs is the number of support vectors used.

Table 5.6 shows that SMC and SMA outperform sparse KPLS, both in terms of average precision and the number of kernel evaluations required to project a new example. Recall that for SMA and SMC, the latter quantity is simply the number of iterations since the projection of a new test example requires only the selected training examples and the precomputed matrix $((\mathbf{T}'\mathbf{T})^{-1}\mathbf{T}'\mathbf{K}\mathbf{A})^{-1}$. Notice that SMA achieves a similar average precision to the raw SVM using a much smaller dimensionality. We might hope for an improvement over the SVM results, however, Joachims (1998) shows that few features in text categorisation datasets are irrelevant. With sparse KPLS, we believe that the heuristics used for selecting the non-zero elements in the dual directions were ineffective and resulted in model selection making a preference for non-sparse directions.

The number of support vectors may appear to be useful in computing the total number of kernel evaluations required for the classification of a new example. However, once a new example is projected into the sparse subspace one can work in the primal space and the number of support vectors is no longer relevant. In this case an efficient primal space algorithm can be used, for example the one described in Joachims (2006). This has a complexity of $O(s\ell)$ where s is the average number of non-zero features per example. When applied to sparse data, the overall complexity of SMA or SMC followed by this SVM classifier is $O(ck^2\ell + ck\ell p)$, i.e. linear in the number of examples.

5.5.4 Face Detection

Facial recognition has seen a lot of attention in recent years (Zhao et al. (2003)), with applications such as Human Computer Interaction (HCI), law enforcement and security. An important stage in any face recognition system is face detection, which is concerned with determining whether a particular image contains a face. If a raw image is used as the input to a pattern recognition algorithm, the number of features is often large and feature extraction is a useful process. PCA and KPCA have enjoyed success in face recognition, since in face datasets most of the variation can be captured with relatively few projection directions (Turk and Pentland (1991a)). In this final experiment, we apply KPCA, SMA and SMC followed by SVM classification to the MIT CBCL Face Dataset 1 (cbc (1996)). The aim is to observe how effectively the examples are ranked in each subspace using a Receiver Operating Characteristic (ROC) curve. This curve shows the quality of the ranking of examples using different classifier thresholds. It plots the false positive rate against the true positive rate, where the false positive rate is the fraction of negative examples that are incorrectly predicted as positive and the true positive rate is the fraction of positive examples that are correctly classified.

The MIT CBCL face dataset consists of a training set composed of 2,429 faces and 4,548 non-faces, and a test set with 472 faces and 23,573 non-faces. The test set is particularly challenging since it contains those non-face images from the CMU Test Set 1 (Rowley et al. (1998)) which are most similar to faces. All images are captured in grayscale at a resolution of 19×19 pixels, but rather than use pixel values as features, we use those proposed in Viola and Jones (2004) since they give excellent performance in conjunction with their face detection system. Viola and Jones describe three kinds of features computed for each image. A two-rectangle feature is the difference between the sum of pixels within two adjacent rectangular regions of same size and shape. A three-rectangle feature computes the sum of two outside rectangles subtracted from the sum of a center rectangle, and a four-rectangle feature is the difference between diagonal pairs of rectangles. These features, illustrated in Figure 5.4, are computed for each image over many scales yielding 30,798 features per image.

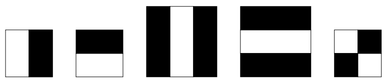


FIGURE 5.4: Illustration of the different kinds of features proposed by Viola and Jones. Each feature is the sum of the pixels values within the white regions subtracted from the sum of the pixels values within the black regions.

The following procedure is used for this experiment. A reduced training set is formed using 3000 examples sampled from the original training set. The Viola and Jones features are used in the RBF feature space with $\sigma = 0.25$, as preliminary tests on the training set showed that this gave good performance with an SVM. Model selection is performed using 3-fold cross validation repeated twice using a sample of 1500 examples from the training set. The parameters which result in the highest value for the Area Under the ROC Curve (AUC) measure are selected at this stage. KPCA, SMA and SMC are iterated from 100 to 1000 times in steps of 100 and the SVM penalty parameter is selected from $\{0.125, 0.25, \ldots, 64\}$. For SMA and SMC we use 500 kernel matrix columns for the selection of the dual projection directions. To evaluate the learned models, a ROC curve is recorded for the predictions made on the test set.

Figure 5.5 shows the resulting ROC curves for the SVM, KPCA, SMA and SMC. There is little to differentiate the curves and all cover approximately 90% of the true positives

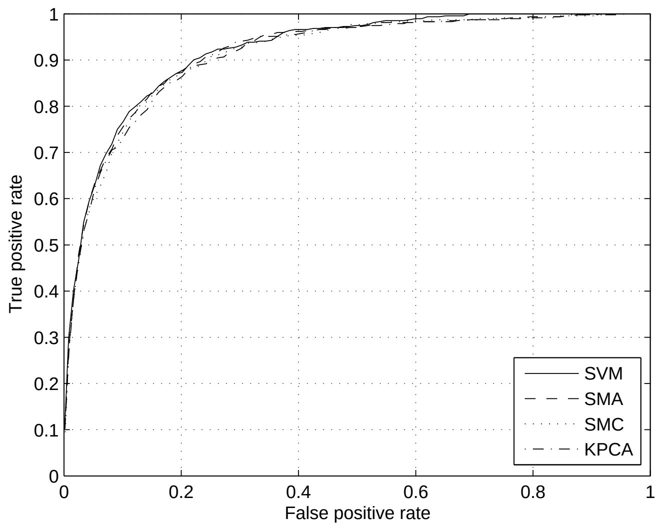


FIGURE 5.5: ROC curves for the MIT CBCL face dataset I.

with a false positive rate of 0.22 to 0.25. This is an improvement over the results given in Heisele et al. (2000) which uses the full training set. Notice that the number of kernel evaluations for the classification of a new test example using the SVM is 1022, compared to 700 and 900 using SMA and SMC respectively. Furthermore, one would expect KPCA to perform worse than SMA and SMC in terms of the number of features it requires to match the SVM since it is an unsupervised method.

5.6 Summary

This chapter has built upon the work presented in the previous ones, by the use of the kernel general feature extraction framework and the sparsity mechanism used in Chapter 4. The focus was on supervised feature extraction and two new sparse algorithms were formulated by maximising kernel target alignment and covariance respectively. Efficient training was achieved by randomly subsampling the kernel matrix columns to choose the dual directions. This was shown to be effective in improving computational efficiency without being significantly worse than the optimal solution. The resulting methods can be trained with a complexity that is linear in the number of examples. Furthermore, the projection of a new test example requires only k kernel evaluations where k is the output dimensionality.

An important property of any learning algorithm is its ability to generalise to unseen examples. To investigate how the covariance of the features produced by the kernel general framework differed between training and test sets, we derived an upper bound on the covariance using the Rademacher approach. Although the bound is not predictive in every case, it suggests when the covariance of the features is stable across different data samples from the same source. In these cases, one could use the bound as a stopping criterion for example.

We showed that the features produced by SMA and SMC compare well with other successful feature extraction methods in both regression and classification scenarios. Scalability of the algorithms was demonstrated using 20,000 examples from the Reuters Corpus Volume 1 dataset. With this data, SMA was shown to match the performance of the original 136,469 features in conjunction with an SVM using just 366 output features. Furthermore, on an example face detection problem, SMA and SMC require fewer features than KPCA to equal the performance of an SVM across a range of thresholds.

Chapter 6

Learning Underlying Semantics of Two-Viewed Data

We previously introduced CCA as a useful approach for finding the underlying semantics of a set of paired examples. Its kernel variant finds the solutions to an eigenproblem of size $2\ell \times 2\ell$ implying an $O(\ell^3)$ complexity, however by using the Incomplete Cholesky decomposition to approximate the kernel matrices one can often reduce the size of the eigenproblem. The resulting algorithm can be trained in $O(\ell k^3)$ complexity if the full eigen-decomposition is used, where k is the output rank of the approximated kernel matrices. However, it has already been shown in Chapter 4 that the Incomplete Cholesky decomposition is not an ideal choice for approximating a kernel matrix, and instead KFA or GSD-KPLS obtain better approximations at a similar computational cost. It would be better still to target the approximations towards maximising correlation, which is the approach we take in this chapter by formulating a cardinality constrained KCCA algorithm.

The new KCCA optimisation limits the cardinality of the dual directions. It is infeasible to solve directly, and we propose a simple and efficient approximation which uses deflation. A variation of this method is formulated which can select features from one view and examples from the other. Using empirical evidence we show that the approximations to the sparse optimisations, whilst not close to optimal, have certain other desirable properties. We further demonstrate the effectiveness of the sparse CCA methods on a selection of synthetic and real-world datasets. The chapter concludes with their application to a novel enzyme function prediction scenario, in which we attempt to predict the reactions catalysed by enzymes.

6.1 Imposing Sparsity in (K)CCA

One approach for formulating a sparse KCCA algorithm is to maximise the correlation between $\mathbf{K}^x\boldsymbol{\alpha}$ and $\mathbf{K}^y\boldsymbol{\beta}$ subject to the constraint that the dual vectors have one non-zero element, and then deflate. A difficulty with the approach is that the correlation of the resulting features is restricted by that of the respective residual kernel matrix columns. One can always improve upon the features evaluated in this way by using less sparse dual directions, however, doing so often increases the computational cost at each iteration. Instead, consider maximising the (regularised) correlation subject to a cardinality constraint on the dual vectors,

$$\begin{aligned} \max \quad & \boldsymbol{\alpha}'\mathbf{K}^{x\prime}\mathbf{K}^y\boldsymbol{\beta} \\ \text{s.t.} \quad & (1-\tau)\boldsymbol{\alpha}'\mathbf{K}^{x\prime}\mathbf{K}^x\boldsymbol{\alpha} + \tau\boldsymbol{\alpha}'\mathbf{K}^x\boldsymbol{\alpha} = 1 \\ & (1-\tau)\boldsymbol{\beta}'\mathbf{K}^{y\prime}\mathbf{K}^y\boldsymbol{\beta} + \tau\boldsymbol{\beta}'\mathbf{K}^y\boldsymbol{\beta} = 1 \\ & \mathrm{card}(\boldsymbol{\alpha}) \leq p \\ & \mathrm{card}(\boldsymbol{\beta}) \leq p, \end{aligned} \tag{6.1}$$

which is related to the p-sparse optimisations observed in Chapter 5. A useful property of this optimisation is that in certain circumstances it results in an eigenvalue problem which supplies at most p directions, hence it avoids the need for deflation. Observe that the optimisation is also similar to the cardinality constrained generalised eigenvalue problems considered in Moghaddam et al. (2006a), however in our case there are two cardinality constraints.

The optimisation of Equation 6.1 can be seen as finding the maximal correlation subject to the constraint that $\mathbf{K}^x\boldsymbol{\alpha}$ and $\mathbf{K}^y\boldsymbol{\beta}$ are in the space of at most p columns of $\mathbf{K}^x$ and $\mathbf{K}^y$ respectively. It is identical to the one used for KCCA when $p = \ell$, and if we define $q = \max(\mathrm{rank}(\mathbf{K}^x), \mathrm{rank}(\mathbf{K}^y))$ and let $p = q$, one can always find a maximum correlation which is equivalent to that obtained when $p = \ell$. Furthermore, since the solution to Equation 6.1 never decreases as p increases, it follows that it is upper bounded by the largest eigenvalue for the KCCA eigenproblem. It may seem that penalising the norm of the projection vectors is unnecessary as they have a sparsity constraint. However, limiting the cardinality of the directions does not necessarily ensure statistical stability, and Theorem 2.2 implies that regularisation in the way used above is a useful property in a KCCA algorithm.

A problem with Equation 6.1 is that it cannot be solved using the Lagrange method since the constraints are non-differentiable. Suppose that we have found which indices of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are non-zero and they are denoted by $I_x = \{i_1^x, \ldots, i_q^x\}$ and $I_y = \{i_1^y, \ldots, i_r^y\}$ respectively for $q, r \leq p$. Let $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ be the vectors of non-zero entries in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, then Equation 6.1 can be written as

$$\begin{aligned} \max \quad & \tilde{\boldsymbol{\alpha}}'\mathbf{K}^x[,I_x]'\mathbf{K}^y[,I_y]\tilde{\boldsymbol{\beta}} \\ \text{s.t.} \quad & (1-\tau)\tilde{\boldsymbol{\alpha}}'\mathbf{K}^x[,I_x]'\mathbf{K}^x[,I_x]\tilde{\boldsymbol{\alpha}} + \tau\tilde{\boldsymbol{\alpha}}'\mathbf{K}^x[I_x,I_x]\tilde{\boldsymbol{\alpha}} = 1 \\ & (1-\tau)\tilde{\boldsymbol{\beta}}'\mathbf{K}^y[,I_y]'\mathbf{K}^y[,I_y]\tilde{\boldsymbol{\beta}} + \tau\tilde{\boldsymbol{\beta}}'\mathbf{K}^y[I_y,I_y]\tilde{\boldsymbol{\beta}} = 1, \end{aligned} \tag{6.2}$$

and solved using an eigenproblem similar to the KCCA one,

$$\begin{pmatrix} \mathbf{0} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{\alpha}} \\ \tilde{\boldsymbol{\beta}} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{C}^r_{xx} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^r_{yy} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{\alpha}} \\ \tilde{\boldsymbol{\beta}} \end{pmatrix}, \tag{6.3}$$

where $\mathbf{C}^r_{xx} = (1-\tau)\mathbf{K}^x[,I_x]'\mathbf{K}^x[,I_x] + \tau\mathbf{K}^x[I_x,I_x]$, $\mathbf{C}^r_{yy} = (1-\tau)\mathbf{K}^y[,I_y]'\mathbf{K}^y[,I_y] + \tau\mathbf{K}^y[I_y,I_y]$, $\mathbf{C}_{xy} = \mathbf{C}'_{yx} = \mathbf{K}^x[,I_x]'\mathbf{K}^y[,I_y]$.

Observe that the above equation is of size $2p \times 2p$, and therefore efficient to solve for small p. Since it is a generalisation of the KCCA eigenproblem, it should come as no surprise that their projections possess the same properties. The features produced in this case are orthogonal in the following ways, provided $\tau = 0$,

$$\begin{aligned} \tilde{\boldsymbol{\alpha}}_i'\mathbf{K}^x[,I_x]'\mathbf{K}^x[,I_x]\tilde{\boldsymbol{\alpha}}_j = 0 \quad &\Rightarrow \quad \boldsymbol{\alpha}_i'\mathbf{K}^{x\prime}\mathbf{K}^x\boldsymbol{\alpha}_j = 0 \\ \tilde{\boldsymbol{\beta}}_i'\mathbf{K}^y[,I_y]'\mathbf{K}^y[,I_y]\tilde{\boldsymbol{\beta}}_j = 0 \quad &\Rightarrow \quad \boldsymbol{\beta}_i'\mathbf{K}^{y\prime}\mathbf{K}^y\boldsymbol{\beta}_j = 0 \\ \tilde{\boldsymbol{\alpha}}_i'\mathbf{K}^x[,I_x]'\mathbf{K}^y[,I_y]\tilde{\boldsymbol{\beta}}_j = 0 \quad &\Rightarrow \quad \boldsymbol{\alpha}_i'\mathbf{K}^{x\prime}\mathbf{K}^y\boldsymbol{\beta}_j = 0 \end{aligned}$$

for $i \neq j$.

An exact solution for the optimisation of Equation 6.1 requires a combinatorial procedure and is NP-hard. To improve efficiency, we use an approximation method for I_x and I_y which greedily selects new indices. As with many search strategies there are two general approaches that one can adopt: *forward selection* and *backward elimination*. In backward elimination, one starts with the full set of indices $[1, \ldots, \ell]$ and successively removes elements until p or fewer remain. If the suitability of the ith and jth columns of $\mathbf{K}_x$ and $\mathbf{K}_y$ respectively can be evaluated at cost $O(\ell)$, then the complexity of backward elimination grows close to $O(\ell^4)$ if $p \ll \ell$. In contrast forward selection, which starts with an empty set of indices and adds to them, is closer to $O(\ell^3)$ cost and hence is preferred.

One possible strategy for choosing indices is to start with $I_x, I_y = \{\}$, iterate through all pairs of possible indices, and choose the one which has the maximum value of Equation 6.2. The process is repeated until the desired number of indices is found. Unfortunately, since $\mathbf{K}^x\boldsymbol{\alpha}$ is in the space of the columns corresponding to the non-zero elements of $\boldsymbol{\alpha}$, choosing a column within this space does not provide any additional scope for increasing correlation. It follows that a useful step at each iteration is to deflate the columns of $\mathbf{K}^x$

and $\mathbf{K}^y$ so that they are orthogonal to the previously selected ones, i.e. using the general kernel feature extraction deflation. Notice that one could also potentially use the KPCA deflation, however the selected residual kernel matrix columns are not orthogonal and its application is less intuitive in this case.

In conjunction with the deflations of $\mathbf{K}^x$ and $\mathbf{K}^y$ using Equation 3.4, the optimisation used to select indices at the jth iteration is

$$\begin{aligned}
\max \quad & \boldsymbol{\alpha}_j' \mathbf{K}_j^{x\prime} \mathbf{K}_j^y \boldsymbol{\beta}_j \\
\text{s.t.} \quad & (1-\tau)\boldsymbol{\alpha}_j' \mathbf{K}_j^{x\prime} \mathbf{K}_j^x \boldsymbol{\alpha}_j + \tau \boldsymbol{\alpha}_j' \mathbf{K}^x \boldsymbol{\alpha}_j = 1 \\
& (1-\tau)\boldsymbol{\beta}_j' \mathbf{K}_j^{y\prime} \mathbf{K}_j^y \boldsymbol{\beta}_j + \tau \boldsymbol{\beta}_j' \mathbf{K}^y \boldsymbol{\beta}_j = 1 \\
& \text{card}(\boldsymbol{\alpha}_j) = 1 \\
& \text{card}(\boldsymbol{\beta}_j) = 1,
\end{aligned} \tag{6.4}$$

which is equivalent to Equation 6.1 with $p = 1$. We have already seen several examples of optimisations involving dual directions with only one non-zero element, and one can approximate the above using Lemma 4.1. A limit of c columns of $\mathbf{K}_j^x$ and $\mathbf{K}_j^y$ for the computation of $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$ respectively implies a complexity of $O(c^2\ell)$. Deflating two $\ell \times c$ matrices $j-1$ times, requires $O(c(j-1)\ell)$ computations and at the end of the algorithm an eigenproblem is solved at $O(p^3)$ cost. It follows that the overall complexity of our algorithm (called p-KCCA) is $O(c^2p\ell + cp^2\ell)$, which is linear in the number of examples and quadratic in the number of iterations p. The pseudo code for p-KCCA is shown in Algorithm 18.

6.1.1 A primal-dual Variant

Sparsity in dual directions ensures that the equivalent primal vectors are chosen from a linear combination of a subset of the examples. One can also enforce sparsity directly in the primal vectors so that the projections use a subset of the original features. This approach is used in Torres et al. (2007) and Hardoon and Shawe-Taylor (2007a) in conjunction with algorithms based on CCA. We propose a method which follows the same general blueprint as Hardoon and Shawe-Taylor (2007a), i.e. finds sparse primal projection vectors in one view and sparse dual projection directions in the other. We refer to this blueprint as a *primal-dual* CCA technique, and later it will become clear why it is intuitive for the enzyme function prediction task.

To adapt p-KCCA to find primal sparse directions in the $\mathcal{X}$ view, one solves the following optimisation based on Equation 6.1

Algorithm 18 Pseudo code for p-Kernel Canonical Correlation Analysis.

```
Inputs: Kernel $\mathbf{K}^x \in \mathbb{R}^{\ell\times\ell}$, $\mathbf{K}^y \in \mathbb{R}^{\ell\times\ell}$, dimension $k$, sparsity $p$, sample size $c$, regularisation parameter $\tau$
Process:
1) Indices $I_x = \{\,\}$, and $I_y = \{\,\}$
2) For $j = 1,\dots,p$
    (a) Randomly pick $I_x^c, I_y^c \in [\ell]^c$ and let $\tilde{\mathbf{K}}_1^x = \mathbf{K}^x[, I_x^c]$, $\tilde{\mathbf{K}}_1^y = \mathbf{K}^y[, I_y^c]$
    (b) For $i = 1,\dots,j-1$
        i) Let $\boldsymbol{\tau}_i^x = \mathbf{K}_i^x\boldsymbol{\alpha}_i$ and $\boldsymbol{\tau}_i^y = \mathbf{K}_i^y\boldsymbol{\beta}_i$
        ii) Deflate: $\tilde{\mathbf{K}}_{i+1}^x = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_i^x\boldsymbol{\tau}_i^{x\prime}}{\boldsymbol{\tau}_i^{x\prime}\boldsymbol{\tau}_i^x}\right)\tilde{\mathbf{K}}_i^x$ and $\tilde{\mathbf{K}}_{i+1}^y = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_i^y\boldsymbol{\tau}_i^{y\prime}}{\boldsymbol{\tau}_i^{y\prime}\boldsymbol{\tau}_i^y}\right)\tilde{\mathbf{K}}_i^y$
    (c) End
    (d) Solve for scalars $s$ and $t$
        $$\begin{array}{ll} \max & \boldsymbol{\alpha}_j'\mathbf{K}_j^{x\prime}\mathbf{K}_j^y\boldsymbol{\beta}_j \\ \text{s.t.} & (1-\tau)\boldsymbol{\alpha}_j'\mathbf{K}_j^{x\prime}\mathbf{K}_j^x\boldsymbol{\alpha}_j + \tau\boldsymbol{\alpha}_j'\mathbf{K}^x\boldsymbol{\alpha}_j = 1 \\ & (1-\tau)\boldsymbol{\beta}_j'\mathbf{K}_j^{y\prime}\mathbf{K}_j^y\boldsymbol{\beta}_j + \tau\boldsymbol{\beta}_j'\mathbf{K}^y\boldsymbol{\beta}_j = 1 \\ & \boldsymbol{\alpha}_j \in s\cdot\mathbf{I}[, I_x^c], \boldsymbol{\beta}_j \in t\cdot\mathbf{I}[, I_y^c] \end{array}$$
    (e) Let chosen directions be $s\mathbf{e}_{i_q^x}$ and $t\mathbf{e}_{i_r^y}$. Update $I_x \leftarrow I_x \cup \{i_q^x\}$ and $I_y \leftarrow I_y \cup \{i_r^y\}$.
3) End
4) Solve Equation 6.3 using $I_x$ and $I_y$
5) Define $\boldsymbol{\alpha}_j = \mathbf{I}[, I_x]\tilde{\boldsymbol{\alpha}}_j$ and $\boldsymbol{\beta}_j = \mathbf{I}[, I_y]\tilde{\boldsymbol{\beta}}_j$, $j = 1,\dots,k$
Output: Directions $\boldsymbol{\alpha}_j$, $\boldsymbol{\beta}_j$ and projections $\mathbf{K}^x\boldsymbol{\alpha}_j$, $\mathbf{K}^y\boldsymbol{\beta}_j$, $j = 1,\dots,k$
```

$$\begin{array}{ll} \max & \mathbf{u}'\mathbf{X}'\mathbf{K}^y\boldsymbol{\beta} \\ \text{s.t.} & (1-\tau)\mathbf{u}'\mathbf{X}'\mathbf{X}\mathbf{u} + \tau\mathbf{u}'\mathbf{u} = 1 \\ & (1-\tau)\boldsymbol{\beta}'\mathbf{K}^{y\prime}\mathbf{K}^y\boldsymbol{\beta} + \tau\boldsymbol{\beta}'\mathbf{K}^y\boldsymbol{\beta} = 1 \\ & \text{card}(\mathbf{u}) \le p \\ & \text{card}(\boldsymbol{\beta}) \le p, \end{array} \tag{6.5}$$

where the number of non-zero entries in $\mathbf{u}$, and hence number of columns of $\mathbf{X}$ used in the projections, is limited to p. Notice that for $p = \max(m, \ell)$, the solution to the above is equivalent to same optimisation without cardinality constraints. For known indices I_x and I_y, the optimisation of Equation 6.5 can be written as

$$\begin{pmatrix} \mathbf{0} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \tilde{\mathbf{u}} \\ \tilde{\boldsymbol{\beta}} \end{pmatrix} = \lambda\begin{pmatrix} \mathbf{C}_{xx}^r & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{yy}^r \end{pmatrix}\begin{pmatrix} \tilde{\mathbf{u}} \\ \tilde{\boldsymbol{\beta}} \end{pmatrix}, \tag{6.6}$$

where $\mathbf{C}^r_{xx} = (1-\tau)\mathbf{X}[,I_x]'\mathbf{X}[,I_x] + \tau\mathbf{I}$, and $\mathbf{C}_{xy} = \mathbf{C}'_{yx} = \mathbf{X}[,I_x]'\mathbf{K}^y[,I_y]$ in this case. The final projection directions are computed using $\mathbf{u} = \mathbf{I}[,I_x]\tilde{\mathbf{u}}$ and $\boldsymbol{\beta} = \mathbf{I}[,I_y]\tilde{\boldsymbol{\beta}}$ respectively.

The indices I_x and I_y are computed using an analogous method to the one used for p-KCCA. We set $\mathbf{X}_1 = \mathbf{X}$ and $\mathbf{K}^y_1 = \mathbf{K}^y$ and solve the following at the jth iteration

$$\begin{array}{ll} \max & \mathbf{u}_j'\mathbf{X}_j'\mathbf{K}^y_j\boldsymbol{\beta}_j \\ \text{s.t.} & (1-\tau)\mathbf{u}_j'\mathbf{X}_j'\mathbf{X}_j\mathbf{u}_j + \tau\mathbf{u}_j'\mathbf{u}_j = 1 \\ & (1-\tau)\boldsymbol{\beta}_j'\mathbf{K}^{y\prime}_j\mathbf{K}^y_j\boldsymbol{\beta}_j + \tau\boldsymbol{\beta}_j'\mathbf{K}^y\boldsymbol{\beta}_j = 1 \\ & \operatorname{card}(\mathbf{u}_j) = 1 \\ & \operatorname{card}(\boldsymbol{\beta}_j) = 1, \end{array} \tag{6.7}$$

using the deflations of Equations 3.1 and 3.4 respectively to compute the residual matrices. One limitation of the deflation approach is that since each deflation is a rank-one reduction, one cannot choose more indices than the minimum rank of the corresponding matrices. However, if at the jth iteration the rank of $\mathbf{K}^y_j$ is zero for example then one can always recover additional indices for $\mathbf{X}_j$ by fixing $\mathbf{K}^y_j \leftarrow \mathbf{K}^y_1$ and then continuing without deflating $\mathbf{K}^y_j$.

The optimisation of Equation 6.7 is approximated by selecting d columns of $\mathbf{X}_j$ and c columns of $\mathbf{K}^y_j$ to find $\mathbf{u}_j$ and $\boldsymbol{\beta}_j$. The resulting method, which we shall refer to as p-primal-dual CCA (p-PDCCA), is shown in Algorithm 19. The complexity of the algorithm is $O(pq^2\ell + qp^2\ell)$, where $q = \max(c,d)$. This is an improvement over the method presented in Hardoon and Shawe-Taylor (2007a), which is at least $O(k\ell^2 + km^2)$.

6.1.2 Efficient Centering

We mentioned in Chapter 3 that data centering can help to reduce compound numerical errors that occur through deflation. In previous experiments, the data has been centered wherever possible. Centering however is a computationally expensive procedure, requiring $O(\ell^2)$ operations to center a kernel matrix, and here we propose a simple strategy which centers a subset of the kernel matrix columns in linear complexity.

We start with a proposition from Shawe-Taylor and Cristianini (2004) which finds the center of mass of a set of points.

Proposition 6.1 (Shawe-Taylor and Cristianini (2004))**.** *The center of mass Φ_S of a set of points solves the following optimisation problem*

$$\min \frac{1}{\ell}\sum_{i=1}^{\ell} \|\phi(\boldsymbol{x}_i) - \boldsymbol{\mu}\|^2,$$

Algorithm 19 Pseudo code for p-primal-dual Canonical Correlation Analysis.

```
Inputs: Matrices $\mathbf{X} \in \mathbb{R}^{\ell \times m}$, $\mathbf{K}^y \in \mathbb{R}^{\ell \times \ell}$, dimension $k$, sparsity $p$, sample sizes $c$, $d$,
regularisation parameter $\tau$
Process:
1) Indices $I_x = \{\ \}$, and $I_y = \{\ \}$
2) For $j = 1, \ldots, p$
    (a) Randomly pick $I_x^d \in [m]^d$, $I_y^c \in [\ell]^c$ and let $\tilde{\mathbf{X}}_1 = \mathbf{X}[, I_x^d]$ , $\tilde{\mathbf{K}}_1^y = \mathbf{K}^y[, I_y^c]$
    (b) For $i = 1, \ldots, j-1$
        i) Let $\boldsymbol{\tau}_i^x = \mathbf{X}_i \mathbf{u}_i$ and $\boldsymbol{\tau}_i^y = \mathbf{K}_i^y \boldsymbol{\beta}_i$
        ii) Deflate: $\tilde{\mathbf{X}}_{i+1} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_i^x \boldsymbol{\tau}_i^{x\prime}}{\boldsymbol{\tau}_i^{x\prime} \boldsymbol{\tau}_i^x}\right) \tilde{\mathbf{X}}_i$ and $\tilde{\mathbf{K}}_{i+1}^y = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_i^y \boldsymbol{\tau}_i^{y\prime}}{\boldsymbol{\tau}_i^{y\prime} \boldsymbol{\tau}_i^y}\right) \tilde{\mathbf{K}}_i^y$
    (c) End
    (d) Solve for scalars $s$ and $t$
        $$\begin{array}{ll} \max & \mathbf{u}_j' \mathbf{X}_j' \mathbf{K}_j^y \boldsymbol{\beta}_j \\ \text{s.t.} & (1-\tau)\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j + \tau \mathbf{u}_j' \mathbf{u}_j = 1 \\ & (1-\tau)\boldsymbol{\beta}_j' \mathbf{K}_j^{y\prime} \mathbf{K}_j^y \boldsymbol{\beta}_j + \tau \boldsymbol{\beta}_j' \mathbf{K}^y \boldsymbol{\beta}_j = 1 \\ & \mathbf{u}_j \in s \cdot \mathbf{I}[, I_x^d], \boldsymbol{\beta}_j \in t \cdot \mathbf{I}[, I_y^c] \end{array}$$
    (e) Let chosen directions be $s\mathbf{e}_{i_q^x}$ and $t\mathbf{e}_{i_r^y}$. Update $I_x \leftarrow I_x \cup \{i_q^x\}$ and $I_y \leftarrow I_y \cup \{i_r^y\}$.
3) End
4) Solve Equation 6.6 using $I_x$ and $I_y$
5) Let $\mathbf{u}_j = \mathbf{I}[, I_x]\tilde{\mathbf{u}}_j$ and $\boldsymbol{\beta}_j = \mathbf{I}[, I_y]\tilde{\boldsymbol{\beta}}_j$, $j = 1, \ldots, k$
Output: Directions $\mathbf{u}_j$, $\boldsymbol{\beta}_j$ and projections $\mathbf{X}\mathbf{u}_j$, $\mathbf{K}^y \boldsymbol{\beta}_j$, $j = 1, \ldots, k$
```

with $\boldsymbol{\mu} = \Phi_S = \frac{1}{\ell}\sum_{i=1}^{\ell} \phi(\boldsymbol{x}_i)$.

The above optimisation can be written as

$$\min \frac{1}{\ell}\|\mathbf{X} - \mathbf{j}\boldsymbol{\mu}'\|_F^2, \tag{6.8}$$

and we let $\boldsymbol{\mu}$ be a linear combination of at most c examples, i.e. $\boldsymbol{\mu} = \mathbf{X}'\boldsymbol{\alpha}$ for some dual vector $\boldsymbol{\alpha}$ with $\text{card}(\boldsymbol{\alpha}) \leq c$. By substituting this value of $\boldsymbol{\mu}$, Equation 6.8 is equivalent to

$$\begin{array}{ll} \min & \boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha} - \frac{2}{\ell}\mathbf{j}'\mathbf{K}\boldsymbol{\alpha} \\ \text{s.t.} & \text{card}(\boldsymbol{\alpha}) \leq c. \end{array} \tag{6.9}$$

Rather than solve this optimisation exactly, we randomly select a set of indices $I = \{i_1, \ldots, i_c\} \in [\ell]^c$. By defining $\tilde{\boldsymbol{\alpha}} = \boldsymbol{\alpha}[I]$, Equation 6.9 becomes

$$\min \quad \tilde{\boldsymbol{\alpha}}'\mathbf{K}[I,I]\tilde{\boldsymbol{\alpha}} - \tfrac{2}{\ell}\mathbf{j}'\mathbf{K}[,I]\tilde{\boldsymbol{\alpha}}, \tag{6.10}$$

which has the solution $\tilde{\boldsymbol{\alpha}} = \frac{1}{\ell}\mathbf{K}[I,I]^{-1}\mathbf{K}[I,]\mathbf{j}$ provided $\mathbf{K}[I,I]$ is invertible. In the case that $c = \ell$, $\boldsymbol{\mu}$ is identical to the solution of Equation 6.8.

In the primal space, the examples are centered using $\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{j}\boldsymbol{\mu}' = \mathbf{X} - \mathbf{j}\boldsymbol{\alpha}'\mathbf{X}$ and it follows that the kernel equivalent is $\tilde{\mathbf{K}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}' = \mathbf{K} - \mathbf{j}\boldsymbol{\alpha}'\mathbf{K} - \mathbf{K}\boldsymbol{\alpha}\mathbf{j}' + \boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}\mathbf{j}\mathbf{j}'$. Hence one can center a vector of inner products between a test example and the training examples $\mathbf{k}$ using

$$\tilde{\mathbf{k}}' = \mathbf{k}' - \boldsymbol{\alpha}'\mathbf{K} - \mathbf{k}'\boldsymbol{\alpha}\mathbf{j}' + \boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}\mathbf{j}',$$

which requires at most c columns of the kernel matrix.

6.2 Connection with Kernel Alignment

A result is now presented connecting the (unregularised) p-KCCA and KCCA optimisations to kernel alignment. It gives insight into the difference in the correlation of the (p-)KCCA features between the training set and a test set.

First observe that the KCCA objective function is equivalent to the kernel alignment between kernel matrices of the projections, $\tilde{\mathbf{K}}^x = \mathbf{K}^x\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}^{x\prime}$ and $\tilde{\mathbf{K}}^y = \mathbf{K}^y\boldsymbol{\beta}\boldsymbol{\beta}'\mathbf{K}^{y\prime}$, since

$$\begin{aligned} A(\tilde{\mathbf{K}}^x, \tilde{\mathbf{K}}^y) &= \frac{\mathrm{tr}(\mathbf{K}^x\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}^{x\prime}\mathbf{K}^y\boldsymbol{\beta}\boldsymbol{\beta}'\mathbf{K}^{y\prime})}{\sqrt{\mathrm{tr}((\mathbf{K}^x\boldsymbol{\alpha}\boldsymbol{\alpha}'\mathbf{K}^{x\prime})^2)\mathrm{tr}((\mathbf{K}^y\boldsymbol{\beta}\boldsymbol{\beta}'\mathbf{K}^{y\prime})^2)}} \\ &= \frac{(\boldsymbol{\alpha}'\mathbf{K}^{x\prime}\mathbf{K}^y\boldsymbol{\beta})^2}{\boldsymbol{\alpha}'\mathbf{K}^{x\prime}\mathbf{K}^x\boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{K}^{y\prime}\mathbf{K}^y\boldsymbol{\beta}}. \end{aligned}$$

One can extend this observation by considering the complete set of features extracted by KCCA. In this case, let $\tilde{\mathbf{K}}^x = \sum_{i=1}^k \mathbf{K}^x\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i'\mathbf{K}^{x\prime}$ and $\tilde{\mathbf{K}}^y = \sum_{i=1}^k \mathbf{K}^y\boldsymbol{\beta}_i\boldsymbol{\beta}_i'\mathbf{K}^{y\prime}$, then the alignment between $\mathbf{K}^x$ and $\mathbf{K}^y$ is

$$\begin{aligned} A(\tilde{\mathbf{K}}^x, \tilde{\mathbf{K}}^y) &= \frac{\sum_{i=1}^k (\boldsymbol{\alpha}_i' \mathbf{K}^{x\prime} \mathbf{K}^y \boldsymbol{\beta}_i)^2}{\sqrt{\sum_{i=1}^k (\boldsymbol{\alpha}_i' \mathbf{K}^{x\prime} \mathbf{K}^x \boldsymbol{\alpha}_i)^2 \sum_{i=1}^k (\boldsymbol{\beta}_i' \mathbf{K}^{y\prime} \mathbf{K}^y \boldsymbol{\beta}_i)^2}} \\ &= \frac{1}{k} \sum_{i=1}^k (\boldsymbol{\alpha}_i' \mathbf{K}^{x\prime} \mathbf{K}^y \boldsymbol{\beta}_i)^2, \end{aligned}$$

where the first line follows from the conjugacy of the eigenvectors for the KCCA eigenproblem, and the second uses the fact that $\mathbf{K}^x \boldsymbol{\alpha}_i$ and $\mathbf{K}^y \boldsymbol{\beta}_i$ are unit norm. Therefore it has been shown that the kernel alignment between the KCCA output kernel matrices is equivalent to the average squared correlation of the resulting features.

A theorem is now derived, similar to that given in Cristianini et al. (2001), which shows that the alignment between two kernel matrices is concentrated. In a slight modification of notation, we denote the alignment between two kernel matrices evaluated from a data sample S as $A(S)$. First, we introduce a theorem from McDiarmid which is informative about the deviation of a function from its expectation.

Theorem 6.2. *(McDiarmid (1989)) Let $X_1, ..., X_n$ be independent random variables taking values in a set A, and assume that $f : A^n \rightarrow \mathbb{R}$ satisfies*

$$\sup_{x_1,...,x_n,\hat{x}_i \in A} |f(x_1, ..., x_n) - f(x_1, ..., \hat{x}_i, x_{i+1}, ..., x_n)| \leq c_i, \quad 1 \leq i \leq n$$

then for all $\epsilon > 0$,

$$P\{|f(X_1, ..., X_n) - \mathbb{E}[f(X_1, ..., X_n)]| \geq \epsilon\} \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right).$$

This leads onto the following theorem.

Theorem 6.3. *The sample based estimate of the alignment between two kernel matrices is concentrated around its expected value. For kernels with feature vectors of norm at most 1, we have*

$$P\{S : |A(S) - \mathbb{E}[A(S)]| \geq \hat{\epsilon}\} \leq \delta,$$

where $\hat{\epsilon} = \frac{\epsilon_1}{\sqrt{A_2(S)}} + \frac{(|A_1(S)|+\epsilon_1)\epsilon_2}{\sqrt{A_2(S)(A_2(S)-\epsilon_2)} \cdot (\sqrt{A_2(S)}+\sqrt{(A_2(S)-\epsilon_2)})}$, $\epsilon_1 = \sqrt{8\ell^3 \log(4/\delta)}$, $\epsilon_2 = \sqrt{8\ell^5 \log(4/\delta)}$, $A_1(S) = \langle \boldsymbol{K}^x, \boldsymbol{K}^y \rangle_F$, and $A_2(S) = \langle \boldsymbol{K}^x, \boldsymbol{K}^x \rangle_F \langle \boldsymbol{K}^y, \boldsymbol{K}^y \rangle_F$.

Proof. First observe that $A(S) = A_1(S)/\sqrt{A_2(S)}$. We derive the concentration of A_1 and A_2 by considering an alternative training set $S' = S\backslash\{\mathbf{x}_k\} \cup \{\hat{\mathbf{x}}_k\}$ and bounding the following

$$\begin{aligned} |A_1(S) - A_1(S')| &\leq 4\ell \\ |A_2(S) - A_2(S')| &\leq (2\ell - 1)^2 \leq 4\ell^2. \end{aligned}$$

Applications of McDiarmid's Theorem result in

$$\begin{aligned} P\{|A_1(S) - \mathbb{E}[A_1(S)]| \leq \epsilon_1\} &\leq 2\exp\left(\frac{-\epsilon_1^2}{8\ell^3}\right) \\ P\{|A_2(S) - \mathbb{E}[A_2(S)]| \leq \epsilon_2\} &\leq 2\exp\left(\frac{-\epsilon_2^2}{8\ell^5}\right). \end{aligned}$$

By setting the right hand side of the above to $\delta/2$, we have $\epsilon_1 = \sqrt{8\ell^3 \log(4/\delta)}$ and $\epsilon_2 = \sqrt{8\ell^5 \log(4/\delta)}$. It follows that with probability at least $1-\delta$, $|A_1(S) - \mathbb{E}[A_1(S)]| < \epsilon_1$ and $|A_2(S) - \mathbb{E}[A_2(S)]| < \epsilon_2$. Hence,

$$\begin{aligned} |A(S) - \mathbb{E}[A(S)]| &= \left| \frac{A_1(S)}{\sqrt{A_2(S)}} - \frac{\mathbb{E}[A_1(S)]}{\sqrt{\mathbb{E}[A_2(S)]}} \right| \\ &= \left| \frac{A_1(S) - \mathbb{E}[A_1(S)]}{\sqrt{A_2(S)}} + \frac{\mathbb{E}[A_1(S)]}{\sqrt{A_2(S)}} - \frac{\mathbb{E}[A_1(S)]}{\sqrt{\mathbb{E}[A_2(S)]}} \right| \\ &\leq \frac{\epsilon_1}{\sqrt{A_2(S)}} + \left| \frac{\mathbb{E}[A_1(S)]}{\sqrt{A_2(S)}} - \frac{\mathbb{E}[A_1(S)]}{\sqrt{\mathbb{E}[A_2(S)]}} \right| \\ &= \frac{\epsilon_1}{\sqrt{A_2(S)}} + \left| \frac{\mathbb{E}[A_1(S)](\sqrt{\mathbb{E}[A_2(S)]} - \sqrt{A_2(S)})}{\sqrt{A_2(S)\mathbb{E}[A_2(S)]}} \right| \\ &\leq \frac{\epsilon_1}{\sqrt{A_2(S)}} + \frac{|\mathbb{E}[A_1(S)]| \cdot |A_2(S) - \mathbb{E}[A_2(S)]|}{\sqrt{A_2(S)\mathbb{E}[A_2(S)]} \cdot (\sqrt{A_2(S)} + \sqrt{\mathbb{E}[A_2(S)]})} \\ &\leq \frac{\epsilon_1}{\sqrt{A_2(S)}} + \frac{(|A_1(S)| + \epsilon_1)\epsilon_2}{\sqrt{A_2(S)\mathbb{E}[A_2(S)]} \cdot (\sqrt{A_2(S)} + \sqrt{\mathbb{E}[A_2(S)]})} \\ &\leq \frac{\epsilon_1}{\sqrt{A_2(S)}} + \frac{(|A_1(S)| + \epsilon_1)\epsilon_2}{\sqrt{A_2(S)(A_2(S) - \epsilon_2)} \cdot (\sqrt{A_2(S)} + \sqrt{(A_2(S) - \epsilon_2)})} \\ &= \hat{\epsilon}. \end{aligned}$$

□

The value of $\hat{\epsilon}$ tends to be small provided $\epsilon_1/\sqrt{A_2(S)}$ is also small. Since $\sqrt{A_2(S)}$ is the product of the norm of the kernel matrices, this implies that examples are not too dissimilar to each other in terms of the magnitude of their inner products in the kernel feature spaces. In this case, it is intuitive that the alignment measured on a training set is similar on a test set. Hence, for the KCCA case it would be better to project the examples into a subspace of small dimensionality to increase the likelihood that $A_2(S)$ is large. As a final remark, note that the concentration result in Cristianini et al. (2001) considers the kernel target alignment (where $\mathbf{K}^y = \mathbf{y}\mathbf{y}'$) and it follows that $\|\mathbf{K}^y\|_F = \ell$ and hence A_2 is likely to be large provided $\|\mathbf{K}^x\|_F$ is also large.

6.3 Computational Results

Using a set of artificial and real datasets, we highlight important properties of the sparse CCA methods and also compare their performance to that of KCCA.

6.3.1 Greedy versus Exhaustive Search

An important question about the sparse CCA variants presented above is how the approximations to the sparse optimisations compare with their exact solutions. Since the exact solutions require combinatorial procedures, one can only evaluate them in a practical time frame for small ℓ or p, and here we make such a comparison using a set of small artificial datasets.

We create 5 synthetic datasets consisting of 20 examples, and 10 features in both $\mathcal{X}$ and $\mathcal{Y}$ views. The examples are generated using $\mathbf{x} = \mathbf{n}(0, 1)$ and $\mathbf{y} = \mathbf{n}(0, 1)$ where $\mathbf{n}(\mu, \sigma^2)$ is a vector of normal random variables with mean μ and variance σ^2. Both exact and approximate solutions to Equation 6.1 are considered for $p = 3$, $\tau = 0$. The approximate solution corresponds to that obtained using Algorithm 18, with $c = 20$. An identical test is used to compare solutions to Equation 6.5, with $c = 20$ and $d = 10$ for the approximate case. The tests are repeated using all 5 datasets and the average eigenvalues across the datasets are recorded.

Method	λ_1	λ_2	λ_3	$\sum_{i=1}^{3} \lambda_i$
Optimisation 6.1	0.9421	0.3856	0.1347	1.4624
Optimisation 6.1 approx	0.7801	0.6394	0.5112	1.9306
Optimisation 6.5	0.9163	0.4228	0.1120	1.4511
Optimisation 6.5 approx	0.8048	0.6365	0.4460	1.8873

TABLE 6.1: Comparison of the mean eigenvalues obtained using approximate and exact computations of Equations 6.1 and 6.5.

Table 6.1 highlights important differences between the exact solutions to Equations 6.1 and 6.5 and their approximations. One would expect that the approximate solutions have a lower first eigenvalue, which is verified empirically in this case. More important however, is that the successive eigenvalues are significantly higher for the approximations than those obtained in the exact cases. The deflations used to formulate the approximations ensure that the chosen columns of the data and kernel matrices are orthogonal and correlated between the two views, and hence have a high cumulative correlation. In contrast, when solving Equation 6.1 for example, the chosen kernel matrix columns are selected solely to maximise the initial correlation and will not necessarily result in a high cumulative correlation.

6.3.2 Cumulative Correlation

An important consideration for any CCA variant is how much correlation is captured when projecting examples from a test set, and here the test correlations of the CCA methods are compared on a pair of artificial datasets.

6.3.2.1 Artificial Dataset 1

We start with a dataset composed of 300 examples of a hidden variable $\mathbf{z}$ which has 1000 features of which 100 on average are non-zero. The $\mathbf{x}$ and $\mathbf{y}$ examples are computed using $\mathbf{x} = \mathbf{z} + \mathbf{n}(0, 0.01, 100)$, $\mathbf{y} = \mathbf{z} + \mathbf{n}(0, 0.01, 100)$ with probability 2/3 and $\mathbf{x} = \mathbf{z} + \mathbf{n}(0, 1, 100)$, $\mathbf{y} = \mathbf{z} + \mathbf{n}(0, 1, 100)$ with probability 1/3, where $\mathbf{n}(\mu, \sigma^2, s)$ is a sparse vector of random normal variables with s non-zero entries, mean μ and variance σ^2.

Using a 10-fold cross validation procedure, the CCA methods are evaluated by allowing each one to generate a maximum of 150 features. We use KCCA with an Incomplete Cholesky decomposition and p-KCCA. Since all of the features in the examples are relevant, p-PDCCA is not included in this experiment. For KCCA, the regularisation parameter is selected from $\{0, 0.2, \dots, 1\}$ and the precision parameter for the Incomplete Cholesky decomposition, η, is 0.01. With p-KCCA, $c = 200$, $\tau = 0$ and p is varied by selecting it from the set $\{60, 90 \dots, 240\}$.

The test cumulative correlations are shown in Table 6.2. With KCCA it is clear that regularisation provides little advantage as the highest correlations occur when $\tau = 0$. The p-KCCA method improves over KCCA in every configuration since it tends to choose indices which correspond to examples with a low noise component. Table 6.3 shows that the average number of low noise examples chosen by p-KCCA is greater than the expectation in general. The test correlations peak at approximately $p = 180$, which corresponds closely to the point at which the proportion of lower noise examples starts to decrease as p increases. After $p = 180$ the choice of projection directions is no longer limited to that of a high proportion of low noise examples and one would expect the correlation to eventually converge to that of KCCA.

	Features				
Method	30	60	90	120	150
KCCA $\tau = 0$	17.3 (1.2)	35.1 (2.5)	53.0 (4.2)	71.3 (5.5)	89.6 (6.5)
KCCA $\tau = 0.2$	15.6 (1.4)	31.7 (2.4)	47.7 (3.5)	64.6 (4.6)	82.4 (5.8)
KCCA $\tau = 0.4$	15.6 (1.4)	31.7 (2.4)	47.8 (3.5)	64.7 (4.5)	82.6 (5.7)
KCCA $\tau = 0.6$	15.6 (1.4)	31.7 (2.3)	47.7 (3.5)	64.7 (4.4)	82.8 (5.6)
KCCA $\tau = 0.8$	15.5 (1.4)	31.5 (2.3)	47.5 (3.4)	64.6 (4.3)	82.8 (5.5)
KCCA $\tau = 1$	15.2 (1.3)	30.7 (2.2)	46.5 (3.2)	63.6 (4.0)	82.0 (5.2)
p-KCCA $p = 60$	21.7 (1.6)	43.4 (3.3)	-	-	-
p-KCCA $p = 90$	**22.1** (1.3)	43.9 (3.0)	65.2 (4.6)	-	-
p-KCCA $p = 120$	**22.1** (1.4)	**44.1** (2.7)	**65.7** (4.2)	86.5 (5.8)	-
p-KCCA $p = 150$	**22.1** (1.5)	**44.1** (2.9)	**65.7** (4.3)	**87.3** (5.8)	108.2 (7.4)
p-KCCA $p = 180$	21.8 (1.6)	43.6 (3.3)	65.6 (4.6)	87.2 (6.0)	**108.4** (7.7)
p-KCCA $p = 210$	20.6 (1.5)	41.3 (3.3)	61.6 (4.8)	82.3 (6.0)	102.8 (7.1)
p-KCCA $p = 240$	19.2 (1.8)	38.1 (2.9)	57.1 (4.0)	76.7 (5.3)	95.9 (6.7)

Table 6.2: Cumulative correlations obtained using KCCA and p-KCCA on an artificial dataset.

p	Relevant examples
30	0.817 (0.123)
60	0.822 (0.115)
90	0.820 (0.127)
120	0.808 (0.128)
150	0.816 (0.119)
180	0.819 (0.117)
210	0.751 (0.064)
240	0.703 (0.029)

Table 6.3: The mean proportion of relevant examples selected using by p-KCCA for different values of p.

6.3.2.2 Artificial Dataset 2

The second artificial dataset has 400 examples, with hidden variable $\mathbf{z}$ being a sparse vector consisting of 400 features of which 40 are non-zero on average. In this case the particular features are important, hence we have $\mathbf{x} = \mathbf{C}_x'\mathbf{z} + \mathbf{n}(0, 0.01, 50)$ where the coefficient matrix $\mathbf{C}_x \in \mathbb{R}^{400\times 500}$ has only 100 non-zero columns. The remaining examples are computed using $\mathbf{y} = \mathbf{z} + \mathbf{n}(0, 0.01, 50)$ with probability 2/3 and $\mathbf{y} = \mathbf{z} + \mathbf{n}(0, 0.1, 50)$ with probability 1/3. It follows that only 100 out of 500 features in the $\mathbf{x}$ examples are meaningfully correlated with those present in the $\mathbf{y}$ examples.

We again compare the cumulative correlations of the CCA methods using 5-fold cross validation, and each method is iterated at most 120 times. For KCCA, the regularisation parameter selected from $\{0, 0.2, \ldots, 1\}$ and $\eta = 0.01$. With p-PDCCA and p-KCCA we set $c = d = 200$ and vary the values of p by selecting it from $\{144, 180, \ldots, 360\}$. Furthermore, the sparse CCA methods are regularised be setting $\tau = 0.1$.

Table 6.4 shows that regularisation improves the performance of KCCA, and its best correlation occurs when $\tau = 0.4$ and $k = 120$. Regularisation is required for this dataset

	Features					
Method	20	40	60	80	100	120
KCCA $\tau = 0$	2.0 (0.3)	4.1 (0.4)	6.2 (0.4)	8.3 (0.3)	10.1 (0.4)	12.2 (1.0)
KCCA $\tau = 0.2$	8.8 (0.4)	15.9 (0.6)	21.8 (1.0)	26.4 (0.6)	30.6 (0.6)	34.4 (0.8)
KCCA $\tau = 0.4$	8.6 (0.4)	15.7 (0.6)	21.6 (1.0)	26.4 (0.7)	30.6 (0.7)	34.6 (0.7)
KCCA $\tau = 0.6$	8.4 (0.4)	15.3 (0.6)	21.2 (0.9)	26.1 (0.7)	30.4 (0.7)	34.4 (0.8)
KCCA $\tau = 0.8$	8.0 (0.4)	14.7 (0.7)	20.5 (0.9)	25.6 (0.6)	30.0 (0.7)	33.8 (0.9)
KCCA $\tau = 1$	7.2 (0.5)	13.6 (0.7)	19.1 (0.8)	24.2 (0.6)	28.7 (0.8)	32.2 (0.7)
p-KCCA $p = 144$	5.9 (0.3)	10.4 (0.6)	14.2 (1.0)	17.7 (0.9)	20.7 (0.8)	23.1 (0.7)
p-KCCA $p = 180$	6.6 (0.6)	11.5 (0.9)	15.8 (0.6)	19.1 (0.9)	22.3 (0.9)	25.2 (0.7)
p-KCCA $p = 216$	7.5 (0.7)	13.1 (0.8)	17.5 (0.5)	21.0 (0.4)	24.6 (0.7)	27.7 (0.6)
p-KCCA $p = 252$	7.9 (0.3)	14.0 (0.4)	19.3 (0.5)	23.3 (0.3)	26.7 (0.5)	29.7 (0.6)
p-KCCA $p = 288$	8.3 (0.4)	15.0 (0.5)	20.6 (0.3)	24.9 (0.3)	28.7 (0.6)	32.1 (0.6)
p-KCCA $p = 324$	8.6 (0.3)	15.6 (0.5)	21.3 (0.8)	25.7 (0.5)	29.7 (0.9)	33.4 (0.7)
p-KCCA $p = 360$	8.9 (0.4)	16.1 (0.6)	21.9 (0.9)	26.5 (0.6)	30.8 (0.7)	34.2 (0.9)
p-PDCCA $p = 144$	**13.4** (0.4)	**23.6** (1.0)	**30.9** (1.3)	35.8 (1.9)	38.9 (2.0)	41.1 (2.2)
p-PDCCA $p = 180$	12.9 (0.5)	23.1 (1.0)	30.8 (1.0)	**36.5** (1.3)	40.5 (1.4)	43.2 (1.6)
p-PDCCA $p = 216$	12.6 (0.5)	22.8 (0.7)	30.4 (0.9)	36.4 (1.4)	**40.6** (1.6)	**44.2** (1.5)
p-PDCCA $p = 252$	12.3 (0.3)	22.1 (1.0)	29.8 (1.2)	35.8 (1.5)	40.5 (2.0)	**44.2** (2.1)
p-PDCCA $p = 288$	12.0 (0.1)	21.6 (0.7)	28.7 (0.9)	34.4 (1.0)	39.0 (1.7)	43.1 (1.9)
p-PDCCA $p = 324$	11.8 (0.4)	20.8 (0.6)	27.5 (1.1)	33.4 (1.3)	37.7 (1.3)	41.7 (1.5)
p-PDCCA $p = 360$	11.4 (0.3)	20.0 (0.5)	26.7 (0.9)	32.0 (1.0)	36.5 (1.1)	40.2 (1.2)

TABLE 6.4: Cumulative correlations obtained using KCCA, p-KCCA and p-PDCCA on an artificial dataset.

p	Features	Examples
36	32.0 (0.0)	24.4 (3.5)
72	63.0 (0.7)	51.6 (5.5)
108	80.4 (2.4)	75.6 (5.0)
144	87.6 (3.4)	94.6 (4.3)
180	89.8 (3.1)	115.0 (1.4)
216	90.6 (2.6)	138.6 (3.0)
252	90.8 (2.5)	163.2 (3.7)
288	91.6 (2.2)	186.6 (4.0)
324	92.0 (1.7)	212.2 (2.2)
360	93.0 (1.7)	239.0 (0.0)

TABLE 6.5: The relevant examples and features selected using p-PDCCA.

since the smallest eigenvalues of $\mathbf{K}^x$ and $\mathbf{K}^y$ are close to zero. As discussed in Chapter 2, this implies that unregularised KCCA could potentially choose projections directions with large norm which would not generalise well. Observe that sparsity in p-KCCA provides little advantage, as the best correlation occurs when $p = 360$. In contrast, with p-PDCCA sparse projection directions allow it to improve over the corresponding correlations of KCCA. Table 6.5 demonstrates that p-PDCCA is effective at choosing the useful features in the $\mathcal{X}$ view, however, the proportion of useful features selected decreases with an increase in p. A higher value of p is required to capture the majority of the lower noise examples, which suggests that one can sometimes improve performance using different values for the cardinality of $\mathbf{u}$ and $\boldsymbol{\beta}$, although it comes at the cost of extra time needed for model selection.

6.3.3 UCI Mate Retrieval Experiment

The CCA methods are now applied to a series of UCI datasets, details of which are given in Table 6.6. They are all multiclass datasets and indicator vectors using the labels form the $\mathbf{Y}$ matrix. The features in $\mathbf{X}$ are centered and normalised to have unit norm, and KCCA, p-KCCA and p-PDCCA are evaluated using 3-fold cross validation. This time we are interested in the application of the CCA methods to a mate retrieval task. For the ith projected example $\mathbf{U}'\mathbf{x}_i$ we retrieve the 5 closest opposite examples, and record whether one of these is the "mate" of the original example, $\mathbf{V}'\mathbf{y}_i$, in its projected space. This process is averaged over all examples to obtain a retrieval rate.

Dataset	Examples	Features	Classes
Arrhythmia	452	279	13
Glass	214	10	7
LRS	531	93	48
Multiple Features	2000	649	10

Table 6.6: Information about the UCI datasets.

Model selection is performed using 2 repetitions of 4-fold cross validation, with parameters selected as follows. For KCCA, the regularisation parameter is chosen from $\{0, 0.2, \ldots, 1\}$ and $\eta \in \{0.05, 0.1, 0.2\}$. The setup of the sparse CCA methods is modified to introduce a sparsity parameter $\nu = p/\ell$ (the resulting methods are referred to as ν-PDCCA and ν-KCCA), which is selected from $\{0.2, 0.4, 0.6\}$. For these methods $c = d = 200$, the regularisation parameter $\tau \in \{0, 0.2, 0.4, 0.6\}$, and ν-KCCA uses the sparse centering procedure introduced in Section 6.1.2. The value of k is selected from 10 values in equal intervals from 1 to $\min(\text{rank}(\mathbf{X}), \text{rank}(\mathbf{Y}))$. We also apply the RBF kernel with $\sigma \in \{2^{-4}, 2^{-2} \ldots, 2^4\}$ on the $\mathbf{x}$ examples for KCCA and ν-KCCA, and use 2 repetitions of 3-fold cross validation for model selection in these cases.

	KCCA			ν-KCCA			ν-PDCCA		
Dataset	Rate	k	γ	Rate	k	ν	Rate	k	ν
Arrhythmia	**.629** (.023)	8	.70	.613 (.024)	9	.60	.611 (.028)	6.7	.20
Glass	.836 (.035)	1	.06	.854 (.089)	1	.20	**.892** (.022)	1	.33
LRS	.537 (.015)	4	.16	.531 (.034)	3.3	.27	**.548** (.065)	2.7	.33
Multiple Feat.	.984 (.005)	7	.46	.980 (.006)	7.7	.33	**.988** (.002)	8	.40
Arrhythmia	**.584** (.064)	10.7	.78	.536 (.057)	8.3	.53			
Glass	**.859** (.037)	2.3	.53	.845 (.061)	1.3	.27			
LRS	.672 (.020)	14	.94	**.706** (.043)	12.3	.60			
Multiple Feat.	.983 (.004)	7	.99	**.988** (.006)	8	.47			

Table 6.7: Average mate retrieval rates of the CCA methods for linear (top) and RBF (bottom) kernels.

The results are shown in Table 6.7. For KCCA, γ is the maximum rank of the Incomplete Cholesky decompositions of $\mathbf{K}^x$ and $\mathbf{K}^y$ divided by the number of training examples, hence the sparsity of the solution. The linear results indicate the advantage of using ν-PDCCA which results in the highest retrieval rates for most datasets, implying a redundancy of some of the features. A comparison of the sparsity of the CCA methods reveals that KCCA is broadly competitive with ν-KCCA and ν-PDCCA when using the linear kernel. However, the rank of the linear kernel matrix is often small and one would expect a high degree of sparsity. The RBF results are more insightful in terms of the sparsity of the solutions, and here ν-KCCA improves over KCCA in every case whilst remaining competitive with its mate retrieval rates.

The overall picture is that ν-KCCA and ν-PDCCA are competitive with KCCA, and in some cases can provide a significant improvement over its performance. In particular, ν-KCCA obtains sparse projection directions since it chooses examples that are targeted towards finding a high cumulative correlation. An improvement in sparsity implies fewer computational operations are needed for training and projecting new test data.

6.4 Case Study: Enzyme Function Prediction

Proteins are large organic molecules essential to all organisms, with a unique 3-dimensional structure which they naturally fold into. They exist in every living cell, applying themselves to a variety of functions, and in the human body it is estimated that there may be as many as a million different proteins. Hence, their analysis is important in understanding organisms, and in particular to the study of human biology. Finding the function of a protein with known sequence and structure using experimental evidence remains a difficult, time and cost intensive task. For this reason, a computational approach is desirable. This is not a simple task however, since proteins have complex structures and can serve a large variety of functions.

This study attempts to learn the function of proteins, focusing on their catalytic properties, i.e. their action as *enzymes*. Enzymes are often specific to one or a few different reactions, for example, DNA repair, DNA replication and those involved in *metabolism*[1]. Most enzymes are much larger than the *substrates* they act on, and only a small part of the enzyme[2] (around 3 or 4 *amino acids*[3]) is directly involved in catalysis. It follows that feature selection is important for this investigation since proteins are typically represented using a large number of features of which only a small fraction are useful for catalysis.

6.4.1 Background and Related Work

One of the difficulties in predicting reactions from enzymes is the choice of data representation. An unfolded protein is essentially a linear sequence of amino acids, often

[1]Metabolism is the breakdown of food into energy or the use of energy to construct components of cells.
[2]Known as the *active site*.
[3]An amino acid is a molecule containing nitrogen and a carboxyl group (CO_2H).

called the *primary structure*. The corresponding secondary structure is characterised by regularly repeating local structures stabilized by hydrogen bonds. For example, the structure of a few residues[4] can be described in terms of different length helices. Using protein structure alone for function classification can be difficult since proteins with a similar function may have dissimilar structure, and proteins with a similar structure may have distinct functions. In fact, a single amino acid mutation can alter the function of a protein and make a pair of structurally closely related proteins functionally different.

The standard method for describing enzymatic reactions is using an Enzyme Commission (EC) number, which is a sequence of 4 numbers specifying a particular type of reaction. The sequence of numbers represent progressively more specific classifications of the reaction, of which the top level is shown in Table 6.8. The complete EC hierarchy consists of 4 levels with 1633 nodes.

EC No.	Name	Description	Frequency
1	Oxidoreductases	To catalyse oxidation/reduction reactions; transfer of H and O atoms or electrons from one substance to another.	1088
2	Transferases	Transfer of a functional group from one substance to another. The group may be methyl-, acyl-, amino- or phosphate group.	2317
3	Hydrolases	Formation of two products from a substrate by hydrolysis.	1480
4	Lyases	Non-hydrolytic addition or removal of groups from substrates. C-C, C-N, C-O or C-S bonds may be cleaved.	589
5	Isomerases	Intramolecule rearrangement, i.e. isomerization changes within a single molecule.	384
6	Ligases	Join together two molecules by synthesis of new C-O, C-S, C-N or C-C bonds with simultaneous breakdown of ATP.	760

Table 6.8: Description and frequency (in our training set) of top level EC numbers (Sourced from Wikipedia (2007)).

Recent work in function prediction has made use of a variety of enzyme feature representations, for example in Cai et al. (2004) enzymes are encoded using characteristics such as hydrophobicity, polarisability, charge and frequency of amino acid bases. The resulting features are then used to classify, using an SVM, the enzymes into 46 different EC Number categories belonging to the second level of the hierarchy. In Borgwardt et al. (2005), the primary and secondary structures of proteins are encoded using graph kernels, which are shown to perform well when predicting the top level of the EC hierarchy. Lanckriet et al. (2004b) uses a combination of kernel representations in an optimal fashion by formulating a convex optimisation problem. They are applied to predicting functional classification associated with Yeast proteins.

[4]A residue is a portion of a larger molecule.

Several authors have relied solely on amino acid sequences to encode proteins. In Leslie et al. (2002) an efficient spectrum kernel is formulated and used for protein classification. The features used by the spectrum kernel are the set of all possible subsequences of amino acids of a fixed length n, which is computable in linear time in the lengths of the input sequences. The features are used with an SVM on the Structural Classification of Proteins (SCOP) database, and shown to be comparable to state-of-the-art using $n = 3$. A more general approach is given in Ben-Hur and Brutlag (2005), which searches for *sequence motifs*. These are elements that are conserved across different proteins corresponding to functional regions of a protein. The motifs are based on a regular expression applied to the amino acid sequences. The resulting features are shown to be good predictors of the EC classification when used in conjunction with an SVM.

Our work differs from these publications, as it attempts to pinpoint which reactions are catalysed by a given enzyme by modelling reactions directly as graphs. Furthermore, we perform EC classification of the enzymes using all 4 levels of the EC hierarchy. Related work from the same project is presented in Astikainen et al. (2007), which discovers microlabel predictions for the EC hierarchy using Maximum Margin Regression (MMR, Szedmak et al. (2005)).

6.4.2 Data Description and Feature Representation

Our dataset is composed of 28,765 protein sequences from the KEGG LIGAND database, of which 9889 are enzymes. Of the enzymes, 9455 have a complete EC-classification and 434 have a partial classification. For the purposes of evaluation, the enzymes are subsampled into a training set of size 6618 examples. Since proteins are made of sequences of amino acids, these form the basis of their feature representation. There are twenty-two naturally occurring amino acids, however, only twenty are required for the enzymes in our dataset. Figure 6.1 shows the distribution of protein lengths in the training set.

6.4.2.1 Sequence Representations

The feature representations used for the enzymes fall into three categories: substring features (Lodhi et al. (2002)), gap or mismatch features (Leslie et al. (2004)) and Global Trace Graph (GTG) features.

The substring features make a count of each substring of length p in the enzyme sequences. Given an alphabet of size q, the number of features is q^p, with $q = 21$ in the case of the enzymes sequence features (20 amino acids plus a missing value). One would expect the computation of this set of features to become prohibitive with large p. Hence, Lodhi et al. (2002) supplies a technique for computing the inner product between features at a reduced computational cost, using a dynamic programming technique. However, since p is small in our case and there is an interest in the underlying

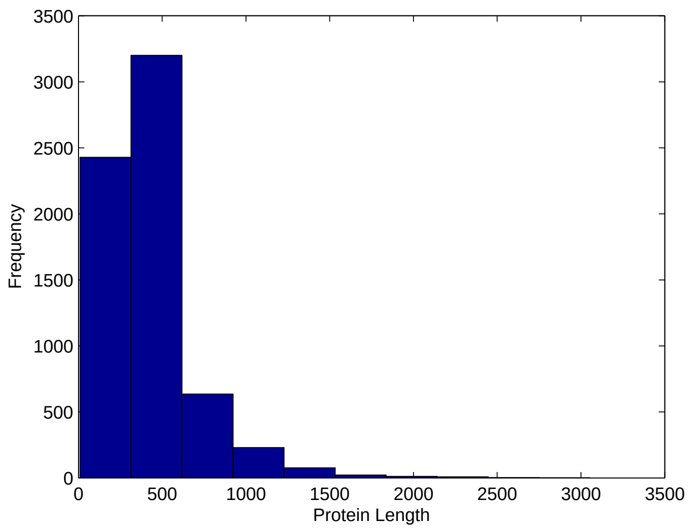


FIGURE 6.1: Distribution of the number of amino acids in each protein sequence.

features, the substring features are computed explicitly. A small variation of the standard substring features is to allow gaps or mismatches in the substrings. The set of sequence substring features of length p is referred to as Sp, and the set of sequence features of length p with at most r gaps of length t is denoted by Sp-rGt.

An alternative protein encoding is the Alignment Trace Graph (Heger et al. (2003)) technique which finds residues that are potentially well conserved and thus may be part of the active center. The GTG kernel obtained from this method has a feature for each residual of a particular type in a particular cluster. Since a cluster could potentially be within the active center, these features may be useful for predicting the reaction of an enzyme. Table 6.9 shows some properties of the different enzyme feature representations.

Encoding	Number of features	Non-zero features
S3	9261	8338
S4	194481	155902
S4-1G1	194481	160877
S4-1G2	194481	160846
S4-2G1	194481	160851
S5	4084101	1452776
S5-1G1	4084101	2431977
S6	85766121	2879536
GTG	1659550	590411

TABLE 6.9: Number of features for each protein sequence encoding.

6.4.2.2 Reaction Representations

As mentioned earlier, one method of representing the catalysed reactions is by using EC numbers. In the following experiments, these are encoded using indicator vectors with a binary label corresponding to each node of the EC hierarchy. Hence, for a complete EC number, only 4 labels are non-zero.

An alternative and more precise method of representing reactions is to model the reaction substrates and products directly using graph kernel techniques (Takimoto and Warmuth (2002)). There are two kernels that are computed in this way. The first is computed using

$$\kappa_r(\mathbf{x}, \mathbf{z}) = m(\mathbf{x})'\mathbf{K}_m m(\mathbf{z}),$$

where $m(\mathbf{x})$ is a vector indicating which molecules are present in reaction $\mathbf{x}$, and $\mathbf{K}_m \in \mathbb{R}^{1767\times1767}$ is kernel matrix of molecules whose (i, j)th entry is the number of common subgraphs of size less than 10 in molecules i and j. Hence, κ_r can be considered as the similarity between two reactions based on common molecules. The second reaction kernel is computed as

$$\kappa_s(\mathbf{x}, \mathbf{z}) = n(\mathbf{x})'\mathbf{K}_m n(\mathbf{z}),$$

where $n(\mathbf{x})$ is a vector indicating the difference between product and substrate molecule numbers for reaction $\mathbf{x}$.

6.4.3 Learning the Semantics of Enzymes and their Reactions

We now apply the CCA variants to the enzyme data and reaction kernels, recording the average number of correctly retrieved reactions. To retrieve a reaction for a particular enzyme we project the enzymes and reactions into the corresponding CCA-learned subspaces, and then record whether the matching reaction is within the top 10 closest reactions. The average retrieval accuracy is measured using 3-fold cross validation, and an inner 3-fold cross validation procedure is used to select parameters using 900 examples.

For these tests we use 2000 examples and some of the properties of the corresponding reaction kernels are shown in Table 6.10. All of the sequence features in Table 6.9 are used in conjunction with the two reaction kernels described above, giving 18 combinations of sequence-reaction features. The sequence data is preprocessed by removing the zero norm features, and the resulting examples are then normalised to have unit norm.

The reaction kernels are centered and normalised so that the examples have zero mean and unit norm.

Property	κ_r	κ_s
Rank	580	562
Unique columns	603	593
Column norm mode	0	0
Column norm mode frequency	209	221

TABLE 6.10: Some properties of the reaction kernel matrices for 2000 examples.

As before, we use a grid search to select model parameters. The KCCA, ν-KCCA, ν-PDCCA methods are applied to the enzyme data using values of $\{50, 100, \ldots, 500\}$ features. For KCCA, the regularisation parameter is selected from $\{0, 0.2, \ldots, 1\}$ and $\eta = 0.01$. The c parameter of ν-KCCA and ν-PDCCA is set to 500, and $\nu \in \{0.25, 0.5, \ldots, 1\}$. Since many of the sequence encodings have a large number of features, we fix $d = 1000$ for ν-PDCCA. The ν-KCCA algorithm is used in conjunction with the sparse centering procedure of Section 6.1.2 for the sequence features. Only the linear kernel is applied on the sequence features, since preliminary tests showed that polynomial and RBF kernels produced consistently worse retrieval rates when compared to the equivalent linear results.

	KCCA		ν-KCCA			ν-PDCCA		
Features	Accuracy	k	Accuracy	k	ν	Accuracy	k	ν
S3	.053 (.046)	416.7	.075 (.042)	50	0.6	**.105** (.011)	66.7	0.5
S4	.075 (.049)	366.7	.049 (.039)	83.3	0.6	**.105** (.011)	100	0.5
S4-1G1	.055 (.052)	366.7	**.105** (.011)	50	0.8	**.105** (.011)	83.3	0.5
S4-1G2	.051 (.030)	350	.087 (.040)	50	0.7	**.105** (.011)	83.3	0.5
S4-2G1	.079 (.054)	333.3	.075 (.042)	50	0.6	**.105** (.011)	100	0.5
S5	.058 (.051)	383.3	.048 (.041)	66.7	0.7	**.105** (.011)	66.7	0.5
S5-1G1	.080 (.053)	366.7	.051 (.045)	66.7	0.7	**.105** (.011)	66.7	0.7
S6	.030 (.011)	416.7	.035 (.004)	50	0.7	**.104** (.007)	50	0.6
GTG	.071 (.039)	250	**.105** (.011)	50	0.8	.066 (.041)	183.3	0.8
S3	.072 (.037)	283.3	**.115** (.006)	50	1.0	.112 (.009)	83.3	1.0
S4	.053 (.033)	66.7	.108 (.021)	116.7	0.8	**.111** (.008)	83.3	0.6
S4-1G1	**.112** (.011)	183.3	.111 (.016)	116.7	0.8	.111 (.008)	100	0.6
S4-1G2	.107 (.012)	350	.080 (.057)	116.7	1.0	**.112** (.009)	66.7	1.0
S4-2G1	.106 (.018)	200	**.118** (.011)	50	0.9	.114 (.007)	116.7	0.8
S5	.085 (.045)	200	**.107** (.020)	116.7	0.8	.103 (.014)	83.3	0.7
S5-1G1	.095 (.046)	66.7	**.104** (.015)	183.3	0.8	.081 (.044)	50	0.7
S6	.095 (.046)	66.7	**.109** (.011)	116.7	0.6	.079 (.055)	150	0.5
GTG	.089 (.024)	233.3	**.111** (.020)	116.7	0.8	.066 (.049)	183.3	0.7

TABLE 6.11: Average mate retrieval accuracies on the enzyme data. Top results use κ_r, and bottom ones use κ_s for the reaction kernel evaluations.

Table 6.11 shows that the mate retrieval accuracies are low in general, with a best accuracy of 0.118 with ν-KCCA using the S4-2G1 features and the reaction kernel generated using κ_s. If one predicts all zeros for these reaction features then an accuracy of at least $221/2000 = 0.111$ can be achieved. If the κ_r function is used for reaction kernel evaluations, the base accuracy is $209/2000 = 0.105$. This implies that the CCA variants do not find a significant common semantics for the enzyme features and reactions kernels in this case.

A possible explanation for the low mate retrieval rates is that the reaction kernels are not useful for this scenario. Furthermore, the number of unique kernel matrix columns (shown in Table 6.10) computed using κ_r is 603, which implies that the same number of unique reactions are measured by the corresponding kernel function. In the same way, the kernel matrix computed using κ_s has 593 unique reactions. Notice that there are a large number of reactions which have a corresponding zero kernel matrix column. It may be preferable to consider molecules of size greater than 10 for example, as well as alternative reaction representations which better model their similarities.

6.4.4 Learning the Semantics of Enzymes and their EC Numbers

An experiment which uses the EC numbers of the enzymes as opposed to the reaction kernels is now conducted. The EC numbers are slightly more descriptive than the reaction kernels, having 700 unique EC classifications for the 2000 example training sample. Each EC number is represented using an indicator vector of length 1633 containing four non-zero entries matching the respective EC nodes. Since the lower nodes are more fine-grained descriptions of reactions than the nodes above, each level is weighted as 100^{w-1} where $w = 1$ for the top level and $w = 4$ for the bottom level. The experimental setup is identical to that of the previous experiment.

	KCCA		ν-KCCA			ν-PDCCA		
Features	Accuracy	k	Accuracy	k	ν	Accuracy	k	ν
S3	**.267** (.006)	50	.160 (.008)	250	0.8	.217 (.007)	316.7	0.8
S4	.172 (.019)	66.7	.129 (.010)	200	0.8	**.222** (.077)	333.3	1
S4-1G1	**.212** (.008)	50	.159 (.007)	50	0.8	.121 (.014)	333.3	0.8
S4-1G2	**.206** (.011)	50	.157 (.008)	100	0.8	.130 (.013)	333.3	0.8
S4-2G1	**.214** (.013)	50	.160 (.009)	50	0.8	.150 (.055)	283.3	0.9
S5	.097 (.007)	383.3	.104 (.008)	183.3	0.6	**.261** (.020)	316.7	0.8
S5-1G1	.103 (.008)	433.3	**.110** (.003)	166.7	0.8	.063 (.012)	316.7	0.9
S6	.075 (.006)	283.3	.088 (.001)	183.3	0.5	**.205** (.016)	283.3	0.8
GTG	**.593** (.008)	266.7	.575 (.015)	233.3	0.9	.456 (.024)	216.7	0.8

TABLE 6.12: Average EC classification accuracies with different sequence features.

The results of Table 6.12 show that the most effective features are the GTG ones. The mate retrieval rate for KCCA with the GTG features is 0.593, which is a significant improvement over all of the substring features. The corresponding retrieval rate with ν-KCCA is only slightly worse at 0.575. An examination of the average correlations in these cases reveals that KCCA has a correlation of 0.675 and ν-KCCA has a correlation of 0.732, hence a higher value does not necessarily imply improved retrieval rates. Since the retrieval rate is concerned with the rank of the examples in the projected spaces, there is no reason to expect it to be proportional to the angle between corresponding features. However, perfect correlation does imply a mate retrieval rate of 1. Further evidence of these observations can be found with the GTG result for ν-PDCCA, which is significantly worse than that of both KCCA and ν-KCCA, however, the average correlation of the resulting features is 0.656. It is worth noting that the retrieval rate measure is sensitive to noisy features, which may account for cases in which a high correlation does not result in a high retrieval rate.

Of the substring features it appears that substrings of length 3, and 4 with mismatches result in the highest retrieval rates with KCCA. In these cases, the number of output features is generally small. As a base case, the most frequent EC number occurred 93 times, hence one could easily obtain a retrieval rate of $93/2000 = 0.0465$. Interestingly, the results with ν-PDCCA point to the plain substring features as being the most predictive, and in most of these cases the retrieval rates are higher than the corresponding KCCA ones. The intuition behind performing feature selection for the sequence features is to capture relevant amino acid sequences, as it is likely that this gives ν-PDCCA its advantage in these cases.

Overall, the GTG features are the most effective in predicting the EC numbers, although the substring features also have some merit. Further testing which combined the GTG and substring features by concatenating feature vectors resulted in worse retrieval rates, but improved correlations. For example, with ν-KCCA a combination of the S3 and GTG features results in a retrieval rate of 0.501 and a correlation of 0.801.

6.4.4.1 All Examples

The case study concludes by repeating part of the previous experiment with the complete dataset consisting of 6618 examples. Since the GTG and plain substring features are the most effective on a subsample of this data, only these features are used.

Table 6.13 shows the retrieval rates for both EC numbers and enzymes. Both ν-KCCA and ν-PDCCA are more competitive with KCCA in terms of their retrieval rates when compared to the respective results using 2000 examples. The retrieval of EC numbers in conjunction with the GTG features improves to 0.682 with KCCA, and the corresponding ν-KCCA result is only slightly worse at 0.678. Although ν-PDCCA performs worse than these two algorithms, with an accuracy of 0.587, it uses only 3530 out of the 590,411 original features. Furthermore, the retrieval rate for the enzymes is 0.659, indicating the success of feature extraction for finding common semantics in the paired examples. Of the substring features, it appears that shorter substrings are more effective for predicting EC numbers, with ν-KCCA producing a retrieval rate of 0.468 with the S3 features and only 50 output features. The equivalent ν-PDCCA result is 0.421, which also only uses

Features	Enzyme → EC	EC → Enzyme	k	ν
	KCCA			
S3	.374 (.021)	.273 (.023)	50	-
S4	.319 (.003)	.247 (.022)	50	-
S5	.220 (.010)	**.693** (.016)	483.3	-
GTG	**.682** (.013)	.624 (.002)	300	-
	ν-KCCA			
S3	**.468** (.005)	**.511** (.032)	50	0.8
S4	.305 (.016)	**.526** (.169)	200	0.7
S5	.233 (.008)	.471 (.127)	316.7	0.7
GTG	.678 (.016)	.626 (.028)	233.3	0.8
	ν-PDCCA			
S3	.421 (.001)	.150 (.053)	266.7	0.6
S4	**.392** (.028)	.211 (.020)	400	1
S5	**.390** (.035)	.550 (.046)	483.3	0.8
GTG	.587 (.006)	**.659** (.007)	316.7	0.8

Table 6.13: Average EC and enzyme mate retrieval rates using all 6618 examples.

2647 of the 8338 original features. We stated earlier that typically 3 or 4 amino acids act on substrate molecules, and these results back up this claim. As a whole, the results suggest that a suitable combination of S3, S4, S5 and GTG features has the potential to result in high retrieval rates for both EC numbers and enzymes.

6.5 Summary

Solving KCCA using Incomplete Cholesky decompositions of the kernel matrices ignores the fact that the resulting approximations are used to find correlations. This motivated the use of a KCCA-based optimisation where the dual directions have their cardinality constrained to have at most p elements. The non-zero entries for the dual directions are approximated by maximising correlation and then deflating, giving rise to a method we refer to as p-KCCA. A variation of p-KCCA chooses sparse primal directions in one view, and hence performs feature selection in a CCA-based framework.

An empirical evaluation of the CCA methods on artificial datasets highlights cases in which p-KCCA and p-PDCCA improve over KCCA in terms of the cumulative correlation of the features. Further testing on real-world datasets demonstrates a comparable performance and improved sparsity over KCCA when used for mate retrieval. An enzyme function prediction case study using the CCA methods was also conducted, in which the reactions of enzymes are predicted from their sequences. With the GTG enzyme representation and 6618 examples we were able to retrieve EC classifications with an accuracy of 0.682 using KCCA, and 0.678 with ν-KCCA using just 50 output features. Novel graph kernel reaction representations did not yield high retrieval rates, which ties in with the observation made in Borgwardt and Kriegel (2005) that current kernel methods are more appropriate for function class prediction than for specific function determination.

Chapter 7

Conclusions

Feature extraction reduces the dimensionality of a set of examples, and in doing so highlights important characteristics of the data, reduces computational and memory requirements and often improves prediction accuracy. Traditional approaches to feature extraction have often relied on solving eigenvalue problems, which are effective for relatively small datasets. New applications such as bioinformatics, image and text classification produce large amounts of data and have encouraged researchers to focus on more scalable approaches. Many of these algorithms, however, rely on complex optimisation procedures and are difficult to implement and analyse. The focus of this thesis was on scalable feature extraction, using a high degree of sparsity and with simple implementations.

A key element of the novel algorithms developed this thesis is the PLS deflation, which was used in a number of instances. One such case was the formulation of a general feature extraction framework, introduced in Chapter 3. It is a generalisation of PLS which allows for projection directions to be generated according to a user defined criterion. The primary advantages of the general framework are that one can target features towards any particular application domain and the framework maintains several useful properties of PLS. Furthermore, using the framework, close connections between PCA, PLS, BLF and their kernel variants were demonstrated. A limitation of the framework however is that the resulting features require all of the examples, regardless of how projection directions are chosen, and hence, the generated subspace may be statistically unstable in certain circumstances. Furthermore, numerical stability issues may arise through successive deflation, however, we observed that centering the data reduces the effect of this problem.

Another important theme in this thesis is the use of projection directions with a high degree of sparsity, enforced by choosing a single example per direction. The advantage of sparsity is that the resulting formulations can often be easily interpreted and efficiently solved, however, one must sacrifice some of the quality of the resulting directions compared to non-sparse ones. In our feature extraction approaches, sparsity resulted in algorithms with simple implementations, and that scaled linearly in the number of examples in both computational and memory requirements. Given that the size of the kernel matrix is ℓ^2, where ℓ is the number of examples, this implies that not all of the kernel matrix elements are used. Randomisation was applied to limit the number of kernel matrix columns used at each iteration. This was effective in many cases, although it can degrade performance when the distribution of qualities of the columns is long tailed.

The first feature extraction scenario considered was matrix approximation, in which PCA is a popular approach. The observation that PCA can be understood as a method in which one chooses directions and then deflates led to an important question: Given a deflation scheme, which projection direction maximises the Frobenius norm of the difference between successively deflated matrices? An answer is provided to this question for a number of different deflations in Chapter 4. The results show that the PCA directions are optimal for the PCA and PLS deflations, and the KPCA directions are optimal for the KPCA and left-sided KPLS deflations. A move to sparse directions, using a single example per direction, led to an alternative derivation of KFA. A similar derivation results in the novel GSD-KPLS and GDD-KPLS methods, based on the left and double-sided KPLS deflations. Our empirical comparison of the methods showed that sparsity did not significantly influence the quality of the resulting approximations and with GSD-KPLS and GDD-KPLS the test kernel matrix approximations often improve over that of KPCA.

A common use of feature extraction is as a step before classification or regression, hence Chapter 5 examined supervised methods. Instead of focusing feature extraction towards a particular classifier or regressor, two general approaches are formulated by maximising covariance and alignment (called SMC and SMA respectively). Both of these methods were derived using the general feature extraction method, and possess all of the useful properties it provides. As part of the analysis of their properties, a bound on the expectation of the covariance of the features produced by these algorithms is presented using Rademacher theory, which is general enough to be applied to PLS. The bound gives insight into the situations in which the features generated on a training set are likely to have a cumulative squared covariance close to that of a test set. We concluded the study of the new algorithms with an empirical comparison to several other popular feature extraction methods. Our new methods were shown to be competitive, when followed by classification or regression, to the other approaches on a selection of small and large real-world datasets. In particular, using 20,000 examples from the Reuters Corpus Volume 1 dataset, SMA was shown to match the performance of all 136,469 original features in conjunction with an SVM with just 366 output features.

The final feature extraction scenario we explored was learning using paired examples, which is useful for finding common semantics in a set of English documents and their corresponding Japanese translations for example. KCCA is commonly used for this kind of task, and to reduce computational requirements one often approximates each kernel matrix using an Incomplete Cholesky decomposition. There are two disadvantages with this approach: the Incomplete Cholesky decomposition is not the most effective means for approximating a kernel matrix at linear cost in the number of examples, and one might be able to improve upon matrix approximation by targeting sparsity towards maximising correlation. The latter reason motivates p-KCCA, which uses a subset of the columns of the kernel matrices in order to solve an optimisation similar to the KCCA one. A variation of p-KCCA, called p-PDCCA, operates in the primal space for one view and in a dual space for the other. The advantage of p-PDCCA is that one can perform feature selection in the primal view, although the number of features is limited to the rank of the data matrix. With these approaches, the enzyme function prediction problem is studied using GTG and substring representations of enzymes. The first task was to predict the reactions catalysed by the enzymes by modelling reactions using graph kernels and then using the CCA variants in a mate retrieval manner. Unfortunately, the reaction representations were not informative enough to provide high retrieval rates. Improved results were obtained when trying to predict the EC classifications of the enzymes. We obtained EC number retrieval rates of 0.682 and 0.678 with KCCA and p-KCCA respectively, using the GTG features.

7.1 Future Work

The work presented in this thesis has opened up several avenues for further research. Some minor extensions include the use of the general framework in conjunction with different projection directions (e.g. one based on LDA), and the extension of SMA and SMC to cater for vectorial labels. Another interesting area to investigate is finding an efficient method for choosing the value of c used for the selection of dual projection directions. Additionally, one could study effective caching strategies for reducing the need to recompute kernel matrix columns for the selection of dual directions. Further to these, we outline several more substantial extensions to our work.

The first extension considers the use of sparse stability bounds on the extracted features. Stability bounds have already been seen for SMA, SMC and KCCA using the Rademacher approach. However, for sparse methods the size of the function class involved is smaller than the set of linear functions with bounded norm. This suggests a way of tightening the bound of Theorem 5.5, for example. As an alternative to the Rademacher approach, one could formulate a bound using the PAC-Bayesian method (Langford and Shawe-Taylor (2003)). Whilst such bounds are not always close to the empirical expectation, the shape of the bound as the number of dimensions varies may be useful as a stopping criterion.

Another interesting research area is the combination of feature extraction with a prediction algorithm. For example in Zwald et al. (2005) the authors notice that using KPCA as a step before SVM classification can be seen as double regularisation. Hence, they combine KPCA regularisation in an empirical risk minimisation algorithm to form a new classifier called the Kernel Projection Machine (KPM). The deficiencies of the approach is that all of the examples are used for the projections, and the labels are only used to guide dimensionality as opposed to the directions themselves. An improvement over KPM could explore the use of sparse supervised directions (e.g. by using SMA and SMC) to improve efficiency and generalisation.

Enzyme function prediction is a relatively new application area for machine learning, and our work possesses scope for further research. A key area of interest is appropriate feature representations for both enzymes and their reactions. It was observed that the current reaction representations are not informative enough, and it is useful to consider alternative ones e.g. bond strengths and geometric information about the shape of the substrate and product molecules. One can also potentially improve accuracy by modelling enzymes with features indicative of the secondary structure, particularly relating to the active site. If accurate prediction is achieved then an interesting further avenue of research is to find the pre-image (e.g. in Kwok and Tsang (2004)) of the reaction kernels. This would enable the discovery of the original reaction representation corresponding to a novel protein, even if the reaction did not exist in the training set.

As well as formulating new sequence representations, we observed that independently both substring and GTG representations are useful for predicting EC numbers, however a simple combination of the different features proved to be ineffective. This suggests the importance of researching suitable data combination techniques. Related work is presented in Ong et al. (2003, 2005) and Lanckriet et al. (2004a) which learn the kernel matrix for large margin classifiers. The learned representation could instead be targeted towards minimising the average rank of the mate examples. One way of combining features is to use a committee of feature extraction methods, with weights determined by individual performances. A related strategy of pooling selected features from different feature subsets can be used to overcome the limitation that p-PDCCA chooses at most $\min(\text{rank}(\mathbf{X}), \text{rank}(\mathbf{K}^y))$ features.

As a final direction for research, recall that with the enzyme function prediction data, features with high correlation did not necessarily have a high mate retrieval rate. One could improve upon KCCA and related approaches by maximising the clustering of the paired examples as opposed to correlation, which would better ensure mate examples are close together.

Appendix A

Data Preprocessing

A.1 Centering

To center a set of examples ensures that the center of mass of the data, given by

$$\Phi_s = \frac{1}{\ell}\sum_{i=1}^{\ell}\mathbf{x}_i = \frac{1}{\ell}\mathbf{X}'\mathbf{j},$$

is zero. It follows that a data matrix is centered using

$$\begin{aligned}\tilde{\mathbf{X}} &= \mathbf{X} - \frac{1}{\ell}\mathbf{j}\mathbf{j}'\mathbf{X} \\ &= \left(\mathbf{I} - \frac{\mathbf{j}\mathbf{j}'}{\ell}\right)\mathbf{X},\end{aligned}$$

and the kernel matrix is centered by

$$\tilde{\mathbf{K}} = \left(\mathbf{I} - \frac{\mathbf{j}\mathbf{j}'}{\ell}\right)\mathbf{K}\left(\mathbf{I} - \frac{\mathbf{j}\mathbf{j}'}{\ell}\right).$$

A.2 Normalisation

If the features of a dataset are scaled differently, it becomes difficult to compare them since large variations in the scalings can over-emphasise certain features. One solution is to standardise the data by rescaling the features to have unit norm,

$$\tilde{\mathbf{X}} = \mathbf{X} \cdot \mathrm{diag}(\mathbf{X}'\mathbf{X})^{-1/2},$$

where diag(·) is a diagonal matrix composed of the diagonal entries of its input.

Unfortunately if one only has access to the kernel matrix, normalisation is not possible in the same way. An alternative is to normalise the examples to have unit norm, i.e.

$$\tilde{\mathbf{X}} = \mathrm{diag}(\mathbf{X}\mathbf{X}')^{-1/2} \cdot \mathbf{X},$$

so that

$$\tilde{\mathbf{K}} = \mathrm{diag}(\mathbf{K})^{-1/2} \cdot \mathbf{K} \cdot \mathrm{diag}(\mathbf{K})^{-1/2}.$$

Bibliography

MIT CBCL face database 1, 1996. Center for Biological and Computational Learning, MIT.

Jerónimo Arenas-García, Kaare Brandt Petersen, and Lars Kai Hansen. Sparse kernel orthonormalized PLS for feature extraction in large data sets. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 33–40, Cambridge, MA, 2006. MIT Press.

Nachman Aronszajn. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

Katja Astikainen, Juho Rousu, Liisa Holm, Esa Pitkanen, and Sandor Szedmak. Towards structured prediction of enzyme function. *Machine Learning in Systems Biology*, 2007.

Francis R. Bach and Michael I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3(1):1–48, 2003.

Francis R. Bach and Michael I. Jordan. Predictive low-rank decomposition for kernel methods. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning*, pages 33–40. ACM Press New York, NY, USA, 2005.

Matthew Barker and William Rayens. Partial least squares for discrimination. *Journal of Chemometrics*, 17(3):166–173, 2003.

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2003. ISSN 1533-7928.

Asa Ben-Hur and Douglas Brutlag. Protein sequence motifs: Highly predictive features of protein function. In Isabelle M. Guyon, Steve R. Gunn, Masoud Nikravesh, and Lofti Zadeh, editors, *Feature Extraction, Foundations and Applications*, pages 625–645. Springer, 2005.

Kristin P. Bennett and Mark J. Embrechts. An optimization perspective on kernel partial least squares. In *Advances in Learning Theory: Methods, Models and Applications.*

NATO Science Series III: Computer & Systems Science, volume 190, pages 227–250, 2003.

Jinbo Bi, Kristin P. Bennett, Mark J. Embrechts, Curt M. Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006. ISBN 0387310738.

Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

Karsten M. Borgwardt and Hans-Peter Kriegel. Protein Function Prediction via Graph Kernel. In *Intelligent Systems in Molecular Biology*, pages 47–56, 2005.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, 2005. ISSN 1367-4803.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Conference on Computational Learning Theory*, pages 144–152, New York, NY, USA, 1992. ACM Press. ISBN 0-89791-497-X.

C. Z. Cai, L. Y. Han, Z. L. Ji, and Y. Z. Chen. Enzyme family classification by support vector machines. *Proteins Structure Function and Bioinformatics*, 55(1):66–76, 2004.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Koby Crammer, Joseph Keshet, and Yoram Singer. Kernel design using boosting. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 537–544, Cambridge, MA, 2002. MIT Press.

Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz S. Kandola. On kernel-target alignment. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 367–373, Cambridge, MA, 2001. MIT Press.

Nello Cristianini, John Shawe-Taylor, and Huma Lodhi. Latent Semantic Kernels. *Journal of Intelligent Information Systems*, 18(2):127–152, 2002.

Alexandre d'Aspremont, Laurent El Ghaoui, Michael I. Jordan, and Gert R.G. Lanckriet. A Direct Formulation for Sparse PCA Using Semidefinite Programming. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 41–48, Cambridge, MA, 2005. MIT Press.

Tijl De Bie, Nello Cristianini, and Roman Rosipal. Eigenproblems in pattern recognition. In E. Bayro-Corrochano, editor, *Handbook of Computational Geometry for Pattern Recognition, Computer Vision, Neurocomputing and Robotics*, pages 129–167. Springer-Verlag, Heidelberg, April 2004.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, November 2000. ISBN 0471056693.

Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

Vojtvech Franc and Václav Hlavác. Greedy kernel principal component analysis. In Henrik I. Christensen and Hans-Hellmut Nagel, editors, *Cognitive Vision Systems*, pages 87–106, Heidelberg, Germany, February 2006. Springer-Verlag. ISBN 3-540-33971-X.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the 2nd European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2.

Jerome H. Friedman and John W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9):881–889, 1974.

Halil Altay Guvenir and Ilhan Uysal. Bilkent university function approximation repository, 2000.

Isabelle Guyon. Practical feature selection: from correlation to causality. In *Mining Massive Data Sets for Security*. IOS Press, 2008.

Isabelle M. Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Isabelle M. Guyon, Steve R. Gunn, Masoud Nikravesh, and Lofti Zadeh, editors. *Feature Extraction, Foundations and Applications*. Springer, 2006. ISBN 3540354875.

David R. Hardoon and John Shawe-Taylor. Sparse Canonical Correlation Analysis. Technical report, University College London, UK, 2007a.

David R. Hardoon and John Shawe-Taylor. Stability Analysis of Kernel Canonical Correlation Analysis: Theory and Practice. Technical report, University College London, UK, 2007b.

David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: an overview with application to learning methods. *Neural Computation*, 16: 2639–2664, 2004.

Andreas Heger, Michael Lappe, and Liisa Holm. Accurate detection of very sparse sequence motifs. In *Proceedings of the 7th annual international conference on Research in Computational Molecular Biology*, pages 139–147, New York, NY, USA, 2003. ACM. ISBN 1-58113-635-8.

Bernd Heisele, Tomaso Poggio, and Massimiliano Pontil. Face detection in still gray images. A.I. memo 1687, Center for Biological and Computational Learning, MIT, USA, Cambridge, MA, 2000.

Luc Hoegaerts, Johan A. K. Suykens, Joos Vandewalle, and Bart De Moor. Primal space sparse kernel partial least squares regression for large scale problems. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 561–566, 2004.

Agnar Hoskuldsson. PLS regression methods. *Journal of Chemometrics*, 2(3):211–228, 1988.

David W. Hosmer and Stanley Lemeshow. *Applied Logistic Regression*. Wiley-Interscience, 2nd edition, 2000. ISBN 0-471-35632-8.

Harold Hotelling. Analysis of a complex of statistical variables into principle components. *Journal of Educational Psychology*, 24:417–441 and 498–520, 1933.

Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28:312–377, 1936.

Peter J. Huber. Projection Pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

Anil K. Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-336165-9.

Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142. Springer Verlag, Heidelberg, DE, 1998.

Thorsten Joachims. Training linear SVMs in linear time. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining*, pages 217–226, New York, NY, USA, 2006. ACM Press.

Ian T. Jolliffe, Nickolay T. Trendafilov, and Mudassir Uddin. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics*, 12(3):531–547, 2003.

Erwin Kreyszig. *Introductory functional analysis with applications.* Wiley, New York, NY, USA, 1978. ISBN 0-471-03729-X.

Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semi-supervised graph clustering: a kernel approach. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine learning*, pages 457–464. ACM Press New York, NY, USA, 2005.

Malte Kuss and Thore Graepel. The Geometry of Kernel Canonical Correlation Analysis. Technical report, Max Plank Institute for Biological Cybernetics, Germany, 2003.

James T. Kwok and Ivor W. Tsang. The pre-image problem in kernel methods. *IEEE Transactions on Neural Networks*, 15(6):1517–1525, 2004.

Gert R.G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research*, 5:27–72, 2004a.

Gert R.G. Lanckriet, Minghua Deng, Nello Cristianini, Michael I. Jordan, and William S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 300–311, 2004b.

John Langford and John Shawe-Taylor. PAC-Bayes and Margins. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 423–430, Cambridge, MA, 2003. MIT Press.

Michel Ledoux and Michel Talagrand. *Probability in Banach Spaces: isoperimetry and processes.* Springer, May 1991. ISBN 0387520139.

Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 566–575, 2002.

Christina S. Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004. ISSN 1367-4803.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

Rolf Manne. Analysis of two partial-least-squares algorithms for multivariate calibration. *Chemometrics and Intelligent Laboratory Systems*, 2(1):187–197, 1987.

William F. Massy. Principal Components Regression in Exploratory Statistical Research. *Journal of the American Statistical Association*, 60(309):234–256, 1965.

Colin McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics 1989*, pages 148–188. Cambridge University Press, Cambridge, 1989.

Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. Fisher discriminant analysis with kernels. In *Proceedings of the IEEE Signal Processing Society Workshop*, pages 41–48, 1999.

Tom Mitchell. *Machine learning.* McGraw Hill, New York, NY, USA, 2nd edition, 1997. ISBN 978-0070428072.

Baback Moghaddam, Yair Weiss, and Shai Avidan. Generalized spectral bounds for sparse LDA. In William W. Cohen and Andrew Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning*, pages 641–648, New York, NY, USA, 2006a. ACM. ISBN 1-59593-383-2.

Baback Moghaddam, Yair Weiss, and Shai Avidan. Spectral Bounds for Sparse PCA: Exact and Greedy Algorithms. In Yair Weiss, Bernhard Schölkopf, and John Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 915–922, Cambridge, MA, 2006b. MIT Press.

Michinari Momma. Efficient computations via scalable sparse kernel partial least squares and boosted latent features. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery in Data mining*, pages 654–659, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-135-X.

Michinari Momma and Kristin P. Bennett. Sparse kernel partial least squares regression. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory*, pages 216–230, 2003.

Michinari Momma and Kristin P. Bennett. Constructing orthogonal latent features for arbitrary loss. In Isabelle M. Guyon, Steve R. Gunn, Masoud Nikravesh, and Lofti Zadeh, editors, *Feature Extraction, Foundations and Applications*, pages 551–583. Springer, 2005.

D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.

Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In Bernhard Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 903–910, 2001.

Cheng Soon Ong, Alex J. Smola, and Robert C. Williamson. Hyperkernels. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, volume 15, pages 495–502, Cambridge, MA, 2003. MIT Press.

Cheng Soon Ong, Alex J. Smola, and Robert C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005. ISSN 1533-7928.

Aloke Phatak and Frank de Hoog. Exploiting the connection between PLS, Lanczos methods and conjugate gradients: alternative proofs of some properties of PLS. *Journal of Chemometrics*, 16(7):361–367, 2002.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Chris Burges, and Alex Smola, editors, *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999. ISBN 0262194163.

David Poole. *Linear Algebra: A Modern Introduction.* Thomson Brooks/Cole, Pacific Grove, CA, USA, 2nd edition, 2003. ISBN 978-0534341749.

Tony Rose, Mark Stevenson, and Miles Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, pages 827–832, 2002.

Roman Rosipal and Nicole Kramer. Overview and Recent Advances in Partial Least Squares. *Subspace, latent structure and feature selection techniques, Lecture Notes in Computer Science*, pages 34–51, 2006.

Roman Rosipal and Leonard J. Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. *Journal of Machine Learning Research*, 2:97–123, 2001.

Roman Rosipal, Leonard J. Trejo, and Bryan Matthews. Kernel PLS-SVC for linear and nonlinear classification. In Armand Prieditis and Stuart J. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 640–647, 2003.

Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1): 23–38, 1998.

Paul D. Sampson, Ann P. Streissguth, Helen M. Barr, and Fred L. Bookstein. Neurobehavioral effects of prenatal alcohol: Part II. Partial Least Squares analysis. *Neurotoxicology and Teratology*, 11(5):477–491, 1989.

Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. ISSN 0899-7667.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.

John Shawe-Taylor, Christopher K. I. Williams, Nello Cristianini, and Jaz S. Kandola. On the eigenspectrum of the gram matrix and the generalization error of kernel PCA. *IEEE Transactions on Information Theory*, 51(7):2510–2522, 2005.

Alex J. Smola, Olvi L. Mangasarian, and Bernhard Scholkopf. Sparse kernel feature analysis. Technical report, Data Mining Institute, University of Wisconsin, Madison, USA, 1999.

Alex J. Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 911–918, 2000. ISBN 1-55860-707-2.

Bharath K. Sriperumbudur, David A. Torres, and Gert R. G. Lanckriet. Sparse eigen methods by d.c. programming. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning*, pages 831–838, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

Mervyn Stone and Rodney J. Brooks. Continuum Regression: Cross-Validated Sequentially Constructed Prediction Embracing Ordinary Least Squares, Partial Least Squares and Principal Components Regression. *Journal of the Royal Statistical Society. Series B (Methodological)*, 52(2):237–269, 1990.

Gilbert Strang. *Introduction to Linear Algebra*. Wellesley Cambridge Press, Wellesley, MA, USA, 3rd edition, 2003. ISBN 0961408898.

Sandor Szedmak, Tijl De Bie, and David R. Hardoon. A metamorphosis of canonical correlation analysis into multivariate maximum margin learning. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, Bruges, April 2007.

Sandor Szedmak, John Shawe-Taylor, and Emilio Parado-Hernandez. Learning via linear operators: Maximum margin regression. Technical report, University of Southampton, UK, 2005.

Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. In Jyrki Kivinen and Robert H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory*, pages 74–89, London, UK, 2002. Springer-Verlag. ISBN 3-540-43836-X.

Michael E. Tipping. Sparse kernel principal component analysis. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 633–639, Cambridge, MA, 2001. MIT Press.

David Torres, Douglas Turnbull, Luke Barrington, and Gert R. Lanckriet. Identifying words that are musically meaningful. In *Proceedings of the 8th International Conference on Music Information Retrieval*, pages 405–410, 2007.

Matthew A. Turk and Alex P. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991a.

Matthew A. Turk and Alex P. Pentland. Face recognition using eigenfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991b.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998. ISBN 0471030031.

Alexei Vinokourov, John Shawe-Taylor, and Nello Cristianini. Inferring a semantic representation of text via cross-language correlation analysis. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1473–1480, Cambridge, MA, 2003. MIT Press.

Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004. ISSN 0920-5691.

Jacob A. Wegelin. A survey of partial least squares (PLS) methods, with emphasis on the two-block case. Technical report, University of Washington, USA, Washington, D.C., 2000.

Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461, 2003. ISSN 1533-7928.

Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. Feature selection for SVMs. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 668–674, Cambridge, MA, 2000. MIT Press.

Wikipedia. EC Number — Wikipedia, the free encyclopedia, 2007.

Christopher K. I. Williams and Matthias Seeger. The effect of the input density distribution on kernel-based classifiers. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 1159–1166, 2000a. ISBN 1-55860-707-2.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688, Cambridge, MA, 2000b. MIT Press.

Herman Wold. Estimation of principal components and related models by iterative least squares. *Multivariate analysis*, pages 391–420, 1966.

Herman Wold. Path models with latent variables: The NIPALS approach. *Quantitative Sociology: International perspectives on mathematical and statistical model building*, pages 307–357, 1975.

K. J. Worsley, J. B. Poline, K. J. Friston, and A. C. Evans. Characterizing the Response of PET and fMRI Data Using Multivariate Linear Models. *Neuroimage*, 6(4):305–319, 1997.

Wen-Yi Zhao, Rama Chellappa, P. Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458, 2003. ISSN 0360-0300.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.

Laurent Zwald, Regis Vert, Gilles Blanchard, and Pascal Massart. Kernel projection machine: a new tool for pattern recognition. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1649–1656, Cambridge, MA, 2005. MIT Press.